

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Industrial communication networks – Fieldbus specifications –
Part 6-27: Application layer protocol specification – Type 27 elements**

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-27: Spécification du protocole de la couche application – Éléments de
type 27**

IECNORM.COM : Click to view the full PDF of IEC 61158-6-27:2023



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2023 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC de votre pays de résidence.

IEC Secretariat
3, rue de Varembe
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
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 corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform

The advanced search enables 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 It also gives information on projects, replaced and withdrawn publications.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 online and once a month by email.

IEC 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: sales@iec.ch.

IEC Products & Services Portal - products.iec.ch

Discover our powerful search engine and read freely all the publications previews. With a subscription you will always have access to up to date content tailored to your needs.

Electropedia - www.electropedia.org

The world's leading online dictionary on electrotechnology, containing more than 22 300 terminological entries in English and French, with equivalent terms in 19 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

A propos de l'IEC

La Commission Electrotechnique Internationale (IEC) 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 IEC

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

Recherche de publications IEC -

webstore.iec.ch/advsearchform

La recherche avancée permet de trouver des publications IEC 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.

IEC Just Published - webstore.iec.ch/justpublished

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

Service Clients - webstore.iec.ch/csc

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

IEC Products & Services Portal - products.iec.ch

Découvrez notre puissant moteur de recherche et consultez gratuitement tous les aperçus des publications. Avec un abonnement, vous aurez toujours accès à un contenu à jour adapté à vos besoins.

Electropedia - www.electropedia.org

Le premier dictionnaire d'électrotechnologie en ligne au monde, avec plus de 22 300 articles terminologiques en anglais et en français, ainsi que les termes équivalents dans 19 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Industrial communication networks – Fieldbus specifications –
Part 6-27: Application layer protocol specification – Type 27 elements**

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-27: Spécification du protocole de la couche application – Éléments de
type 27**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 25.040

ISBN 978-2-8322-6620-5

Warning! Make sure that you obtained this publication from an authorized distributor. Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.

CONTENTS

FOREWORD.....	7
INTRODUCTION.....	9
1 Scope.....	10
1.1 General.....	10
1.2 Specifications	10
1.3 Conformance	11
2 Normative references	11
3 Terms, definitions, abbreviated terms, symbols and conventions	12
3.1 Referenced terms and definitions.....	12
3.1.1 Terms and definitions from ISO/IEC 7498-1	12
3.1.2 Terms and definitions from ISO/IEC 9545	12
3.1.3 Terms and definitions from ISO/IEC 8824-1	13
3.1.4 Terms and definitions from ISO/IEC 10731	13
3.1.5 Terms and definitions from ISO/IEC 19501	13
3.2 Additional terms and definitions	13
3.3 Abbreviations and symbols	22
3.4 Conventions.....	24
3.4.1 General conventions.....	24
3.4.2 PDU data type conventions.....	24
3.4.3 State machine conventions	24
4 Abstract syntax.....	26
4.1 General.....	26
4.2 Basic Data types.....	27
4.3 FAL PDU types	28
4.3.1 General	28
4.3.2 Top of APDU types: _APDU.....	31
4.3.3 PDUs for field system management service	31
4.3.4 Detailed definitions of _FID-PDUs	34
4.3.5 PDUs for field device control service	48
4.3.6 PDUs for message service.....	57
4.4 Detailed definitions of _FDCService-PDUs.....	58
4.4.1 Enhanced PDU type	58
4.5 Device profile.....	72
5 Transfer syntax.....	72
5.1 Concepts	72
5.2 Encode rules.....	73
5.2.1 INTEGER and its subtypes	73
5.2.2 REAL type and its subtypes	74
5.2.3 BIT STRING type.....	76
5.2.4 OCTET STRING type and IA5String type	77
5.2.5 NULL type	77
5.2.6 Structure type and Array type	77
6 Structure of FAL protocol state machine	78
7 AP-context state machine (APC SM)	80
7.1 Overview.....	80
7.2 State descriptions	81

7.3	Triggering events	81
7.4	Action descriptions at state transitions	82
8	FAL service protocol machines (FSPM)	84
8.1	Overview	84
8.2	RT Protocol Machine (RT PM)	84
8.2.1	Link layer discovery	84
8.2.2	MAC bridges	84
8.2.3	Virtual bridges	105
8.2.4	IP suite	105
8.2.5	DLL mapping protocol machine (DMPM)	105
8.3	Field System Management Protocol Machine (FSM PM)	110
8.3.1	Overview	110
8.3.2	Discovery and basic configuration	112
8.3.3	Starting up of system	115
8.3.4	Sync methods	144
8.3.5	Plug-and-play entry	153
8.4	Field Device Control Protocol Machine (FDC PM)	154
8.4.1	Protocol overview	154
8.4.2	Cyclic communication mode	156
8.4.3	Event driven communication mode	160
8.4.4	Master Protocol Machine (FDCPM-M)	161
8.4.5	Slave Protocol Machine (FDCPM-S)	169
8.4.6	Error procedure summary	179
8.5	Message Protocol Machine (MSG PM)	181
8.5.1	Protocol overview	181
8.5.2	Requester Protocol Machine (MSGPM-RQ)	183
8.5.3	Responder Protocol Machine (MSGPM-RS)	186
9	Application relationship protocol machine (ARPM)	188
9.1	General	188
9.2	ARPM for FDC ASE	188
9.2.1	Overview	188
9.2.2	ARPM for FDC Master (ARPM-FDCM)	189
9.2.3	ARPM for FDC Slave (ARPM-FDCS)	196
9.3	ARPM for MSG ASE (ARPM-MSG)	203
9.3.1	State descriptions	203
9.3.2	Triggering events	203
9.3.3	Action descriptions at state transitions	204
10	DLL mapping protocol machines (DMPMs)	204
	Annex A (informative) Device profile and FDC command sets	205
	Annex B (normative) Virtual memory space and Device Information	207
	B.1 Overview	207
	B.2 Communication Data Object	207
	B.3 Device Information	208
	B.3.1 Device identifier area structure	208
	B.3.2 Detail specifications of device IDs	209
	Annex C (informative) Basic message function	217
	Bibliography	218

Figure 1 – Encode of Integer subtypes	73
Figure 2 – Example of transfer of INTEGER value	74
Figure 3 – Encode of Unsigned subtypes	74
Figure 4 – Float32 type encode.....	75
Figure 5 – Float64 type encode.....	75
Figure 6 – Bit field definition example with named bits	76
Figure 7 – Bit field definition example with field size	77
Figure 8 – SEQUENCE type encode	78
Figure 9 – Structure of FAL protocol state machines	80
Figure 10 – State chart diagram of APC SM.....	81
Figure 11 – Master CTC state transition chart.....	86
Figure 12 – Slave CTC state transition chart.....	96
Figure 13 – Structuring of the protocol machines within the DMPM (bridge).....	106
Figure 14 – SRC state transition chart	106
Figure 15 – Example of network configuration.....	111
Figure 16 – Network initialization procedures.....	112
Figure 17 – Neighboring node notification sequence	113
Figure 18 – Connected node detection sequence.....	114
Figure 19 – Example of neighboring node information.....	115
Figure 20 – FSMUL state transition chart for the master with SM function.....	116
Figure 21 – FSMUL state transition chart for the master without SM function	129
Figure 22 – FSMUL state transition chart for a slave.....	138
Figure 23 – Sequence of SM delay time notification by CDO writing.....	145
Figure 24 – Delay measurement sequence (SM: BM).....	147
Figure 25 – Master delay measurement sequence for the BM (SM: master with band master function (BM))	148
Figure 26 – Delay measurement sequence for the master other than the BM (SM: BM).....	149
Figure 27 – Delay measurement sequence (SM: S#1).....	151
Figure 28 – Master delay measurement sequence for the BM (SM: S#1).....	152
Figure 29 – Plug-and-play entry sequence	154
Figure 30 – Example communication cycle of FDC master AP.....	156
Figure 31 – Example communication cycle of FDC slave AP	157
Figure 32 – Synchronous command communication in sync state	158
Figure 33 – Asynchronous command communication in sync state.....	159
Figure 34 – Asynchronous command communication in async state.....	160
Figure 35 – Event-driven communication	161
Figure 36 – State chart diagram of FDCPM-M.....	162
Figure 37 – State chart diagram of FDCPM-S	170
Figure 38 – PDU transmission flow for user message	182
Figure 39 – PDU transmission flow for one-way message	183
Figure 40 – State chart diagram of MSGPM-RQ.....	184
Figure 41 – State chart diagram of MSGPM-RS	186
Figure 42 – Example of single transfer process.....	188

Figure 43 – Example of dual transfer process	189
Figure 44 – State chart diagram of ARPM-FDCM	190
Figure 45 – Statechart diagram of ARPM-FDCS	196
Figure 46 – Statechart diagram of ARPM-MSG	203
Figure B.1 – Memory map of virtual memory space	207
Figure B.2 – Memory map of device ID area	208
Table 1 – State transition descriptions	25
Table 2 – Description of state machine elements	25
Table 3 – Conventions used in state machines	26
Table 4 – Multicast address	29
Table 5 – Length_or_type	29
Table 6 – IPv4 Header	30
Table 7 – IPv6 Header	30
Table 8 – UDP Header	30
Table 9 – Node address	33
Table 10 – Mapping for Protocol State Machines	79
Table 11 – State descriptions of APC SM	81
Table 12 – Trigger event descriptions of APC SM	82
Table 13 – Transitions of APC SM	83
Table 14 – Master CTC state transition table	87
Table 15 – Slave CTC state transition table	96
Table 16 – List of CTC macros	101
Table 17 – The primitives and parameters for FDC interface issued by FDC	103
Table 18 – The primitives and parameters for FSM interface issued by CTC	104
Table 19 – The list of primitives and parameters (FSM source)	104
Table 20 – The list of primitives and parameters (FSMUL source)	105
Table 21 – SRC state transition table	107
Table 22 – List of SRC macros	108
Table 23 – List of SRC functions	109
Table 24 – Primitives and parameters for SRC-CTC interface	110
Table 25 – Send frame primitive and parameters	110
Table 26 – Receive frame primitives and parameters	110
Table 27 – Primitives and parameters of repeat select service	110
Table 28 – FSMUL state transition table for the master with SM function	117
Table 29 – FSMUL state transition table for the master without SM function	129
Table 30 – FSMUL state transition table for a slave	138
Table 31 – List of FSMUL macros	142
Table 32 – FDC protocol mode	155
Table 33 – State descriptions of FDCPM-M	162
Table 34 – Trigger event descriptions of FDCPM-M	163
Table 35 – Transitions of main SM of FDCPM-M	164
Table 36 – Transitions of submachine of FDCPM-M	166

Table 37 – State descriptions of FDCPM-S	170
Table 38 – Trigger event descriptions of FDCPM-S	171
Table 39 – Transitions of main SM of FDCPM-S	172
Table 40 – Transitions of submachine of FDCPM-S	174
Table 41 – State descriptions of MSGPM-RQ	184
Table 42 – Trigger event descriptions of MSGPM-RQ	185
Table 43 – Transitions of MSGPM-RQ	185
Table 44 – State descriptions of MSGPM-RS	186
Table 45 – Trigger event descriptions of MSGPM-RS	187
Table 46 – Transitions of MSGPM-RS	187
Table 47 – State descriptions of ARPM-FDCM	191
Table 48 – Trigger event descriptions of ARPM-FDCM	193
Table 49 – Transitions of main SM of ARPM-FDCM	193
Table 50 – Transitions of submachine of ARPM-FDCM	195
Table 51 – State descriptions of ARPM-FDCS	197
Table 52 – Trigger event descriptions of ARPM-FDCS	199
Table 53 – Transitions of main SM of ARPM-FDCS	200
Table 54 – Transitions of submachine of ARPM-FDCS	201
Table 55 – State descriptions of ARPM-MSG	203
Table 56 – Trigger event descriptions of ARPM-MSG	203
Table 57 – Transitions of ARPM-MSG	204
Table A.1 – Example of registered device profiles	205
Table A.2 – Example command list of the profile '00'H	206
Table B.1 – Memory map of CDO area	207
Table B.2 – Specifications of device IDs	209
Table C.1 – Example of message command set	217

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**INDUSTRIAL COMMUNICATION NETWORKS –
FIELDBUS SPECIFICATIONS –****Part 6-27: Application layer protocol specification –
Type 27 elements****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.
- 9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

Attention is drawn to the fact that the use of the associated protocol type is restricted by its intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a layer protocol type to be used with other layer protocols of the same type, or in other type combinations explicitly authorized by its intellectual-property-right holders.

NOTE Combinations of protocol types are specified in the IEC 61784 1 series and the IEC 61784 2 series.

IEC 61158-6-27 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation. It is an International Standard.

The text of this International Standard is based on the following documents:

Draft	Report on voting
65C/1205/FDIS	65C/1234/RVD

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

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

A list of all parts of the IEC 61158 series, under the general title *Industrial communication networks – Fieldbus specifications*, can be found on the IEC web site.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

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

IECNORM.COM : Click to view the full PDF of IEC 61158-6-27:2023

INTRODUCTION

This document is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the "three-layer" fieldbus reference model described in IEC 61158-1.

The application protocol provides the application service by making use of the services available from the data-link or other immediately lower layer. The primary aim of this document is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer application entities (AEs) at the time of communication. These rules for communication are intended to provide a sound basis for development in order to serve a variety of purposes:

- as a guide for implementers and designers;
- for use in the testing and procurement of equipment;
- as part of an agreement for the admittance of systems into the open systems environment;
- as a refinement to the understanding of time-critical communications within OSI.

This document is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this document together with other documents positioned within the OSI or fieldbus reference models, otherwise incompatible systems can work together in any combination.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent. IEC takes no position concerning the evidence, validity, and scope of this patent right.

The holder of this patent right has assured IEC that s/he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with IEC. Information may be obtained from the patent database available at patents.iec.ch.

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-27: Application layer protocol specification – Type 27 elements

1 Scope

1.1 General

The Fieldbus Application Layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a "window between corresponding application programs."

This part of IEC 61158 provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 27 fieldbus. The term "time-critical" is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This document defines in an abstract way the externally visible behavior provided by the Type 27 fieldbus application layer in terms of

- the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,
- the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,
- the application context state machines defining the application service behavior visibly between communicating application entities, and
- the application relationship state machines defining the communication behavior visibly between communicating application entities.

The purpose of this document is to define the protocol provided to

- define the representation-on-wire of the service primitives defined in IEC 61158-5-27, and
- define the externally visible behavior associated with their transfer.

This document specifies the protocol of the Type 27 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498-1) and the OSI Application Layer Structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this document is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-27.

A secondary objective is to provide migration paths from previously existing industrial communications protocols. It is this latter objective which gives rise to the diversity of protocols standardized in IEC 61158-6 series.

1.3 Conformance

This document does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

Conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE All parts of the IEC 61158 series, as well as the IEC 61784-1 series and the IEC 61784-2 series are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

IEC 61158-5-27:2023, *Industrial communication networks – Fieldbus specifications – Part 5-27: Application layer service definition – Type 27 elements*

ISO/IEC 646, *Information technology – ISO 7-bit coded character set for information interchange*

ISO/IEC 7498-1, *Information technology – Open Systems Interconnection – Basic Reference Model – Part 1: The Basic Model*

ISO/IEC 9545, *Information technology – Open Systems Interconnection – Application Layer structure*

ISO/IEC 9899, *Information technology – Programming languages – C*

ISO/IEC 10731, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

ISO/IEC 19501:2005, *Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*

ISO/IEC/IEEE 60559:2020, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

ISO/IEC/IEEE 8802-3, *Telecommunications and exchange between information technology systems – Requirements for local and metropolitan area networks – Part 3: Standard for Ethernet*

IEEE Std 802.1D, *IEEE Standards for local and metropolitan area networks – Media Access Control (MAC) Bridges*

IEEE Std 802.1Q, *IEEE Standards for local and metropolitan area networks – Bridges and Bridged Networks*

IEEE Std 802.1AB, *IEEE Standards for Local and Metropolitan Area Networks: Station and Media Access Control Connectivity Discovery*

IEEE Std 802.1AS, *IEEE Standards for Local and Metropolitan Area Networks: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*

IETF RFC 768, J. Postel, "*User Datagram Protocol*", August 1980, available at <<https://www.rfc-editor.org/info/rfc768>> [viewed 2022-09-16]

IETF RFC 791, J. Postel, "*Internet Protocol*", September 1981, available at <<https://www.rfc-editor.org/info/rfc791>> [viewed 2022-09-16]

IETF RFC 826, D. Plummer, "*An Ethernet Address Resolution Protocol: or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*", November 1982, available at <<https://www.rfc-editor.org/info/rfc826>> [viewed 2022-09-16]

IETF RFC 1112, S.E. Deering, "*Host Extensions for IP Multicasting*", August 1989, available at <<https://www.rfc-editor.org/info/rfc1112>> [viewed 2022-09-16]

IETF RFC 2460, S. Deering and R. Hinden, "*Internet Protocol, Version 6 (IPv6) Specification*", December 1998, available at <<https://www.rfc-editor.org/info/rfc2460>> [viewed 2022-09-16]

3 Terms, definitions, abbreviated terms, symbols and conventions

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

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- IEC Electropedia: available at <https://www.electropedia.org/>
- ISO Online browsing platform: available at <https://www.iso.org/obp>

3.1 Referenced terms and definitions

3.1.1 Terms and definitions from ISO/IEC 7498-1

For the purposes of this document, the following terms as defined in ISO/IEC 7498-1 apply:

- abstract syntax;
- application-entity;
- application process;
- application protocol data unit;
- application-process-invocation;
- (N)-facility;
- (N)-function;
- peer-(N)-entities;
- presentation context;
- real system;
- transfer syntax.

3.1.2 Terms and definitions from ISO/IEC 9545

For the purposes of this document, the following terms as defined in ISO/IEC 9545 apply:

- application-association;
- application-context;
- application-entity-invocation;
- application-entity-type;

- application-service-element.

3.1.3 Terms and definitions from ISO/IEC 8824-1

For the purposes of this document, the following terms as defined in ISO/IEC 8824-1 apply:

- simple type;
- component;
- component type;
- integer type;
- bitstring type;
- octetstring type;
- null type;
- sequence type;
- sequence of type;
- choice type;
- IA5String type;
- encoding.

3.1.4 Terms and definitions from ISO/IEC 10731

For the purposes of this document, the following terms as defined in ISO/IEC 10731 apply:

- OSI-service-primitive; primitive;
- OSI-service-provider; provider;
- OSI-service-user; user.

3.1.5 Terms and definitions from ISO/IEC 19501

For the purposes of this document, the following terms as defined in ISO/IEC 19501 apply:

- event;
- state;
- state machine;
- substate;
- submachine;
- transition.

3.2 Additional terms and definitions

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

3.2.1

alarm

field device status to tell that the device has detected a fatal problem to be solved and cannot continue normal working, through the field device control (FDC) service of the Type 27 fieldbus

Note 1 to entry: Any alarm statuses are latched and need some operations to be cleared.

Note 2 to entry: Alarms are classified into three groups; communication alarms, illegal-command-related ones, and application specific ones. But concrete definitions are dependent on implementation of each field devices.

3.2.2

application process object

network representation of a specific aspect of an application process (AP), which is modelled as a network accessible object contained within an AP or within another APO

Note 1 to entry: Refer IEC 61158-1, 9.3.4.

3.2.3

application process context

AP context

shared knowledge or a common set of rules, governing communication of FAL application entities (AEs) and describing the permissible collective communications behavior between the AEs that are party to a specific set of application relationships (ARs)

Note 1 to entry: Data within AP context can be specified by the user in advance, by the option selected while the user uses a field bus management (FSM) service to read out the facility of peer AP, by the automatic negotiation function that the FSM system handles, and so on. The method that is to be adopted depends on the specification of each implementation.

3.2.4

application process type

AP type

description of a classification of application processes (APs) in terms of a set of capabilities for FAL of the Type 27 fieldbus

Note 1 to entry: AP types are classified into two, C1 master AP and slave AP, by their application roles in the fieldbus network.

3.2.5

async command

type of a command application protocol data unit (APDU) of the FDC service of the Type 27 FAL, which can be issued any time after the previous transaction without consideration of synchronization with the communication cycle

Note 1 to entry: Definitions, which command should be async one or not, are dependent on an application. They can be provided as a registered set of commands and responses or device profiles, see Annex A.

3.2.6

asynchronous communication

state or a way of communication for the FDC service of the Type 27 FAL, in which a command can be issued any time after the previous transaction without consideration of synchronization with the communication cycle

Note 1 to entry: In this state, sync commands cannot be issued, but async commands can.

3.2.7

attribute

information or parameter contained in variable portions of an object

Note 1 to entry: Typically, they provide status information or govern the operation of an object. Attributes also affect the behavior of an object.

3.2.8

C1 master

AP type that has master facilities for the FDC service of the Type 27 FAL, or the device implementing that AP type

Note 1 to entry: Only one C1 master exists in a network of the Type 27 fieldbus.

3.2.9

command

PDU issued by a requester or a master to make a responder or a slave execute some functions

**3.2.10
communication**

process to exchange information in a formal manner between two or more devices, users, APs or entities

**3.2.11
transfer**

process to convey a PDU from a sender to a receiver

**3.2.12
transmission**

process to send out and propagate electrical signals or encoded data

**3.2.13
communication cycle**

period of repetitive activities synchronized with the transmission cycle while the connection establishing for the FDC protocol of the Type 27 FAL

Note 1 to entry: Communication cycle can synchronize with the transmission cycle multiplied by a specified scaling factor.

**3.2.14
connection**

context or logical binding under specific conditions for the FDC protocol between a master object and a slave object for the Type 27 FAL

**3.2.15
cyclic**

repetitive in a regular manner

**3.2.16
cyclic communication**

transmission mode in which request PDUs and response PDUs are exchanged repetitively in the scheduled time slots synchronized with a transmission cycle of the Type 27

Note 1 to entry: In the AL, the communication cycle arises from the transmission cycle in this mode.

**3.2.17
cycle scale counter**

counter to generate a communication cycle by means of scaling a primary cycle or a transmission cycle

**3.2.18
device ID**

part of "Device Information" to identify the device for a specific product type or model of the Type 27 fieldbus

**3.2.19
device information**

formatted and device-embedded information to characterize a device, which mainly consists of data for device model identification and device-profile specific parameters for the Type 27 fieldbus

**3.2.20
device profile**

collection of device-model-common information and functionality providing consistency between different device models among the same kind of device

3.2.21

dual transfer

transfer mode for the FDC protocol of the Type 27 FAL, in which a sender sends a same PDU twice a transaction and a receiver uses them to detect and recover a communication error such as data-corruption or data-loss in cyclic communication mode

3.2.22

event driven communication

transmission mode for the lower layer protocol of the Type 27 fieldbus in which a transaction of command-response-exchanging arises as user's demands

Note 1 to entry: Both the transmission cycle and the communication cycle don't arise in this mode.

3.2.23

error

abnormal condition or malfunction for communication or any other activities

3.2.24

field device control

FDC service

time-critical communication service that handles a fixed length command data to control a field device and the corresponding feedback response data in a severe restriction on delay or jitter for the communication timing for the Type 27 FAL

3.2.25

field device protocol

FDC protocol

time-critical communication protocol that handles a fixed length command data to control a field device and the corresponding feedback response data in a severe restriction on delay or jitter for the communication timing for the Type 27 FAL

3.2.26

master

class or its instance object of FDC application service element (ASE) that plays a role of a command requester for the Type 27 FAL

Note 1 to entry: One or more units exist within the network as the node to send output data (command data) to slaves and receive input data (response data) from slaves in the data link layer. Also, it executes message data sending to/receiving from slaves. It can execute both or either one of input/output data sending/receiving and message data sending/receiving.

Note 2 to entry: Input/output data sending/receiving and message data sending/receiving shall be executed at a constant cycle and in a constant band.

Note 3 to entry: Or, it is an application type that executes field device control in the application layer.

3.2.27

message service

MSG service

communication service that handles the variable length data and does not require a severe restriction on response time

3.2.28

network clock

synchronized and periodically running counter that each node in a same network has, which becomes an oscillation source of the transmission cycle

3.2.29

primary cycle

period of repetitive activities synchronized with the transmission cycle before the connection establishing for the FDC protocol in Type 27 FAL

3.2.30**protocol machine**

state machine that realizes a protocol as the main function of the entity in each layer

3.2.31**requester**

class or its instance object of MSG ASE that plays a role of a command requester or sender for the Type 27 FAL

3.2.32**responder**

class or its instance object of MSG ASE that plays a role of a command responder or receiver for the Type 27 FAL

3.2.33**response**

PDU issued by a responder or a slave to inform a result or some status for the received command to a requester or a master

3.2.34**service**

operation or process that an object performs upon request from another object

3.2.35**single transfer**

normal transfer mode for the FDC protocol of the Type 27 FAL in which a sender sends a PDU once a transaction

3.2.36**slave AP**

AP type that has slave facilities for the FDC service of the Type 27 FAL, or the device implementing that AP type

3.2.37**slave**

class or its instance object of FDC ASE that plays a role of a responder for the Type 27 FAL

Note 1 to entry: A node to receive output data (command data) from a master and send input data (response data) to the master that sent the output data in the data link layer. Also, it executes message data sending to/receiving from the master. It can execute input/output data and message data sending to/receiving from multiple masters. Also, it can receive input data of other slaves.

3.2.38**sync command**

type of command APDU of the FDC service of the Type 27 FAL, which is issued at the synchronized timing with every communication cycle

Note 1 to entry: The definitions, which command is sync one or not, are dependent on an application. They can be provided as a registered set of commands and responses or device profiles, see Annex A.

3.2.39**synchronous communication**

state or a way of communication for the FDC service of the Type 27 FAL, in which a command is issued at the synchronized timing with every communication cycle

Note 1 to entry: In this state, both sync commands and async ones can be issued.

Note 2 to entry: In this state, an out-of-synchronization error of APs shall be detected by measures of the watchdog counter.

3.2.40**transmission cycle**

period of repetitive activities for the lower layers of the Type 27 fieldbus, which of all the slave devices are synchronized with that of a C1 master device by the lower layer protocol

Note 1 to entry: Cycle in which the sync master sends a sync packet. The master having the band master function informs each node in the network. The sync band, the control domain transmission band, the IP transmission band and the guard band need to be within the transmission cycle. The transmission cycle starting point is managed by the sync master and is informed by the sync packet to all nodes. Between the master and slave within the control domain, or in the data link layer between the master having the band master function and the master without it, cyclic transmissions can be repeated at a cycle of n -times (n is an integer equal to or greater than zero) of the transmission cycle and it can vary from slave to slave.

3.2.41**transmission mode**

state or a way of transmission for the lower layer protocol of the Type 27 fieldbus; cyclic mode, event driven mode

3.2.42**virtual memory space**

large data block of APOs for the Type 27 FAL that can be read and written with pseudo-memory-addresses to provide consistency between different device models

Note 1 to entry: The virtual memory space includes the device information and other vender specific area, see Annex B.

3.2.43**warning**

field device status to tell that the device has detected a slight or passing problem but still working normally through the field device control (FDC) service of the Type 27 fieldbus

Note 1 to entry: Any warning statuses are latched and need to be operated to clear them.

Note 2 to entry: Warnings are classified into three groups, communication warnings, illegal-command-related ones, and application specific ones. But concrete definitions are dependent on an implementation of each field devices.

3.2.44**node**

device that comply with the protocols described in this specification

3.2.45**node type**

master or slave

3.2.46**multi control domain system**

configuration where multiple control domains exist in the network

3.2.47**connected node**

collective designation of master and slaves that are connected to the same network

3.2.48**primary node****initiator**

node to start cyclic transmissions and message transmissions

Note 1 to entry: Only the master serves as the primary node.

3.2.49
secondary node
responder

node to execute cyclic transmissions and message transmissions according to the command from the primary node

3.2.50
output data
command data

data to be sent from a master to a slave

3.2.51
input data
response data

data to be sent from a slave to a master

3.2.52
input/output data

input data and output data

3.2.53
control data

collective designation of output data and input data

3.2.54
message data

data transmitted separately from the control data, which are exchanged between a master and a slave and sent when requested by the application

3.2.55
cyclic transmission

cyclic exchanging of control data between a master and a slave

3.2.56
message transmission

exchanging of message data to transmit information separately from the control data, which is, unlike cyclic transmission, conducted only when required by the application

3.2.57
IP transmission

communication using IP packets

3.2.58
control domain

logical group which consists of a master and slaves to exchange control data and/or message data

Note 1 to entry: Multiple control domains can exist in the network, and in this case, they are indicated as control domain #*n* (*n* is an integer).

Note 2 to entry: A slave shall belong to one or more control domains.

3.2.59
sync band

band to transmit sync packet

3.2.60
cyclic transmission band

band to execute cyclic transmissions

3.2.61

output data transmission band

band to transmit output data in cyclic transmissions

3.2.62

input data transmission band

band to transmit input data in cyclic transmissions

3.2.63

message transmission band

band to execute message transmissions

3.2.64

control domain transmission band

total band of the cyclic transmission band and the message transmission band within a control domain

3.2.65

IP transmission band

band to execute IP packet transmissions

3.2.66

guard band

band to prohibit sending/receiving by the devices in the network

3.2.67

destination address

address to specify the control data destination, having 16-bit information

3.2.68

source address

address to specify the control data source, having 16-bit information

3.2.69

node address

address to be specified as destination address or source address

3.2.70

logical node number

number from 1 to 127 to be used as serial number in ascending order, which enables the master to specify the slave, because there are 127 connectable slaves to the node address having 16-bit space in the data link layer

3.2.71

transmission delay time

time required for the frame sent by the sync master or the master in the data link layer to reach each node

Note 1 to entry: A total time of the signal propagation time in the transmission cables and the repeat time at the relay nodes.

3.2.72

response monitoring time

time to monitor the reception of responses

Note 1 to entry: This is measured from the starting of sending of data sent from the primary node (including the transmission time of the data sent from the primary node).

3.2.73**consecutive transmissions**

continuous transmissions of output data

3.2.74**retransmission request**

executing of input data retransmission request

3.2.75**event driven communication**

communication mode where the master and the slave exchange data at an arbitrary timing, without executing clock synchronization in the devices by the sync packet sent from the sync master

3.2.76**transmission delay measurement**

individual measurements by the sync master of transmission delay time for the nodes other than its own node

3.2.77**upstream**

direction of data forwarding from the primary node to the secondary node

3.2.78**downstream**

direction of data forwarding from the secondary node to the primary node

3.2.79**upstream port**

port that becomes upstream among the communication ports to be implemented

Note 1 to entry: While multiple control domains exist in the network (two or more masters exist), in cases where the upstream port differs from control domain to control domain, it is referred to as the control domain # n (n is an integer) upstream port.

3.2.80**downstream port**

port that becomes downstream among the communication ports to be implemented

Note 1 to entry: While multiple control domains exist in the network (two or more masters exist), in cases where the downstream port differs from control domain to control domain, it is referred to as the control domain # n (n is an integer) downstream port.

3.2.81**transfer method to upstream**

upstream transfer method among the transfer methods in the node where two or more communication ports are implemented

3.2.82**transfer method to downstream**

downstream transfer method among the transfer methods in the node where two or more communication ports are implemented

3.2.83**store and forward**

transfer method to store the received packets in the node first and forward them to the ports other than the receiving ports according to the result of stored data analysis and the band conditions upon receiving in the band

3.2.84

cut-through

transfer method to forward the received packets immediately to the ports other than the receiving port

3.2.85

repeat

transfer method to transfer the received packets in the same way as cut-through in cases where sending is not in progress in its own node, or to retain the received packets during sending from its own node and forward to the ports other than the receiving port after finishing sending from its own node

3.2.86

IP packet

all packets other than those with the MAC frame length/type field of RT-PDU ('F68E'H) in this document

3.2.87

sync packet

packet to synchronize transmission cycles of all the devices in the network, which is sent from the sync master

3.3 Abbreviations and symbols

For the purposes of this document, the following abbreviations and symbols apply.

AE	Application Entity
AL	Application Layer
A-, AL-	Application Layer (as a prefix)
ALME	Application Layer Management Entity
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Invocation
APO	Application Process Object
APC	Application Process Context (as prefix of a protocol for Type 27 fieldbus)
APC SM	Application Process Context State Machine (for Type 27 fieldbus)
AR	Application Relationship
AR ASE	Application Relationship Application Service Element
AREP	Application Relationship End Point
ARPM	Application Relationship Protocol Machine (for Type 27 fieldbus)
ARPM-FDCM	ARPM for Field Device Control service Master (for Type 27 fieldbus)
ARPM-FDCS	ARPM for Field Device Control service Slave (for Type 27 fieldbus)
ARPM-MSG	ARPM for Message service (for Type 27 fieldbus)
ASCII	American Standard Code for Information Interchange
ASDU	Application Service Data Unit
ASE	Application Service Element
ASN.1	Abstract Syntax Notation One
BM	Band Master
CDO	Communication Data Object
CMD	Command PDU for FDC service (for Type 27 fieldbus)

Cnf	confirm primitive
CTC	Communication Transfer Control Underlying Layer
DA	Destination Node Address
DLPDU	Data Link-Protocol Data Unit
DLSDU	Data Link Service Data Unit
DMPM	Data Link Mapping Protocol Machine
DMA	Destination MAC Address
E2PROM	Electrically erasable programmable read only memory
FAL	Fieldbus Application Layer
FCS	Frame check sequence
FDC-	Field Device Control (as prefix of a service or a protocol for Type 27 fieldbus)
FDC ASE	Field Device Control Application Service Element (for Type 27 fieldbus)
FDCPM	Field Device Control Protocol Machine (for Type 27 fieldbus)
FDCPM-M	Field Device Control Protocol Machine for Master (for Type 27 fieldbus)
FDCPM-S	Field Device Control Protocol Machine for Slave (for Type 27 fieldbus)
FIFO	First In First Out
FSPM	FAL service protocol machine
FSM-	Fieldbus System Management (as prefix of a service for Type 27 fieldbus)
FSM ASE	Fieldbus System Management Application Service Element for Type 27 fieldbus
FSMUL	Field System Management Underlying Layer
HMI	Human-machine Interface
HS	Hand Shake
I/O	Input/output
ID	Identifier
Ind	indication primitive
LME	Layer Management Entity
Lsb	least significant bit
LT	Length or Type
MAC	Media Access Control
Msb	most significant bit
MSG	Message (as prefix of a service or a protocol for Type 27 fieldbus)
MSG ASE	Message Application Service Element for Type 27 fieldbus
MSGPM	Message Protocol Machine for Type 27 fieldbus
MSGPM-RQ	MSGPM for Requester for Type 27 fieldbus
MSGPM-RS	MSGPM for Responder for Type 27 fieldbus
OSI	Open Systems Interconnection
PM	Protocol machine
PDU	Protocol Data Unit
PhL	Ph-layer
QoS	Quality of Service
RAM	Random access memory
Req	request primitive

Rsp	response primitive
RSP	Response PDU for FDC service (for Type 27 fieldbus)
SA	Source Node Address
SAP	Service Access Point
SDN	Send Data with no Acknowledge
SDU	Service Data Unit
SFD	Start Frame Delimiter
SM	Sync Master
SMA	Source MAC Address
SMIB	System Management Information Base
SRC	Send Receive Control Underlying Layer
UML	Unified Modelling Language

3.4 Conventions

3.4.1 General conventions

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate subclause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two are contained in IEC 61158-5-27. The protocol specification for each of the ASEs is defined in this document.

The class definitions define the attributes of the classes supported by each ASE. In this document, it is assumed that every attribute are accessible only from the owner's instance object itself directly or through services of the class.

This document uses the descriptive conventions given in ISO/IEC 10731.

3.4.2 PDU data type conventions

The data types of FAL PDUs are defined with the notations of ASN.1.

Character strings that begin with "_" (low line) are used as data type symbols of FAL PDUs or their fields. A same string as a PDU type without "_" means a PDU instance of the corresponding data type.

Example CMD-PDU means an instance of _CMD-PDU and a following statement is omitted.

```
CMD-PDU _CMD-PDU ::= { value }
```

3.4.3 State machine conventions

The protocol sequences are described by means of State Machines.

In state chart diagrams, states are represented as boxes, and state transitions are shown as arrows. Their conventions are given in ISO/IEC 19501 as Universal Modeling Language (UML).

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 Table 1.

The first row contains the name of the transition by an index number.

The second row defines the source state or the current state before the transition.

The third row contains an event and actions. The event is followed by optional arguments in parentheses, "(" and ")", and guard conditions in brackets, "[" and "]", as the first line. And the actions with starting character "/" follow the event line.

The last row contains the target state or the next state after the transition.

If the event occurs and the conditions are fulfilled the transition fires, i.e. the actions are executed and the next state is entered.

Table 1 – State transition descriptions

T#	Source State	Event (arguments) [conditions] / Action	Target state

A meaning of each element in Table 1 is shown in Table 2, based on UML "transition" definition (ISO/IEC 19501).

Table 2 – Description of state machine elements

Description element	Meaning
Source state Target state	Names of the originating state and the target state of transition.
T#	Name or number of the transition.
Event	Name or description of the trigger event that fire the transition.
(arguments)	A parameter value, an expression, or a sequence of those divided by "," for the event. The preceding "(" and the succeeding ")" are not part of the argument list.
[conditions]	Boolean expression, which is true for the transition to be fired. The preceding "[" and the succeeding "]" are not part of the condition.
/ Action	List of assignments and service or function invocations. The action should be atomic. The preceding "/" is not part of the action.
NOTE "(arguments)" and "[condition]" can be omitted if not necessary.	

The conventions used in the state machines are shown in Table 3.

Table 3 – Conventions used in state machines

Convention	Meaning
<code>/*-- description --*/</code>	Describes and explains conditions and/or procedure by using normal sentence between <code>/*--</code> and <code>--*/</code> instead of using the pseudo code notation. Example: <code>/*-- The PM examines an occurrence of any alarms for the corresponding remote device.</code> <code>-- If an alarm is detected, the PM notifies it to the FSM ASE.</code> <code>-- If not, the PM transfers the MSGService-PDU to the DLL.</code> <code>--*/</code>
<code>=</code>	Value of an item on the left is replaced by value of an item on the right. If an item on the right is a parameter, it comes from the primitive shown as an input event
<code>Axx</code>	Parameter name if 'a' is a letter Example: <code>Identifier = reason;</code> means value of a 'reason' parameter is assigned to a parameter called 'Identifier'
<code>"xxx"</code>	Fixed visible string Example: <code>Identifier = "abc";</code> means value "abc" is assigned to a parameter named 'Identifier'
<code>Nnn</code>	If all elements are digits, the item represents a numerical constant shown in decimal representation.
<code>0xnn</code>	If all elements nn are digits, the item represents a numerical constant shown in hexadecimal representation.
<code>==</code>	Logical condition to indicate an item on the left is equal to an item on the right
<code><</code>	Logical condition to indicate an item on the left is less than the item on the right
<code>></code>	Logical condition to indicate an item on the left is greater than the item on the right
<code>!=</code>	Logical condition to indicate an item on the left is not equal to an item on the right
<code>&&</code>	Logical "AND"
<code> </code>	Logical "OR"
<code>!</code>	Logical "NOT"
<code>+</code> <code>-</code> <code>*</code> <code>/</code>	Arithmetic operators
<code>;</code>	Separator of expressions

Further notations as defined in C language (ISO/IEC 9899) can be used to describe conditions and actions.

4 Abstract syntax

4.1 General

Packet formats and frame formats comply with the ISO/IEC/IEEE 8802-3.

4.2 Basic Data types

The following data types can be used in the Type 27 FAL.

- Basic types as simple types of ASN.1:
 - INTEGER;
 - REAL;
 - BIT STRING;
 - OCTET STRING;
 - NULL.
- Specific basic types as subtypes from ASN.1:
 - Integer8 ::= INTEGER (–128..127);
 - Integer16 ::= INTEGER (–32 768 .. 32 767);
 - Integer32 ::= INTEGER (–2 147 483 648 .. 2 147 483 647);
 - Integer64 ::= INTEGER ((–1)×'8000 0000 0000 0000'H..'7FFF FFFF FFFF FFFF'H);
 - Unsigned8 ::= INTEGER (0..'FF'H);
 - Unsigned16 ::= INTEGER (0..'FFFF'H);
 - Unsigned32 ::= INTEGER (0..'FFFF FFFF'H);
 - Unsigned64 ::= INTEGER (0..'FFFF FFFF FFFF FFFF'H);
 - Float32 ::= 0 | negative infinity | positive infinity
 | REAL (WITH COMPONENTS(mantissa ((–1)× c_{32}), base (2), exponent (q_{32})))
 | REAL (WITH COMPONENTS(mantissa (c_{32}), base (2), exponent (q_{32})))

where,

$$c_{32} ::= \text{REAL} (1.. \left(\sum_{i=0}^{23} \frac{1}{2^i} \right)) = \text{REAL} (1..(2-2^{-23})), \text{ and}$$

$$q_{32} ::= \text{INTEGER} (-127..127).$$

NOTE 1 This type has a length of four octets. The format for float32 is that defined by ISO/IEC/IEEE 60559 as single precision, see 5.2.2.

- Float64 ::= 0 | negative infinity | positive infinity
 | REAL (WITH COMPONENTS(mantissa ((–1)× c_{64}), base (2), exponent (q_{64})))
 | REAL (WITH COMPONENTS(mantissa (c_{64}), base (2), exponent (q_{64})))

where,

$$c_{64} ::= \text{REAL} (1.. \left(\sum_{i=0}^{52} \frac{1}{2^i} \right)) = \text{REAL} (1.. (2-2^{-52})), \text{ and}$$

$$q_{64} ::= \text{INTEGER} (-1\ 023..1\ 023).$$

NOTE 2 This type has a length of eight octets. The format for float64 is that defined by ISO/IEC/IEEE 60559 as double precision. See 5.2.2.

- IA5String
- Structure types with component types of ASN.1:
 Structure type is defined by a combination of SEQUENCE type, CHOICE type of ASN.1, basic types and subtype elements defined in ASN.1. The FAL PDU described in 4.3 and 4.4 that follows this subclause correspond this Structure type.
 - SEQUENCE
 - CHOICE

- Array types with a component type of ASN.1:

Array types are a list of any data type defined with SEQUENCE OF type of ASN.1. Typical array types are shown with SEQUENCE OF type, below:

- SEQUENCE OF;
- BitArray ::= SEQUENCE OF (BIT STRING SIZE (8));
- Array8 ::= SEQUENCE OF Integer8;
- Array16 ::= SEQUENCE OF Integer16;
- Array32 ::= SEQUENCE OF Integer32;
- Array64 ::= SEQUENCE OF Integer64.

4.3 FAL PDU types

4.3.1 General

4.3.1.1 Packet format

The overview of the MAC frame used for this protocol is as described below. VLAN tags specified by IEEE Std 802.1Q can be used as an option to designate the priority level of the packets (see 4.3.1.2.4, 8.2). Normally, a format without IP headers is used (see 4.3.1.2.5). In cases where connected devices exchange data with a device connected to an external IP network, the UDP/IP protocol shall be used as option. Note that a part of UDP/IP header fields are not used for this protocol (see 4.3.1.2.60).

```

DLPDU ::= SEQUENCE {
    preamble          OCTET STRING SIZE (7),
    start_frame_delimiter OCTET STRING SIZE (1),
    destination_MAC_address OCTET STRING SIZE (6),
    source_MAC_address  OCTET STRING SIZE (6),
    data               _DLSDU,
    frame_check_sequence OCTET STRING SIZE (4)
}

_DLSDU ::= SEQUENCE {
    ( VLAN_tag          OCTET STRING SIZE (4) ) | Empty,
    length_or_type      OCTET STRING SIZE (2),
    data                _IP4-UDP-PDU | _IP6-UDP-PDU | _RT-PDU
    padding             OCTET STRING SIZE (0..46)
}

_IP4-UDP-PDU ::= SEQUENCE {
    IPv4_header      OCTET STRING SIZE (20),
    UDP_header       OCTET STRING SIZE (8),
    payload           OCTET STRING SIZE (1..1472)
}

_IP6-UDP-PDU ::= SEQUENCE {
    IPv6_header      OCTET STRING SIZE (40),
    UDP_header       OCTET STRING SIZE (8),
    payload           OCTET STRING SIZE (1..1452)
}

_RT-PDU ::= {
    payload           OCTET STRING SIZE (1..1500)
}

```

4.3.1.2 Common component

4.3.1.2.1 Preamble field and start_frame_delimiter field

This field shall comply with ISO/IEC/IEEE 8802-3.

4.3.1.2.2 Destination_MAC_address field

This field shall comply with ISO/IEC/IEEE 8802-3. Note that the usages of multicast address in Table 4 are as specified below.

Table 4 – Multicast address

Multicast address	Name	Usage
01:20:B5:00:04:00	Common multicast address	Simultaneous transmissions in the same network
01:20:B5:00:04:01	Multicast address to neighbouring node	This is used for sending to a node whose MAC address is not known. The received node shall not transfer to other ports.
01:20:B5:00:04:21 to 01:20:B5:00:04:2F	For control domain multicast address	For simultaneous transmissions in a control domain

4.3.1.2.3 Source_MAC_address field

This field shall comply with ISO/IEC/IEEE 8802-3.

4.3.1.2.4 VLAN_tag field

This field shall comply with ISO/IEC/IEEE 8802-3. Implementation is optional.

4.3.1.2.5 Length_or_type field

Without an IP header, this field shall be RT-PDU. With an IP header, this field shall comply with ISO/IEC/IEEE 8802-3. Values mainly used are as shown in Table 5.

Table 5 – Length_or_type

Value	Meaning
'F68E'H	RT-PDU
'0800'H	IP4-UDP-PDU, IP4-UDP-PDU
other	In accordance with ISO/IEC/IEEE 8802-3

4.3.1.2.6 IPv4_header field, IPv6_header field and UDP_header field

This field shall comply with IETF RFC 791, IETF RFC 2460 and IETF RFC 768. Note that a part of fields are not used (see Table 6, Table 7 and Table 8).

Table 6 – IPv4 Header

Alignment [octet]	Descriptions	Remarks
0	Version 4 (IPv4) and header length: 5 quadlets (20 octets)	'45'H, fixed
1	Service type	'00'H, fixed
2 to 3	Total length	
4 to 5	Identifier	Not used
6 to 7	Flag and fragment offset	0, fixed
8	Time To Live (TTL)	'01'H, fixed
9	High-order protocol type	'11'H(UDP), fixed
10 to 11	IP header checksum	Without checking
12 to 15	Source IP address	
16 to 19	Destination IP address	

Table 7 – IPv6 Header

Alignment [octet]	Descriptions	Remarks
0 to 3	Version 6 (IPv6)	6, fixed
	Traffic class	Not used (0, fixed)
	Flow label	Not used (0, fixed)
4 to 5	Payload length	
6	Subsequent header	'11'H(UDP), fixed
7	Maximum hop number	
8 to 23	Source IP address	
24 to 39	Destination IP address	

Table 8 – UDP Header

Alignment [octet]	Descriptions	Remarks
0 to 1	Source UDP port number	Not used
2 to 3	Destination UDP port number	'F68E'H, fixed
4 to 5	UDP data length	Not used
6 to 7	UDP checksum	Not used (0, fixed)

4.3.1.2.7 Payload field

_APDU is stored in this field (see 4.3.2).

4.3.1.2.8 Frame_check_SEQUENCE field

This field shall comply with ISO/IEC/IEEE 8802-3.

4.3.2 Top of APDU types: _APDU

_APDU consists of 14 types of FIDAP-PDU. And _FDCServicePDU shall consist of a pair of PDUs; _CMD-PDU and _RSP-PDU, used for FDC commands and its responses respectively. This is used in _OUTPUT-FIDAP-PDU (See 4.3.4.2), _INPUT-FIDAP-PDU (See 4.3.4.3), and _EVD-FIDAP-PDU (See 4.3.4.11). Furthermore, _MSGServicePDU shall consist of a pair of PDUs; _MSGREQ-PDU and _MSGRSP-PDU, used for MSG requests and its responses respectively. This is used in _MSG-FIDAP-PDU (See 4.3.4.9).

```

_APDU          ::= _FIDAP-PDU                                -- FrameID Application Service PDU
_FIDAP-PDU     := _SYNC-FIDAP-PDU
                | _OUTPUT-FIDAP-PDU
                | _INPUT-FIDAP-PDU
                | _TOKEN-FIDAP-PDU
                | _REQ_SEND-FIDAP-PDU
                | _MEAS_DLY-CMD-FIDAP-PDU | _MEAS_DLY-RSP-FIDAP-PDU
                | _READ_CDO-CMD-FIDAP-PDU | _READ_CDO-RSP-FIDAP-PDU
                | _WRITE_CDO-CMD-FIDAP-PDU | _WRITE_CDO-RSP-FIDAP-PDU
                | _MSG-FIDAP-PDU
                | _ASYNCR_IND-FIDAP-PDU
                | _EVD-FIDAP-PDU
_FDCServicePDU ::= _CMD-PDU                                -- FDC command PDU type
                | _RSP-PDU                                -- FDC response PDU type
_MSGServicePDU ::= _MSGREQ-PDU                             -- MSG request PDU type
                | _MSGRSP-PDU                             -- MSG response PDU type

```

4.3.3 PDUs for field system management service

4.3.3.1 Overview

FID-PDU shall be edited with a SDU in the FSM service primitive and be sent via AR. And a FSM ASE receives the PDU from the peer ASE via AR.

An FSM ASE user distinguishes the types by using the command code field as the key code.

```

_FIDAP-PDU     ::= SEQUENCE {
                                fpduLength  _FPDULength,
                                fcmd        _FCMDCode,
                                fHS,        _FHandShake,
                                fpduBody    OCTET STRING SIZE (1..1496)
                                }

```

- fpduLength field

This field shows the octet number of fpduBody. The Data Type is _FPDULength. It shall have three sub fields, data length, divided packet number and resend flag.

```

_FPDULength ::= SEQUENCE {
    Data length          BIT STRING SIZE (12),
    Divided packet number BIT STRING SIZE (3),
    Resend flag,         BIT STRING SIZE (1)
}

```

Data length

Octet number of MAC data, not including the data length itself (2 octets)

Divided packet number

Store the packet number when packets are divided into multiple packets due to size restriction of the MAC data. (Without dividing: 0)

Resend flag

A flag indicating resending of packets. Set to 1 upon resending.

- fcmd field

A command code field to indicate the code for identifying the contents of fpduBody. This field shall be encoded as _FCMDCode for Unsigned8. Already defined common commands are indicated as _PrimaryFCMD.

```

_FCMDCode      ::= Unsigned8
_PrimaryFCMD   ::= _FCMDCode (
    SYNC ('01'H),      1)
    OUTPUT ('02'H),    1)
    INPUT ('03'H),     1)
    TOKEN ('04'H),     1) 2)
    REQ_SEND ('05'H),  1) 2)
    MEAS_DLY ('06'H),  1)
    READ_CDO ('07'H),
    WRITE_CDO ('08'H),
    MSG ('09'H),
    ASYNC_IND ('0A'H),
    EVD ('0B'H)
)

```

- 1) They cannot be used for _IP4-UDP-PDU and _IP6-UDP-PDU. They cannot be forwarded to an external IP network.
- 2) While the send right moves from the source to the destination for a token, the right to send a sending request remains at the source side and one sending request is made to the destination.

- fHS field

This field shall indicate handshake/notification. It shall have five meaningful sub fields, data length, divided packet number and resend flag.

```

_FHandShake ::= SEQUENCE {
    Request / Response flag      BIT STRING SIZE (1),
    Completed flag               BIT STRING SIZE (1),
    reserve1                     BIT STRING SIZE (1),
    Network reset start flag     BIT STRING SIZE (1),
    Network reset start confirmation flag BIT STRING SIZE (1),
    Network reset start completed flag BIT STRING SIZE (1),
    reserve2                     BIT STRING SIZE (1)
}

```

Request / Response flag

Flag to identify request or response

- 0: Request;
- 1: Response.

Completed flag

Flag to identify acknowledgment or negative acknowledgment

- 0: Acknowledgment;
- 1: Negative acknowledgment.

Network reset start flag

Flag to notify network reset start

- 0: Not notify;
- 1: Notify

Network reset start confirmation flag

Flag to notify network reset start confirmation

- 0: Not notify;
- 1: Notify.

Network reset start completed flag

Flag to notify network reset start completed

- 0: Not notify;
- 1: Notify.

- fpduBody field

This is a data field. The data format will be explained by command code in the following subclauses. In this subclause, node address common to all commands, which is included in the data, are specified. The configuration of destination address is the same as that of source address, as shown in Table 9.

Table 9 – Node address

Alignment [octet]	Name	Data type	Descriptions	Remarks
0 to 1	Node address	Unsigned16	ID number of the node '0000'H to 'FFEF'H: for unicast 'FFF0'H to 'FFFE'H: for multicast 'FFFF'H: for broadcast	An exclusive number within the network shall be allocated.

4.3.4 Detailed definitions of _FID-PDUs

4.3.4.1 SYNC command

The SYNC command shall be used for sync information notification from the sync master.

```

_SYNC-FIDAP-PDU      ::= _FIDAP-PDU ( WITH COMPONENTS
                        { ..., fcmd (sync), fHS,
                          fpduBody (fcmdBody _SYNC-FIDAPBody)
                        }
                      )

_SYNC-FIDAPBody      ::= SEQUENCE {
                        pDstAddress    Unsigned16,
                        pSrcAddress    Unsigned16,
                        pCycleCounter  Unsigned16,
                        reserve        Unsigned32,
                        pTimeStamp     Unsigned64,
                        pUserData      OCTET STRING SIZE (0..24)
                      }

```

- fcmd field
This field shall indicate SYNC command.
- fHS field
This is a data field to indicate handshake/notification with the following value range: '00'H: Normally (When not in use);
– '10'H (NRS): Network reset start;
– '40'H (NRS_COMP): Network reset start completed.
- pSrcAddress field
This field shall indicate the source node address. Set the sync master address (see Table 9).
- pCycleCounter field
This field shall be used to increment a counter by one every transmission cycle. This shall be used to handle multiple transmission cycles.
- pTimeStamp (TS_fsync) field
This field stores a time stamp. (Unit: 1 ns)
- pUserData field
This field stores user defined data. This is an option area. When sending a packet where sync command is stored, the common multicast address shall be used as the destination MAC address (see Table 4).

4.3.4.2 OUTPUT command

The OUTPUT command is used for command data from a master to slaves. There are two types of output data formats. The basic format shall be used for one node destination. The extended format shall be made by joining multiple basic formats to multiple node destinations. For user data, output data for the number of devices in the source node shall be stored.

```

_OUTPUT-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (output), fHS,
      fpduBody (fcmdBody _OUTPUT-FIDAPBody)
    }
  )

_OUTPUT-FIDAPBody ::= SEQUENCE {
    pDstAddress  Unsigned16,
    reserve      Unsigned2,
    pLength      Unsigned14,
    pUserData    _FDCServicePDU +
  }
| _OUTPUT-FIDAPBody -- Extended format
-- Basic format

```

- fcmd field
- This field shall indicate OUTPUT command. fHS field
This field shall indicate handshake/notification with the following value range:
 - '00'H: Normally (When not in use);
 - '10'H (NRS): Network reset start;
 - '20'H (NRS_CONF): Network reset start confirmation.
- pDstAddress field
This field shall indicate the destination node address. The address of slaves to receive output data shall be set (see Table 9).
- pLength field
This field shall indicate the user data octet number.
- pUserData field
This field stores output data. FDCServicePDU shall be stored as the output data by device for all the devices (see 4.3.5).

4.3.4.3 INPUT command

The INPUT command is used for command data from slaves to a master. For user data, input data for the number of devices in the source node shall be stored.

```

_INPUT-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (input), fHS,
      fpduBody (fcmdBody _INPUT-FIDAPBody)
    }
  )

_INPUT-FIDAPBody ::= SEQUENCE {
    pSrcAddress  Unsigned16,
    reserve      Unsigned2,
    pLength      Unsigned14,
    pUserData    _FDCServicePDU +
  }

```

- fcmd field
This field shall indicate INPUT command.

- fHS field
This field shall indicate handshake/notification with the following value range:
 - '00'H: Normally (When not in use);
 - '10'H (NRS): Network reset start.
- pSrcAddress field
This field shall indicate the source node address. Address of the slaves which sent input data shall be set (see Table 9).
- pLength field
This field shall indicate the user data octet number.
- pUserData field
This field stores input data. _FDCServicePDU shall be stored as the input data by device for all the devices (see 4.3.5).

4.3.4.4 TOKEN command

The TOKEN command is used to notify receiving/giving of the send right between masters.

```

_TOKEN-FIDAP-PDU      ::= _FIDAP-PDU ( WITH COMPONENTS
                        { ..., fcmd (token), fHS,
                          fpduBody (fcmdBody _TOKEN-FIDAPBody)
                        }
                        )

_TOKEN-FIDAPBody      ::= SEQUENCE {
                        pDstAddress      Unsigned16,
                        pSrcAddress      Unsigned16,
                        reserve          Unsigned8,
                        pTokenType       Usingned8
                        }
  
```

- fcmd field
This field shall indicate TOKEN command.
- fHS field
This field shall indicate handshake/notification with the following value range:
 - '00'H: Normally (not used).
- pDstAddress field
This field shall indicate the destination node address to which the send right is given. The destination master address to which the send right is given shall be set (see Table 9).
- pSrcAddress field
This field shall indicate the source node address that gives the send right. The source master address which gives the send right shall be set (see Table 9).
- reserve
This is a reserve area. Set to 0.
- pTokenType field
This field shall indicate token types with the following value range:
 - 1 (MST_MEAS_DLY): Master delay measurement;
 - 2 (CONFIG): Configuration.

4.3.4.5 REQ_SEND command

The REQ_SEND command shall be used when making a request to send a packet. There are three types of send request formats by request type. They shall be respectively used as the send command for input data retransmission request, delay measurement request, sync notification and network reset start notification.

```

_REQ_SEND-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (req_send), fHS,
      fpduBody (fcmdBody _REQ_SEND-FIDAPBody)
    }
  )
_REQ_SEND-FIDAPBody ::= SEQUENCE {
    reserve1      Unsigned16,
    pPollingType  Unsigned8,
    pNumAddr      Unsigned8,
    ( pReqDstAddress Unsigned16 )
  }
  | SEQUENCE {
    pDstAddress    Unsigned16,
    pPollingType   Unsigned8,
    pCalcType      Unsigned8,
    pSyncAddr      Unsigned16,
    pSyncMACAddr   Unsigned48
  }
  | SEQUENCE {
    pDstAddress    Unsigned16,
    pPollingType   Unsigned8,
    reserve2       Unsigned8
  }
  }

```

-- Input data retransmission

-- Delay measurement request

-- Sync notification and network reset start notification

- fcmd field
This field shall indicate SEND command.
- fHS field
This field shall indicate handshake/notification with the following value:
 - '00'H: Normally (not used).
- reserve1
This is a reserve area. Set to 0.
- pPollingType field
This field shall indicate request types with the following value range:
 - 1 (INPUT): Input data retransmission;
 - 2 (MEAS_DLY): Delay measurement request;
 - 3 (SYNC): Sync notification;
 - 4 (NET_NOTI): Network reset start notification.
- pNumAddr field
This field shall indicate the number of addresses and this is valid when the request type is 1. The number of stored addresses shall be set.

- **pReqDstAddress field**
This field shall indicate the node address of the input data transmission request destination and this is valid when the request type is 1. The address of the slave which requests the input data retransmission shall be set (see Table 9).
- **pDstAddress field**
This field shall indicate the destination node address, and this is valid when the request type is 2, 3, or 4. The address of the slave to receive sending request shall be set (see Table 9).
- **pCalcType field**
This field shall indicate the calculation type of the delay measurement, and this is valid when the request type is 2. It takes the following values range:
 - 0 (INIT): Initialization processing
 - 1 (PAP): Plug-and-play joining processing
- **pSyncAddr field**
This field shall indicate the sync master address as the reply destination, and this is valid when the request type is 2. The sync master address shall be set (see Table 9).
- **pSyncMACAddr field**
This field shall indicate the sync master MAC address as the reply destination, and this is valid when the request type is 2. The sync master MAC address shall be set.
- **reserve2**
This is a reserve area. Set to 0.

4.3.4.6 MEAS_DLY command and response

The MEAS_DLY shall be used for command and response of delay measurements of transmission paths (see 8.3.4.1).

```

_MEAS_DLY-CMD-      ::= _FIDAP-PDU ( WITH COMPONENTS
_FIDAP-PDU
    { ..., fcmd (measure_dly), fHS,
      fpduBody (fcmdBody  _MEAS_DLY-CMD-FIDAPBody)
    }
  )

_MEAS_DLY-CMD-      ::= SEQUENCE {
_FIDAPBody
    pDstAddress      Unsigned16,
    pSrcAddress      Unsigned16,
    reserve          Unsigned16,
    pTimeStamp3      Unsigned32
  }

_MEAS_DLY-RSP-      ::= _FIDAP-PDU ( WITH COMPONENTS
_FIDAP-PDU
    { ..., fcmd (measure_dly), fHS,
      fpduBody (fcmdBody  _MEAS_DLY-RSP-FIDAPBody)
    }
  )
    
```

```

_MEAS_DLY-RSP-      ::= SEQUENCE {
_FIDAPBody
    pDstAddress      Unsigned16,
    pSrcAddress      Unsigned16,
    reserve          Unsigned16,
    pTimeStampT3     Usingned32,
    pTimeStampT4     Usingned32
}

```

- **fcmd field**
This field shall indicate MEAS_DLY.
- **fHS field**
This field shall indicate handshake/notification with the following value range to identify command or response:
 - '00'H (REQ): Request;
 - '01'H (ACK): Acknowledgment;
 - '03'H (NACK): Negative acknowledgment.
- **pDstAddress field**
This field shall indicate the destination node address. The sync master address or the broadcast address shall be set (see Table 9).
- **pSrcAddress field**
This field shall indicate the source node address. The source node address shall be set (see Table 9).
- **reserve**
This is a reserve area. Set to 0.
- **pTimeStampT3 field**
This field shall indicate the time stamp T3. The time stamp upon sending of the delay measurement request shall be set. (Unit: 1 ns)
- **pTimeStampT4 field**
This field shall indicate the time stamp T4. The time stamp upon receiving of the delay measurement request shall be set. (Unit: 1 ns)

4.3.4.7 READ_CDO command and response

The READ_CDO shall be used for a DLO ID read command/response.

```

_READ_CDO-CMD-      ::= _FIDAP-PDU( WITH COMPONENTS
_FIDAP-PDU
    { ..., fcmd (read_cdo), fHS,
      fpduBody (fcmdBody  _READ_CDO-CMD-FIDAPBody)
    }
)

_READ_CDO-CMD-      ::= SEQUENCE {
_FIDAPBody
    pDstAddress      Unsigned16,
    pSrcAddress      Unsigned16,
    reserve          Unsigned8,
    pNumCDO_ID       Usingned8,
    ( pCDO_ID        Usingned16 ) +
}

```

```

_READ_CDO-RSP- ::= _FIDAP-PDU ( WITH COMPONENTS
FIDAP-PDU
    { ..., fcmd (read_cdo), fHS,
      fpduBody (fcmdBody _READ_CDO-RSP-FIDAPBody)
    }
  )

_READ_CDO-RSP- ::= SEQUENCE {
FIDAPBody
    pDstAddress    Unsigned16,
    pSrcAddress    Unsigned16,
    reserve        Unsigned8,
    pNumCDO_ID    Usingned8,
    ( SEQUENCE {
        pCDO_ID    Usingned16,
        pResult    Usingned16,
        pCDO_Data  OCTET STRING SIZE (1..48),
        padding    OCTET STRING SIZE (0..3)
    } ) +
  }

```

- fcmd field
This field shall indicate READ_CDO.
- fHS field
This field shall indicate handshake/notification with the following value range to identify command or response:
 - '00'H (REQ): Request;
 - '01'H (ACK): Acknowledgment;
 - '03'H (NACK): Negative acknowledgmente.
- pDstAddress field
This field shall indicate the destination node address. For a command, the read destination node address shall be set. For a response, the source node address shall be set (see Table 9).
- pSrcAddress field
This field shall indicate the source node address. The source node address shall be set (see Table 9).
- reserve
This is a reserve area. Set to 0.
- pNumCDO_ID field
This field shall indicate the read target CDO_ID number. The stored CDO_ID number shall be set.
- pCDO_ID field
This field shall indicate the read target CDO_ID. CDO_ID shall be set.
- pResult field
This field shall indicate the result of CDO ID read with the following value range:
 - 0: Normally completed;
 - 1: CDO_ID error (corresponding COD_ID does not exist);
 - 2: Cannot read (reading of CDO_ID with a write only attribute);

– 3: Reading of unimplemented CDO_ID.

- pCDO_Data field

This field shall indicate the CDO ID value. The value that has been read according to the CDO ID data size shall be set.

- padding field

This shall be inserted to become a 4-octet border according to the CDO ID data size. Set to 0.

4.3.4.8 WRITE_CDO command and response

The WRITE_CDO shall be used for a DLO ID read request/response.

```

_WRITE_CDO-CMD-      ::= _FIDAP-PDU ( WITH COMPONENTS
FIDAP-PDU
    { ..., fcmd (write_cdo), fHS,
      fpduBody (fcmdBody  _WRITE_CDO-CMD-FIDAPBody)
    }
  )

_WRITE_CDO-CMD-      ::= SEQUENCE {
FIDAPBody
    pDstAddress    Unsigned16,
    pSrcAddress    Unsigned16,
    reserve1       Unsigned8,
    pNumCDO_ID    Usingned8,
    ( SEQUENCE {
        pCDO_ID    Usingned16,
        reserve2    Usingned16,
        pCDO_Data  OCTET STRING SIZE (1..48),
        padding     OCTET STRING SIZE (0..3)
      })
  }

_WRITE_CDO-RSP-      ::= FIDAP-PDU ( WITH COMPONENTS
FIDAP-PDU
    { ..., fcmd (write_cdo), fHS,
      fpduBody (fcmdBody  _WRITE_CDO-RSP-FIDAPBody)
    }
  )

_WRITE_CDO-RSP-      ::= SEQUENCE {
FIDAPBody
    pDstAddress    Unsigned16,
    pSrcAddress    Unsigned16,
    reserve1       Unsigned8,
    pNumCDO_ID    Usingned8,
    ( SEQUENCE {
        pCDO_ID    Usingned16,
        pResult     Usingned16 )
      }) +
  )

```

- fcmd field

This field shall indicate WRITE_CDO.

- fHS field
This field shall indicate handshake/notification with the following value range to identify command or response:
 - '00'H (REQ): Request;
 - '01'H (ACK): Acknowledgment;
 - '03'H (NACK): Negative acknowledgment.
- pDstAddress field
This field shall indicate the destination node address. For a command, the write destination node address shall be set. For a response, the source node address shall be set (see Table 9).
- pSrcAddress field
This field shall indicate the source node address. The source node address shall be set (see Table 9).
- reserve1
This is a reserve area. Set to 0.
- pNumCDO_ID field
This field shall indicate the write target CDO_ID number. The stored CDO_ID number shall be set.
- pCDO_ID field
This field shall indicate the write target CDO_ID. CDO_ID shall be set.
- reserve2
This is a reserve area. Set to 0.
- pCDO_Data field
This field shall indicate the CDO_ID value. The value to write shall be set according to the CDO_ID data size.
- padding field
This shall be inserted to become a 4-octet border according to the CDO_ID data size. Set to 0.
- pResult field
This field shall indicate the result of CDO ID write with the following value range:
 - 0: Normally completed;
 - 1: CDO_ID error (corresponding CDO_ID does not exist);
 - 2: Cannot write (writing of CDO_ID with a read only attribute);
 - 3: Writing of unimplemented CDO_ID;
 - 4: Writing value error.

4.3.4.9 MSG command

The MSG command shall be used to notify receiving and giving of message between nodes.

```

_MSG-FIDAP-PDU      ::= _FIDAP-PDU ( WITH COMPONENTS
                        { ..., fcmd (msg), fHS,
                          fpduBody (fcmdBody _MSG-FIDAPBody)
                        }
                      )

_MSG-FIDAPBody      ::= SEQUENCE {
                        pDstAddress  Unsigned16,
                        pSrcAddress  Unsigned16,
                        pMsgControl  _MSG_CONTROL,
                        pUserData    _MSGServicePDU
                      }

```

- fcmd field
This field shall indicate MSG command.
- fHS field
This field shall indicate handshake/notification with the following value:
 - '00'H: Normally (not used).
- pDstAddress field
This field shall indicate the destination node address. The destination node address shall be set (see Table 9).
- pSrcAddress field
This field shall indicate the source node address. The source node address shall be set (see Table 9).
- pMsgControl field
This field shall indicate the message control information. There are two types of message control information formats: I-format and S-format.

```

_MSG_CONTROL ::= SEQUENCE {                                     -- I-format
                    pNR      BIT STRING SIZE (7),
                    pPF      BIT STRING SIZE (1),
                    pNS      BIT STRING SIZE (1),
                    pType    BIT STRING SIZE (1)
                  }

| SEQUENCE {                                                  -- S-format
                    pNR      BIT STRING SIZE (7),
                    reserve1 BIT STRING SIZE (1),
                    reserve2 BIT STRING SIZE (4),
                    pStype   BIT STRING SIZE (2),
                    reserve3 BIT STRING SIZE (1),
                    pType    BIT STRING SIZE (1)
                  }

```

pNR

A field to indicate the serial number of receiving data.

pPF

A flag to identify polling/final fragment.

- 0: Polling;
- 1: Final fragment.

pNS

A field to indicate the serial number of sending data.

pType

A flag to identify the format.

- 0: I-format;
- 1: S-format.

reserve1

A reserve area. Set to 1.

reserve2

A reserve area. Set to 0.

pSType

A field to indicate the S-format frame type.

reserve3

A reserve area. Set to 0.

- pUserData field

This field shall store the message. It shall store a part of divided _MSGServicePDU or _MSGServicePDU (see 4.3.6).

4.3.4.10 ASYNC_IND command

The ASYNC_IND command shall be used for an event notification.

```

_ASYNC_IND-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (async_ind), fHS,
      fpduBody (fcmdBody _ASYNC_IND-FIDAPBody)
    }
)

_ASYNC_IND-FIDAPBody ::= SEQUENCE {
    pDstAddress  Unsigned16,
    pSrcAddress  Unsigned16,
    pEventType   Unsigned8,
    pParam       _ASYNC_IND_PARAM,
    pUserData    _ASYNC_IND_DATA | empty
}
    
```

- fcmd field

This field shall indicate ASYNC_IND command.

- **fHS field**
This field shall indicate handshake/notification with the following value:
 - '00'H: Normally (not used).
- **pDstAddress field**
This field shall indicate the destination node address. The destination master address shall be set. Note that this shall be set to broadcast for the event type 1 (see Table 9).
- **pSrcAddress field**
This field shall indicate the source node address. The source master address shall be set (see Table 9).
- **pEventType field**
This field shall indicate the event type with the following value range:
 - 1 (NEI_NOTI): Neighbouring node notification;
 - 2 (SUP_NOTI): Start-up notification;
 - 3 (SUP_RESP): Start-up notification response);
 - 4 (NEI_LIST): Neighbouring node list notification;
 - 5 (MEAS_DLY): Delay measurement;
 - 6 (MEAS_DLY_COMP): Delay measurement completed;
 - 7 (CHG_ADDR): Node address change;
 - 8 (NRS_COMP): Network reset start completed.
- **pParam field**
This field shall indicate the parameter, and this is valid for event type 3. For other event types, this shall be set to 0.

```

_ASYNC_IND_PARAM ::= SEQUENCE {
    pNodeType      BIT STRING SIZE (4),
    pPossibilitySM  BIT STRING SIZE (1),
    pMeasureDly    BIT STRING SIZE (1),
    reserve        BIT STRING SIZE (2)
}

```

pNodeType

A field to indicate the operation type. .

- 1: Master;
- 2: Slave.

pPossibilitySM

A flag to identify SM handling capability.

- 0: With SM capability;
- 1: Without SM capability.

pMeasureDly

This is a flag to identify the master delay measurement completion and this shall be set only for a master. For a slave, this shall be set to 0.

- 0: Master delay measurement unimplemented;
- 1: Master delay measurement completed.

reserve

This is a reserve area. Set to 0,

- **pUserData field**

This field shall store the notification content, and this is valid for event type 2, event type 3, event type 6 and event type 7. This field shall not be added when it is not valid. There are four types of formats by notification content.

```

_ASYNC_IND_DATA ::= SEQUENCE {
    pSMDlyTime    Unsigned32,
    pNumStaInfo    Unsigned32,
    ( SEQUENCE {
        pPortNum    Unsigned8,
        reserve      Unsigned8,
        pMACAddr     Unsigned48
    } ) +
}
| SEQUENCE {
    pRemainNum    Unsigned8,
    reserve        Unsigned8,
    pNumNode       Unsigned16,
    ( pNodeInfo     _ASYNC_IND_NODE_INFO ) +
}
| ( pDlyTime      Unsigned32 )
| SEQUENCE {
    pTagMACAddr    Unsigned48,
    pTagAddr        Unsigned16
}

```

-- Start-up notification response

-- Neighbouring node list notification

-- Delay measurement completed

-- Node address change

pSMDlyTime

This field shall indicate the SM delay time, and this is valid for event type 3. The SM delay time shall be set. When measurement is not executed or not registered, this shall be set to 0. (Unit: 1 ns)

pNumStaInfo

This field shall indicate the number of neighbouring node information and this is valid for event type 3. The number of stored neighbouring node information shall be set.

pPortNum

This field shall indicate the port number of neighbouring node and this is valid for event type 3.

reserve

This is a reserve area. Set to 0. .

pMACAddr

This field shall indicate the MAC address of neighbouring node and this is valid for event type 3.

pRemainNum

This field shall indicate the number of remaining packets and this is valid for event type 6. The number of remaining packets after dividing into multiple packets shall be set.

pNumNode

This field shall indicate the total number of nodes and this is valid for event type 6. The total number of nodes including the master shall be set. (Maximum number: 128)

pNodeInfo

This field shall store the node information, and this is valid for event type 6.

```

_ASYNC_IND_NODE_INFO ::= SEQUENCE {
    pStaAddr      Unsigned16,
    reserve       Unsigned8,
    pNumStaInfo   Unsigned8,
    ( SEQUENCE {
        pPortNum   Unsigned8,
        reserve    Unsigned8,
        pStaAddr   Unsigned16
    } ) +
}

```

pStaAddr

This field shall store the node address.

reserve

This is a reserve area. Set to 0.

pNumStaInfo

This field shall indicate the number of neighbouring node information. The number of stored neighbouring node information shall be set.

pNodeInfo

This field shall indicate the port number of neighbouring node.

pStaAddr

This field shall store the node address of neighbouring node.

pMDlyTime

This field shall indicate the master delay time and this is valid for event type 6. The master delay time shall be set. When measurement is not executed or not registered, this shall be set to 0. (Unit: 1 ns)

pTagMACAddr

This field shall indicate the MAC address of node address change target node and this is valid for event type 7. The MAC address of the node address change target node shall be set.

pTagAddr

This field shall indicate the node address of node address change target node and this is valid for event type 7. The node address of the node address change target node shall be set.

NOTE When sending event type 1, the multicast address with the neighbouring node as the destination shall be used as the destination MAC address. For other event types, the common multicast address shall be used as the destination MAC address (see Table 4).

4.3.4.11 EVD command

The EVD command shall be used for an event driven notification.

```

_EVD-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (evd), fHS,
      pduBody (fcmdBody _EVD-FIDAPBody)
    }
)

_EVD-FIDAPBody ::= SEQUENCE {
    pDstAddress  Unsigned16,
    pSrcAddress  Unsigned16,
    pLength      Unsigned16,
    reserve      Unsigned16,
    pUserData    _FDCServicePDU +
}

```

- fcmd field
This field shall indicate EVD command.
- fHS field
This field shall indicate handshake/notification with the following value:
 - '00'H: Normally (not used).
- pDstAddress field
This field shall indicate the destination node address. The destination master address shall be set (see Table 9).
- pSrcAddress field
This field shall indicate the source node address. The source master address shall be set (see Table 9).
- pLength field
This field shall indicate the number of user data octets.
- reserve
This is a reserve area. Set to 0.
- pUserData field
This field shall store user data. Store _FDCServicePDU (see 4.3.5).

4.3.5 PDUs for field device control service

4.3.5.1 Overview

FDCServicePDU shall be edited with a SDU in the FDC service primitive and be sent via AR. And a FDC ASE receives the PDU from the peer ASE via AR.

The FDC command PDU type: _CMD_PDU shall be classified into sync type command (sync command): _SYNCType-CMD-PDU and async type command (async command): _ASYNCType-CMD-PDU. Both types are the identical PDU formats: _UCMD-PDU. Therefore, the FDC PM cannot distinguish their type's PDUs. The user of FDC ASE should know which commands belong to the sync commands or not, and can distinguish the types by using cmd field as a key code in the PDU header part.

The async command PDU: _ASYNCType-CMD_PDU shall include _UCMD-PDU and other fourteen PDU types as common commands, which are also variants of _UCMD-PDU.

Derived from the _UCMD-PDU and the _URSP-PDU explained later, a set of commands and responses can be defined for a specific application, such as motor-drive control, servo control, etc., by FAL users, and the set can be registered as a device profile (see 4.5 and Annex A). In addition, a FAL user can select any device profiles supported by a target field device, when a connection is established by FDC-Connect service (see 8.4.4.3, and IEC 61158-5-27).

_CMD-PDU	::=	_SYNCType-CMD-PDU _ASYNCType-CMD-PDU
_SYNCType-CMD-PDU	::=	_UCMD-PDU
_ASYNCType-CMD-PDU	::=	_UCMD-PDU _NOP-CMD-PDU _PRM_RD-CMD-PDU _PRM_WR-CMD-PDU _ID_RD-CMD-PDU _CONFIG-CMD-PDU _ALM_RD-CMD-PDU _ALM_CLR-CMD-PDU _SYNC_SET-CMD-PDU _CONNECT-CMD-PDU _DISCONNECT-CMD-PDU _PPRM_RD-CMD-PDU _PPRM_WR-CMD-PDU _MEM_RD-CMD-PDU _MEM_WR-CMD-PDU
_UCMD-PDU	::=	_CMD3-PDU
_NOP-CMD-PDU	::=	_NOP-CMD3-PDU
_PRM_RD-CMD-PDU	::=	_PRM_RD-CMD3-PDU
_PRM_WR-CMD-PDU	::=	_PRM_WR-CMD3-PDU
_ID_RD-CMD-PDU	::=	_ID_RD-CMD3-PDU
_CONFIG-CMD-PDU	::=	_CONFIG-CMD3-PDU
_ALM_RD-CMD-PDU	::=	_ALM_RD-CMD3-PDU
_ALM_CLR-CMD-PDU	::=	_ALM_CLR-CMD3-PDU
_SYNC_SET-CMD-PDU	::=	_SYNC_SET-CMD3-PDU
_CONNECT-CMD-PDU	::=	_CONNECT-CMD3-PDU
_DISCONNECT-CMD-PDU	::=	_DISCONNECT-CMD3-PDU
_PPRM_RD-CMD-PDU	::=	_PPRM_RD-CMD3-PDU
_PPRM_WR-CMD-PDU	::=	_PPRM_WR-CMD3-PDU
_MEM_RD-CMD-PDU	::=	_MEM_RD-CMD3-PDU
_MEM_WR-CMD-PDU	::=	_MEM_WR-CMD3-PDU

The FDC response PDU type: _RSP_PDU shall be also classified into sync type response (sync response): _SYNCType-RSP-PDU and async type response (async response): _ASYNCType-RSP-PDU. Both types shall be derived from the identical PDU formats: _URSP-PDU. Therefore, the FDC PM cannot distinguish their type's PDUs. The user of FDC ASE should know which responses belong to the sync responses or not, and can distinguish the types by using rcmd field as a key code in the PDU header part. Moreover, this prefix code of a response PDU corresponds to one of a command PDU, since a response shall be exchanged for a command.

The async type response PDU: _ASYNCType-RSP_PDU shall include _URSP-PDU and other fourteen PDU types as common responses to any application, which are also variants of _URSP-PDU.

_RSP-PDU	::= _SYNCType-RSP-PDU _ASYNCType-RSP-PDU
_SYNCType-RSP-PDU	::= _URSP-PDU
_ASYNCType-RSP-PDU	::= _URSP-PDU _NOP-RSP-PDU _PRM_RD-RSP-PDU _PRM_WR-RSP-PDU _ID_RD-RSP-PDU _CONFIG-RSP-PDU _ALM_RD-RSP-PDU _ALM_CLR-RSP-PDU _SYNC_SET-RSP-PDU _CONNECT-RSP-PDU _DISCONNECT-RSP-PDU _PPRM_RD-RSP-PDU _PPRM_WR-RSP-PDU _MEM_RD-RSP-PDU _MEM_WR-RSP-PDU
_URSP-PDU	::= _RSP3-PDU
_NOP-RSP-PDU	::= _NOP-RSP3-PDU
_PRM_RD-RSP-PDU	::= _PRM_RD-RSP3-PDU
_PRM_WR-RSP-PDU	::= _PRM_WR-RSP3-PDU
_ID_RD-RSP-PDU	::= _ID_RD-RSP3-PDU
_CONFIG-RSP-PDU	::= _CONFIG-RSP3-PDU
_ALM_RD-RSP-PDU	::= _ALM_RD-RSP3-PDU
_ALM_CLR-RSP-PDU	::= _ALM_CLR-RSP3-PDU
_SYNC_SET-RSP-PDU	::= _SYNC_SET-RSP3-PDU
_CONNECT-RSP-PDU	::= _CONNECT-RSP3-PDU
_DISCONNECT-RSP-PDU	::= _DISCONNECT-RSP3-PDU
_PPRM_RD-RSP-PDU	::= _PPRM_RD-RSP3-PDU
_PPRM_WR-RSP-PDU	::= _PPRM_WR-RSP3-PDU
_MEM_RD-RSP-PDU	::= _MEM_RD-RSP3-PDU
_MEM_WR-RSP-PDU	::= _MEM_WR-RSP3-PDU

4.3.5.2 Common component of PDU

4.3.5.2.1 FDC command PDU

_CMD-PDU has enhanced type: _CMD3-PDU.

The pduBody length of _CMD3-PDU shall be able to be select from 4, 12, 28, 44, and 60 octets, but need not be switched on the fly.

```

_CMD3-PDU ::= SEQUENCE {
    cmd      _CMDCode,
    wdt      _WDT,
    cmd_ctrl _CMD_CTRL,
    pduBody  OCTET STRING SIZE (4|12|28|44|60)
}
-- enhanced PDU type

```

- cmd field

cmd

data field which contains the code to identify contents of each command.

This field shall be coded as _CMDCode, subtype of Unsigned8, with the following value range. The predefined common commands are shown as _PrimaryCMD below.

- '00'H to '1F'H: used or reserved for common commands;
- '20'H to 'BF'H: reserved for application specific commands;
- 'C0'H to 'FF'H: reserved for vendors.

```

_CMDCode ::= Unsigned8
_PrimaryCMD ::= _CMDCode (
    nop ('00'H),
    prm_rd ('01'H), prm_wr ('02'H),
    id_rd ('03'H),
    config ('04'H),
    alm_rd ('05'H), alm_clr ('06'H),
    sync_set ('0D'H),
    connect ('0E'H),
    disconnect ('0F'H),
    pprm_rd ('1B'H), pprm_wr ('1C'H),
    mem_rd ('1D'H), mem_wr ('1E'H)
)

```

- wdt field

wdt

watchdog counter field

This field shall be used for the slave to detect out-of synchronous activity or the WDT error of the corresponding master. The master shall count the wdt field of sending PDUs every communication cycle in case of the synchronous communication state. In addition, the slave shall examine the field value of receiving PDUs every communication cycle to detect the wdt error.

The Data type is _WDT. It shall have two sub fields, mn and sn.

```

_WDT ::= SEQUENCE {
    mn  BIT STRING SIZE (4),
    sn  BIT STRING SIZE (4)
}

```

mn (Master count value)

The value range shall be from 0 to 15.

The mn field in a next CMD-PDU to be sent shall be counted up by one. An FDC master shall do that every communication cycle. On the other hand, when an FDC slave receives the PDU, it shall examine the counter every cycle. And, if the counter isn't match the last mn value plus one, it shall alert the watchdog counter error.

sn (Slave count value)

The value range shall be from 0 to 15.

The sn field in a next CMD-PDU to be sent shall be set the same value of the rsn field in the last received RSP-PDU. An FDC master shall copy it from the last RSP-PDU every communication cycle.

- cmd_ctrl field

cmd_ctrl

set of control bits that is independent of any command transactions, from FDC master to FDC slave

The Data type is `_CMD_CTRL`. It shall have two meaningful sub-fields, `alm_clr` and `cmd_id`.

```
_CMD_CTRL ::= SEQUENCE {
    reserve1  BIT STRING SIZE (3),
    alm_clr   BIT STRING SIZE (1),
    reserve2  BIT STRING SIZE (2),
    cmd_id    BIT STRING SIZE (2),
    reserve3  BIT STRING SIZE (8)
}
```

alm_clr

This field shall be used to command to clear alarm/warning status of the slave device.

- value 0: to execute nothing;
- value changes to 1: to execute the status clear.

cmd_id

When the master device continuously issues identical contents of a CMD-PDU to the slave, it can be used to make the slave device recognizing the command as another command.

The value range shall be from 0 to 3.

The FDC master can use the `cmd_id` to have a FDC slave acknowledge that the command is a new command when the master sends the same command repeatedly to the slave.

Since the slave shall respond the echo of the `cmd_id` of every command through RSP-PDU.`cmd_stat.rcmd_id` field explained later, the master can also judge the command for which the slave station sent the response.

Use of the `cmd_id` is not mandatory for the master.

While `RSP-PDU.cmd_stat.cmdRdy = 0` in response PDUs, the slave shall disregard commands that have a different `cmd_id` and continues to execute the command which has been already accepted.

4.3.5.2.2 FDC response PDU

_RSP-PDU has enhanced type: _RSP3-PDU.

The pduBody length of _RSP3-PDU shall be able to be select from 4, 12, 28, 44, and 60 octets, but need not be switched on the fly.

```
_RSP3-PDU ::= SEQUENCE {
    rcmd      _CMDCode,
    rwdt      _RWDT,
    cmd_stat  _CMD_STAT,
    pduBody   OCTET STRING SIZE (4|12|28|44|60)
}
-- enhanced PDU type
```

- rcmd field

rcmd

data field which contains the same code as a received and processing command

So, this field shall be coded as _CMDCode, see 4.3.5.2.1.

- alarm field

alarm

This field shall contain the alarm code.

The Data type is _ALARM. The value range shall be from 0 to 255.

```
_ALARM ::= Unsigned8
```

- status field

status

This field shall indicate the status of the slave device.

The Data type is _STATUS. It shall have three meaningful sub-fields as follows.

```
_STATUS ::= SEQUENCE {
    alarm      BIT STRING SIZE (1),
    warning    BIT STRING SIZE (1),
    cmdRdy     BIT STRING SIZE (1),
    reserve    BIT STRING SIZE (13)
}
```

alarm

This bit field shall indicate the alarm status in the slave device.

- 0: No alarm;
- 1: any alarms occur.

warning

This bit field shall indicate the warning status in the slave device.

- 0: No warning;
- 1: any warnings occur.

cmdRdy

This bit field shall indicate the command progress status of the FDC slave.

The slave device shall keep the bit cmdRdy 0, while it is processing the command. The slave device shall change the bit from 0 to 1, when it has completed the processing.

To notice the completion of the specific command the FDC master shall not only examine the bit, but also collate some other RSP-PDU fields with the corresponding CMD-PDU fieldsto discriminate from the preceding transaction.

When the retention time of cmdRdy = 0 is expired, the master shall raise a command time out error. The timer value depends on each product specification for slave devices.

A change of this bit status shall be independent of the alarm or warning status.

- 0: busy with command execution in progress,
- 1: ready for new command.

• rwdt field

rwdt

watchdog counter field

This field shall be used for the master to detect out-of synchronous activity or the WDT error of the corresponding slave. The slave shall count and update the rwdt field of sending PDUs every communication cycle in case of the synchronous communication state. In addition, the master shall examine the field value of receiving PDUs every communication cycle to detect the rwdt error.

The Data type is `_RWDT`. It shall have two sub fields, `rmn` and `rsn`.

```
_RWDT ::= SEQUENCE {
    rmn  BIT STRING SIZE (4),
    rsn  BIT STRING SIZE (4)
}
```

rmn (Master count value)

The value range shall be from 0 to 15.

The rmn field in a next RSP-PDU to be sent shall be set the same value of the mn field in the last received CMD-PDU. An FDC slave shall copy it from the CMD-PDU every communication cycle.

rsn (Slave count value)

The value range shall be from 0 to 15.

The rsn field in a next RSP-PDU to be sent shall be counted up by one. An FDC slave shall do that every communication cycle. On the other hand, when an FDC master receives the PDU, it shall examine the counter every cycle. And, if the counter isn't match the last rsn value plus one, it shall alert the watchdog counter error.

- cmd_stat field

cmd_stat

This field shall indicate the status of the slave device.

The Data type is _CMD_STAT. It shall have seven sub fields as follows.

```
_CMD_STAT ::= SEQUENCE {
    d_alm      BIT STRING SIZE (1),
    d_war      BIT STRING SIZE (1),
    cmdRdy     BIT STRING SIZE (1),
    alm_clr_cmp BIT STRING SIZE (1),
    reserve    BIT STRING SIZE (2),
    rcmd_id    BIT STRING SIZE (2),
    cmd_alm    BIT STRING SIZE (4),
    comm_alm   BIT STRING SIZE (4)
}
```

d_alm

This bit field shall indicate occurrence status of device-specific alarm in the slave device.

- 0: the slave is not in device specific alarm status;
- 1: the slave is in device specific alarm status except neither comm_alm nor cmd_alm.

Specification of device alarm status depends on specified product implementation, but a factor of the alarms should be classified separately from a communication problem and a command issue.

After the slave has been recovered to the alarm status, through the execution of the command ALM_CLR-CMD-PDU or cmd_ctrl.alm_clr bit in any CMD-PDU, this bit shall be cleared to 0.(d_alm = 0)

d_war

This bit field shall indicate occurrence status of device-specific warning in the slave station.

- 0: the slave is not in device specific alarm status;
- 1: the slave is in device specific alarm status except neither comm_alm nor cmd_alm.

Specification of device alarm status depends on specified product implementation, but a factor of the alarms should be classified separately from a communication problem and a command issue.

After the slave has been recovered to the warning status, through the execution of the command ALM_CLR-CMD-PDU or cmd_ctrl.alm_clr bit in any CMD-PDU, this bit shall be cleared to 0.(d_war = 0)

cmdRdy

This bit field shall indicate the command progress status of the FDC slave.

Meaning of this bit is status.cmdRdy.

- 0: busy with command execution in progress;
- 1: ready for new command.

alm_clr_cmp

This bit field shall indicate the execution status of alarm clear process.

- 0: Alarm/warning clear not completed;
- 1: Alarm/warning clear completed.

rcmd_id

This field shall be a key code to indicate which command the response PDU corresponds to, by echoing back cmd_id of the corresponding command PDU.

The value range is from 0 to 3.

cmd_alm

This field shall notify an alarm code for command abnormality.

Each code shall mean as follows.

- '00'H: Normal;
- '01'H: Warning on out-of-range data;
- '02'H to '07'H: reserved for warning codes;
- '08'H: Alarm on out-of-support;
- '09'H: Alarm on out-of-range data;
- '0A'H: Alarm on abnormal command execution condition;
- '0B'H: Alarm on abnormal subcommand combination;
- '0C'H: Alarm on abnormal phase;
- '0D'H to '0F'H: reserved for alarm codes.

comm_alm

This field shall notify an alarm code for communication error status.

Each code shall mean as follows.

- '00'H: Normal;
- '01'H: Warning on abnormal FCS;
- '02'H: Warning on abnormal reception;
- '03'H to '07'H: reserved for warning;
- '08'H: Alarm on abnormal FCS;
- '09'H: Alarm on abnormal reception;
- '0A'H to '0F'H: reserved for alarm.

4.3.6 PDUs for message service

```

_MSGREQ-PDU ::= SEQUENCE {
    responder      Unsigned8,
    funcCode       BIT STRING SIZE (7),
    reserve1       BIT STRING SIZE (1),
    extAddress     Unsigned8,
    reserve2       OCTET STRING ('00'H),
    requestData    OCTET STRING SIZE (4..4092)
}
| requestUData    OCTET STRING SIZE (8..4096)
_MSGRSP-PDU ::= SEQUENCE {
    responder      Unsigned8,
    funcCode       BIT STRING SIZE (7),
    errorFlag      BIT STRING SIZE (1),
    extAddress     Unsigned8,
    reserve        OCTET STRING ('00'H),
    responseData   OCTET STRING SIZE (4..4092)
}
| responseUData   OCTET STRING SIZE (8..4096)

```

- responder field

responder

This field shall contain the destination node address or responder of the message.

In this field, the value of octet 0 of the node address shall be set (see Table 9).

- funcCode field

funcCode

This field shall contain the code indicating the function of the message.

The code range shall be from '00'H to '7F'H.

- errorFlag field

errorFlag

This field shall indicate the response state of the message. It shall be set to 0 on a normal response, and to 1 on an abnormal response.

- 0: a normal response;
- 1: an error response.

- extAddress field

extAddress

This field shall contain the sub address representing a sub device when the destination node is an integrated device with two or more sub devices.

In this field, the value of octet 0 of the node address shall be set (see Table 9).

- requestData field and responseData field

requestData and responseData

These fields shall contain the SDUs that is defined by user for each function code. An example of a user message command set for function code: '42'H is shown in Annex C.

4.4 Detailed definitions of _FDCService-PDUs

4.4.1 Enhanced PDU type

4.4.1.1 NOP command and response

NOP symbolizes a "no operation" command. Receiving this command, the slave shall do nothing but respond with a NOP-RSP3-PDU showing the latest alarm and status.

```
_NOP-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (nop)
    } )
_NOP-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (nop)
    } )
```

4.4.1.2 PRM_RD command and response

PRM_RD symbolizes a "parameter-read" command. Receiving this command, the slave shall read a specified parameter value into PRM_RD-RSP3-PDU.parameter field and shall respond with it.

```
_PRM_RD-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (prm_rd),
      pduBody (cmdBody _PRM_RD-CMD3Body)
    } )
_PRM_RD-CMD3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    reserve  OCTET STRING SIZE (1|9|25|41|57)
}
_PRM_RD-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (prm_rd),
      pduBody (rspBody _PRM_RD-RSP3Body)
    } )
```

```

_PRm_RD-RSP3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    reserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}

```

- pNo field

pNo

This field shall contain the parameter number to read out.

The value range shall be from 0 to 65 535.

- pSize field

pSize

This field shall contain an octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter

This field shall contain the read data corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.4.1.3 PRM_WR command and response

PRM_WR symbolizes a "parameter-write" command. Receiving this command, the slave shall write a specified parameter value from PRM_WR-CMD3-PDU.parameter field and shall respond with the echo of it to tell completion of the command.

```

_PRm_WR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (prm_wr),
      pduBody (cmdBody _PRm_WR-CMD3Body)
    } )
_PRm_WR-CMD3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    reserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}
_PRm_WR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (prm_wr),
      pduBody (rspBody _PRm_WR-RSP3Body)
    } )
_PRm_WR-RSP3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    reserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}

```

- pNo field

pNo

This field shall contain the parameter number to write in.

The value range shall be from 0 to 65 535.

- pSize field

pSize

This field shall contain the octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter

This field shall contain the data to write corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.4.1.4 ID_RD command and response

ID_RD symbolizes a "device-ID-read" command. Receiving this command, the slave shall read a part of a specified item in the Device Information into ID_RD-RSP3-PDU.idData field and shall respond with it. The semantics and data types of the Device Information are dependent on each device model, as shown in Annex B.

```

_ID_RD-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (id_rd),
      pduBody (cmdBody _ID_RD-CMD3Body)
    } )

_ID_RD-CMD3Body ::= SEQUENCE {
    idCode      Unsigned8,
    idOffcet    Unsigned8,
    idSize      Unsigned16,
    reserve     OCTET STRING SIZE (0|8|24|40|56)
}

_ID_RD-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (id_rd),
      pduBody (rspBody _ID_RD-RSP3Body)
    } )

_ID_RD-RSP3Body ::= SEQUENCE {
    idCode      Unsigned8,
    idOffcet    Unsigned8,
    idSize      Unsigned16,
    idData      OCTET STRING SIZE (0|8|24|40|56)
}

```

- idCode field

idCode

This field shall contain the device-ID code to read out.

The value range shall be from 0 to 255, see B.3.2.

- '00'H: reserved;
- '01'H: Vender ID code;

- '02'H: Device code or device model code;
- '03'H: Device version;
- '04'H: Version of Device Information file;
- '05'H: Number of extended addresses;
- '06'H: Serial number of product;
- '07'H to '0F'H: reserved;
- '10'H: Supported device profile code (primary);
- '11'H: Supported device profile version (primary);
- '12'H: Supported device profile code (secondary);
- '13'H: Supported device profile version (secondary);
- '14'H: Supported device profile code (tertiary);
- '15'H: Supported device profile version (tertiary);
- '16'H: Minimum transmission cycle;
- '17'H: Maximum transmission cycle;
- '18'H: Granularity of transmission cycle;
- '19'H: Minimum communication cycle;
- '1A'H: Maximum communication cycle;
- '1B'H: Number of the transmittable octets;
- '1C'H: Number of the transmitted octets (current setting);
- '1D'H: Device profile code (current setting);
- '1E'H to '1F'H: reserved;
- '20'H: Supported communication mode list;
- '21'H: MAC address;
- '22'H to '2F'H: reserved;
- '30'H: Supported main command list;
- '31'H to '37'H: reserved;
- '38'H: Supported sub command list;
- '39'H to '3F'H: reserved;
- '40'H: Supported common parameter list;
- '41'H to '7F'H: reserved;
- '80'H: Name of main device;
- '81'H to '8F'H: reserved;
- '90'H: Name of sub device 1;
- '91'H to '97'H: reserved;
- '98'H: Version of sub device 1;
- '99'H to '9F'H: reserved;
- 'A0'H: Name of sub device 2;
- 'A1'H to 'A7'H: reserved;
- 'A8'H: Version of sub device 2;
- 'A9'H to 'AF'H: reserved;
- 'B0'H: Name of sub device 3;
- 'B1'H to 'B7'H: reserved;
- 'B8'H: Version of sub device 3;

- 'B9'H to 'BF'H: reserved;
- 'C0'H to 'FF'H: reserved for vender specific information.

- idOffset field

idOffset

This field shall contain the offset address of the device-ID to read out.

The value range shall be from 0 to 255.

- idSize field

idSize

This field shall contain the read octet size of the device-ID.

The value range shall be from 0 to 255.

- idData field

idData

This field shall contain the read data corresponding to the idCode. The data size, data structure and its meaning depend on the idCode, see B.3.2.

4.4.1.5 CONFIG command and response

CONFIG symbolizes a "configure-device" command. Receiving this command, the slave shall activate a set of parameters on RAM updated with a PAR_WR command and shall respond to tell completion of the command.

```

_CONFIG-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (config),
      pduBody (cmdBody _CONFIG-CMD3Body)
    } )
_CONFIG-CMD3Body ::= SEQUENCE {
    config_mode Unsigned8 {
        pActive (0),
        pAllSave (1),
        pReset (2)
    },
    reserve OCTET STRING SIZE (3|11|27|43|59)
}

_CONFIG-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (config),
      pduBody (rspBody _CONFIG-RSP3Body)
    } )
_CONFIG-RSP3Body ::= SEQUENCE {
    config_mode Unsigned8 {
        pActive (0),
        pAllSave (1),
        pReset (2)
    },
    reserve OCTET STRING SIZE (3|11|27|43|59)
}

```

- config_mode field

config_mode

This field shall contain the mode of device configuration.

This field shall be coded as data type Unsigned8 with the following value range.

- pActive ('00'H): to recalculate with the parameters and set up;
- pAllSave ('01'H): to write parameters into nonvolatile memory with batch;
- pReset ('02'H): to restore to the factory setting of parameters;
- '03'H to 'FF'H: reserved.

4.4.1.6 ALM_RD command and response

ALM_RD symbolizes an "alarm-read" command. Receiving this command, the slave shall read a set of alarm data that includes details of status of alarm and warning into ALM_RD-RSP3-PDU.alm_data field and shall respond with it. The semantics and data types of the alm_data are dependent on each device model.

```

_ALM_RD-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (alm_rd),
      pduBody (cmdBody _ALM_RD-CMD3Body)
    } )
_ALM_RD-CMD3Body ::= SEQUENCE {
    alm_rd_mode Unsigned16 {
        currentAlm (0),
        historyAlm (1),
        cAlmDetail (2),
        hAlmDetail (3)
    },
    alm_index Unsigned16 (0..11),
    reserve OCTET STRING SIZE (0|8|24|40|56)
}
_ALM_RD-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (alm_rd),
      pduBody (cmdBody _ID_RD-RSP3Body)
    } )
_ALM_RD-RSP3Body ::= SEQUENCE {
    alm_rd_mode Unsigned16 {
        currentAlm (0),
        historyAlm (1),
        cAlmDetail (2),
        hAlmDetail (3)
    },
    alm_index Unsigned16 (0..11),
    alm_data OCTET STRING SIZE (8|24|40|56)
}

```

- alm_rd_mode field

alm_rd_mode

This field shall contain the mode for the alarm reading.

This field shall be coded as data type Unsigned8 with the following value range.

- currentAlm ('00'H): to read out the current alarm/warning state;
- historyAlm ('01'H): to read out the alarm history;
- cAlmDetail ('02'H): to retrieve details of individual current alarm/warning information;
- hAlmDetail ('03'H): to retrieves details of individual alarm history information;
- '04'H to 'FF'H: reserved.

- alm_index field

alm_index

This field shall contain the index for reading point of alarm/warning.

The value range shall be from 0 to 11.

- alm_data field

alm_data

This field shall contain the read alarm status corresponding to alm_rd_mode.

4.4.1.7 ALM_CLR command and response

ALM_CLR symbolizes an "alarm-clear" command. Receiving this command, the slave shall clear internal alarm and warning status and shall respond to tell completion of the command.

This command cannot work to solve a cause of alarm or warning. The command should be issued after solved the real causes or problems, to recover the device status to normal.

```

_ALM_CLR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (alm_clr),
      pduBody (cmdBody _ALM_CLR-CMD3Body)
    } )
_ALM_CLR-CMD3Body ::= SEQUENCE {
    alm_clr_mode Unsigned16 (
        cAlmClear (0),
        hAlmClear (1)
    ),
    reserve OCTET STRING SIZE (2|10|26|42|58)
}
_ALM_CLR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (alm_clr),
      pduBody (rspBody _ALM_CLR-RSP3Body)
    } )
_ALM_CLR-RSP3Body ::= SEQUENCE {
    alm_clr_mode Unsigned16 (
        cAlmClear (0),
        hAlmClear (1)
    ),
    reserve OCTET STRING SIZE (2|10|26|42|58)
}

```

- alm_clr_mode field

alm_clr_mode

This field shall contain the mode for alarm clear.

This field shall be coded as data type Unsigned8 with the following value range.

- cAlmClear ('00'H): to clear the current alarm/warning status;
- hAlmClear ('01'H): to clear the alarm history;
- '02'H to 'FF'H: reserved.

4.4.1.8 SYNC_SET command and response

SYNC_SET symbolizes a "set into synchronous-state" command. Receiving this command, the slave shall make a transition of slave PM (FDCPM-S) from S2: AsyncConnected state into S3: SyncConnected state. And receiving the response, the master shall make a similar transition of master PM (FDCPM-M) from S2: AsyncConnected state into S3: SyncConnected state, see 8.4.4 and for more details.

```
_SYNC_SET-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (sync_set)
    } )
```

```
_SYNC_SET-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (sync_set)
    } )
```

4.4.1.9 CONNECT command and response

Exchange this command and response, the master and the slave shall establish a FDC AR connection. At that time, some communication options or parameters can be specified with the PDUs. They shall make their transition of each PM; FDCPM-M and FDCPM-S, see 8.4.4 and 8.4.5 for more details.

```
_CONNECT-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (connect),
      pduBody (cmdBody _CONNECT-CMD3Body)
    } )
```

```
_CONNECT-CMD3Body ::= SEQUENCE {
    ver            Unsigned8,
    com_mod        SEQUENCE {
        reserve1    BIT STRING SIZE (1),
        syncmode    BIT STRING SIZE (1),
        dtmode      BIT STRING SIZE (2),
        reserve2    BIT STRING SIZE (3),
        subcmd      BIT STRING SIZE (1)
    },
    com_time       Unsigned8,
    profile_type   Unsigned8,
    reserve        OCTET STRING SIZE (0|8|24|40|56)
}
```



```

_CONNECT-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (connect),
      pduBody (rspBody _CONNECT-RSP3Body)
    } )
_CONNECT-RSP3Body ::= SEQUENCE {
    ver            Unsigned8,
    com_mod        SEQUENCE {
        reserve1    BIT STRING SIZE (1),
        syncmode    BIT STRING SIZE (1),
        dtmode      BIT STRING SIZE (2),
        reserve2    BIT STRING SIZE (3),
        subcmd      BIT STRING SIZE (1)
    },
    com_time       Unsigned8,
    profile_type    Unsigned8,
    reserve        OCTET STRING SIZE (0|8|24|40|56)
}

```

- com_mod field

com_mod

This field shall contain several modes about communication in the connection.

syncmode

This bit field shall indicate the communication mode.

- 0: asynchronous communication state;
- 1: synchronous communication state.

dtmode

This bit field shall indicate the transfer mode

This field shall be with the following value range.

- '00'H: single transfer mode;
- '01'H: dual transfer mode;
- '02'H to '03'H: reserved.

subcmd

This bit field shall indicate whether to use subcommand or not.

- 0: Subcommand field disabled;
- 1: Subcommand field enabled.

- com_time field

com_time

This field shall contain the communication cycle period in the form of a multiple of the transmission cycle period.

The value range shall be from 0 to 255.

- profile_type field

profile_type

This field shall contain the device profile code to be used. As for the device profile, see 4.5 and Annex A.

The value range shall be from 0 to 255.

4.4.1.10 DISCONNECT command and response

Exchange this command and response, the master and the slave shall release a FDC AR connection and specified communication options shall be reset. They shall make their transition of each PM; FDCPM-M and FDCPM-S, see 8.4.4 and 8.4.5 for more details.

```
_DISCONNECT-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (disconnect)
    } )
_DISCONNECT-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (disconnect)
    } )
```

4.4.1.11 PPRM_RD command and response

PRM_RD symbolizes a "PROM-parameter-read" command. Receiving this command, the slave shall read a specified parameter on E2PROM or non-volatile memory into PPRM_RD-RSP3-PDU.parameter field and shall respond with it.

```
_PPRM_RD-CMD3-PDU ::= CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (pprm_rd),
      pduBody (cmdBody _PPRM_RD-CMD3Body)
    } )
_PPRM_RD_CMD3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    reserve  OCTET STRING SIZE (1|9|25|41|57)
}
_PPRM_RD-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (pprm_rd),
      pduBody (rspBody _PPRM_RD-RSP3Body)
    } )
_PPRM_RD-RSP3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    reserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}
```

- pNo field

pNo

This field shall contain the parameter number to read out.

The value range shall be from 0 to 65 535.

- pSize field

pSize

This field shall contain an octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter

This field shall contain the data corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.4.1.12 PPRM_WR command and response

PPRM_WR symbolizes a "PROM-parameter-write" command. Receiving this command, the slave shall write a specified parameter value from PPRM_WR-CMD3-PDU.parameter field to E2PROM or non-volatile memory and shall respond with the echo of it to tell completion of the command.

```

_PPRM_WR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (pprm_wr),
      pduBody (cmdBody _PPRM_WR-CMD3Body)
    } )
_PPRM_WR_CMD3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize     Unsigned8,
    reserve   OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}
_PPRM_WR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (pprm_wr),
      pduBody (rspBody _PPRM_WR-RSP3Body)
    } )
_PPRM_WR-RSP3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize     Unsigned8,
    reserve   OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}

```

- pNo field

pNo

This field shall contain the parameter number to write in.

The value range shall be from 0 to 65 535.

- pSize field

pSize

This field shall contain the octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter

This field shall contain the data to write corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.4.1.13 MEM_RD command and response

MEM_RD symbolizes a "memory-read" command. Receiving this command, the slave shall read a chunk of specified virtual memory into MEM_RD-RSP3-PDU.data field and shall respond with it.

As for virtual memory, see Annex A.

```

_MEM_RD-CMD3-PDU    ::= _CMD3-PDU ( WITH COMPONENTS
                        { ..., cmd (mem_rd),
                          pduBody (cmdBody  _MEM_RD-CMD3Body)
                        } )
_MEM_RD-CMD3Body    ::= SEQUENCE {
                        reserve  OCTET STRING,
                        mMode    SEQUENCE {
                                data_type  BIT STRING SIZE (4),
                                mode       BIT STRING SIZE (4)
                        },
                        mSize     Unsigned16,
                        mAddress   Unsigned32,
                        reserve    OCTET STRING SIZE (4|20|36|52)
                        }
_MEM_RD-RSP3-PDU    ::= _RSP3-PDU ( WITH COMPONENTS
                        { ..., rcmd (mem_rd),
                          pduBody (rspBody  _MEM_RD-RSP3Body)
                        } )
_MEM_RD-RSP3Body    ::= SEQUENCE {
                        reserve  OCTET STRING,
                        mMode    SEQUENCE {
                                data_type  BIT STRING SIZE (4),
                                mode       BIT STRING SIZE (4)
                        },
                        mSize     Unsigned16,
                        mAddress   Unsigned32,
                        data      OCTET STRING SIZE (4|20|36|52)
                        }

```

- mMode field

mMode

This field shall contain the mode for the memory reading-out.

data_type

This bit field contains a type of the memory as source to read out.

- '00'H: reserved;
- '01'H: Volatile memory;
- '02'H: Non-volatile memory;
- '03'H to 'FF'H: reserved.

mode

This bit field contains the data type of the specified data on memory to read out.

- '00'H: reserved;
- '01'H: Integer8;
- '02'H: Integer16;
- '03'H: Integer32;
- '04'H: Integer64;
- '05'H to 'FF'H: reserved.

- mSize field

mSize

This field shall contain the number of data on memory to read out.

The value range shall be from 0 to 20.

- mAddress field

mAddress

This field shall contain the start address of memory to read out.

The value range shall be from 0 to 'FFFF FFFF'H.

- data field

data

This field shall contain the read data with the specified mMode, mSize and mAddress.

4.4.1.14 MEM_WR command and response

MEM_WR symbolizes a "memory-write" command. Receiving this command, the slave shall write a chunk of specified virtual memory value from MEM_WR-CMD3-PDU.data field to the memory and shall respond with the echo of it to tell completion of the command.

As for virtual memory, see Annex A.

```

_MEM_WR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (mem_wr),
      pduBody (cmdBody _MEM_RD-CMD3Body)
    } )

_MEM_WR-CMD3Body ::= SEQUENCE {
    reserve    OCTET STRING,
    mMode      SEQUENCE {
        data_type  BIT STRING SIZE (4),
        mode       BIT STRING SIZE (4)
    },
    mSize      Unsigned16,
    mAddress    Unsigned32,
    data       OCTET STRING SIZE (4|20|36|52)
}

_MEM_WR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (mem_wr),
      pduBody (rspBody _MEM_RD-RSP3Body)
    } )

_MEM_WR-RSP3Body ::= SEQUENCE {
    reserve    OCTET STRING,
    mMode      SEQUENCE {
        data_type  BIT STRING SIZE (4),
        mode       BIT STRING SIZE (4)
    },
    mSize      Unsigned16,
    mAddress    Unsigned32,
    data       OCTET STRING SIZE (4|20|36|52)
}

```

- mMode field

mMode

This field shall contain the mode for the memory writing-in.

data_type

This bit field shall contain a type of the memory as destination to write in.

- '00'H: reserved;
- '01'H: Volatile memory;
- '02'H: Non-volatile memory;
- '03'H to 'FF'H: reserved.

mode

This bit field shall contain the data type of the specified data on memory to write in.

- '00'H: reserved;
- '01'H: Integer8;
- '02'H: Integer16;
- '03'H: Integer32;
- '04'H: Integer64;
- '05'H to 'FF'H: reserved.

- mSize field

mSize

This field shall contain the number of data on memory to write in.

The value range shall be from 0 to 20.

- mAddress field

mAddress

This field shall contain the start address of memory to write in.

The value range shall be from 0 to 'FFFF FFFF'H.

- data field

data

This field shall contain the data to write with the specified mMode, mSize and mAddress.

4.5 Device profile

A field device can provide some sets of field-device-dependent PDUs and their CMDCodes for FDC service, called field device profiles. The user of an FDC master can decide which of them he should choose for FDC communication when the connection of FDC protocol is established.

The device Profile system and related information are shown in Annex A.

5 Transfer syntax**5.1 Concepts**

The Type 27 FAL is a technology for a time critical application. Therefore, it is important to make the size of encoded data compact, and to enable to encode and decode in a simple manner.

Consequently, the encoding and decoding processes shall be defined as such specific to the Type 27 FAL, and require neither any commonality with the other protocol type nor versatility. The data format and the encode rule of APDU shall be pre-defined for entities that transfer the data, and require no presentation layer service.

In the encoding rule of the Type 27 FAL, the tag (data type code) and the data length need not be encoded and the only values of each data types defined with ASN.1 abstract syntax shall be encoded into a data row.

The length of PDU shall be fixed in case of a FDC service where the time factor is particularly strict. For a MSG service where the time factor is not so strict, PDU with variable length can be transferred. In this case, a data field that specifies the data length should be defined in the abstract syntax explicitly.

The order of the bits flow on the transmission path should be defined by the lower layer.

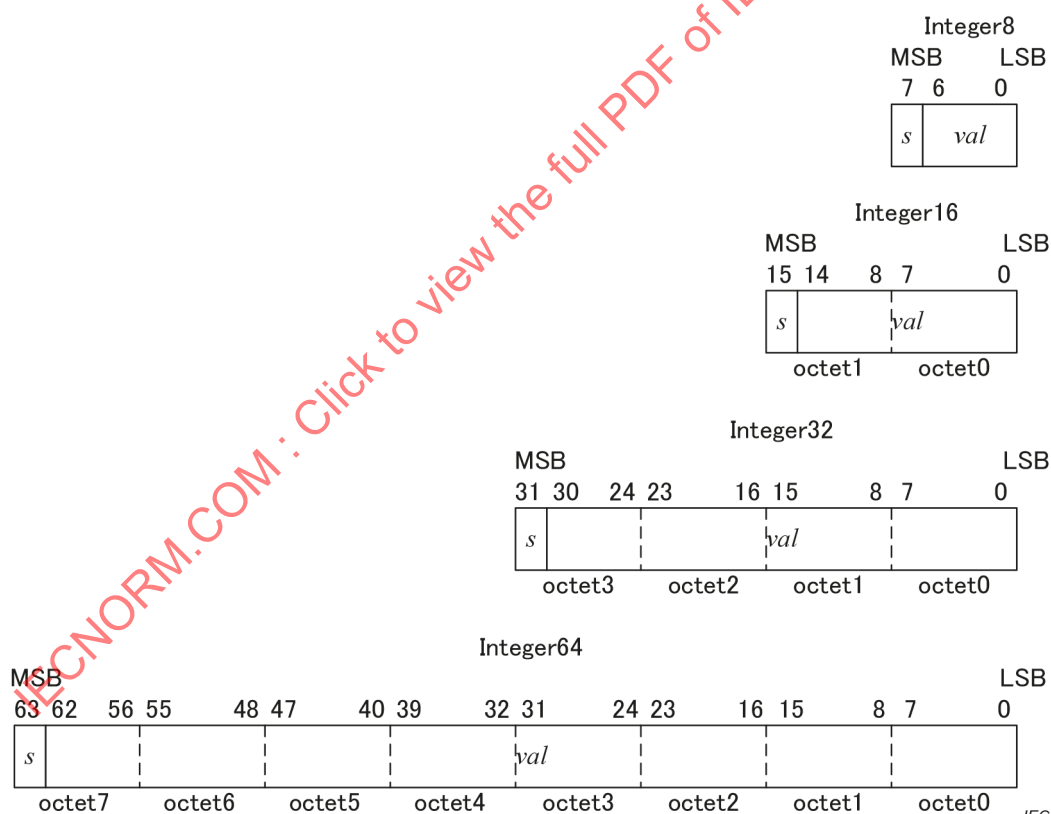
5.2 Encode rules

5.2.1 INTEGER and its subtypes

INTEGER and its derivatives; Integer8, Integer16, Integer32, Integer64, Unsigned8, Unsigned16, Unsigned32 and Unsigned64; shall encode and transfer only the data value of their octet size.

The encoded form of Integer8, Integer16, Integer32 and Integer64 shall consist of a sign bit: *s* on the most significant bit (see Figure 1). The *val* shall contain the value itself when it is 0 or a positive number (the sign bit is '0'B). When *val* is a negative number (the sign bit is '1'B), the encoded data shall be represented as a complement of 2 by using *s* and *val*.

NOTE This encode rule is applied to the data structure when PDU is transferred, but need not specifies the data structure on an actual memory in the data processing system in an actual device.



Key

s sign bit (0: + / 1: -)

val positive value, if *s*=0;

"*s+val*" represents a negative value as a complement of 2, if *s*=1.

Figure 1 – Encode of Integer subtypes

When transferring the data of a multi-octets' data type, the octets shall be transmitted in order from the lower octet to the higher octet (see the example in Figure 2).

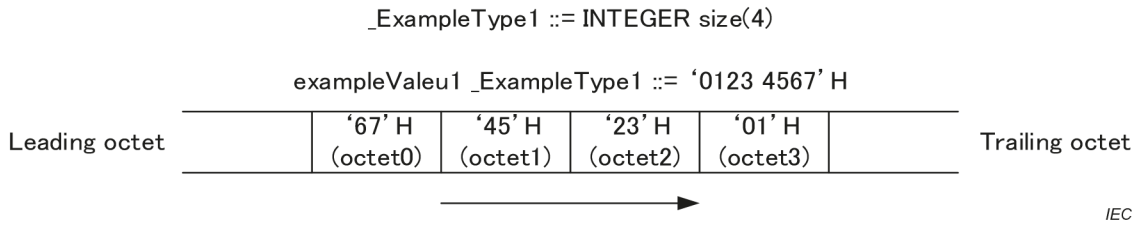


Figure 2 – Example of transfer of INTEGER value

In the encoded form of Unsigned8, Unsigned16, Unsigned32 and Unsigned64, the numeric value data (*val*) shall be set in the data area of its octet size as shown in Figure 3. In this case, the numeric value shall be set in order that the number in a lower digit is put into a lower bit in sequence, and 0 value shall be set to the unused upper bit.

When transferring data of a multi-octets' data type, the octets shall be transmitted in order from the lower octet to the higher octet.

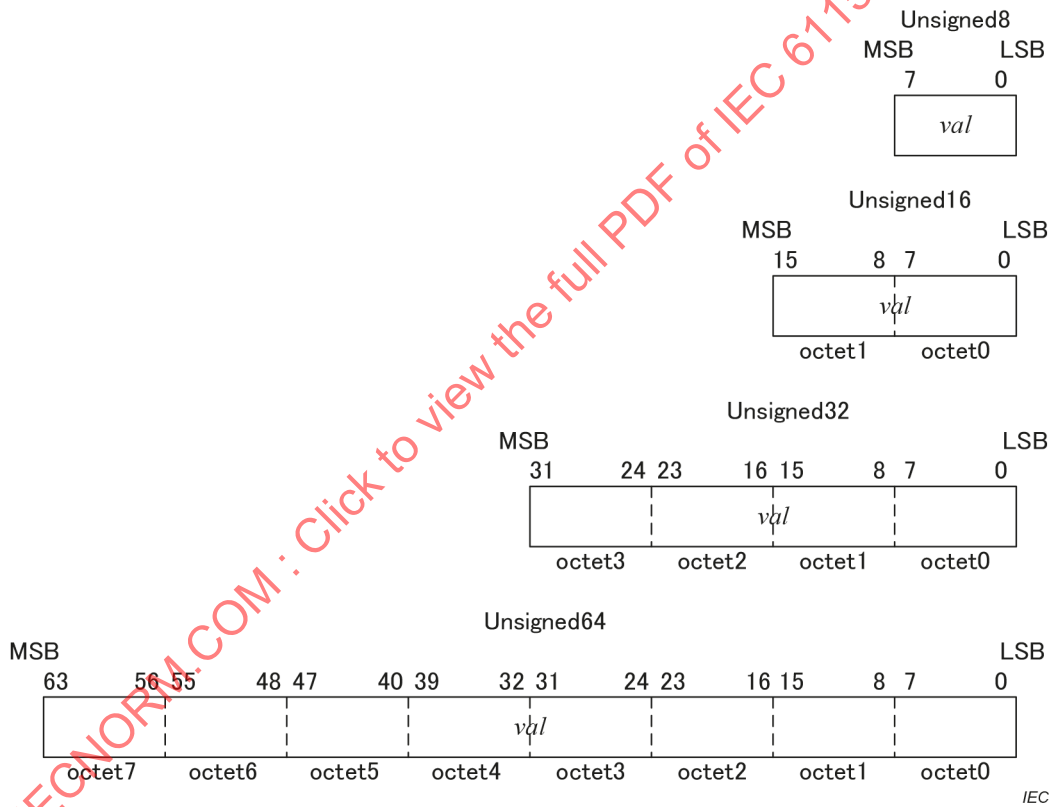
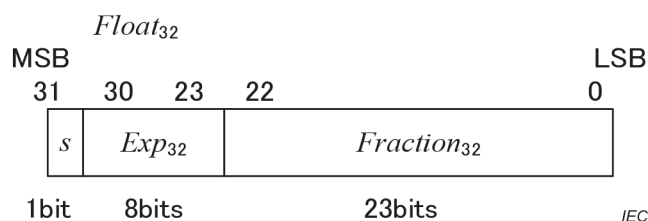


Figure 3 – Encode of Unsigned subtypes

5.2.2 REAL type and its subtypes

The Type 27 FAL does not directly define the data of REAL type within the abstract syntax, and uses *Float₃₂* and *Float₆₄* that are derivatives of REAL type. This encoding shall comply with IEC 60559. The encoded formats of them are shown in Figure 4 and Figure 5.

**Figure 4 – Float32 type encode**

Data of *Float₃₂* type shall be 4 octets' data as shown in Figure 4. The octets shall be transmitted in order from the lower octet to the higher octet. In this case, the value of the floating-point data of *Float₃₂* type shall be calculated by using the following formula:

$$Float_{32} = (-1)^s \times c_{32} \times 2^{q_{32}},$$

where

s: sign bit (0:+ / 1:-),

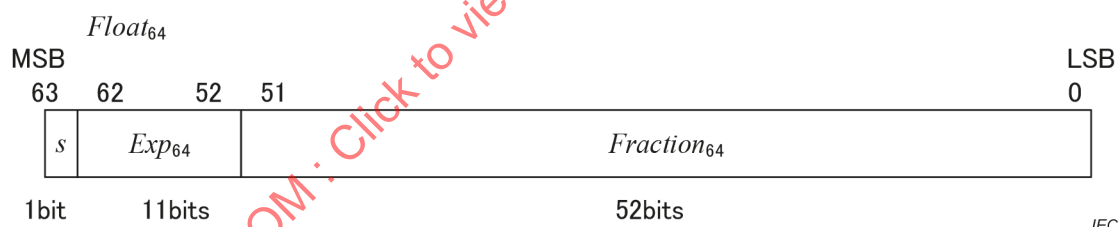
$$c_{32} = (1 \times 2^{23} + Fraction_{32}) \times 2^{-23} =$$

$$1 + \frac{b_{22}}{2} + \frac{b_{21}}{2^2} + \dots + \frac{b_0}{2^{23}}: \text{ mantissa,}$$

$$q_{32} = Exp_{32} - 127: \text{ exponent } (-127..127).$$

If both *Fraction₃₂* and *Exp₃₂* equal 0, then *Float₃₂* represents 0.

If *Fraction₃₂* equals 0 and *Exp₃₂* equals 255, then *Float₃₂* represents positive or negative infinity as the sign bit *s*.

**Figure 5 – Float64 type encode**

Data of *Float₆₄* type shall be 8 octets' data as shown in Figure 5. The octets shall be transmitted in order from the lower octet to the higher octet. In this case, the value of the floating-point data of *Float₆₄* type shall be calculated by using the following formula:

$$Float_{64} = (-1)^s \times c_{64} \times 2^{q_{64}},$$

where

s: sign bit (0:+ / 1:-),

$$c_{64} = (1 \times 2^{52} + Fraction_{64}) \times 2^{-52} =$$

$$= 1 + \frac{b_{51}}{2} + \frac{b_{50}}{2^2} + \dots + \frac{b_0}{2^{52}}: \text{ mantissa,}$$

$$q_{64} = Exp_{64} - 1023: \text{ exponent } (-1023..1023).$$

If both *Fraction₆₄* and *Exp₆₄* equal 0, then *Float₆₄* represents 0.

If $Fraction_{64}$ equals 0 and Exp_{64} equals 2 047, then $Float_{64}$ represents positive or negative infinity as the sign bit s .

5.2.3 BIT STRING type

In BIT STRING type, the leading bit (0th bit) shall be corresponding to the LSB of the encoded data and the trailing bit shall be corresponding to the MSB according to the bit numbers attached to the named bits. However, the padding data shall be fulfilled by the octet boundary. In this case, an area can be reserved even for undefined bit; however, the value of the bit is not defined. Figure 6 shows an example of the bit field definition in the BIT STRING type.

For a BIT STRING data block of multi-octets' data size, the octets shall be transmitted in order from the octet with a lower bit.

NOTE The definition of "leading bit" and "trailing bit" is according to ISO/IEC 8824-1:2021, 22.2.

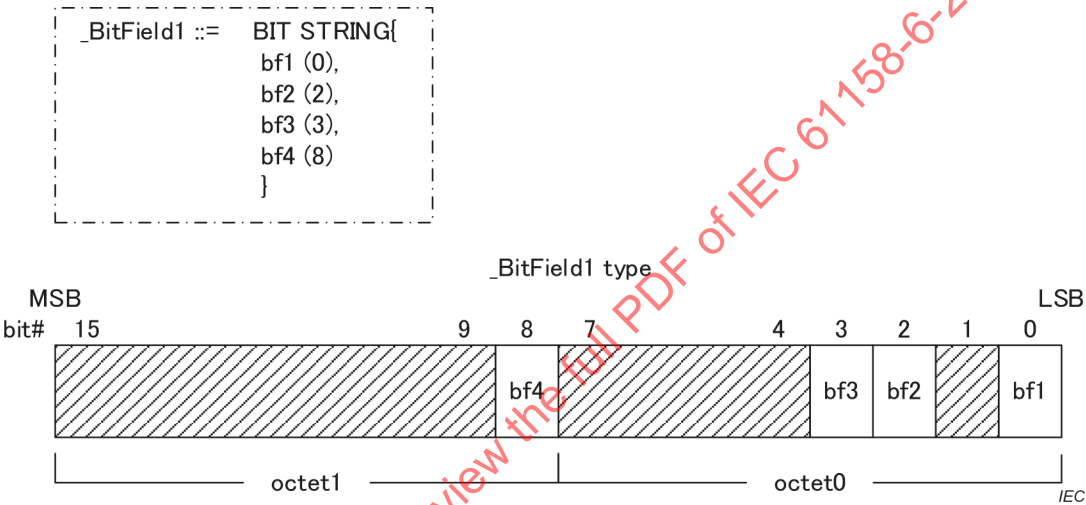


Figure 6 – Bit field definition example with named bits

A bit field is defined by combining SEQUENCE type and BIT STRING type. In this case, the boundary of each field shall be defined according to the specified bit size and fill from the lower bit (see the example shown in Figure 7).

To encode this bit field, an octet shall be filled from its lower bit according to the order defined by the field definition described between "{" and "}" in the SEQUENCE syntax to define a data area that is confined by the octet boundary. When undefined fraction bits remain, they shall be a reserve area and their values shall be undefined.

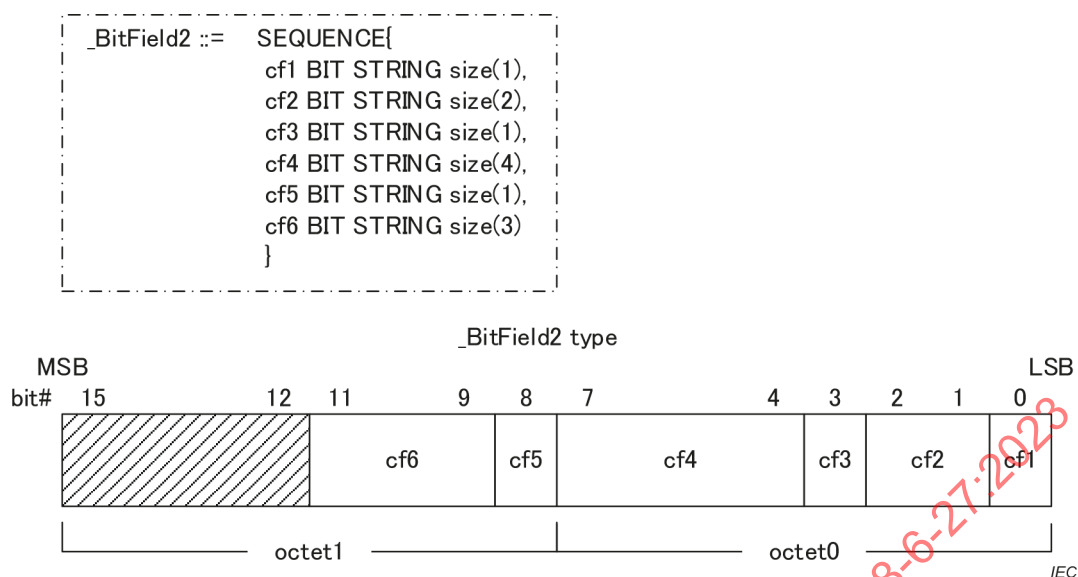


Figure 7 – Bit field definition example with field size

5.2.4 OCTET STRING type and IA5String type

Data of OCTET STRING type shall be transferred without converting the code in order from the most significant octet of the given string data.

Data of IA5String type shall be encoded into a data row (OCTET STRING) by converting the given String data into a 1-octet code, according to ISO/IEC 646, one character by one character, and then add one octet of null code ('00'H) to the last character code. The data shall be transferred in order from the first character code data.

5.2.5 NULL type

For data of NULL type, the data length is 0 and no value exists. Therefore, it shall not be encoded nor transferred.

5.2.6 Structure type and Array type

Among the elements of the Structure type, the SEQUENCE type and SEQUENCE OF type themselves shall not be encoding. Components included in these types shall be encoded and transferred according to each rule and the order of octet transfer, as shown in Figure 8.

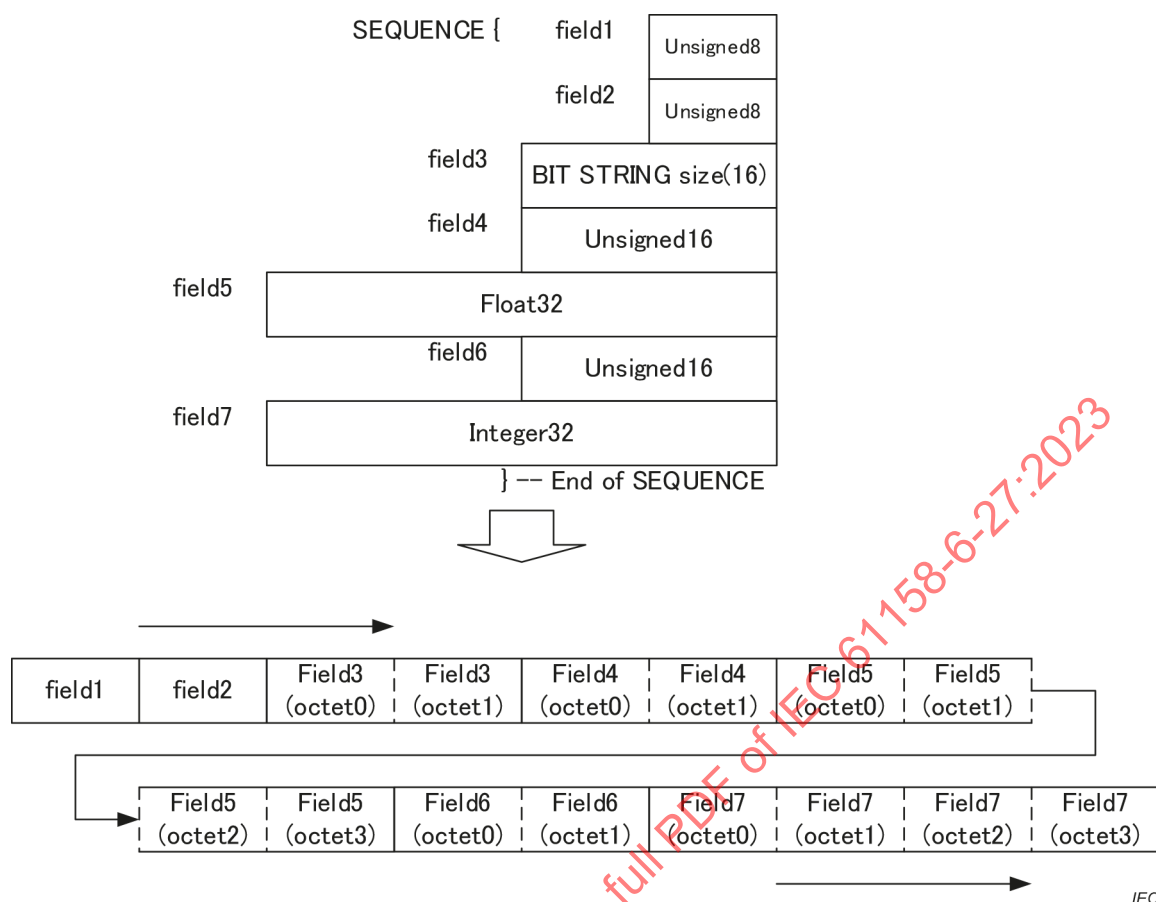


Figure 8 – SEQUENCE type encode

Also, the CHOICE type itself shall not be encoding. To edit communication data in any one of multiple alternatives of this type, the data should be selected according to the context of the communication AP and the agreement between the communication devices should be established. The method for this process depends on implementation and out of range of this document. The data shall be encoded according to the rule of the selected component.

6 Structure of FAL protocol state machine

In this clause and subsequent ones, FAL ASEs are characterized with protocol state machine (PM) models.

Although this type of fieldbus has its own structure of PMs, a mapping table is shown below for a clear understanding (see Table 10). It maps the structure for this type onto the typical four-sublayered one adopted by most of other types, which contains AP-Context PM, FSPM, ARPM, and DMPM.

Figure 9 shows the structure of PMs.

- There is a formally defined AP-Context State Machine (APC SM) for FAL user's initiation of a specific communication application.
- There is no formal definition of FSPM Machine just serving as an interface between FAL User and ARPM. Instead of that, two types of protocol machines are defined:
 - a set of Field Device Control machines (FDC PM) as master, slave, and monitor for FDC service users;
 - a pair of Message machines (MSG PM) as requester and responder for Message service users.

- Two different types of ARPM Machines are defined at the interface to FSMUL:
 - a set of ARPM machines for connection-oriented application relationships between classes of Master, Slave or Monitor of FDC Service;
 - a kind of ARPM machine for connection-less application relationships between classes of Requester and Responder of Messages Service.
- There is no formal definition of the DLL Mapping Protocol Machine (DMPM), unified into ARPM or APC SM instead. And DL services are used directly by ARPM or APC SM.

It is assumed that FDCPM functions in FDC ASE; and also MSGPM in MSG ASE; ARPM in AR ASE; and AP-CONTEXT SM in FSM ASE.

Table 10 – Mapping for Protocol State Machines

AP Type	ASE	Activated Class	Protocol Machine	Mapping to a typical structure for most of other types
C1 Master AP	FSM ASE	FieldbusSystemManager	APC SM	AP-Context-PM
	FDC ASE	Master	FDC PM-M	FSPM
	MSG ASE	Requester	MSG PM-RQ	FSPM
		Responder	MSG PM-RS	
	AR ASE	FDCMaster-AR	ARPM-FDCM	ARPM with DMPM
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a
Slave AP	FSM ASE	FieldbusSystemManager	APC SM	AP-Context-PM
	FDC ASE	Slave	FDC PM-S	FSPM
		Monitor (option)	FDC PM-MN	
	MSG ASE	Responder	MSG PM-RS	FSPM
	AR ASE	FDCSlave-AR	ARPM-FDCS	ARPM with DMPM
		FDCMonitor-AR	ARPM-FDCMN	
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a

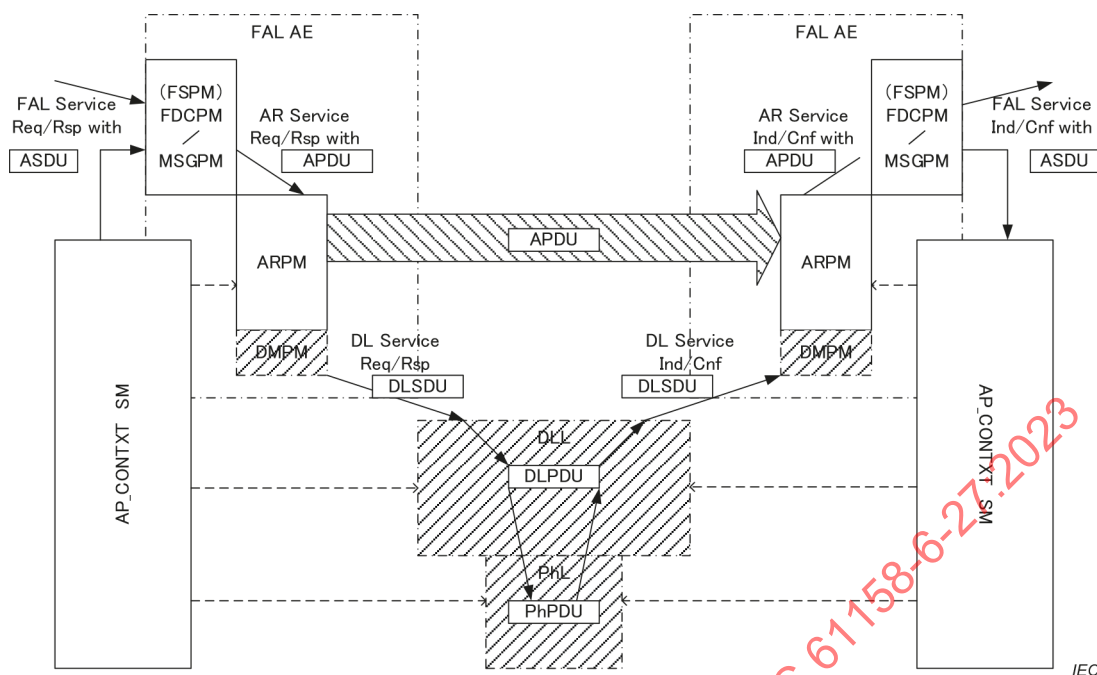


Figure 9 – Structure of FAL protocol state machines

7 AP-context state machine (APC SM)

7.1 Overview

An AP-context is a set of information and rules related to a communication system. It shall be created by instructions from an FAL user while the application process (AP) is invoked. For example, it contains selected communication parameters, based on the system configuration, an application field of the system, the own device profile, and difference of communication abilities between own device and peer ones.

In this ASE model, an FAL should retain the information mentioned above within attributes of the FieldbusSystemManager class according to user's instructions or configurations. The information should be able to characterize the AP types such as C1 Master and Slave. In addition, it should be able to configure other communication parameters and the device profile; such as a servo drive, an inverter drive or an I/O device, in order to establish the communication environment.

The APC SM shall control a series of state transitions, in which a device boots up, the APC SM shall get the AP-context data from the FAL user, and it shall initialize the FAL and the lower layer with the context. In consequence, it shall enable steady communication. The APC SM realizes the service that the FSM ASE provides.

The APC SM need not directly edit an APDU, nor realize a protocol to transfer PDUs. However, it indirectly acts as a trigger of PDU-exchange by using the other ASE service and the lower layer service during the FAL initialization process.

Figure 10 shows the APC SM state chart diagram.

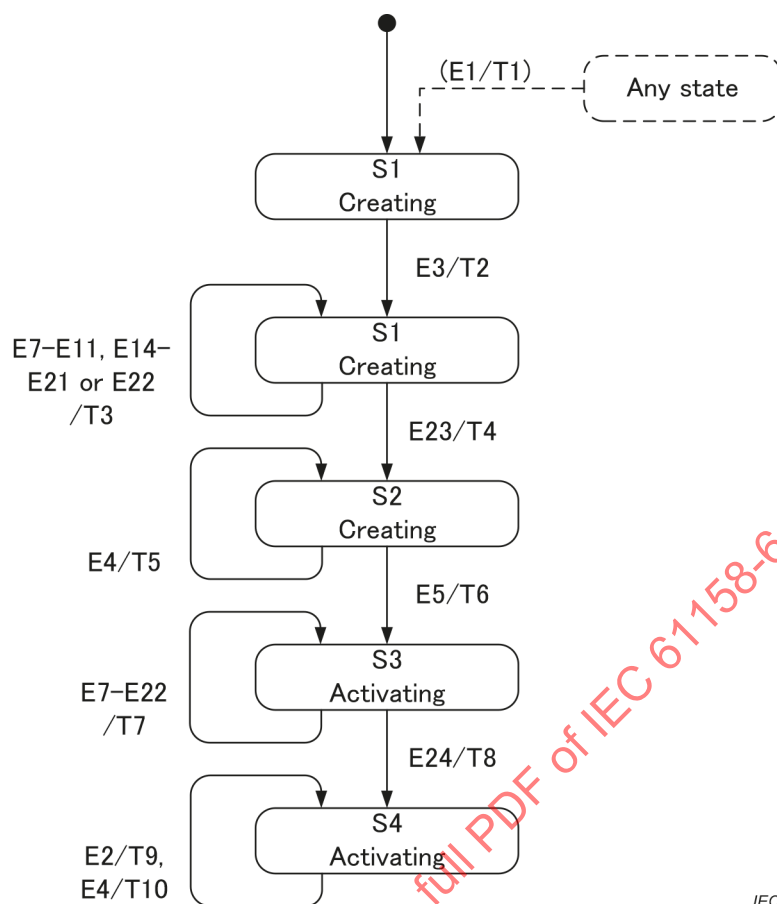


Figure 10 – State chart diagram of APC SM

7.2 State descriptions

Table 11 describes each state of the APC SM.

Table 11 – State descriptions of APC SM

S#	State	Substate	Description
S0	Cleared	-	State when the FieldbusSystemManager Class of FSM ASE has just been instantiated as this SM All the entities in each layer and ASEs in the device are reset to the initial state, and AP-context is cleared.
S1	Creating		Transient state where AP-context or communication environment is generated, based on the input CONTEXT-DATA and the result of negotiation between the corresponding devices
S2	Created	-	State when AP-context has been generated
S3	Activating	-	Transient state when entities in each layer and ASEs are sequentially enabled
S4	Active	-	Steady state when all entities are activated and normal communication services are provided

7.3 Triggering events

Table 12 lists each trigger event of the APC SM.

Table 12 – Trigger event descriptions of APC SM

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	FSM-Reset.req	FAL user		
E2	FSM-GetStatus.req	FAL user	INFO-ID	
E3	FSM-SetContext.req	FAL user	CONTEXT-DATA	
E4	FSM-GetContext.req	FAL user		
E5	FSM-Start.req	FAL user		
E6	<s>-Open.cnf	<s> ASE	ServiceStatus	
E7	FSMUL-RESET.cnf	FSMUL		
E8	FSMUL-SET-VALUE.cnf	FSMUL	Result	
E9	FSMUL-GET-VALUE.cnf	FSMUL	Result, Val	
E10	FSMUL-DETECT.cnf	FSMUL		
E11	FSMUL-RECOG.cnf	FSMUL		
E12	FSMUL-SW-TRANS-MODE.cnf	FSMUL		
E13	FSMUL-MEAS-DELAY.ind	FSMUL	Delay_Time	
E14	FSMUL-MEAS-DELAY.cnf	FSMUL	Delay_Time	
E15	FSMUL-CONFIG.cnf			
E16	FSMUL-START-SYNC.ind			
E17	FSMUL-START-SYNC.cnf			
E18	FSMUL-START.ind	FSMUL	Com_Mode, Cycle_time, Max_Delay, TM_unit	
E19	FSMUL-START.cnf	FSMUL	Result	
E20	FSMUL-CLR-ERR.cnf	FSMUL	Result	
E21	Ph-SET-VALUE.cnf	PhL		
E22	Ph-GET-VALUE.cnf	PhL	Value	
E23	APC-Created	APC SM (S1 state)		Internal event
E24	APC-Activated	APC SM (S3 state)		Internal event

7.4 Action descriptions at state transitions

Detailed specifications such as initialization processing, activate processing, and various communication parameters in each layer and each ASE are outside the scope of this document because they depend on the implementation. Just outline of the related action is shown in Table 13.

Table 13 – Transitions of APC SM

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1:FSM-Reset.req / Ph-RESET.req; FSMUL_RESET.req; EVM-Reset.req; for all AR ASE objects(AR-Reset.req); for all FDC ASE objects(FDC_Reset.req); for all MSG ASE objects(MSG-Reset.req);	S0:Cleared
T2	S0:Cleared	E3: FSM-SetContext.req(CONTEXT-DATA) / /*-- APC SM starts initializing PhL -- with Ph-SET-VALUE.req --*/; /*-- APC SM starts initializing FSMUL -- with FSMUL_SET_PAR.req -- and FSMUL_DELAY.req --*/; for all AR ASE objects(AR-Open.req); for all FDC ASE objects(FDC-Open.req); for all MSG ASE objects(MSG-Open.re));	S1:Creating
T3	S1:Creating	E7, E8, E9, E10, E11, E12, E13, E14, E15, E18, E19, E20, E21, or E22 / /*-- APC SM keeps initializing PhL, FSMUL, and FAL --*/; /*-- If initializing procedures have finished, -- E28: APC-Created is issued. --*/;	S1:Creating
T4	S1:Creating	E23:APC-Created / FSM-SetContext.cnf;	S2:Created
T5	S2:Created	E4:FSM-GetContext.req / FSM-GetContext.cnf;	S2:Created
T6	S2:Created	E5:FSM-Start.req / /*-- start activating PhL --*/; /*-- start activating FSMUL --*/; EVM-Enable.req; for all AR ASE objects(AR-Enable.req); for all FDC ASE objects(FDC-Enable.req); for all MSG ASE objects(MSG-Enable.req);	S3:Activating
T7	S3:Activating	E7, E8, E9, E10, E11, E12, E13, E14, E15, E16, E17, E18, E19, E20, E21, or E22 / /*-- APC SM keeps activating PhL, FSMUL, FAL --*/; /*-- If activating procedures have finished, -- E29:APC-Activated is issued --*/;	S3:Activating

T#	Source State	Event (arguments) [conditions] / Action	Target State
T8	S3:Activating	E24: APC-Activated / FSM-Start.cnf;	S4:Active
T9	S4:Active	E2: FSM-GetStatus.req(INFO-ID) / /*-- APC SM reads appropriate status info for INFO-ID, -- using Ph-GET_VALUE.req, -- CTC-GET-STATUS.req, -- DLM_GET_ERR.req -- or other service primitives --*/; FSM-GetStatus.cnf;	S4:Active
T10	S4:Active	FSM-GetContext.req / FSM-GetContext.cnf;	S4:Active

8 FAL service protocol machines (FSPM)

8.1 Overview

When the FSPM receives a service request primitive or a response primitive from an FAL user, then it shall edit an APDU from an SDU as the parameter of the primitives, and then request the ARPM to transmit. FSPM shall also take out an SDU from the APDU received by the ARPM and then deliver it to FAL user as an indication primitive or conform.

Two types of application services (FDC ASE and MSG ASE) can be provided to the FAL user in the Type 27 FAL (refer to IEC 61158-5-27), and the FSPM consists of two types of PM (FDC PM and MSG PM) corresponding to the ASEs.

8.2 RT Protocol Machine (RT PM)

8.2.1 Link layer discovery

The concepts according to IEEE Std 802.1AB shall be applied. This part of the specification defines the utilization of the IEEE Std 802.1AB.

8.2.2 MAC bridges

8.2.2.1 Overview

The concepts according to IEEE Std 802.1D shall be applied. This part of the specification defines the utilization of the IEEE Std 802.1D. It includes extensions for RT phase regarding forwarding of frames. The routing mechanisms of IEEE Std 802.1D shall be temporary disabled within the RT phase. In this case, the routing of RT frames shall be done according to the attributes defined by the appropriate ASE. The values of the attributes define a special routing table, which is used.

Apart from IEEE Std 802.1D behavior, this protocol specification defines the following protocol machines to provide special forwarding actions:

- RT Forwarding Protocol Machine (RT_FW);
- Precision Time Protocol Forwarding Protocol Machine (PTP_FW).

8.2.2.2 RT Forwarding Protocol Machine (RT_FW)

8.2.2.2.1 General

RT_FW consists of CTC (Communication Transmission Control), SRC (Send Receive Control), and FSMUL (Field System Management Underlying Layer) as the three underlying layers. SRC is located in the lower of CTC, while FSMUL is located across both CTC and SRC.

CTC creates FIDPDU and it is the sublayer to execute the sync protocol machine and the event driven communication protocol machine. These protocol machines are operated according to the command from FSM or FSMUL.

SRC is a sublayer to send or receive a frame according to the command from CTC. FSMUL is a sublayer to set the internal variants to be used for RT_FW and control errors detected in each sublayer.

8.2.2.2.2 Communication Transmission Control Underlying Layer

8.2.2.2.2.1 CTC Protocol Machine of the master

The master CTC state transition chart is as shown in Figure 11 and the master CTC state transition table is as shown in Table 14.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-27:2023

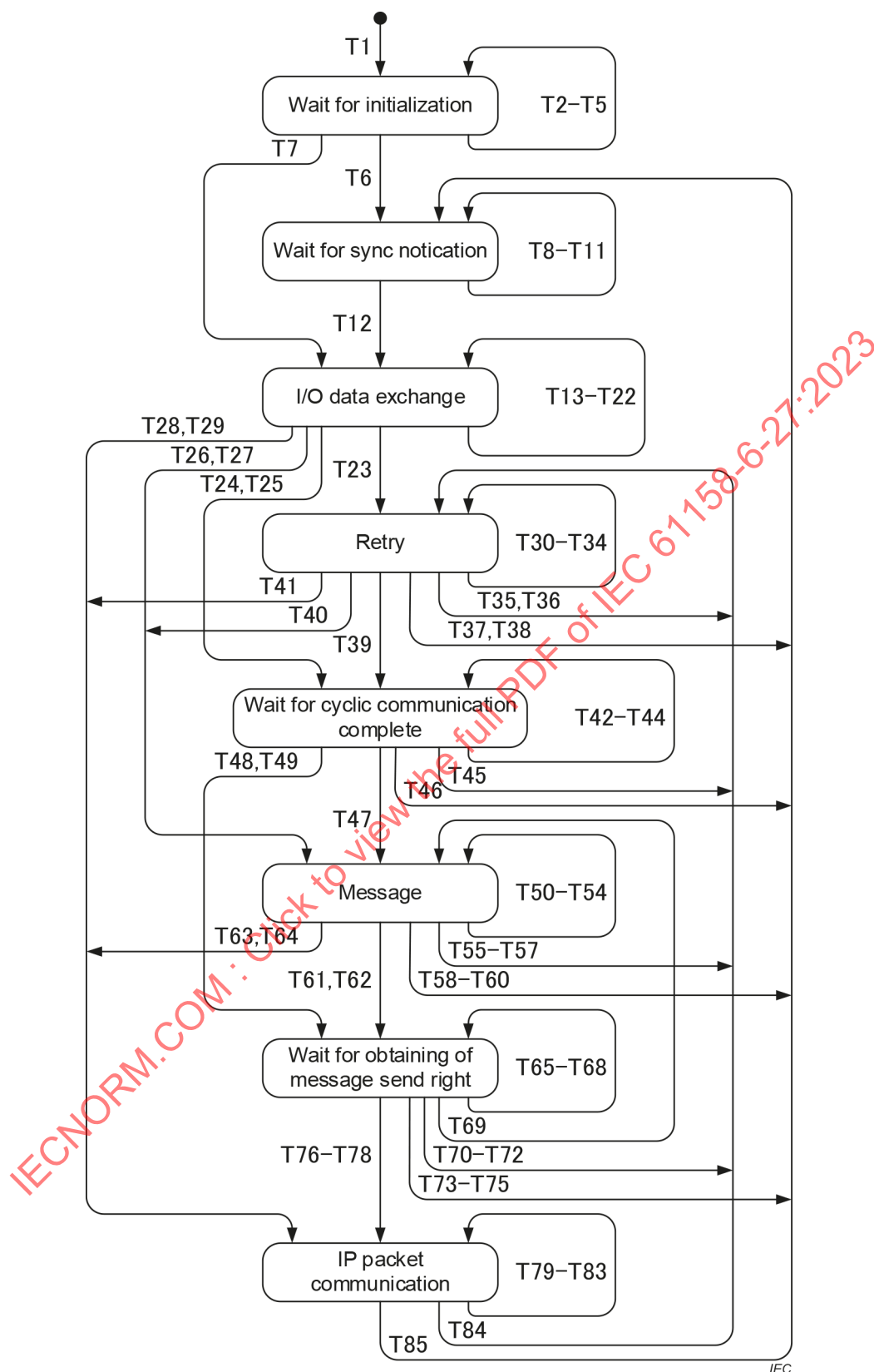


Figure 11 – Master CTC state transition chart

Table 14 – Master CTC state transition table

T#	Source State	Event (arguments) [conditions] / Action	Target State
1	All	Power ON or CTC_Reset.req / none	Wait for initialization
2	Wait for initialization	CTC_Set_Par.req(Par_ID, Val) / CTC_SET_PAR(Par_ID, Val, Result); CTC_Set_Par.cnf(Result);	Wait for initialization
3	Wait for initialization	CTC_Get_Par.req(Par_ID) / CTC_GET_PAR(Par_ID, Val, Result); CTC_Get_Par.cnf(Result, Par_ID, Val);	Wait for initialization
4	Wait for initialization	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Wait for initialization
5	Wait for initialization	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Wait for initialization
6	Wait for initialization	CTC_Start.req [CHECK_SYNC_RIGHT() = "SLV"] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Wait for sync notification
7	Wait for initialization	CTC_Start.req [CHECK_SYNC_RIGHT() = "OWN"] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	I/O data exchange
8	Wait for sync notification	EXPIRED_TIMER(T(Tcycle)) = "True" / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Wait for sync notification
9	Wait for sync notification	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if(Result == FCMD_SYNC) { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tsync)); RESTART_TIMER(T(Tipst)); }	Wait for sync notification
10	Wait for sync notification	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Wait for sync notification
11	Wait for sync notification	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU) CTC-READ-DATA.cnf(Result, SDU)	Wait for sync notification
12	Wait for sync notification	EXPIRED_TIMER(T(Tsync)) = "True" / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI"); { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Input data forward

T#	Source State	Event (arguments) [conditions] / Action	Target State
13	I/O data exchange	EXPIRED_TIMER(T(Tcycle)) = "True" / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	I/O data exchange
14	I/O data exchange	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "LO") && (T(Trspst) < V(Trspst)) && (Rcv_sts = "OK")] / EXEC_IP_SEG(Port_num, RcvIPDLSDU, None);	I/O data exchange
15	I/O data exchange	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "LO") && (T(Trspst) >= V(Trspst)) && (Rcv_sts = "OK")] / Result = CHECK_PDU_CYCR(DLSDU, V(Nslave)); if(Result == "OK") { V(Nrcv) = V(Nrcv) + 1; PUT_RCVCHK_LIST(V(Nslave)); } STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	I/O data exchange
16	I/O data receive	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "LO") && (T(Trspst) >= V(Trspst)) && (Rcv_sts <> "OK")] / CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	I/O data exchange
17	I/O data exchange	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "HI") && (Rcv_sts = "OK")] / Result = CHECK_PDU_CYCR(DLSDU, V(Nslave)); if(Result == "OK") { V(Nrcv) = V(Nrcv) + 1; PUT_RCVCHK_LIST(V(Nslave)); } STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	I/O data exchange
18	I/O data receive	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "HI") && (Rcv_sts <> "OK")] / CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	I/O data exchange
19	I/O data exchange	EXPIRED_TIMER(T(Trspst)) = "True" / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp));	I/O data exchange
20	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) = 0) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	I/O data exchange
21	I/O data exchange	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	I/O data exchange
22	I/O data exchange	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	I/O data exchange

T#	Source State	Event (arguments) [conditions] / Action	Target State
23	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) > 0) && (V(Nrcv_max) > V(Nrcv))] / for((V(Nslave) = GET_RCVCHK_LIST()) <> 0) { PUT_RETRY_LIST(V(Nslave)); PUT_RCVCHK_LIST(V(Nslave)); } V(Tretry) = CALC_RETIME(); DLSDU = BUILD_PDU_RETREQ(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tretry), V(Tretry)); V(Nretry) = V(Nmax_retry) – 1;	Retry
24	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) > 0) && (V(Nrcv_max) >= V(Nrcv)) && (V(Nsmst) > 0)] / none	Wait for cyclic communication complete
25	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [V(Nmax_retry) = 0 && V(Nsmst) > 0] / none	Wait for cyclic communication complete
26	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) > 0) && (V(Nrcv_max) >= V(Nrcv)) && (V(Nsmst) = 0) && (CHECK_MSG_EN() <> 0)] / none	Message
27	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) = 0) && (V(Nsmst) = 0) && (CHECK_MSG_EN() <> 0)] / none	Message
28	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [((V(Nmax_retry) > 0) && (V(Nrcv_max) >= V(Nrcv))) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0)] / none	IP packet communication
29	I/O data exchange	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) = 0) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0)] / none	IP packet communication
30	Retry	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_CYCR(DLSDU, V(Nslave)); if(Result == "OK") { RMV_RETRY_LIST(V(Nslave)); } STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	Retry
31	Retry	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts <> "OK"] / CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	Retry
32	Retry	EXPIRED_TIMER(T(Tretry)) = "True" [(V(Nretry) <> 0) && (GET_RETRY_LIST() <> 0) && (CMP_RTIM(T(Tcycle), CALC_RETIME()) > 0)] / V(Tretry) = CALC_RETIME(); DLSDU = BUILD_PDU_RETREQ(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tretry), V(Tretry)); V(Nretry) = V(Nretry) – 1;	Retry
33	Retry	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Retry
34	Retry	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Retry

T#	Source State	Event (arguments) [conditions] / Action	Target State
35	Retry	EXPIRED_TIMER(T(Tretry)) = "True" [(CHECK_SYNC_RIGHT() = "OWN") && ((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0))) && ((V(Nsmst) = 0) && (CHECK_MSG_EN() = 0)) && (CHECK_IP_EN() = 0)] / none	I/O data exchange
36	Retry	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() = "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() = "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	I/O data exchange
37	Retry	EXPIRED_TIMER(T(Tretry)) = "True" [(CHECK_SYNC_RIGHT() = "SLV") && ((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0))) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	Wait for sync notification
38	Retry	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Wait for sync notification
39	Retry	EXPIRED_TIMER(T(Tretry)) = "True" [((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0))) && (V(Nsmst) > 0)] / none	Wait for cyclic communication complete
40	Retry	EXPIRED_TIMER(T(Tretry)) = "True" [((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0))) && (V(Nsmst) = 0) && (CHECK_MSG_EN() <> 0)] / none	Message
41	Retry	EXPIRED_TIMER(T(Tretry)) = "True" [((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0))) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0)] / none	IP packet communication
42	Wait for cyclic communication complete	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / DESTROY_PDU_CYCR(DLSDU);	Wait for cyclic communication complete
43	Wait for cyclic communication complete	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Wait for cyclic communication complete
44	Wait for cyclic communication complete	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Wait for cyclic communication complete

T#	Source State	Event (arguments) [conditions] / Action	Target State
45	Wait for cyclic communication complete	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() = "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() = "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	I/O data exchange
46	Wait for cyclic communication complete	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Wait for sync notification
47	Wait for cyclic communication complete	EXPIRED_TIMER(T(Tmsgst)) = "True" [(CHECK_MSG_RIGHT() <> 0) && (CHECK_MSG_EN() <> 0)] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
48	Wait for cyclic communication complete	EXPIRED_TIMER(T(Tmsgst)) = "True" [(CHECK_MSG_RIGHT() <> 0) && (CHECK_MSG_EN() = 0)] / DLSDU = BUILD_PDU_TOKEN(V(Nnext_add), Msg); SRC_Send_Frame.req(Port_sel, DLSDU);	Wait for obtaining of message send right
49	Wait for cyclic communication complete	EXPIRED_TIMER(T(Tmsgst)) = "True" [CHECK_MSG_RIGHT() = 0] / none	Wait for obtaining of message send right
50	Message	EXPIRED_TIMER(T(Tmsgst)) = "True" [CHECK_MSG_RIGHT() <> 0] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
51	Message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / EXEC_MSGC_SEG(Result, DLSDU, None);	Message
52	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) > 0)] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
53	Message	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Message
54	Message	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Message

T#	Source State	Event (arguments) [conditions] / Action	Target State
55	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "OWN") && (CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	I/O data exchange
56	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "OWN") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_IP_EN() = 0)] / none	I/O data exchange
57	Message	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	I/O data exchange
58	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "SLV") && (CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	Wait for sync notification
59	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "SLV") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID))) <= 0) && (CHECK_IP_EN() = 0)] / none	Wait for sync notification
60	Message	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Wait for sync notification
61	Message	EXPIRED_TIMER(T(Tmsgst)) = "True" [CHECK_MSG_RIGHT() = 0] / none	Wait for obtaining of message send right
62	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) > 0)] / DLSDU = BUILD_PDU_TOKEN(V(Nnext_add), Msg); SRC_Send_Frame.req(Port_sel, DLSDU);	Wait for obtaining of message send right
63	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() <> 0)] / none	IP packet communication

T#	Source State	Event (arguments) [conditions] / Action	Target State
64	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_IP_EN() <> 0)] / none	IP packet communication
65	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [CHECK_PDU_TOKEN(TokenType, DLSDU) <> 0] / EXEC_MSGS_SEG(Result, DLSDU, None);	Wait for obtaining of message send right
66	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) <> 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) > 0)] / DESTROY_PDU_CYCR(DLSDU);	Wait for obtaining of message send right
67	Wait for obtaining of message send right	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Wait for obtaining of message send right
68	Wait for obtaining of message send right	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Wait for obtaining of message send right
69	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) > 0)] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
70	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "OWN") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	I/O output data send
71	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "OWN") && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	I/O output data send
72	Wait for obtaining of message send right	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	I/O data exchange

T#	Source State	Event (arguments) [conditions] / Action	Target State
73	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "SLV") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	I/O output data send
74	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "SLV") && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	I/O output data send
75	Wait for obtaining of message send right	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Wait for sync notification
76	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() <> 0)] / none	IP packet communication
77	Wait for obtaining of message send right	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0)] / none	IP packet communication
78	Wait for obtaining of message send right	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF) if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	IP packet communication
79	IP packet communication	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF) if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	IP packet communication
80	IP packet communication	EXPIRED_TIMER(T(Tip)) = "True" [CHECK_IP_PKT() <> 0] / if(CMP_RTIM(T(Tcycle), CALC_IPTIME()) > 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	IP packet communication
81	IP packet communication	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, RcvIPDLSDU) [Rcv_sts = "OK"] / EXEC_IP_SEG(Port_sel, RcvIPDLSDU, None);	IP packet communication
82	IP packet communication	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	IP packet communication

T#	Source State	Event (arguments) [conditions] / Action	Target State
83	IP packet communication	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	IP packet communication
84	IP packet communication	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() = "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() = "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	I/O data exchange
85	IP packet communication	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Wait for sync notification

8.2.2.2.2 CTC Protocol Machine of the slave

The slave CTC state transition chart is as shown in Figure 12 and the slave CTC state transition table is as shown in Table 15.

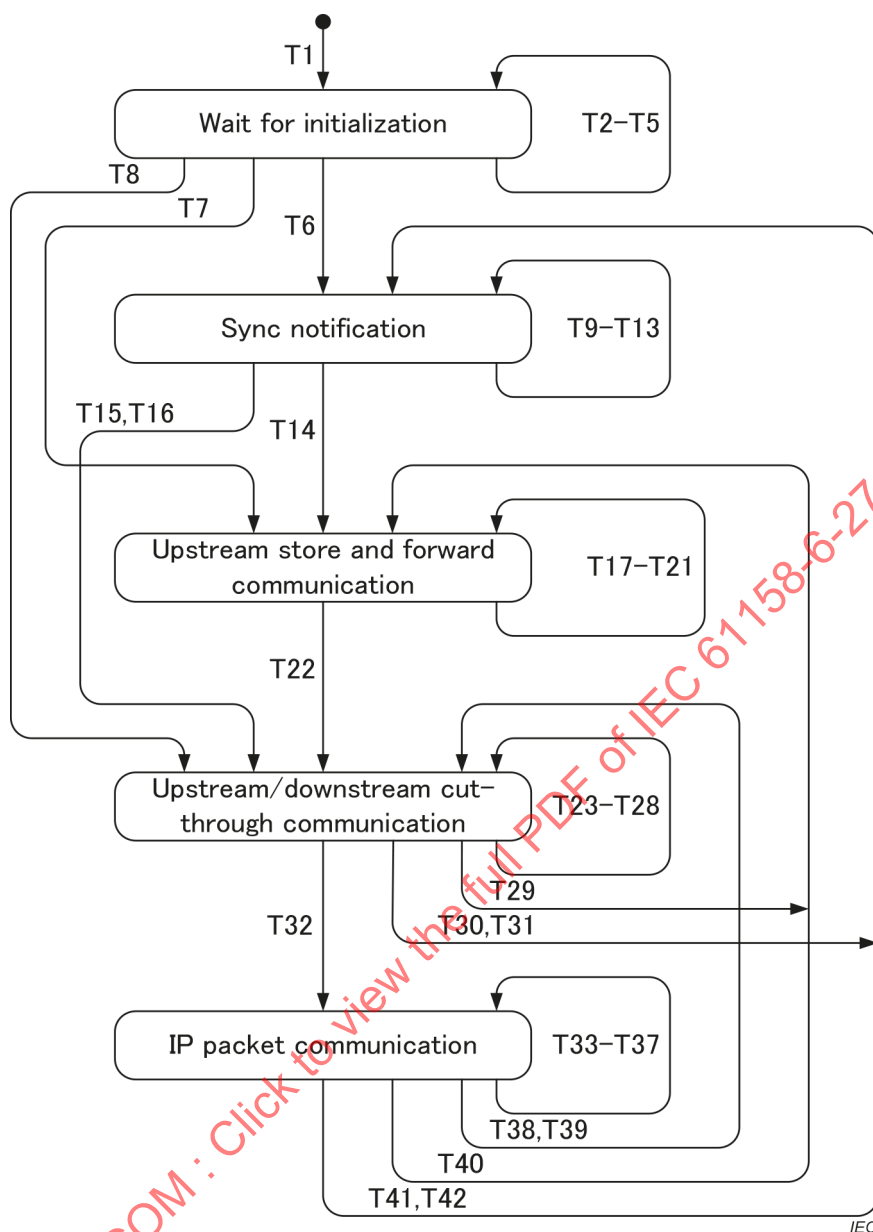


Figure 12 – Slave CTC state transition chart

Table 15 – Slave CTC state transition table

T#	Source State	Event (arguments) [conditions] / Action	Target State
1	All	Power ON or CTC_Reset.req / none	Wait for initialization
2	Wait for initialization	CTC_Set_Par.req(Par_ID, Val) / CTC_SET_PAR(Par_ID, Val, Result); CTC_Set_Par.cnf(Result);	Wait for initialization
3	Wait for initialization	CTC_Get_Par.req(Par_ID) / CTC_GET_PAR(Par_ID, Val, Result); CTC_Get_Par.cnf(Result, Par_ID, Val);	Wait for initialization
4	Wait for initialization	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Wait for initialization

T#	Source State	Event (arguments) [conditions] / Action	Target State
5	Wait for initialization	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Wait for initialization
6	Wait for initialization	CTC_Start.req [(CHECK_SYNC_RIGHT() = "OWN") (CHECK_SYNC_RIGHT() = "SLV")] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Sync notification
7	Wait for initialization	CTC_Start.req [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "LO")] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Upstream store and forward communication
8	Wait for initialization	CTC_Start.req [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI")] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Upstream / downstream cut-through communication
9	Sync notification	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Sync notification
10	Sync notification	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel,DLSDU); SRC_Repeat_Sel.req(CandT,CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Sync notification
11	Sync notification	SRC_Recv_Frame.ind (Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if(Result == "sync") { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tsync)); RESTART_TIMER(T(Tipst)); }	Sync notification
12	Sync notification	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU) CTC-WRITE-DATA.cnf(Result)	Sync notification
13	Sync notification	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Sync notification
14	Sync notification	EXPIRED_TIMER(T(Tsync)) = "True" [MST1_TRANS_MODE() = "LO"] / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end));	Upstream store and forward communication
15	Sync notification	EXPIRED_TIMER(T(Tsync)) = "True" [(MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() <> 0)] / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel,DLSDU); SRC_Repeat_Sel.req(CandT,CandT); SET_SEND_TIMING();	Upstream / downstream Upstream/downstream cut-through communication
16	Sync notification	EXPIRED_TIMER(T(Tsync)) = "True" [(MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() = 0)] / SRC_Repeat_Sel.req(CandT, CandT); SET_SEND_TIMING();	Upstream / downstream cut-through communication

T#	Source State	Event (arguments) [conditions] / Action	Target State
17	Upstream store and forward communication	EXPIRED_TIMER(T(Tcycle)) = "True" / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end)); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Upstream store and forward communication
18	Upstream store and forward communication	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Port_num = 0] / Result = CHECK_PDU_FCMD(DLSDU) domain = CHECK_PDU_SRC(DLSDU) if(Result == "sync") { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tc_end)); RESTART_TIMER(T(Tipst)); } if(BELONG_1ST_DOMAIN() <> 0) { STORE_PDU_CYCC(domain, V(Nslave), DLSDU, Rcv_sts) }	Upstream store and forward communication
19	Upstream store and forward communication	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Port_num <> 0] / EXEC_IP_SEG(Port_num, RcvIPDLSDU, None);	Upstream store and forward communication
20	Upstream store and forward communication	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Upstream store and forward communication
21	Upstream store and forward communication	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Upstream store and forward communication
22	Upstream store and forward communication	EXPIRED_TIMER(T(Tc_end)) = "True" / if(BELONG_1ST_DOMAIN() <> 0) { CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel, DLSDU); } SRC_Repeat_Sel.req(CandT, CandT);	Upstream / downstream cut-through communication
23	Upstream / downstream cut-through communication	SEND_TIMING_EVENT(mst) = "True" / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel, DLSDU);	Upstream / downstream cut-through communication
24	Upstream / downstream cut-through communication	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if(Result == FCMD_SYNC) { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tipst)); if(BELONG_1ST_DOMAIN() <> 0) { STORE_PDU_CYCC(V(Nslave), DLSDU, Rcv_sts) } } else if(Result == FCMD_OUTPUT) { STORE_PDU_CYCC(V(Nslave), DLSDU, Rcv_sts) } else if(Result == FCMD_INPUT) { CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts) } else if(Result == FCMD_REQ_SEND) { Result = CHECK_RETRY(DLSDU, Rcv_sts, Port_num); if(Result == "TRUE") { DLSDU = BUILD_PDU_CYCR(); SRC_Send_Frame.req(Port_sel, DLSDU); } } else if(Result == "FCMD_MSG") { EXEC_MSGS_SEG(Result, DLSDU, DLSDU); }	Upstream / downstream cut-through communication

T#	Source State	Event (arguments) [conditions] / Action	Target State
25	Upstream / downstream cut- through communication	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() <> 0)] / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Upstream / downstream cut- through communication
26	Upstream / downstream cut- through communication	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() = 0)] / START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Upstream / downstream Upstream/downstream cut-through communication
27	Upstream / downstream cut- through communication	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Upstream / downstream cut- through communication
28	Upstream / downstream cut- through communication	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Upstream / downstream Upstream/downstream cut-through communication
29	Upstream / downstream Upstream/downstream cut-through communication	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "LO")] / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end)); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Upstream store and forward communication
30	Upstream / downstream Upstream/downstream cut-through communication	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Sync notification
31	Upstream / downstream Upstream/downstream cut-through communication	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Sync notification
32	Upstream / downstream Upstream/downstream cut-through communication	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF); if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	IP packet communication
33	IP packet communication	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF); if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	IP packet communication

T#	Source State	Event (arguments) [conditions] / Action	Target State
34	IP packet communication	EXPIRED_TIMER(T(Tip)) = "True" [CHECK_IP_PKT() <> 0] / if(CMP_RTIM(T(Tcycle), CALC_IPTIME()) > 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	IP packet communication
35	IP packet communication	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / EXEC_IP_SEG(Port_sel, DLSDU, None);	IP packet communication
36	IP packet communication	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	IP packet communication
37	IP packet communication	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	IP packet communication
38	IP packet communication	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() <> 0)] / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Upstream / downstream Upstream/downstream cut-through communication
39	IP packet communication	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() = 0)] / START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Upstream / downstream Upstream/downstream cut-through communication
40	IP packet communication	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "LO")] / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end)); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Upstream store and forward communication
41	IP packet communication	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Sync notification
42	IP packet communication	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Sync notification

8.2.2.2.2.3 List of CTC macros

Table 16 shows the list of macros used for CTC state transition.

Table 16 – List of CTC macros

Function name	Parameter		Return value	Action
	Input	Output		
CHECK_SYNC_RIGHT	None	None	SM_ID	Notify the node ID when a master other than the 1st master is the sync manager. Return the ID of its own to its own node or other slave (including the 2nd master).
START_TIMER	Tim_ID, Val	None	None	Start the timer specified by Tim_ID at the time-up value specified by Val. This timer executes auto-reload operations.
RESTART_TIMER	Tim_ID	None	None	Correct the timer specified by Tim_ID.
EXPIRED_TIMER	Tim_ID	None	None	Notify the Tim_ID that timer is expired.
CALC_RETIME	None	None	Retry_Tim	Calculate the retry response monitoring time according to the retry list and the retry node number.
CALC_MSGTIME	None	None	Msg_Tim	Calculate the message response monitoring time according to the message data size.
CALC_TOKENTIME	Node_ID	None	Token_Tim	Calculate the time to transmit the token packet to the node specified by Node_ID.
CALC_IPTIME	None	None	IP_Tim	Calculate the IP transmission time according to the IP data size.
CMP_RTIM	Tim_ID1, Tim_ID2	None	Result	Compare the current value of Tim_ID1 and Tim_ID2. Then, if the time until the time up of Tim_ID1 equals to or is greater than Tim_ID2, store 1 as Result.
SET_CYC_DATA	SAP_ID, Node_ID, SDU	None	Result	Update the buffer for I/O data exchange in the CTC specified by SAP_ID and Node_ID in SDU. Return the result regarding whether or not updating has been normally completed.
GET_CYC_DATA	SAP_ID, Node_ID	SDU	Result	Store the data in the buffer for I/O data exchange in the CTC specified by SAP_ID and Node_ID in SDU. Return the result regarding whether or not storing has been normally completed.
BUILD_PDU_SYNC	None	None	DLSDU	Create DLSDU where output data is stored.
BUILD_PDU_CYCC	None	None	DLSDU	Create DLSDU where sync notifications and the output data is stored.
BUILD_PDU_SYNC_CYC C	None	None	DLSDU	Create DLSDU where retransmission requests are stored.
BUILD_PDU_RETREQ	None	None	DLSDU	Create DLSDU where output data is stored.

Function name	Parameter		Return value	Action
	Input	Output		
BUILD_PDU_TOKEN	Node_ID TokenType	None	DLSDU	Create DLSDU where tokens specified by TokenType are stored.
CHECK_PDU_CYCR	DLSDU	AXIS_NUM	Result	Detect the source axis number, AXIS_NUM, of the received DLSDU, compare and check with the axis parameters, and return the result as Result.
CHECK_PDU_FCMD	DLSDU	None	Result	Detect the head command code of the received DLSDU and return the result as Result. Also, store the source axis number as AXIS_NUM.
STORE_PDU_CYCR	AXIS_NUM, Rcv_sts, DLSDU	None	None	Store DLSDU in the received data buffer of AXIS_NUM according to the receiving status, Rcv_sts.
DESTROY_PDU_CYCR	DLSDU	None	None	Destroy the specified DLSDU.
SET_RCVCHK_LIST	None	None	None	Create the input data receive check list.
PUT_RCVCHK_LIST	AXIS_NUM	None	None	Register the axis of axis number specified by AXIS_NUM as normally received in the input data receive check list.
GET_RCVCHK_LIST	None	None	AXIS_NUM	Return the smallest number among the axis numbers described as not normally received in the input data receive check list as the return value.
PUT_RETRY_LIST	AXIS_NUM	None	None	Register the axis number, AXIS_NUM, in the retry list.
RMV_RETRY_LIST	AXIS_NUM	None	None	Delete the axis number, AXIS_NUM, from the retry list.
GET_OUTPUT_NODENUM	None	None	None	Obtain the output data sending node number.
EXEC_MSGC_SEG	Rcv_sts, RcvDLSDU	SndDLSDU	None	This is a message for primary node control function. When receiving message, the receive status is stored to Rcv_sts, and the received data is stored in RcvDLSDU. When sending message, the sending message stored in SndDLSDU is sent.
EXEC_MSGS_SEG	Rcv_sts, RcvDLSDU	SndDLSDU	None	This is a message for secondary node control function. When receiving message, the receive status is stored to Rcv_sts, and the received data is stored in RcvDLSDU. When sending message, the sending message stored in SndDLSDU is sent.
EXEC_IP_SEG	Port_sel, RcvIPDLSDU	SndIPDLSDU	None	Store the packet received at the port by Port_sel in RcvIPDLSDU. Or, send the packet, SndIPDLSDU, from the port by Port_sel.

Function name	Parameter		Return value	Action
	Input	Output		
CHECK_MSG_EN	None	None	Enable_flag	Notify if the message band is set and DLS user issues a message request
CHECK_MSG_RIGHT	None	None	right_flag	Notify if the own node has the message sending right
CHECK_IP_EN	None	None	Enable_flag	Notify if the IP communication band is set
CHECK_TRANS_MODE	None	None	Trans_Mode	Notify whether or not the cyclic communication of its own band is high transfer mode.
MST1_TRANS_MODE	None	None	Trans_Mode	Notify whether or not the cyclic communication of the 1st control domain is high transfer mode.
CHECK_PDU_TOKEN	TokenType, DLSDU	None	Result	Check whether or not the received DLSDU is the token specified by TokenType and notify the result as Result.
CHECK_IP_PKT	None	None	Result	Check whether or not the IP packet to send exists and notify the result as Result.
BELONG_1ST_DOMAIN	None	None	Result	Notify whether or not its own node belongs to the 1st control domain.
BUILD_PDU_CYCR	MST_NUM	None	DLSDU	Create DLSDU where the data to be input to the master specified by MST_NUM is stored.
SET_SEND_TIMING	None	None	None	Set the timing to send a response for each 2nd control domain that it belongs to.
SEND_TIMING_EVENT	MST_NUM	None	None	Notify an occurrence of send event in the control domain that MST_NUM controls.
STORE_PDU_CYCC	AXIS_NUM, Rcv_sts, DLSDU	None	None	Detect data to be output from DLSDU to its own axis number, AXIS_NUM, according to receive status. Rcv_sts, and store in received data buffer.

8.2.2.2.4 CTC-FDC interface

The exchanges of primitives between FDC and CTC are shown below. Table 17 is for the case that issued by FDC and Table 18 is for the case that issued by CTC.

Table 17 – The primitives and parameters for FDC interface issued by FDC

Primitive	Source	Parameter	Function
CTC-WRITE-DATA.req	FDC	SAP_ID, FDCSDU	Send data writing request
CTC-READ-DATA.req	FDC	SAP_ID	Receive data reading request
CTC-SDA.req	FDC	SAP_ID, Node_ID, Length, FDCSDU	Send data request with transmission confirmation
CTC-SDN.req	FDC	SAP_ID, Node_ID, Length, FDCSDU	Send data request without transmission confirmation
CTC-GET-STATUS.req	FDC	Sts_ID	FSMUL parameter referring request

Table 18 – The primitives and parameters for FSM interface issued by CTC

Primitive	Source	Parameter
CTC-WRITE-DATA.cnf	CTC	Result
CTC-READ-DATA.cnf	CTC	Result, FDCSDU
CTC-SDA.ind	CTC	Node_ID, Length, FDCSDU
CTC-SDA.cnf	CTC	Result
CTC-SDN.ind	CTC	Node_ID, Length, FDCSDU
CTC-SDN.cnf	CTC	Result
CTC-GET-STATUS.cnf	CTC	Result, Cur_Status
CTC-EVENT.ind	CTC	Event_ID

8.2.2.2.3 Field System Management Underlying Layer

8.2.2.2.3.1 FSM Protocol Machine

See 8.3.

8.2.2.2.3.2 Primitives exchanged between FSM and FSMUL

The exchanges of primitives between FSM and FSMUL are shown below. Table 19 is for the case that issued by FSM and Table 20 is for the case that issued by FSMUL.

Table 19 – The list of primitives and parameters (FSM source)

Primitive	Source	Parameters
FSMUL-RESET.req	FSM	
FSMUL-SET-VALUE.req	FSM	Var_ID, Val
FSMUL-GET-VALUE.req	FSM	Var_ID
FSMUL-DETECT.req	FSM	
FSMUL-RECOG.req	FSM	
FSMUL-MEAS-DELAY.req	FSM	
FSMUL-ASYNC.req	FSM	
FSMUL-CONFIG.req	FSM	
FSMUL-START-SYNC.req	FSM	
FSMUL-START.req	FSM	
FSMUL-WDLYCNFG.req	FSM	
FSMUL-CLR-ERR.req	FSM	Err_Status

Table 20 – The list of primitives and parameters (FSMUL source)

Primitive	Source	Parameters
FSMUL-RESET.cnf	FSMUL	Result
FSMUL-SET-VALUE.cnf	FSMUL	Result
FSMUL-GET-VALUE.cnf	FSMUL	Result, Val
FSMUL-DETECT.cnf	FSMUL	
FSMUL-RECOG.cnf	FSMUL	
FSMUL-MEAS-DELAY.cnf	FSMUL	Result
FSMUL-MEAS-DELAY.ind	FSMUL	Delay_time
FSMUL-CONFIG.cnf	FSMUL	
FSMUL-START-SYNC.cnf	FSMUL	
FSMUL-START-SYNC.ind	FSMUL	
FSMUL-START.cnf	FSMUL	Result
FSMUL-START.ind	FSMUL	Com_Mode, Cycle_time, Max_Delay, TM_unit
FSMUL-CLR-ERR.cnf	FSMUL	Result
FSMUL-EVENT.ind	FSMUL	Event_ID

8.2.2.2.3.3 Parameters used with FSM primitives

See IEC 61158-5-27, 6.3.1.2.3 for parameters of service primitive.

8.2.2.3 PTP Forwarding Protocol Machine (PTP_FW)

The concepts according to IEEE Std 802.1AS shall be applied. This part of the specification defines the utilization of the IEEE Std 802.1AS.

8.2.3 Virtual bridges

The concepts according to IEEE Std 802.1Q shall be applied. This part of the specification defines the utilization of the IEEE Std 802.1Q.

8.2.4 IP suite

The concepts according to IETF RFC 768, IETF RFC 791, IETF RFC 826 and IETF RFC 1112 shall be applied. This part of the specification defines the utilization of the IETF RFC 791, IETF RFC 826 and IETF RFC 1112.

8.2.5 DLL mapping protocol machine (DMPM)

8.2.5.1 General

The DLL Mapping Protocol consists of several protocol machines:

- Handling of the services invoked by other Application Layer entities;
- Forwarding for RT and PTP frames (RT_FW, PTP_FW);

NOTE 1 Forwarding of standard frames is according to IEEE Std 802.1D and therefore not specified here.

- Mapping to Media Access Control interface (MMAC) and send/receive functions for synchronization messages;

NOTE 2 Invoking of MAC is according to ISO/IEC/IEEE 8802-3 and therefore not specified here.

- PTP is a time stamp function delivered by hardware in the PHY-MAC interface to realize the IEEE Std 802.1AS. This implementation is optional.

The Interface between the service handler, forwarding machines and Mapping to Media Access Control (MMAC) consists of a set of queues and global data.

Figure 13 illustrates the structuring of the protocol machines for the DMPM for a device.

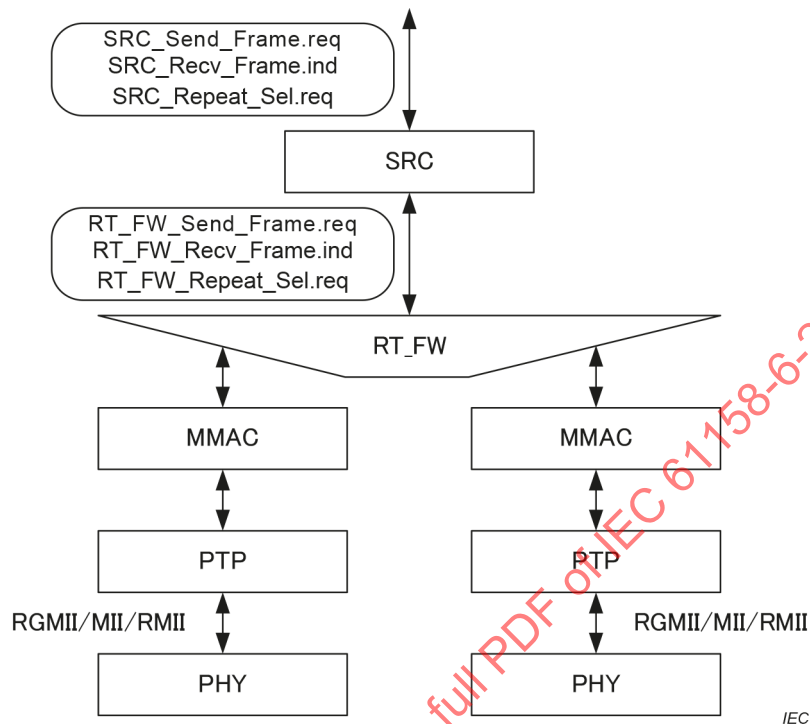


Figure 13 – Structuring of the protocol machines within the DMPM (bridge)

8.2.5.2 Send Receive Control Underlying Layer

8.2.5.2.1 SRC Protocol Machine

The SRC state transition chart is as shown in Figure 14 and the SRC state transition table is as shown in Table 21.

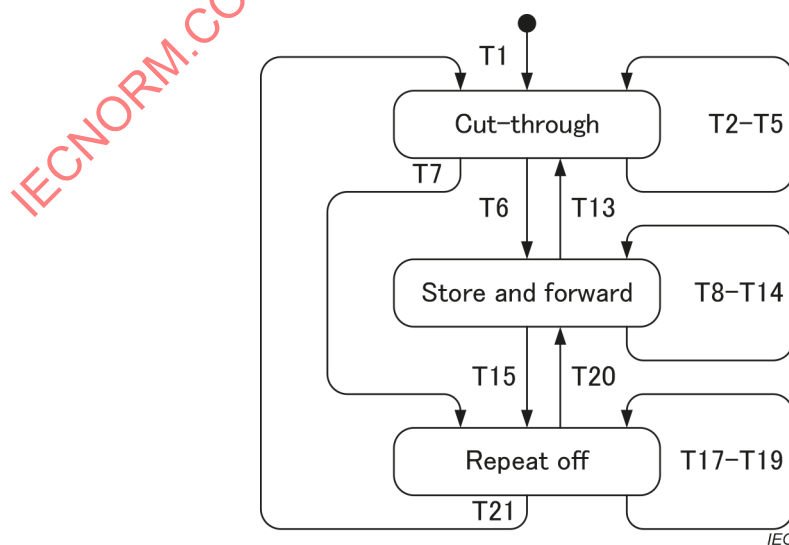


Figure 14 – SRC state transition chart

Table 21 – SRC state transition table

T#	Source State	Event (arguments) [conditions] / Action	Target State
1	All	Power ON / none	Cut-through
2	Cut-through	SRC_Send_Frame.req(Port_sel, DLSDU) / RT_FW_Send_Frame.ind(Port_sel, DLSDU)	Cut-through
3	Cut-through	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "OWN") && (CHECK_LT(DLSDU) = "RTPDU")] / RT_FW_Send_Frame.ind(Sed_sts, DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Cut-through
4	Cut-through	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "COM") && (CHECK_FCMD(DLSDU) = "SYNC")] / RT_FW_Send_Frame.ind(Port_sel, DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU); COR_SYNC_TIMER(V(Trcv), V(TS_fsync));	Cut-through
5	Cut-through	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, DA, SA, DLSDU) [!((CHECK_ADDR(DA) = "OWNDOM") (CHECK_LT(DLSDU) = "RT_PDU")) && !((CHECK_ADDR(DA) = "COM") && (CHECK_FCMD(DLSDU) = "SYNC"))] / RT_FW_Send_Frame.ind(Port_sel, DLSDU);	Cut-through
6	Cut-through	SRC_Repeat_Sel.req(SandF, SandF) / RT_FW_Repeat_Sel.req(SandF, SandF);	Store and forward
7	Cut-through	SRC_Repeat_Sel.req(OFF, OFF) / RT_FW_Repeat_Sel.req(OFF, OFF);	Repeat-off
8	Store and forward	SRC_Send_Frame.req(Port_sel, DLSDU) / RT_FW_Send_Frame.ind(Port_sel, DLSDU);	Store and forward
9	Store and forward	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "BC") (CHECK_ADDR_TABLE(DA) = "True")] / STORE_SANDF(DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Store and forward
10	Store and forward	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [CHECK_ADDR_TABLE(DA) = "True"] / SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Store and forward
11	Store and forward	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [CHECK_ADDR_TABLE(SA) = "True"] / STORE_SANDF(DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Store and forward
12	Store and forward	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) <> "BC") && (CHECK_ADDR_TABLE(DA) <> "True") && (CHECK_ADDR_TABLE(SA) <> "True")] / STORE_SANDF(DLSDU);	Store and forward
13	Store and forward	REPEAT_STS(SandF) = "True" [CHECK_SANDF() <> 0] / while((CHECK_SANDF() <> 0) && (CMP_RTIM(T(Tcycle), (V(Tim) + CALC_SANDF())) > 0) { GET_SANDF(SedSDU); RT_FW_Send_Frame.ind(Sed_sts, SedSDU); }	Store and forward

T#	Source State	Event (arguments) [conditions] / Action	Target State
14	Store and forward	REPEAT_STS(SandF) = "True" [CHECK_IP_PKT() <> 0] / while((CHECK_IP_PKT() <> 0) && (CMP_RTIM(T(Tcycle), (V(Tim) + CALC_IP_PKT()) > 0)) { GET_IP_PKT(DLSDU); RT_FW_Send_Frame.ind(Sed_sts, DLSDU); }	Store and forward
15	Store and forward	SRC_Repeat_Sel.req(CandT, CandT) / RT_FW_Repeat_Sel.req(CandT, CandT);	Cut-through
16	Store and forward	SRC_Repeat_Sel.req(OFF, OFF) / RT_FW_Repeat_Sel.req(OFF, OFF);	Repeat-off
17	Repeat-off	SRC_Send_Frame.req(Port_sel, DLSDU) / RT_FW_Send_Frame.ind(Port_sel, DLSDU);	Repeat-off
18	Repeat-off	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "MC_dom") && (CHECK_LT(DLSDU) = "RT_PDU")) / SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Repeat-off
19	Repeat-off	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "COM") && (CHECK_FCMD() = "SYNC")] / SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU); COR_SYNC_TIMER(V(Trcv), V(TS_fsync));	Repeat-off
20	Repeat-off	SRC_Repeat_Sel.req(SandF, SandF) / RT_FW_Repeat_Sel.req(SandF, SandF);	Store and forward
21	Repeat-off	SRC_Repeat_Sel.req(CandT, CandT) / RT_FW_Repeat_Sel.req(CandT, CandT);	Cut-through

8.2.5.2.2 List of SRC macros

Table 22 shows the list of macros used for SRC state transition.

Table 22 – List of SRC macros

Function name	Parameter		Return value	Action
	Input	Output		
CHACK_ADDR	Addr	None	Result	Return COM when Addr is the common multicast address, return MC when it is the multicast address to neighbouring node, return OWN when it is the multicast address for its own control domain, return DOM when it is the multicast address for other control domain and return OTH for other cases.
CHACK_LT	DLSDU	None	Result	Check LT of DLSDU, and return RTPDU for RT-PDU and return OTH for other cases.
CHECK_FCMD	DLSDU	None	Result	Check _FCMDCode of DLSDU, and return the command code for any one of _PrimaryFCMD and return OTH for other cases.
COR_SYNC_TIME	V(Trcv), V(TS_fsync)	None	None	Correct the sync timer by using Trcv and TS_fsync.

Function name	Parameter		Return value	Action
	Input	Output		
CHECK_ADDR_TABLE	Addr	None	Result	Check whether Addr is registered to the address table. If registered, return "True". If not registered, return "False".
STORE_SANDF	DLSDU	None	None	Store DLSDU in SANDF.
CHECK_SANDF	None	None	Result	Return the number of SDU stored in SANDF.
CALC_SANDF	None	None	Result	Calculate and return the time required to transmit the first SDU stored in SANDF.
GET_SANDF	None	DLSDU	None	Return the first DLSDU stored in SANDF.
CMP_RTIM	Tim_ID1, Tim_ID2	None	Result	Compare the current value of Tim_ID1 and Tim_ID2. Then, if the time of Tim_ID1 until time up is equal to or is greater than Tim_ID2, store 1 as Result.
CHECK_IP_PKT	None	None	Result	Return the number of SRCSDU stored in IP_PKT.
CALC_IP_PKT	None	None	Result	Calculate and return the time required to transmit the first SRCSDU stored in IP_PKT.
GET_IP_PKT	None	DLSDU	None	Return the first DLSDU stored in IP_PKT.

8.2.5.2.3 List of SRC functions

Table 23 shows the functions used for SRC and the descriptions.

Table 23 – List of SRC functions

Name	Meaning
RT_FW_Send_Frame	Send a frame by specifying the port number to send. More than one port number can be specified.
RT_FW_Recv_Frame	Return the received port number, status, received time, destination address, source address and the received frame upon receiving of the frame.
RT_FW_Repeat_Sel	Select the received frame forwarding method for each port.

8.2.5.2.4 SRC-CTC interface

8.2.5.2.4.1 Overview

SRC provides the following services to CTC.

- Send frame service (SRC_Send_Frame);
- Receive frame service (SRC_Recv_Frame);
- Repeat select service (SRC_Repeat_Sel.req).

8.2.5.2.4.2 Detailed definitions of the service primitives and parameters

Table 24 indicates the primitives and parameters which exchanged between SRC and CTC. Details of these parameters are shown in the following subclauses.

Table 24 – Primitives and parameters for SRC-CTC interface

Primitives	Source	Parameters
SRC_Send_Frame.req	CTC	Port_sel, DLSDU
SRC_Send_Frame.cnf	SRC	Result
SRC_Rcv_Frame.ind	SRC	Rcv_sts, Length, DLSDU

a) SRC_Send_Frame

Table 25 indicates the primitives and parameters of send frame service.

Table 25 – Send frame primitive and parameters

SRC_Send_Frame Parameter name	Request	Confirm
	Input	Output
Port_sel	M	
DLSDU	M	
Result		M

b) SRC_Rcv_Frame

Table 26 indicates the primitives and parameters of receive frame service.

Table 26 – Receive frame primitives and parameters

SRC_Rcv_Frame Parameter name	Indication
	Output
Rcv_Sts	M
Length	M
DLSDU	M

c) SRC_Repeat_Sel

Table 27 – Primitives and parameters of repeat select service

SRC_Repeat_Sel Parameter name	Request
	Input
Select_port1	M
Select_port2	M

8.3 Field System Management Protocol Machine (FSM PM)**8.3.1 Overview**

The interface protocol between the FSMUL and the FSM is described in this subclause. FSMUL provide services to FSM for network initialize stations.

In this subclause, the implementation issue and individual issues are intentionally excluded. Figure 15 shows an example of network configuration. In this network configuration example, the master having the band master function (BM) (the 1st master), the slave #1 (S#1), the slave #2 (S#2), the master without having a band master function (M), (the 2nd master), the slave #3 (S#3), the slave #4 (S#4) and a node not supported by this protocol (NS) are connected from one end in series. While BM, S#1 and S#2 belong to the 1st control domain, M, S#3, and S#4 belong to the 2nd control domain.

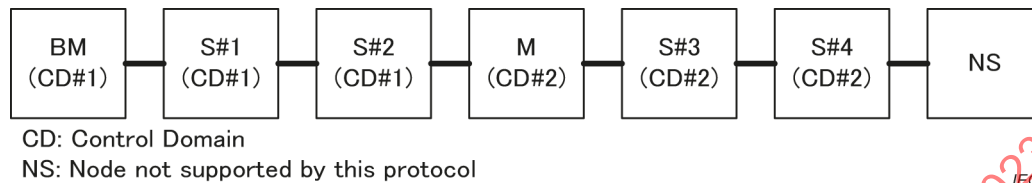


Figure 15 – Example of network configuration

Figure 16 shows the network initialization procedures upon starting up of the system. As shown in Figure 16, some of the procedures can be skipped.

The network initialization before starting the cyclic communication shall be implemented by the master having the band master function (BM). The master having the band master function controls the connection status of all nodes in the network. Therefore, the master having the band master function needs either to have the connecting device information registered prior to the starting up of the system or to collect the connecting device information by detecting connecting nodes. By using the example of network configuration in Figure 15, the sequence with each procedure in Figure 16 is as described below. The case with BM as the sync master (SM) is as described in 8.3.3.1.1 and the case with the master other than BM as the sync master (SM) is as described in 8.3.3.1.2. Note that all nodes connected to the network shall be transferred by the store and forward mode after the Power ON (P-ON) until "Switch forward mode" in Figure 16.

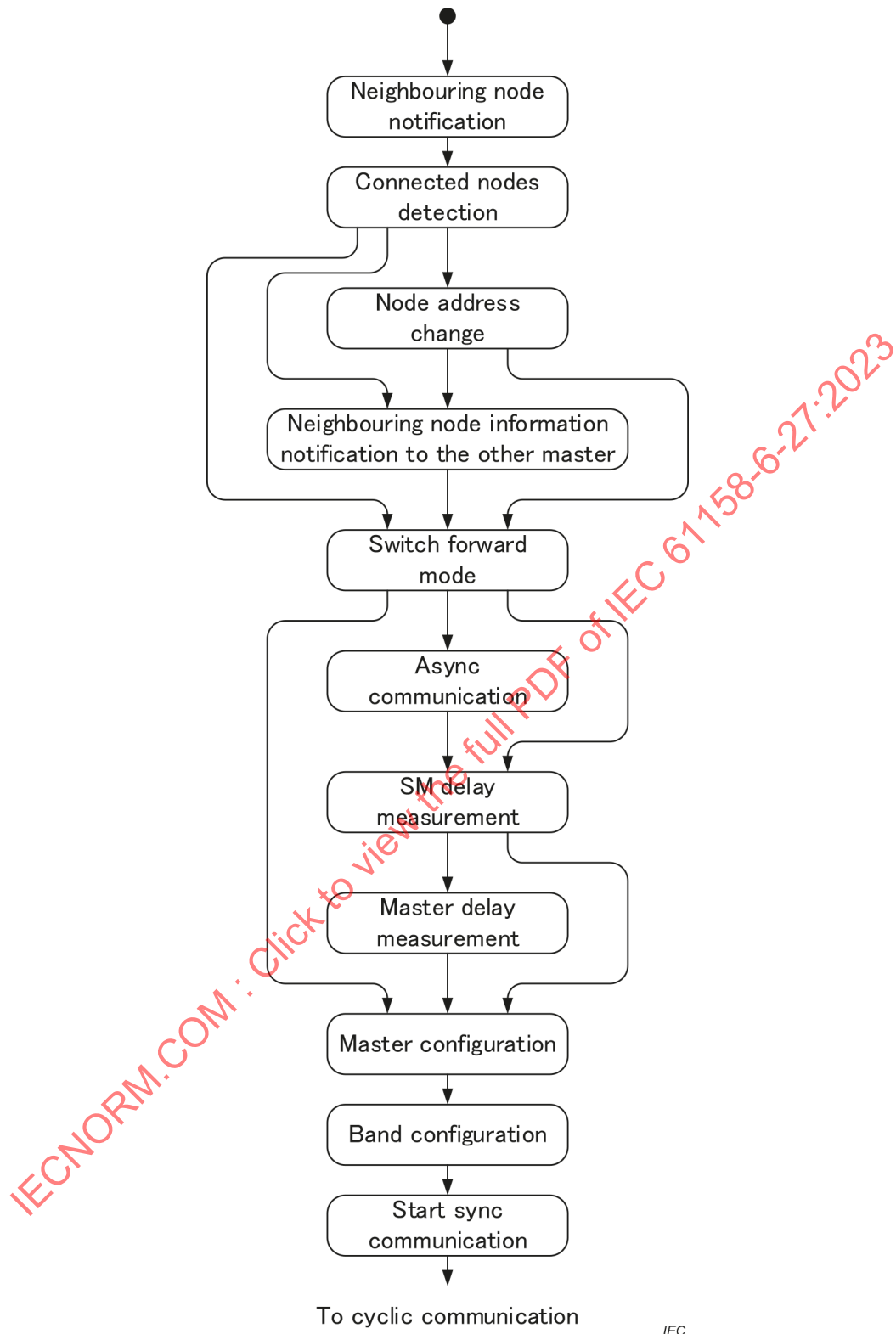


Figure 16 – Network initialization procedures

8.3.2 Discovery and basic configuration

8.3.2.1 Connected node notification

Figure 17 shows the neighbouring node notification sequence to be executed upon Power On of the nodes in the network.

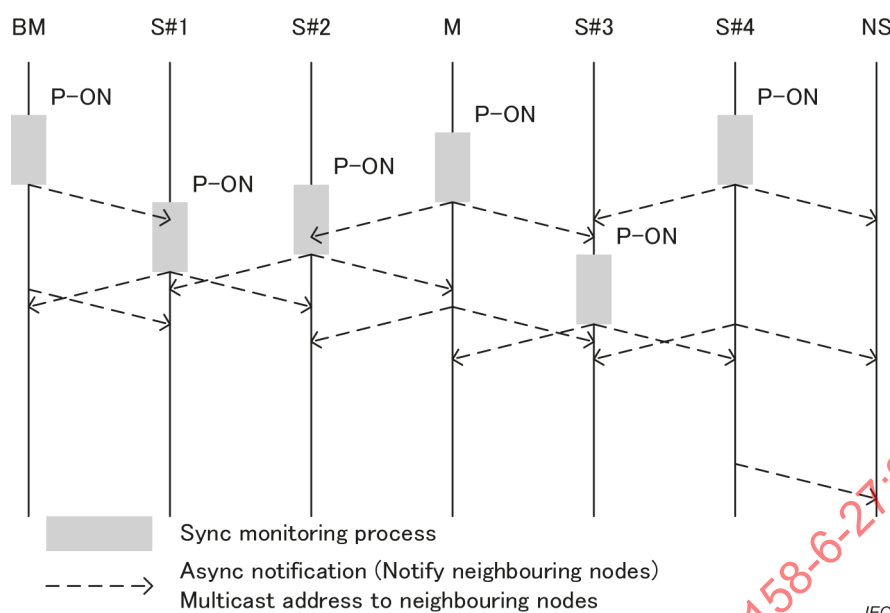


Figure 17 – Neighboring node notification sequence

All nodes supported by this protocol shall monitor the existence/absence of sync packet for a certain period of time after Power ON (P-ON). The (recommended) monitoring time shall be 500 ms. In cases where a sync packet is not detected, an asynchronous notification [event type: neighbouring node notification] shall be multicasted from all the ports (see 4.3.4.10). In this case, the neighbouring node multicast address shall be specified as the destination (See Table 4). In cases where a sync packet is detected, the state shifts to "wait for cyclic communication start" without sending an asynchronous notification [event type: neighbouring node notification] (see 8.3.5 for the following steps).

This process to notify neighbouring nodes shall be executed for each port of the nodes. In cases where a port does not receive a neighbouring node notification from other nodes, the neighbouring node notification shall be continuously sent to the port concerned at certain intervals. The (recommended) intervals of continuous sending shall be 200 ms or less.

In cases where a port receives the neighbouring node notification from other port, sending of neighbouring node notification to the port shall be stopped and the "source MAC address" of the received neighbouring node notification shall be memorized as the neighbouring node information for the port.

8.3.2.2 Connected node notification

After a certain time from the neighbouring node notification described in the preceding subclause, BM sends an asynchronous notification [event type: start-up notification] to other nodes (see 4.3.4.10) by using the common multicast address (see Figure 18). It is recommended to secure the time from executing of neighbouring node notification described in the preceding subclause to sending of start-up notification so that all nodes can send the neighbouring node notification in consideration of the unevenness in the start-up timing for all nodes. As this time is dependent on the system, it shall not be specified in this document.

Other nodes that received the start-up notification from BM shall memorize the source address (node address of BM) and the port received as the upstream port. As described as S#4 in Figure 18, in cases where the neighbouring node notification is continuously sent from the previous step to the node not supported by this protocol, sending of the neighbouring node notification shall be stopped when the start-up notification is received and the neighbouring node information for its own node shall be confirmed. After this, a packet specified by this protocol shall not be sent to the port to which a node not supported by this protocol is connected.

Each node (other than BM) which received the start-up notification shall return an asynchronous notification [event type: start-up notification response] to BM (see 4.3.4.10). In the preceding step, the interval from receiving of the neighbouring node notification from other node to sending of the start-up notification at the BM side shall be 1 s (recommended). Also, the timing for each node to issue a response after receiving the start-up notification shall be within 100 µs from receiving. The start-up notification parameters (operation type, SM supported flag, master delay completed flag (only for master), SM delay time and neighbouring node information) shall be set to the start-up notification response (see 4.3.4.10). BM shall memorize the source address (node address of each node) stored in the start-up notification and the start-up notification parameters.

BM shall wait for the arrival of start-up notification from each node after sending the start-up notification for the detection waiting time. This detection waiting time shall be 2 ms (recommended).

In cases where BM has the start-up notification parameters of all nodes in advance, a mechanism to check with the start-up notification parameters obtained from the start-up notification response are implemented.

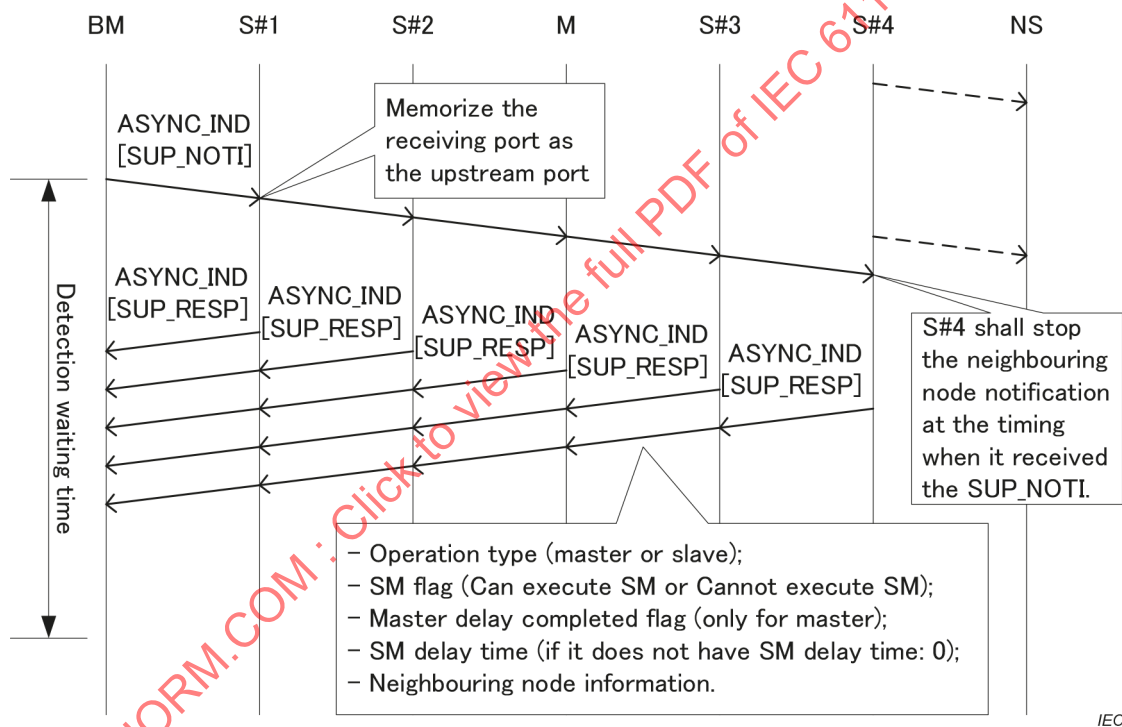


Figure 18 – Connected node detection sequence

Neighbouring node information shall be a list of information obtained in the connected node notification sequence from the neighbouring node notification packet (MAC address) for each port. An example is as shown in Figure 19. In cases where the switch exists between nodes as shown in Figure 19, multiple information (MAC address) exist in one port.

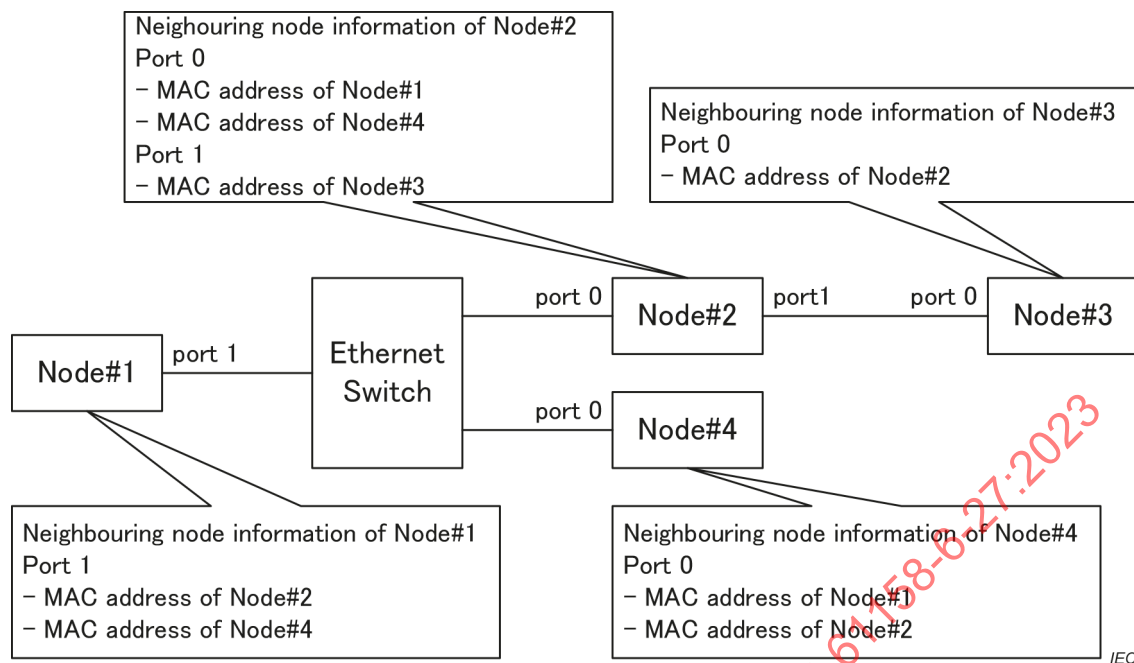


Figure 19 – Example of neighboring node information

8.3.3 Starting up of system

8.3.3.1 Master Protocol Machine

8.3.3.1.1 Master with SM

As for the master with SM function upon network initialization, Figure 20 shows the FSMUL state transition chart and Table 28 shows the FSMUL state transition table.

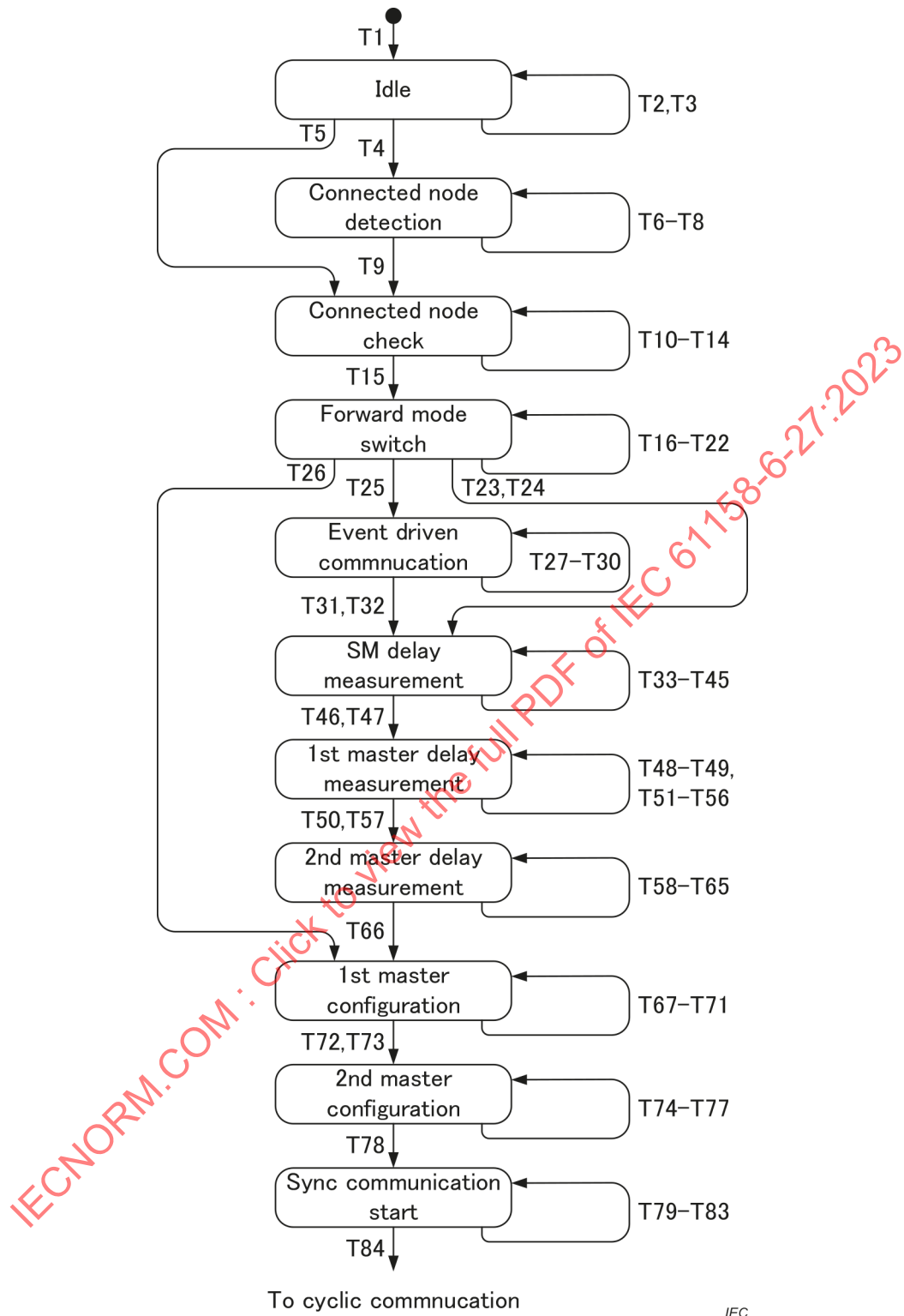


Figure 20 – FSMUL state transition chart for the master with SM function

Table 28 – FSMUL state transition table for the master with SM function

T#	Source State	Event (arguments) [conditions] / Action	Target State
1	All	Power ON or FSMUL-RESET.req / FSMUL-RESET.cnf; CTC_Reset.req; SRC_Reset.req;	Idle
2	Idle	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Idle
3	Idle	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Idle
4	Idle	FSMUL-DETECT.req / = BUILD_PDU_ASYNC_IND(V(Nmul_add), SUP_NOTI); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; START_TIMER(T(Twait), V(Tset_up));	Connected node detection
5	Idle	FSMUL-RECOG.req / V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	Connected node check
6	Connected node detection	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if (Result == FCMD_ASYNC_IND) && (DLSDU.EventType == SUP_RESP) { SET_NODE_INFO(V(Niomap), MAC_Address, DLSDU.SMA); SET_NODE_INFO(V(Niomap), Node_Address, DLSDU.SA); V(Niomap)++; }	Connected node detection
7	Connected node detection	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Connected node detection
8	Connected node detection	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Connected node detection
9	Connected node detection	EXPIRED_TIMER(T(Twait)) = "True" / V(Nmax_nodes) = V(Niomap); V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); FSMUL-DETECT.cnf; DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Connected node check

T#	Source State	Event (arguments) [conditions] / Action	Target State
10	Connected node check	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) < V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Nstatus)); V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	Connected node check
11	Connected node check	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) < V(Nmax_nodes)] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Connected node check
12	Connected node check	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) >= V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Nstatus)) } } </pre>	Connected node check
13	Connected node check	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Connected node check
14	Connected node check	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Connected node check
15	Connected node check	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) >= V(Nmax_nodes)] / FSMUL-RECOG.cnf; DLSDU = BUILD_PDU_CDO_WRITE(V(Nmulti_add), Req, trans_mode, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; </pre>	Forward mode switch
16	Forward mode switch	<pre> SRC_Send_Frame.cnf() [V(Niomap) == 0] / V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, trans_mode, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Forward mode switch

T#	Source State	Event (arguments) [conditions] / Action	Target State
17	Forward mode switch	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) < V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Trans_Mode, V(Ntrans_mode)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Trans_Mode, V(Ntrans_mode)); V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), V(Niomap), Node_Address) DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, trans_mode, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	Forward mode switch
18	Forward mode switch	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) < V(Nmax_nodes)] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, trans_mode, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Forward mode switch
19	Forward mode switch	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) >= V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Ntrans_mode)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Ntrans_mode)); } Result = CHECK_TRANS_MODE(); if(Result == "OK") { FSMUL-SW-TRANS-MODE.cnf; } DLSDU = BUILD_PDU_CDO_WRITE(V(Nmulti_add), Req, Trans_Mode, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; } </pre>	Forward mode switch
20	Forward mode switch	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) >= V(Nmax_nodes)] / DLSDU = BUILD_PDU_CDO_WRITE(V(Nmulti_add), Req, Trans_Mode, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; </pre>	Forward mode switch
21	Forward mode switch	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Forward mode switch
22	Forward mode switch	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Forward mode switch
23	Forward mode switch	<pre> FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; START_TIMER(T(Twait), V(Tresp)); </pre>	SM delay measurement (SM of its own node)

T#	Source State	Event (arguments) [conditions] / Action	Target State
24	Forward mode switch	FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "SLV"] / DLSDU = BUILD_PDU_ASYNC_IND(V(Nsm_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 'FF'H; START_TIMER(T(Twait), V(Tresp));	SM delay measurement (SM of other node)
25	Forward mode switch	FSMUL-ASYNC.req / none	Event driven communication
26	Forward mode switch	FSMUL-CONFIG.req / none	1st master configuration
27	Event driven communication	CTC-SDN.req(SAP_ID, Node_ID, Length, SDU) / DLSDU = BUILD_PDU_EVD(Node_ID, Length, SDU); Len = GET_LEN(DLSDU); SRC_Send_Frame.req(Port_sel, Node_ID, Len, DLSDU); CTC-SDN.cnf("OK");	Event driven communication
28	Event driven communication	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Node_ID = GET_DA(DLSDU); Len = GET_LEN(DLSDU); DLSDU = GET_DATA(DLSDU); CTC-SDN.ind(SAP_ID, Node_ID, Len, SDU);	Event driven communication
29	Event driven communication	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Event driven communication
30	Event driven communication	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Event driven communication
31	Event driven communication	FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "OWN"] / SDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; START_TIMER(T(Twait), V(Tresp));	SM delay measurement (SM of its own node)
32	Event driven communication	FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "SLV"] / DLSDU = BUILD_PDU_ASYNC_IND(V(Nsm_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 'FF'H; START_TIMER(T(Twait), V(Tresp));	SM delay measurement (SM of other node)
33	SM delay measurement (SM of its own node)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) =< V(Nmax_nodes)] / V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_REQ_SEND(V(Nnode_add), MEAS_DLY, V(Nmy_add), V(Nmy_mac_add)); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	SM delay measurement (SM of its own node)
34	SM delay measurement (SM of its own node)	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if(Result == FCMD_MEAS_DLY) && (DLSDU.HS == REQ) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), None); GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Resp, V(Treq_s), V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap)++; }	SM delay measurement (SM of its own node)

T#	Source State	Event (arguments) [conditions] / Action	Target State
35	SM delay measurement (SM of its own node)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) > V(Nmax_nodes)] / V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	SM delay measurement
36	SM delay measurement (SM of other node)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) = 'FF'H] / DLSDU = BUILD_PDU_ASYNC_IND(V(Nsm_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	SM delay measurement (SM of other node)
37	SM delay measurement (SM of other node)	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Niomap) = 'FF'H)] / Result = CHECK_PDU_FCMD(DLSDU) if(Result == FCMD_SYNC) { GET_SYNC_DATA(DLSDU, V(Tsyn_s)); GET_RECV_TIME(V(Tsyn_r)); V(Niomap) = 0; DLSDU = BUILD_PDU_MEAS_DLY(V(Nsm_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); }	SM delay measurement (SM of other node)
38	SM delay measurement (SM of other node)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) = 0] / DLSDU = BUILD_PDU_MEAS_DLY(V(Nmy_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	SM delay measurement (SM of other node)
39	SM delay measurement (SM of other node)	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && V(Niomap) = 0] / Result = CHECK_PDU_FCMD(DLSDU) && if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == RESP) (DLSDU.SA == V(Nsm_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = ((Treq_r - Tsyn_s) - (Treq_s - Tsyn_r)) / 2; SET_NODE_INFO(V(Nmst), Delay, V(Ndelay)); V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_REQ_SEND(V(Nnode_add), MEAS_DLY, V(Nmy_add), V(Nmy_mac_add)); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Treq_intvl)); }	SM delay measurement (SM of other node)
40	SM delay measurement (SM of other node)	EXPIRED_TIMER(T(Twait)) = "True" [V(Nmax_nodes) > V(Niomap) > 0] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_REQ_SEND(V(Nnode_add), MEAS_DLY, V(Nmy_add), V(Nmy_mac_add)); SRC_Send_Frame.req(Port_sel, DLSDU); if(V(Niomap) <> V(Nmax_nodes)) { START_TIMER(T(Twait), V(Treq_intvl)); }	SM delay measurement (SM of other node)

T#	Source State	Event (arguments) [conditions] / Action	Target State
41	SM delay measurement (SM of other node)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Niomap) >= V(Nmax_nodes))] / Result = CHECK_PDU_FCMD(DLSDU) if((Result == FCMD_ASYNC_IND) && (DLSDU.EventType == MEAS_DLY)) { V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts)); } </pre>	SM delay measurement
42	SM delay measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) < V(Nmax_nodes)] / Result := CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Nstatus)); V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	SM delay measurement
43	SM delay measurement	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) < V(Nmax_nodes)] / DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	SM delay measurement
44	SM delay measurement	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	SM delay measurement
45	SM delay measurement	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	SM delay measurement
46	SM delay measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(V(Niomap) >= V(Nmax_nodes)) && (CHECK_SYNC_RIGHT() = "OWN")] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Nstatus)); V(Ndmn1) = 1; V(Niomap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, delay_time, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	1st master delay measurement (SM of its own node)

T#	Source State	Event (arguments) [conditions] / Action	Target State
47	SM delay measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(V(Niormap) >= V(Nmax_nodes)) && (CHECK_SYNC_RIGHT() = "SLV")] / Result:= CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Status, V(Nstatus)); DLSDU = BUILD_PDU_REQ_SEND(V(Nsm_add), SYNC, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Ndmn1) = 'FF'H; START_TIMER(T(Twait), V(Tresp)); } } </pre>	1st master delay measurement (SM of other node)
48	1st master delay measurement (SM of its own node)	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Ndmn1) < V(Nmax_dmn1)] / Result = CHECK_PDU_FCMD(DLSDU) if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Delay, V(Ndelay)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Delay, V(Ndelay)); V(Ndmn1)++; V(Niormap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, delay_time, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	1st master delay measurement (SM of its own node)
49	1st master delay measurement (SM of its own node)	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niormap) < V(Nmax_nodes)] / DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1st master delay measurement (SM of its own node)
50	1st master delay measurement (SM of its own node)	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Ndmn1) >= V(Nmax_dmn1)] / Result = CHECK_PDU_FCMD(DLSDU) if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Delay, V(Ndelay)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Delay, V(Ndelay)); V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); } } </pre>	2nd master delay measurement
51	1st master delay measurement (SM of other node)	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niormap) = 'FF'H] / DLSDU = BUILD_PDU_REQ_SEND(V(Nsm_add), SYNC, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1st master delay measurement (SM of other node)

T#	Source State	Event (arguments) [conditions] / Action	Target State
52	1st master delay measurement (SM of other node)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Ndmn1) = 'FF'H)] / Result = CHECK_PDU_FCMD(DLSDU); if(Result == FCMD_SYNC) { RESTART_TIMER(Tcycle); V(Ndmn1) = 1; V(Niormap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } </pre>	2nd master delay measurement (SM of other node)
53	1st master delay measurement (SM of other node)	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Ndmn1) < V(Nmax_dmn1)] / DLSDU = BUILD_PDU_MEAS_DLY(V(Niormap), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	2nd master delay measurement (SM of other node)
54	1st master delay measurement (SM of other node)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Ndmn1) < V(Nmax_dmn1))] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == RESP) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = (Treq_r – Treq_s) / 2; SET_NODE_INFO(V(Nmst), Delay, V(Ndelay)); V(Ndmn1)++; V(Niormap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } </pre>	2nd master delay measurement (SM of other node)
55	1st master delay measurement	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	1st master delay measurement
56	1st master delay measurement	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	1st master delay measurement
57	1st master delay measurement (SM of other node)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Ndmn1) >= V(Nmax_dmn1))] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == RESP) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = (Treq_r – Treq_s) / 2; SET_NODE_INFO(V(Nmst), Delay, V(Ndelay)); V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	2nd master delay measurement

T#	Source State	Event (arguments) [conditions] / Action	Target State
58	2nd master delay measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_WRITE_CDO) && (DLSDU.HS == REQ) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if((Result == "OK") && (CDO_ID == mst2_uport)) { SET_CDO_DATA(CDO_ID, Port_num); Result = 0; DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Ack, CDO_ID, Result); SRC_Send_Frame.req(Port_sel, DLSDU); } } </pre>	2nd master delay measurement
59	2nd master delay measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_ASYNC_IND) && (DLSDU.EventType == MEAS_DLY) && (DLSDU.SA == V(Nnode_add))) { DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	2nd master delay measurement
60	2nd master delay measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU) && (DLSDU.SA == V(Nnode_add)); if(Result == FCMD_SYNC) { RESTART_TIMER(Tcycle) } </pre>	2nd master delay measurement
61	2nd master delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == REQ) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), None); GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Resp, V(Treq_s), V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	2nd master delay measurement
62	2nd master delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (Result = FCMD_TOKEN) && (DLSDU.TokenType == MST_MEAS_DLY) && (DLSDU.SA == V(Nnode_add)) && (V(Nmst2_index) < V(Nmst2_index_max))] / V(Nmst2_index)++; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	2nd master delay measurement
63	2nd master delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType == MST_MEAS_DLY) && (DLSDU.SA == V(Nnode_add)) && (V(Nmst2_index) >= V(Nmst2_index_max))] / FSMUL-MEAS-DELAY.cnf; </pre>	2nd master delay measurement
64	2nd master delay measurement	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	2nd master delay measurement
65	2nd master delay measurement	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	2nd master delay measurement

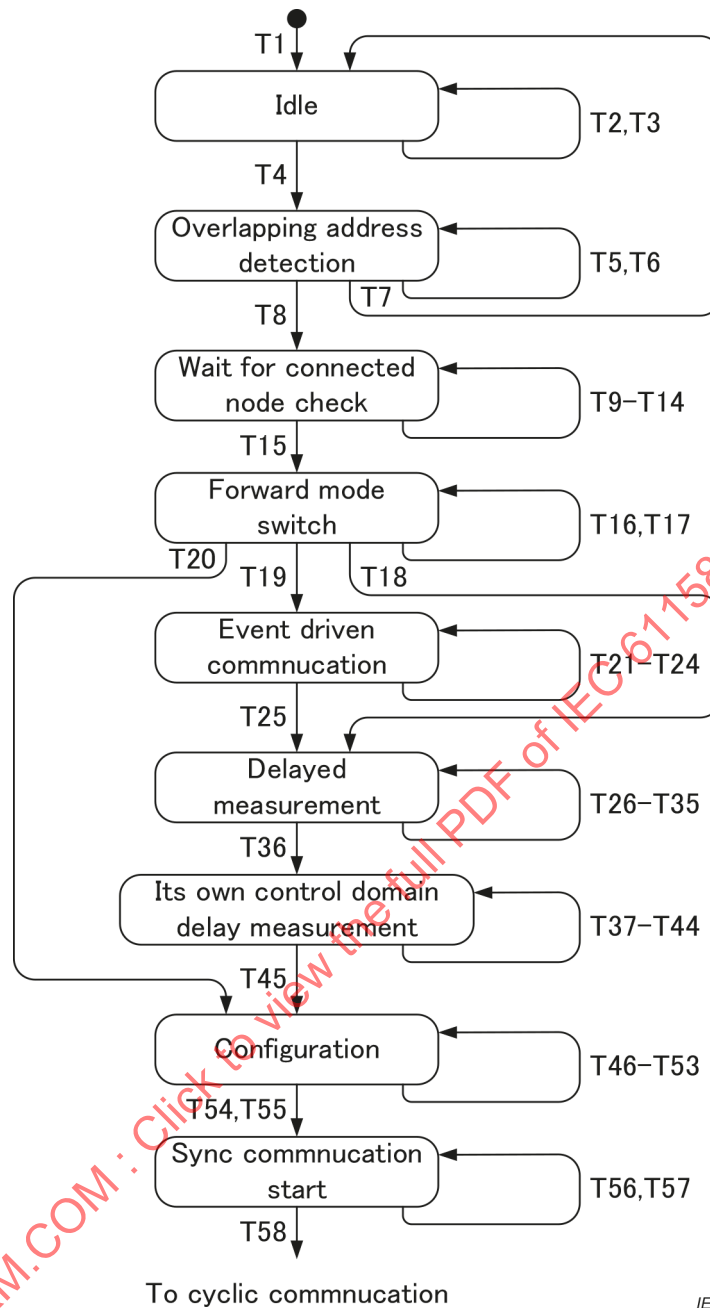
T#	Source State	Event (arguments) [conditions] / Action	Target State
66	2nd master delay measurement	<pre> FSMUL-CONFIG.req / V(Ncnf_index) = 0; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1st master configuration
67	1st master configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / Result = GET_CDO_READ_DATA(DLSDU, 1, V(Ncdo_id), V(Ncdo_data)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), V(Ncdo_id), V(Ncdo_data)); V(Ncnf_index)++; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = SET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } </pre>	1st master configuration
68	1st master configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / V(Ncnf_index)++; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = SET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1st master configuration
69	1st master configuration	<pre> EXPIRED_TIMER(T(Twait)) = "True" / if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1st master configuration

T#	Source State	Event (arguments) [conditions] / Action	Target State
70	1st master configuration	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	1st master configuration
71	1st master configuration	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	1st master configuration
72	1st master configuration	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) >= V(Ncnf_index_max))] / Result = GET_CDO_READ_DATA(DLSDU, 1, V(Ncdo_id), V(Ncdo_data)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), V(Ncdo_id), V(Ncdo_data)); V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), CONFIG); SRC_Send_Frame.req(Port_sel, DLSDU); }	2nd master configuration
73	1st master configuration	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), CONFIG); SRC_Send_Frame.req(Port_sel, DLSDU);	2nd master configuration
74	2nd master configuration	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType = CONFIG) && (DLSDU.SA = V(Nnode_add)) && (V(Nmst2_index) < V(Nmst2_index_max))] / V(Nmst2_index)++; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU);	2nd master configuration
75	2nd master configuration	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType = MST_MEAS_DLY) && (DLSDU.SA = V(Nnode_add)) && (V(Nmst2_index) >= V(Nmst2_index_max))] / FSMUL-CONFIG.cnf;	2nd master configuration
76	2nd master configuration	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	2nd master configuration
77	2nd master configuration	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	2nd master configuration
78	2nd master configuration	FSMUL-START-SYNC.req / V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); if(V(Nnode_add) = V(Nsm_add)) { V(Nnode_add)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); } DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Sync communication start

T#	Source State	Event (arguments) [conditions] / Action	Target State
79	Sync communication start	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = cyclic_start) && (V(Niomap) < V(Nmax_nodes))] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); if(V(Nnode_add) == V(Nsm_add)) { V(Nnode_add)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); } DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts)); </pre>	Sync communication start
80	Sync communication start	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = cyclic_start) && (V(Niomap) >= V(Nmax_nodes))] / V(Nnode_add) = V(Nsm_add); DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts)); </pre>	Sync communication start
81	Sync communication start	<pre> EXPIRED_TIMER(T(Twait)) = "True" / DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts)); </pre>	Sync communication start
82	Sync communication start	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Sync communication start
83	Sync communication start	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Sync communication start
84	Sync communication start	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nsm_add)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = cyclic_start)] / CTC_Start.req; </pre>	To cyclic communication

8.3.3.1.2 Master without SM

As for the master without SM function upon network initialization, Figure 21 shows the FSMUL state transition chart and Table 29 shows the FSMUL state transition table.



IEC

Figure 21 – FSMUL state transition chart for the master without SM function**Table 29 – FSMUL state transition table for the master without SM function**

T#	Source State	Event (arguments) [conditions] / Action	Target State
1	All	Power ON or FSMUL-RESET.req / FSMUL-RESET.cnf; CTC_Reset.req; SRC_Reset.req; SET_CDO_DATA(Status, IDLE);	Idle
2	Idle	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Idle
3	Idle	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Idle

T#	Source State	Event (arguments) [conditions] / Action	Target State
4	Idle	FSMUL-START.req / RECEIVE_ALL_PKT(ON); START_TIMER(T(Twait), V(Tdt_same_add)); SET_CDO_DATA(Status, DT_ADD_ERR);	Overlapping address detection
5	Overlapping address detection	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Overlapping address detection
6	Overlapping address detection	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Overlapping address detection
7	Overlapping address detection	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (DLSDU.SA = V(Nmy_add))] / FSMUL-CLR-ERR.cnf(Adress_error);	Overlapping address detection
8	Overlapping address detection	EXPIRED_TIMER(T(Twait)) = "True" / RECEIVE_ALL_PKT(OFF); SET_CDO_DATA(Status, WRECOG);	Wait for connected node check
9	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_ASYNC_IND = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.EventType = SUP_NOTI)] / SET_CDO_DATA(Mst1_mac_add, DLSDU.SRC_MAC_ADD); SET_CDO_DATA(Mst1_node_add, DLSDU.SA); SET_CDO_DATA(Mst1_uport, Port_num); DLSDU = BUILD_PDU_ASYNC_IND(DLSDU.SA, SUP_RESP); SRC_Send_Frame.req(Port_num, DLSDU);	Wait for connected node check
10	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) if(Result == "OK") { GET_CDO_DATA(CDO_ID, CDO_data) DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, CDO_ID, OK, CDO_data) SRC_Send_Frame.req(Port_num, DLSDU) if(CDO_ID == Status) { DLSDU = BUILD_PDU_CDO_WRITE(V(Nmul_next), REQ, Connect, 1); SRC_Send_Frame.req(Port_sel, DLSDU); } } }	Wait for connected node check
11	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if((Result == "OK") && (CDO_ID == Connect)) { SET_CDO_DATA(GET_CDO_PORTID(Port_num), 1); }	Wait for connected node check
12	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if(Result == "OK") { SET_CDO_DATA(CDO_ID, CDO_data); DLSDU = BUILD_PDU_CDO_WRITE(DLSDU.SA, ACK, CDO_ID, OK); SRC_Send_Frame.req(Port_num, DLSDU); if(CDO_ID == Connect) { SET_CDO_DATA(GET_CDO_PORTID(Port_num), 1); }	Wait for connected node check
13	Wait for connected node check	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Wait for connected node check

T#	Source State	Event (arguments) [conditions] / Action	Target State
14	Wait for connected node check	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Wait for connected node check
15	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = Trans_Mode)] / CHANGE_TRANS_MODE(); SET_CDO_DATA(Status, TRANS_MODE);	Forward mode switch
16	Forward mode switch	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Forward mode switch
17	Forward mode switch	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Forward mode switch
18	Forward mode switch	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)) && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode) && FSMUL-MEAS-DELAY.req] / GET_CDO_DATA(Trans_Mode, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); V(Ndly_cnt) = 0; SET_CDO_DATA(Status, MEASURED_DLY);	Delayed measurement
19	Forward mode switch	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)) && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode) && FSMUL-ASYNC.req] / GET_CDO_DATA(Trans_Mode, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); SET_CDO_DATA(Status, ASYNC);	Event driven communication
20	Forward mode switch	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)) && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode) && FSMUL-CONFIG.req] / GET_CDO_DATA(Trans_Mode, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); SET_CDO_DATA(Status, WCONFIG);	Wait for configuration
21	Event driven communication	CTC-SDN.req(SAP_ID, Node_ID, Length, DLSDU) / DLSDU = BUILD_PDU_EVD(Node_ID, Length, SDU); Len = GET_LEN(DLSDU); SRC_Send_Frame.req(Port_sel, Node_ID, Len, DLSDU); CTC-SDN.cnf("OK");	Event driven communication
22	Event driven communication	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Node_ID = GET_DA(DLSDU); Len = GET_LEN(DLSDU); DLSDU = GET_DATA(DLSDU); CTC-SDN.ind(SAP_ID, Node_ID, Len, SDU);	Event driven communication

T#	Source State	Event (arguments) [conditions] / Action	Target State
23	Event driven communication	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Event driven communication
24	Event driven communication	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Event driven communication
25	Event driven communication	FSMUL-MEAS-DELAY.req / V(Ndly_cnt) = 0; SET_CDO_DATA(Status, MEASURED_DLY);	Delayed measurement
26	Delayed measurement	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_ASYNC_IND = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.EventType = MEAS_DLY)] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU);	Delayed measurement
27	Delayed measurement	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS = REQ)) && (V(Ndly_cnt) < V(Nmax_nodes))] / GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SA, RESP, DLSDU.TimestampT3, V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU) V(Ndly_cnt)++;	Delayed measurement
28	Delayed measurement	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS = REQ)) && (V(Ndly_cnt) >= V(Nmax_nodes))] / GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SA, RESP, DLSDU.TimestampT3, V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); DLSDU = BUILD_PDU_ASYNC_IND(V(Nmst1_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); SET_CDO_DATA(Status, MEAS_DLY_CMP); V(Ndly_cnt) = 0;	Delayed measurement
29	Delayed measurement	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_SYNC = CHECK_PDU_FCMD(DLSDU))] / GET_CDO_DATA(Status, V(Nstatus)); if (V(Nstatus) == MEAS_DLY_CMP) { RESTART_TIMER(Tcycle); } else { GET_SYNC_DATA(DLSDU, V(Tsyn_s)); GET_RECV_TIME(V(Tsyn_r)); }	Delayed measurement
30	Delayed measurement	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_REQ_SEND = CHECK_PDU_FCMD(DLSDU) && (DLSDU.PollingType = MEAS_DLY))] / DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SM_ADD, REQ, none, none); SRC_Send_Frame.req(Port_sel, DLSDU);	Delayed measurement
31	Delayed measurement	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS = RESP) && (DLSDU.SA = V(Nsm_add)))] => GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = ((Treq_r - Tsyn_s) - (Treq_s - Tsyn_r)) / 2; SET_CDO_DATA(DELAY, V(Ndelay)); SET_CDO_DATA(Status, MEAS_DLY_CMP);	Delayed measurement

T#	Source State	Event (arguments) [conditions] / Action	Target State
32	Delayed measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none); if(Result == "OK") { GET_CDO_DATA(CDO_ID, CDO_data) DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, CDO_ID, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); } </pre>	Delayed measurement
33	Delayed measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_REQ_SEND = CHECK_PDU_FCMD(DLSDU) && (DLSDU.PollingType = SYNC))] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	Delayed measurement
34	Delayed measurement	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Delayed measurement
35	Delayed measurement	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Delayed measurement
36	Delayed measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType = MST_MEAS_DLY)] / V(Ndmn2) = 1; V(Niomap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, mst2_uport, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); </pre>	Its own control domain delay measurement
37	Its own control domain delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) < V(Nmax_dmn2))] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) if(Result = "OK") { V(Ndmn2)++; V(Niomap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, mst2_uport, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); } </pre>	Its own control domain delay measurement

T#	Source State	Event (arguments) [conditions] / Action	Target State
38	Its own control domain delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) >= V(Nmax_dmn2)) && (CHECK_SYNC_RIGHT() = "OWN")] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); V(Ndmn2) = 1; V(Niomap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); </pre>	Its own control domain delay measurement
39	Its own control domain delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) >= V(Nmax_dmn2)) && (CHECK_SYNC_RIGHT() = "SLV")] / DLSDU = BUILD_PDU_REQ_SEND(V(Nsm_add), SYNC, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); </pre>	Its own control domain delay measurement
40	Its own control domain delay measurement	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_SYNC = CHECK_PDU_FCMD(DLSDU))] / RESTART_TIMER(Tcycle); V(Ndmn2) = 1; V(Niomap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); </pre>	Its own control domain delay measurement
41	Its own control domain delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = RESP) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) < V(Nmax_dmn2))] / GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = (Treq_r - Treq_s) / 2; SET_NODE_INFO(V(Niomap), Delay, V(Ndelay)); V(Ndmn2)++; V(Niomap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); </pre>	Its own control domain delay measurement

T#	Source State	Event (arguments) [conditions] / Action	Target State
42	Its own control domain delay measurement	<pre> EXPIRED_TIMER(T(Twait)) = "True" / if(V(Nlast_frame) == FCMD_WRITE_CDO) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, mst2_uport, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } else if(V(Nlast_frame) == FCMD_REQ_SEND) { DLSDU = BUILD_PDU_REQ_SEND(V(Nsm_add), SYNC, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } else if(V(Nlast_frame) == FCMD_MEAS_DLY) { DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } </pre>	Its own control domain delay measurement
43	Its own control domain delay measurement	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Its own control domain delay measurement
44	Its own control domain delay measurement	<pre> FSMUL-GET-VALUE.req(Par_ID) => Result = GET_VALUE(Var_ID, CurVal) FSMUL-GET-VALUE.cnf(Result, CurVal) </pre>	Its own control domain delay measurement
45	Its own control domain delay measurement	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = RESP) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) >= V(Nmax_dmn2))] / GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = (Treq_r - Treq_s) / 2; SET_NODE_INFO(V(Niomap), Delay, V(Ndelay)); DLSDU = BUILD_PDU_TOKEN(V(Nmst1_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	Configuration
46	Configuration	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none); if(Result == "OK") { GET_CDO_DATA(CDO_ID, CDO_data); DLSDU = BUILD_PDU_CDO_READ(DLSDU.SA, ACK, CDO_ID, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); } </pre>	Configuration
47	Configuration	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if(Result == "OK") { SET_CDO_DATA(CDO_ID, CDO_data); DLSDU = BUILD_PDU_CDO_WRITE(DLSDU.SA, ACK, CDO_ID, OK); SRC_Send_Frame.req(Port_num, DLSDU); } </pre>	Configuration

T#	Source State	Event (arguments) [conditions] / Action	Target State
48	Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType = CONFIG)] / V(Ncnf_index) = 0; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Configuration
49	Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / Result = GET_CDO_READ_DATA(DLSDU, 1, V(Ncdo_id), V(Ncdo_data)); if(Result = "OK") { SET_NODE_INFO(V(Niomap), V(Ncdo_id), V(Ncdo_data)); V(Ncnf_index)++; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = SET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } </pre>	Configuration
50	Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / V(Ncnf_index)++; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = SET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Configuration
51	Configuration	<pre> EXPIRED_TIMER(T(Twait)) = "True" / if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Configuration
52	Configuration	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Configuration

T#	Source State	Event (arguments) [conditions] / Action	Target State
53	Configuration	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Configuration
54	Configuration	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_READ_DATA(DLSDU, 1, V(Ncdo_id), V(Ncdo_data)) = "OK") && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) >= V(Ncnf_index_max))] / SET_NODE_INFO(V(Niomap), V(Ncdo_id), V(Ncdo_data)); DLSDU = BUILD_PDU_TOKEN(V(Nmst1_add), CONFIG); SRC_Send_Frame.req(Port_sel, DLSDU);	Sync communication start
55	Configuration	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) >= V(Ncnf_index_max))] / DLSDU = BUILD_PDU_TOKEN(V(Nmst1_add), CONFIG); SRC_Send_Frame.req(Port_sel, DLSDU);	Sync communication start
56	Sync communication start	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Sync communication start
57	Sync communication start	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Sync communication start
58	Sync communication start	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (DLSDU.HS = REQ) && (CDO_ID = cyclic_start)] / SET_CDO_DATA(cyclic_start, CDO_data); DLSDU = BUILD_PDU_CDO_WRITE(DLSDU.SA, ACK, CDO_ID, OK); SRC_Send_Frame.req(Port_num, DLSDU); CTC_Start.req;	To cyclic communication

8.3.3.2 Slave Protocol Machine

As for a slave upon network initialization, Figure 22 shows the FSMUL state transition chart and Table 30 shows the FSMUL state transition table.

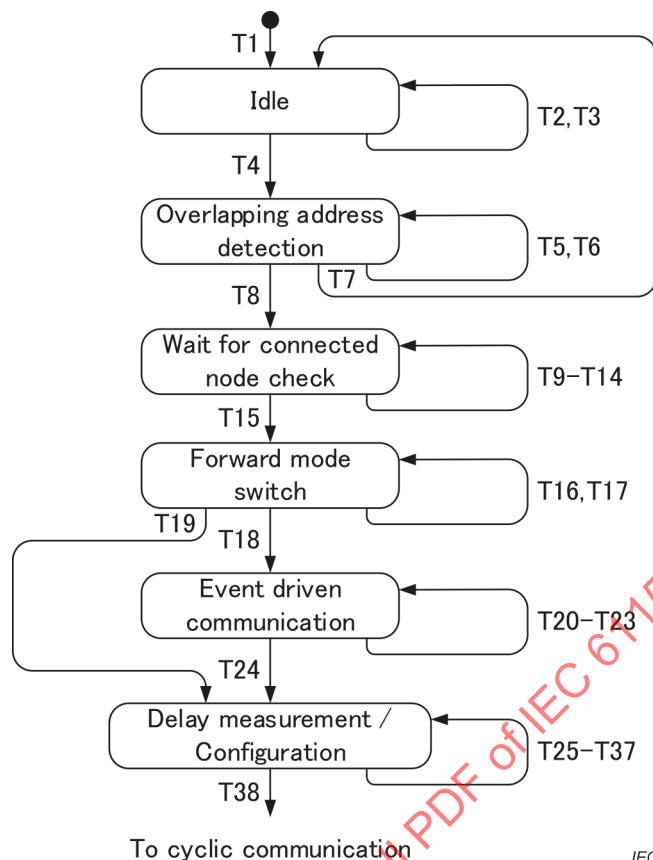


Figure 22 – FSMUL state transition chart for a slave

Table 30 – FSMUL state transition table for a slave

T#	Source State	Event (arguments) [conditions] / Action	Target State
1	All	Power ON or FSMUL-RESET.req / FSMUL-RESET.cnf CTC_Reset.req SRC_Reset.req SET_CDO_DATA(Status, IDLE)	Idle
2	Idle	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Idle
3	Idle	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Idle
4	Idle	FSMUL-START.req / RECEIVE_ALL_PKT(ON); START_TIMER(T(Twait), V(Tdt_same_add)); SET_CDO_DATA(Status, DT_ADD_ERR);	Overlapping address detection
5	Overlapping address detection	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Overlapping address detection
6	Overlapping address detection	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Overlapping address detection
7	Overlapping address detection	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (DLSDU.SA = V(Nmy_add))] / FSMUL-CLR-ERR.cnf(Adress_error);	Idle

T#	Source State	Event (arguments) [conditions] / Action	Target State
8	Overlapping address detection	EXPIRED_TIMER(T(Twait)) = "True" / RECEIVE_ALL_PKT(OFF); SET_CDO_DATA(Status, WRECOG);	Wait for connected node check
9	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_ASYNC_IND = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.EventType = SUP_NOTI)] / SET_CDO_DATA(Mst1_mac_add, DLSDU.SMA); SET_CDO_DATA(Mst1_node_add, DLSDU.SA); SET_CDO_DATA(Mst1_uport, Port_num); DLSDU = BUILD_PDU_ASYNC_IND(DLSDU.SA, SUP_RESP); SRC_Send_Frame.req(Port_num, DLSDU);	Wait for connected node check
10	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none); if(Result == "OK") { GET_CDO_DATA(CDO_ID, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, CDO_ID, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); if(CDO_ID = Status) { DLSDU = BUILD_PDU_CDO_WRITE(V(Nmul_next), REQ, Connect, 1); SRC_Send_Frame.req(Port_sel, DLSDU); } }	Wait for connected node check
11	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if((Result == "OK") && (CDO_ID == Connect)) { SET_CDO_DATA(GET_CDO_PORTID(Port_num), 1); }	Wait for connected node check
12	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if(Result == "OK") { SET_CDO_DATA(CDO_ID, CDO_data); DLSDU = BUILD_PDU_CDO_WRITE(DLSDU.SA, ACK, CDO_ID, OK); SRC_Send_Frame.req(Port_num, DLSDU); if(CDO_ID = Connect) { SET_CDO_DATA(GET_CDO_PORTID(Port_num), 1); }	Wait for connected node check
13	Wait for connected node check	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Wait for connected node check
14	Wait for connected node check	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Wait for connected node check
15	Wait for connected node check	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = Trans_Mode)] / CHANGE_TRANS_MODE(); SET_CDO_DATA(Status, TRANS_MODE);	Forward mode switch
16	Forward mode switch	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Forward mode switch

T#	Source State	Event (arguments) [conditions] / Action	Target State
17	Forward mode switch	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Forward mode switch
18	Forward mode switch	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS= REQ)) && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode) && FSMUL-ASYNC.req] / GET_CDO_DATA(Trans_Mode, CDO_data) DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); SET_CDO_DATA(Status, ASYNC);	Event driven communication
19	Forward mode switch	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS= REQ)) && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode) && FSMUL-WDLYCNFG.req] / GET_CDO_DATA(Trans_Mode, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); SET_CDO_DATA(Status, WDLYONFG);	Delay measurement / Configuration
20	Event driven communication	CTC-SDN.req(SAP_ID, Node_ID, Length, DLSDU) / DLSDU = BUILD_PDU_EVD(Node_ID, Length, SDU); Len = GET_LEN(DLSDU); SRC_Send_Frame.req(Port_sel, Node_ID, Len, DLSDU); CTC-SDN.cnf("OK");	Event driven communication
21	Event driven communication	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Node_ID = GET_DA(DLSDU); Len = GET_LEN(DLSDU); DLSDU = GET_DATA(DLSDU); CTC-SDN.ind(SAP_ID, Node_ID, Len, SDU);	Event driven communication
22	Event driven communication	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Event driven communication
23	Event driven communication	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Event driven communication
24	Event driven communication	FSMUL-MEAS-DELAY.req / V(Ndly_cnt) = 0; SET_CDO_DATA(Status, MEASURED_DLY)	Delay measurement / Configuration
25	Delay measurement / Configuration	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_ASYNC_IND = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.EventType = MEAS_DLY))] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU);	Delay measurement / Configuration
26	Delay measurement / Configuration	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS= REQ)) && (V(Ndly_cnt) < V(Nmax_nodes))] / GET_RECV_TIME(V(Treq_r)); BUILD_PDU_MEAS_DLY(DLSDU.SA, RESP, DLSDU.TimeStampT3, V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); V(Ndly_cnt)++;	Delay measurement / Configuration

T#	Source State	Event (arguments) [conditions] / Action	Target State
27	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS= REQ)) && (V(Ndly_cnt) >= V(Nmax_nodes))] / GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SA, RESP, DLSDU.TimeStampT3, V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); DLSDU = BUILD_PDU_ASYNC_IND(V(Nmst1_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); SET_CDO_DATA(Status, MEAS_DLY_CMP); V(Ndly_cnt) = 0; </pre>	Delay measurement / Configuration
28	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_SYNC = CHECK_PDU_FCMD(DLSDU)] / GET_CDO_DATA(Status, V(Nstatus)); if(V(Nstatus) = MEAS_DLY_CMP) { RESTART_TIMER(Tcycle); } else { GET_SYNC_DATA(DLSDU, V(Tsyn_s)); GET_RECV_TIME(V(Tsyn_r)); } </pre>	Delay measurement / Configuration
29	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_REQ_SEND = CHECK_PDU_FCMD(DLSDU) && (DLSDU.PollingType = MEAS_DLY))] / DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SM_ADD, REQ, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	Delay measurement / Configuration
30	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS= ACK) && (DLSDU.SA = V(Nsm_add)))] / GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = ((Treq_r - Tsyn_s) - (Treq_s - Tsyn_r)) / 2; SET_CDO_DATA(DELAY, V(Ndelay)); SET_CDO_DATA(Status, MEAS_DLY_CMP); </pre>	Delay measurement / Configuration
31	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_REQ_SEND = CHECK_PDU_FCMD(DLSDU) && (DLSDU.PollingType = SYNC))] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	Delay measurement / Configuration
32	Delay measurement / Configuration	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Delay measurement / Configuration
33	Delay measurement / Configuration	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Delay measurement / Configuration
34	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS= REQ))] / Result = GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none); if(Result == "OK") { GET_CDO_DATA(CDO_ID, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, CDO_ID, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU) } </pre>	Delay measurement / Configuration

T#	Source State	Event (arguments) [conditions] / Action	Target State
35	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / (Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS= REQ)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if(Result = "OK") { SET_CDO_DATA(CDO_ID, CDO_data); Result = "OK"; DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Ack, CDO_ID, Result); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	Delay measurement / Configuration
36	Delay measurement / Configuration	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Delay measurement / Configuration
37	Delay measurement / Configuration	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Delay measurement / Configuration
38	Delay measurement / Configuration	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) / (Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (DLSDU.HS= REQ) && (CDO_ID = cyclic_start)] / SET_CDO_DATA(cyclic_start, CDO_data); DLSDU = BUILD_PDU_CDO_WRITE(DLSDU.SA, ACK, CDO_ID, OK); SRC_Send_Frame.req(Port_num, DLSDU); CTC_Start.req; </pre>	To cyclic communication

8.3.3.3 List of FSMUL macros

Table 31 shows the list of macros used for the FSMUL state transitions.

Table 31 – List of FSMUL macros

Function name	Parameter		Return value	Action
	Input	Output		
BUILD_PDU_ASYNC_IND	Node_ID, EventType	None	DLSDU	Create DLSDU which stored the asynchronous notification.
BUILD_PDU_CDO_READ	Node_ID, HS, CDO_ID, Result, CDO_data	None	DLSDU	Create DLSDU to read CDO which is specified by CDO_ID. When H/S is command, data shall not be stored in "Result" "DLOCDO_data".
BUILD_PDU_CDO_WRITE	Node_ID, HS, CDO_ID, CDO_data/Result	None	DLSDU	Create DLSDU to write CDO_Val in CDO which is specified by CDO_ID.
BUILD_PDU_EVD()	Node_ID, Length, SDU	None	DLSDU	Create DLSDU from the specified Node_ID, Length and SDU.
BUILD_PDU_SEND_REQ	Node_ID, SEND_TYP, SM_ADD, SM_MAC_ADD	None	DLSDU	Create a send request which stored SEND_TYP. When SEND_TYP is SYN, values shall not be stored in SM_ADD and SM_MAC_ADD.

Function name	Parameter		Return value	Action
	Input	Output		
BUILD_PDU_MEAS_DLY	Node_ID, HS, TimeStampT3, TimeStampT4	None	DLSDU	Create a delay measurement frame. When H/S is command, values shall not be stored in TimeStampT3 and TimeStampT4. When it is response, the request sending time shall be stored in TimeStampT3 and the request receiving time shall be stored in TimeStampT4.
GET_DA	DLSDU	None	Node_ID	Return DA stored in DLSDU as Node_ID.
GET_LEN	DLSDU	None	Length	Return the data length stored in DLSDU as Length.
GET_DATA	DLSDU	None	SDU	Get SDU from DLSDU.
GET_SYNC_DATA	DLSDU	TimeStamp	none	Obtain the time stamp in the sync notification.
GET_DELAY_DATA	DLSDU	TimeStampT3, TimeStampT4	none	Obtain the time stamp in the delay measurement frame.
GET_RECV_TIME	none	TimeStamp	none	Obtain the packet receiving time immediately before this.
GET_NODE_INFO	AXIS_NUM, IO_Par_ID	None	IO_Par_Val	Return the IO map parameter, IO_Par_ID, of the axis number, AXIS_NUM, as IO_Par_Val.
SET_NODE_INFO	AXIS_NUM, IO_Par_ID, IO_Par_Val	None	none	Store IO_Par_Val in the IO map parameter, IO_Par_ID, of the axis number, AXIS_NUM.
GET_DOMAIN_AXISNUM	Node_ID, Num	None	AXIS_NUM	Return the axis number of the control number, Num, of the node which belongs to the control domain managed by the master with Node_ID. Num shall be from 1 to the number of nodes in the domain. The number of nodes in the domain shall include other masters.
GET_VALUE	Var_ID	CurVal	Result	Store the current value of parameter specified by Var_ID in CurVal and return the result as Result.
SET_VALUE	Var_ID	Val	Result	Store Val in the parameter specified by Var_ID and return the result as Result.
GET_CDO_READ_DATA	DLSDU, Num, CDO_ID, CDO_data	None	Result	Read the n (Num)-th CDO_ID and CDO_data of the stored in CDO read response and return the result as Result.
GET_CDO_WRITE_DATA	DLSDU, Num, CDO_ID, CDO_data	None	Result	Read the n (Num)-th CDO_ID and CDO_data stored in CDO write request and return the result as Result.

Function name	Parameter		Return value	Action
	Input	Output		
GET_CDO_DATA	CDO_ID, CDO_data	None	Result	Store the CDO value specified by CDO_ID in CDO_data.
SET_CDO_DATA	CDO_ID, CDO_data	None	Result	Store the value stored in CDO_data in CDO specified by CDO_ID.
GET_MST2_NODE_ADD	Index	None	Node_ID	Return Node_ID of the n (Index)-th 2nd master.
GET_CONFIG_CDO	Index	CDO_ID, CDO_data, Read/Write, Node_ID	none	Return the n (Index)-th information in the configuration data table.
RECEIVE_ALL_PKT	ON/OFF	None	none	Turn ON/OFF all received settings.
GET_CDO_PORTID	Port_num	None	CDO_ID	Return the ID of port connection information CDO of the number specified by Port_num.
CHANGE_TRANS_MODE	none	None	none	Switch the transfer mode according to the port connection information.

8.3.4 Sync methods

8.3.4.1 Delay measurement

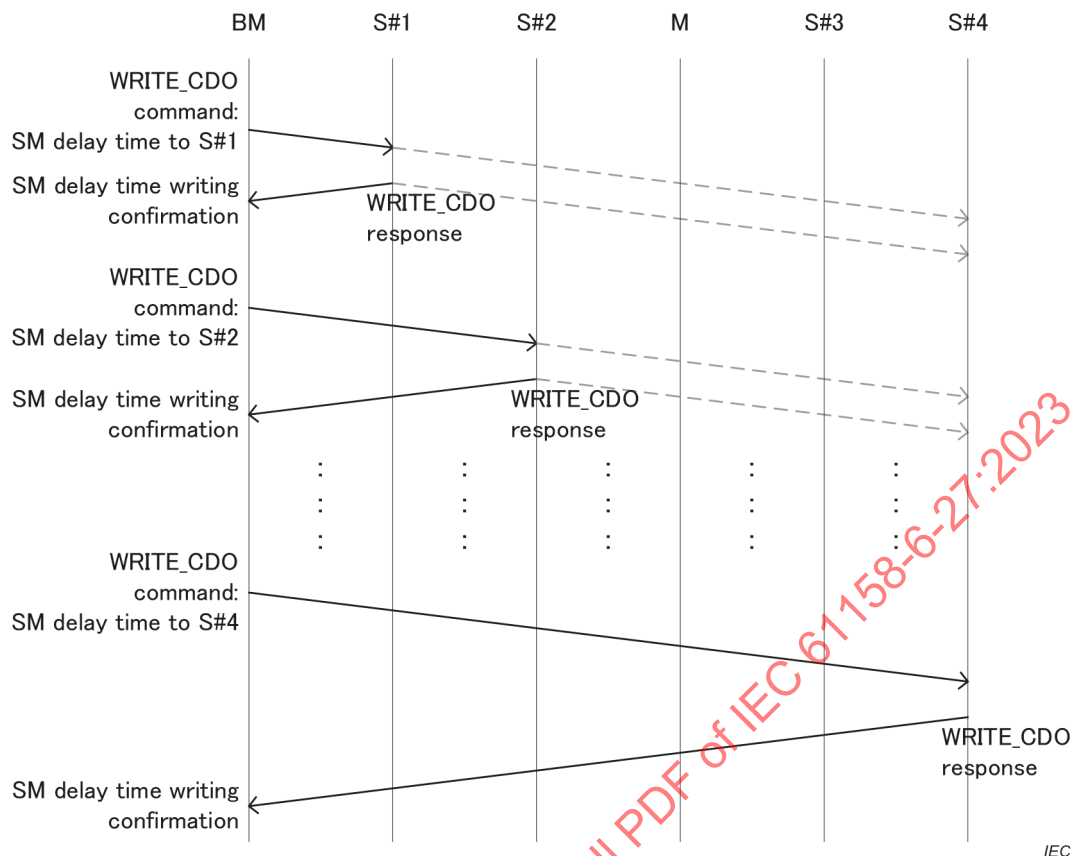
8.3.4.1.1 When the BM is the sync master

8.3.4.1.1.1 Delay measurement of SM

After completing the sequence of either "forward mode switch" or "event driven communication", a delay measurement (SM delay measurement) with the sync master as the reference point shall be executed. In the aforementioned connected node detection (see 8.3.2.2), the BM can acknowledge whether or not each node has the SM delay time. In cases where each node has the SM delay time in advance, the process described in this subclause can be omitted.

- Notification of the SM delay time by WRITE_CDO.

In cases where BM has the SM delay time information of each node, the process of SM delay information notification, as shown in Figure 23, shall be executed by WRITE_CDO. BM sends the SM delay time to each node by WRITE_CDO command. WRITE_CDO command destination is the node object [configuration: transmission delay from the sync master to its own node (offset address: '004C'H)]. After updating CDO, each node returns the WRITE_CDO response. BM confirms that the SM delay time information notification has been completed, by receiving the WRITE_CDO response.



IEC

Figure 23 – Sequence of SM delay time notification by CDO writing

- SM delay measurement

In cases where each node does not have the SM delay time in advance and that the BM does not have the SM delay time information of each node, SM delay measurement shall be executed. The procedures are as describe below according to the number labels in Figure 24.

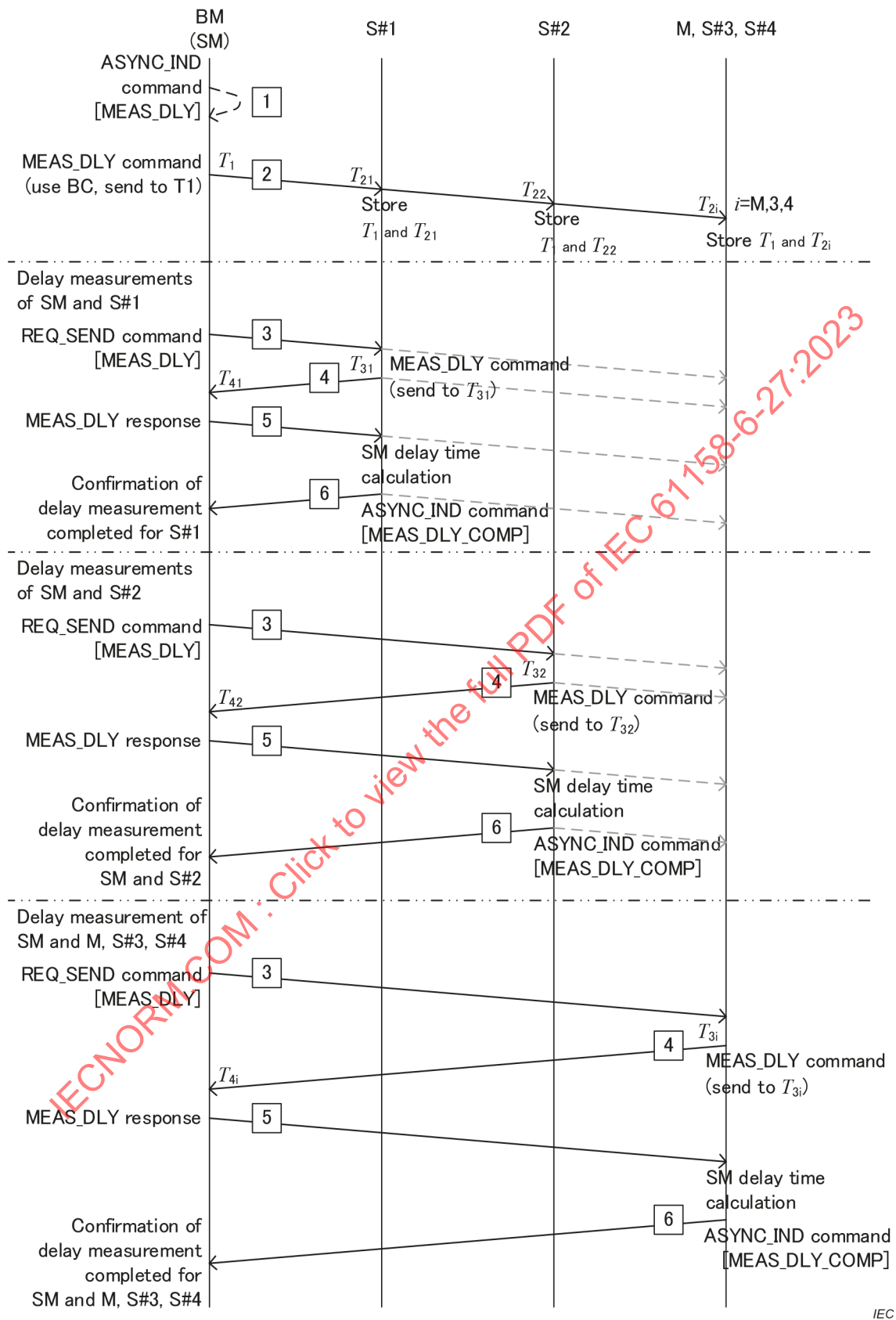
- 1) First, the BM sends the **ASYNC_IND** command [event type: delay measurement] (see 4.3.4.10) to the SM. Actually, as the BM itself is SM in Figure 24, the **ASYNC_IND** command is not sent.
- 2) Next, the BM sends the **MEAS_DLY** command with the destination address which is the broadcast address by using the common multicast (see Table 4). The sending time of BM for this time, T_1 , is stored in the **MEAS_DLY** command. The **MEAS_DLY** command receiving nodes other than the BM saves "the **MEAS_DLY** command received time by the broadcast address: T_{2n} (in Figure 24, this is T_{21} , T_{22} and T_{2i})" of its own node and "the **MEAS_DLY** command frame sending time from the SM: T_1 " stored in the **MEAS_DLY** command.
- 3) After that, the BM sends the **REQ_SEND** command [request type: delay measurement request] (see 4.3.4.5) to the delay measurement target nodes other than the SM. This sending request stores the BM address and the MAC address.
- 4) The delay measurement target nodes which received the **REQ_SEND** command [request type: delay measurement request] send the **MEAS_DLY** command (see 4.3.4.6) to the BM. The **MEAS_DLY** command stores the sending time from the delay measurement target nodes for this time, T_{3n} (in Figure 24, this is T_{31} , T_{32} or T_{3i}).

- 5) The BM which received the MEAS_DLY command sends the receiving time, T_{4n} (in Figure 24, this is T_{41} , T_{42} or T_{4i}), and the MEAS_DLY response which stores the sending time from the delay measurement target nodes stored in the frame, T_{3n} , to the delay measurement target nodes. The delay measurement target nodes which received the MEAS_DLY response calculate the delay time, $V(T_{dly}(n))$, according to the equation described below according to the stored "the MEAS_DLY command frame receiving time from SM: T_{4n} " and "the MEAS_DLY command frame sending time from its own node: T_{3n} " and T_1 and T_{2n} saved in the step 2 above, and retain the value as the parameter of "transmission delay time from the sync master to its own node".

$$V(T_{dly}(n)) = \frac{(T_{4n} - T_1) - (T_{3n} - T_{2n})}{2} \quad (1)$$

- 6) After finishing the calculation of delay time, the delay measurement target nodes send the ASYNC_IND command [event type: delay measurement completed] to the BM. This ASYNC_IND stores the delay time of its own node, $V(T_{dly}(n))$. The BM acknowledges that the delay measurements at the delay measurement target nodes have been completed by receiving the ASYNC_IND command [event type: delay measurement completed], and shall memorize the delay time of delay measurement target nodes, $V(T_{dly}(n))$.
- 7) The steps from 3) to 6) above shall be implemented for all nodes.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-27:2023



IEC

Figure 24 – Delay measurement sequence (SM: BM)

8.3.4.1.1.2 Delay measurement of master

To control the band, the master shall obtain delay time to each node which communicates with its own node. Therefore, the master, in the same way as the SM, shall execute the delay measurement sequence to the nodes in the control domain which communicates with its own node. Note that the BM needs to measure the delay time to the master other than the BM. In the aforementioned connected node detection (see 8.3.2.2), the BM can acknowledge whether the master other than the BM has completed the master delay measurement. In cases where the BM and the master other than the BM have all completed the master delay measurements, the process described in this subclause can be omitted.

- Master delay measurement of the BM.

As the BM is the SM, the delay time already measured by the SM delay measurements shall be obtained by READ_CDO command as the master delay time (Figure 25). The READ_CDO command destination is the node object [configuration: transmission delay from the sync master to its own node (offset address: '004C'H)].

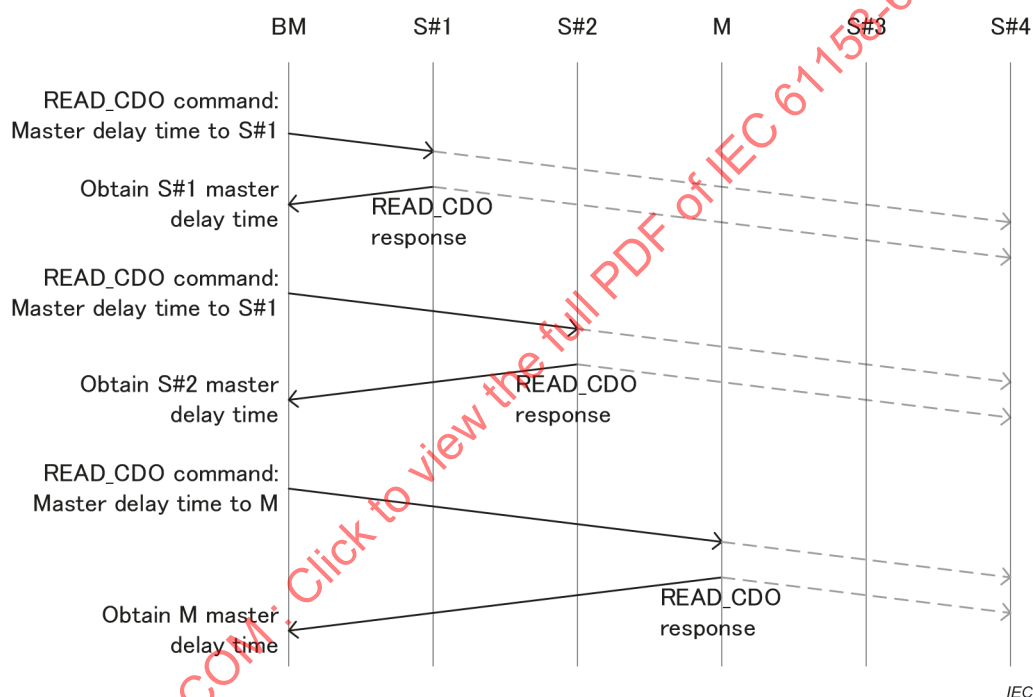


Figure 25 – Master delay measurement sequence for the BM (SM: master with band master function (BM))

- Master delay measurement of the master other than the BM

The master other than the BM shall execute the delay measurement sequence to obtain delay time to each node which communicates with its own node. In cases where multiple masters other than the BM exist, each master shall execute the delay measurement sequence. Figure 26 shows the master delay measurement sequence of the master other than BM. The delay measurements shall be executed according to the sequence as described below.

- 1) The BM sends the TOKEN command [token type: master delay measurement] to the M.
- 2) Upon receiving the TOKEN command [token type: master delay measurement], the M sends the REQ_SEND command [request type: sync notification] to the BM.
- 3) The BM which received the REQ_SEND command [request type: sync notification] sends the sync notification. Each node which received the SYNC command executes the sync correction.

- 4) After receiving the sync notification from the BM, the M sends the MEAS_DLY command to S#*n* (*n* = 3 or 4) which belongs to a control domain. The MEAS_DLY command stores the sending time of M, T_{5n} , for this time.
- 5) The S#*n* which received the MEAS_DLY command sends the time of receiving the MEAS_DLY command, T_{6n} , and the MEAS_DLY response which is store the sending time of the M stored in the frame, T_{5n} , to the M. Also, the port which received the MEAS_DLY command shall be memorized here as the upstream port to the M. The M which received the MEAS_DLY response calculates the delay time, $V(T_{\text{mst\_dly}}(n))$, with the following equation, using the stored values of T_{5n} and T_{6n} , and retains the value as the "master delay time" parameter.

$$V(T_{\text{mst\_dly}}(n)) = T_{6n} - T_{5n} \quad (2)$$

- 6) The M sends the TOKEN command [token type: master delay measurement] to the BM and notifies that the master delay measurement has been completed.

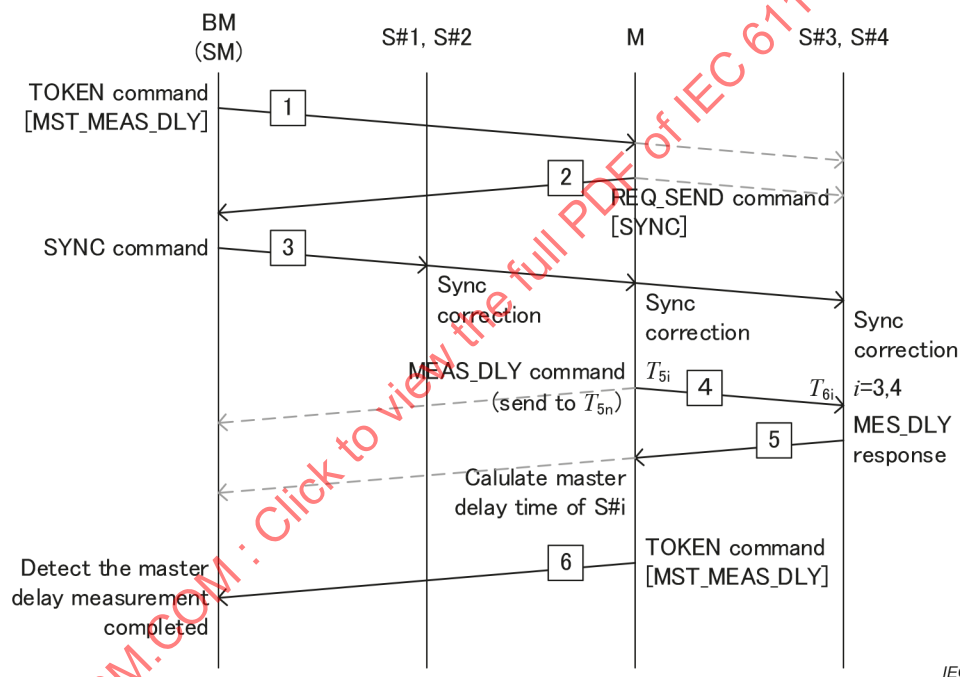


Figure 26 – Delay measurement sequence for the master other than the BM (SM: BM)

8.3.4.1.2 When the master other than the BM is the sync master

8.3.4.1.2.1 Delay measurement of the SM

Even in cases where the master other than the BM is the SM, the process described in this subclause can be omitted in the same way as the SM delay measurement described in 8.3.4.1.1.1.

- Notification of the SM delay time by WRITE_CDO.

See 8.3.4.1.1.1.

- SM delay measurement

In cases where each node does not have the SM delay time in advance and that the BM does not have the SM delay time information of each node, the SM delay measurement shall be executed. Figure 27 shows the sequence. In this case, S#1 is set as the SM. Although a general flow is the same as the case with the BM as the SM, the reference point for delay measurement is S#1 here. The procedures are as described below according to the number labels in Figure 27.

- 1) First, the BM sends the ASYNC_IND command [event type: delay measurement] (see 4.3.4.10) to the S#1 (SM).
- 2) Next, the S#1 sends the MEAS_DLY command with the destination address which is the broadcast address by using the common multicast (see Table 4). The sending time of S#1 for this time, T_1 , is stored in the MEAS_DLY command. The MEAS_DLY command receiving nodes other than the S#1 save "the SYNC command receiving time for the delay measurement request by the broadcast address: T_{2n} (in Figure 27, this is T_{2BM} , T_{22} and T_{2i})" of its own node and "delay measurement request sending time from the SM: T_1 " stored in the MEAS_DLY command.
- 3) After that, the BM sends the REQ_SEND command [request type: delay measurement request] to the delay measurement target nodes other than the SM. This REQ_SEND command stores the S#1 address and the MAC address. In Figure 27, as the BM is not the SM, the destination becomes its own node; however, it is not sent actually.
- 4) The delay measurement target nodes which received REQ_SEND command [request type: delay measurement request] sends the MEAS_DLY command to the S#1. The MEAS_DLY command stores the sending time from the delay measurement target nodes for this time, T_{3n} (in Figure 27, this is T_{3BM} , T_{32} or T_{3i}).
- 5) The S#1 which received the MEAS_DLY command sends the receiving time, T_{4n} (in Figure 27, this is T_{4BM} , T_{42} or T_{4i}), and the MEAS_DLY response which stores the sending time from the delay measurement target nodes stored in the frame, T_{3n} , to the delay measurement target nodes. The delay measurement target nodes that received the MEAS_DLY response calculate the delay time (T_{dly}) according to equation (1) according to the stored "the MEAS_DLY command frame receiving time from SM: T_{4n} " and "the MEAS_DLY command sending time from its own node: T_{3n} " and T_1 and T_{2n} saved in the step 2 above, and retain the value as the parameter of "transmission delay time from the sync master to its own node".
- 6) After finishing the calculation of delay time, the delay measurement target nodes send the ASYNC_IND command [event type: delay measurement completed] to the BM. This ASYNC_IND command stores the delay time of its own node (T_{dly}). The BM acknowledges that the delay measurements at the delay measurement target nodes have been completed by receiving the ASYNC_IND command [event type: delay measurement completed] and shall memorize the delay time of delay measurement target nodes (T_{dly}).
- 7) The steps from 3) to 6) above shall be implemented for all nodes.

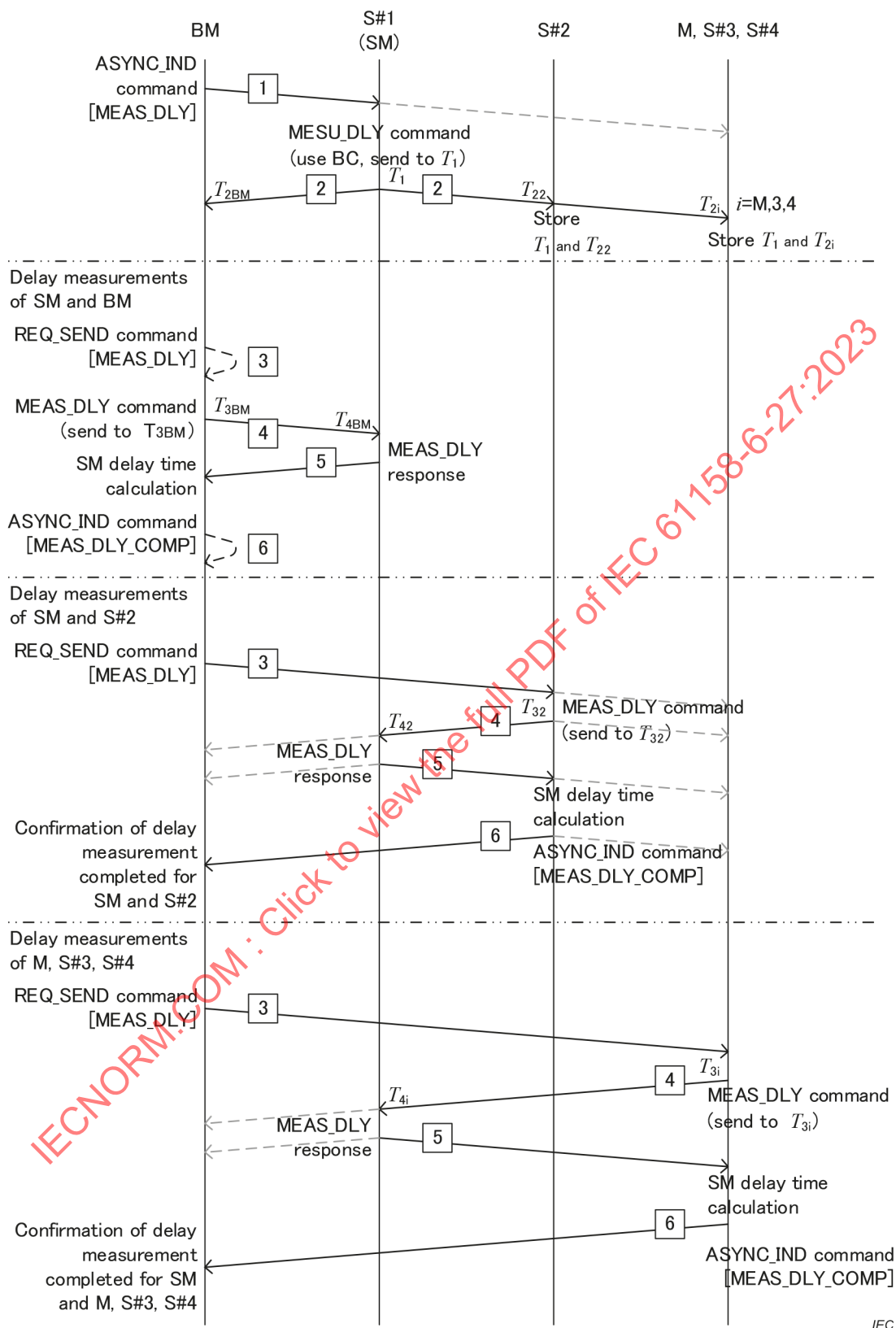


Figure 27 – Delay measurement sequence (SM: S#1)

8.3.4.1.2.2 Master delay measurement

Even in cases where the master other than the BM is the SM, the process described in this subclause can be omitted in the same way as the master delay measurement described in 8.3.4.1.1.2.

- Master delay measurement of the BM.

In cases where the BM is the SM, it is only required to read the already measured delay time; however, in other cases, delay measurements with the BM as the reference point shall be executed as shown in Figure 28.

- 1) First, the BM sends the REQ_SEND command [request type: sync notification] to the S#1 (SM).
- 2) The S#1 which received the REQ_SEND command [request type: sync notification] sends the SYNC command. Each node which received the SYNC command executes the sync correction.
- 3) The BM sends the MEAS_DLY command to the S# m ($m = 1$ or 2) that belongs to its own control domain or to the M. The BM stores the sending time of BM, T_{5n} ($n = 1$ or 2 or M), for this time.
- 4) The S# m or the M which received the MEAS_DLY command sends the MEAS_DLY response which stores the MEAS_DLY command receiving time, T_{6n} , and the BM sending time stored in the frame, T_{5n} , to the BM. The BM which received the MEAS_DLY response calculates the delay time, $V(T_{\text{mst\_dly}}(n))$, according to equation (2) by using the stored T_{5n} and T_{6n} , and retains the values as the "master delay time" parameter.

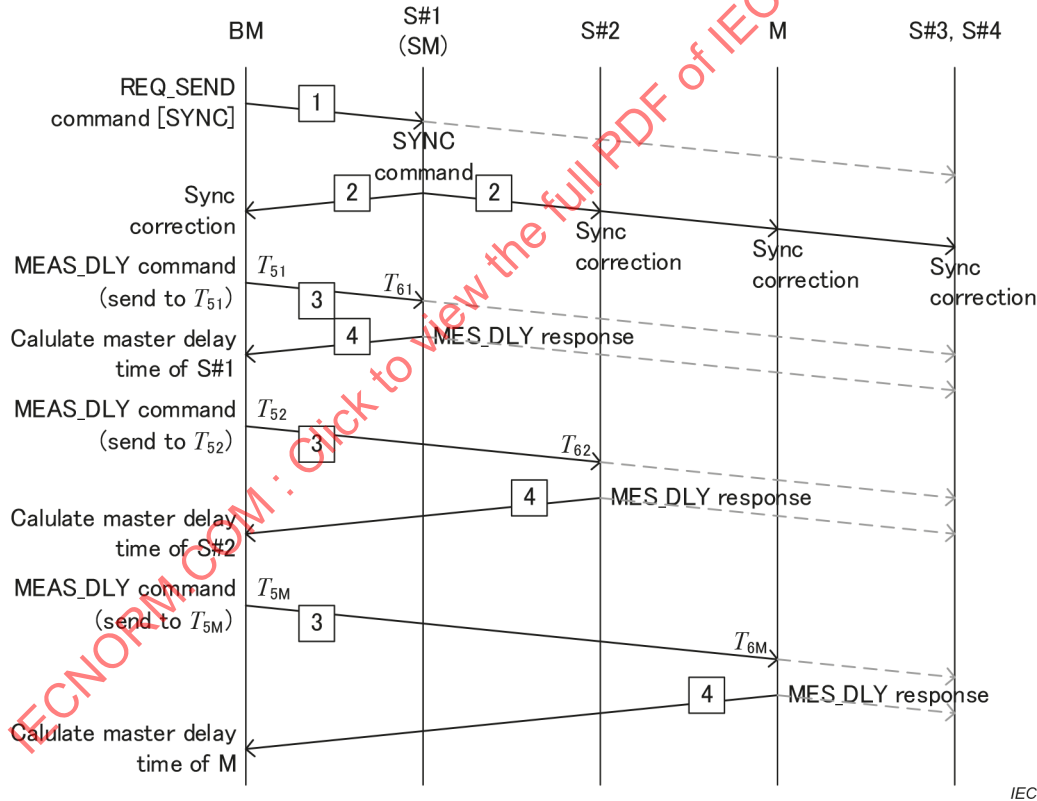


Figure 28 – Master delay measurement sequence for the BM (SM: S#1)

- Master delay measurement for the master other than the BM.

The master other than the BM shall execute the delay measurement sequence to obtain delay time to each node which communicates with its own node in the same way as the master delay measurement (see 8.3.4.1.1.2). The only difference from cases with the BM as the SM is that the destination of sending request [request type: sync notification] is not the BM but the S#1 (SM in this section) in the step 2 as shown in Figure 26.

In cases where the master other than the BM is the SM, the steps 2 and 3 of Figure 26 shall be replaced with the sync notification sending from the M and the steps 4 and 5 shall be the same procedures as those in Figure 25.

8.3.4.2 Sync correction

All nodes connected to the same network shall execute the sync correction by using the time stamp stored in the sync notification which they received. Sync correction updates the local clock by using the difference in the current time between the aforementioned time stamp and the local clock.

8.3.5 Plug-and-play entry

8.3.5.1 Overview

In this protocol, participation of a node in the middle of sync communication (plug-and-play entry) is available. The following conditions shall be satisfied to enable a plug-and-play entry.

- Each node connected to the network that the plug-and-play entry node is entering and the plug-and-play entry node need to already have the parameter set values that assume the participation of plug-and-play entry node;
- A plug-and-play entry node cannot connect to nodes other than the nodes supporting this protocol.

8.3.5.2 Plug-and-play entry sequence

The plug-and-play entry sequence is as described below with the connection configuration in Figure 29. Note that a slave which belongs to the 2nd control domain, S#n, shall be used as the plug-and-play entry node instead of "node not supported by this protocol" in Figure 29.

- 1) Figure 29 shows the sequence from detection of plug-and-play entry node to forward method switching. In cases where the power is turned on after connecting the plug-and-play entry node, S#n, to the end node, S#4, the state shifts to cyclic communication start wait state after receiving the sync packet during the sync notification packet monitoring time. Also, in cases where the power of plug-and-play entry node, S#n, has been turned on and that it is connected to the end node, S#4, after starting sending of the ASYNC_IND command [event type: neighbouring node notification], S#n becomes the cyclic communication start wait state upon receiving of the SYNC command after stopping sending of the ASYNC_IND command [event type: neighbouring node notification].
- 2) Receiving the SYNC command V(Nstop_tx) times, S#n updates the port status of its own node and starts sending the INPUT command after executing the sync correction when it receives the OUTPUT command addressed to its own node. When the plug-and-play entry node is master, updates the port status of its own node and starts sending the OUTPUT command after executing the sync correction.
- 3) Receiving the OUTPUT command or INPUT command from S#n, S#4 updates the port status of the port which received the packet and switches the upstream transfer method from "store and forward" to "cut-through".

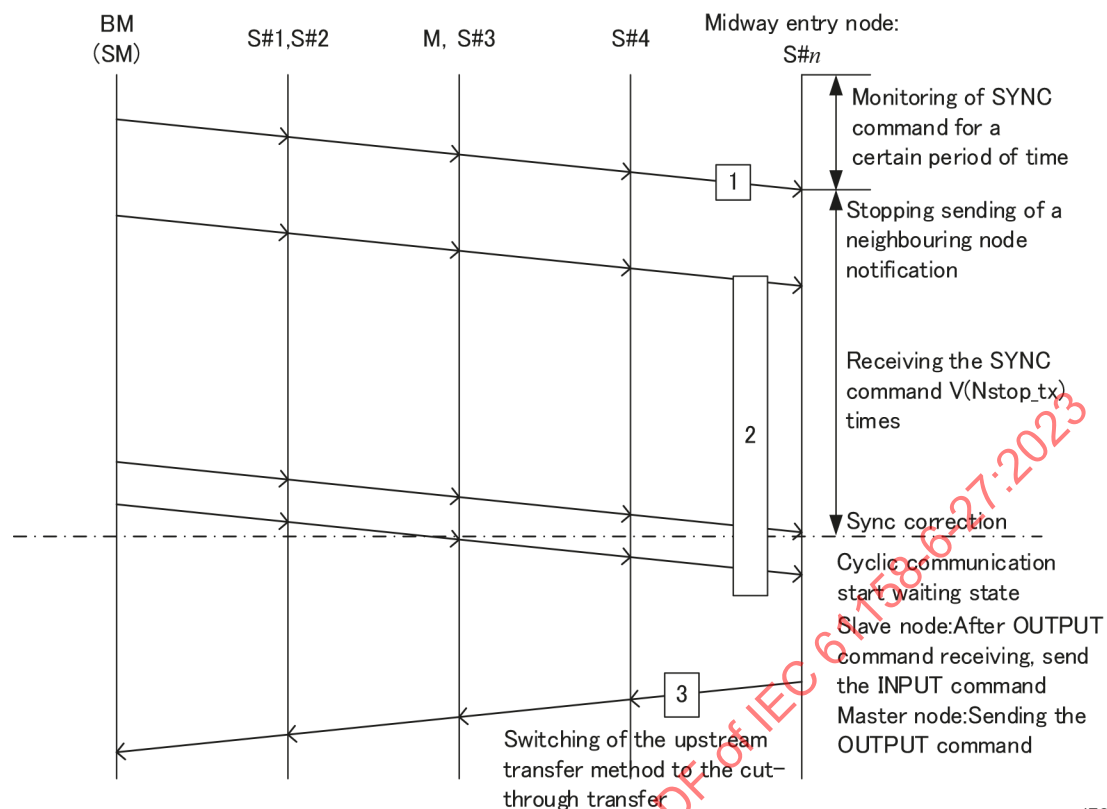


Figure 29 – Plug-and-play entry sequence

8.4 Field Deice Control Protocol Machine (FDC PM)

8.4.1 Protocol overview

The FDC PM is a protocol state machine (PM) that realizes the services provided by the FDC ASE. The protocols are categorized as shown in Table 32.

Table 32 – FDC protocol mode

Transmission mode	Communication state	Communication command type	Description
Cyclic	Synchronous communication (Sync)	Synchronous communication type command (Sync command)	The FDC ASE, both the master and the slave, shall notify firing of an event to the user in each communication cycle with FDC-ComCycle.ind. The user of the FDC master can update a next command to the slave AP in each communication cycle to request to transfer it without waiting the process completion response (FDC-Command.cnf) to the previous command. The FDC slave shall receive a command from the master in each communication cycle and pass it to the user of the FDC slave with FDC-Command.ind. The user should process the command, then return the response (FDC-Command.rsp) within the one communication cycle.
		Asynchronous communication type command (Async command)	The FDC ASE, both the master and the slave, shall notify firing of an event to the user in each communication cycle with FDC-ComCycle.ind. The user of the FDC master should request to transfer a command to the slave AP in each communication cycle while waiting the process completion response to the previous command. The contents of the command should be maintained until the user received the process completion response. The FDC slave shall receive a command from the master AP in each communication cycle and pass it to the user. The user should return a corresponding response (FDC-Command.rsp) with a command progress status (cmdRdy) in each communication cycle and should not accept any new command until the user completes the processing command.
	Asynchronous communication (Async)	Async command	The FDC ASE, both the master and the slave, shall notify firing of an event to the user in each communication cycle with FDC-ComCycle.ind. Once the user of the FDC master requests to transfer a command to the slave AP, the user should wait the corresponding process completion response to the previous command. Until then, the user should not request to transfer any updated command. The FDC slave receives a command from the master AP in each communication cycle. When the contents of the command are updated, the FDC slave shall pass it to the user. The user should return a corresponding response (FDC-Command.rsp) with a command progress status (cmdRdy) in each communication cycle and shall not accept any new command until the user completes the processing command.

Transmission mode	Communication state	Communication command type	Description
Event-driven	Async	Async command	The FDC ASE, both the master and the slave, shall not notify any event to the user in a regular cycle because no communication cycle is generated. Once the user of FDC master requests to transfer a command, the user should wait the corresponding process completion response for the previous command and should not transfer any updated command. The FDC slave shall pass the user the command received from the master AP. The user should not return any response nor accept any new command until processing the received command is completed or the pre-defined processing time elapses.

8.4.2 Cyclic communication mode

8.4.2.1 Cyclic communication common specifications

In the cyclic communication mode, an event shall be generated in a constant period, referred as the communication cycle. The communication process provided by the FDC ASE shall be executed repeatedly with this event. The communication cycle shall originate in the cycle of an integral multiple of the transmission cycle managed by the data link layer.

The transmission cycle should be a constant cycle event generated with the periodic running counter synchronized with all devices.

The communication cycle shall start when the connection has established between the master and the slave. Therefore, before the connection is established, even in the cyclic communication mode, the communication process cannot be synchronized with the communication cycle but the primary cycle or the transmission cycle, and only communication by using certain asynchronous type commands can be executed.

The protocol for the establishment and release of the connection shall be also provided as the FDC ASE services.

The communication cycle in the FDC ASE can have some effects on even the user process, such as C1 master AP or slave AP, see Figure 30, Figure 31.

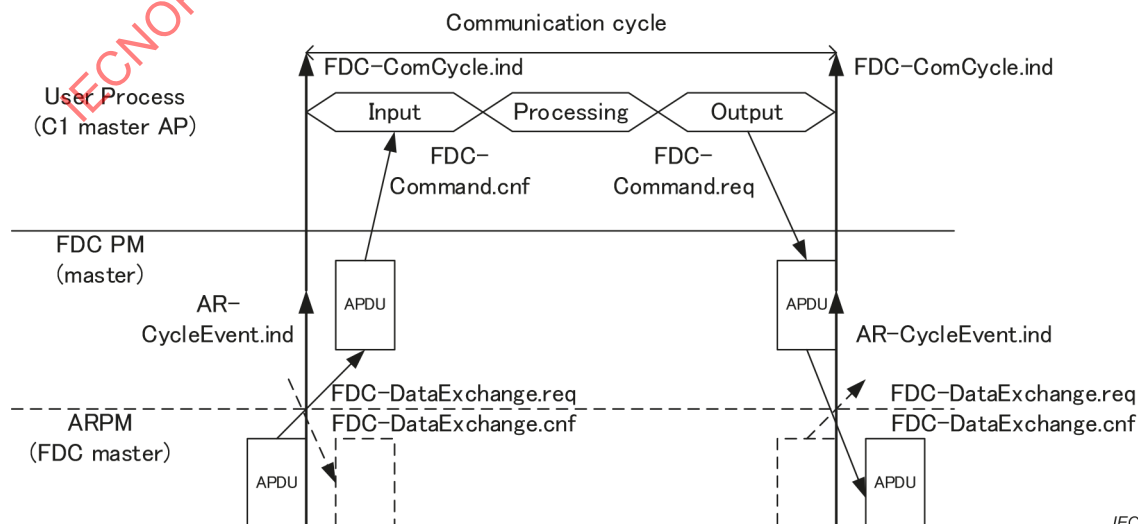


Figure 30 – Example communication cycle of FDC master AP

For example, in the C1 master, the command output timing, the response input timing and the data processing timing is handled most efficiently in the case when they are processed with the cycle event (FDC-ComCycle.ind) as indicated in Figure 31.

NOTE Figure 30 and Figure 31 are shown as just informative examples.

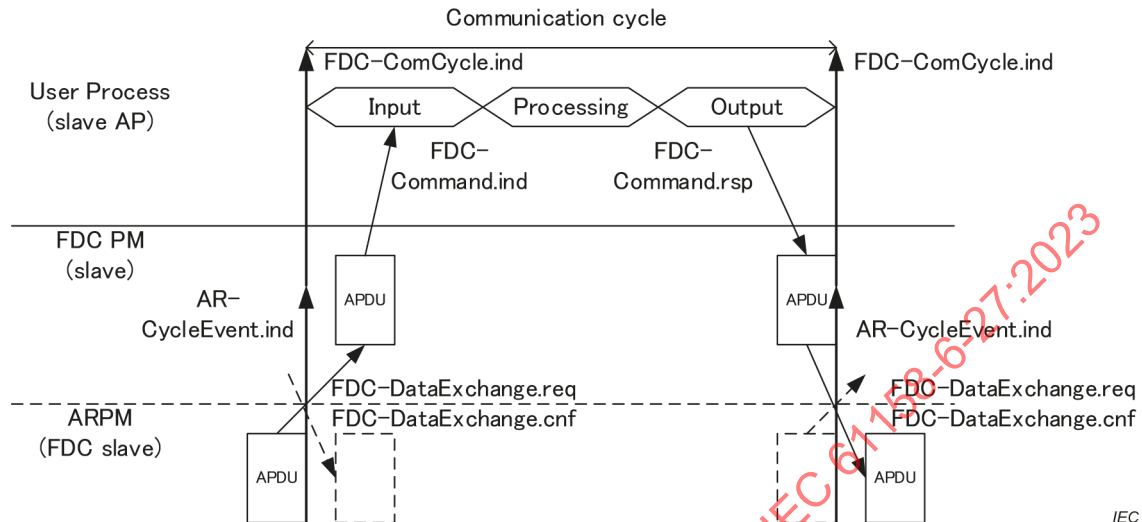


Figure 31 – Example communication cycle of FDC slave AP

In the same way for the slave AP, the command input timing, the response output timing and the data processing timing is handled most efficiently in the case when they are processed as indicated in Figure 31.

8.4.2.2 Synchronous communication state (SyncConnected)

The word "synchronous" within the "synchronous communication state" and "synchronous communication type command" means that also the user processes issuing the commands in the states should be initiated or executed cyclically, as well as the FDC ASE communication process synchronizes with the communication cycle and is executed periodically.

The master AP should request to transfer a command and the slave AP should request to transfer a response respectively once in each communication cycle when the FDCPM is in the synchronous communication state. In this case, the master AP and the slave AP should watch over the status of the synchronization activity each other by using a counter to watch or a _WDT (Watch Dog Timer: WDT) field on each SDU.

When the master AP transfers synchronous type commands continuously for two or more times, it can request to do them in each communication cycle one after another without waiting to receive the corresponding response. The slave AP should complete processing the synchronous command within the communication cycle in which the command is received and transmit the response. The master AP and the slave AP can exchange this type of command and response with the WDT counter to acknowledge that they are synchronized with each other.

Figure 32 shows the timing chart for the synchronous command communication.

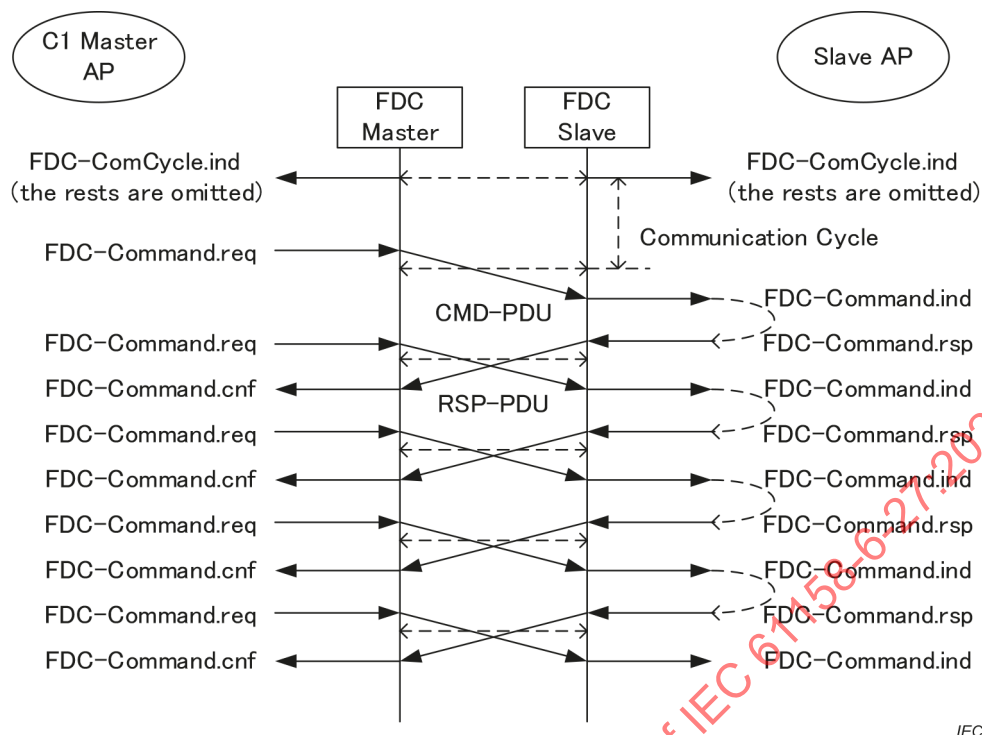


Figure 32 – Synchronous command communication in sync state

When the master AP transfers an asynchronous type command, the master AP should request to transfer the next command after it has confirmed the corresponding response to the previous command indicating the process completes by a cmdRdy-bit = 1 or "command ready".

However, even in that case, the master AP should continuously request to transfer the command with the same contents in every communication cycle. And the slave AP should request to transfer the response with a cmdRdy-bit = 0 or "command busy", so that each watching of the synchronized status through the WDT succeeds.

NOTE The names such as "synchronous type command" and "asynchronous type command" represent whether the user process is executed and completed while being synchronized with the communication cycle or not. On the other hand, from the aspect of the transaction management for the command and the response, it can be said that the synchronous type command processes asynchronous transactions, and the asynchronous type command processes synchronous transactions.

Figure 33 shows the timing chart for the asynchronous command communication in the synchronous state.

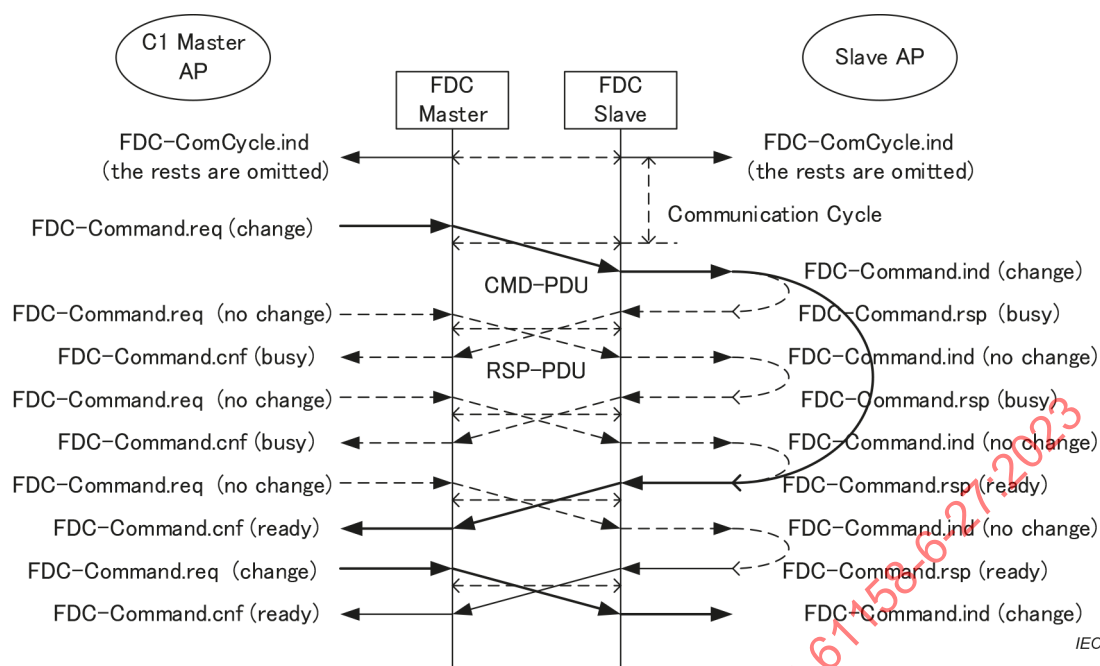


Figure 33 – Asynchronous command communication in sync state

8.4.2.3 Asynchronous communication state (AsyncConnected)

Even under the asynchronous transmission state, the communication process of the FDC ASE shall be periodically initiated synchronous with the communication cycle, because it is in the cyclic communication mode. In this case, the master AP and the slave AP should not use a WDT (Watch Dog Timer; WDT) field on each SDU to watch over the synchronization activity.

Therefore, constant periodic event with FDC-ComCycle.ind can be notified to the user process in each communication cycle. However, the user process need not always be operated synchronous with the event.

In this state, the FDC ASE shall provide the user process with only asynchronous type commands. Therefore, the master AP can transfer a new command only after it confirms the corresponding process completion response to the transferring command.

Figure 34 shows the timing chart in the asynchronous communication state.

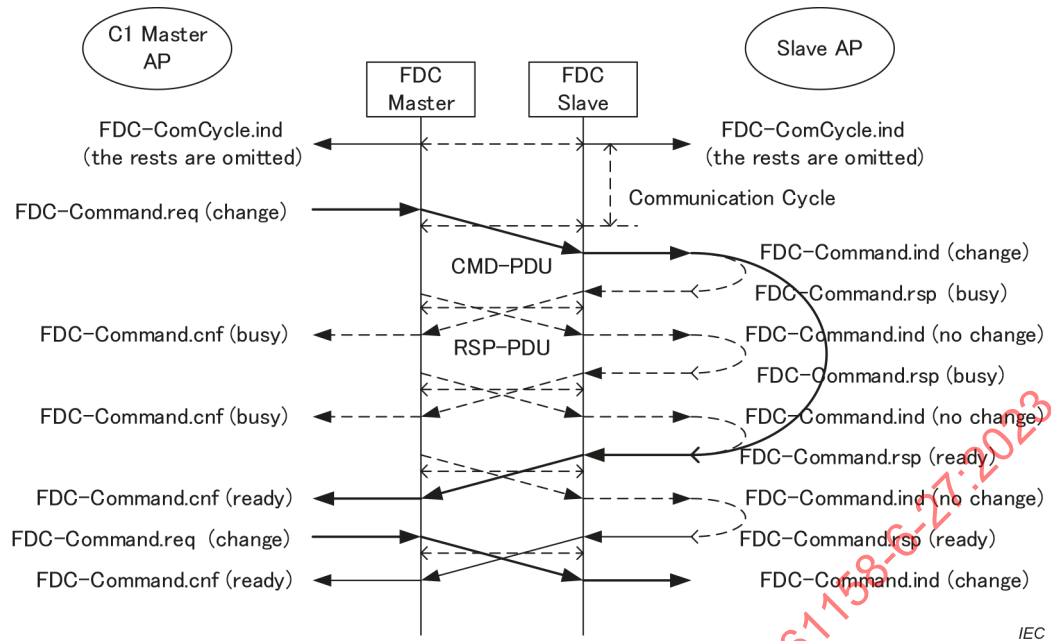


Figure 34 – Asynchronous command communication in async state

8.4.3 Event driven communication mode

In the event driven communication mode, the FDC ASE can have no communication cycle and the communication process shall not be executed cyclically. The network clock cannot function either.

In this mode, the FDC ASE provides no service related to the synchronous type command and the user process can only use asynchronous type commands. And the master AP can request to transfer a command non-periodically whenever the preceding command transaction has already completed.

Figure 35 shows the timing chart in the event driven mode communication.

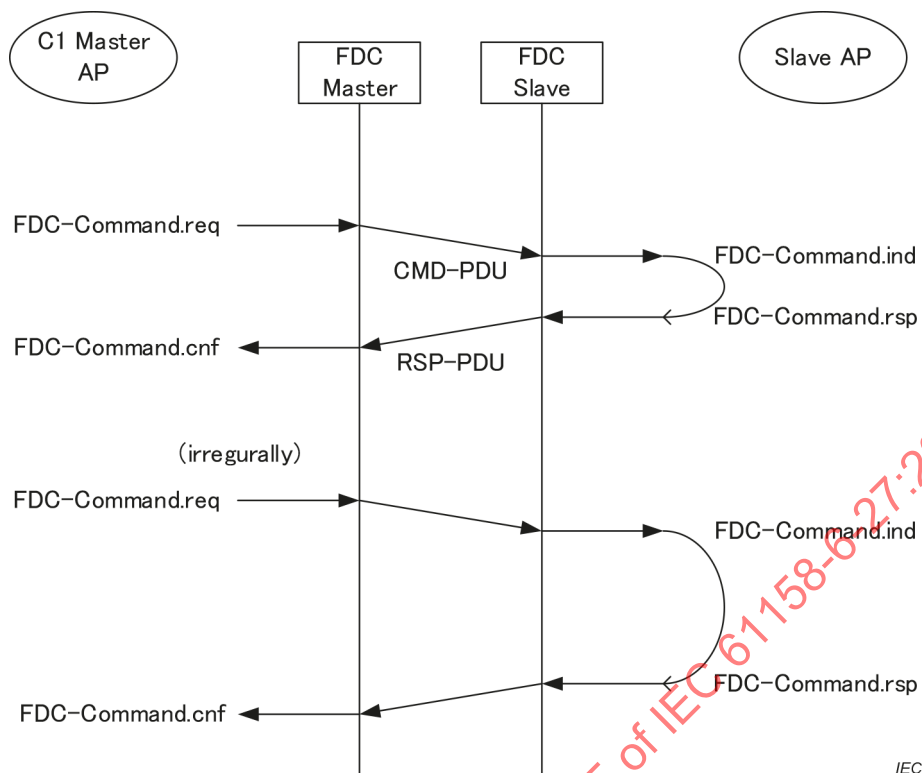


Figure 35 – Event-driven communication

8.4.4 Master Protocol Machine (FDCPM-M)

8.4.4.1 State descriptions

Figure 36 shows the FDCPM-M state chart diagram, and Table 33 describes each state of the FDCPM-M.

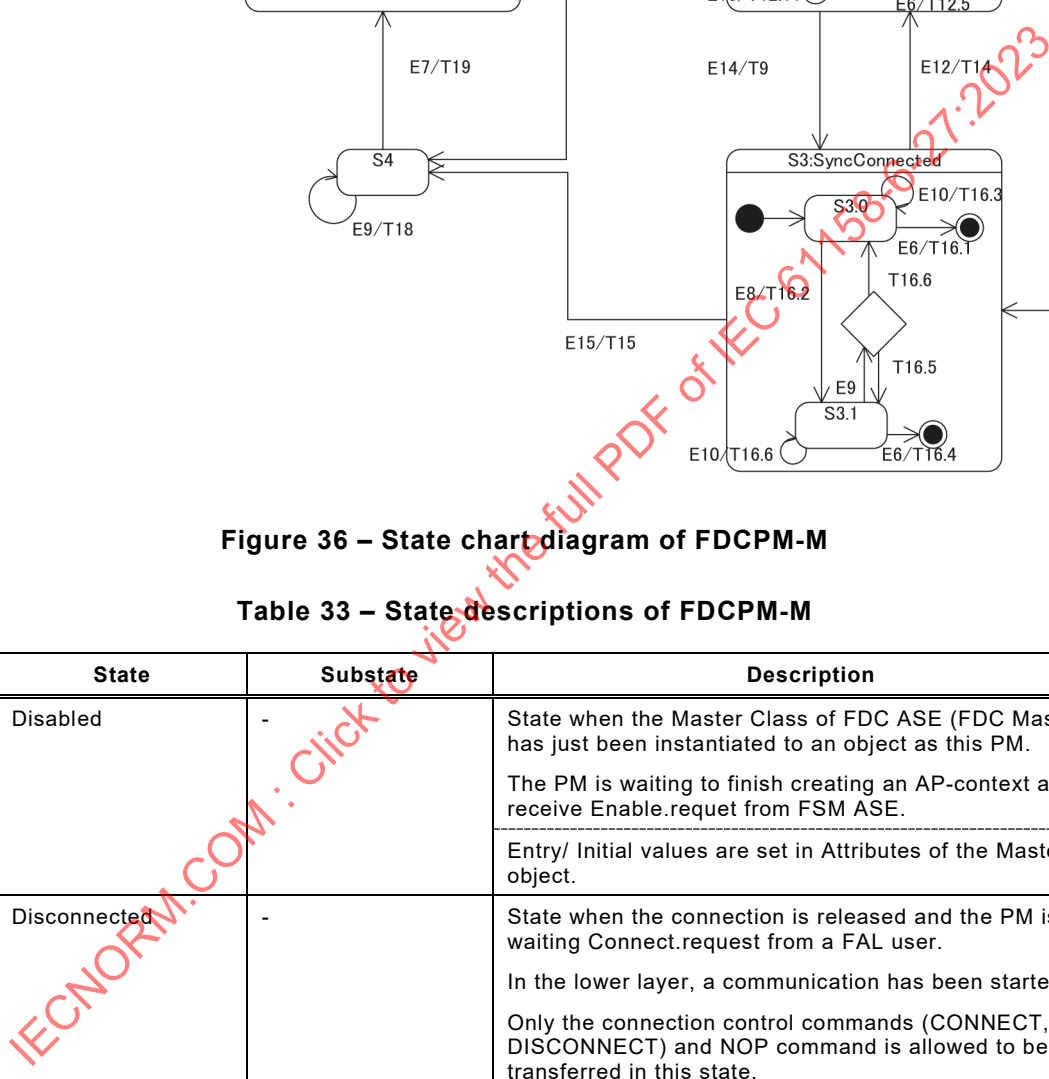


Table 33 – State descriptions of FDCPM-M

S#	State	Substate	Description
S0	Disabled	-	State when the Master Class of FDC ASE (FDC Master) has just been instantiated to an object as this PM. The PM is waiting to finish creating an AP-context and to receive Enable.request from FSM ASE. Entry/ Initial values are set in Attributes of the Master object.
S1	Disconnected	-	State when the connection is released and the PM is waiting Connect.request from a FAL user. In the lower layer, a communication has been started. Only the connection control commands (CONNECT, DISCONNECT) and NOP command is allowed to be transferred in this state. Even if the FSMUL is running in the cyclic communication mode, the communication cycle has not been notified in this state because no connection is established.
S1.0		Idle	Substate when the PM is waiting a request to transfer a CMD-PDU from a FAL user
S1.1		WaitCMDComplete	Substate when the PM is waiting a command completion response from the correspondent peer Slave AP

S#	State	Substate	Description
S2	AsyncConnected	-	A normal operation state to control a Slave AP or a function of a field device through the peer FDC Slave by using asynchronous type command. A connection with the FDC Slave is established and a communication cycle event is notified by AR ASE. That event is transferred as FDC-ComCycle.ind to the C1 master AP. Any asynchronous type command can be transferred in this state. No synchronous type command is allowed to be transferred.
S2.0		Idle	Substate when the PM is waiting a request to transfer a CMD-PDU from a FAL user
S2.1		WaitCMDComplete	Substate when the PM is waiting a command completion response from the correspondent peer Slave AP
S3	SyncConnected	-	The most time-critical operation state to control a Slave AP or a function of a field device through the peer FDC slave by using both synchronous and asynchronous commands. A connection with the FDC Slave has been established and a communication cycle event is notified by AR ASE. That event is transferred as FDC-ComCycle.ind to the C1 master AP. The WDT counter on every CMD-PDUs shall function to watch over the status of the synchronization activity of the C1 master AP. And so as to the RWDT counter on RSP-PDUs for the Slave AP.
S3.0		Idle	Substate when the PM is waiting a request to transfer a CMD-PDU from a FAL user
S3.1		WaitCMDComplete	Substate when the PM is waiting a command completion response from the correspondent peer Slave AP
S4	Disconnecting	-	State when the connection changes to be released because communication errors continuously occur or the FAL user demands it. The PM waits in order to prevent the system malfunctioning as a kind of interlocking mechanism, and then recovers to S1 state when receiving a resume request from the user (ResumeCycle.req).

8.4.4.2 Triggering events

Table 34 lists each trigger event of the FDCPM-M.

Table 34 – Trigger event descriptions of FDCPM-M

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	FDC_Reset.req	FSM ASE		
E2	FDC-Open.req	FSM ASE	AREPID	
E3	FDC-Enable.req	FSM ASE	TransmissionMode	
E4	FDC-Connect.req	FAL user (C1masterAP)	Update, CONNECT-CMD-SDU	
E5	FDC-SyncSet.req	FAL user (C1masterAP)	Update, SYNC_SET-CMD-SDU	
E6	FDC-Disconnect.req	FAL user (C1masterAP)	DISCONNECT-CMD-SDU	

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E7	FDC-ResumeCycle.req	FAL user (C1masterAP)		
E8	FDC-Command.req	FAL user (C1masterAP)	Update, CMD-SDU	
E9	FDC-DataExchange.req	AR ASE (FDCM-AR)	RSP-SDU	
E10	AR-CycleEvent.ind	AR ASE (FDCM-AR)	NetworkClock	
E11	AR-SendCommand.cnf	AR ASE (FDCM-AR)	ServiceStatus, RSP-SDU	
E12	Error detected	This object		See 8.4.6 and footnotes of Table 35. WDT failures are detected twice serially under SyncConnect state. Alternatively, failures are detected continuously under AsyncConnect state.
E13	EventConnectRSP	This object (submachine)	status Unsigned16 rsdu _CONNECT-RSP-PDU	Notification from submachine
E14	EventSyncSetRSP	This object (submachine)	status Unsigned16 rsdu _SYNC_SET-RSP-PDU	Notification from submachine
E15	EventDisconnectCMD	This object (submachine)	csdu _DISCONNECT-CMD-PDU	Notification from submachine

8.4.4.3 Action descriptions at state transitions

Table 35 describes state transitions of the main SM of FDCPM-M. Moreover,

Table 36 describes state transitions of the submachine of FDCM-M.

Table 35 – Transitions of main SM of FDCPM-M

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1:FDC_Reset.req / /*-- Clearing all attributes of the FDC master --*/;	S0:Disabled
T2	S0:Disabled	E2:FDC-Open.req / /*-- Initializing the FDC master --*/; ^a	S0:Disabled
T3	S0:Disabled	E3:FDC-Enable.req(communicationMode) /* Notifying the FSMUL's communication mode, initialized into whether the cyclic mode or the event-driven mode. */ / TransMode = communicationMode;	S1:Disconnected
T4	S1:Disconnected	E1:FDC_Reset.req / /*-- Clearing all attributes of FDC master --*/;	S0:Disabled

T#	Source State	Event (arguments) [conditions] / Action	Target State
T5	S1:Disconnected	E13:EventConnectRSP(status, rsdu) [rsdu.rspBody.syncmode != 1] / AR-StartComCycle.req; ProtocolVersion = rsdu.rspBody.ver; SyncMode = 0; DTMode = rsdu.rspBody.dtmode; SubCMDMode = rsdu.rspBody.subcmd; ComTime =rsdu.rspBody.com_time; DevProfileType = rsdu.rspBody.profile_type;	S2:AsyncConnected
T6	S1:Disconnected	E13:EventConnectRSP(status, rsdu) [rsdu.rspBody.syncmode == 1] / AR-StartComCycle.req; ProtocolVersion = rsdu.rspBody.ver; SyncMode = 1; DTMode = rsdu.rspBody.dtmode; SubCMDMode = rsdu.rspBody.subcmd; ComTime = rsdu.rspBody.com_time; DevProfileType = rsdu.rspBody.profile_type;	S3:SyncConnected
T7	S1:Disconnected	The other events / /*-- state-transition in the submachine --*/;	S1:Disconnected
T8	S2:AsyncConnected	E1:FDC_Reset.req / /*-- Clearing all attributes of FDC master --*/;	S0:Disabled
T9	S2:AsyncConnected	E14:EventSyncSetRSP(status, rsdu) / WDT.LastMN = 0; WDT.LastSN = 0; RWDT.LastRMN = 0; RWDT.LastRSN = 0;	S3:SyncConnected
T10	S2:AsyncConnected	E15:EventDisconnectCMD(csdu) / /*-- no-operation --*/;	S4:Disconnecting
T11	S2:AsyncConnected	E12: Errors occur continuously ^{a, b, c} / /*-- no-operation --*/;	S4:Disconnecting
T12	S2:AsyncConnected	The other events / /*-- state-transition in the submachine --*/;	S2:AsyncConnected
T13	S3:SyncConnected	E1:FDC_Reset.req / /*-- Clearing all attributes of FDC master --*/	S1:Disabled
T14	S3:SyncConnected	E12: Errors occur serially twice. ^b / /*-- no-operation --*/;	S2:AsyncConnected
T15	S3:SyncConnected	E15:EventDisconnectCMD(csdu) / /*-- no-operation --*/;	S4:Disconnecting
T16	S3:SyncConnected	The other events / /*-- state-transition in the submachine --*/;	S3:SyncConnected
T17	S4:Disconnecting	E1:FDC_Reset.req / /*-- Clearing all attributes of the FDC master --*/;	S1:Disabled

T#	Source State	Event (arguments) [conditions] / Action	Target State
T18	S4:Disconnecting	E9:FDC-DataExchange.req(rsdu) / csdu.cmd = disconnect; FDC-DataExchange.cnf(csdu);	S4:Disconnecting
T19	S4:Disconnecting	E7:FDC-ResumeCycle.req / /*-- Clearing internal information related to the communication cycle and alarm information --*/;	S2:Disconnected
^a The detailed process depends on the system implementation. ^b Implementers need some mechanisms to count or manage errors. ^c The threshold of error duration count is an implementation matter.			

Table 36 – Transitions of submachine of FDCPM-M

T#	Source State	Event (arguments) [conditions] / Action	Target State
T7.1	S1.0:Idle	E4:FDC-Connect.req(csdu) / LastCMD-SDU = csdu; If(TransMode == EventDriven) { AR-SendCommand.req(csdu); } else { /*-- no-operation --*/; }	S1.1: WaitCMDCom plete
T7.2	S1.0:Idle		S1.0: Idle
T7.3	S1.0:Idle		S1.0:Idle
T7.4	S1.1: WaitCMDComplete		S1.1: WaitCMDCom plete
T7.5	S1.1: WaitCMDComplete		Exit to the main machine
T7.6	S1.1: WaitCMDComplete	E6:FDC-Disconnect.req(csdu) / LastCMD-SDU = csdu; If(TransMode == EventDriven) { AR-SendCommand.req(csdu); } else { /*-- no-operation --*/; }	S1.1: WaitCMDCom plete
T7.7	S1.1: WaitCMDComplete	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf(csdu);	Exit to the main machine
T12.1	S2.0:Idle	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/ && ((rsdu.rcmd != LastCMD- SDU.cmd) (status.cmdRdy != 1))] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf(csdu);	S2.1: WaitCMDCom plete

T#	Source State	Event (arguments) [conditions] / Action	Target State
T12.2	S2.0:Idle	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1) && (LastCMD-SDU.cmd == connect)] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf(csdu); FDC-Connect.cnf(status, rsdu); /*-- signal E13:EventConnectRSP to the main machine --*/;	Exit to the main machine
T12.3	S2.0:Idle	E8:FDC-Command.req(csdu) / LastCMD-SDU = csdu; If(TransMode == EventDriven) { AR-SendCommand.req(csdu); } else { /*-- no-operation --*/; }	S2.1:WaitCMDComplete
T12.4	S2.0:Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind	S2.0: Idle
T12.5	S2.1:WaitCMDComplete	E6:FDC-Disconnect.req(csdu) / LastCMD-SDU = csdu; If(TransMode == EventDriven) { AR-SendCommand.req(csdu); } else { /*-- no-operation --*/; } /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit to the main machine
T12.6	S2.1:WaitCMDComplete	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy != 1)] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf(csdu);	S2.1:WaitCMDComplete
T12.7	S2.1:WaitCMDComplete	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1) && (LastCMD-SDU.cmd == sync_set)] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf(csdu); FDC-SyncSet.cnf(+); /*-- signal E14:EventSyncSetRSP to the main machine --*/;	Exit to the main machine
T12.8	S2.1:WaitCMDComplete	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1) && (LastCMD-SDU != sync_set)] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf(csdu); FDC-Commmand.cnf(+);	S2.0:Idle

T#	Source State	Event (arguments) [conditions] / Action	Target State
T12.9	S2.1: WaitCMDComplete	E11:AR-SendCommand.cnf(status, rsdu) [/*-- status has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy != 1)] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; If(TransMode == EventDriven) { AR-SendCommand.req(csdu); } else { /*-- no-operation --*/; }	S2.1: WaitCMDComplete
T12.10	S2.1: WaitCMDComplete	E11:AR-SendCommand.cnf(status, rsdu) [/*-- status has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1)] / LastRSP-SDU = rsdu; FDC-Command.cnf(+);	S2.0:Idle
T12.11	S2.1: WaitCMDComplete	E10: AR-CycleEvent.ind / FDC-ComCycle.ind	S2.1: WaitCMDComplete
T16.1	S3.0:Idle	E6:FDC-Disconnect.req(csdu) / LastCMD-SDU = csdu; /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit to the main machine
T16.2	S3.0:Idle	E8:FDC-Command.req(csdu) / LastCMD-SDU = csdu;	S3.1: WaitCMDComplete
T16.3	S3.0:Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind	S3.0: Idle
T16.4	S3.1: WaitCMDComplete	E6:FDC-Disconnect.req(csdu) / LastCMD-SDU = csdu; /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit to the main machine
T16.5	S3.1: WaitCMDComplete	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy != 1)] / LastRSP-SDU = rsdu; if(rsdu.rwdt.rsn != LastRSN) { /*-- signal E12 watchdog counter error to the main machine--*/; } else { LastRSN = rsdu.rwdt.rsn; csdu = LastCMD-SDU; LastSN = LastRSN; csdu.wdt.mn = LastMN; csdu.wdt.sn = LastSN; FDC-DataExchange.cnf(csdu); }	S3.1: WaitCMDComplete or Exit to the main machine, when E12 has been issued.

T#	Source State	Event (arguments) [conditions] / Action	Target State
T16.6	S3.1: WaitCMDComplete	E9:FDC-DataExchange.req(rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == CMD-SDU.cmd) && (status.cmdRdy == 1)] / LastRSP-SDU = rsdu; if(rsdu.rwdt.rsn != LastRSN) { /*-- signal E12 watchdog counter error to the main machine --*/; } else { LastRSN = rsdu.rwdt.rsn; csdu = LastCMD-SDU; LastSN = LastRSN; csdu.wdt.mn = LastMN; csdu.wdt.sn = LastSN; FDC-DataExchange.cnf(csdu); FDC-Command.cnf(status, rsdu); }	S3.0:Idle or Exit to the main machine, when E12 has been issued.
T16.7	S3.1: WaitCMDComplete	E10: AR-CycleEvent.ind / FDC-ComCycle.ind LastMN++; LastRSN++;	S3.1: WaitCMDComplete

8.4.5 Slave Protocol Machine (FDCPM-S)

8.4.5.1 State descriptions

Figure 37 shows the FDCPM-S state chart diagram, and Table 37 describes each state of the FDCPM-S.

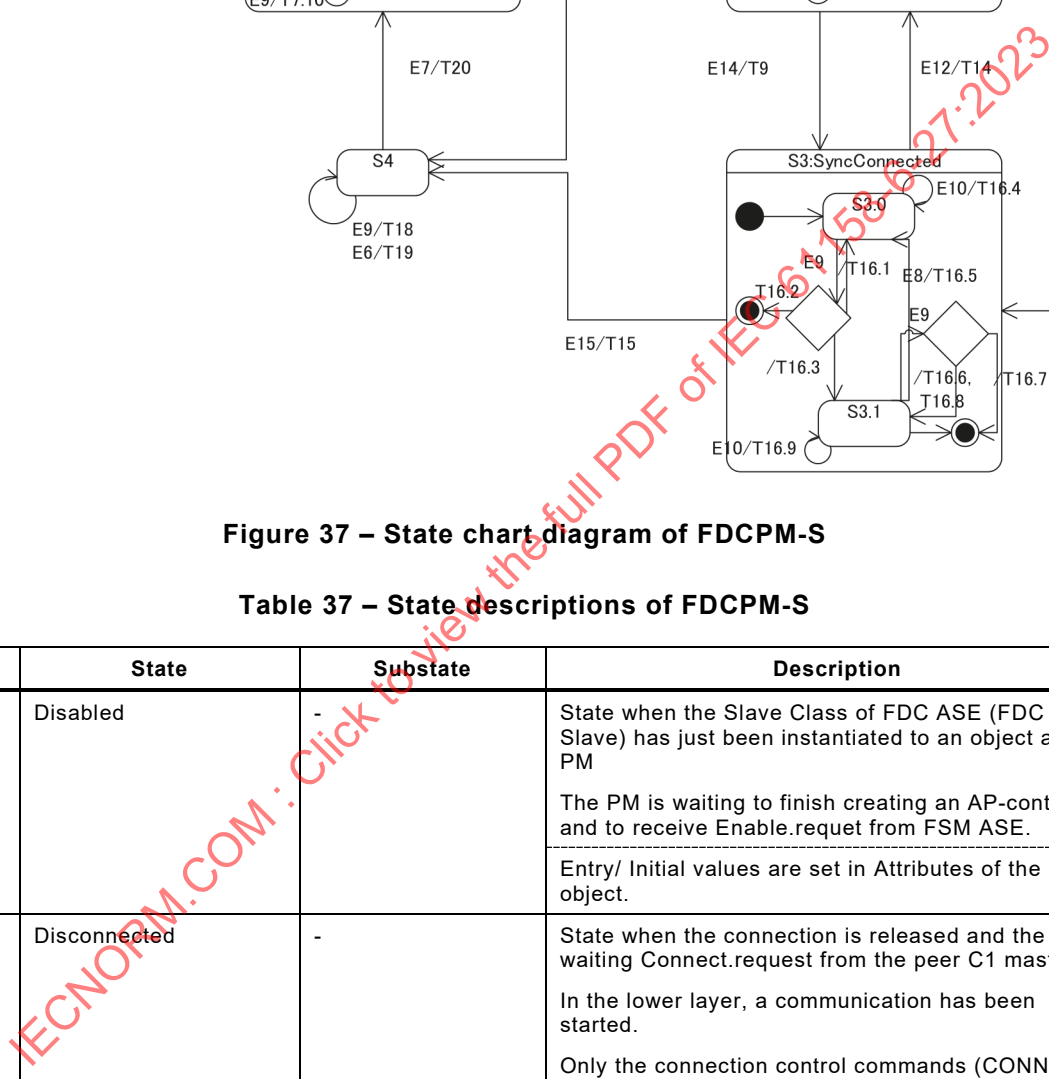


Table 37 – State descriptions of FDCPM-S

S#	State	Substate	Description
S0	Disabled	-	State when the Slave Class of FDC ASE (FDC Slave) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from FSM ASE. Entry/ Initial values are set in Attributes of the Slave object.
S1	Disconnected	-	State when the connection is released and the PM is waiting Connect.request from the peer C1 master AP In the lower layer, a communication has been started. Only the connection control commands (CONNECT, DISCONNECT) and NOP command is allowed to be transferred in this state. Even if the FSMUL is running in the cyclic communication mode, the communication cycle has not been notified in this state because no connection is established.
S1.0		Idle	Substate when the PM is waiting to receive a CMD-PDU from a peer C1 master AP
S1.1		WaitRSPComplete	Substate when the PM is waiting for the corresponding response to transfer from a FAL user or the Slave AP.

S#	State	Substate	Description
S2	AsyncConnected		A normal operation state to transfer the controlling command from the C1 Master AP to the Slave AP and execute a function of a field device with the asynchronous type command A connection with the FDC Master is established and a communication cycle event is notified by AR ASE. Any asynchronous type command can be transferred in this state. No synchronous type command is allowed to be transferred.
S2.0		Idle	Substate when the PM is waiting to receive a CMD-PDU from a peer C1 master AP
S2.1		WaitRSPComplete	Substate when the PM is waiting for the corresponding response to transfer from a FAL user or the Slave AP.
S3	SyncConnected	-	The most time-critical operation state to transfer the controlling command from the C1 Master AP to the Slave AP and execute a function of a field device with both synchronous and asynchronous commands A connection with the FDC Master has been established and a communication cycle event is notified by AR ASE. That event is transferred as FDC-ComCycle.ind to the Slave AP. The WDT counter on every CMD-PDUs shall function to watch over the status of the synchronization activity of the C1 master AP. And so as to the RWDT counter on RSP-PDUs for the Slave AP.
S3.0		Idle	Substate when the PM is waiting to receive a CMD-PDU from a peer C1 master AP
S3.1		WaitRSPComplete	Substate when the PM is waiting for the corresponding response to transfer from a FAL user or the Slave AP
S4	Disconnecting	-	State when the connection changes to be released because communication errors continuously occur or the FAL user demands it. The PM waits in order to prevent the system malfunctioning as a kind of interlocking mechanism, and then recovers to S1 state when receiving a resume request from the user (ResumeCycle.req).

8.4.5.2 Triggering events

Table 38 lists each trigger event of the FDCPM-S.

Table 38 – Trigger event descriptions of FDCPM-S

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	FDC_Reset.req	FSM ASE		
E2	FDC-Open.req	FSM ASE	AREPID	
E3	FDC-Enable.req	FSM ASE	TransmissionMode	
E4	FDC-Connect.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	
E5	FDC-SyncSet.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	
E6	FDC-Disconnect.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E7	FDC-ResumeCycle.req	FAL user (Slave AP)		
E8	FDC-Command.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	
E9	FDC-DataExchange.req	AR ASE (FDCS-AR)	CMD-SDU	
E10	AR-CycleEvent.ind	AR ASE (FDCS-AR)	NetworkClock	
E11	AR-SendCommand.ind	AR ASE (FDCS-AR)	CMD-SDU	
E12	Error detected	This object		See 8.4.6. WDT failures are detected twice serially under SyncConnect state. Alternatively, failures are detected continuously under AsyncConnect state.
E13	EventConnectRSP	This object (submachine)	status Unsigned16 rsdu _CONNECT-RSP-PDU	Notification from submachine
E14	EventSyncSetRSP	This object (submachine)	status Unsigned16 rsdu _SYNC_SET-RSP-PDU	Notification from submachine
E15	EventDisconnectCMD	This object (submachine)	csdu _DISCONNECT-CMD-PDU	Notification from submachine

8.4.5.3 Action descriptions at state transitions

Table 39 describes state transitions of the main SM of FDCPM-S. Moreover,

Table 40 describes state transitions of the submachine of FDCPM-S.

Table 39 – Transitions of main SM of FDCPM-S

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1:FDC_Reset.req / /*-- Clearing all attributes of the FDC slave --*/;	S0:Disabled
T2	S0:Disabled	E2:FDC-Open.req / /*-- Initializing the FDC slave --*/; ^a	S0:Disabled
T3	S0:Disabled	E3:FDC-Enable.req(communicationMode) / TransMode = communicationMode;	S1:Disconnected
T4	S1:Disconnected	E1:FDC_Reset.req / /*-- Clearing all attributes of FDC master --*/;	S0:Disabled

T#	Source State	Event (arguments) [conditions] / Action	Target State
T5	S1:Disconnected	E13:EventConnectRSP(status, rsdu) [rsdu.rspBody.syncmode != 1] / AR-StartComCycle.req; ProtocolVersion = rsdu.rspBody.ver; SyncMode = 0; DTMode = rsdu.rspBody.dtmode; SubCMDMode = rsdu.rspBody.subcmd; ComTime = rsdu.rspBody.com_time; DevProfileType = rsdu.rspBody.profile_type;	S2:AsyncConnected
T6	S1:Disconnected	E13:EventConnectRSP(status, rsdu) [rsdu.rspBody.syncmode == 1] / AR-StartComCycle.req; ProtocolVersion = rsdu.rspBody.ver; SyncMode = 1; DTMode = rsdu.rspBody.dtmode; SubCMDMode = rsdu.rspBody.subcmd; ComTime = rsdu.rspBody.com_time; DevProfileType = rsdu.rspBody.profile_type;	S3:SyncConnected
T7	S1:Disconnected	The other events / /*-- state-transition in the submachine --*/;	S1:Disconnected
T8	S2:AsyncConnected	E1:FDC_Reset.req / /*-- Clearing all attributes of FDC slave --*/;	S0:Disabled
T9	S2:AsyncConnected	E14:EventSyncSetRSP(status, rsdu) / WDT.LastMN = 0; WDT.LastSN = 0; RWDI.LastRMN = 0; RWDI.LastRSN = 0;	S3:SyncConnected
T10	S2:AsyncConnected	E15:EventDisconnectCMD(csdu) / /*-- no-operation --*/;	S4:Disconnecting
T11	S2:AsyncConnected	E12: Errors occur continuously ^{b, c} / /*-- no-operation --*/;	S4:Disconnecting
T12	S2:AsyncConnected	The other events / /*-- state-transition in the submachine --*/;	S2:AsyncConnected
T13	S3:SyncConnected	E1:FDC_Reset.req / /*-- Clearing all attributes of FDC master --*/;	S1:Disabled
T14	S3:SyncConnected	E12: Errors occur serially twice ¹⁾ / /*-- no-operation --*/;	S2:AsyncConnected
T15	S3:SyncConnected	E15:EventDisconnectCMD(csdu) / /*-- no-operation --*/;	S4:Disconnecting
T16	S3:SyncConnected	The other events / /*-- state-transition in the submachine --*/;	S3:SyncConnected
T17	S4:Disconnecting	E1:FDC_Reset.req / /*-- Clearing all attributes of the FDC slave --*/;	S1:Disabled

T#	Source State	Event (arguments) [conditions] / Action	Target State
T18	S4:Disconnecting	E9:FDC-DataExchange.req(csdu) / rsdu.cmd = disconnect; FDC-DataExchange.cnf(rsdu);	S4:Disconnecting
T19	S4:Disconnecting	E6:FDC-Disconnect.rsp(rsdu) / LastRSP-SDU = rsdu; If(TransMode == EventDriven) { AR-SendCommand.rsp(rsdu); } else { /*-- no-operation --*/; }	S4:Disconnecting
T20	S4:Disconnecting	E7:FDC-ResumeCycle.req / /*-- Clearing internal information related to the communication cycle and alarm information --*/;	S2:Disconnected
^a The detailed process depends on the system implementation. ^b Implementers need some mechanisms to count or manage errors. ^c The threshold of error duration count is an implementation matter.			

Table 40 – Transitions of submachine of FDCPM-S

T#	Source State	Event (arguments) [conditions] / Action	Target State
T7.1	S1.0:Idle	E9:FDC-DataExchnage.req(csdu) [csdu == LastCMD-SDU] / rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu);	S1.0: Idle
T7.2	S1.0:Idle	E9:FDC-DataExchnage.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == connect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Connect.ind(csdu);	S1.1: WaitRSPComplete
T7.3	S1.0:Idle	E9:FDC-DataExchnage.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Disconnect.ind(csdu);	S1.1: WaitRSPComplete
T7.4	S1.0:Idle	E9:FDC-DataExchange.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == nop)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Command.ind(csdu);	S1.1: WaitRSPComplete

T#	Source State	Event (arguments) [conditions] / Action	Target State
T7.5	S1.0:Idle	E11:AR-SendCommand.ind(status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == connect)] / LastCMD-SDU = csdu; FDC-Connect.ind(csdu);	S1.1: WaitRSPComplete
T7.6	S1.0:Idle	E11:AR-SendCommand.ind(status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; FDC-Disconnect.ind(csdu);	S1.1: WaitRSPComplete
T7.7	S1.0:Idle	E11:AR-SendCommand.ind(status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == nop)] / LastCMD-SDU = csdu; FDC-command.ind(csdu);	S1.1: WaitRSPComplete
T7.8	S1.1: WaitRSPComplete	E4: FDC-Connect.rsp(rsdu) / LastRSP-SDU = rsdu; If(TransMode == EventDriven) { AR-SendCommand.rsp(rsdu); /*-- signal E13:EventConnectRSP to the main machine --*/; } else { /*-- no-operation --*/; }	Exit from the submachine
T7.9	S1.1: WaitRSPComplete	E6:FDC-Disconnect.rsp(rsdu) / LastRSP-SDU = rsdu; If(TransMode == EventDriven) { AR-SendCommand.rsp(rsdu); } else /*-- no-operation --*/;	S1.0: Idle
T7.10	S1.1: WaitRSPComplete	E8:FDC-Command.rsp(rsdu) / LastRSP-SDU = rsdu; If(TransMode == EventDriven) { AR-SendCommand.rsp(rsdu); } else { /*-- no-operation --*/; }	S1.0: Idle
T7.11	S1.1: WaitRSPComplete	E9: FDC-DataExchnage.req(csdu) / rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu);	S1.1: WaitRSPComplete
T12.1	S2.0: Idle	E9: FDC-DataExchange.req(csdu) [csdu == LastCMD-SDU] / rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu);	S2.0: Idle
T12.2	S2.0:Idle	E9:FDC-DataExchnage.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == sync_set)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-SyncSet.ind(csdu);	S2.1: WaitRSPComplete

T#	Source State	Event (arguments) [conditions] / Action	Target State
T12.3	S2.0:Idle	E9:FDC-DataExchnage.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Disconnect.ind(csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit from the submachine
T12.4	S2.0: Idle	E9: FDC-DataExchange.req(csdu) [(csdu != LastCMD-SDU) && ((csdu.cmd != sync_set) (csdu.cmd != disconnect))] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Command.ind(csdu);	S2.1: WaitRSPComplete
T12.5	S2.0:Idle	E11:AR-SendCommand.ind(status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; FDC-Disonnect.ind(csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit from the submachine
T12.6	S2.0:Idle	E11:AR-SendCommand.ind(status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)] / LastCMD-SDU = csdu; FDC-Command.ind(csdu);	S2.1: WaitRSPComplete
T12.7	S2.0:Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind;	S2.0:Idle
T12.8	S2.1: WaitRspComplete	E5: FDC-SyncSet.rsp(rsdu) / LastRSP-SDU = rsdu; /*-- signal E14:EventSyncSetRSP to the main machine --*/;	Exit from the submachine
T12.9	S2.1: WaitRspComplete	E8: FDC-Command.rsp(rsdu) / LastRSP-SDU = rsdu; If(TransMode == EventDriven) { AR-SendCommand.rsp(rsdu); } else /*-- no-operation --*/;	S2.0: Idle
T12.10	S2.1: WaitRSPComplete	E9: FDC-DataExchange.req(csdu) [csdu == LastCMD-SDU] / rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu);	S2.1: WaitRSPComplete

T#	Source State	Event (arguments) [conditions] / Action	Target State
T12.11	S2.1: WaitRSPComplete	E9:FDC-DataExchnage.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Disconnect.ind(csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit from the submachine
T12.12	S2.1: WaitRSPComplete	E9: FDC-DataExchange.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Command.ind(csdu);	S2.1: WaitRSPComplete
T12.13	S2.1: WaitRSPComplete	E11:AR-SendCommand.ind(status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; FDC-Disonnect.ind(csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit from the submachine
T12.14	S2.1: WaitRSPComplete	E10: AR-CycleEvent.ind / FDC-ComCycle.ind;	S2.1: WaitRSPComplete
T16.1	S3.0: Idle	E9: FDC-DataExchange.req(csdu) [csdu == LastCMD-SDU] / LastCMD-SDU = csdu; if(csdu.wdt.mn != LastMN) { /*-- signal E12 watchdog counter error to the main machine --*/; } else { LastMN = csdu.wdt.mn; rsdu = LastRSP-SDU;LastRMN = LastMN; rsdu.rwdt.rmn = LastRMN; rsdu.rwdt.rsn = LastRSN; FDC-DataExchange.cnf(rsdu); }	S3.0: Idle or Exit to the main machine, when E12 has been issued.
T16.2	S3.0: Idle	E9:FDC-DataExchnage.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Disconnect.ind(csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit from the submachine

T#	Source State	Event (arguments) [conditions] / Action	Target State
T16.3	S3.0: Idle	E9: FDC-DataExchange.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)] / LastCMD-SDU = csdu; if(csdu.wdt.mn != LastMN) { /*-- signal E12 watchdog counter error to the main machine --*/; } else { LastMN = csdu.wdt.mn; rsdu = LastRSP-SDU; LastRMN = LastMN; rsdu.rwdt.rmn = LastRMN; rsdu.rwdt.rsn = LastRSN; FDC-DataExchange.cnf(rsdu); FDC-Command.ind(csdu); }	S3.1: WaitRSPComplete or Exit to the main machine, when E12 has been issued.
T16.4	S3.0: Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind;	S3.0: Idle
T16.5	S3.1: WaitRspComplete	E8: FDC-Command.rsp(rsdu) / LastRSP-SDU = rsdu;	S3.0: Idle
T16.6	S3.1: WaitRSPComplete	E9: FDC-DataExchange.req(csdu) [csdu == LastCMD-SDU] / if(csdu.wdt.mn != LastMN) { /*-- signal E12 watchdog counter error to the main machine --*/; } else { LastMN = csdu.wdt.mn; rsdu = LastRSP-SDU; LastRMN = LastMN; rsdu.rwdt.rmn = LastRMN; rsdu.rwdt.rsn = LastRSN; FDC-DataExchange.cnf(rsdu); }	S3.1: WaitRSPComplete or Exit to the main machine, when E12 has been issued.
T16.7	S3.1: WaitRSPComplete	E9: FDC-DataExchnage.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf(rsdu); FDC-Disconnect.ind(csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit from the submachine

T#	Source State	Event (arguments) [conditions] / Action	Target State
T16.8	S3.1: WaitRSPComplete	E9: FDC-DataExchange.req(csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)] / LastCMD-SDU = csdu; if(csdu.wdt.mn != LastMN) { /*-- signal E12 watchdog counter error to the main machine --*/; } else { LastMN = csdu.wdt.mn; rsdu = LastRSP-SDU; LastRMN = LastMN; rsdu.rwdt.rmn = LastRMN; rsdu.rwdt.rsn = LastRSN; FDC-DataExchange.cnf(rsdu); FDC-Command.ind(csdu); }	S3.1: WaitRSPComplete or Exit to the main machine, when E12 has been issued.
T16.9	S3.1: WaitRSPComplete	E10: AR-CycleEvent.ind / FDC-ComCycle.ind; /*-- counting up watchdog counter --*/; LastMN++; LastRSN++;	S3.1: WaitRSPComplete

8.4.6 Error procedure summary

8.4.6.1 Error detecting overview

The following shows the errors detected in the FDC service. When an error is detected, if any command is executed, both master and slave shall stop it and transition into a state where it can accept a new command to execute the resume process performed by the AP.

Only the definition of the errors is described in this document. The primary error detection processing or mechanism depends on the implementation, and so do the alarm notifying method and the resume processing from the error. They are out of scope of this document.

The following errors are defined:

- communication error: receipt status such as no DLPDU receipt and FCS error detected in the lower layer;
- watchdog counter error: abnormal value of the watchdog counter of the partner in synchronous communication state, which is detected in the mutual monitoring by the master and the slave;
- illegal cycle timing: abnormal value (beyond the jitter allowance) in the transmission cycle detected by the slave;
- response timeout: response time-out detected by the master;
- illegal sync command: synchronization command reception in the state other than the synchronous communication, which is detected by the slave user.

8.4.6.2 Communication error

FDC master and slave can receive a notification in FDC-DataExchange request from AR ASE when FCS error or no-reception error for the reception DLPDU is detected by the lower layer.

If the PM is under the synchronous communication state when the error is detected, it shall make a state transition to the asynchronous communication state.

In case of detection by an FDC master PM, if it has a pending command and waiting the response, it shall discard the command to abort the waiting state.

In case of detection by an FDC slave PM, if it has a processing command, it shall discard the command and the just sending response as well, if it has. And the RSP-PDU shall be edited in the way how the cmd_stat field or the status field should show the error as one of alarm factors.

8.4.6.3 Watchdog counter error

The watchdog counters shall be a mechanism in the synchronous communication state, for one FDC ASE to detect the peer ASE's out of sync with the communication cycle. Once the ASE detects the illegal counter value in a received FDCService-PDU, it shall alert the error to the FDC user through the FSM ASE.

When either of an FDC master PM or an FDC slave PM detects the error serially twice, it shall make the state transition to an asynchronous communication state, not to accept synchronous commands after that.

In case of detection by an FDC master PM, if it has a pending command and waiting the response, it shall discard the command to abort the waiting state.

In case of detection by an FDC slave PM, if it has a processing command, it shall discard the command and the just sending response as well, if it has. And the RSP-PDU shall be edited in the way how the cmd_stat field or the status field should show the error as one of alarm factors.

8.4.6.4 Illegal cycle timing

The "illegal cycle timing" is a kind of error that shows that an FDC slave detects an abnormal cycle jitter by measuring the transmission cycle period. At every transmission cycle event, the FDC slave shall measure the elapsed time since the previous event. And if the measured value is different from the pre-defined parameter for the transmission cycle, the slave shall alert the error to the FDC user through the FSM ASE.

When an FDC slave PM detects the error in the synchronous state, it shall make the state transition to an asynchronous communication state, not to accept synchronous commands after that. And the RSP-PDU shall be edited in the way how the cmd_stat field or the status field should show the error as one of alarm factors.

If the slave has a processing command, it shall discard the command and the just sending response as well, if it has.

8.4.6.5 Response timeout

The "response timeout" is a kind of error that shows that an FDC master detects no response from a FDC slave. After sending a command, the FDC master measures the elapsed time since a falling edge of cmdRdy bit in receiving each RSP-PDUs until the bit change to on ('1'B), which means completion of the command processing of the slave. And if the time exceeds the pre-defined time limit, the master shall alert the error to the FDC user through the FSM ASE.

When the FDC master PM detects the error, it shall discard the command and abort the waiting state. For more details of the recovery process depends on the implementation.

8.4.6.6 Illegal Sync command

The "illegal sync command" is a kind of error that shows that an FDC slave receives a synchronous command in the wrong state other than the synchronous communication state.

When the FDC slave PM detects the error, it shall discard the command and execute alternative command pre-defined in implementation specifications for the device. And the RSP-PDU shall be edited in the way how the cmd field in the CMD-PDU echoes back with rcmd field in the RSP-PDU and the cmd_stat field or the status field should show the error as one of alarm factors.

8.4.6.7 Alarm through RSP-PDU.CMD_STAT

The error described above means an abnormal condition or malfunction for communication activities, whereas other troubles on a remote device should be informed to the FAL user, too.

Those statuses can be transferred as an alarm or a warning through the status fields in the short PDU type responses or the cmd_stat field in the enhanced PDU type responses. The status changes have no effect on the PM directly.

The alarm means a status to tell that the field device has detected a fatal problem to be solved and cannot continue normal working. And the warning means a status to tell that the device has detected a slight or passing problem but still working normally.

8.5 Message Protocol Machine (MSG PM)

8.5.1 Protocol overview

8.5.1.1 General

In the Message service, two types of protocols are defined in the Type 27 FAL. One is a user message service that transfers message data by using requester-responder relationship and another is a one-way message service that handles a message from the sender to the acceptor. An FAL user can execute message communication with a peer FAL user by using both user message and one-way message.

8.5.1.2 Requester-responder type message (user message)

The user message communication shall be processed according to the PDU flow diagram in Figure 38.

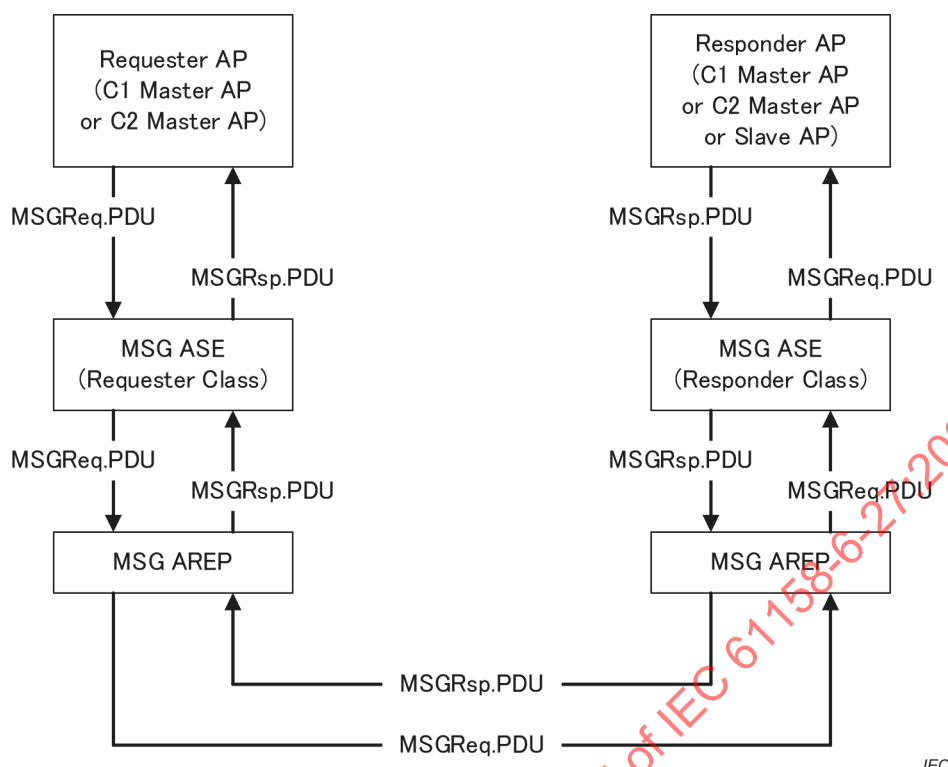


Figure 38 – PDU transmission flow for user message

A message communication is started from a Requester AP.

The Requester AP sends an MSGReq-PDU to a Responder AP via an MSG ASE and an MSG AREP in the Requester side.

The Responder AP receives the MSGReq-PDU via an MSG AREP and an MSG ASE in the Responder side.

The Responder AP decodes the MSGReq-PDU, executes some activity according to the contents of the MSGReq-PDU, and generates the response data as an MSGRsp-PDU.

The MSGRsp-PDU is transferred to the Requester AP in the inverse process for the MSGReq-PDU.

The Requester AP decodes the MSGRsp-PDU and executes some activity according to the contents of it.

8.5.1.3 One-way message

A message communication that requires no response shall be processed according to the PDU flow diagram in Figure 39.

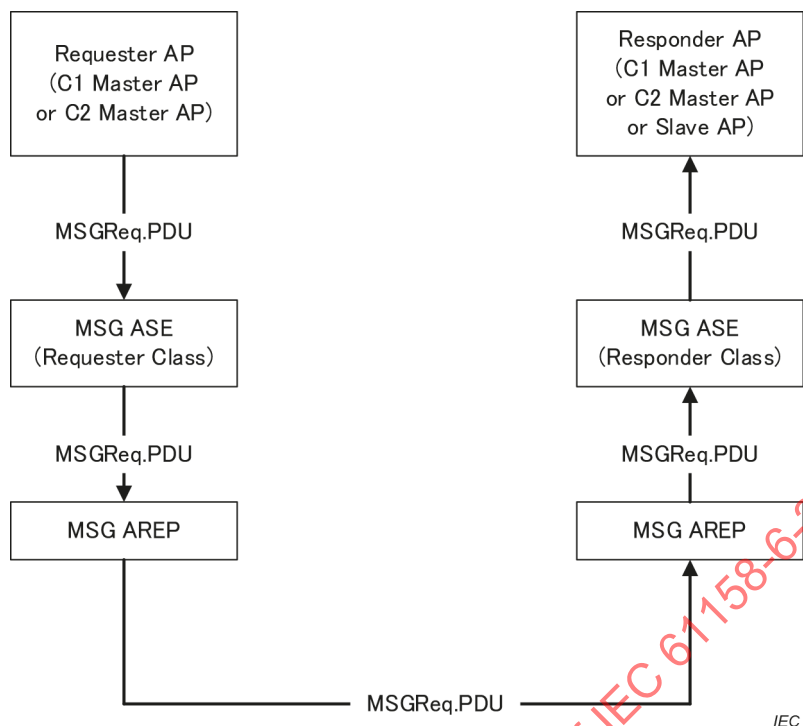


Figure 39 – PDU transmission flow for one-way message

A message communication is started from a Requester AP.

The Requester AP sends MSGReq-PDU to a Responder AP via an MSG ASE and an MSG AREP in the Requester side.

The Responder AP receives the MSGReq-PDU via an MSG AREP and an MSG ASE in the Responder side.

The Responder-AP decodes the MSGReq-PDU and executes some activity according to the contents of it.

8.5.2 Requester Protocol Machine (MSGPM-RQ)

8.5.2.1 State descriptions

Figure 40 shows the MSGPM-RQ state chart diagram, and Table 41 describes each state of the MSGAR-RQ.

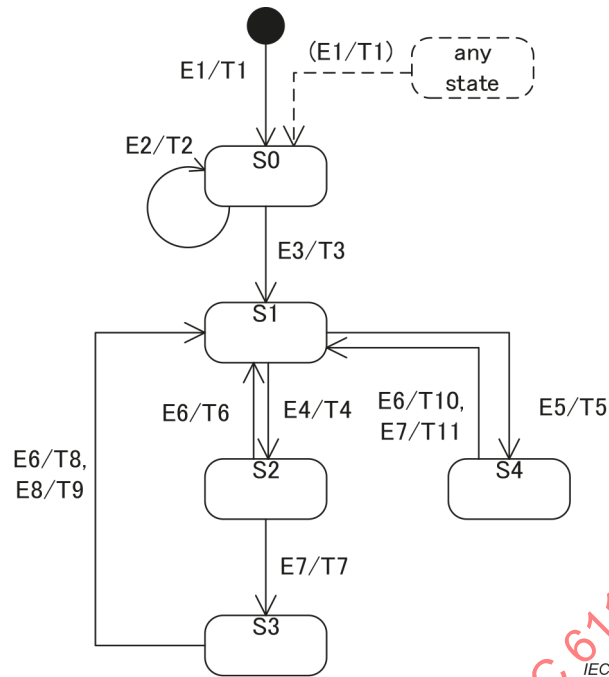


Figure 40 – State chart diagram of MSGPM-RQ

Table 41 – State descriptions of MSGPM-RQ

S#	State	Substate	Description
S0	Disabled		State when the Requester Class of MSG ASE (Requester) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. Entry/ Initial values are set in Attributes of the MSG Requester object.
S1	WaitRequestMSG		State when the PM is waiting for a request from a FAL user to transfer a message
S2	SendingRequestMSG		State when the PM is transmitting the MSGREQ-PDU
S3	WaitConfirmMSG		State when the PM is waiting for the response from the peer Responder
S4	SendingOnewayMSG		State when the PM is transmitting the MSGREQ-PDU as a one-way message

8.5.2.2 Triggering events

Table 42 lists each trigger event of the MSGPM-RQ.

Table 42 – Trigger event descriptions of MSGPM-RQ

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	MSG-Reset.req	FSM ASE		
E2	MSG-Open.req	FSM ASE	AREPID	
E3	MSG-Enable.req	FSM ASE		
E4	MSG-UserMessage.req	FAL user (Requester AP)	TID, SPID, DPID, Length, MSGReq-SDU	
E5	MSG-OnewayMessage.req	FAL user (Requester AP)	TID, SPID, DPID, Length, MSGReq-SDU	
E6	MSG-AbortTransaction.req	FAL user (Requester AP)	TID	
E7	AR-SendMessage.cnf	AR ASE (MSG-AR)	ServiceStatus	
E8	AR-ReceiveMessage.cnf	AR ASE (MSG-AR)	ServiceStatus, SPID, Length, MSGService-SDU, Result	

8.5.2.3 Action descriptions at state transitions

Table 43 describes state transitions of the state machine of MSGPM-RQ.

Table 43 – Transitions of MSGPM-RQ

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1: MSG-Reset.req /*-- Clearing all attributes of MSG requester --*/;	S0:Disabled
T2	S0:Disabled	E2: MSG-Open.req(AREPID) /*-- Initializing MSG requester --*/; ^a	S0:Disabled
T3	S0:Disabled	E3: MSG-Enable.req /*-- no-operation --*/;	S1:WaitRequestMSG
T4	S1:WaitRequestMSG	E4: MSG-UserMessage.req(SAPIID, TID, Length, MSGReq-PDU) / AR-SendMessage.req(TID, Length, MSGReq-PDU);	S2:SendingRequestMSG
T5	S1:WaitRequestMSG	E5: MSG-OnewayMessage.req(SAPIID, TID, Length, MSGReq-PDU) / AR-SendMessage.req(TID, Length, MSGReq-PDU);	S4:SendingOnewayMSG
T6	S2:SendingRequestMSG	E6: MSG-AbortTransaction.req(SAPIID, TID) / AR-AbortMessage.req(SAPIID, TID);	S1:WaitRequestMSG
T7	S2:SendingRequestMSG	E7: AR-SendMessage.cnf(+) / AR-ReceiveMessage.req;	S3:WaitConfirmMSG
T8	S3:WaitConfirmMSG	E6: MSG-AbortTransaction.req(SAPIID, TID) / AR-AbortMessage.req(SAPIID, TID);	S1:WaitRequestMSG
T9	S3:WaitConfirmMSG	E8: AR-ReceiveMessage.cnf(TID, Length, MSGRsp-PDU) / MSG-UserMessage.cnf(TID, Length, MSGRsp-PDU);	S1:WaitRequestMSG

T#	Source State	Event (arguments) [conditions] / Action	Target State
T10	S4:SendingOnewayMSG	E6: MSG-AbortTransaction.req(SAPID, TID) / AR-AbortMessage.req;	S1:WaitRequestMSG
T11	S4:SendingOnewayMSG	E7: AR-SendMessage.cnf(Result) / MSG-OnewayMessage.cnf(TID);	S1:WaitRequestMSG
^a The detailed process depends on the system implementation.			

8.5.3 Responder Protocol Machine (MSGPM-RS)

8.5.3.1 State descriptions

Figure 41 shows the MSGPM-RS state chart diagram, and Table 44 describes each state of the MSGPM-RS.

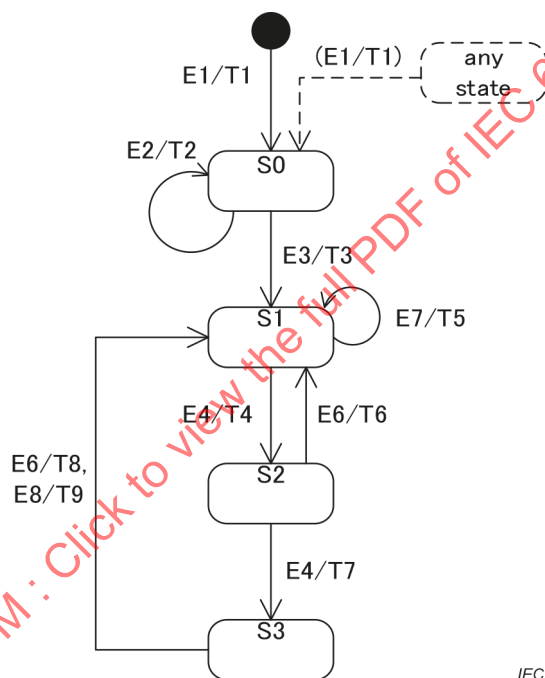


Figure 41 – State chart diagram of MSGPM-RS

Table 44 – State descriptions of MSGPM-RS

S#	State	Substate	Description
S0	Disabled		State when the Responder Class of MSG ASE (Responder) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. Entry/ Initial values are set in Attributes of the MSG Responder object.
S1	WaitIndicationMSG		State when the PM is waiting for a message from a Requester
S2	WaitResponseMSG		State when the PM is waiting for a response message from the FAL user (Responder AP)
S3	SendingResponseMSG		State when the PM is transmitting the response message to the Requester after receiving it from the Responder AP

8.5.3.2 Triggering events

Table 45 lists each trigger event of the MSGPM-RS.

Table 45 – Trigger event descriptions of MSGPM-RS

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	MSG-Reset.req	FSM ASE		
E2	MSG-Open.req	FSM ASE	AREPID	
E3	MSG-Enable.req	FSM ASE		
E4	MSG-UserMessage.rsp	FAL user (Responder AP)	ServiceStatus, TID, DPID, Length, MSGResp-PDU	
E5	MSG-AbortTransaction.req	FAL user (Responder AP)	TID	
E6	AR-ReceiveMessage.cnf	AR ASE (MSG-AR)	ServiceStatus, SPID, Length, MSGService-PDU	
E7	AR-SendMessage.cnf	AR ASE (MSG-AR)	ServiceStatus	

8.5.3.3 Action descriptions at state transitions

Table 46 describes state transitions of the state machine of MSGPM-RS.

Table 46 – Transitions of MSGPM-RS

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1: MSG-Reset.req / /*-- Clearing all attributes of the MSG responder --*/;	S0:Disabled
T2	S0:Disabled	E2: MSG-Open.req / /*-- Initializing the MSG responder --*/; ^a	S0:Disabled
T3	S0:Disabled	E3: MSG-Enable.req / AR-ReceiveMessage.req;	S1:WaitIndicationMSG
T4	S1:WaitIndicationMSG	E7: AR-ReceiveMessage.cnf(TID, Length, MSGReq-PDU, Result) [Message that requires no response ^b] / MSG-OnewayMessage.ind(TID, Length, MSGReq-PDU);	S1:WaitIndicationMSG
T5	S1:WaitIndicationMSG	E7: AR-ReceiveMessage.cnf(TID, Length, MSGReq-PDU, Result) [Message that requires a response ^b] / MSG-UserMessage.ind(TID, Length, MSGReq-PDU);	S2:WaitResponseMSG
T6	S2:WaitResponseMSG	E6: MSG-AbortTransaction.req(SAPID, TID) / AR-AbortMessage.req; AR-ReceiveMessage.req;	S1:WaitIndicationMSG

T#	Source State	Event (arguments) [conditions] / Action	Target State
T7	S2:WaitResponseMSG	E4: MSG-UserMessage.rsp (TID, Length, MSGRsp-PDU) / AR-SendMessage.req(TID, Length, MSGRsp-PDU);	S3:SendingResponseMSG
T8	S3:SendingResponseMSG	E6: MSG-AbortTransaction.req(SAPID, TID) / AR-AbortMessage.req; AR-ReceiveMessage.req;	S1:WaitIndicationMSG
T9	S3:SendingResponseMSG	E8: AR-SendMessage.cnf / AR-ReceiveMessage.req;	S1:WaitIndicationMSG
^a The detailed process depends on the system implementation. ^b Whether a response is required or not is defined according to the contents of the MSGReq-PDU. However, the details depend on the implemented user message specifications above the FAL.			

9 Application relationship protocol machine (ARPM)

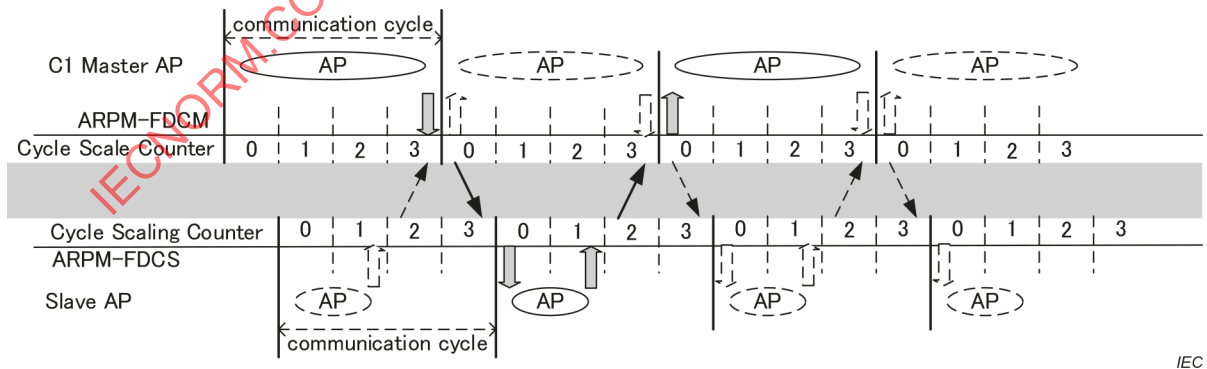
9.1 General

ARPM is the state machine for the AR ASE. It shall handle the lower layer to transmit and receive APDUs to and from the peer ASE. And it shall exchange those APDUs with other ASEs through providing the AR services in the local FAL.

9.2 ARPM for FDC ASE

9.2.1 Overview

This type of ARPM shall handle state transition to exchange of a CMD-PDU and a RSP-PDU with the peer ARPM, shall manage a communication cycle and shall control the transfer mode (single transfer or dual transfer), as shown in Figure 42 and Figure 43. But it can neither manage the connection, the command transaction and the WDT values nor check the validity of contents of a CMD-PDU and a RSP-PDU.



IEC

NOTE The figure shows the case of $T_{COMCYC} = T_{CYC} \times 4$,

where

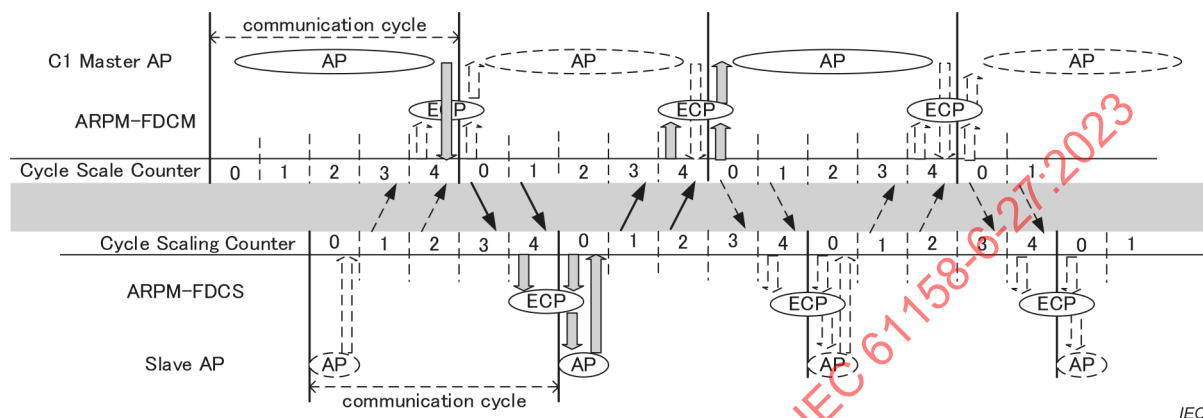
T_{COMCYC} is the communication cycle;

T_{CYC} is the transmission cycle.

Figure 42 – Example of single transfer process

FDC ASE can optionally provide the function of the dual transfer mode to improve the reliability of the received APDU. This mode is a communication method in which an identical APDU is transferred twice within one communication cycle, and then they are checked and verified on the receiver side to detect an error or missing PDU, see Figure 43.

The dual transfer mode can be operated only under the synchronous communication state and $T_{\text{COMCYC}} \geq T_{\text{CYC}} \times 2$, where T_{COMCYC} is a communication cycle and T_{CYC} is a transmission cycle.



NOTE The figure shows the case of $T_{\text{COMCYC}} = T_{\text{CYC}} \times 5$,

where

T_{COMCYC} is the communication cycle;

T_{CYC} is the transmission cycle;

ECP is the Error-detection and PDU-comparison process.

Figure 43 – Example of dual transfer process

9.2.2 ARPM for FDC Master (ARPM-FDCM)

9.2.2.1 State descriptions

Figure 44 shows the ARPM-FDCM state chart diagram, and Table 47 describes each state of the ARPM-FDCM.

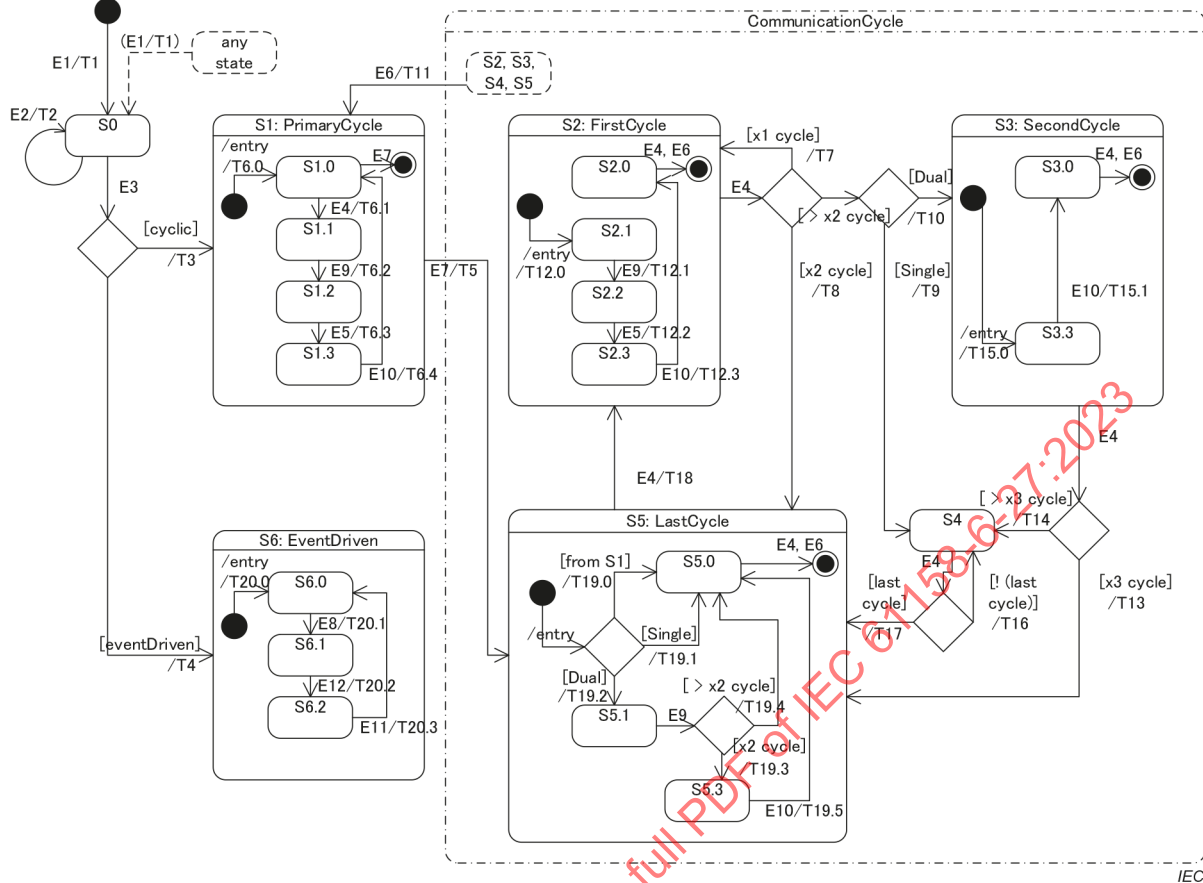


Figure 44 – State chart diagram of ARPM-FDCM

Table 47 – State descriptions of ARPM-FDCM

S#	State	Substate	Description
S0	Disabled	-	State when the FDC Master AR Class has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. Entry/ Initial values are set in Attributes of the FDCMaster AR object
S1	PrimaryCycle	-	State when the cyclic transmission with the transmission cycle PrimaryCycle has already started in the lower layer ARPM-FDCM sends and receives data with FDC Slave AR Class object in the communication partner. However, since the connection between FDC ASEs is disconnected, the communication cycle is not effective yet. Entry/ Transition into the sub state: S1.0.
S1.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S1.1		WaitGET_DATAcnf	Substate when the PM is waiting for the associated CTC-READ-DATA.cnf after requesting the CTC-READ-DATA.req to the related CTC-SAPID
S1.2		WaitFDC-DataExchngecnf	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S1.3		WaitSET_DATAcnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID
S2	FirstCycle	-	State when a connection in FDC ASE is established, the communication cycle is specified in the form of integer multiple of the primary cycle as ComTime, and in this first primary cycle the ARPM-FDCM exchanges PDUs between the FDC ASE and CTC If the communication cycle is the same as the primary cycle, the PM remains in this state while the FDC PM is in connected. Entry/ Transition into the sub state: S2.1.
S2.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S2.1		WaitGET_DATA.cnf	Substate when the PM is waiting for the associated CTC-READ-DATA.cnf after requesting the CTC-READ-DATA.req to the related CTC-SAPID
S2.2		WaitFDC-DataExchnge.cnf	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S2.3		WaitSET_DATA.cnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID

S#	State	Substate	Description
S3	SecondCycle	-	State when a connection in FDC ASE is established and the ARPM-FDCM is in the second primary cycle of the communication cycle If the communication cycle is twice primary cycle, the PM does not have a transition to this state but to LastCycle at the second primary cycle. In the dual transfer mode, the second CMD-PDU transmission is executed after the regular PDU exchange in the first cycle. Entry/ Transition into the sub state: S3.3.
S3.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S3.3		WaitSET_DATAcnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID
S4	MiddleCycle	-	State when a connection in FDC ASE is established and the ARPM-FDCM is in the middle primary cycle; not the 1st, the 2nd nor the last cycle If the communication cycle is treble primary cycle, the PM does not have a transition to this state but to LastCycle at the third primary cycle.
S5	LastCycle	-	State when a connection in FDC ASE is established and the ARPM-FDCM is in the last primary cycle of the communication cycle In the dual transfer mode, the PM receives a RSP-PDU from the slave and stores it into the internal buffer before the regular PDU exchange in the first cycle. entry [(the source state == S1) (single transfer mode)] / Transition into the sub state: S5.0 entry [dual transfer mode] / Transition into the sub state: S5.1
S5.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S5.1		WaitGET_DATAcnf	Substate when the PM is waiting for the associated CTC-READ-DATA.cnf after requesting the CTC-READ-DATA.req to the related CTC-SAPID
S5.3		WaitSET_DATAcnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID, if the communication cycle is twice primary cycle
S6	EventDriven	-	Corresponding state to the event-driven mode in CTC A CMD-PDU is not cyclically transmitted but in case of necessity it's done and after that, sequentially RSP-PDU is received using DL services provided in the event-driven mode. Entry/ Transition into the sub state: S6.0.
S6.0		WaitRequestCMD	Substate when the PM is waiting for AR-SendCommand.req from FDC ASE
S6.1		SendingRequestCMD	Substate when the PM is waiting for completion of CTC-SDN.req
S6.2		WaitConformCMD	Substate when the PM is waiting for the reception of CTC-SDN.ind from CTC as a RSP-PDU

9.2.2.2 Triggering events

Table 48 lists each trigger event of the ARPM-FDCM.

Table 48 – Trigger event descriptions of ARPM-FDCM

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	AR-Reset.req	FSM ASE		
E2	AR-Open.req	FSM ASE	DLSAPID, FDCSAPID	
E3	AR-Enable.req	FSM ASE	TransmissionMode	
E4	EVM-SyncEvent.ind	EVM ASE		
E5	FDC-DataExchange.cnf	FDC ASE (Master)	CMD-SDU	
E6	AR-ResetCycle.req	FDC ASE (Master)		
E7	AR-StartComCycle.req	FDC ASE (Master)	ComTime, DTMode, CMDForm	
E8	AR-SendCommand.req	FDC ASE (Master)	CMD-SDU	
E9	CTC-READ-DATA.cnf	CTC	Result, SDU	
E10	CTC-WRITE-DATA.cnf	CTC	Result	
E11	CTC-SDN.ind	CTC	SAP_ID, Node_ID, Length, SDU	
E12	CTC-SDN.cnf	CTC	Result	
E13	Error detected	FDC ASE (Master)		See 8.4.6

9.2.2.3 Action descriptions at state transitions

Table 49 describes state transitions of the main SM of ARPM-FDCM. Moreover,

Table 50 describes state transitions of the submachine of ARPM-FDCM.

Table 49 – Transitions of main SM of ARPM-FDCM

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1:AR-Reset.req / /*-- Clearing all attributes of the FDC-Master AR --*/;	S0:Disabled
T2	S0:Disabled	E2:AR-Open.req / /*-- Initializing the FDC-Master AR --*/; ^a	S0:Disabled
T3	S0:Disabled	E3:AR-Enable.req(comMode) [comMode == "cyclic"] / /*-- no-operation --*/;	S1:PrimaryCycle
T4	S0:Disabled	E3:AR-Enable.req(comMode) [comMode == "eventDriven"] / /*-- no-operation --*/;	S6:EventDriven
T5	S1:PrimaryCycle	E7:AR-StartComCycle / /*-- no-operation --*/;	S5:LastCycle
T6	S1:PrimaryCycle	The other events / /*-- state transition in the submachine --*/;	S1:PrimaryCycle

T#	Source State	Event (arguments) [conditions] / Action	Target State
T7	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 1] / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Master objects;	S2:FirstCycle
T8	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 2] / CycleScaleCounter++;	S5:LastCycle
T9	S2:FirstCycle	E4:EVM-SyncEvent.ind [(ComTime > 2) && (DTMode == "Single")] / CycleScaleCounter++;	S4:MiddleCycle
T10	S2:FirstCycle	E4:EVM-SyncEvent.ind [(ComTime > 2) && (DTMode == "Dual")] / CycleScaleCounter++;	S3:SecondCycle
T11	S2, S3, S4, or S5	E6:AR-ResetCycle.req / /*-- no-operation --*/;	S1:PrimaryCycle
T12	S2:FirstCycle	The other events / /*-- state transition in the submachine --*/;	S2:FirstCycle
T13	S3:SecondCycle	E4:EVM-SyncEvent.ind [ComTime == 3] / CycleScaleCounter++;	S5:LastCycle
T14	S3:SecondCycle	E4:EVM-SyncEvent.ind [ComTime > 3] / CycleScaleCounter++;	S4:MiddleCycle
T15	S3:SecondCycle	The other events / /*-- state transition in the submachine --*/;	S3:SecondCycle
T16	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter < (ComTime - 2)] / CycleScaleCounter++;	S4:MiddleCycle
T17	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter == (ComTime - 2)] / CycleScaleCounter++;	S5:LastCycle
T18	S5:LastCycle	E4:EVM-SyncEvent.ind / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Master objects;	S2:FirstCycle
T19	S5:LastCycle	The other events / /*-- state transition in the submachine --*/;	S5:LastCycle
T20	S6:EventDriven	Any events except E1 / /*-- state transition in the submachine --*/;	S6:EventDriven
^a The detailed process depends on the system implementation.			

Table 50 – Transitions of submachine of ARPM-FDCM

T#	Source State	Event (arguments) [conditions] / Action	Target State
T6.0 (T3)	(S0)	/entry / /*-- no-operation --*/;	S1.0:Idle
T6.1	S1.0:Idle	E4:EVM-SyncEvent.ind / CTC-READ-DATA.req;	S1.1:WaitGET_DAT Acnf
T6.2	S1.1:WaitGET_DAT Acnf	E9:CTC-READ-DATA.cnf(rsdu) / FDC-DataExchange.req(rsdu);	S1.2:WaitFDC- DataExchangecnf
T6.3	S1.2:WaitFDC- DataExchangecnf	E5:FDC-DataExchange.cnf(csdu) / CTC-WRITE-DATA.req(csdu);	S1.3:WaitSET_DAT Acnf
T6.4	S1.3:WaitSET_DAT Acnf	E10:CTC-WRITE-DATA.cnf(+) / /*-- no-operation --*/;	S1.0:Idle
T12.0 (T7, T18)	(S2 or S5)	/entry / CTC-READ-DATA.req;	S.2.1:WaitGET_DAT Acnf
T12.1	S2.1:WaitGET_DAT Acnf	E9: CTC-READ-DATA.cnf(rsdu) / if(DTMode == "Dual") { /*-- Error detection and PDU comparison process -- with lastRsdu and rsdu --*/ } else { /*-- no-operation --*/; } FDC-DataExchange.req(rsdu);	S.2.2: WaitFDC- DataExchangecnf
T12.2	S2.2: WaitFDC- DataExchangecnf	E5:FDC-DataExchange.cnf(csdu) / CTC-WRITE-DATA.req(csdu); if(DTMode == "Dual") { lastCsdu = csdu; }	S2.3:WaitSET_DAT Acnf
T12.3	S2.3:WaitSET_DAT Acnf	E10:CTC-WRITE-DATA.cnf(+) / /*-- no-operation --*/;	S.2.0:Idle
T15.0 (T10)	(S2:FirstCycle)	/entry / CTC-WRITE-DATA.req(csdu);	S3.3:WaitSET_DAT Acnf
T15.1	S3.3:WaitSET_DAT Acnf	E10: CTC-WRITE-DATA.cnf(+) / /*-- no-operation --*/;	S3.0:Idle
T19.0 (T5)	(S1:PrimaryCycle)	/entry / /*-- no-operation --*/;	S5.0:Idle
T19.1	(S2, S3, or S4)	/entry [DTMode == "Single"] / /*-- no-operation --*/;	S5.0:Idle
T19.2	(S2, S3, or S4)	/entry: [DTMode == "Dual"] / CTC-READ-DATA.req;	S5.1:WaitGET_DAT Acnf
T19.3	S5.1:WaitGET_DAT Acnf	E9: CTC-READ-DATA.cnf(rsdu) [ComTime == 2] / lastRsdu = rsdu; CTC-WRITE-DATA.req(lastCsdu);	S5.3:WaitSET_DAT Acnf
T19.4	S5.1:WaitGET_DAT Acnf	E9: CTC-READ-DATA.cnf(rsdu) [ComTime > 2] / lastRsdu = rsdu;	S5.0:Idle

Figure 45 shows the ARPM-FDCS state chart diagram, and Table 51 describes each state of the ARPM-FDCS.



Table 51 – State descriptions of ARPM-FDCS

S#	State	Substate	Description
S0	Disabled		State when the FDC Slave AR Class has just instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. Entry/ Initial values are set in Attributes of the FDC Slave AR object
S1	PrimaryCycle		State when the cyclic transmission with the transmission cycle PrimaryCycle has already started in the lower layer ARPM-FDCM sends and receives data with FDC Slave AR Class object in the communication partner. However, since the connection between FDC ASEs is disconnected, the communication cycle is not effective yet. Entry/ Transition into the sub state: S1.0.
S1.0		Idle	Substate where the PM is waiting for the start signal based on the network clock for the next primary cycle
S1.1		WaitGET_DATAcnf	Substate when the PM is waiting for the associated CTC-READ-DATA.cnf after requesting the CTC-READ-DATA.req to the related CTC-SAPID
S1.2		WaitFDC-DataExchange	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S1.3		WaitSET_DATAcnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID
S2	FirstCycle		State when a connection in FDC ASE is established, the communication cycle is specified in the form of integer multiple of the primary cycle as ComTime, and in this first primary cycle the ARPM-FDCS exchanges PDUs between the FDC ASE and CTC If the communication cycle is the same as the primary cycle, the PM remains in this state while the FDC PM is in connected. entry [the source state != S7] / Transition into the substate: S2.1. entry [the source state == S7] / Transition into the substate: S2.2.
S2.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S2.1		WaitGET_DATAcnf	Substate when the PM is waiting for the associated CTC-READ-DATA.cnf after requesting the CTC-READ-DATA.req to the related CTC-SAPID
S2.2		WaitFDC-DataExchange	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S2.3		WaitSET_DATAcnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID

S#	State	Substate	Description
S4	MiddleCycle		State when a connection in FDC ASE is established, and in each middle primary cycle of the communication cycle the ARPM-FDCS relay a RSP-PDU from the FDC ASE to CTC
S4.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S4.1		WaitFDC-DataExchange	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S4.2		WaitSET_DATAcnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID
S5	LastCycle		State when a connection in FDC ASE is established, and in the last primary cycle of the communication cycle the ARPM-FDCS relay a RSP-PDU from the FDC ASE to CTC In the dual transfer mode, the PM receives a CMD-PDU from the master and stores it into the internal buffer before the regular PDU exchange in the first cycle. entry [(the source state != S7) && dual transfer mode] / Transition into the sub state: S5.1. entry [(the source state == S7) single transfer mode] / Transition into the sub state: S5.2.
S5.0		Idle	Substate when the PM is waiting for the start signal for the next primary cycle
S5.1		WaitGET_DATAcnf	Substate when the PM is waiting for the associated CTC-READ-DATA.cnf after requesting the CTC-READ-DATA.req to the related CTC-SAPID if in the dual transfer mode
S5.2		WaitFDC-DataExchange	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S5.3		WaitSET_DATAcnf	Substate when the PM is waiting for the associated CTC-WRITE-DATA.cnf after requesting the CTC-WRITE-DATA.req to the related CTC-SAPID
S6	EventDriven		Corresponding state to the event-driven mode in CTC A CMD-PDU is not cyclically received but as in case of necessity, it's done and after that, sequentially a RSP-PDU is transmitted using DL services in the event-driven mode. Entry/ Transition into the sub state: S6.0.
S6.0		WaitCMDIndication	Substate when the PM is waiting for the CTC-SDN.ind from CTC, which originates with the peer master AP
S6.1		WaitCMDResponse	Substate when the PM is waiting for the FDC-Command.rsp
S6.2		SendingCMDResponse	Substate when the PM is waiting for completion of CTC-SDN.req

S#	State	Substate	Description
S7	WaitComCycle		State when the PM is detecting the start edge of the communication cycle with a change of CMD-PDUs The destination state of the transition splits in two states depending on the transfer mode.
S7.0		Idle	Substate when the PM is waiting for the start signal for the next primary cycle
S7.1		WaitGET_DATAcnf	Substate when the PM is waiting for the associated CTC-READ-DATA.cnf to check its contents after requesting the CTC-READ-DATA.req to the related CTC-SAPID When the CMD Code and the WDT in the got PDU with the identical values to the previous ones, ARPM-FDCS transits to S7.0 and then waits for the next cycle without executing any process. On the other hand, when those values have changed, the PM has a transition to exit from this state.

9.2.3.2 Triggering events

Table 52 lists each trigger event of the ARPM-FDCS.

Table 52 – Trigger event descriptions of ARPM-FDCS

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	AR-Reset.req	FSM ASE		
E2	AR-Open.req	FSM ASE	DLSAPID, FDCSAPID	
E3	AR-Enable.req	FSM ASE	TransmissionMode	
E4	EVM-SyncEvent.ind	EVM ASE	NetClock	
E5	FDC-DataExchange.cnf	FDC ASE (Slave)	CMD-SDU	
E6	AR-ResetCycle.req	FDC ASE (Slave)		
E7	AR-StartComCycle.req	FDC ASE (Slave)	ComTime, DTMode, CMDForm	
E8	AR-SendCommand.rsp	FDC ASE (Slave)	ServiceStatus, RSP_ASDU	
E9	CTC-READ-DATA.cnf	CTC	Result, , SDU	
E10	CTC-WRITE-DATA.cnf	CTC	Result	
E11	CTC-SDN.ind	CTC	SAP_ID, Node_ID,Length, SDU	
E12	CTC-SDN.cnf	CTC	Result	
E13	Error detected	FDC ASE (Slave)		See 8.4.6
E14	ComCycleStart	This object		Internal event from S7

9.2.3.3 Action descriptions at state transitions

Table 53 describes state transitions of the main SM of ARPM-FDCS. Moreover,

Table 54 describes state transitions of the submachine of ARPM-FDCS.

Table 53 – Transitions of main SM of ARPM-FDCS

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1:AR-Reset.req / /*-- Clearing all attributes of the FDC-Slave AR --*/;	S0:Disabled
T2	S0:Disabled	E2:AR-Open.req / /*-- Initializing the FDC-Slave AR --*/; ^a	S0:Disabled
T3	S0:Disabled	E3:AR-Enable.req(comMode) [comMode == "cyclic"] / /*-- no-operation --*/;	S1:PrimaryCycle
T4	S0:Disabled	E3:AR-Enable.req(comMode) [comMode == "eventDriven"] / /*-- no-operation --*/;	S6:EventDriven
T5	S1:PrimaryCycle	E7:AR-StartComCycle / /*-- no-operation --*/;	S7:WaitComCycle
T6	S1:PrimaryCycle	The other events / /*-- state transition in the submachine --*/;	S1:PrimaryCycle
T7	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 1] / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Slave objects;	S2:FirstCycle
T8	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 2] / CycleScaleCounter++;	S5:LastCycle
T9	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime > 2] / CycleScaleCounter++;	S4:MiddleCycle
T10	S2, S4, or S5	E6:AR-ResetCycle.req / /*-- no-operation --*/;	S1:PrimaryCycle
T11	S2:FirstCycle	The other events / /*-- state transition in the submachine --*/;	S2:FirstCycle
T12	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter < (ComTime - 2)] / CycleScaleCounter++;	S4:MiddleCycle
T13	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter == (ComTime - 2)] / CycleScaleCounter++;	S5:LastCycle
T14	S4:MiddleCycle	The other events / /*-- state transition in the submachine --*/;	S4:MiddleCycle
T15	S5:LastCycle	E4:EVM-SyncEvent.ind / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Slave objects;	S2:FirstCycle

T#	Source State	Event (arguments) [conditions] / Action	Target State
T16	S5:LastCycle	The other events / /*-- state transition in the submachine --*/;	S5:LastCycle
T17	S6:EventDriven	The other events / /*-- state transition in the submachine --*/;	S6:EventDriven
T18	S7:WaitComCycle	E14:ComCycleStart [DTMode == Single] / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Slave objects;	S2:FirstCycle
T19	S7:WaitComCycle	E14:ComCycleStart [DTMode == Dual] / CycleScaleCounter = ComTime - 1;	S5:LastCycle
T20	S7:WaitComCycle	The other events / /*-- state transition in the submachine --*/;	S7:WaitComCycle
^a The detailed process depends on the system implementation.			

Table 54 – Transitions of submachine of ARPM-FDCS

T#	Source State	Event (arguments) [conditions] / Action	Target State
T6.0 (T3)	(S0)	/entry / /*-- no-operation --*/;	S1.0:Idle
T6.1	S1.0:Idle	E4:EVM-SyncEvent.ind / CTC-READ-DATA.req;	S1.1:WaitGET_DATA cnf
T6.2	S1.1:WaitGET_DATA cnf	E9:CTC-READ-DATA.cnf(csdu) / FDC-DataExchange.req(csdu);	S1.2:WaitFDC- DataExchangecnf
T6.3	S1.2:WaitFDC- DataExchangecnf	E5:FDC-DataExchange.cnf(rsdu) / CTC-WRITE-DATA.req(rsdu);	S1.3:WaitSET_DATA cnf
T6.4	S1.3:WaitSET_DATA cnf	E10:CTC-WRITE-DATA.cnf(+) / /*-- no-operation --*/;	S1.0:Idle
T11.0 (T7, T15, T18)	(S2, S5)	/entry / CTC-READ-DATA.req;	S2.1:WaitGET_DATA cnf
T11.1	(S7)	/entry / FDC-DataExchange.req(lastCsdu);	S2.2: WaitFDC- DataExchangecnf
T11.2	S2.1:WaitGET_DATA cnf	E9: CTC-READ-DATA.cnf(csdu) / if(DTMode == "Dual") { /*-- Error detection and PDU comparison process -- with lastCsdu and csdu --*/; } else { /*-- no-operation --*/; } FDC-DataExchange.req(csdu); lastCsdu = csdu;	S2.2: WaitFDC- DataExchangecnf
T11.3	S2.2: WaitFDC- DataExchangecnf	E5:FDC-DataExchange.cnf(rsdu) / CTC-WRITE-DATA.req(rsdu);	S2.3::WaitSET_DATA cnf

T#	Source State	Event (arguments) [conditions] / Action	Target State
T11.4	S2.3:WaitSET_DATAcnf	E10:CTC-WRITE-DATA.cnf(+) / /*-- no-operation --*/;	S.2.0:Idle
T14.0 (T10)	(S2:FirstCycle)	/entry / FDC-DataExchange.req(csdu1);	S4.1: WaitFDC-DataExchangecnf
T14.1	S4.1: WaitFDC-DataExchangecnf	E5:FDC-DataExchange.cnf(rsdu) / CTC-WRITE-DATA.req(rsdu);	S4.2::WaitSET_DATAcnf
T14.2	S4.2:WaitSET_DATAcnf	E10:CTC-WRITE-DATA.cnf(+) / /*-- no-operation --*/;	S.4.0:Idle
T16.0	(S2, S4)	/entry / CTC-READ-DATA.req;	S5.1:WaitGET_DATAcnf
T16.1	(S7)	/entry / /*-- no-operation --*/;	S5.0:Idle
T16.2	S5.1:WaitGET_DATAcnf	E9: CTC-READ-DATA.cnf(csdu) / FDC-DataExchange.req(lastCsdu); lastCsdu = csdu;	S5.2:WaitFDC-DataExchangecnf
T16.3	S5.2:WaitFDC-DataExchangecnf	E5: FDC-DataExchange.cnf(rsdu) / CTC-WRITE-DATA.req(rsdu);	S5.3:WaitSET_DATAcnf
T16.4	S5.3:WaitSET_DATAcnf	E10:CTC-WRITE-DATA.cnf(+) / /*-- no-operation --*/;	S5.0:Idle
T17.0 (T4)	(S0)	/entry: / /*-- no-operation --*/;	S6.0:WaitRequestCMD
T17.1	S6.0:WaitCMDIndication	E11: CTC-SDN.ind (SAP_ID, Node_ID, Length, SDU) / AR-SendCommand.ind(csdu);	S6.1:WaitCMDResponse
T17.2	S6.1:WaitCMDResponse	E8: AR-SendCommand.rsp(serviceStatus, rsdu) / CTC-SDN.req(SAP_ID, Node_ID,Length, SDU);	S6.2:SendingCMDResponse
T17.3	S6.2:SendingCMDResponse	E12: CTC-SDN.cnf(+) / /*-- no-operation --*/;	S6.0:WaitCMDIndication
T17.4	S6.2:SendingCMDResponse	E13:Communication time-out / /*-- Communication alarm processing/resume processing --*/;	S6.0:WaitCMDIndication
T20.0		/entry / /*-- no-operation --*/;	S7.0:Idle
T20.1	S7.0:Idle	E4: EVM-SyncEvent.ind / CTC-WRITE-DATA.req;	S7.1: WaitGET_DATAcnf
T20.2	S7.1: WaitGET_DATAcnf	E9:CTC-READ-DATA.cnf(csdu) [csdu == lastCsdu] / lastCsdu = csdu;	S7.0:Idle
T20.3	S7.1: WaitGET_DATAcnf	E9:CTC-READ-DATA.cnf(csdu) [csdu != lastCsdu] / /*-- signal E14:StartComCycle to the main machine --*/;	Exit from the submachine
NOTE "lastCsdu" is a local variable.			

9.3 ARPM for MSG ASE (ARPM-MSG)

9.3.1 State descriptions

The MSGPM manages the transaction of messages. On the other hand, this ARPM only manages each activity of transmission or reception.

Figure 46 shows the ARPM-MSG state chart diagram, and Table 55 describes each state of the ARPM-MSG.

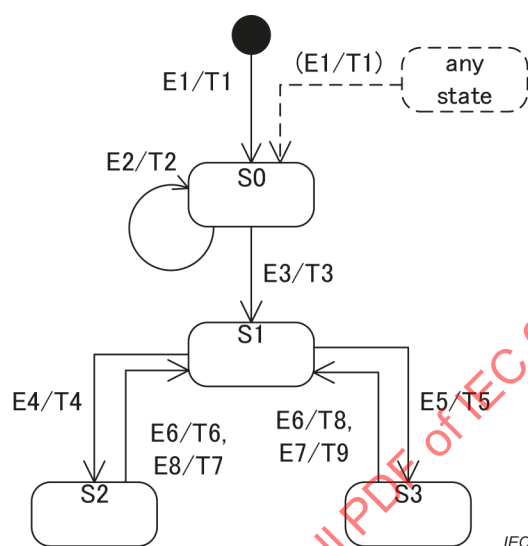


Figure 46 – Statechart diagram of ARPM-MSG

Table 55 – State descriptions of ARPM-MSG

S#	State	Substate	Description
S0	Disabled		State when the Message AR Class has just instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. Entry/ Initial values are set in Attributes of the Message AR object
S1	Ready		State when the PM is waiting for the service request, the AR-SendMessage.req or the AR-ReceiveMessage.req, from the MSG ASE
S2	SendingMSG		State when the PM is waiting for the completion of the transmission, CTC-SDA.cnf, after receiving AR-SendMessage.req
S3	ReceivingMSG		State when the PM is waiting for message reception, CTC-SDA.ind, after receiving AR-ReceiveMessage.req

9.3.2 Triggering events

Table 56 lists each trigger event of the ARPM-MSG.

Table 56 – Trigger event descriptions of ARPM-MSG

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	AR-Reset.req	FSM ASE		

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E2	AR-Open.req	FSM ASE	DLSAPID, MSGSAPID	
E3	AR-Enable.req	FSM ASE		
E4	AR-SendMessage.req	MSG ASE	DPID, Length, MSGService-PDU	
E5	AR-ReceiveMessage.req	MSG ASE	SPID, MaxLength	
E6	AR-AbortMessage.req	MSG ASE		
E7	CTC-SDA.ind	CTC	SAP_ID, NodeID, Length, SDU	
E8	CTC-SDA.cnf	CTC	Result	

9.3.3 Action descriptions at state transitions

Table 57 describes state transitions of the state machine of ARPM-MSG.

Table 57 – Transitions of ARPM-MSG

T#	Source State	Event (arguments) [conditions] / Action	Target State
T1	(any state)	E1: AR-Reset.req / /*-- Clearing all attributes of the Message-AR --*/;	S0:Disabled
T2	S0:Disabled	E2: AR-Open.req(AREPTYPE, DLSAPID) / /*-- Initializing the Message-AR --*/; ^a	S0:Disabled
T3	S0:Disabled	E3: AR-Enable.req / /*-- no-operation --*/;	S1:Ready
T4	S1:Ready	E4: AR-SendMessage.req(TID, Length, MSGReq-PDU) / /*-- Generate SDU from MSGRsp-PDU. --*/; CTC-SDA.req(SDU);	S2:SendingMSG
T5	S1:Ready	E5: AR-ReceiveMessage.req / /*-- no-operation --*/;	S3:ReceivingMSG
T6	S2:SendingMSG	E6: AR-AbortMessage.req(SAPID, TID) / /*-- no-operation --*/;	S1:Ready
T7	S2:SendingMSG	E8: CTC-SDA.cnf(+) (Result) / /*-- no-operation --*/;	S1:Ready
T8	S3:ReceivingMSG	E6: AR-AbortMessage.req(SAPID, TID) / /*-- no-operation --*/;	S1:Ready
T9	S3:ReceivingMSG	E7: CTC-SDA.ind (SAPID, NodeID, Length, SDU) / /*-- Generate MSGRsp-PDU from SDU. --*/; AR-ReceiveMessage.cnf(+) (TID, Length, MSGRsp-PDU);	S1:Ready
^a The detailed process depends on the system implementation.			

10 DLL mapping protocol machines (DMPMs)

The DLL mapping shall be according to 8.2.5.

Annex A (informative)

Device profile and FDC command sets

Table A.1 shows an example of the device profiles. The device profile provides FAL users a command set of the _FDCServicePDU for a certain application field.

Table A.1 – Example of registered device profiles

Range	Category	Profile Code	Profile name
'00'H to '0F'H	System common profiles	'00'H	Legacy command set
		'01'H	Minimum command set for device configuration
		...	Reserved
'10'H to '1F'H	Servo drive profiles	'10'H	Standard type servo drive
		'11'H	High resolution control type servo drive
		...	Reserved
'20'H to '2F'H	Inverter drive profiles	'20'H	Standard type inverter drive
		...	Reserved
'30'H to '3F'H	I/O device profiles	'30'H	Standard type I/O device
		...	Reserved
'40'H to '7F'H	Reserved	n/a	n/a
'80'H to 'FF'H	Reserved	n/a	n/a

Table A.2 shows an example of the command set. The contents are a set of command codes of the profile '00'H, or the legacy but most common command set.

Table A.2 – Example command list of the profile '00'H

Code	+0	+1	+2	+3	Description
'00'H	NOP	PRM_RD	PRM_WR	ID_RD	Device common commands
'04'H	CONFIG	ALM_RD	ALM_CLR	n/a	
'08'H	n/a	n/a	n/a	n/a	
'0C'H	n/a	SYNC_SET	CONNECT	DISCONNECT	
'10'H	n/a	n/a	n/a	n/a	
'14'H	n/a	n/a	n/a	n/a	
'18'H	n/a	n/a	n/a	PPRM_RD	
'1C'H	PPRM_WR	n/a	n/a	n/a	
'20'H	POS_SET	BRK_ON	BRK_OFF	SENS_ON	Servo commands, inverter commands, and so on
'24'H	SENS_OFF	HOLD	n/a	n/a	
'28'H	n/a	n/a	n/a	n/a	
'2C'H	n/a	n/a	n/a	n/a	
'30'H	SMON	SV_ON	SV_OFF	n/a	
'34'H	INTERPOLATE	POSING	FEED	n/a	
'38'H	LATCH	EX_POSING	ZRET	n/a	
'3C'H	VELCTRL	TRQCTRL	ADJ	SVCTRL	
'40'H	INV_CTRL	INV_IO	n/a	n/a	Omitted
...	...	...	...	...	
'50'H	DATA_RWA	DATA_RWS	n/a	n/a	
...	...	...	...	...	Omitted
'60'H	CF_POSCTRL	CF_VELCTRL	CF_TRQCTRL	n/a	Servo commands
...	...	...	...	...	Omitted

Annex B (normative)

Virtual memory space and Device Information

B.1 Overview

The Device Information shall be defined and embedded as data blocks on virtual memory space (see Figure B.1). Therefore, every communication system user can access the information through the network about any vendor's products in a standardized manner.

The Device Information can be also provided as a script file with a specific manner. Such a file is called the "Mechatronic Device Information" (MDI) file in the Type 27 FAL. But the script specification is out of the scope for this document.

Pseudo memory address	
'FFFF FFFF'H	Vender specific area
'8000 0000'H	
'7FFF FFFF'H	Reserved
'1001 0000'H	
'1000 FFFF'H	Communication Data Object
'1000 0000'H	
'0FFF FFFF'H	Device Information includes - setting parameter for the device profile, and - device identifier information.
'0000 0000'H	

Figure B.1 – Memory map of virtual memory space

B.2 Communication Data Object

The CDO (Communication Data Object) area is allocated CDO of the FSMUL (Field System Management Underlying Layer) and is defined as follows. The CDO area of the virtual memory space is a read-only area.

Table B.1 shows the memory map of the CDO area.

Table B.1 – Memory map of CDO area

Pseudo memory address	CDO address	Description
'1000 0000'H to '1000 001F'H	'0000'H to '001F'H	Node object – Information
'1000 0020'H to '1000 002F'H	'0020'H to '002F'H	Node object – status
'1000 0030'H to '1000 003F'H	'0030'H to '003F'H	Node object – control
'1000 0040'H to '1000 41FF'H	'0040'H to '41FF'H	Node object – configuration
'1000 5000'H to '1000 9FFF'H	'5000'H to '9FFF'H	Module object

B.3 Device Information

B.3.1 Device identifier area structure

Device identifier information (Device ID) area is a part of the Device Information in the virtual memory space. Therefore, it shall be able to be read out by using either ID_RD-CMD-PDU or MEM_RD-CMD-PDU.

Figure B.2 shows the memory map of the device ID area.

				'0000 03FF'H	Vendor Specific
				'0000 0300'H	
'0000 00FF'H	Support sub command	'0000 01FF'H	Reserved	'0000 02FF'H	Reserved
				'0000 02E4'H	
'0000 00E0'H				'0000 02E0'H	Sub device version#3
	Support main command				Sub device name#3
				'0000 02C0'H	
'0000 00C0'H	Reserved				Reserved
				'0000 02A4'H	
				'0000 02A0'H	Sub device version#2
					Sub device name#2
'0000 008C'H	MAC address				
'0000 0084'H					
'0000 0080'H	SupportComMode			'0000 0280'H	
'0000 007C'H	Reserved				Reserved
'0000 0078'H	Reserved				
'0000 0074'H	CurrentDevProfile				
'0000 0070'H	Reserved	'0000 0170'H			
'0000 006C'H	Reserved	'0000 016C'H	DeviceSubCode		
'0000 0068'H	Reserved	'0000 0168'H	MaxComCycle		
'0000 0064'H	Reserved	'0000 0164'H	MinComCycle	'0000 0264'H	
'0000 0060'H	CycleGranularity	'0000 0160'H	MaxTransCycle	'0000 0260'H	Sub device version#1
					Sub device name#1
'0000 005C'H	Reserved	'0000 015C'H	MinTransCycle		
'0000 0058'H	Reserved	'0000 0158'H	CurrentTransRspSize		
'0000 0054'H	ProfileVersion#3	'0000 0154'H	MaxTransRspSize		
'0000 0050'H	DeviceProfile#3	'0000 0150'H	MinTransRspSize		
'0000 004C'H	ProfileVersion#2	'0000 014C'H	CurrentTransCmdSize		
'0000 0048'H	DeviceProfile#2	'0000 0148'H	MaxTransCmdSize		
'0000 0044'H	ProfileVersion#1	'0000 0144'H	MinTransCmdSize		
'0000 0040'H	DeviceProfile#1	'0000 0140'H	AllimentTransSize	'0000 0240'H	
'0000 003C'H	Reserved				Reserved
'0000 0038'H	Reserved				
	Product serial	'0000 0120'H		'0000 0220'H	
'0000 0018'H			Support common Parameter		Main device name
'0000 0014'H	Reserved				
'0000 0010'H	DevInfoVersion				
'0000 000C'H	Device version				
'0000 0008'H	Device code				
'0000 0004'H	Vender ID code				
'0000 0000'H	Reserved ('0000 0000'H)	'0000 0100'H		'0000 0200'H	

Figure B.2 – Memory map of device ID area

B.3.2 Detail specifications of device IDs

Table B.2 describes the detail specifications of each device IDs.

Table B.2 – Specifications of device IDs

ID CODE	Contents	Data size	Data type	Provision					
'0001'H	Vendor ID code	4 octets	Unsigned32	Required					
	Description: ID code that identifies the vendor								
'0002'H	Device code	4 octets	Unsigned32	Required					
	Description: code number that represents each device model								
	This number shall be unique to each product series within a vender ID code.								
	The number 'FFFF FFFF'H shall not be specified because of the reserved code for the system use.								
'0003'H	Device version	4 octets	Unsigned32	Required					
	Description: version information of the device								
'0004'H	Version of Device Information file	4 octets	Unsigned32	Required					
	Description: version of the Mechatronic Device Information (MDI) file provided by the device vendor								
	Bit number	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	Version information	Additional code							
	Bit number	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
	Version information	Major version				Minor version			
	Major version: Sequential number updated when substantial changes have been made to the MDI file due to improvement and/or modification of function;								
	Minor version: Sequential number updated when changes have been made to the MDI file due to addition and/or modification of a function in a small scale;								
	Additional code: not a kind of version but a vendor specific code number for a history of the MDI file;								
	bit16~bit31:Reserved(0).								
'0006'H	Serial number of product	32 octets	IA5String (Delimiter '00'H)	Optional					
	Description:serial number that specifies the individual product								
'0010'H	Supported Device Profile 1 (Primary)	4 octets	Unsigned32	Required					
	Description: code of the primary device profile implemented in the device								
	For a device that can accept more than one profile, additional codes can be retrieved from the secondary one ('0012'H) and the third one ('0014'H). In that case, the priority for the profiles will be as shown below:								
	Device Profile 1 (primary) (this ID_CODE);								
Device Profile 2 (secondary) ('0012'H);									
Device Profile 3 (tertiary) ('0014'H).									

ID CODE	Contents	Data size	Data type	Provision					
'0011'H	Supported Profile version 1 (primary)	4 octets	Unsigned32	Required					
	Description: version of the primary device profile								
	For a device that can accept more than one profile, additional versions can be retrieved from the secondary one ('0013'H) and the third one ('0015'H).								
	Bit number	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	Version information	Minor version							
	Bit number	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
	Version information	Major version							
	Major version: sequential number updated when substantial changes have been made to the profile specification								
	Minor version: sequential number updated when changes in a small scale have been made to the profile specification								
	'0012'H	Supported Device profile 2 (secondary)	4 octets	Unsigned32	Required				
		Description: code of the secondary device profile implemented in the device							
If the device can accept only the primary one, the code shall be set to '00FF'H (unsupported code).									
'0013'H	Supported Profile version 2 (secondary)	4 octets	Unsigned32	Required					
	Description: version of the secondary device profile								
	When the device does not handle the device profile 2, the code shall be set to '0000'H.								
'0014'H	Supported Device profile 3 (tertiary)	4 octets	Unsigned32	Required					
	Description: code of the tertiary device profile implemented in the device								
	If the device cannot accept the tertiary one, the code shall be set to '00FF'H (unsupported code).								
'0015'H	Supported Profile version 3 (tertiary)	4 octets	Unsigned32	Required					
	Description: version of the tertiary device profile								
	When the device does not handle the device profile 3, the code shall be set to '0000'H.								
'0018'H	Granularity level of transmission cycle	4 octets	Unsigned32	Required					
	Description: the combination of the step width of the selectable transmission cycle for the device								
	There are four levels as shown in the followings.								
	'0000'H: 15,625/31,25/ 62,5/ 125/ 250/ 500 [μs], and 2 ms to 64 ms by 2 m;								
	'0001'H: 15,625/31,25/ 62,5/ 125/ 250/ 500 [μs], and 1 ms to 64 ms by 1 ms;								
	'0002'H: 15,625/31,25/ 62,5/ 125/ 250/ 500 [μs], and 1 ms to 64 ms by 0,5 ms;								
	'0003'H: 15,625/31,25/ 62,5/ 125/ 250/ 500/ 750 [μs], and 1 ms to 64 ms by 0,5 ms.								
'0020'H	Supported communication mode List	4 octets	BitArray	Required					
	Description: the list of the supported transmission mode and messages communication mode for the device								
	Each communication mode is assigned to a bit as follows (supported: 1, not-supported: 0).								
	bit3		bit2		bit1		bit0		
	Ethernet communication		Message communication		Cyclic communication		Event-driven communication		
	bit4 to bit31:reserved(0)								

ID CODE	Contents	Data size	Data type	Provision				
'0021'H	MAC address	8 octets	Unsigned64	Required				
	Description: ethernet MAC address for the device It is a physical address uniquely assigned to identify each node on the network.							
'0030'H	Supported main command list	32 octets	BitArray	Required				
	Description: the list of supported main commands for the device Each command code is assigned to a corresponding bit position as shown in the following tables. Command codes of bit32 to bit255 are dependent on the selected device profile. Each bit shows 0: not-supported command; 1: supported command.							
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	Reserved (0)	ALM_CL R	ALM_RD	CONFIG	ID_RD	PRM_W R	PRM_RD	NOP
	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
	DISCON NECT	CONNEC T	SYNC _SET	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)
	Bit16 to bit23: reserved (0).							
	Bit31	bit30	bit29	bit28	bit27	bit26	bit25	bit24
	Reserved (0)	MEM_W R	MEM_RD	PPRM_W R	PPRM_R D	Reserved (0)	Reserved (0)	Reserved (0)
	Bit39	bit38	bit37	bit36	bit35	bit34	bit33	bit32
	Reserved (0)	Reserved (0)	Reserved (0)	SENS_O FF	SENS_O N	BRK_OF F	BRK_ON	POS_SE T
	Bit40 to bit47: dependent on the device profile.							
	bit55	bit54	bit53	bit52	bit51	bit50	bit49	bit48
	EX_FEE D	FEED	POSING	INTERP ORATE	Reserved (0)	SV_OFF	SV_ON	SMON
	Bit63	bit62	bit61	bit60	bit59	bit58	bit57	bit56
	Reserved (0)	Reserved (0)	TRQCTR L	VELCTR L	Reserved (0)	ZRET	EX_POSI NG	Reserved (0)
Bit71	bit70	bit69	bit68	bit67	bit66	bit65	bit64	
Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	BRK_OF F	SVPRM_ WR	SVPRM_ RD	
Bit72 to bit95: dependent on the device profile.								
Bit103	bit102	bit101	bit100	bit99	bit98	bit97	bit96	
Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	CF_TRQ CTRL	CF_VEL CTRL	CF_POS CTRL	
Bit104 to bit255: dependent on the device profile.								

ID CODE	Contents	Data size	Data type	Provision				
'0038'H	Supported sub command list	32 octets	BitArray	Required				
	Description: the list of supported subcommands for the device Each command code is assigned to a corresponding bit position as shown in the following tables. Command codes of bit32 to bit255 are dependent on the selected device profile. Each bit shows 0: not-supported command; 1: supported command.							
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	Reserved (0)	ALM_CL R	ALM_RD	Reserved (0)	Reserved (0)	PRM_W R	PRM_RD	NOP
	Bit8 to bit23: reserved (0).							
	bit31	bit30	bit29	bit28	bit27	bit26	bit25	bit24
	Reserved (0)	MEM_W R	MEM_RD	PPRM _WR	PPRM _RD	Reserved (0)	Reserved (0)	Reserved (0)
	Bit32 to bit47: dependent on the device profile.							
	bit55	bit54	bit53	bit52	bit51	bit50	bit49	bit48
	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	SMON
	Bit48 to bit63: dependent on the device profile.							
	bit71	bit70	bit69	bit68	bit67	bit66	bit65	bit64
Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	SVPRM _WR	SVPRM _RD	
Bit72 to bit255: dependent on the device profile.								
'0040'H	Supported common parameter list	32 octets	BitArray	Required				
	Description: the list of supported common parameters for the device Each parameter code is assigned to a corresponding bit position as shown in the following tables. The means of common parameter codes is dependent on the device profile. Each bit shows 0: not-supported parameter code; 1: supported parameter code.							
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	'07'H	'06'H	'05'H	'04'H	'03'H	'02'H	'01'H	Reserved (0)
	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
	Reserved (0)	Reserved (0)	Reserved (0)	'0C'H	'0B'H	'0A'H	'09'H	'08'H
	Bit16 to bit31: reserved (0).							
	bit39	bit38	bit37	bit36	bit35	bit34	bit33	bit32
	'27'H	'26'H	'25'H	'24'H	'23'H	'22'H	'21'H	Reserved (0)
	bit47	bit46	bit45	bit44	bit43	bit42	bit41	bit40
	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	'29'H	'28'H
	Bit48 to bit63: reserved (0).							

ID CODE	Contents			Data size	Data type			Provision
	bit71	bit70	bit69	bit68	bit67	bit66	bit65	bit64
	'47'H	'46'H	'45'H	'44'H	'43'H	'42'H	'41'H	Reserved (0)
	bit79	bit78	bit77	bit76	bit75	bit74	bit73	bit72
	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	'49'H	'48'H
	Bit80 to bit95: reserved (0).							
	bit103	bit102	bit101	bit100	bit99	bit98	bit97	bit96
	'67'H	'66'H	'65'H	'64'H	'63'H	'62'H	'61'H	Reserved (0)
	bit111	bit110	bit109	bit108	bit107	bit106	bit105	bit104
	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)
	Bit112 to bit127: reserved (0).							
	bit135	bit134	bit133	bit132	bit131	bit130	bit129	bit128
	'87'H	'86'H	'85'H	'84'H	'83'H	'82'H	'81'H	Reserved (0)
	bit143	bit142	bit141	bit140	bit139	bit138	bit137	bit136
	'8F'H	'8E'H	'8D'H	'8C'H	'8B'H	'8A'H	'89'H	'88'H
	bit151	bit150	bit149	bit148	bit147	bit146	bit145	bit144
	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	'93'H	'92'H	'91'H	'90'H
	Bit152 to bit158: reserved (0).							
	bit167	bit166	bit165	bit164	bit163	bit162	bit161	bit160
	'A7'H	'A6'H	'A5'H	'A4'H	'A3'H	'A2'H	'A1'H	Reserved (0)
	bit175	bit174	bit173	bit172	bit171	bit170	bit169	bit168
	Reserved (0)	Reserved (0)	Reserved (0)	'AC'H	'AB'H	'AA'H	'A9'H	'A8'H
	bit183	bit182	bit181	bit180	bit179	bit178	bit177	bit176
	'B7'H	'B6'H	'B5'H	'B4'H	'B3'H	'B2'H	'B1'H	Reserved (0)
	bit191	bit190	bit189	bit188	bit187	bit186	bit185	bit184
	Reserved (0)	Reserved (0)	Reserved (0)	'BC'H	'BB'H	'BA'H	'B9'H	'B8'H

ID CODE	Contents	Data size	Data type	Provision				
	bit199	bit198	bit197	bit196	bit195	bit194	bit193	bit192
	'C7'H	'C6'H	'C5'H	'C4'H	'C3'H	'C2'H	'C1'H	Reserved (0)
	bit207	bit206	bit205	bit204	bit203	bit202	bit201	bit200
	Reserved (0)	Reserved (0)	Reserved (0)	'CC'H	'CB'H	'CA'H	'C9'H	'C8'H
	bit215	bit214	bit213	bit212	bit211	bit210	bit209	bit208
	'D7'H	'D6'H	'D5'H	'D4'H	'D3'H	'D2'H	'D1'H	Reserved (0)
	bit223	bit222	bit221	bit220	bit219	bit218	bit217	bit216
	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	'D8'H
	bit231	bit230	bit229	bit228	bit227	bit226	bit225	bit224
	'E7'H	'E6'H	'E5'H	'E4'H	'E3'H	'E2'H	'E1'H	'E0'H
	Bit232 to bit255: reserved (0).							
	'0050'H	Alignment of transmission data in octet units	4 octets	Unsigned32	Required			
Description: this is the alignment of the command / response data corresponding to the device in octet units. Unit [octets]								
'0051'H	Minimum transmittable command data size	4 octets	Unsigned32	Required				
	Description: the octet minimum number of the command data supported by the _FDCServicePDU for the device. Unit [Data alignment]							
'0052'H	Maximum transmittable command data size	4 octets	Unsigned32	Required				
	Description: the octet maximum number of the command data supported by the _FDCServicePDU for the device. Unit [Data alignment]							
'0053'H	Transmittable command data size (current setting)	4 octets	Unsigned32	Required				
	Description: selected octet size of the _FDCServicePDU under the condition of the number of transmittable octets ('0051'H and '0052'H) Unit [Data alignment]							
'0054'H	Minimum transmittable response data size	4 octets	Unsigned32	Required				
	Description: the octet minimum number of the response data supported by the _FDCServicePDU for the device. Unit [Data alignment]							
'0055'H	Maximum transmittable response data size	4 octets	Unsigned32	Required				
	Description: the octet maximum number of the response data supported by the _FDCServicePDU for the device. Unit [Data alignment]							
'0056'H	Transmittable command data size (current setting)	4 octets	Unsigned32	Required				
	Description: selected octet size of the _FDCServicePDU under the condition of the number of transmittable octets ('0054'H and '0055'H) Unit [Data alignment]							

ID CODE	Contents	Data size	Data type	Provision
'0057'H	Minimum transmission cycle	4 octets	Unsigned32	Required
	Description: the minimum value of the selectable transmission cycle by the unit 1 ns for the device under the condition of the granularity level ('0018'H) Example Setting value: 15 625 (decimal) means the minimum transmission cycle: 15,625 s.			
'0058'H	Maximum transmission cycle	4 octets	Unsigned32	Required
	Description: the maximum value of the selectable transmission cycle by the unit 1 ns for the device under the condition of the granularity level ('0018'H) Example Setting value: 8 000 000 (decimal) means the maximum transmission cycle: 8 ms			
'0059'H	Minimum communication cycle	4 octets	Unsigned32	Required
	Description: the minimum value of the selectable communication cycle by the unit 1 ns for the device under the condition of the granularity level ('0018'H) Example Setting value: 15 625 (decimal) means the minimum communication cycle: 15,625 µs.			
'005A'H	Maximum communication cycle	4 octets	Unsigned32	Required
	Description: the maximum value of the selectable communication cycle by the unit 1 ns for the device under the condition of the granularity level ('0018'H) Example Setting value: 8 000 000 (decimal) means the maximum communication cycle: 8 ms.			
'0080'H	Name of main device	32 octets	IA5String (Delimiter '00'H)	Optional
	Description: name of the device with readable characters It means the delegate name of the main device when a physical device contains plural logical APs or subdevices. NOTE A user shall not use this data but the device code (ID CODE: '0002'H) to identify the device type or model.			
'0090'H	Name of subdevice 1	4 octets	Unsigned32	Optional
	Description: name of the first subdevice with readable characters The subdevice is specified in the product specification.			
'0098'H	Version of subdevice 1	32 octets	IA5String (Delimiter '00'H)	Optional
	Description: version information of the first subdevice in the device			
'00A0'H	Name of sub device 2	4 octets	Unsigned32	Optional
	Description: name of the second subdevice with readable characters The subdevice is specified in the product specification.			
'00A8'H	Version of subdevice 2	32 octets	IA5String (Delimiter '00'H)	Optional
	Description: version information of the second subdevice in the device			
'00B0'H	Name of sub device 3	4 octets	Unsigned32	Optional
	Description: name of the third subdevice with readable characters The subdevice is specified in the product specification.			
'00B8'H	Version of subdevice 3	32 octets	IA5String (Delimiter '00'H)	Optional
	Description: name of the third subdevice with readable characters The subdevice is specified in the product specification.			

ID CODE	Contents	Data size	Data type	Provision
'00BC'H to '00BF'H	Reserved			
'00C0'H to '00FF'H	Vendor specific area			

IECNORM.COM : Click to view the full PDF of IEC 61158-6-27:2023

Annex C (informative)

Basic message function

The framework of the message PDUs is described in 4.3.6 (see also below). However, more details of message PDU syntax should be defined by the FAL user in case of the Type 27 FAL.

```

_MSGREQ-PDU      ::= SEQUENCE { responder  Unsigned8 ('00'H..'FE'H),
                                   funcCode  BIT STRING SIZE (7),
                                   reserve1   BIT STRING SIZE (1),
                                   extAddress Unsigned8,
                                   reserve2   OCTET STRING ('00'H),
                                   requestData OCTET STRING SIZE (4..4092)}

_MSGRSP-PDU      ::= SEQUENCE { responder  Unsigned8 ('00'H..'FE'H),
                                   funcCode  BIT STRING SIZE (7),
                                   errorFlag  BIT STRING SIZE (1),
                                   extAddress Unsigned8,
                                   reserve    OCTET STRING ('00'H),
                                   responseData OCTET STRING SIZE (4..4092)}
  
```

In this Annex C, an example command set for function code:'42'H is shown in Table C.1.

Table C.1 – Example of message command set

Function code	Command code	Command description
'42'H	'01'H	To read data from a single memory block
	'02'H	To write data to a single memory block
	'03'H	To read data from multiple memory blocks
	'04'H	To write data to multiple memory blocks
	'06'H	To write data to memory with bit mask pattern

Bibliography

NOTE All parts of the IEC 61158 series, as well as the IEC 61784-1 series and the IEC 61784-2 series are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this bibliography.

IEC 61158-1:2023, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

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

IEC 61158-6 (all parts), *Industrial communication networks – Fieldbus specifications – Part 6-x: Application layer protocol specification*

IEC 61784-1 (all parts), *Industrial networks – Profiles – Part 1-x: Fieldbus profiles – CPF x*

IEC 61784-2 (all parts), *Industrial networks – Profiles – Part 2-x: Additional real-time fieldbus profiles based on ISO/IEC/IEEE 8802-3 – CPF x*

ISO/IEC 8824-1:2021, *Information technology – Abstract Syntax Notation One (ASN-1): Specification of basic notation*

ISO/IEC 8859-1, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*

ISO 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

IECNORM.COM : Click to view the full PDF of IEC 61158-6-27:2023

SOMMAIRE

AVANT-PROPOS	225
INTRODUCTION.....	227
1 Domaine d'application	228
1.1 Généralités	228
1.2 Spécifications	228
1.3 Conformité	229
2 Références normatives	229
3 Termes, définitions, abréviations, symboles et conventions	230
3.1 Termes et définitions référencés	230
3.1.1 Termes et définitions de l'ISO/IEC 7498-1	230
3.1.2 Termes et définitions de l'ISO/IEC 9545	231
3.1.3 Termes et définitions de l'ISO/IEC 8824-1	231
3.1.4 Termes et définitions de l'ISO/IEC 10731	231
3.1.5 Termes et définitions de l'ISO/IEC 19501	231
3.2 Termes et définitions supplémentaires	232
3.3 Abréviations et symboles	241
3.4 Conventions.....	243
3.4.1 Conventions générales	243
3.4.2 Conventions du type de données PDU.....	243
3.4.3 Conventions dans les diagrammes d'états	244
4 Syntaxe abstraite.....	246
4.1 Généralités	246
4.2 Types de données de base	246
4.3 Types FAL PDU	247
4.3.1 Généralités	247
4.3.2 Plans de types APDU: _APDU	250
4.3.3 PDU pour le service de gestion système de terrain	250
4.3.4 Définitions détaillées des _FID-PDU	253
4.3.5 PDU pour le service de commande de l'appareil de terrain	267
4.3.6 PDU pour le service de message	275
4.4 Définitions détaillées des _FDCService-PDUs.....	277
4.4.1 Type de PDU amélioré.....	277
4.5 Profil d'appareil.....	291
5 Syntaxe de transfert	291
5.1 Concepts	291
5.2 Règles de codage	291
5.2.1 INTEGER et ses sous-types	291
5.2.2 Type REAL et ses sous-types	293
5.2.3 Type BIT STRING	294
5.2.4 Type OCTET STRING et type IA5String.....	296
5.2.5 Type NULL	296
5.2.6 Type Structure et type Array	296
6 Structure du diagramme d'états de protocole FAL.....	297
7 Diagramme d'états de contexte AP (APC SM)	299
7.1 Vue d'ensemble	299
7.2 Descriptions des états.....	300

7.3	Événements déclencheurs	301
7.4	Descriptions des actions aux transitions d'état.....	301
8	Machine de protocole de service FAL (FSPM)	303
8.1	Vue d'ensemble	303
8.2	Machine de protocole RT (RT PM)	303
8.2.1	Découverte de la couche de liaison	303
8.2.2	Ponts MAC	303
8.2.3	Ponts virtuels.....	325
8.2.4	Suite IP	325
8.2.5	Machine de protocole de correspondance de DLL (DMPM)	325
8.3	Machine de protocole de gestion d'appareil de terrain (FSM PM)	331
8.3.1	Vue d'ensemble	331
8.3.2	Reconnaissance et configuration de base	333
8.3.3	Démarrage du système	336
8.3.4	Méthodes de synchronisation	365
8.3.5	Entrée à l'état prêt à fonctionner.....	374
8.4	Machine de protocole de commande d'appareil de terrain (FDC PM)	375
8.4.1	Présentation du protocole	375
8.4.2	Mode de communication cyclique	377
8.4.3	Mode de communication déclenchée par les événements	381
8.4.4	Machine de protocole de l'appareil principal (FDCPM-M).....	382
8.4.5	Machine de protocole de l'appareil subordonné (FDCPM-S)	391
8.4.6	Récapitulatif des procédures d'erreur	401
8.5	Machine de protocole de message (MSG PM).....	403
8.5.1	Présentation du protocole	403
8.5.2	Machine de protocole de demandeur (MSGPM-RQ).....	404
8.5.3	Machine de protocole de répondeur (MSGPM-RS).....	407
9	Machine de protocole de relations entre applications (ARPM).....	409
9.1	Généralités	409
9.2	ARPM de l'ASE FDC	409
9.2.1	Vue d'ensemble	409
9.2.2	ARPM de l'appareil principal FDC (ARPM-FDCM).....	411
9.2.3	ARPM de l'appareil subordonné FDC (ARPM-FDCS)	417
9.3	ARPM pour ASE MSG (ARPM-MSG).....	425
9.3.1	Descriptions des états	425
9.3.2	Événements déclencheurs	425
9.3.3	Descriptions des actions aux transitions d'état.....	426
10	Machines de protocole de correspondance de DLL (DMPM)	427
	Annexe A (informative) Profil d'appareil et jeux d'instructions FDC	428
	Annexe B (normative) Espace en mémoire virtuelle et informations relatives aux appareils.....	430
B.1	Vue d'ensemble	430
B.2	Communication Data Object (Objet de données de communication).....	431
B.3	Informations relatives à l'appareil.....	431
B.3.1	Structure de la zone de l'identifiant de l'appareil.....	431
B.3.2	Spécifications particulières des ID d'appareil	433
	Annexe C (informative) Fonction de message de base	440
	Bibliographie.....	441

Figure 1 – Codage des sous-types Integer	292
Figure 2 – Exemple de transfert de la valeur INTEGER	292
Figure 3 – Codage des sous-types Unsigned	293
Figure 4 – Codage du type Float32	293
Figure 5 – Codage du type Float64	294
Figure 6 – Exemple de définition de champ binaire avec bits nommés	295
Figure 7 – Exemple de définition de champ binaire avec taille de champ	295
Figure 8 – Codage du type SEQUENCE	296
Figure 9 – Structure des diagrammes d'états de protocole FAL	298
Figure 10 – Diagramme d'états de l'APC SM	300
Figure 11 – Schéma de transition d'états de la CTC de l'appareil principal	305
Figure 12 – Schéma de transition d'états de la CTC de l'appareil subordonné	315
Figure 13 – Structure des machines de protocole à l'intérieur de la DMPM (pont)	326
Figure 14 – Schéma de transition d'états de la SRC	326
Figure 15 – Exemple de configuration de réseau	331
Figure 16 – Exemple de configuration de réseau	332
Figure 17 – Séquence de notification du nœud voisin	333
Figure 18 – Séquence de détection de nœud connecté	335
Figure 19 – Exemple d'informations relatives au nœud voisin	335
Figure 20 – Schéma de transition d'états FSMUL pour l'appareil principal avec fonction SM	336
Figure 21 – Schéma de transition d'états FSMUL pour l'appareil principal sans fonction SM	349
Figure 22 – Schéma de transition d'états FSMUL pour un appareil subordonné	358
Figure 23 – Séquence de notification de temps de retard SM par écriture de CDO	366
Figure 24 – Séquence de mesure du retard (SM: BM)	368
Figure 25 – Séquence de mesure du retard de l'appareil principal pour le BM (SM: appareil principal avec fonction appareil principal de bande (BM))	369
Figure 26 – Séquence de mesure de retard pour l'appareil principal autre que le BM (SM: BM)	370
Figure 27 – Séquence de mesure du retard (SM: S N° 1)	372
Figure 28 – Séquence de mesure du retard de l'appareil principal pour le BM (SM: S N° 1)	373
Figure 29 – Séquence d'entrée à l'état prêt à fonctionner	375
Figure 30 – Exemple de cycle de communication de l'AP d'appareil principal FDC	378
Figure 31 – Exemple de cycle de communication de l'AP d'appareil subordonné FDC	378
Figure 32 – Communication de l'instruction synchrone à l'état de synchronisation	379
Figure 33 – Communication de l'instruction asynchrone dans l'état sync	380
Figure 34 – Communication de l'instruction asynchrone à l'état async	381
Figure 35 – Communication déclenchée par les événements	382
Figure 36 – Diagramme d'états de FDCPM-M	383
Figure 37 – Diagramme d'états de la FDCPM-S	391
Figure 38 – Flux de transmission PDU pour le message utilisateur	403
Figure 39 – Flux de transmission PDU pour message à sens unique	404

Figure 40 – Diagramme d'états de MSGPM-RQ	405
Figure 41 – Diagramme d'états de MSGPM-RS.....	407
Figure 42 – Exemple de processus de transfert unique	410
Figure 43 – Exemple de processus de transfert double	410
Figure 44 – Diagramme d'états de ARPM-FDCM	411
Figure 45 – Schéma d'états de l'ARPM-FDCS.....	418
Figure 46 – Schéma d'états de l'ARPM-MSG	425
Figure B.1 – Image mémoire de l'espace mémoire virtuel	430
Figure B.2 – Image mémoire de la zone d'ID d'appareil	432
Tableau 1 – Descriptions de transition d'état.....	244
Tableau 2 – Description des éléments de diagramme d'états	244
Tableau 3 – Conventions utilisées dans les diagrammes d'états	245
Tableau 4 – Adresse de multidiffusion	248
Tableau 5 – Length_or_type	248
Tableau 6 – En-tête IPv4	249
Tableau 7 – En-tête IPv6	249
Tableau 8 – En-tête UDP	249
Tableau 9 – Adresse du nœud	252
Tableau 10 – Mise en correspondance des diagrammes d'états de protocole.....	298
Tableau 11 – Descriptions des états de l'APC SM.....	300
Tableau 12 – Événements déclencheurs de l'APC SM	301
Tableau 13 – Transitions de l'APC SM	302
Tableau 14 – Table de transition d'états de la CTC de l'appareil principal.....	306
Tableau 15 – Tableau de transition d'états de la CTC de l'appareil subordonné.....	316
Tableau 16 – Liste des macros de la CTC.....	320
Tableau 17 – Primitives et paramètres pour l'interface FDC émise par la FDC.....	323
Tableau 18 – Primitives et paramètres pour l'interface FSM émise par la CTC.....	324
Tableau 19 – Liste des primitives et des paramètres (source FSM).....	324
Tableau 20 – Liste des primitives et des paramètres (source FSMUL)	325
Tableau 21 – Table de transition d'états de la SRC.....	327
Tableau 22 – Liste des macros de la SRC	329
Tableau 23 – Liste des fonctions de la SRC.....	330
Tableau 24 – Primitives et paramètres pour l'interface SRC-CTC	330
Tableau 25 – Primitives et paramètres d'envoi de trame	330
Tableau 26 – Primitives et paramètres de réception de trame	331
Tableau 27 – Primitives et paramètres du service de répétition de la sélection	331
Tableau 28 – Table de transition d'états FSMUL pour l'appareil principal avec fonction SM	337
Tableau 29 – Table de transition d'états FSMUL pour l'appareil principal sans fonction SM	350
Tableau 30 – Table de transition d'états FSMUL pour un appareil subordonné	358
Tableau 31 – Liste des macros de la FSMUL	363
Tableau 32 – Mode de protocole FDC.....	376

Tableau 33 – Descriptions des états de la FDCPM-M.....	383
Tableau 34 – Descriptions des événements déclencheurs de la FDCPM-M.....	385
Tableau 35 – Transitions du diagramme d'états principal de la FDCPM-M	386
Tableau 36 – Transitions de la sous-machine de la FDCPM-M.....	388
Tableau 37 – Descriptions des états de la FDCPM-S	391
Tableau 38 – Descriptions des événements déclencheurs de la FDCPM-S	393
Tableau 39 – Transitions du diagramme d'états principal de la FDCPM-S.....	394
Tableau 40 – Transitions de la sous-machine de la FDCPM-S	396
Tableau 41 – Descriptions des états de la MSGPM-RQ.....	405
Tableau 42 – Descriptions des événements déclencheurs de MSGPM-RQ.....	406
Tableau 43 – Transitions de la MSGPM-RQ.....	406
Tableau 44 – Descriptions des états de la MSGPM-RS.....	407
Tableau 45 – Descriptions des événements déclencheurs de MSGPM-RS.....	408
Tableau 46 – Transitions de la MSGPM-RS	408
Tableau 47 – Descriptions des états de l'ARPM-FDCM.....	412
Tableau 48 – Descriptions des événements déclencheurs de l'ARPM-FDCM	414
Tableau 49 – Transitions du diagramme d'états principal de l'ARPM-FDCM.....	415
Tableau 50 – Transitions de la sous-machine de l'ARPM-FDCM	416
Tableau 51 – Descriptions des états de l'ARPM-FDCS.....	418
Tableau 52 – Descriptions des événements déclencheurs de l'ARPM-FDCS.....	421
Tableau 53 – Transitions du diagramme d'états principal de l'ARPM-FDCS	422
Tableau 54 – Transitions de la sous-machine de l'ARPM-FDCS.....	423
Tableau 55 – Descriptions des états de la ARPM-MSG.....	425
Tableau 56 – Descriptions des événements déclencheurs de ARPM-MSG.....	426
Tableau 57 – Transitions de l'ARPM-MSG	426
Tableau A.1 – Exemple de profils d'appareil enregistrés	428
Tableau A.2 – Exemple de liste d'instructions du profil '00'H.....	429
Tableau B.1 – Image mémoire de la zone CDO.....	431
Tableau B.2 – Spécifications des ID d'appareil	433
Tableau C.1 – Exemple de jeu d'instructions de message.....	440

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

**RÉSEAUX DE COMMUNICATION INDUSTRIELS –
SPÉCIFICATIONS DES BUS DE TERRAIN –****Partie 6-27: Spécification du protocole de la couche application –
Éléments de type 27****AVANT-PROPOS**

- 1) La Commission Électrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. À cet effet, l'IEC – 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 l'IEC"). 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 l'IEC, participent également aux travaux. L'IEC 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 l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC 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 l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) L'IEC 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 l'IEC. L'IEC 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 l'IEC, 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 l'IEC, 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 l'IEC ou de toute autre Publication de l'IEC, 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.
- 9) L'attention est attirée sur le fait que certains des éléments de la présente publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets.

L'attention est attirée sur le fait que l'utilisation du type de protocole associé est restreinte par les détenteurs des droits de propriété intellectuelle. En tout état de cause, l'engagement de renonciation partielle aux droits de propriété intellectuelle pris par les détenteurs de ces droits autorise l'utilisation d'un type de protocole de couche avec les autres protocoles de couche du même type, ou dans des combinaisons avec d'autres types autorisées explicitement par les détenteurs des droits de propriété intellectuelle pour ce type.

NOTE Les combinaisons de types de protocoles sont spécifiées dans la série IEC 61784–1 et la série IEC 61784–2.

L'IEC 61158-6-27 a été établie par le sous-comité 65C: Réseaux industriels, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels. Il s'agit d'une Norme internationale.

Le texte de cette Norme internationale est issu des documents suivants:

Projet	Rapport de vote
65C/1205/FDIS	65C/1234/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à son approbation.

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

La langue employée pour l'élaboration de cette Norme internationale est l'anglais.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2, il a été développé selon les Directives ISO/IEC, Partie 1 et les Directives ISO/IEC, Supplément IEC, disponibles sous www.iec.ch/members_experts/refdocs. Les principaux types de documents développés par l'IEC sont décrits plus en détail sous www.iec.ch/publications.

Une liste de toutes les parties de la série IEC 61158, publiées sous le titre général *Réseaux de communication industriels – Spécifications des bus de terrain*, se trouve sur le site Web de l'IEC.

Le comité a décidé que le contenu de ce document ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous webstore.iec.ch dans les données relatives au document recherché. A cette date, le document sera

- reconduit,
- supprimé,
- remplacé par une édition révisée, ou
- amendé.

INTRODUCTION

Le présent document fait partie d'une série élaborée pour faciliter l'interconnexion des composants de systèmes d'automatisation. Il est lié aux autres normes de la série telle que définie par le modèle de référence de bus de terrain "à trois couches" décrit dans l'IEC 61158-1.

Le protocole d'application fournit le service d'application au moyen des services disponibles au niveau de la couche liaison de données ou de la couche immédiatement inférieure. Le principal objectif du présent document est de définir un ensemble de règles de communication, exprimées en matière de procédures qu'ont à suivre les entités d'application (AE, Application Entity) homologues au moment de la communication. Ces règles de communication sont destinées à fournir une base de développement solide de façon à atteindre plusieurs objectifs:

- servir de guide aux intégrateurs et aux concepteurs;
- être appliquées dans le cadre des essais et de l'achat d'équipements;
- être incorporées dans un accord sur l'accès des systèmes à l'environnement de systèmes ouverts;
- affiner la compréhension des communications à temps critique au sein de l'OSI.

Le présent document porte en particulier sur la communication et l'interfonctionnement des capteurs, des effecteurs et d'autres appareils d'automatisation. Au moyen de l'utilisation conjointe du présent document avec d'autres documents entrant dans les modèles de référence OSI ou de bus de terrain, des systèmes autrement incompatibles peuvent fonctionner dans toute combinaison.

La Commission Électrotechnique Internationale (IEC) attire l'attention sur le fait qu'il est déclaré que la conformité avec les dispositions du présent document peut impliquer l'utilisation d'un brevet. L'IEC ne prend pas position quant à la preuve, à la validité et à la portée de ces droits de propriété.

Le détenteur de ces droits de propriété a donné l'assurance à l'IEC qu'il consent à négocier des licences avec des demandeurs du monde entier à des termes et conditions raisonnables et non discriminatoires. A ce propos, la déclaration du détenteur des droits de propriété est enregistrée à l'IEC. Des informations peuvent être obtenues dans la base de données des droits de propriété, disponible à l'adresse suivante: <http://patents.iec.ch>.

L'attention est attirée sur le fait que certains des éléments du présent document peuvent faire l'objet de droits de propriété autres que ceux figurant dans la base de données des brevets. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et averti de leur existence.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-27: Spécification du protocole de la couche application – Éléments de type 27

1 Domaine d'application

1.1 Généralités

La couche application de bus de terrain (FAL, Fieldbus Application Layer) fournit aux programmes d'utilisateur un moyen d'accéder à l'environnement de communication du bus de terrain. A cet égard, la FAL peut être considérée comme une "fenêtre entre programmes d'application correspondants".

La présente partie de l'IEC 61158 donne les éléments communs visant à assurer les communications de messagerie de base à temps critique et à temps non critique entre les programmes d'application d'un environnement d'automatisation et d'un matériel spécifique au bus de terrain de type 27. Le terme "à temps critique" est utilisé pour indiquer la présence d'une fenêtre temporelle, dans laquelle il est exigé de réaliser une ou plusieurs actions spécifiées selon un niveau défini de certitude. Tout manquement à réaliser ces actions dans la fenêtre de temps prévue risque de provoquer la défaillance des applications qui les demandent, avec le risque de mettre en danger l'équipement, l'usine, voire les personnes.

Le présent document définit de manière abstraite les caractéristiques visibles de l'extérieur fournies par la couche application de bus de terrain de type 27 en ce qui concerne:

- la syntaxe abstraite définissant les unités de données du protocole de la couche application transmises entre les entités d'application de communication;
- la syntaxe de transfert définissant les unités de données du protocole de la couche application transmises entre les entités d'application de communication;
- les diagrammes d'états de contexte d'application définissant le comportement de service d'application observable entre les entités d'application en communication; et
- les diagrammes d'états de relations entre applications définissant le comportement de communication visible entre les entités d'application en communication.

Le but du présent document est de définir le protocole permettant de:

- définir la représentation filaire des primitives de service définies dans l'IEC 61158-5-27; et
- définir le comportement visible de l'extérieur associé à leur transfert.

Le présent document spécifie le protocole de la couche application de bus de terrain de type 27, conformément au modèle de référence de base de l'OSI (ISO/IEC 7498-1) et à la structure de la couche application de l'OSI (ISO/IEC 9545).

1.2 Spécifications

L'objet principal du présent document est de spécifier la syntaxe et le comportement du protocole de la couche application qui achemine les services de couche application définis dans l'IEC 61158-5-27.

Un objectif secondaire est de fournir des chemins de migration à partir de protocoles de communication industriels préexistants. C'est ce dernier objectif qui donne lieu à la diversité des protocoles normalisés dans la série IEC 61158-6.

1.3 Conformité

Le présent document ne spécifie aucune mise en œuvre ou aucun produit individuel, de même qu'elle ne restreint nullement les mises en œuvre des entités de couche application dans les systèmes d'automatisation industriels.

La conformité est assurée par la mise en œuvre de la présente spécification du protocole de la couche application.

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences 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).

NOTE Toutes les parties de la série IEC 61158, ainsi que la série IEC 61784-1 et la série IEC 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent par conséquent aux éditions datées dans la présente liste de références normatives.

IEC 61158-5-27:2023, *Réseaux de communication industriels - Spécifications des bus de terrain – Partie 5-27: Définition des services de la couche application – Eléments de type 27*

ISO/IEC 646, *Technologies de l'information – Jeu ISO de caractères codés à 7 éléments pour l'échange d'information*

ISO/IEC 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: – Partie 1: Le modèle de base*

ISO/IEC 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche application*

ISO/IEC 9899, *Technologies de l'information – Langages de programmation – C*

ISO/IEC 10731, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base – Conventions pour la définition des services OSI*

ISO/IEC 19501:2005, *Technologies de l'information – Traitement distribué ouvert – Langage de modélisation unifié (UML), version 1.4.2*

ISO/IEC/IEEE 80559:2020, *Technologies de l'information – Systèmes de microprocesseurs – Arithmétique flottante*

ISO/IEC/IEEE 8802-3, *Télécommunications et échange entre systèmes informatiques – Exigences pour les réseaux locaux et métropolitains – Partie 3: Norme pour Ethernet*

IEEE 802.1D, *IEEE Standards for local and metropolitan area networks – Media Access Control (MAC) Bridges* (disponible en anglais seulement)

IEEE 802.1Q, *IEEE Standards for local and metropolitan area networks – Bridges and Bridged Networks* (disponible en anglais seulement)

IEEE 802.1AB, *IEEE Standards for Local and Metropolitan Area Networks: Station and Media Access Control Connectivity Discovery* (disponible en anglais seulement)

IEEE 802.1AS, *IEEE Standards for Local and Metropolitan Area Networks: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks* (disponible en anglais seulement)

IETF RFC 768, J. Postel, "*User Datagram Protocol*", août 1980, disponible à l'adresse <<https://www.rfc-editor.org/info/rfc768>> (disponible en anglais seulement) [vue 16/09/2022]

IETF RFC 791, J. Postel, "*Internet Protocol*", septembre 1981, disponible à l'adresse <<https://www.rfc-editor.org/info/rfc791>> (disponible en anglais seulement) [vue 16/09/2022]

IETF RFC 826, D. Plummer, "*An Ethernet Address Resolution Protocol: or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*", novembre 1982, disponible à l'adresse <<https://www.rfc-editor.org/info/rfc826>> (disponible en anglais seulement) [vue 16/09/2022]

IETF RFC 1112, S.E. Deering, "*Host Extensions for IP Multicasting*", août 1989, disponible à l'adresse <<https://www.rfc-editor.org/info/rfc1112>> (disponible en anglais seulement) [vue 16/09/2022]

IETF RFC 2460, S. Deering and R. Hinden, "*Internet Protocol, Version 6 (IPv6) Specification*", décembre 1998, disponible à l'adresse <<https://www.rfc-editor.org/info/rfc2460>> (disponible en anglais seulement) [vue 16/09/2022]

3 Termes, définitions, abréviations, symboles et conventions

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

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse <https://www.electropedia.org/>
- ISO Online browsing platform: disponible à l'adresse <https://www.iso.org/obp>

3.1 Termes et définitions référencés

3.1.1 Termes et définitions de l'ISO/IEC 7498-1

Pour les besoins du présent document, les termes suivants définis dans l'ISO/IEC 7498-1 s'appliquent:

- syntaxe abstraite;
- entité d'application;
- processus d'application;
- unité de données de protocole d'application;
- invocation de processus d'application;
- fonctionnalité (N);
- fonction (N);
- entités (N) correspondantes;
- contexte de présentation;
- système réel;
- syntaxe de transfert.

3.1.2 Termes et définitions de l'ISO/IEC 9545

Pour les besoins du présent document, les termes suivants définis dans l'ISO/IEC 9545 s'appliquent:

- association d'applications;
- contexte d'application;
- invocation d'entité d'application;
- type d'entité d'application;
- élément de service d'application.

3.1.3 Termes et définitions de l'ISO/IEC 8824-1

Pour les besoins du présent document, les termes et les définitions de l'ISO/IEC 8824-1 s'appliquent.

- type simple;
- composant;
- type de composant;
- type entier;
- type bitstring;
- type octetstring;
- type nul;
- type séquence;
- séquence de types;
- type de choix;
- type IA5String;
- codage.

3.1.4 Termes et définitions de l'ISO/IEC 10731

Pour les besoins du présent document, les termes suivants définis dans l'ISO/IEC 10731 s'appliquent:

- primitive de service OSI; primitive;
- fournisseur de service OSI; fournisseur;
- utilisateur de service OSI; utilisateur.

3.1.5 Termes et définitions de l'ISO/IEC 19501

Pour les besoins du présent document, les termes suivants définis dans l'ISO/IEC 19501 s'appliquent:

- événement;
- état;
- diagramme d'états;
- sous-état;
- sous-machine;
- transition.

3.2 Termes et définitions supplémentaires

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

3.2.1 alarme

statut d'appareil de terrain indiquant, par l'intermédiaire du service de commande de l'appareil de terrain (FDC) du bus de terrain de type 27, que l'appareil a détecté un problème grave à résoudre et qu'il ne peut pas continuer à fonctionner normalement

Note 1 à l'article: Tous les états d'alarme sont verrouillés. Une intervention est nécessaire pour les effacer.

Note 2 à l'article: Les alarmes sont classées en trois groupes: alarmes de communication, alarmes associées à une instruction illégale et alarmes spécifiques à l'application. Cependant, les définitions concrètes dépendent de la mise en œuvre de chaque appareil de terrain.

3.2.2 objet de processus d'application

représentation réseau d'un aspect particulier d'un processus d'application (AP), modélisée sous la forme d'un objet accessible sur le réseau présent dans un AP ou un autre APO

Note 1 à l'article: Voir l'IEC 61158-1, 9.3.4.

3.2.3 contexte de processus d'application contexte d'AP

connaissances partagées ou ensemble de règles commun, régissant la communication des entités d'application (AE) de la FAL et décrivant le comportement admis en matière de communications collectives entre les AE qui font partie d'un ensemble spécifique de relations entre applications (AR)

Note 1 à l'article: Les données du contexte AP peuvent être spécifiées au préalable par l'utilisateur, au moyen d'une option sélectionnée alors que l'utilisateur a recours à un service de gestion de bus de terrain (FSM) pour lire la fonctionnalité d'AP homologue, par la fonction de négociation automatique que le système FSM gère, etc. La méthode qui doit être adoptée dépend de la spécification de chaque mise en œuvre.

3.2.4 type de processus d'application Type d'AP

description d'une classification de processus d'application (AP) en termes d'ensemble de capacités pour la FAL du bus de terrain de type 27

Note 1 à l'article: Les types d'AP sont classés en deux catégories: AP d'appareil principal C1 et AP d'appareil subordonné, en fonction de leurs rôles d'application dans le réseau de bus de terrain.

3.2.5 instruction asynchrone

type d'unité de données de protocole d'application (APDU) d'instruction du service FDC de la FAL de type 27, qui peut être émis à tout moment à l'issue de la transaction précédente sans tenir compte de la synchronisation avec le cycle de communication

Note 1 à l'article: Les définitions indiquant s'il convient qu'une instruction soit de nature asynchrone ou synchrone dépendent d'une application. Elles peuvent être fournies en tant qu'ensemble enregistré d'instructions et de réponses ou en tant que profils d'appareil, voir l'Annexe A.

3.2.6 communication asynchrone

état ou moyen de communication du service FDC de la FAL de type 27, dans lequel une instruction peut être émise à tout moment à l'issue de la transaction précédente sans tenir compte de la synchronisation avec le cycle de communication

Note 1 à l'article: Dans cet état, les instructions synchrones ne peuvent pas être émises, à l'inverse des instructions asynchrones.

3.2.7**attribut**

information ou paramètre contenu(e) dans les parties variables d'un objet

Note 1 à l'article: En règle générale, ils donnent des informations d'état ou gouvernent le fonctionnement d'un objet. Les attributs ont également un impact sur le comportement d'un objet.

3.2.8**appareil principal C1**

type d'AP doté des fonctionnalités d'appareil principal pour le service FDC de la FAL de type 27 ou appareil qui met en œuvre ce type d'AP

Note 1 à l'article: Un réseau du bus de terrain de type 27 ne peut contenir qu'un seul appareil principal C1

3.2.9**instruction**

PDU émise par un demandeur ou un appareil principal pour qu'un destinataire ou un appareil subordonné exécute certaines fonctions

3.2.10**communication**

processus permettant l'échange formel d'informations entre au moins deux appareils, utilisateurs, AP ou entités

3.2.11**transfert**

processus d'acheminement d'une PDU d'un émetteur à un destinataire

3.2.12**transmission**

processus permettant d'envoyer et de propager des signaux électriques ou des données codées

3.2.13**cycle de communication**

période d'activités répétitives, synchronisée avec le cycle de transmission, simultanée à l'établissement de la connexion pour le protocole FDC de la FAL de type 27

Note 1 à l'article: Le cycle de communication peut être synchronisé avec le cycle de transmission multiplié par un facteur d'échelle spécifié.

3.2.14**connexion**

contexte ou liaison logique, dans des conditions particulières, pour le protocole FDC entre un objet principal et un objet subordonné pour la FAL de type 27

3.2.15**cyclique**

répétitif de manière régulière

3.2.16**communication cyclique**

mode de transmission dans le cadre duquel des PDU de demande et des PDU de réponse sont échangées de façon répétitive, dans les créneaux prévus, synchronisés avec un cycle de transmission du type 27

Note 1 à l'article: Dans l'AL, le cycle de communication provient du cycle de transmission de ce mode.

3.2.17

compteur d'échelle de cycle

compteur permettant de générer un cycle de communication en procédant à la mise à l'échelle d'un cycle primaire ou d'un cycle de transmission

3.2.18

ID d'appareil

partie des "informations relatives à l'appareil" permettant d'identifier l'appareil pour un type ou un modèle de produit spécifique du bus de terrain de type 27

3.2.19

informations relatives à l'appareil

informations formatées et intégrées à un appareil permettant de caractériser ce dernier; elles comprennent principalement des données destinées à l'identification du modèle et des paramètres spécifiques au profil de l'appareil pour le bus de terrain de type 27

3.2.20

profil d'appareil

ensemble d'informations et de fonctionnalités communes au modèle d'appareil assurant la cohérence entre les différents modèles parmi un même type d'appareil

3.2.21

transfert double

mode de transfert pour le protocole FDC de la FAL de type 27, dans le cadre duquel un émetteur envoie une même PDU deux fois par transaction et un destinataire l'utilise pour détecter et résoudre une erreur de communication telle qu'une corruption de données ou une perte de données en mode de communication cyclique

3.2.22

communication déclenchée par les événements

mode de transmission pour le protocole de couche inférieure du bus de terrain de type 27 dans le cadre duquel une transaction instruction-réponse-échange a lieu à chaque demande de l'utilisateur

Note 1 à l'article: Le cycle de transmission et le cycle de communication ne surviennent pas dans ce mode.

3.2.23

erreur

condition anormale ou dysfonctionnement de la communication ou de toute autre activité

3.2.24

commande d'appareil de terrain

service FDC

service de communication à temps critique qui gère des données d'instruction à longueur fixe dans le but de contrôler un appareil de terrain et les données de réponse de retour correspondantes avec une restriction stricte du retard ou de l'instabilité pour le cadencement de la communication de la FAL de type 27

3.2.25

protocole d'appareil de terrain

protocole FDC

protocole de communication à temps critique qui gère des données d'instruction à longueur fixe dans le but de contrôler un appareil de terrain et les données de réponse de retour correspondantes avec une restriction stricte du retard ou de l'instabilité pour le cadencement de la communication de la FAL de type 27

3.2.26**appareil principal**

classe ou objet d'instance d'élément de service d'application (ASE) FDC qui joue un rôle de demandeur d'instruction pour la FAL de type 27

Note 1 à l'article: Il existe une ou plusieurs unités au sein du réseau en tant que nœud pour envoyer des données de sortie (données d'instruction) aux appareils subordonnés et recevoir des données d'entrée (données de réponse) des appareils subordonnés de la couche liaison de données. Un appareil principal exécute également l'envoi/la réception de données de message vers/depuis les appareils subordonnés. Il peut exécuter les deux opérations, ou alors l'une ou l'autre, d'envoi/réception de données d'entrée/sortie et d'envoi/réception de données de message.

Note 2 à l'article: L'envoi/la réception de données d'entrée/de sortie et l'envoi/la réception de données de message doivent être effectués à un cycle constant et dans une bande constante.

Note 3 à l'article: Ou alors il s'agit d'un type d'application qui exécute la commande d'appareil de terrain dans la couche application.

3.2.27**service de message****services MSG**

service de communication qui gère les données de longueur variable et n'exigeant pas de limitation rigoureuse quant au temps de réponse

3.2.28**horloge réseau**

compteur synchronisé, à fonctionnement périodique dont chaque nœud du réseau est doté, qui devient une source d'oscillation du cycle de transmission

3.2.29**cycle primaire**

période d'activités répétitives, synchronisée avec le cycle de transmission, avant l'établissement de la connexion pour le protocole FDC de la FAL de type 27

3.2.30**machine de protocole**

diagramme d'états qui réalise un protocole comme fonction principale de l'entité de chaque couche

3.2.31**demandeur**

classe ou objet d'instance d'ASE MSG qui joue un rôle de demandeur ou d'émetteur d'instruction pour la FAL de type 27

3.2.32**répondeur**

classe ou objet d'instance d'ASE MSG qui joue un rôle de répondeur ou de destinataire d'instruction pour la FAL de type 27

3.2.33**réponse**

PDU émise par un répondeur ou un appareil subordonné pour informer d'un résultat ou d'un état pour l'instruction reçue par un demandeur ou un appareil principal

3.2.34**service**

opération ou processus qu'un objet exécute à la demande d'un autre objet

3.2.35**transfert unique**

mode de transfert normal pour le protocole FDC de la FAL de type 27 dans le cadre duquel un émetteur envoie une PDU une fois par transaction

3.2.36

AP d'appareil subordonné

type d'AP qui dispose de fonctionnalités subordonnées pour le service FDC de la FAL de type 27 ou l'appareil de mise en œuvre de ce type d'AP

3.2.37

appareil subordonné

classe ou objet d'instance d'ASE FDC qui joue un rôle de répondeur pour la FAL de type 27

Note 1 à l'article: Un nœud pour recevoir des données de sortie (données d'instruction) provenant d'un appareil principal et envoyer des données d'entrée (données de réponse) à l'appareil principal qui a envoyé les données de sortie dans la couche liaison de données. Un appareil subordonné exécute également l'envoi/la réception de données de message vers/depuis l'appareil principal. Il peut exécuter l'envoi/la réception de données d'entrée/sortie et de données de message vers/depuis plusieurs appareils principaux. De plus, il peut recevoir des données d'entrée d'autres appareils subordonnés.

3.2.38

instruction synchrone

type d'instruction ou APDU du service FDC de la FAL de type 27, émis de manière synchronisée avec chaque cycle de communication

Note 1 à l'article: Les définitions, que l'instruction soit synchrone ou pas, dépendent d'une application. Elles peuvent être fournies en tant qu'ensemble enregistré d'instructions et de réponses ou en tant que profils d'appareil, voir l'Annexe A.

3.2.39

communication synchrone

état ou méthode de communication du service FDC de la FAL de type 27, dans le cadre duquel/de laquelle une instruction est émise de manière synchronisée avec chaque cycle de communication

Note 1 à l'article: Dans cet état, les instructions synchrones et asynchrones peuvent être émises.

Note 2 à l'article: Dans cet état, une erreur de désynchronisation des AP doit être détectée par les mesures du compteur de l'horloge de surveillance.

3.2.40

cycle de transmission

période d'activités répétitives pour les couches inférieures du bus de terrain de type 27; tous les appareils subordonnés sont synchronisés avec celle d'un appareil principal C1 par le protocole de couche inférieure

Note 1 à l'article: Cycle dans lequel l'appareil principal de synchronisation envoie un paquet de synchronisation. L'appareil principal ayant la fonction appareil principal de bande informe chaque nœud du réseau. Il est nécessaire que la bande de synchronisation, la bande de transmission du domaine de commande, la bande de transmission IP et la bande de garde fassent partie du cycle de transmission. Le point de départ du cycle de transmission est géré par l'appareil principal de synchronisation et est informé par le paquet de synchronisation à tous les nœuds. Entre l'appareil principal et l'appareil subordonné dans le domaine de commande, ou dans la couche liaison de données entre l'appareil principal ayant la fonction appareil principal de bande et l'appareil principal dénué de cette fonction, les transmissions cycliques peuvent être répétées à un cycle de n fois (n est un entier supérieur ou égal à zéro) le cycle de transmission et il peut varier d'un appareil subordonné à un autre appareil subordonné.

3.2.41

mode de transmission

état ou méthode de transmission du protocole de la couche inférieure du bus de terrain de type 27; mode cyclique, mode déclenchement par un événement

3.2.42

espace de mémoire virtuelle

gros bloc de données d'APO pour la FAL de type 27 qui peut être lu et sur lequel il est possible d'écrire avec des pseudo-adresses de mémoire afin de proposer une certaine cohérence entre différents modèles d'appareils

Note 1 à l'article: L'espace de mémoire virtuelle comprend les informations de l'appareil et une zone spécifique au fournisseur, voir l'Annexe B.

3.2.43**avertissement**

statut d'appareil de terrain indiquant, par l'intermédiaire du service de commande de l'appareil de terrain (FDC) du bus de terrain de type 27, que l'appareil a détecté un problème léger ou passager, mais continue à fonctionner normalement

Note 1 à l'article: Tous les états d'avertissement sont verrouillés. Une intervention est nécessaire pour les effacer.

Note 2 à l'article: Les avertissements sont classés en trois groupes: avertissements de communication, avertissements associés à une instruction illégale et avertissements spécifiques à l'application. Cependant, les définitions concrètes dépendent de la mise en œuvre de chaque appareil de terrain.

3.2.44**nœud**

appareil qui est conforme aux protocoles décrits dans la présente spécification

3.2.45**type de nœud**

appareil principal ou appareil subordonné

3.2.46**système à domaines de commande multiples**

configuration dans laquelle il existe plusieurs domaines de commande dans le réseau

3.2.47**nœud connecté**

désignation commune de l'appareil principal et des appareils subordonnés connectés au même réseau

3.2.48**nœud primaire****initiateur**

nœud pour lancer les transmissions cycliques et les transmissions de messages

Note 1 à l'article: Seul l'appareil principal sert de nœud primaire

3.2.49**nœud secondaire****répondeur**

nœud chargé d'exécuter les transmissions cycliques et les transmissions de messages conformément à l'instruction du nœud primaire

3.2.50**données de sortie****données d'instruction**

données à envoyer d'un appareil principal à un appareil subordonné

3.2.51**données d'entrée****données de réponse**

données à envoyer d'un appareil subordonné à un appareil principal

3.2.52**données d'entrée/sortie**

données d'entrée et données de sortie

3.2.53**données de commande**

désignation commune des données de sortie et des données d'entrée

3.2.54

données de message

données transmises séparément des données de commande, qui sont échangées entre un appareil principal et un appareil subordonné et envoyées à la demande de l'application

3.2.55

transmission cyclique

échange cyclique de données de commande entre un appareil principal et un appareil subordonné

3.2.56

transmission de messages

échange de données de message pour transmettre des informations séparément des données de commande, qui n'est, contrairement à la transmission cyclique, conduite que lorsque l'application l'exige

3.2.57

transmission IP

communication utilisant des paquets IP

3.2.58

domaine de commande

groupe logique constitué d'un appareil principal et d'appareils subordonnés pour échanger des données de commande et/ou des données de message

Note 1 à l'article: Plusieurs domaines de commande peuvent exister dans le réseau et, dans ce cas, ils sont indiqués comme domaine de commande N° n (n est un entier)

Note 2 à l'article: Un appareil subordonné doit appartenir à un ou plusieurs domaines de commande.

3.2.59

bande de synchronisation

bande pour transmettre un paquet de synchronisation

3.2.60

bande de transmission cyclique

bande pour exécuter les transmissions cycliques

3.2.61

bande de transmission de données de sortie

bande pour transmettre des données de sortie dans les transmissions cycliques

3.2.62

bande de transmission de données d'entrée

bande pour transmettre des données d'entrée dans les transmissions cycliques

3.2.63

bande de transmission de messages

bande pour exécuter les transmissions de messages

3.2.64

bande de transmission de domaine de commande

bande totale de la bande de transmission cyclique et de la bande de transmission de messages dans un domaine de commande

3.2.65

bande de transmission IP

bande pour exécuter les transmissions de paquets IP

3.2.66**bande de garde**

bande pour empêcher l'envoi/la réception par les appareils dans le réseau

3.2.67**adresse de destination**

adresse pour spécifier la destination des données de commande, avec des informations sur 16 bits

3.2.68**adresse de la source**

adresse pour spécifier la source de données de commande, avec des informations sur 16 bits

3.2.69**adresse de nœud**

adresse à spécifier comme adresse de destination ou adresse de la source

3.2.70**numéro de nœud logique**

nombre de 1 à 127 à utiliser comme numéro de série dans l'ordre croissant, ce qui permet à l'appareil principal de spécifier l'appareil subordonné, car il y a 127 appareils subordonnés connectables à l'adresse de nœud ayant un espace de 16 bits dans la couche liaison de données

3.2.71**temps de propagation**

temps exigé pour que la trame envoyée par l'appareil principal de synchronisation ou l'appareil principal dans la couche liaison de données atteigne chaque nœud

Note 1 à l'article: Temps total de propagation du signal dans les câbles de transmission et temps de répétition au niveau des nœuds de relais

3.2.72**temps de surveillance de la réponse**

temps pour la surveillance de la réception des réponses

Note 1 à l'article: Ceci est mesuré à partir du début de l'envoi des données envoyées par le nœud primaire (y compris le temps de transmission des données envoyées par le nœud primaire)

3.2.73**transmissions consécutives**

transmissions continues de données de sortie

3.2.74**demande de retransmission**

exécution de la demande de retransmission de données d'entrée

3.2.75**communication déclenchée par les événements**

mode de communication où l'appareil principal et l'appareil subordonné échangent des données à un cadencement arbitraire, sans exécuter de synchronisation d'horloge dans les appareils par le paquet de synchronisation envoyé depuis l'appareil principal de synchronisation

3.2.76**mesure du retard de transmission**

mesures individuelles par l'appareil principal de synchronisation du temps de propagation pour les nœuds autres que son propre nœud

3.2.77

en amont

direction du transfert de données du nœud primaire vers le nœud secondaire

3.2.78

en aval

direction du transfert de données du nœud secondaire vers le nœud primaire

3.2.79

port amont

port qui devient en amont parmi les ports de communication à mettre en œuvre

Note 1 à l'article: Bien qu'il existe plusieurs domaines de commande dans le réseau (il existe au moins deux appareils principaux), dans les cas où le port amont diffère d'un domaine de commande à l'autre, il est appelé port amont du domaine de commande N° n (n est un entier).

3.2.80

port aval

port qui devient en aval parmi les ports de communication à mettre en œuvre

Note 1 à l'article: Bien qu'il existe plusieurs domaines de commande dans le réseau (il existe au moins deux appareils principaux), dans les cas où le port aval diffère d'un domaine de commande à l'autre, il est appelé port aval du domaine de commande N° n (n est un entier)

3.2.81

méthode de transfert vers l'amont

méthode de transfert vers l'amont parmi les méthodes de transfert dans le nœud où sont mis en œuvre au moins deux ports de communication

3.2.82

méthode de transfert vers l'aval

méthode de transfert vers l'aval parmi les méthodes de transfert dans le nœud où sont mis en œuvre au moins deux ports de communication

3.2.83

stocker et transférer

méthode de transfert pour stocker les paquets reçus dans le nœud en premier et les transmettre aux ports autres que les ports de réception en fonction du résultat de l'analyse des données stockées et des conditions de bande lors de la réception dans la bande

3.2.84

traversée

méthode de transfert pour transférer immédiatement les paquets reçus aux ports autres que le port de réception

3.2.85

répétition

méthode de transfert pour transférer les paquets reçus de la même manière que les paquets en traversée dans les cas où l'envoi n'est pas en cours dans son propre nœud, ou pour retenir les paquets reçus pendant l'envoi depuis son propre nœud et les transférer aux ports autres que le port de réception après avoir terminé l'envoi depuis son propre nœud

3.2.86

paquet IP

tous les paquets autres que ceux avec le champ longueur/type de trame MAC RT-PDU ('F68E'H) dans le présent document

3.2.87**paquet de synchronisation**

paquet pour synchroniser les cycles de transmission de tous les appareils au sein du réseau, qui est envoyé depuis l'appareil principal de synchronisation

3.3 Abréviations et symboles

Pour les besoins du présent document, les abréviations et symboles suivants s'appliquent.

AE	Application Entity (entité d'application)
AL	Application Layer (couche application)
A-, AL-	Application Layer (couche application) (utilisé comme préfixe)
ALME	Application Layer Management Entity (entité de gestion de couche application)
AP	Application Process (processus d'application)
APDU	Application Protocol Data Unit (unité de données de protocole de couche application)
API	Application Process Invocation (invocation de processus d'application)
APO	Application Process Object (objet de processus d'application)
APC	Application Process Context (contexte de processus d'application; utilisé comme préfixe d'un protocole pour bus de terrain de type 27)
APC SM	Application Process Context State Machine (diagramme d'états de contexte de processus d'application; pour bus de terrain de type 27)
AR	Application Relationship (relation d'application)
ASE AR	Application Relationship Application Service Element (élément de service d'application de relation entre applications)
AREP	Application Relationship End Point (extrémité de relation entre applications)
ARPM	Application Relationship Protocol Machine (machine de protocole de relation entre applications; pour bus de terrain de type 27)
ARPM-FDCM	ARPM for Field Device Control service Master (ARPM de l'appareil principal de service de commande d'appareil de terrain; pour bus de terrain de type 27)
ARPM-FDCS	ARPM for Field Device Control service Slave (ARPM de l'appareil subordonné de service de commande d'appareil de terrain; pour bus de terrain de type 27)
ARPM-MSG	ARPM for Message service (ARPM du service de messagerie; pour bus de terrain de type 27)
ASCII	American Standard Code for Information Interchange (code normalisé américain pour l'échange d'informations)
ASDU	Application Service Data Unit (unité de données de service d'application)
ASE	Application Service Element (élément de service d'application)
ASN.1	Abstract Syntax Notation One (notation de syntaxe abstraite numéro un)
BM	Band Master (appareil principal de bande)
CDO	Communication Data Object (objet de données de communication)
CMD	PDU d'instruction pour service FDC (pour bus de terrain de type 27)
Cnf	confirm primitive (primitive "confirm")
CTC	Communication Transfer Control (couche sous-jacente de commande de transfert de communication)
DA	Destination node Address (adresse de nœud de destination)

DLPDU	Data Link Protocol Data Unit (unité de données de protocole de liaison de données)
DLSDU	Data Link Service Data Unit (unité de données du service de liaison de données)
DMPM	Data Link Mapping Protocol Machine (machine de protocole de correspondance de couche liaison de données)
DMA	Destination MAC Address (adresse MAC de destination)
E2PROM	Electrically Erasable Programmable Read Only Memory (mémoire morte programmable effaçable électriquement)
FAL	Fieldbus Application Layer (couche application de bus de terrain)
FCS	Frame Check Sequence (séquence de contrôle de trame)
FDC-	Field Device Control (commande d'appareil de terrain; utilisation comme préfixe d'un service ou un protocole pour le bus de terrain de type 27)
ASE FDC	Field Device Control Application Service Element (élément de service d'application de commande d'appareil de terrain; pour bus de terrain de type 27)
FDCPM	Field Device Control Protocol Machine (machine de protocole de commande d'appareil de terrain; pour bus de terrain de type 27)
FDCPM-M	Field Device Control Protocol Machine for Master (machine de protocole de commande d'appareil de terrain pour appareil principal; pour bus de terrain de type 27)
FDCPM-S	Field Device Control Protocol Machine for Slave (machine de protocole de commande d'appareil de terrain pour appareil subordonné; pour bus de terrain de type 27)
FIFO	First In First Out (premier entré, premier sorti)
FSPM	FAL Service Protocol Machine (machine de protocole de service FAL)
FSM-	Fieldbus System Management (gestion système de bus de terrain; utilisation comme préfixe d'un service pour le bus de terrain de type 27)
ASE FSM	Fieldbus System Management Application Service Element (élément de service d'application de gestion système de bus de terrain; pour bus de terrain de type 27)
FSMUL	Field System Management Underlying Layer (couche sous-jacente de gestion système de terrain)
HMI	Human-machine Interface (interface homme-machine)
HS	Hand Shake (prise de contact)
I/O	Input/output (entrée/sortie)
ID	Identifier (identifiant)
Ind	indication primitive (primitive "indication")
LME	Layer Management Entity (entité de gestion de couche)
Lsb	Least Significant Bit (bit de poids faible)
LT	Length or Type (longueur ou type)
MAC	Media Access Control (contrôle d'accès au support)
Msb	Most Significant Bit (bit de poids fort)
MSG	Message (utilisation comme préfixe d'un service ou un protocole pour le bus de terrain de type 27)
ASE MSG	Message Application Service Element (élément de service d'application de message; pour bus de terrain de type 27)
MSGPM	Message Protocol Machine (machine de protocole de message) pour bus de terrain de type 27

MSGPM-RQ	MSGPM for ReQuester (MSGPM pour demandeur pour bus de terrain de type 27)
MSGPM-RS	MSGPM for ReSponder (MSGPM pour répondeur pour bus de terrain de type 27)
OSI	Open Systems Interconnection (interconnexion de systèmes ouverts)
PM	Protocol Machine (machine de protocole)
PDU	Protocol Data Unit (unité de données de protocole)
PhL	Ph-Layer (couche Ph)
QoS	Quality of Service (qualité de service)
RAM	Random Access Memory (mémoire vive)
Req	request primitive (primitive "request")
Rsp	response primitive (primitive "response")
RSP	PDU de réponse pour service FDC (pour bus de terrain de type 27)
SA	Source node Address (adresse de nœud source)
SAP	Service Access Point (point d'accès au service)
SDN	Send Data with No acknowledge (transmission de données sans acquittement)
SDU	Service Data Unit (unité de données de service)
SFD	Start Frame Delimiter (délimiteur de début de trame)
SM	Sync Master (appareil principal de synchronisation)
SMA	Source MAC Address (adresse MAC source)
SMIB	System Management Information Base (base d'informations de gestion système)
SRC	Send Receive Control underlying layer (couche sous-jacente de commande d'émission et réception)
UML	Unified Modelling Language (langage de modélisation unifié)

3.4 Conventions

3.4.1 Conventions générales

La couche application de bus de terrain (FAL) se compose d'un ensemble d'éléments ASE orientés objet. Chaque ASE est spécifié dans un paragraphe distinct. Chaque spécification d'élément ASE est divisée en trois parties: ses définitions de classe, ses services et la spécification de ses protocoles. Les deux premières se trouvent dans l'IEC 61158-5-27. La spécification de protocole de chaque ASE est définie dans le présent document.

Les définitions de classe définissent les attributs des classes prises en charge par chaque élément ASE. Dans le présent document, chaque attribut est présumé être accessible uniquement à partir de l'objet d'instance du propriétaire lui-même, directement ou par le biais des services de la classe.

Le présent document emploie les conventions de description énoncées dans l'ISO/IEC 10731.

3.4.2 Conventions du type de données PDU

Les types de données des FAL PDU sont définis avec les notations d'ASN.1.

Les chaînes de caractères qui commencent par "_" (trait de soulignement) font office de symboles de type de données des FAL PDU ou de leurs champs. Une même chaîne qu'un type PDU mais sans "_" indique une instance de PDU du type de données correspondant.

Exemple: CMD-PDU indique une instance de _CMD-PDU et une instruction suivante est ignorée.

CMD-PDU _CMD-PDU::= { valeur }

3.4.3 Conventions dans les diagrammes d'états

Les séquences de protocole sont décrites au moyen de diagrammes d'états.

Dans les schémas d'état, les états sont représentés par des cases et les transitions d'état par des flèches. Leurs conventions sont présentées dans l'ISO/IEC 19501 au format UML (Universal Modeling Language).

Les noms d'états et de transitions du schéma d'état correspondent aux noms indiqués dans la liste des transitions d'état.

Cette liste de transitions d'état est structurée comme indiqué dans le Tableau 1.

La première ligne contient le nom de la transition par un numéro d'index.

La deuxième ligne définit l'état source ou l'état en cours avant la transition.

La troisième ligne contient un événement et des actions. L'événement est suivi d'arguments facultatifs entre parenthèses, "(" et ")", et de conditions de garde entre crochets, "[" et "]", comme la première ligne. Et les actions commençant par le caractère "/" suivent la ligne d'événement.

La dernière ligne contient l'état cible ou l'état suivant après la transition.

Si l'événement se produit et que les conditions sont remplies, la transition se lance, c'est-à-dire que les actions sont exécutées et il y a une entrée dans le prochain état.

Tableau 1 – Descriptions de transition d'état

N° T	Etat source	Événement (arguments) [conditions] / Action	Etat cible

Une signification de chaque élément dans le Tableau 1 est indiquée dans le Tableau 2, selon la définition de "transition" UML (ISO/IEC 19501).

Tableau 2 – Description des éléments de diagramme d'états

Élément de description	Signification
Etat source	Noms de l'état d'origine et de l'état cible de la transition.
Etat cible	
N° T	Nom ou numéro de la transition.
Événement	Nom ou description de l'événement déclencheur de la transition.
(arguments)	Valeur de paramètre, expression ou séquence d'éléments séparés par "," pour l'événement. La parenthèse ouvrante "(" et la parenthèse fermante ")" ne font pas partie de la liste d'arguments.
[conditions]	Expression booléenne, qui est vraie pour la transition à lancer. Le crochet ouvrant "[" et le crochet fermant "]" ne font pas partie de la condition.
/ Action	Liste des affectations et des appels de service ou de fonction. Il convient que l'action soit atomique. La barre oblique "/" qui précède ne fait pas partie de l'action.
NOTE "(arguments)" et "[condition]" peuvent être ignorés s'ils s'avèrent inutiles.	

Les conventions utilisées dans les diagrammes d'états sont indiquées dans le Tableau 3.

Tableau 3 – Conventions utilisées dans les diagrammes d'états

Convention	Signification
<code>/*-- description --*/</code>	Décrit et explique les conditions et/ou la procédure à l'aide d'une phrase normale placée entre <code>/*--</code> et <code>--*/</code> au lieu d'utiliser la notation de pseudo-code. Exemple: <code>/*-- La PM examine une occurrence d'alarmes pour l'appareil distant correspondant.</code> <code>-- Si une alarme est détectée, la PM le signale à l'ASE FSM.</code> <code>-- Dans le cas contraire, la PM transfère MSGService-PDU à la DLL.</code> <code>--*/</code>
<code>=</code>	La valeur d'une entité de gauche est remplacée par celle d'une entité de droite. Si une entité de droite est un paramètre, il provient de la primitive identifiée comme un événement d'entrée.
<code>Axx</code>	Nom de paramètre si "a" est une lettre Exemple: <code>Identifieur = reason;</code> signifie que la valeur d'un paramètre 'reason' est attribuée à un paramètre appelé 'Identifieur.'
<code>"xxx"</code>	Chaîne visible fixe Exemple: <code>Identifieur = "abc";</code> signifie que la valeur "abc" est attribuée à un paramètre nommé 'Identifieur'
<code>Nnn</code>	Si tous les éléments sont des chiffres, l'élément représente une constante numérique notée en représentation décimale.
<code>0xnn</code>	Si tous les éléments nn sont des chiffres, l'élément représente une constante numérique notée en représentation hexadécimale.
<code>==</code>	Condition logique indiquant qu'un élément de gauche est égal à un élément de droite.
<code><</code>	Condition logique indiquant qu'un élément de gauche est inférieur à un élément de droite.
<code>></code>	Condition logique indiquant qu'un élément de gauche est supérieur à un élément de droite.
<code>!=</code>	Condition logique indiquant qu'un élément de gauche n'est pas égal à un élément de droite.
<code>&&</code>	"AND" (c'est-à-dire: ET) logique
<code> </code>	"OR" (c'est-à-dire: OU) logique
<code>!</code>	"NOT" (c'est-à-dire: PAS) logique
<code>+</code> <code>-</code> <code>*</code> <code>/</code>	Opérateurs arithmétiques
<code>;</code>	Séparateur d'expressions

D'autres notations définies en langage C (ISO/IEC 9899) peuvent être utilisées pour décrire les conditions et les actions.

4 Syntaxe abstraite

4.1 Généralités

Les formats de paquets et les formats de trame sont conformes à l'ISO/IEC/IEEE 8802-3.

4.2 Types de données de base

Les types de données suivants peuvent être utilisés dans la FAL de type 27.

- Types de base (types simples d'ASN.1):
 - INTEGER;
 - REAL;
 - BIT STRING;
 - OCTET STRING;
 - NULL.
- Types de base spécifiques (sous-types issus de l'ASN.1):
 - Integer8 ::= INTEGER (–128..127);
 - Integer16 ::= INTEGER (–32 768 .. 32 767);
 - Integer32 ::= INTEGER (–2 147 483 648 .. 2 147 483 647);
 - Integer64 ::= INTEGER ((–1)×'8000 0000 0000 0000'H..'7FFF FFFF FFFF FFFF'H);
 - Unsigned8 ::= INTEGER (0..'FF'H);
 - Unsigned16 ::= INTEGER (0..'FFFF'H);
 - Unsigned32 ::= INTEGER (0..'FFFF FFFF'H);
 - Unsigned64 ::= INTEGER (0..'FFFF FFFF FFFF FFFF'H);
 - *Float*₃₂ ::= 0 | negative infinity | positive infinity
 | REAL (WITH COMPONENTS(mantisse ((–1)× c_{32}), base (2), exposant (q_{32})))
 | REAL (WITH COMPONENTS(mantisse (c_{32}), base (2), exposant (q_{32})))
 où
 c_{32} ::= REAL (1.. ($\sum_{i=0}^{23} \frac{1}{2^i}$)) = REAL (1..(2–2^{–23})), et
 q_{32} ::= INTEGER (–127..127).

NOTE 1 Ce type a une longueur de quatre octets. Le format de float32 est celui défini par l'ISO/IEC/IEEE 60559 comme la simple précision, voir 5.2.2.

- *Float*₆₄ ::= 0 | negative infinity | positive infinity
 | REAL (WITH COMPONENTS(mantisse ((–1)× c_{64}), base (2), exposant (q_{64})))
 | REAL (WITH COMPONENTS(mantisse (c_{64}), base (2), exposant (q_{64})))
 où
 c_{64} ::= REAL (1.. ($\sum_{i=0}^{52} \frac{1}{2^i}$)) = REAL (1.. (2–2^{–52})), et
 q_{64} ::= INTEGER (–1 023..1 023).

NOTE 2 Ce type a une longueur de huit octets. Le format de float64 est celui défini par l'ISO/IEC/IEEE 60559 comme la double précision. Voir 5.2.2.

- IA5String

- Types de structure avec types de composant ASN.1:

Le type Structure est défini par une combinaison de types SEQUENCE, de types CHOICE d'ASN.1, de types de base et d'éléments de sous-type définis dans l'ASN.1. La FAL PDU décrite en 4.3 et 4.4 après le présent paragraphe correspond à ce type Structure.

- SEQUENCE
- CHOICE

- Types de matrice avec un type de composant ASN.1:

Les types de matrice sont une liste de types de données définis avec le type SEQUENCE OF de l'ASN.1. Les types de matrice classiques sont présentés avec le type SEQUENCE OF ci-dessous:

- SEQUENCE OF;
- BitArray ::= SEQUENCE OF (BIT STRING SIZE (8));
- Array8 ::= SEQUENCE OF Integer8;
- Array16 ::= SEQUENCE OF Integer16;
- Array32 ::= SEQUENCE OF Integer32;
- Array64 ::= SEQUENCE OF Integer64.

4.3 Types FAL PDU

4.3.1 Généralités

4.3.1.1 Format de paquet

La vue d'ensemble de la trame MAC utilisée pour ce protocole est telle que décrite ci-dessous. Les balises VLAN spécifiées par l'IEEE 802.1Q peuvent être utilisées en option pour désigner le niveau de priorité des paquets (voir 4.3.1.2.4, 8.2). Normalement, un format sans en-tête IP est utilisé (voir 4.3.1.2.5). Dans les cas où des appareils connectés échangent des données avec un appareil connecté à un réseau IP externe, le protocole UDP/IP doit être utilisé comme option. Noter qu'une partie des champs d'en-tête UDP/IP n'est pas utilisée pour ce protocole (voir 4.3.1.2.60).

```

DLPDU ::= SEQUENCE {
    preamble                OCTET STRING SIZE (7),
    start_frame_delimiter    OCTET STRING SIZE (1),
    destination_MAC_address OCTET STRING SIZE (6),
    source_MAC_address       OCTET STRING SIZE (6),
    data                     _DLSDU,
    frame_check_sequence     OCTET STRING SIZE (4)
}

_DLSDU ::= SEQUENCE {
    ( VLAN_tage             OCTET STRING SIZE (4) ) | vide,
    length_or_type          OCTET STRING SIZE (2),
    données                 _IP4-UDP-PDU | _IP6-UDP-PDU | _RT-PDU
    remplissage             OCTET STRING SIZE (0..46)
}

_IP4-UDP-PDU ::= SEQUENCE {
    IPv4_header             OCTET STRING SIZE (20),
    UDP_header              OCTET STRING SIZE (8),
    payload                 OCTET STRING SIZE (1..1472)
}

```

```

_IP6-UDP-PDU ::= SEQUENCE {
    IPv6_header    OCTET STRING SIZE (40),
    UDP_header     OCTET STRING SIZE (8),
    payload        OCTET STRING SIZE (1..1452)
}

_RT-PDU ::= {
    payload        OCTET STRING SIZE (1..1500)
}
    
```

4.3.1.2 Composant commun

4.3.1.2.1 Champ préambule et champ start_frame_delimiter

Ce champ doit être conforme à l'ISO/IEC/IEEE 8802-3.

4.3.1.2.2 Champ Destination_MAC_address

Ce champ doit être conforme à l'ISO/IEC/IEEE 8802-3. Noter que les utilisations de l'adresse de multidiffusion dans le Tableau 4 sont telles que spécifiées ci-dessous.

Tableau 4 – Adresse de multidiffusion

Adresse de multidiffusion	Nom	Utilisation
01:20:B5:00:04:00	Adresse commune de multidiffusion	Transmissions simultanées sur le même réseau
01:20:B5:00:04:01	Adresse de multidiffusion au nœud voisin	Celle-ci est utilisée pour l'envoi à un nœud dont l'adresse MAC n'est pas connue. Le nœud reçu ne doit pas être transféré vers d'autres ports.
01:20:B5:00:04:21 à 01:20:B5:00:04:2F	Pour l'adresse de multidiffusion du domaine de commande	Pour les transmissions simultanées dans un domaine de commande

4.3.1.2.3 Champ Source_MAC_address

Ce champ doit être conforme à l'ISO/IEC/IEEE 8802-3.

4.3.1.2.4 Champ VLAN_tag

Ce champ doit être conforme à l'ISO/IEC/IEEE 8802-3. La mise en œuvre est facultative.

4.3.1.2.5 Champ Length_or_type

Sans en-tête IP, ce champ doit être RT-PDU. Avec un en-tête IP, ce champ doit être conforme à l'ISO/IEC/IEEE 8802-3. Les valeurs principalement utilisées sont celles indiquées dans le Tableau 5.

Tableau 5 – Length_or_type

Valeur	Signification
'F68E'H	RT-PDU
'0800'H	IP4-UDP-PDU, IP4-UDP-PDU
autre	Conformément à l'ISO/IEC/IEEE 8802-3

4.3.1.2.6 Champ IPv4_header, champ IPv6_header et champ UDP_header

Ce champ doit être conforme aux normes IETF RFC 791, IETF RFC 2460 et IETF RFC 768. Noter qu'une partie des champs n'est pas utilisée (voir Tableau 6, Tableau 7 et Tableau 8).

Tableau 6 – En-tête IPv4

Alignement [octet]	Descriptions	Remarques
0	Version 4 (IPv4) et longueur de l'en-tête: 5 quadlets (20 octets)	'45'H, fixe
1	Type de service	'00'H, fixe
2 à 3	Longueur totale	
4 à 5	Identifiant	Non utilisé
6 à 7	Fanion et décalage de fragment	0, fixe
8	Durée de vie (TTL)	'01'H, fixe
9	Type de protocole d'ordre supérieur	'11'H(UDP), fixe
10 à 11	Somme de contrôle de l'en-tête IP	Sans vérification
12 à 15	Adresse IP de la source	
16 à 19	Adresse IP de destination	

Tableau 7 – En-tête IPv6

Alignement [octet]	Descriptions	Remarques
0 à 3	Version 6 (IPv6)	6, fixe
	Classe de trafic	Non utilisé (0, fixe)
	Etiquette de débit	Non utilisé (0, fixe)
4 à 5	Longueur de charge utile	
6	En-tête suivant	'11'H(UDP), fixe
7	Nombre de sauts maximal	
8 à 23	Adresse IP de la source	
24 à 39	Adresse IP de destination	

Tableau 8 – En-tête UDP

Alignement [octet]	Descriptions	Remarques
0 à 1	Numéro de port UDP de la source	Non utilisé
2 à 3	Numéro de port UDP de la destination	'F68E'H, fixe
4 à 5	Longueur des données UDP	Non utilisé
6 à 7	Somme de contrôle UDP	Non utilisé (0, fixe)

4.3.1.2.7 Champ de charge utile

_APDU est stocké dans ce champ (voir 4.3.2).

4.3.1.2.8 Champ Frame_check_SEQUENCE

Ce champ doit être conforme à l'ISO/IEC/IEEE 8802-3.

4.3.2 Plans de types APDU: _APDU

_APDU est constituée de 14 types de FIDAP-PDU. _FDCServicePDU doit être composé d'une paire de PDU. _CMD-PDU et _RSP-PDU sont respectivement utilisés pour les instructions FDC et ses réponses. Ceci est utilisé dans _OUTPUT-FIDAP-PDU (voir 4.3.4.2), _INPUT-FIDAP-PDU (voir 4.3.4.3) et _EVD-FIDAP-PDU (voir 4.3.4.11). De plus _MSGServicePDU doit être composée d'une paire de PDU; _MSGREQ-PDU et _MSGRSP-PDU sont respectivement utilisées pour les demandes MSG et ses réponses. Ceci est utilisé dans _MSG-FIDAP-PDU (voir 4.3.4.9).

```

_APDU          ::= _FIDAP-PDU                                -- PDU du service d'application FrameID
_FIDAP-PDU     := _SYNC-FIDAP-PDU
               | _OUTPUT-FIDAP-PDU
               | _INPUT-FIDAP-PDU
               | _TOKEN-FIDAP-PDU
               | _REQ_SEND-FIDAP-PDU
               | _MEAS_DLY-CMD-FIDAP-PDU | _MEAS_DLY-RSP-FIDAP-PDU
               | _READ_CDO-CMD-FIDAP-PDU | _READ_CDO-RSP-FIDAP-PDU
               | _WRITE_CDO-CMD-FIDAP-PDU | _WRITE_CDO-RSP-FIDAP-PDU
               | _MSG-FIDAP-PDU
               | _ASYNC_IND-FIDAP-PDU
               | _EVD-FIDAP-PDU
_FDCServicePDU ::= _CMD-PDU                                -- Type de PDU d'instruction FDC
               | _RSP-PDU                                -- Type de PDU de réponse FDC
_MSGServicePDU ::= _MSGREQ-PDU                             -- Type de PDU de demande MSG
               | _MSGRSP-PDU                             -- Type de PDU de réponse MSG
    
```

4.3.3 PDU pour le service de gestion système de terrain

4.3.3.1 Vue d'ensemble

La FID-PDU doit être éditée avec une SDU dans la primitive de service FSM et être envoyée par l'intermédiaire de l'AR. Un ASE FSM reçoit la PDU de l'ASE homologue par l'intermédiaire de l'AR.

Un utilisateur ASE FSM différencie les types en utilisant le champ de code d'instruction comme code clé.

```

FIDAP-PDU      ::= SEQUENCE {
                                fpduLength  _FPDULength,
                                fcmd        _FCMDCode,
                                fHS,        _FHandShake,
                                fpduBody    OCTET STRING SIZE (1..1496)
                                }
    
```

- Champ fpduLength

Ce champ affiche le nombre d'octets de fpduBody. Le type de données est _FPDULength. Il doit comporter trois sous-champs, la longueur des données, le numéro de paquet divisé et le fanion de renvoi.

```

_FPDULength ::= SEQUENCE {
    Longueur des données          BIT STRING SIZE (12),
    Numéro de paquet divisé      BIT STRING SIZE (3),
    Fanion de renvoi,             BIT STRING SIZE (1)
}

```

Longueur des données

Nombre d'octets de données MAC, à l'exclusion de la longueur de données elle-même (2 octets)

Numéro de paquet divisé

Stocke le numéro de paquet lorsque les paquets sont divisés en plusieurs paquets en raison de la restriction de taille des données MAC. (sans division: 0)

Fanion de renvoi

Un fanion indiquant le renvoi des paquets. Fixer à 1 en cas de renvoi.

- Champ fcmd

Un champ de code d'instruction pour indiquer le code permettant d'identifier le contenu de f pduBody. Ce champ doit être codé comme _FCMDCode pour Unsigned8. Les instructions communes déjà définies sont indiquées comme _PrimaryFCMD.

```

_FCMDCode      ::= Unsigned8
_PrimaryFCMD   ::= _FCMDCode(
    SYNC ('01'H),      1)
    OUTPUT ('02'H),    1)
    INPUT ('03'H),      1)
    TOKEN ('04'H),     1) 2)
    REQ_SEND ('05'H),  1) 2)
    MEAS_DLY ('06'H),  1)
    READ_CDO ('07'H),
    WRITE_CDO ('08'H),
    MSG ('09'H),
    ASYNC_IND ('0A'H),
    EVD ('0B'H)
)

```

- 1) Ils ne peuvent pas être utilisés pour _IP4-UDP-PDU et _IP6-UDP-PDU. Ils ne peuvent pas être acheminés vers un réseau IP externe.
- 2) Alors que le droit d'envoi se déplace de la source vers la destination pour un jeton, le droit d'envoyer une demande d'envoi reste du côté source et une demande d'envoi est faite vers la destination.

- Champ fHS

Ce champ doit indiquer une prise de contact/notification. Il doit comporter cinq sous-champs significatifs, la longueur des données, le numéro de paquet divisé et le fanion de renvoi.

```

_FHandShake ::= SEQUENCE {
    Fanion de demande / réponse          BIT STRING SIZE (1),
    Fanion d'achèvement                  BIT STRING SIZE (1),
    réserve1                             BIT STRING SIZE (1),
    Fanion de démarrage de réinitialisation du réseau  BIT STRING SIZE (1),
    Fanion de confirmation de démarrage de réinitialisation du réseau  BIT STRING SIZE (1),
    Fanion d'achèvement de démarrage de réinitialisation du réseau  BIT STRING SIZE (1),
    reserve2                             BIT STRING SIZE (1)
}

```

Fanion de demande / réponse

Fanion servant à identifier la demande ou la réponse

- 0: Demande;
- 1: Réponse;

Fanion d'achèvement

Fanion servant à identifier l'accusé de réception ou l'accusé de réception négatif

- 0: Accusé de réception
- 1: Accusé de réception négatif.

Fanion de démarrage de réinitialisation du réseau

Fanion pour notifier le démarrage de la réinitialisation du réseau

- 0: Ne pas notifier;
- 1: Notifier

Fanion de confirmation de démarrage de réinitialisation du réseau

Fanion pour notifier la confirmation du démarrage de la réinitialisation du réseau

- 0: Ne pas notifier;
- 1: Notifier.

Fanion d'achèvement de démarrage de réinitialisation du réseau

Fanion pour notifier l'achèvement du démarrage de réinitialisation du réseau

- 0: Ne pas notifier;
- 1: Notifier.

• Champ pduBody

Il s'agit d'un champ de données. Le format des données sera expliqué par le code d'instruction dans les paragraphes suivants. Dans ce paragraphe, l'adresse de nœud commune à toutes les instructions, qui est incluse dans les données, est spécifiée. La configuration de l'adresse de destination est la même que celle de l'adresse de la source, comme indiqué dans le Tableau 9.

Tableau 9 – Adresse du nœud

Alignement [octet]	Nom	Type de données	Descriptions	Remarques
0 à 1	Adresse du nœud	Unsigned16	Numéro ID du nœud '0000'H à 'FFEF'H: pour l'unidiffusion 'FFF0'H à 'FFFE'H: pour la multidiffusion 'FFFF'H: pour la diffusion	Un nombre exclusif au sein du réseau doit être attribué.

4.3.4 Définitions détaillées des _FID-PDU

4.3.4.1 Instruction SYNC

L'instruction SYNC doit être utilisée pour la notification d'informations de synchronisation provenant de l'appareil principal de synchronisation.

```

_SYNC-FIDAP-PDU      ::= _FIDAP-PDU ( WITH COMPONENTS
                        { ..., fcmd (sync), fHS,
                          fpduBody (fcmdBody _SYNC-FIDAPBody)
                        }
                      )

_SYNC-FIDAPBody      ::= SEQUENCE {
                        pDstAddress    Unsigned16,
                        pSrcAddress    Unsigned16,
                        pCycleCounter  Unsigned16,
                        réserve        Unsigned32,
                        pTimeStamp     Unsigned64,
                        pUserData      OCTET STRING SIZE (0..24)
                      }

```

- Champ fcmd
Ce champ doit indiquer l'instruction SYNC.
- Champ fHS
Il s'agit d'un champ de données qui indique une prise de contact/notification avec la plage de valeurs suivante: '00'H: Normalement (lorsqu'il n'est pas utilisé);
– '10'H (NRS): Démarrage de la réinitialisation du réseau;
– '40'H (NRS_COMP): Le démarrage de la réinitialisation du réseau est terminé.
- Champ pSrcAddress
Ce champ doit indiquer l'adresse du nœud source. Définit l'adresse de l'appareil principal de synchronisation (voir le Tableau 9).
- Champ pCycleCounter
Ce champ doit être utilisé pour incrémenter un compteur de un à chaque cycle de transmission. Ceci doit être utilisé pour traiter plusieurs cycles de transmission.
- Champ pTimeStamp (TS_fsync)
Ce champ stocke un horodatage. (Unité: 1 ns)
- Champ pUserData
Ce champ stocke les données définies par l'utilisateur. Il s'agit d'une zone facultative. Lors de l'envoi d'un paquet où l'instruction SYNC est stockée, l'adresse de multidiffusion commune doit être utilisée comme adresse MAC de destination (voir le Tableau 4).

4.3.4.2 Instruction OUTPUT

L'instruction OUTPUT est utilisée pour les données d'instruction d'un appareil principal à des appareils subordonnés. Il existe deux types de formats de données de sortie. Le format de base doit être utilisé pour une destination de nœud. Le format étendu doit être réalisé en joignant plusieurs formats de base vers plusieurs destinations de nœuds. Pour les données d'utilisateur, les données de sortie pour le nombre d'appareils dans le nœud source doivent être stockées.

```

_OUTPUT-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (sortie), fHS,
      fpduBody (fcmdBody _OUTPUT-FIDAPBody)
    }
  )

_OUTPUT-FIDAPBody ::= SEQUENCE {
    pDstAddress  Unsigned16,
    réserve      Unsigned2,
    pLength      Unsigned14,
    pUserData    _FDCServicePDU +
  }
| _OUTPUT-FIDAPBody -- Format étendu

```

- Champ fcmd
- Ce champ doit indiquer l'instruction OUTPUT. Champ fHS
Ce champ doit indiquer la prise de contact/notification avec la plage de valeurs ci-après.
 - '00'H: Normalement (lorsqu'il n'est pas utilisé);
 - '10'H (NRS): Démarrage de la réinitialisation du réseau;
 - '20'H (NRS_CONF): Confirmation de redémarrage de la réinitialisation du réseau.
- Champ pDstAddress
Ce champ doit indiquer l'adresse du nœud de destination. L'adresse des appareils subordonnés pour recevoir des données de sortie doit être définie (voir le Tableau 9).
- Champ pLength
Ce champ doit indiquer le numéro d'octet de données utilisateur.
- Champ pUserData
Ce champ stocke les données de sortie. _FDCServicePDU doit être stockée en tant que données de sortie par l'appareil pour tous les appareils (voir 4.3.5).

4.3.4.3 Instruction INPUT

L'instruction INPUT est utilisée pour les données d'instruction des appareils subordonnés vers un appareil principal. Pour les données d'utilisateur, les données d'entrée pour le nombre d'appareils dans le nœud source doivent être stockées.

```

_INPUT-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (entrée), fHS,
      fpduBody (fcmdBody _INPUT-FIDAPBody)
    }
  )

_INPUT-FIDAPBody ::= SEQUENCE {
    pSrcAddress  Unsigned16,
    réserve      Unsigned2,
    pLength      Unsigned14,
    pUserData    _FDCServicePDU +
  }

```

- Champ fcmd
Ce champ doit indiquer l'instruction INPUT.

- Champ fHS
Ce champ doit indiquer la prise de contact/notification avec la plage de valeurs ci-après.
 - '00'H: Normalement (lorsqu'il n'est pas utilisé);
 - '10'H (NRS): Démarrage de la réinitialisation du réseau.
- Champ pSrcAddress
Ce champ doit indiquer l'adresse du nœud source. L'adresse des appareils subordonnés qui ont envoyé des données d'entrée doit être définie (voir le Tableau 9).
- Champ pLength
Ce champ doit indiquer le numéro d'octet de données utilisateur.
- Champ pUserData
Ce champ stocke les données d'entrée. _FDCServicePDU doit être stockée en tant que données d'entrée par l'appareil pour tous les appareils (voir 4.3.5).

4.3.4.4 Instruction TOKEN

L'instruction TOKEN est utilisée pour notifier la réception/donner le droit d'envoi entre appareils principaux.

```

_TOKEN-FIDAP-PDU      ::= _FIDAP-PDU ( WITH COMPONENTS
                        { ..., fcmd (token), fHS,
                          fpduBody (fcmdBody _TOKEN-FIDAPBody)
                        }
                        )

_TOKEN-FIDAPBody       ::= SEQUENCE {
                        pDstAddress      Unsigned16,
                        pSrcAddress      Unsigned16,
                        réserve          Unsigned8,
                        pTokenType       Usingned8
                        }
  
```

- Champ fcmd
Ce champ doit indiquer l'instruction TOKEN.
- Champ fHS
Ce champ doit indiquer la prise de contact/notification avec la plage de valeurs ci-après.
 - '00'H: Normalement (non utilisé).
- Champ pDstAddress
Ce champ doit indiquer l'adresse de nœud de destination à laquelle le droit d'envoi est donné. L'adresse de l'appareil principal de destination à laquelle le droit d'envoi est donné doit être définie (voir Tableau 9).
- Champ pSrcAddress
Ce champ doit indiquer l'adresse de nœud source qui donne le droit d'envoi. L'adresse de l'appareil principal source qui donne le droit d'envoi doit être définie (voir le Tableau 9).
- réserve
Il s'agit d'une zone de réserve. Mise à 0.
- Champ pTokenType
Ce champ doit indiquer les types de jeton avec la plage de valeurs ci-après:
 - 1 (MST_MEAS_DLY): Mesure du retard de l'appareil principal;
 - 2 (CONFIG): Configuration.

4.3.4.5 Instruction REQ_SEND

L'instruction REQ_SEND doit être utilisée lors de la présentation d'une demande d'envoi d'un paquet. Il existe trois types de formats de demande d'envoi par type de demande. Ils doivent être utilisés respectivement comme instruction d'envoi pour la demande de retransmission de données d'entrée, la demande de mesure de retard, la notification de synchronisation et la notification de début de réinitialisation du réseau.

```

_REQ_SEND-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (req_send), fHS,
      fpduBody (fcmdBody _REQ_SEND-FIDAPBody)
    }
  )
_REQ_SEND-FIDAPBody ::= SEQUENCE {
    réserve1          Unsigned16,
    pPollingType      Unsigned8,
    pNumAddr          Unsigned8,
    ( pReqDstAddress  Unsigned16 )
  }
  | SEQUENCE {
    pDstAddress      Unsigned16,
    pPollingType     Unsigned8,
    pCalcType        Unsigned8,
    pSyncAddr        Unsigned16,
    pSyncMACAddr     Unsigned48
  }
  | SEQUENCE {
    pDstAddress      Unsigned16,
    pPollingType     Unsigned8,
    réserve2         Unsigned8
  }
  }

```

-- Retransmission des données d'entrée

-- Demande de mesure de retard

-- Notification de synchronisation et notification de début de la réinitialisation du réseau

- Champ fcmd
Ce champ doit indiquer l'instruction SEND.
- Champ fHS
Ce champ doit indiquer une prise de contact/notification avec la valeur suivante:
 - '00'H: Normalement (non utilisé).
- réserve1
Il s'agit d'une zone de réserve. Mise à 0.
- Champ pPollingType
Ce champ doit indiquer les types de demande avec la plage de valeurs ci-après:
 - 1 (INPUT): Retransmission des données d'entrée;
 - 2 (MEAS_DLY): Demande de mesure du retard;
 - 3 (SYNC): Notification de synchronisation;
 - 4 (NET_NOTI): Notification de démarrage de la réinitialisation du réseau.

- Champ pNumAddr
Ce champ doit indiquer le nombre d'adresses et cela est valide lorsque le type de demande est 1. Le nombre d'adresses stockées doit être défini.
- Champ pReqDstAddress
Ce champ doit indiquer l'adresse de nœud de la destination de la demande de transmission de données d'entrée et cela est valide lorsque le type de demande est 1. L'adresse de l'appareil subordonné qui demande la retransmission des données d'entrée doit être définie (voir le Tableau 9).
- Champ pDstAddress
Ce champ doit indiquer l'adresse de nœud de destination, et cela est valide lorsque le type de demande est 2, 3 ou 4. L'adresse de l'appareil subordonné pour recevoir la demande d'envoi doit être définie (voir le Tableau 9).
- Champ pCalcType
Ce champ doit indiquer le type de calcul de la mesure du retard, et cela est valide lorsque le type de demande est 2. Il prend la plage de valeurs ci-après:
 - 0 (INIT): Traitement d'initialisation
 - 1 (PAP): Traitement de rattachement prêt à fonctionner
- Champ pSyncAddr
Ce champ doit indiquer l'adresse de l'appareil principal de synchronisation comme destination de réponse, et cela est valide lorsque le type de demande est 2. L'adresse de l'appareil principal de synchronisation doit être définie (voir le Tableau 9).
- Champ pSyncMACAddr
Ce champ doit indiquer l'adresse MAC de l'appareil principal de synchronisation comme destination de réponse, et cela est valide lorsque le type de demande est 2. L'adresse MAC de l'appareil principal de synchronisation doit être définie.
- réserve2
Il s'agit d'une zone de réserve. Mise à 0.

4.3.4.6 Instruction et réponse MEAS_DLY

Le MEAS_DLY doit être utilisé pour l'instruction et la réponse des mesures de retard des chemins de transmission (voir 8.3.4.1).

```

_MEAS_DLY-CMD-      ::= _FIDAP-PDU ( WITH COMPONENTS
_FIDAP-PDU
    { ..., fcmd (measure_dly), fHS,
      fpduBody (fcmdBody  _MEAS_DLY-CMD-FIDAPBody)
    }
  )

_MEAS_DLY-CMD-      ::= SEQUENCE {
_FIDAPBody
    pDstAddress      Unsigned16,
    pSrcAddress      Unsigned16,
    réserve          Unsigned16,
    pTimeStamp3      Usingned32
  }

_MEAS_DLY-RSP-      ::= _FIDAP-PDU ( WITH COMPONENTS
_FIDAP-PDU
    { ..., fcmd (measure_dly), fHS,
      fpduBody (fcmdBody  _MEAS_DLY-RSP-FIDAPBody)
    }
  )

```



```

_MEAS_DLY-RSP-      ::= SEQUENCE {
_FIDAPBody
    pDstAddress      Unsigned16,
    pSrcAddress      Unsigned16,
    réserve          Unsigned16,
    pTimeStampT3     Usingned32,
    pTimeStampT4     Usingned32
}

```

- Champ fcmd
Ce champ doit indiquer MEAS_DLY.
- Champ fHS
Ce champ doit indiquer une prise de contact/notification avec la plage de valeurs ci-après pour identifier l'instruction ou la réponse:
 - '00'H (REQ): Demande;
 - '01'H (ACK): Accusé de réception
 - '03'H (NACK): Accusé de réception négatif.
- Champ pDstAddress
Ce champ doit indiquer l'adresse du nœud de destination. L'adresse de l'appareil principal de synchronisation ou l'adresse de diffusion doit être définie (voir le Tableau 9).
- Champ pSrcAddress
Ce champ doit indiquer l'adresse du nœud source. L'adresse du nœud source doit être définie (voir le Tableau 9).
- réserve
Il s'agit d'une zone de réserve. Mise à 0.
- Champ pTimeStampT3
Ce champ doit indiquer l'horodatage T3. L'horodatage lors de l'envoi de la demande de mesure du retard doit être réglé. (Unité: 1 ns)
- Champ pTimeStampT4
Ce champ doit indiquer l'horodatage T4. L'horodatage à la réception de la demande de mesure du retard doit être réglé. (Unité: 1 ns)

4.3.4.7 Instruction et réponse READ_CDO

Le READ_CDO doit être utilisé pour une instruction/réponse de lecture d l'ID DLO.

```

_READ_CDO-CMD-      ::= _FIDAP-PDU( WITH COMPONENTS
_FIDAP-PDU
    { ..., fcmd (read_cdo), fHS,
      fpduBody (fcmdBody  _READ_CDO-CMD-FIDAPBody)
    }
)

_READ_CDO-CMD-      ::= SEQUENCE {
_FIDAPBody
    pDstAddress      Unsigned16,
    pSrcAddress      Unsigned16,
    réserve          Unsigned8,
    pNumCDO_ID       Usingned8,
    ( pCDO_ID        Usingned16 ) +
}

```

```

_READ_CDO-RSP- ::= _FIDAP-PDU ( WITH COMPONENTS
FIDAP-PDU
    { ..., fcmd (read_cdo), fHS,
      fpduBody (fcmdBody _READ_CDO-RSP-FIDAPBody)
    }
  )

_READ_CDO-RSP- ::= SEQUENCE {
FIDAPBody
    pDstAddress    Unsigned16,
    pSrcAddress    Unsigned16,
    réserve        Unsigned8,
    pNumCDO_ID     Usingned8,
    ( SEQUENCE {
        pCDO_ID     Usingned16,
        pResult      Usingned16,
        pCDO_Data    OCTET STRING SIZE (1..48),
        remplissage  OCTET STRING SIZE (0..3)
    } ) +
  }

```

- Champ fcmd
Ce champ doit indiquer READ_CDO.
- Champ fHS
Ce champ doit indiquer une prise de contact/notification avec la plage de valeurs ci-après pour identifier l'instruction ou la réponse:
 - '00'H (REQ): Demande;
 - '01'H (ACK): Accusé de réception
 - '03'H (NACK): Accusé de réception négatif.
- Champ pDstAddress
Ce champ doit indiquer l'adresse du nœud de destination. Pour une instruction, l'adresse de nœud de destination de lecture doit être définie. Pour une réponse, l'adresse du nœud source doit être définie (voir le Tableau 9).
- Champ pSrcAddress
Ce champ doit indiquer l'adresse du nœud source. L'adresse du nœud source doit être définie (voir le Tableau 9).
- réserve
Il s'agit d'une zone de réserve. Mise à 0.
- Champ pNumCDO_ID
Ce champ doit indiquer le numéro CDO_ID cible de lecture. Le numéro CDO_ID stocké doit être défini.
- Champ pCDO_ID
Ce champ doit indiquer la cible de lecture CDO_ID. CDO_ID doit être défini.
- Champ pResult
Ce champ doit indiquer le résultat de l'ID CDO lu avec la plage de valeurs ci-après:
 - 0: Normalement achevé;
 - 1: Erreur de CDO_ID (le COD_ID correspondant n'existe pas);
 - 2: Lecture impossible (lecture de CDO_ID ayant un attribut d'écriture seule);
 - 3: Lecture de CDO_ID non mis en œuvre.

- Champ pCDO_Data
Ce champ doit indiquer la valeur de CDO ID. La valeur qui a été lue conformément à la taille des données CDO ID doit être définie.
- Champ de remplissage
Ceci doit être inséré pour devenir une frontière de 4 octets conformément à la taille des données de CDO ID. Mise à 0.

4.3.4.8 Instruction et réponse WRITE_CDO

Le WRITE_CDO doit être utilisé pour une demande/réponse de lecture de DLO ID.

```

_WRITE_CDO-CMD- ::= _FIDAP-PDU ( WITH COMPONENTS
FIDAP-PDU
    { ..., fcmd (write_cdo), fHS,
      fpduBody (fcmdBody _WRITE_CDO-CMD-FIDAPBody)
    }
  )

_WRITE_CDO-CMD- ::= SEQUENCE {
FIDAPBody
    pDstAddress    Unsigned16,
    pSrcAddress    Unsigned16,
    réserve1       Unsigned8,
    pNumCDO_ID    Usingned8,
    ( SEQUENCE {
        pCDO_ID    Usingned16,
        réserve2    Usingned16,
        pCDO_Data  OCTET STRING SIZE (1..48),
        remplissage OCTET STRING SIZE (0..3)
      } )
  }

_WRITE_CDO-RSP- ::= _FIDAP-PDU ( WITH COMPONENTS
FIDAP-PDU
    { ..., fcmd (write_cdo), fHS,
      fpduBody (fcmdBody _WRITE_CDO-RSP-FIDAPBody)
    }
  )

_WRITE_CDO-RSP- ::= SEQUENCE {
FIDAPBody
    pDstAddress    Unsigned16,
    pSrcAddress    Unsigned16,
    réserve1       Unsigned8,
    pNumCDO_ID    Usingned8,
    ( SEQUENCE {
        pCDO_ID    Usingned16,
        pResult     Usingned16 )
      } ) +
  }
  )

```

- Champ fcmd
Ce champ doit indiquer WRITE_CDO.

- Champ fHS
Ce champ doit indiquer une prise de contact/notification avec la plage de valeurs ci-après pour identifier l'instruction ou la réponse:
 - '00'H (REQ): Demande;
 - '01'H (ACK): Accusé de réception
 - '03'H (NACK): Accusé de réception négatif.
- Champ pDstAddress
Ce champ doit indiquer l'adresse du nœud de destination. Pour une instruction, l'adresse de nœud de destination d'écriture doit être définie. Pour une réponse, l'adresse du nœud source doit être définie (voir le Tableau 9).
- Champ pSrcAddress
Ce champ doit indiquer l'adresse du nœud source. L'adresse du nœud source doit être définie (voir le Tableau 9).
- réserve1
Il s'agit d'une zone de réserve. Mise à 0.
- Champ pNumCDO_ID
Ce champ doit indiquer le numéro CDO_ID cible en écriture. Le numéro CDO_ID stocké doit être défini.
- Champ pCDO_ID
Ce champ doit indiquer le CDO_ID cible d'écriture. CDO_ID doit être défini.
- reserve2
Il s'agit d'une zone de réserve. Mise à 0.
- Champ pCDO_Data
Ce champ doit indiquer la valeur CDO_ID. La valeur à écrire doit être définie en fonction de la taille des données de CDO_ID.
- Champ de remplissage
Celui-ci doit être inséré pour devenir une frontière de 4 octets selon la taille des données de CDO_ID. Mise à 0.
- Champ pResult
Ce champ doit indiquer le résultat de l'écriture de CDO ID avec la plage de valeurs ci-après:
 - 0: Normalement achevé;
 - 1: Erreur de CDO_ID (le COD_ID correspondant n'existe pas);
 - 2: Ecriture impossible (écriture de CDO_ID ayant un attribut de lecture seule);
 - 3: Ecriture de CDO_ID non mis en œuvre;
 - 4: Erreur de valeur d'écriture

4.3.4.9 Instruction MSG

L'instruction MSG doit être utilisée pour notifier la réception et la transmission de message entre les nœuds.

```

_MSG-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (msg), fHS,
      fpduBody (fcmdBody _MSG-FIDAPBody)
    }
)

```

```

_MSG-FIDAPBody ::= SEQUENCE {
    pDstAddress  Unsigned16,
    pSrcAddress  Unsigned16,
    pMsgControl  _MSG_CONTROL,
    pUserData    _MSGServicePDU
}

```

- **Champ fcmd**
Ce champ doit indiquer l'instruction MSG.
- **Champ fHS**
Ce champ doit indiquer une prise de contact/notification avec la valeur suivante.
– '00'H: Normalement (non utilisé).
- **Champ pDstAddress**
Ce champ doit indiquer l'adresse du nœud de destination. L'adresse de nœud de destination doit être définie (voir le Tableau 9).
- **Champ pSrcAddress**
Ce champ doit indiquer l'adresse du nœud source. L'adresse du nœud source doit être définie (voir le Tableau 9).
- **Champ pMsgControl**
Ce champ doit indiquer les informations de commande de message. Il existe deux types de formats d'informations de commande de messages: I-format et S-format.

```

_MSG_CONTROL ::= SEQUENCE {
    pNR  BIT STRING SIZE (7),
    pPF  BIT STRING SIZE (1),
    pNS  BIT STRING SIZE (1),
    pType BIT STRING SIZE (1)
}
-- I-format

-- S-format
SEQUENCE {
    pNR  BIT STRING SIZE (7),
    réserve1 BIT STRING SIZE (1),
    réserve2 BIT STRING SIZE (4),
    pStype BIT STRING SIZE (2),
    réserve3 BIT STRING SIZE (1),
    pType BIT STRING SIZE (1)
}

```

pNR

Un champ pour indiquer le numéro de série des données de réception.

pPF

Un fanion pour identifier l'interrogation/le fragment final.

- 0: Interrogation;
- 1: Fragment final.

pNS

Un champ pour indiquer le numéro de série des données d'envoi.

pType

Un fanion pour identifier le format.

- 0: I-format;
- 1: S-format.

réserve1

Une zone de réserve. Mise à 1.

reserve2

Une zone de réserve. Mise à 0.

pSType

Un champ pour indiquer le type de trame de S-format.

réserve3

Une zone de réserve. Mise à 0.

- Champ pUserData

Ce champ doit stocker le message. Il doit stocker une partie de _MSGServicePDU ou _MSGServicePDU divisée (voir 4.3.6).

4.3.4.10 Instruction ASYNC_IND

L'instruction ASYNC_IND doit être utilisée pour une notification d'événement.

```

_ASYNC_IND-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (async_ind), fHS,
      fpduBody (fcmdBody _ASYNC_IND-FIDAPBody)
    }
  )
_ASYNC_IND-FIDAPBody ::= SEQUENCE {
    pDstAddress  Unsigned16,
    pSrcAddress  Unsigned16,
    pEventType   Unsigned8,
    pParam       _ASYNC_IND_PARAM,
    pUserData    _ASYNC_IND_DATA | empty
  }

```

- Champ fcmd

Ce champ doit indiquer l'instruction ASYNC_IND.

- Champ fHS

Ce champ doit indiquer une prise de contact/notification avec la valeur suivante:

- '00'H: Normalement (non utilisé).

- Champ pDstAddress

Ce champ doit indiquer l'adresse du nœud de destination. L'adresse de l'appareil principal de destination doit être définie. Noter que cela doit être défini à diffusion pour l'événement de type 1 (voir le Tableau 9).

- Champ pSrcAddress

Ce champ doit indiquer l'adresse du nœud source. L'adresse de l'appareil principal source doit être définie (voir le Tableau 9).

- Champ pEventType

Ce champ doit indiquer le type d'événement avec la plage de valeurs ci-après:

- 1 (NEI_NOTI): Notification du nœud voisin;
- 2 (SUP_NOTI): Notification de démarrage;
- 3 (SUP_RESP): Réponse à la notification de démarrage;
- 4 (NEI_LIST): Notification de liste de nœuds voisins;
- 5 (MEAS_DLY): Mesure du retard;
- 6 (MEAS_DLY_COMP): La mesure du retard est terminée;
- 7 (CHG_ADDR): Changement d'adresse de nœud;
- 8 (NRS_COMP): Le démarrage de la réinitialisation du réseau est terminé.

- Champ pParam

Ce champ doit indiquer le paramètre, et cela est valide pour un événement de type 3. Pour les autres types d'événements, il doit être mis à 0.

```
_ASYNC_IND_PARAM ::= SEQUENCE {
    pNodeType      BIT STRING SIZE (4),
    pPossibilitySM  BIT STRING SIZE (1),
    pMeasureDly    BIT STRING SIZE (1),
    réserve        BIT STRING SIZE (2)
}
```

pNodeType

Un champ indiquant le type d'opération.

- 1: Appareil principal;
- 2: Appareil subordonné.

pPossibilitySM

Fanion permettant d'identifier la capacité de traitement SM.

- 0: Avec capacité SM;
- 1: Sans capacité SM.

pMeasureDly

Il s'agit d'un fanion destiné à identifier l'achèvement de la mesure du retard de l'appareil principal, qui ne doit être défini que pour un appareil principal. Pour un appareil subordonné, cela doit être mis à 0.

- 0: Mesure du retard de l'appareil principal non mise en œuvre;
- 1: Mesure du retard de l'appareil principal terminée.

réserve

Il s'agit d'une zone de réserve. Mise à 0,

- Champ pUserData

Ce champ doit stocker le contenu de la notification, et cela est valide pour le type d'événement 2, le type d'événement 3, le type d'événement 6 et le type d'événement 7. Ce champ ne doit pas être ajouté lorsqu'il n'est pas valide. Il existe quatre types de formats par le contenu de notification.

```

_ASYNC_IND_DATA ::= SEQUENCE {
    pSMDlyTime      Unsigned32,
    pNumStaInfo     Unsigned32,
    ( SEQUENCE {
        pPortNum     Usingned8,
        réserve      Usingned8,
        pMACAddr     Usinged48
    } ) +
}
| SEQUENCE {
    pRemainNum      Unsigned8
    réserve         Unsigned8,
    pNumNode        Unsigned16,
    ( pNodeInfo     _ASYNC_IND_NODE_INFO ) +
}
| ( pDlyTime      Unsigned32 )
| SEQUENCE {
    pTagMACAddr     Unsigned48,
    pTagAddr        Unsigned16
}

```

-- Réponse de notification de démarrage

-- Notification de liste de nœuds voisins

-- Mesure du retard terminée

-- Changement d'adresse de nœud

pSMDlyTime

Ce champ doit indiquer le temps de retard du SM, et cela est valide pour l'événement de type 3. Le temps de retard du SM doit être défini. Lorsque la mesure n'est pas exécutée ou n'est pas enregistrée, ce champ doit être mis à 0. (Unité: 1 ns)

pNumStaInfo

Ce champ doit indiquer le nombre d'informations relatives au nœud voisin et cela est valide pour l'événement de type 3. Le nombre d'informations de nœud voisin stockées doit être défini.

pPortNum

Ce champ doit indiquer le numéro de port du nœud voisin et cela est valide pour l'événement de type 3.

réserve

Il s'agit d'une zone de réserve. Mise à 0. .

pMACAddr

Ce champ doit indiquer l'adresse MAC du nœud voisin et cela est valide pour l'événement de type 3.

pRemainNum

Ce champ doit indiquer le nombre de paquets restants et cela est valide pour l'événement de type 6. Le nombre de paquets restants après la division en plusieurs paquets doit être défini.

pNumNode

Ce champ doit indiquer le nombre total de nœuds et cela est valide pour l'événement de type 6. Le nombre total de nœuds, y compris l'appareil principal, doit être défini. (Nombre maximal: 128)

pNodeInfo

Ce champ doit stocker les informations relatives au nœud, et cela est valide pour l'événement de type 6.


```

_ASYNC_IND_NODE_INFO ::= SEQUENCE {
    pStaAddr      Unsigned16,
    réserve       Unsigned8,
    pNumStaInfo   Unsigned8,
    ( SEQUENCE {
        pPortNum   Unsigned8,
        réserve    Unsigned8,
        pStaAddr   Unsigned16
    } ) +
}

```

pStaAddr

Ce champ doit stocker l'adresse du nœud.

réserve

Il s'agit d'une zone de réserve. Mise à 0.

pNumStaInfo

Ce champ doit indiquer le nombre d'informations du nœud voisin. Le nombre d'informations de nœud voisin stockées doit être défini.

pNodeInfo

Ce champ doit indiquer le numéro de port du nœud voisin.

pStaAddr

Ce champ doit stocker l'adresse de nœud du nœud voisin.

pMDlyTime

Ce champ doit indiquer le temps de retard de l'appareil principal et cela est valide pour l'événement de type 6. Le temps de retard de l'appareil principal doit être défini. Lorsque la mesure n'est pas exécutée ou n'est pas enregistrée, ce champ doit être mis à 0. (Unité: 1 ns)

pTagMACAddr

Ce champ doit indiquer l'adresse MAC du nœud cible de changement d'adresse de nœud et cela est valide pour le type d'événement 7. L'adresse MAC du nœud cible de changement d'adresse de nœud doit être définie.

pTagAddr

Ce champ doit indiquer l'adresse de nœud du nœud cible de changement d'adresse de nœud et cela est valide pour le type d'événement 7. L'adresse de nœud du nœud cible de changement d'adresse de nœud doit être définie.

NOTE Lors de l'envoi de l'événement de type 1, l'adresse de multidiffusion avec le nœud voisin comme destination doit être utilisée comme adresse MAC de destination. Pour d'autres types d'événements, l'adresse de multidiffusion commune doit être utilisée comme adresse MAC de destination (voir le Tableau 4).

4.3.4.11 Instruction EVD

L'instruction EVD doit être utilisée pour une notification d'événement.

```

_EVD-FIDAP-PDU ::= _FIDAP-PDU ( WITH COMPONENTS
    { ..., fcmd (evd), fHS,
      pduBody (fcmdBody _EVD-FIDAPBody)
    }
)

```

```

_EVD-FIDAPBody ::= SEQUENCE {
    pDstAddress  Unsigned16,
    pSrcAddress  Unsigned16,
    pLength      Unsigned16,
    réserve      Unsigned16,
    pUserData    _FDCServicePDU +
}

```

- Champ fcmd
Ce champ doit indiquer l'instruction EVD.
- Champ fHS
Ce champ doit indiquer une prise de contact/notification avec la valeur suivante:
 - '00'H: Normalement (non utilisé).
- Champ pDstAddress
Ce champ doit indiquer l'adresse du nœud de destination. L'adresse de l'appareil principal de destination doit être définie (voir le Tableau 9).
- Champ pSrcAddress
Ce champ doit indiquer l'adresse du nœud source. L'adresse de l'appareil principal source doit être définie (voir le Tableau 9).
- Champ pLength
Ce champ doit indiquer le nombre d'octets de données d'utilisateur.
- réserve
Il s'agit d'une zone de réserve. Mise à 0.
- Champ pUserData
Ce champ doit stocker les données d'utilisateur. Stocke _FDCServicePDU (voir 4.3.5).

4.3.5 PDU pour le service de commande de l'appareil de terrain

4.3.5.1 Vue d'ensemble

FDCServicePDU doit être édité avec une SDU dans la primitive de service FDC et être envoyé par l'intermédiaire de l'AR. Un ASE FDC reçoit la PDU de l'ASE homologue par l'intermédiaire de l'AR.

Le type de PDU de l'instruction FDC: _CMD_PDU doit être classé dans l'instruction de type sync (instruction synchrone): _SYNCType-CMD-PDU et instruction de type async (instruction asynchrone): _ASYNCType-CMD-PDU. Les deux types sont des formats PDU identiques: _UCMD-PDU. Par conséquent, la FDC PM ne peut pas distinguer les PDU de leurs types. Il convient que l'utilisateur de l'ASE FDC sache quelles instructions appartiennent aux instructions synchrones ou celles qui ne lui appartiennent pas, et puisse distinguer les types à l'aide du champ cmd en tant que code clé dans l'en-tête PDU.

PDU d'instruction asynchrone: _ASYNCType-CMD_PDU doit inclure _UCMD-PDU et quatorze autres types de PDU en tant qu'instructions communes, qui sont également des variantes de _UCMD-PDU.

Un ensemble d'instructions et de réponses, déduit de _UCMD-PDU et de _URSP-PDU présentés ci-après, peut être défini pour une application particulière (commande d'entraînement moteur, servocommande, etc.) par les utilisateurs FAL. Cet ensemble peut être enregistré en tant que profil d'appareil (voir 4.5 et l'Annexe A). De plus, un utilisateur FAL peut choisir les profils d'appareil pris en charge par un appareil de terrain cible, lorsqu'une connexion est établie par le service FDC-Connect (voir 8.4.4.3 et l'IEC 61158-5-27).

_CMD-PDU	::= _SYNCType-CMD-PDU _ASYNCType-CMD-PDU
_SYNCType-CMD-PDU	::= _UCMD-PDU
_ASYNCType-CMD-PDU	::= _UCMD-PDU _NOP-CMD-PDU _PRM_RD-CMD-PDU _PRM_WR-CMD-PDU _ID_RD-CMD-PDU _CONFIG-CMD-PDU _ALM_RD-CMD-PDU _ALM_CLR-CMD-PDU _SYNC_SET-CMD-PDU _CONNECT-CMD-PDU _DISCONNECT-CMD-PDU _PPRM_RD-CMD-PDU _PPRM_WR-CMD-PDU _MEM_RD-CMD-PDU _MEM_WR-CMD-PDU
_UCMD-PDU	::= _CMD3-PDU
_NOP-CMD-PDU	::= _NOP-CMD3-PDU
_PRM_RD-CMD-PDU	::= _PRM_RD-CMD3-PDU
_PRM_WR-CMD-PDU	::= _PRM_WR-CMD3-PDU
_ID_RD-CMD-PDU	::= _ID_RD-CMD3-PDU
_CONFIG-CMD-PDU	::= _CONFIG-CMD3-PDU
_ALM_RD-CMD-PDU	::= _ALM_RD-CMD3-PDU
_ALM_CLR-CMD-PDU	::= _ALM_CLR-CMD3-PDU
_SYNC_SET-CMD-PDU	::= _SYNC_SET-CMD3-PDU
_CONNECT-CMD-PDU	::= _CONNECT-CMD3-PDU
_DISCONNECT-CMD-PDU	::= _DISCONNECT-CMD3-PDU
_PPRM_RD-CMD-PDU	::= _PPRM_RD-CMD3-PDU
_PPRM_WR-CMD-PDU	::= _PPRM_WR-CMD3-PDU
_MEM_RD-CMD-PDU	::= _MEM_RD-CMD3-PDU
_MEM_WR-CMD-PDU	::= _MEM_WR-CMD3-PDU

Le type de PDU de réponse FDC: _RSP_PDU doit également être classé dans la commande de type sync (réponse de synchronisation): _SYNCType-RSP-PDU et commande de type async (réponse asynchrone): _ASYNCType-RSP-PDU. Les deux types doivent être déduits de formats PDU identiques: _URSP-PDU. Par conséquent, la FDC PM ne peut pas distinguer les PDU de leurs types. Il convient que l'utilisateur de l'ASE FDC sache quelles réponses appartiennent aux réponses synchrones ou celles qui ne lui appartiennent pas, et puisse distinguer les types à l'aide du champ rcmd en tant que code clé dans l'en-tête PDU. De plus, le code de préfixe d'une PDU de réponse correspond à celui d'une PDU d'instruction, une réponse devant être échangée contre une instruction.

Le type de PDU de réponse asynchrone: _ASYNCType-RSP_PDU doit inclure _URSP-PDU et quatorze autres types de PDU en tant que réponses communes à une application, qui sont également des variantes de _URSP-PDU.

```

_RSP-PDU ::= _SYNCType-RSP-PDU
           | _ASYNCType-RSP-PDU
_SYNCType-RSP-PDU ::= _URSP-PDU
_ASYNCType-RSP-PDU ::= _URSP-PDU
                    | _NOP-RSP-PDU
                    | _PRM_RD-RSP-PDU
                    | _PRM_WR-RSP-PDU
                    | _ID_RD-RSP-PDU
                    | _CONFIG-RSP-PDU
                    | _ALM_RD-RSP-PDU
                    | _ALM_CLR-RSP-PDU
                    | _SYNC_SET-RSP-PDU
                    | _CONNECT-RSP-PDU
                    | _DISCONNECT-RSP-PDU
                    | _PPRM_RD-RSP-PDU
                    | _PPRM_WR-RSP-PDU
                    | _MEM_RD-RSP-PDU
                    | _MEM_WR-RSP-PDU
_URSP-PDU ::= _RSP3-PDU
_NOP-RSP-PDU ::= _NOP-RSP3-PDU
_PRM_RD-RSP-PDU ::= _PRM_RD-RSP3-PDU
_PRM_WR-RSP-PDU ::= _PRM_WR-RSP3-PDU
_ID_RD-RSP-PDU ::= _ID_RD-RSP3-PDU
_CONFIG-RSP-PDU ::= _CONFIG-RSP3-PDU
_ALM_RD-RSP-PDU ::= _ALM_RD-RSP3-PDU
_ALM_CLR-RSP-PDU ::= _ALM_CLR-RSP3-PDU
_SYNC_SET-RSP-PDU ::= _SYNC_SET-RSP3-PDU
_CONNECT-RSP-PDU ::= _CONNECT-RSP3-PDU
_DISCONNECT-RSP-PDU ::= _DISCONNECT-RSP3-PDU
_PPRM_RD-RSP-PDU ::= _PPRM_RD-RSP3-PDU
_PPRM_WR-RSP-PDU ::= _PPRM_WR-RSP3-PDU
_MEM_RD-RSP-PDU ::= _MEM_RD-RSP3-PDU
_MEM_WR-RSP-PDU ::= _MEM_WR-RSP3-PDU

```

4.3.5.2 Composant commun de la PDU

4.3.5.2.1 PDU d'instruction FDC

_CMD-PDU a un type amélioré: _CMD3-PDU.

La longueur pduBody de _CMD3-PDU doit pouvoir être sélectionnée entre 4, 12, 28, 44, et 60 octets, mais elle peut ne pas être commutée à la volée.

```

_CMD3-PDU ::= SEQUENCE {
    cmd      _CMDCode,
    wdt      _WDT,
    cmd_ctrl _CMD_CTRL,
    pduBody  OCTET STRING SIZE (4|12|28|44|60)
}
-- type de PDU
-- améliorée

```

- Champ cmd

cmd

champ de données contenant le code d'identification du contenu de chaque instruction

Ce champ doit être codé en `_CMDCode`, sous-type d'Unsigned8, avec la plage de valeurs ci-dessous. Les instructions communes prédéfinies sont présentées en tant que `_PrimaryCMD` ci-dessous.

- '00'H à '1F'H: utilisé ou réservé pour les instructions communes;
- '20'H à 'BF'H: réservé aux instructions spécifiques à l'application;
- 'C0'H à 'FF'H: réservé aux fournisseurs.

```
_CMDCode      ::= Unsigned8
_PrimaryCMD    ::= _CMDCode (
    nop ('00'H),
    prm_rd ('01'H), prm_wr ('02'H),
    id_rd ('03'H),
    config ('04'H),
    alm_rd ('05'H), alm_clr ('06'H),
    sync_set ('0D'H),
    connect ('0E'H),
    disconnect ('0F'H),
    pprm_rd ('1B'H), pprm_wr ('1C'H),
    mem_rd ('1D'H), mem_wr ('1E'H)
)
```

- Champ wdt

wdt

champ du compteur d'horloge de surveillance

Ce champ doit être utilisé pour permettre à l'appareil subordonné de détecter l'activité non synchrone ou l'erreur WDT de l'appareil principal correspondant. L'appareil principal doit compter le champ wdt des PDU d'envoi à chaque cycle de communication en cas d'état de communication synchrone. De plus, l'appareil subordonné doit examiner la valeur du champ des PDU destinataires à chaque cycle de communication pour détecter l'erreur wdt.

Le type de données est `_WDT`. Il doit comporter deux sous-champs: mn et sn.

```
_WDT          ::= SEQUENCE {
    mn  BIT STRING SIZE (4),
    sn  BIT STRING SIZE (4)
}
```

mn (valeur de comptage de l'appareil principal)

La plage de valeurs doit s'étendre de 0 à 15.

Le champ mn d'une CMD-PDU suivante à envoyer doit faire l'objet d'un comptage progressif de un. Un appareil principal FDC doit procéder ainsi à chaque cycle de communication. D'autre part, lorsqu'un appareil subordonné FDC reçoit la PDU, il doit examiner le compteur à chaque cycle. Si le compteur ne correspond pas à la dernière valeur mn plus un, il doit alerter le compteur de l'horloge de surveillance de l'erreur.

sn (valeur de comptage de l'appareil subordonné)

La plage de valeurs doit s'étendre de 0 à 15.

La même valeur que le champ rsn de la dernière RSP-PDU reçue doit être attribuée au champ sn d'une CMD-PDU suivante à envoyer. Un appareil principal FDC doit la copier depuis la dernière RSP-PDU à chaque cycle de communication.

- Champ cmd_ctrl

cmd_ctrl

ensemble de bits de contrôle indépendants de toute transaction d'instruction, entre l'appareil principal FDC et l'appareil subordonné FDC

Le type de données est _CMD_CTRL. Il doit comporter deux sous-champs significatifs: alm_clr et cmd_id.

```
_CMD_CTRL ::= SEQUENCE {
    reserve1  BIT STRING SIZE (3),
    alm_clr   BIT STRING SIZE (1),
    reserve2  BIT STRING SIZE (2),
    cmd_id    BIT STRING SIZE (2),
    reserve3  BIT STRING SIZE (8)
}
```

alm_clr

Ce champ doit être utilisé pour une instruction de libération de l'état d'alarme/avertissement de l'appareil subordonné.

- valeur 0: ne rien exécuter
- la valeur passe à 1: exécuter la libération de l'état.

cmd_id

Si l'appareil principal adresse en permanence le contenu d'une CMD-PDU à l'appareil subordonné, il peut être utilisé pour permettre à l'appareil subordonné de reconnaître l'instruction comme étant une instruction différente.

La plage de valeurs doit s'étendre de 0 à 3.

L'appareil principal FDC peut utiliser cmd_id pour permettre à un appareil subordonné FDC de reconnaître qu'il s'agit d'une nouvelle instruction lorsque l'appareil principal envoie la même instruction à l'appareil subordonné de manière répétée.

Étant donné que l'appareil subordonné doit renvoyer l'écho du champ cmd_id de chaque instruction par l'intermédiaire du champ RSP-PDU.cmd_stat.rcmd_id présenté ci-après, l'appareil principal peut également juger l'instruction pour laquelle la station subordonnée a envoyé la réponse.

L'utilisation du champ cmd_id n'est pas obligatoire pour l'appareil principal.

Étant donné que RSP-PDU.cmd_stat.cmdRdy = 0 dans les PDU de réponse, l'appareil subordonné doit ignorer les instructions dont le champ cmd_id est différent, et poursuivre l'exécution de l'instruction qui a déjà été acceptée.

4.3.5.2.2 PDU de réponse FDC

_RSP-PDU a un type amélioré: _RSP3-PDU.

La longueur pduBody de _RSP3-PDU doit pouvoir être sélectionnée entre 4, 12, 28, 44, et 60 octets, mais elle peut ne pas être commutée à la volée.

```
_RSP3-PDU ::= SEQUENCE {
    rcmd      _CMDCode,
    rwdt      _RWDT,
    cmd_stat  _CMD_STAT,
    pduBody   OCTET STRING SIZE (4|12|28|44|60)
}
```

-- type de PDU améliorée

- Champ rcmd

rcmd

champ de données contenant le même code qu'une instruction reçue et de traitement

Ce champ doit donc être codé en tant que _CMDCode, voir 4.3.5.2.1.

- Champ alarm

alarm

Ce champ doit contenir le code d'alarme.

Le type de données est _ALARM. La plage de valeurs doit s'étendre de 0 à 255.

```
_ALARM ::= Unsigned8
```

- Champ status

status

Ce champ doit indiquer l'état de l'appareil subordonné.

Le type de données est _STATUS. Il doit comporter trois sous-champs significatifs, comme suit.

```
_STATUS ::= SEQUENCE {
    alarm      BIT STRING SIZE (1),
    warning    BIT STRING SIZE (1),
    cmdRdy     BIT STRING SIZE (1),
    réserve    BIT STRING SIZE (13)
}
```

alarm

Ce champ binaire doit indiquer l'état d'alarme dans l'appareil subordonné.

- 0: Pas d'alarme;
- 1: Des alarmes se sont produites.

warning

Ce champ binaire doit indiquer l'état d'avertissement dans l'appareil subordonné.

- 0: Pas d'avertissement;
- 1: Des avertissements se sont produits.

cmdRdy

Ce champ binaire doit indiquer l'état de progression de l'instruction de l'appareil subordonné FDC.

L'appareil subordonné doit conserver le bit cmdRdy 0 lors du traitement de l'instruction. A l'issue du traitement, l'appareil subordonné doit remplacer le bit 0 par le bit 1.

Pour signaler l'exécution de l'instruction spécifique, l'appareil principal FDC ne doit pas uniquement examiner le bit, mais également certains autres champs RSP-PDU contenant le champ CMD-PDU correspondant afin de faire la distinction avec la transaction précédente.

Si la durée de rétention de cmdRdy = 0 a expiré, l'appareil principal doit générer une erreur de temporisation d'instruction. La valeur du temporisateur dépend de chaque spécification de produit pour les appareils subordonnés.

Une modification de l'état de ce bit doit être indépendante de l'état d'alarme ou d'avertissement.

- 0: occupé avec exécution d'instruction en cours;
- 1: prêt pour la nouvelle instruction.

- Champ rwdt

rwdt

champ du compteur d'horloge de surveillance

Ce champ doit être utilisé pour permettre à l'appareil principal de détecter l'activité non synchrone ou l'erreur WDT de l'appareil subordonné correspondant. L'appareil subordonné doit compter et mettre à jour le champ rwdt des PDU d'envoi à chaque cycle de communication en cas d'état de communication synchrone. De plus, l'appareil principal doit examiner la valeur du champ des PDU destinataires à chaque cycle de communication pour détecter l'erreur rwdt.

Le type de données est _RWDT. Il doit comporter deux sous-champs: rmn et rsn.

```
_RWDT ::= SEQUENCE {
    rmn  BIT STRING SIZE (4),
    rsn  BIT STRING SIZE (4)
}
```

rmn (valeur de comptage de l'appareil principal)

La plage de valeurs doit s'étendre de 0 à 15.

La même valeur que le champ mn de la dernière CMD-PDU reçue doit être attribuée au champ rmn d'une RSP-PDU suivante à envoyer. Un appareil subordonné FDC doit la copier depuis la CMD-PDU à chaque cycle de communication.

rsn (valeur de comptage de l'appareil subordonné)

La plage de valeurs doit s'étendre de 0 à 15.

Le champ rsn d'une RSP-PDU suivante à envoyer doit faire l'objet d'un comptage progressif de un. Un appareil subordonné FDC doit procéder ainsi à chaque cycle de communication. D'autre part, lorsqu'un appareil principal FDC reçoit la PDU, il doit examiner le compteur à chaque cycle. Si le compteur ne correspond pas à la dernière valeur rsn plus un, il doit alerter le compteur de l'horloge de surveillance de l'erreur.

- Champ cmd_stat

cmd_stat

Ce champ doit indiquer l'état de l'appareil subordonné.

Le type de données est _CMD_STAT. Il doit comporter sept sous-champs, comme suit.

```
_CMD_STAT ::= SEQUENCE {
    d_alm          BIT STRING SIZE (1),
    d_war          BIT STRING SIZE (1),
    cmdRdy         BIT STRING SIZE (1),
    alm_clr_cmp    BIT STRING SIZE (1),
    réserve        BIT STRING SIZE (2),
    rcmd_id        BIT STRING SIZE (2),
    cmd_alm        BIT STRING SIZE (4),
    comm_alm       BIT STRING SIZE (4)
}
```

d_alm

Ce champ binaire doit indiquer l'état d'occurrence de l'alarme spécifique à l'appareil dans l'appareil subordonné.

- 0: l'appareil subordonné n'est pas dans un état d'alarme spécifique à l'appareil;
- 1: l'appareil subordonné se trouve dans un état d'alarme spécifique à l'appareil, à l'exception de comm_alm et de cmd_alm.

La spécification de l'état d'alarme de l'appareil dépend de la mise en œuvre du produit spécifiée, mais il convient de ne pas classer un facteur d'alarmes parmi les problèmes de communication et d'instruction.

Après la reprise de l'appareil subordonné à la suite de l'état d'alarme, au moyen de l'exécution de l'instruction ALM_CLR-CMD-PDU ou du bit cmd_ctrl.alm_clr dans toute CMD-PDU, ce bit doit être mis à 0 (d_alm = 0)

d_war

Ce champ binaire doit indiquer l'état d'occurrence de l'avertissement spécifique à l'appareil dans la station subordonnée.

- 0: l'appareil subordonné n'est pas dans un état d'alarme spécifique à l'appareil;
- 1: l'appareil subordonné se trouve dans un état d'alarme spécifique à l'appareil, à l'exception de comm_alm et de cmd_alm.

La spécification de l'état d'alarme de l'appareil dépend de la mise en œuvre du produit spécifiée, mais il convient de ne pas classer un facteur d'alarmes parmi les problèmes de communication et d'instruction.

Après la reprise de l'appareil subordonné à la suite de l'état d'avertissement, au moyen de l'exécution de l'instruction ALM_CLR-CMD-PDU ou du bit cmd_ctrl.alm_clr dans toute CMD-PDU, ce bit doit être mis à 0 (d_war = 0)

cmdRdy

Ce champ binaire doit indiquer l'état de progression de l'instruction de l'appareil subordonné FDC.

Ce bit a pour signification status.cmdRdy.

- 0: occupé avec exécution d'instruction en cours;
- 1: prêt pour la nouvelle instruction.

alm_clr_cmp

Ce champ binaire doit indiquer l'état d'exécution du processus de libération d'alarme.

- 0: Libération d'alarme/avertissement non terminée;
- 1: Libération d'alarme/avertissement terminée.

rcmd_id

Ce champ doit être un code clé permettant d'indiquer à quelle instruction correspond la PDU de réponse, en écho au cmd_id de la PDU d'instruction correspondante.

La plage de valeurs est comprise entre 0 et 3.

cmd_alm

Ce champ doit notifier un code d'alarme en cas d'anomalie d'instruction.

La signification de chaque code doit être la suivante:

- '00'H: Normal;
- '01'H: Avertissement sur données en dehors de la plage;
- '02'H à '07'H: réservé aux codes d'avertissement;
- '08'H: Alarme sur absence de prise en charge;
- '09'H: Alarme sur données en dehors de la plage;
- '0A'H: Alarme sur condition d'exécution d'instruction anormale;
- '0B'H: Alarme sur combinaison de sous-instruction anormale;
- '0C'H: Alarme sur phase anormale;
- '0D'H à '0F'H: réservé aux codes d'alarme;

comm_alm

Ce champ doit notifier un code d'alarme pour l'état d'erreur de communication.

La signification de chaque code doit être la suivante:

- '00'H: Normal;
- '01'H: Avertissement sur FCS anormale;
- '02'H: Avertissement sur réception anormale;
- '03'H à '07'H: réservé à l'avertissement;
- '08'H: Alarme sur FCS anormale;
- '09'H: Alarme sur réception anormale;
- '0A'H à '0F'H: réservé à l'alarme.

4.3.6 PDU pour le service de message

```

_MSGREQ-PDU ::= SEQUENCE {
    responder      Unsigned8,
    funcCode       BIT STRING SIZE (7),
    réserve1       BIT STRING SIZE (1),
    extAddress     Unsigned8,
    réserve2       OCTET STRING ('00'H),
    requestData    OCTET STRING SIZE (4..4092)
}
| requestUData   OCTET STRING SIZE (8..4096)

```

```

_MSGRSP-PDU ::= SEQUENCE {
    responder      Unsigned8,
    funcCode       BIT STRING SIZE (7),
    errorFlag      BIT STRING SIZE (1),
    extAddress     Unsigned8,
    réserve        OCTET STRING ('00'H),
    responseData   OCTET STRING SIZE (4..4092)
}
| responseUData  OCTET STRING SIZE (8..4096)

```

- Champ responder

responder

Ce champ doit contenir l'adresse de nœud de destination ou le répondeur du message.

Dans ce champ, la valeur de l'octet 0 de l'adresse de nœud doit être définie (voir le Tableau 9).

- Champ funcCode

funcCode

Ce champ doit contenir le code indiquant la fonction du message.

La plage de codes doit être comprise entre '00'H et '7F'H.

- Champ errorFlag

errorFlag

Ce champ doit indiquer l'état de réponse du message. La valeur 0 doit être attribuée pour une réponse normale et la valeur 1 pour une réponse anormale.

- 0: réponse normale;
- 1: réponse erronée.

- Champ extAddress

extAddress

Ce champ doit contenir la sous-adresse représentant un sous-appareil lorsque le nœud de destination est un appareil intégré à au moins deux sous-appareils.

Dans ce champ, la valeur de l'octet 0 de l'adresse de nœud doit être définie (voir le Tableau 9).

- Champs requestData et responseData

requestData et responseData

Ces champs doivent contenir les SDU qui sont définies par l'utilisateur pour chaque code de fonction. Un exemple d'ensemble d'instructions de message utilisateur pour le code de fonction est présenté ci-après: '42'H est indiqué dans l'Annexe C.

4.4 Définitions détaillées des _FDCService-PDUs

4.4.1 Type de PDU amélioré

4.4.1.1 Instruction NOP et réponse

NOP symbolise une instruction "no operation" (pas d'opération). A la réception de cette instruction, l'appareil subordonné ne doit rien faire, sauf répondre par une NOP-RSP3-PDU indiquant les derniers état et alarme.

```
_NOP-CMD3-PDU      ::= _CMD3-PDU ( WITH COMPONENTS
                        { ..., cmd (nop)
                        } )
_NOP-RSP3-PDU      ::= _RSP3-PDU ( WITH COMPONENTS
                        { ..., rcmd (nop)
                        } )
```

4.4.1.2 Instruction PRM_RD et réponse

PRM_RD symbolise une instruction "parameter-read" (lecture de paramètre). A la réception de cette instruction, l'appareil subordonné doit lire une valeur de paramètre spécifiée dans le champ PRM_RD-RSP3-PDU.parameter et doit y répondre

```
_PRM_RD-CMD3-PDU   ::= _CMD3-PDU ( WITH COMPONENTS
                        { ..., cmd (prm_rd),
                        pduBody (cmdBody _PRM_RD-CMD3Body)
                        } )
_PRM_RD-CMD3Body   ::= SEQUENCE {
                        pNo      Unsigned16,
                        pSize    Unsigned8,
                        réserve   OCTET STRING SIZE (1|9|25|41|57)
                        }
_PRM_RD-RSP3-PDU   ::= _RSP3-PDU ( WITH COMPONENTS
                        { ..., rcmd (prm_rd),
                        pduBody (rspBody _PRM_RD-RSP3Body)
                        } )
_PRM_RD-RSP3Body   ::= SEQUENCE {
                        pNo      Unsigned16,
                        pSize    Unsigned8,
                        réserve   OCTET STRING SIZE (1),
                        parameter OCTET STRING SIZE (0|8|24|40|56)
                        }
```

- Champ pNo

pNo

Ce champ doit contenir le nombre de paramètres à lire.

La plage de valeurs doit s'étendre de 0 à 65 535.

- Champ pSize

pSize

Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit s'étendre de 0 à 255.

- Champ parameter

parameter

Ce champ doit contenir les données lues correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.4.1.3 Instruction PRM_WR et réponse

PRM_WR symbolise une instruction "parameter-write" (écriture de paramètre). A la réception de cette instruction, l'appareil subordonné doit écrire une valeur de paramètre spécifiée dans le champ PRM_WR-CMD3-PDU.parameter et doit répondre en écho pour signaler l'exécution de l'instruction.

```

_PRM_WR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (prm_wr),
      pduBody (cmdBody _PRM_WR-CMD3Body)
    } )
_PRM_WR-CMD3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    réserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}
_PRM_WR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (prm_wr),
      pduBody (rspBody _PRM_WR-RSP3Body)
    } )
_PRM_WR-RSP3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    réserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}

```

- Champ pNo

pNo

Ce champ doit contenir le nombre de paramètres à écrire.

La plage de valeurs doit s'étendre de 0 à 65 535.

- Champ pSize

pSize

Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit s'étendre de 0 à 255.

- Champ parameter

parameter

Ce champ doit contenir les données à écrire correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.4.1.4 Instruction ID_RD et réponse

ID_RD symbolise une instruction "device-ID-read" (lecture d'ID d'appareil). A la réception de cette instruction, l'appareil subordonné doit lire une partie d'un élément spécifié dans les informations relatives à l'appareil dans le champ ID_RD-RSP3-PDU.idData et doit y répondre. La sémantique et les types de données des informations relatives à l'appareil peuvent dépendre du modèle d'appareil (voir l'Annexe B).

```

_ID_RD-CMD3-PDU      ::= _CMD3-PDU ( WITH COMPONENTS
                        { ..., cmd (id_rd),
                          pduBody (cmdBody _ID_RD-CMD3Body)
                        } )
_ID_RD-CMD3Body      ::= SEQUENCE {
                        idCode   Unsigned8,
                        idOffcet  Unsigned8,
                        idSize    Unsigned16,
                        réserve   OCTET STRING SIZE (0|8|24|40|56)
                        }
_ID_RD-RSP3-PDU      ::= _RSP3-PDU ( WITH COMPONENTS
                        { ..., rcmd (id_rd),
                          pduBody (rspBody _ID_RD-RSP3Body)
                        } )
_ID_RD-RSP3Body      ::= SEQUENCE {
                        idCode   Unsigned8,
                        idOffcet  Unsigned8,
                        idSize    Unsigned16,
                        idData    OCTET STRING SIZE (0|8|24|40|56)
                        }

```

- Champ idCode

idCode

Ce champ doit contenir le code d'ID d'appareil à lire.

La plage de valeurs doit s'étendre de 0 à 255, voir B.3.2.

- '00'H: réservé;
- '01'H: Code d'ID fournisseur;
- '02'H: Code d'appareil ou code de modèle d'appareil;
- '03'H: Version de l'appareil;
- '04'H: Version du fichier d'informations relatives à l'appareil;
- '05'H: Nombre d'adresses étendues;
- '06'H: Numéro de série du produit;
- '07'H à '0F'H: Réservé;
- '10'H: Code de profil de l'appareil pris en charge (primaire);
- '11'H: Version de profil de l'appareil pris en charge (primaire);
- '12'H: Code de profil de l'appareil pris en charge (secondaire);
- '13'H: Version de profil de l'appareil pris en charge (secondaire);
- '14'H: Code de profil de l'appareil pris en charge (tertiaire);
- '15'H: Version de profil de l'appareil pris en charge (tertiaire);

- '16'H: Cycle de transmission minimal;
- '17'H: Cycle de transmission maximal;
- '18'H: Granularité du cycle de transmission;
- '19'H: Cycle de communication minimal;
- '1A'H: Cycle de communication maximal;
- '1B'H: Nombre d'octets transmissibles;
- '1C'H: Nombre d'octets transmis (paramètre en cours);
- '1D'H: Code de profil d'appareil (paramètre en cours);
- '1E'H à '1F'H: Réservé;
- '20'H: Liste de modes de communication pris en charge;
- '21'H: Adresse MAC;
- '22'H à '2F'H: Réservé;
- '30'H: Liste d'instructions principales prises en charge;
- '31'H à '37'H: réservés;
- '38'H: Liste de sous-instructions principales prises en charge;
- '39'H à '3F'H: Réservé;
- '40'H: Liste de paramètres communs pris en charge;
- '41'H à '7F'H: Réservé;
- '80'H: Nom de l'appareil principal;
- '81'H à '8F'H: Réservé;
- '90'H: Nom du sous-appareil 1;
- '91'H à '97'H: réservés;
- '98'H: Version du sous-appareil 1;
- '99'H à '9F'H: Réservé;
- 'A0'H: Nom du sous-appareil 2;
- 'A1'H à 'A7'H: Réservé;
- 'A8'H: Version du sous-appareil 2;
- 'A9'H à 'AF'H: Réservé;
- 'B0'H: Nom du sous-appareil 3;
- 'B1'H à 'B7'H: Réservé;
- 'B8'H: Version du sous-appareil 3;
- 'B9'H à 'BF'H: Réservé;
- 'C0'H à 'FF'H: Réservé aux informations spécifiques au fournisseur.

- Champ idOffset

idOffset

Ce champ doit contenir l'adresse de décalage de l'ID d'appareil à lire.

La plage de valeurs doit s'étendre de 0 à 255.

- Champ idSize

idSize

Ce champ doit contenir la taille de lecture en octets de l'ID d'appareil.

La plage de valeurs doit s'étendre de 0 à 255.

- Champ idData

idData

Ce champ doit contenir les données lues correspondant au champ idCode. La taille des données, la structure des données et leur signification dépendent du champ idCode, voir B.3.2.

4.4.1.5 Instruction CONFIG et réponse

CONFIG symbolise une instruction "configure-device" (configurer appareil). A la réception de cette instruction, l'appareil subordonné doit activer un ensemble de paramètres sur la RAM mise à jour avec une instruction PAR_WR, puis doit répondre pour signaler l'exécution de l'instruction.

```

_CONFIG-CMD3-PDU      ::= _CMD3-PDU ( WITH COMPONENTS
                        { ..., cmd (config),
                          pduBody (cmdBody _CONFIG-CMD3Body)
                        } )
_CONFIG-CMD3Body      ::= SEQUENCE {
                        config_mode  Unsigned8 {
                            pActive (0),
                            pAllSave (1),
                            pReset (2)
                        },
                        réserve  OCTET STRING SIZE (3|11|27|43|59)
                        }
_CONFIG-RSP3-PDU      ::= _RSP3-PDU ( WITH COMPONENTS
                        { ..., rcmd (config),
                          pduBody (rspBody _CONFIG-RSP3Body)
                        } )
_CONFIG-RSP3Body      ::= SEQUENCE {
                        config_mode  Unsigned8 {
                            pActive (0),
                            pAllSave (1),
                            pReset (2)
                        },
                        réserve  OCTET STRING SIZE (3|11|27|43|59)
                        }

```

- Champ config_mode

config_mode

Ce champ doit contenir le mode de configuration de l'appareil.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

- pActive ('00'H): pour procéder à un nouveau calcul avec les paramètres et la configuration;
- pAllSave ('01'H): pour écrire des paramètres dans la mémoire non volatile avec un lot;
- pReset ('02'H): pour rétablir le jeu de paramètres d'usine;
- '03'H à 'FF'H: réservé.

4.4.1.6 Instruction ALM_RD et réponse

ALM_RD symbolise une instruction "alarm-read" (lecture d'alarme). A la réception de cette instruction, l'appareil subordonné doit lire un ensemble de données d'alarme comprenant les détails de l'état d'alarme et d'avertissement dans le champ ALM_RD-RSP3-PDU.alm_data, puis il doit répondre avec ces éléments. La sémantique et les types de données du champ alm_data dépendent du modèle de chaque appareil.

```

_ALM_RD-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (alm_rd),
      pduBody (cmdBody _ALM_RD-CMD3Body)
    } )
_ALM_RD-CMD3Body ::= SEQUENCE {
    alm_rd_mode Unsigned16 {
        currentAlm (0),
        historyAlm (1),
        cAlmDetail (2),
        hAlmDetail (3)
    },
    alm_index Unsigned16 (0..11),
    réserve OCTET STRING SIZE (0|8|24|40|56)
}
_ALM_RD-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (alm_rd),
      pduBody (cmdBody _ID_RD-RSP3Body)
    } )
_ALM_RD-RSP3Body ::= SEQUENCE {
    alm_rd_mode Unsigned16 {
        currentAlm (0),
        historyAlm (1),
        cAlmDetail (2),
        hAlmDetail (3)
    },
    alm_index Unsigned16 (0..11),
    alm_data OCTET STRING SIZE (8|24|40|56)
}

```

- Champ alm_rd_mode

alm_rd_mode

Ce champ doit contenir le mode de lecture d'alarme.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

- currentAlm ('00'H): lire l'état d'alarme/avertissement en cours;
- historyAlm ('01'H): lire l'historique d'alarme;
- cAlmDetail ('02'H): extraire les détails des informations d'alarme/avertissement individuelles en cours;
- hAlmDetail ('03'H): extraire les détails des informations individuelles de l'historique d'alarme;
- '04'H à 'FF'H: réservé.

- Champ `alm_index`

alm_index

Ce champ doit contenir l'index de point de lecture de l'alarme/avertissement.

La plage de valeurs doit s'étendre de 0 à 11.

- Champ `alm_data`

alm_data

Ce champ doit contenir l'état d'alarme lu correspondant au champ `alm_rd_mode`.

4.4.1.7 Instruction ALM_CLR et réponse

ALM_CLR symbolise une instruction "alarm-clear" (annulation d'alarme). A la réception de cette instruction, l'appareil subordonné doit annuler l'état d'alarme et d'avertissement interne, puis il doit répondre pour signaler l'exécution de l'instruction.

Avec cette instruction, on ne peut pas remédier à la cause de l'alarme ou de l'avertissement. Il convient de l'utiliser après avoir résolu les causes ou problèmes réels afin de rétablir l'état normal de l'appareil.

```

_ALM_CLR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (alm_clr),
      pduBody (cmdBody _ALM_CLR-CMD3Body)
    } )
_ALM_CLR-CMD3Body ::= SEQUENCE {
    alm_clr_mode Unsigned16 (
        cAlmClear (0),
        hAlmClear (1)
    ),
    réserve OCTET STRING SIZE (2|10|26|42|58)
}
_ALM_CLR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (alm_clr),
      pduBody (rspBody _ALM_CLR-RSP3Body)
    } )
_ALM_CLR-RSP3Body ::= SEQUENCE {
    alm_clr_mode Unsigned16 (
        cAlmClear (0),
        hAlmClear (1)
    ),
    réserve OCTET STRING SIZE (2|10|26|42|58)
}

```

- Champ `alm_clr_mode`

alm_clr_mode

Ce champ doit contenir le mode d'annulation d'alarme.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

- `cAlmClear ('00'H)`: annuler l'état d'alarme/avertissement en cours;
- `hAlmClear ('01'H)`: annuler l'historique d'alarme;
- `'02'H à 'FF'H`: réservé.

4.4.1.8 Instruction SYNC_SET et réponse

SYNC_SET symbolise une instruction "set into synchronous-state" (définir à l'état synchrone). A la réception de cette instruction, l'appareil subordonné doit basculer la PM d'appareil subordonné (FDCPM-S) de l'état S2: AsyncConnected à l'état S3: SyncConnected. A la réception de la réponse, l'appareil principal doit, de la même manière, basculer la PM d'appareil principal (FDCPM-M) de l'état S2: AsyncConnected à l'état S3: Etat SyncConnected, voir 8.4.4 pour plus de détails.

```
_SYNC_SET-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (sync_set)
    } )
```

```
_SYNC_SET-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (sync_set)
    } )
```

4.4.1.9 Instruction CONNECT et réponse

Dans le cadre de l'échange de cette instruction et de la réponse, l'appareil principal et l'appareil subordonné doivent établir une connexion FDC AR. A cet instant, certaines options ou certains paramètres de communication peuvent être spécifiés avec les PDU. Ils doivent assurer la transition de chaque PM (FDCPM-M et FDCPM-S). Voir 8.4.4 et 8.4.5 pour plus de détails.

```
_CONNECT-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (connect),
      pduBody (cmdBody _CONNECT-CMD3Body)
    } )
```

```
_CONNECT-CMD3Body ::= SEQUENCE {
    ver            Unsigned8,
    com_mod        SEQUENCE {
        réserve1    BIT STRING SIZE (1),
        syncmode    BIT STRING SIZE (1),
        dtmode      BIT STRING SIZE (2),
        réserve2    BIT STRING SIZE (3),
        subcmd      BIT STRING SIZE (1)
    },
    com_time       Unsigned8,
    profile_type   Unsigned8,
    réserve        OCTET STRING SIZE (0|8|24|40|56)
}
```

```
_CONNECT-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (connect),
      pduBody (rspBody _CONNECT-RSP3Body)
    } )
```

```

_CONNECT-RSP3Body ::= SEQUENCE {
    ver      Unsigned8,
    com_mod  SEQUENCE {
        réserve1  BIT STRING SIZE (1),
        syncmode  BIT STRING SIZE (1),
        dtmode    BIT STRING SIZE (2),
        réserve2  BIT STRING SIZE (3),
        subcmd    BIT STRING SIZE (1)
    },
    com_time  Unsigned8,
    profile_type Unsigned8,
    réserve   OCTET STRING SIZE (0|8|24|40|56)
}

```

- Champ com_mod

com_mod

Ce champ doit contenir plusieurs modes relatifs à la communication dans la connexion.

syncmode

Ce champ binaire doit indiquer le mode de communication.

- 0: état de communication asynchrone;
- 1: état de communication synchrone.

dtmode

Ce champ binaire doit indiquer le mode de transfert.

Ce champ doit contenir la plage de valeurs ci-dessous.

- '00'H: mode de transfert unique;
- '01'H: mode de transfert double;
- '02'H à '03'H: réservés.

subcmd

Ce champ binaire doit préciser l'utilisation ou pas de la sous-instruction.

- 0: Champ de sous-instruction désactivé;
- 1: Champ de sous-instruction activé.

- Champ com_time

com_time

Ce champ doit contenir la durée de cycle de communication sous la forme d'un multiple de la durée de cycle de transmission.

La plage de valeurs doit s'étendre de 0 à 255.

- Champ profile_type

profile_type

Ce champ doit contenir le code de profil d'appareil à utiliser. Comme pour le profil d'appareil, voir 4.5 et l'Annexe A.

La plage de valeurs doit s'étendre de 0 à 255.

4.4.1.10 Instruction DISCONNECT et réponse

Dans le cadre de l'échange de cette instruction et de la réponse, l'appareil principal et l'appareil subordonné doivent libérer une connexion FDC AR et les options de communication spécifiées doivent être réinitialisées. Ils doivent assurer la transition de chaque PM (FDCPM-M et FDCPM-S). Voir 8.4.4 et 8.4.5 pour plus de détails.

```
_DISCONNECT-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (disconnect)
    } )
_DISCONNECT-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (disconnect)
    } )
```

4.4.1.11 Instruction PPRM_RD et réponse

PRM_RD symbolise une instruction "PROM-parameter-read" (lecture de paramètre PROM). A la réception de cette instruction, l'appareil subordonné doit lire un paramètre spécifié sur E2PROM ou dans la mémoire non volatile dans le champ PPRM_RD-RSP3-PDU.parameter et doit y répondre

```
_PPRM_RD-CMD3-PDU ::= CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (pprm_rd),
      pduBody (cmdBody _PPRM_RD-CMD3Body)
    } )
_PPMM_RD_CMD3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    réserve  OCTET STRING SIZE (1|9|25|41|57)
}
_PPMM_RD-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (pprm_rd),
      pduBody (rspBody _PPRM_RD-RSP3Body)
    } )
_PPMM_RD-RSP3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    réserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}
```

- Champ pNo

pNo

Ce champ doit contenir le nombre de paramètres à lire.

La plage de valeurs doit s'étendre de 0 à 65 535.

- Champ pSize

pSize

Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit s'étendre de 0 à 255.

- Champ parameter

parameter

Ce champ doit contenir les données correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.4.1.12 Instruction PPRM_WR et réponse

PRM_WR symbolise une instruction "PROM-parameter-write" (écriture de paramètre PROM). A la réception de cette instruction, l'appareil subordonné doit écrire une valeur de paramètre spécifiée dans le champ PPRM_WR-CMD3-PDU.parameter dans la mémoire E2PROM ou la mémoire non volatile, puis il doit répondre en écho pour signaler l'exécution de l'instruction.

```

_PPRM_WR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (pprm_wr),
      pduBody (cmdBody _PPRM_WR-CMD3Body)
    } )
_PPRM_WR-CMD3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    réserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}
_PPRM_WR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (pprm_wr),
      pduBody (rspBody _PPRM_WR-RSP3Body)
    } )
_PPRM_WR-RSP3Body ::= SEQUENCE {
    pNo      Unsigned16,
    pSize    Unsigned8,
    réserve  OCTET STRING SIZE (1),
    parameter OCTET STRING SIZE (0|8|24|40|56)
}

```

- Champ pNo

pNo

Ce champ doit contenir le nombre de paramètres à écrire.

La plage de valeurs doit s'étendre de 0 à 65 535.

- Champ pSize

pSize

Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit s'étendre de 0 à 255.

- Champ parameter

parameter

Ce champ doit contenir les données à écrire correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.4.1.13 Instruction MEM_RD et réponse

MEM_RD symbolise une instruction "memory-read" (lecture de mémoire). A la réception de cette instruction, l'appareil subordonné doit lire un bloc de mémoire virtuelle spécifiée dans le champ MEM_RD-RSP3-PDU.data et doit y répondre.

Comme pour la mémoire virtuelle, voir Annexe A.

```

_MEM_RD-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (mem_rd),
      pduBody (cmdBody _MEM_RD-CMD3Body)
    } )
_MEM_RD-CMD3Body ::= SEQUENCE {
    réserve OCTET STRING,
    mMode SEQUENCE {
        data_type BIT STRING SIZE (4),
        mode BIT STRING SIZE (4)
    },
    mSize Unsigned16,
    mAddress Unsigned32,
    réserve OCTET STRING SIZE (4|20|36|52)
}
_MEM_RD-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (mem_rd),
      pduBody (rspBody _MEM_RD-RSP3Body)
    } )
_MEM_RD-RSP3Body ::= SEQUENCE {
    réserve OCTET STRING,
    mMode SEQUENCE {
        data_type BIT STRING SIZE (4),
        mode BIT STRING SIZE (4)
    },
    mSize Unsigned16,
    mAddress Unsigned32,
    data OCTET STRING SIZE (4|20|36|52)
}

```

- Champ mMode

mMode

Ce champ doit contenir le mode de lecture de la mémoire.

data_type

Ce champ binaire contient un type de mémoire faisant office de source pour la lecture.

- '00'H: réservé;
- '01'H: Mémoire volatile;
- '02'H: Mémoire non volatile;
- '03'H à 'FF'H: réservé.

mode

Ce champ binaire contient le type des données spécifiées dans la mémoire à lire.

- '00'H: réservé;
- '01'H: Integer8;
- '02'H: Integer16;
- '03'H: Integer32;
- '04'H: Integer64;
- '05'H à 'FF'H: réservé.

- Champ mSize

mSize

Ce champ doit contenir la quantité de données à lire dans la mémoire.

La plage de valeurs doit s'étendre de 0 à 20.

- Champ mAddress

mAddress

Ce champ doit contenir l'adresse de début de la mémoire à lire.

La plage de valeurs doit être comprise entre 0 et 'FFFF FFFF'H.

- Champ data

data

Ce champ doit contenir les données lues avec les champs mMode, mSize et mAddress spécifiés.

4.4.1.14 Instruction MEM_WR et réponse

MEM_WR symbolise une instruction "memory-write" (écriture de mémoire). A la réception de cette instruction, l'appareil subordonné doit écrire un bloc de valeurs spécifiées de mémoire virtuelle du champ MEM_WR-CMD3-PDU.data vers la mémoire, et il doit répondre en écho pour signaler l'exécution de l'instruction.

Comme pour la mémoire virtuelle, voir Annexe A.

```

MEM_WR-CMD3-PDU ::= _CMD3-PDU ( WITH COMPONENTS
    { ..., cmd (mem_wr),
      pduBody (cmdBody _MEM_RD-CMD3Body)
    } )
MEM_WR-CMD3Body ::= SEQUENCE {
    réserve    OCTET STRING,
    mMode      SEQUENCE {
        data_type  BIT STRING SIZE (4),
        mode       BIT STRING SIZE (4)
    },
    mSize      Unsigned16,
    mAddress    Unsigned32,
    data       OCTET STRING SIZE (4|20|36|52)
}

```



```

_MEM_WR-RSP3-PDU ::= _RSP3-PDU ( WITH COMPONENTS
    { ..., rcmd (mem_wr),
      pduBody (rspBody _MEM_RD-RSP3Body)
    } )
_MEM_WR-RSP3Body ::= SEQUENCE {
    réserve OCTET STRING,
    mMode SEQUENCE {
        data_type BIT STRING SIZE (4),
        mode BIT STRING SIZE (4)
    },
    mSize Unsigned16,
    mAddress Unsigned32,
    data OCTET STRING SIZE (4|20|36|52)
}

```

- Champ mMode

mMode

Ce champ doit contenir le mode d'écriture de la mémoire.

data_type

Ce champ binaire doit contenir un type de mémoire de destination dans laquelle écrire.

- '00'H: réservé;
- '01'H: Mémoire volatile;
- '02'H: Mémoire non volatile;
- '03'H à 'FF'H: réservé.

mode

Ce champ binaire doit contenir le type des données spécifiées dans la mémoire dans laquelle écrire.

- '00'H: réservé;
- '01'H: Integer8;
- '02'H: Integer16;
- '03'H: Integer32;
- '04'H: Integer64;
- '05'H à 'FF'H: réservé.

- Champ mSize

mSize

Ce champ doit contenir la quantité de données de la mémoire dans laquelle écrire.

La plage de valeurs doit s'étendre de 0 à 20.

- Champ mAddress

mAddress

Ce champ doit contenir l'adresse de début de la mémoire dans laquelle écrire.

La plage de valeurs doit être comprise entre 0 et 'FFFF FFFF'H.

- Champ data

data

Ce champ doit contenir les données à écrire avec les champs *mMode*, *mSize* et *mAddress* spécifiés.

4.5 Profil d'appareil

Un appareil de terrain peut fournir certains ensembles de PDU dépendantes de l'appareil de terrain et leurs *CMDCodes* pour le service FDC, appelés profils d'appareil de terrain. L'utilisateur d'un appareil principal FDC peut décider lequel il convient de choisir pour la communication FDC lorsque la connexion du protocole FDC est établie.

Le système de profil d'appareil et les informations connexes sont présentés à l'Annexe A.

5 Syntaxe de transfert**5.1 Concepts**

La FAL de type 27 est une technologie d'application à temps critique. Par conséquent, il est important de faire en sorte que la taille des données codées soit compacte et de pouvoir les coder et les décoder simplement.

Les processus de codage et de décodage doivent donc être définis de manière aussi spécifique à la FAL de type 27, cela n'exigeant aucun caractère commun avec l'autre type de protocole ni aucune versatilité. Le format de données et la règle de codage de l'APDU doivent être préalablement définis pour les entités qui transfèrent les données, cela n'exigeant aucun service de couche de présentation.

Dans la règle de codage de la FAL de type 27, on peut ne pas coder la balise (code de type de données) et la longueur de données. Les seules valeurs de chaque type de données défini avec la syntaxe abstraite ASN.1 doivent être codées dans une ligne de données.

La longueur de la PDU doit être fixe dans le cas d'un service FDC dont le facteur temps est particulièrement important. Pour un service MSG dont le facteur temps n'est pas strict, la PDU de longueur variable peut être transférée. Dans ce cas, il convient que la syntaxe abstraite définisse explicitement un champ de données précisant la longueur de données.

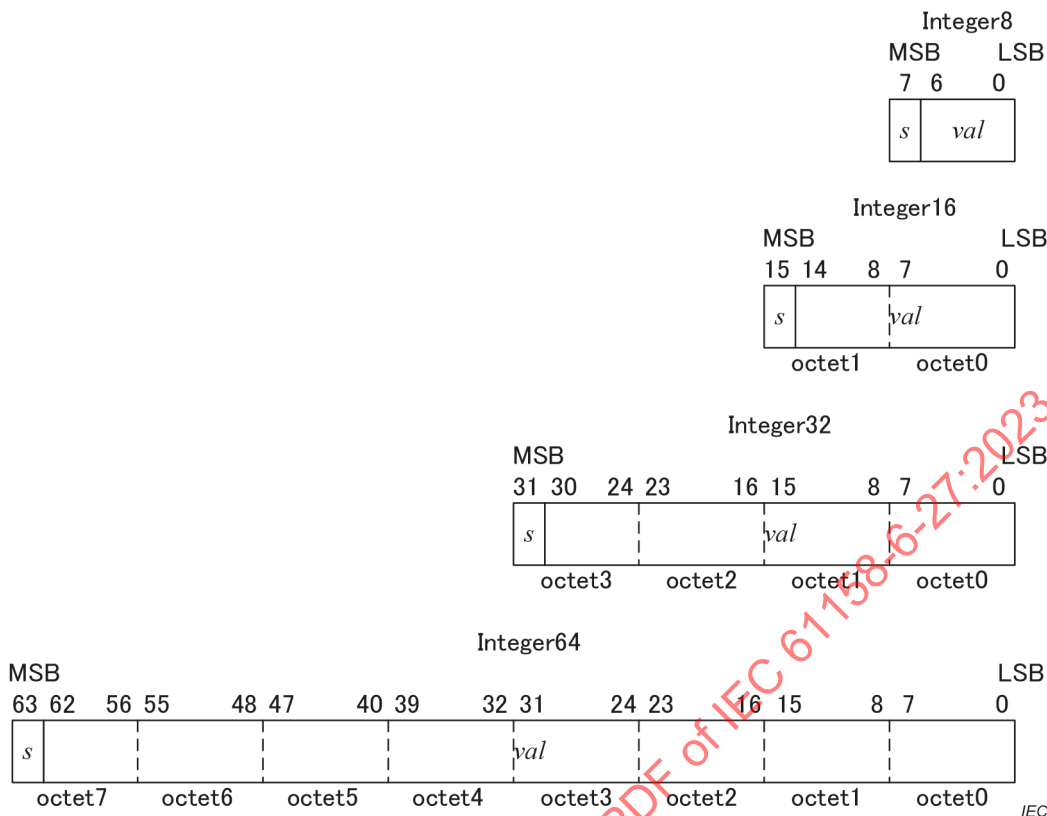
Il convient que la couche inférieure définisse l'ordre des bits sur la voie de transmission.

5.2 Règles de codage**5.2.1 INTEGER et ses sous-types**

INTEGER et ses dérivés (*Integer8*, *Integer16*, *Integer32*, *Integer64*, *Unsigned8*, *Unsigned16*, *Unsigned32* et *Unsigned64*) doivent coder et transférer uniquement la valeur de données dans leur taille d'octet.

La forme codée d'*Integer8*, *Integer16*, *Integer32* et *Integer64* doit être composée d'un bit: *s* de signe sur le bit de poids fort (voir la Figure 1). Le *val* doit contenir la valeur elle-même s'il est supérieur ou égal à 0 (le bit de signe est '0'B). Si *val* est un nombre négatif (le bit de signe est '1'B), les données codées doivent être représentées comme un complément de 2 en utilisant *s* et *val*.

NOTE Cette règle de codage s'applique à la structure de données lors du transfert de la PDU, mais on peut ne pas préciser la structure de données sur une mémoire réelle du système de traitement des données d'un appareil réel.



Légende

s bit de signe (0: + / 1: -)

val valeur positive, si *s*=0;

"*s+val*" représente une valeur négative en complément de 2, si *s*=1.

Figure 1 – Codage des sous-types Integer

Lors du transfert des données d'un type de données à plusieurs octets, les octets doivent être transmis dans l'ordre, du plus bas vers le plus élevé (voir l'exemple à la Figure 2).

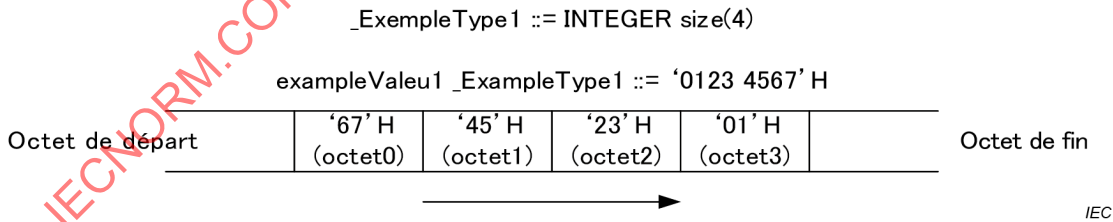


Figure 2 – Exemple de transfert de la valeur INTEGER

Au format codé Unsigned8, Unsigned16, Unsigned32 et Unsigned64, les données de valeur numérique (*val*) doivent être définies dans la zone de données de sa taille d'octet (voir la Figure 3). Dans ce cas, la valeur numérique doit être définie de manière à placer le numéro d'un chiffre inférieur dans un bit inférieur de la séquence, la valeur 0 devant être définie sur le bit supérieur non utilisé.

Lors du transfert des données d'un type de données à plusieurs octets, les octets doivent être transmis dans l'ordre, du plus bas vers le plus élevé.

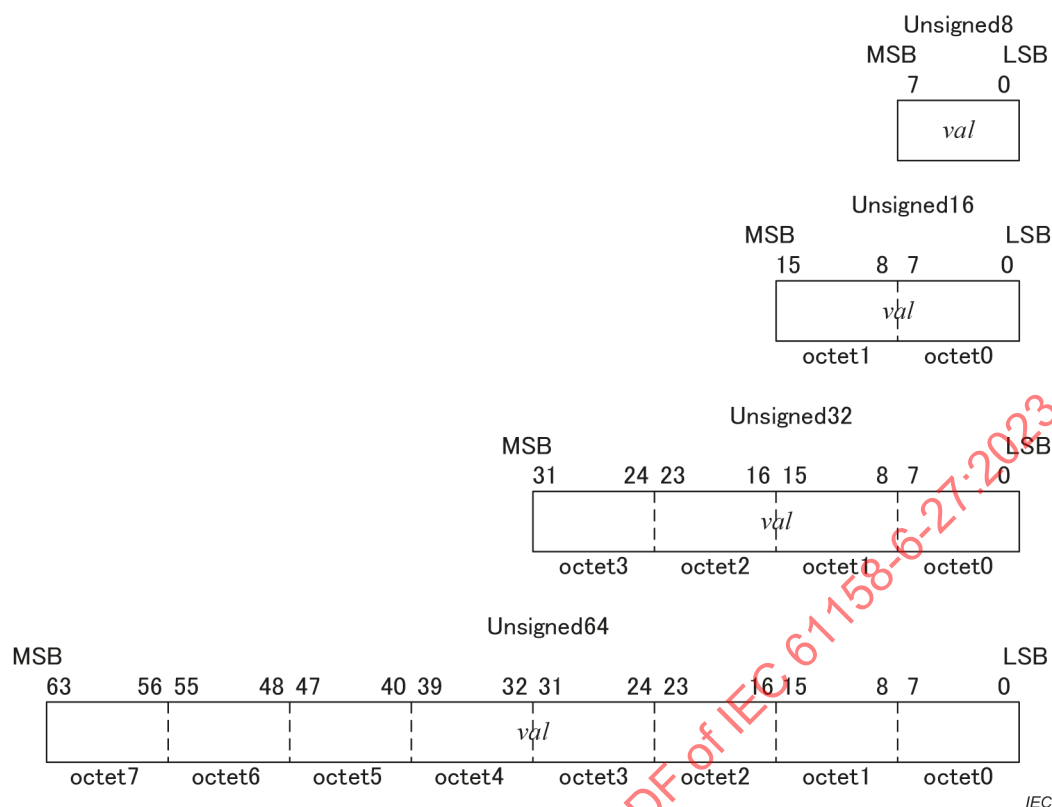


Figure 3 – Codage des sous-types Unsigned

5.2.2 Type REAL et ses sous-types

La FAL de type 27 ne définit pas directement les données du type REAL dans la syntaxe abstraite et utilise $Float_{32}$ et $Float_{64}$, qui sont dérivées du type REAL. Ce codage doit satisfaire à l'IEC 60559. Leurs formats codés sont représentés à la Figure 4 et à la Figure 5.

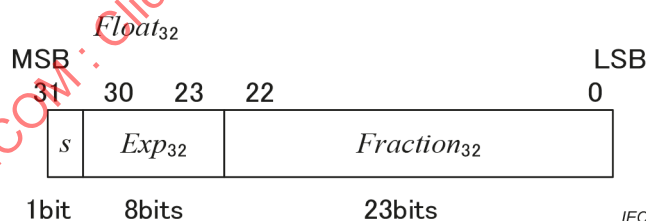


Figure 4 – Codage du type Float32

Les données du type $Float_{32}$ doivent être composées de 4 octets, comme illustré à la Figure 4. Les octets doivent être transmis dans l'ordre, du plus bas au plus élevé. Dans ce cas, la valeur des données à virgule flottante du type $Float_{32}$ doit être calculée à l'aide de la formule suivante:

$$Float_{32} = (-1)^s \times c_{32} \times 2^{q_{32}},$$

où

s : bit de signe (0: + / 1: -),

$$c_{32} = (1 \times 2^{23} + Fraction_{32}) \times 2^{-23} = 1 + \frac{b_{22}}{2} + \frac{b_{21}}{2^2} + \dots + \frac{b_0}{2^{23}}: \text{ mantisse,}$$

$$q_{32} = Exp_{32} - 127: \text{ exposant } (-127..127).$$

Si $Fraction_{32}$ et Exp_{32} sont tous deux égaux à 0, alors $Float_{32}$ représente 0.

Si $Fraction_{32}$ est égal à 0 et Exp_{32} est égal à 255, alors $Float_{32}$ représente l'infinité positive ou négative en tant que bit de signe s .

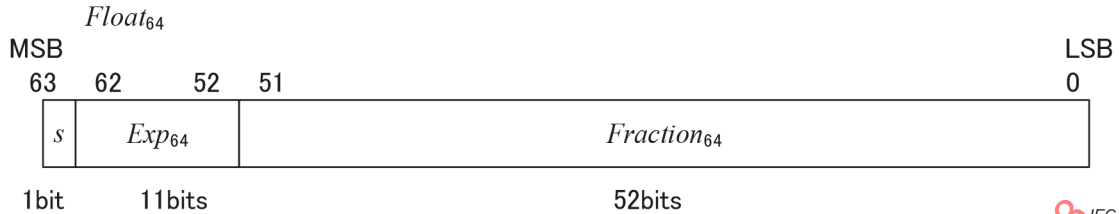


Figure 5 – Codage du type Float64

Les données du type $Float_{64}$ doivent être composées de 8 octets, comme illustré à la Figure 5. Les octets doivent être transmis dans l'ordre, du plus bas au plus élevé. Dans ce cas, la valeur des données à virgule flottante du type $Float_{64}$ doit être calculée à l'aide de la formule suivante:

$$Float_{64} = (-1)^s \times c_{64} \times 2^{q_{64}},$$

où

s : bit de signe (0: + / 1: -),

$$c_{64} = (1 \times 2^{52} + Fraction_{64}) \times 2^{-52}$$

$$= 1 + \frac{b_{51}}{2} + \frac{b_{50}}{2^2} + \dots + \frac{b_0}{2^{52}}: \text{ mantisse,}$$

$$q_{64} = Exp_{64} - 1023: \text{ exposant } (-1023, 1023).$$

Si $Fraction_{64}$ et Exp_{64} sont tous deux égaux à 0, alors $Float_{64}$ représente 0.

Si $Fraction_{64}$ est égal à 0 et Exp_{64} est égal à 2047, alors $Float_{64}$ représente l'infinité positive ou négative en tant que bit de signe s .

5.2.3 Type BIT STRING

Dans le type BIT STRING, le bit de début (bit 0) doit correspondre au bit de poids faible des données codées, le bit de fin devant correspondre au bit de poids fort en fonction des nombres associés aux bits nommés. Toutefois, les données de remplissage doivent être remplies par une frontière d'octet. Dans ce cas, une zone peut être réservée même pour le bit non défini. Toutefois, la valeur du bit n'est pas définie. La Figure 6 représente un exemple de définition de champ binaire du type BIT STRING.

Pour un bloc de données BIT STRING de taille de données à plusieurs octets, les octets doivent être transmis dans l'ordre, à partir de l'octet contenant le bit le plus bas.

NOTE La définition de "bit de début" et "bit de fin" est conforme à l'ISO/IEC 8824-1:2021, 22.2.

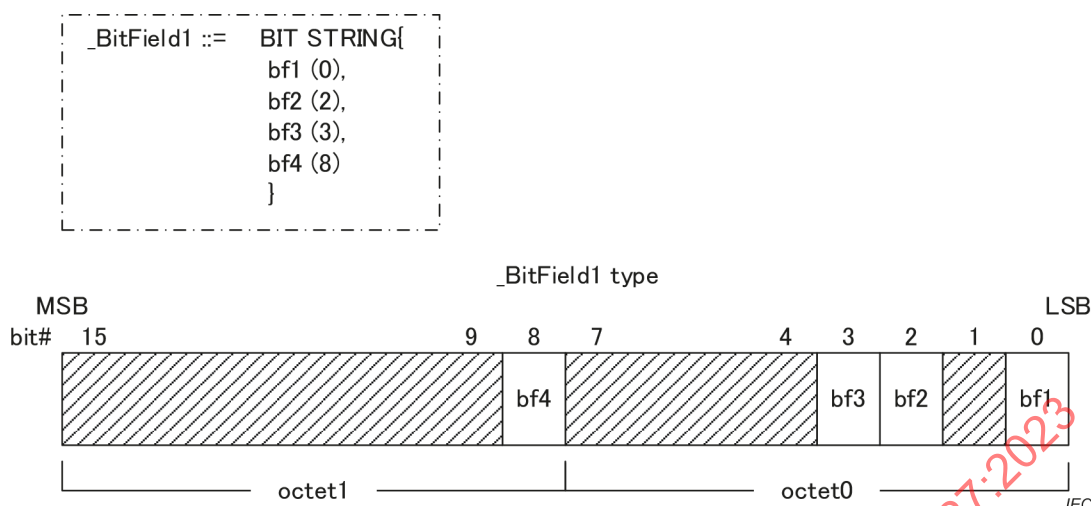


Figure 6 – Exemple de définition de champ binaire avec bits nommés

Un champ binaire est défini en combinant les types SEQUENCE et BIT STRING. Dans ce cas, la frontière de chaque champ doit être définie en fonction de la taille de bit spécifiée et remplie à partir du bit le plus faible (voir l'exemple présenté à la Figure 7).

Pour coder ce champ binaire, un octet doit être rempli à partir de son bit le plus faible, en fonction de l'ordre prévu par la définition de champ décrite entre "{" et "}" de la syntaxe SEQUENCE afin de définir une zone de données confinée par la frontière d'octet. S'il reste des bits de fraction non définis, il doit s'agir d'une zone réservée et leurs valeurs doivent être indéfinies.

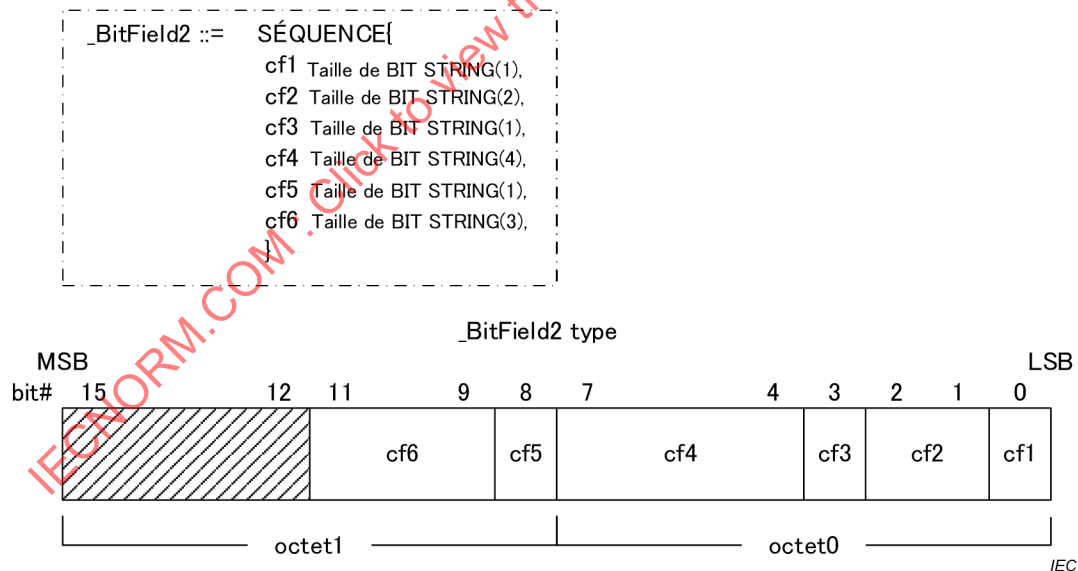


Figure 7 – Exemple de définition de champ binaire avec taille de champ

5.2.4 Type OCTET STRING et type IA5String

Les données de type OCTET STRING doivent être transférées sans convertir le code dans l'ordre à partir de l'octet de poids fort des données de chaîne données.

Les données de type IA5String doivent être codées dans des lignes de données (OCTET STRING) en convertissant les données de chaîne en un code à 1 octet, conformément à l'ISO/IEC 646, caractère par caractère, puis en ajoutant un octet de code nul ('00'H) au dernier code de caractère. Les données doivent être transférées dans l'ordre à partir des données de code du premier caractère.

5.2.5 Type NULL

Pour les données de type NULL, la longueur de données est 0 et aucune valeur n'existe. Par conséquent, elles ne doivent pas être codées ni transférées.

5.2.6 Type Structure et type Array

Parmi les éléments du type Structure, le type SEQUENCE et le type SEQUENCE OF ne doivent pas être codés. Les composants inclus dans ces types doivent être codés et transférés conformément à chaque règle et à l'ordre de transfert d'octet (voir la Figure 8).

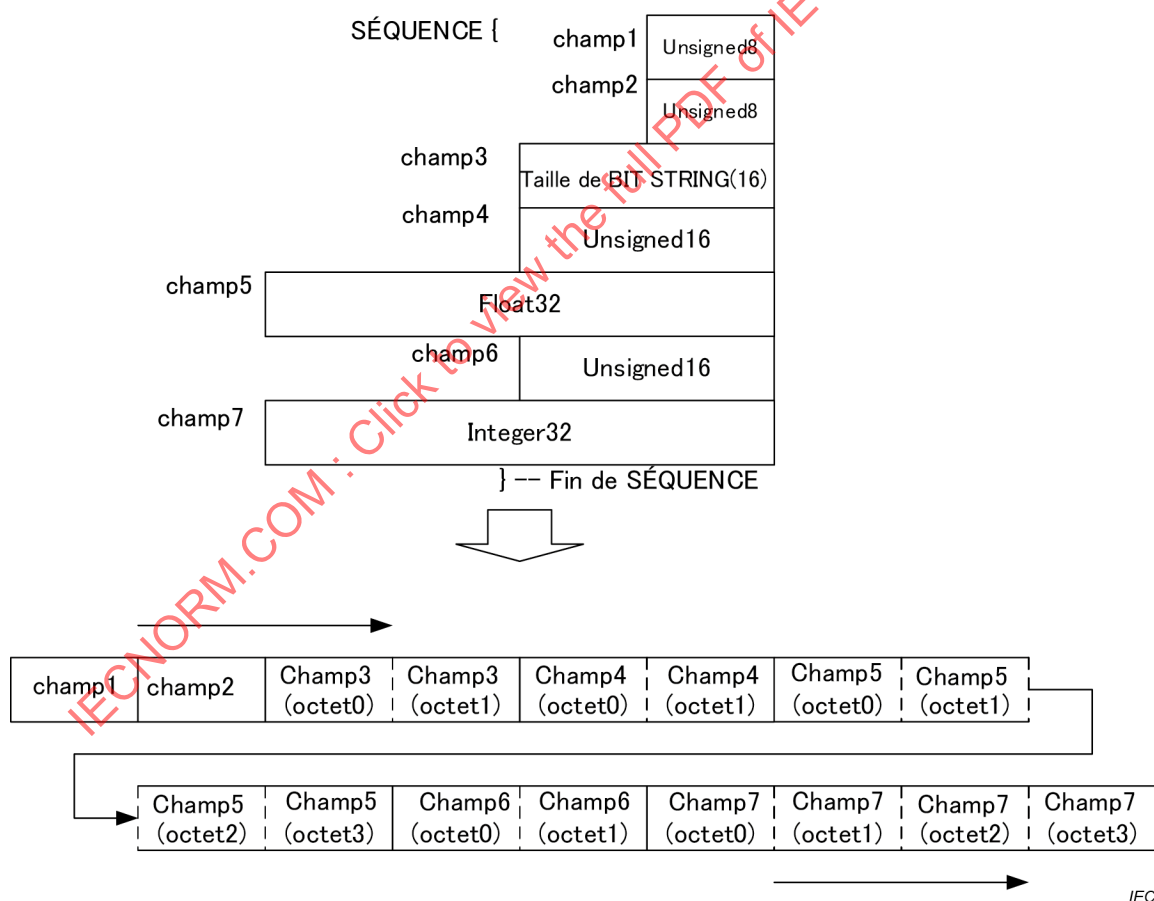


Figure 8 – Codage du type SEQUENCE

De même, le type CHOICE lui-même ne doit pas être codé. Pour modifier les données de communication dans l'une des alternatives de ce type, il convient de sélectionner les données en fonction du contexte de l'AP de communication et il convient d'établir un accord entre les appareils de communication. La méthode pour ce processus dépend de la mise en œuvre et n'entre pas dans le domaine d'application du présent document. Les données doivent être codées en fonction de la règle du composant sélectionné.

6 Structure du diagramme d'états de protocole FAL

Dans le présent article et ceux qui suivent, les ASE FAL sont caractérisés avec des modèles de diagramme d'états de protocole (PM).

Bien que ce type de bus de terrain dispose de sa structure de PM, une table de correspondance est présentée ci-dessous pour la clarté de la compréhension (voir Tableau 10). Elle met en correspondance la structure de ce type avec la structure à quatre couches classique adoptée par la plupart des autres types, contenant la PM de contexte AP, la FSPM, l'ARPM et la DMPM.

La Figure 9 représente la structure des PM.

- Il existe un diagramme d'états de contexte AP défini de manière formelle (APC SM) permettant à l'utilisateur FAL de lancer une application de communication particulière.
- Il n'existe aucune définition de FSPM faisant simplement office d'interface entre l'utilisateur FAL et l'ARPM. Seuls deux types de machines de protocole sont définis:
 - un ensemble de machines de commande d'appareil de terrain (FDC PM) faisant office d'appareil principal, d'appareil subordonné et de moniteur pour les utilisateurs du service FDC;
 - une paire de machines de message (MSG PM) faisant office de demandeur et de répondeur pour les utilisateurs de service de message.
- Deux types différents de machine ARPM sont définis au niveau de l'interface FSMUL:
 - un ensemble de machines ARPM pour les relations d'applications orientées connexion entre les classes d'appareil principal, d'appareil subordonné ou de moniteur de service FDC;
 - un type de machine ARPM pour les relations entre applications sans connexion entre les classes de demandeur et de répondeur du service de messages.
- Il n'existe aucune définition formelle de la machine de protocole de mise en correspondance DLL (DMPM), unifiée dans l'ARPM ou l'APC SM. Les services DL sont utilisés directement par l'ARPM ou l'APC SM.

La FDCPM est censée fonctionner dans l'ASE FDC. De même, la MSGPM est censée fonctionner dans l'ASE MSG, l'ARPM dans l'ASE AR et l'AP-CONTEXT SM dans l'ASE FSM.

Tableau 10 – Mise en correspondance des diagrammes d'états de protocole

Type d'AP	ASE	Classe activée	Machine de protocole	Correspondance avec une structure classique pour la plupart des autres types
AP d'appareil principal C1	ASE FSM	FieldbusSystemManager	APC SM	AP-Context-PM
	ASE FDC	Appareil principal	FDC PM-M	FSPM
	ASE MSG	Demandeur	MSG PM-RQ	FSPM
		Répondeur	MSG PM-RS	
	ASE AR	FDCMaster-AR	ARPM-FDCM	ARPM avec DMPM
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	s.o.	s.o.
AP d'appareil subordonné	ASE FSM	FieldbusSystemManager	APC SM	AP-Context-PM
	ASE FDC	Appareil subordonné	FDC PM-S	FSPM
		Monitor (option)	FDC PM-MN	
	ASE MSG	Répondeur	MSG PM-RS	FSPM
	ASE AR	FDCSlave-AR	ARPM-FDCS	ARPM avec DMPM
		FDCMonitor-AR	ARPM-FDCMN	
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	s.o.	s.o.

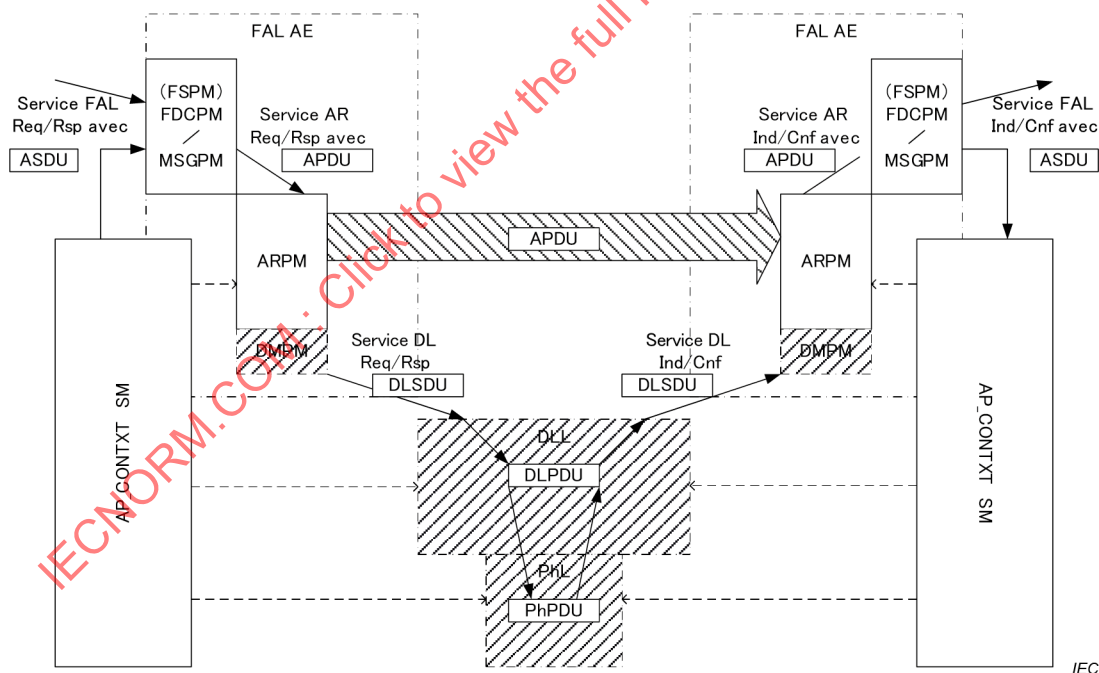


Figure 9 – Structure des diagrammes d'états de protocole FAL

7 Diagramme d'états de contexte AP (APC SM)

7.1 Vue d'ensemble

Un contexte d'AP est un ensemble d'informations et de règles relatives à un système de communication. Il doit être créé par des instructions provenant d'un utilisateur FAL à l'appel du processus d'application (AP). Par exemple, il contient les paramètres de communication sélectionnés, en fonction de la configuration du système, un domaine d'application du système, le profil de l'appareil et les différences d'aptitudes de communication entre l'appareil et ses homologues.

Dans ce modèle ASE, il convient qu'une FAL conserve les informations mentionnées ci-dessus dans les attributs de la classe FieldbusSystemManager, conformément aux instructions et configurations de l'utilisateur. Il convient que les informations puissent permettre de caractériser les types d'AP, comme l'appareil principal C1 et l'appareil subordonné. De plus, il convient de pouvoir configurer d'autres paramètres de communication et le profil d'appareil (une servocommande, un onduleur ou un appareil d'E/S, par exemple) afin d'établir l'environnement de communication.

L'APC SM doit contrôler une série de transitions d'état dans lesquelles un appareil démarre. L'APC SM doit obtenir les données de contexte AP auprès de l'utilisateur FAL et doit initialiser la FAL et la couche inférieure avec le contexte. En conséquence, elle doit établir une communication constante. L'APC SM réalise le service que l'ASE FSM offre.

L'APC SM peut ne pas modifier directement une APDU ni ne réalise un protocole pour transférer des PDU. Toutefois, elle agit indirectement comme un déclencheur d'échange de PDU en utilisant l'autre service ASE et le service de couche inférieure lors du processus d'initialisation de la FAL.

La Figure 10 représente le diagramme d'états APC SM.

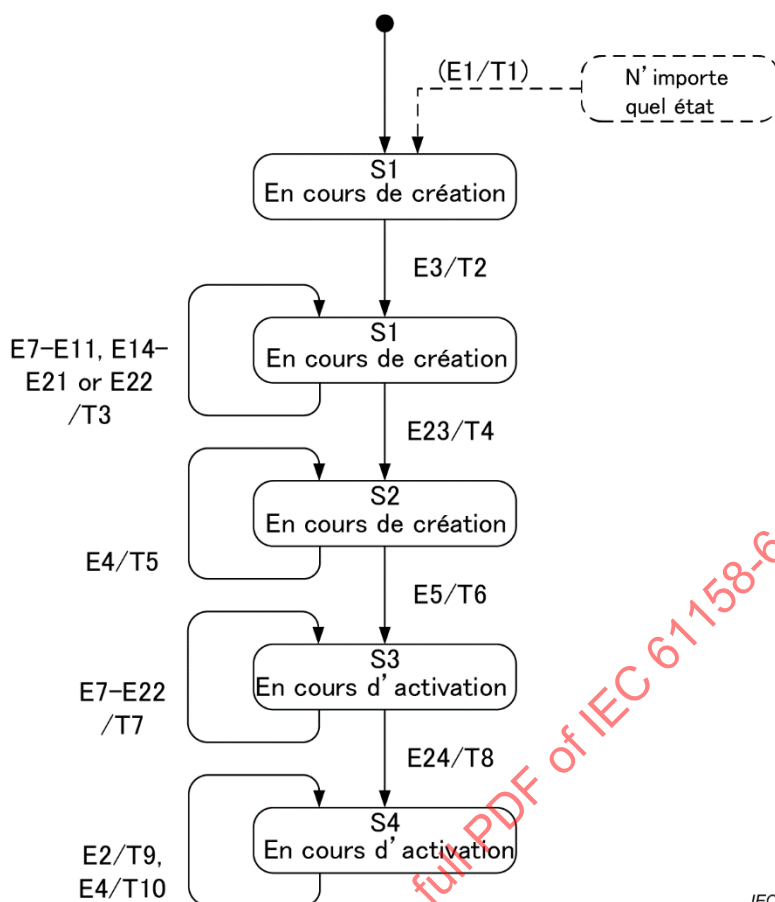


Figure 10 – Diagramme d'états de l'APC SM

7.2 Descriptions des états

Le Tableau 11 décrit chaque état de l'APC SM.

Tableau 11 – Descriptions des états de l'APC SM

N° E	État	Sous-état	Description
S0	Annulé	-	État qui est celui de la classe FieldbusSystemManager de l'ASE FSM qui vient d'être instancié comme ce diagramme d'états Toutes les entités de chaque couche et les ASE de l'appareil sont réinitialisés, et le contexte AP est annulé.
S1	En cours de création		État transitoire dans lequel le contexte AP ou l'environnement de communication est généré, selon l'entrée CONTEXT-DATA et le résultat de la négociation entre les appareils correspondants
S2	Créé	-	État dans lequel le contexte AP a été généré
S3	En cours d'activation	-	État transitoire dans lequel les entités de chaque couche et les ASE sont activés de manière séquentielle
S4	Active	-	État permanent dans lequel toutes les entités sont activées et les services normaux de communication fournis

7.3 Événements déclencheurs

Le Tableau 12 répertorie chaque événement déclencheur de l'APC SM.

Tableau 12 – Événements déclencheurs de l'APC SM

N° Év	Événement déclencheur: primitive ou condition	Source de l'événement	Paramètres associés	Description
E1	FSM-Reset.req	Utilisateur FAL		
E2	FSM-GetStatus.req	Utilisateur FAL	INFO-ID	
E3	FSM-SetContext.req	Utilisateur FAL	CONTEXT-DATA	
E4	FSM-GetContext.req	Utilisateur FAL		
E5	FSM-Start.req	Utilisateur FAL		
E6	<s>-Open.cnf	<s> ASE	ServiceStatus	
E7	FSMUL-RESET.cnf	FSMUL		
E8	FSMUL-SET-VALUE.cnf	FSMUL	Résultat	
E9	FSMUL-GET-VALUE.cnf	FSMUL	Result, Val	
E10	FSMUL-DETECT.cnf	FSMUL		
E11	FSMUL-RECOG.cnf	FSMUL		
E12	FSMUL-SW-TRANS-MODE.cnf	FSMUL		
E13	FSMUL-MEAS-DELAY.ind	FSMUL	Delay_Time	
E14	FSMUL-MEAS-DELAY.cnf	FSMUL	Delay_Time	
E15	FSMUL-CONFIG.cnf			
E16	FSMUL-START-SYNC.ind			
E17	FSMUL-START-SYNC.cnf			
E18	FSMUL-START.ind	FSMUL	Com_Mode, Cycle_time, Max_Delay, TM_unit	
E19	FSMUL-START.cnf	FSMUL	Résultat	
E20	FSMUL-CLR-ERR.cnf	FSMUL	Résultat	
E21	Ph-SET-VALUE.cnf	PhL		
E22	Ph-GET-VALUE.cnf	PhL	Valeur	
E23	APC-Created	APC SM (état S1)		Événement interne
E24	APC-Activated	APC SM (état S3)		Événement interne

7.4 Descriptions des actions aux transitions d'état

Des spécifications particulières telles que le traitement d'initialisation, le traitement d'activation et divers paramètres de communication dans chaque couche et chaque ASE ne relèvent pas du domaine d'application du présent document, car elles dépendent de la mise en œuvre. L'action connexe est simplement présentée dans le Tableau 13.

Tableau 13 – Transitions de l'APC SM

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
T1	(n'importe quel état)	E1:FSM-Reset.req / Ph-RESET.req; FSMUL_RESET.req; EVM-Reset.req; pour tous les objets ASE AR (AR-Reset.req); pour tous les objets ASE FDC (FDC_Reset.req); pour tous les objets ASE MSG (MSG-Reset.req);	S0: Annulé
T2	S0: Annulé	E3: FSM-SetContext.req(CONTEXT-DATA) / /*-- L'APC SM commence à initialiser PhL -- avec Ph-SET-VALUE.req --*/; /*-- L'APC SM commence à initialiser FSMUL -- avec FSMUL_SET_PAR.req -- et FSMUL_DELAY.req --*/; pour tous les objets ASE AR { AR-Open.req;}; pour tous les objets ASE FDC { FDC-Open.req;}; pour tous les objets ASE MSG { MSG-Open.req;};	S1: En cours de création
T3	S1: En cours de création	E7, E8, E9, E10, E11, E12, E13, E14, E15, E18, E19, E20, E21 ou E22 / /*-- L'APC SM continue d'initialiser PhL, FSMUL et FAL --*/; /*-- Si les procédures d'initialisation sont terminées, -- E28, -- E17:APC-Created est émis. --*/;	S1: En cours de création
T4	S1: En cours de création	E23:APC-Created / FSM-SetContext.cnf;	S2: Créé
T5	S2: Créé	E4:FSM-GetContext.req / FSM-GetContext.cnf;	S2: Créé
T6	S2: Créé	E5:FSM-Start.req / /*-- début d'activation de PhL --*/; /*-- début d'activation de FSMUL --*/; EVM-Enable.req; pour tous les objets ASE AR { AR-Enable.req;}; pour tous les objets ASE FDC { FDC-Enable.req;}; pour tous les objets ASE MSG { MSG-Enable.req;};	S3: En cours d'activation
T7	S3: En cours d'activation	E7, E8, E9, E10, E11, E12, E13, E14, E15, E16, E17, E18, E19, E20, E21 ou E22 / /*-- L'APC SM continue d'activer PhL, FSMUL, FAL --*/; /*-- Si les procédures d'activation sont terminées, -- E29:APC-Activated est émis --*/;	S3: En cours d'activation

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
T8	S3: En cours d'activation	E24: APC-Activated / FSM-Start.cnf;	S4:Active
T9	S4:Active	E2: FSM-GetStatus.req(INFO-ID) / /*-- L'APC SM lit les informations d'état appropriées pour INFO-ID, -- utilisation de Ph-GET_VALUE.req, -- CTC-GET-STATUS.req, -- DLM_GET_ERR.req -- ou autres primitives de service --*/; FSM-GetStatus.cnf;	S4:Active
T10	S4:Active	FSM-GetContext.req / FSM-GetContext.cnf;	S4:Active

8 Machine de protocole de service FAL (FSPM)

8.1 Vue d'ensemble

Si la FSPM reçoit une primitive de demande de service ou une primitive de réponse provenant d'un utilisateur FAL, elle doit modifier une APDU provenant d'une SDU comme le paramètre des primitives, puis demander à l'ARPM de la transmettre. La FSPM doit également retirer une SDU de l'APDU reçue par la l'ARPM, puis la livrer à l'utilisateur de la FAL en tant que primitive d'indication ou en conformité.

Deux types de services d'application (ASE FDC et ASE MSG) peuvent être proposés à l'utilisateur FAL dans la FAL de type 27 (voir l'IEC 61158-5-27), et la FSPM se compose de deux types de PM (FDC PM et MSG PM) correspondant aux ASE.

8.2 Machine de protocole RT (RT PM)

8.2.1 Découverte de la couche de liaison

Les concepts conformes à la norme IEEE 802.1AB doivent être appliqués. La présente partie de la spécification définit l'utilisation de la norme IEEE 802.1AB.

8.2.2 Ponts MAC

8.2.2.1 Vue d'ensemble

Les concepts conformes à la norme IEEE 802.1D doivent être appliqués. La présente partie de la spécification définit l'utilisation de la norme IEEE 802.1D. Elle inclut les extensions pour la phase RT concernant le transfert des trames. Les mécanismes d'acheminement de la norme IEEE 802.1D doivent être temporairement désactivés dans la phase RT. Dans ce cas, l'acheminement des trames RT doit être effectué conformément aux attributs définis par l'ASE approprié. Les valeurs des attributs définissent une table d'acheminement spéciale, qui est utilisée.

En dehors du comportement de la norme IEEE 802.1D, la présente spécification de protocole définit les machines de protocole suivantes pour fournir des actions spéciales de transfert:

- Machine de protocole de transfert de RT (RT_FW);
- Machine de protocole de transfert de protocole de temps de précision (PTP_FW).

8.2.2.2 Machine de protocole de transfert de RT (RT_FW)

8.2.2.2.1 Généralités

RT_FW est constitué de la CTC (commande de transmission de communication), de la SRC (commande d'envoi réception) et de la FSMUL (couche sous-jacente de gestion système de terrain) en tant que trois couches sous-jacentes. La SRC est située dans la partie inférieure de la CTC, tandis que la FSMUL est située à travers la CTC et la SRC.

La CTC crée la FIDPDU et il s'agit de la sous-couche pour exécuter la machine de protocole de synchronisation et la machine de protocole de communication déclenchée par les événements. Ces machines de protocole sont opérées conformément à l'instruction de la FSM ou de la FSMUL.

La SRC est une sous-couche destinée à envoyer ou recevoir une trame conformément à l'instruction de la CTC. La FSMUL est une sous-couche destinée à définir les variantes internes à utiliser pour RT_FW et les erreurs de commande détectées dans chaque sous-couche.

8.2.2.2.2 Couche sous-jacente de commande de transmission de communication

8.2.2.2.2.1 Machine de protocole CTC de l'appareil principal

Le schéma de transition d'états CTC de l'appareil principal est tel que représenté à la Figure 11 et la table de transition d'états CTC de l'appareil principal est telle que représentée dans le Tableau 14.

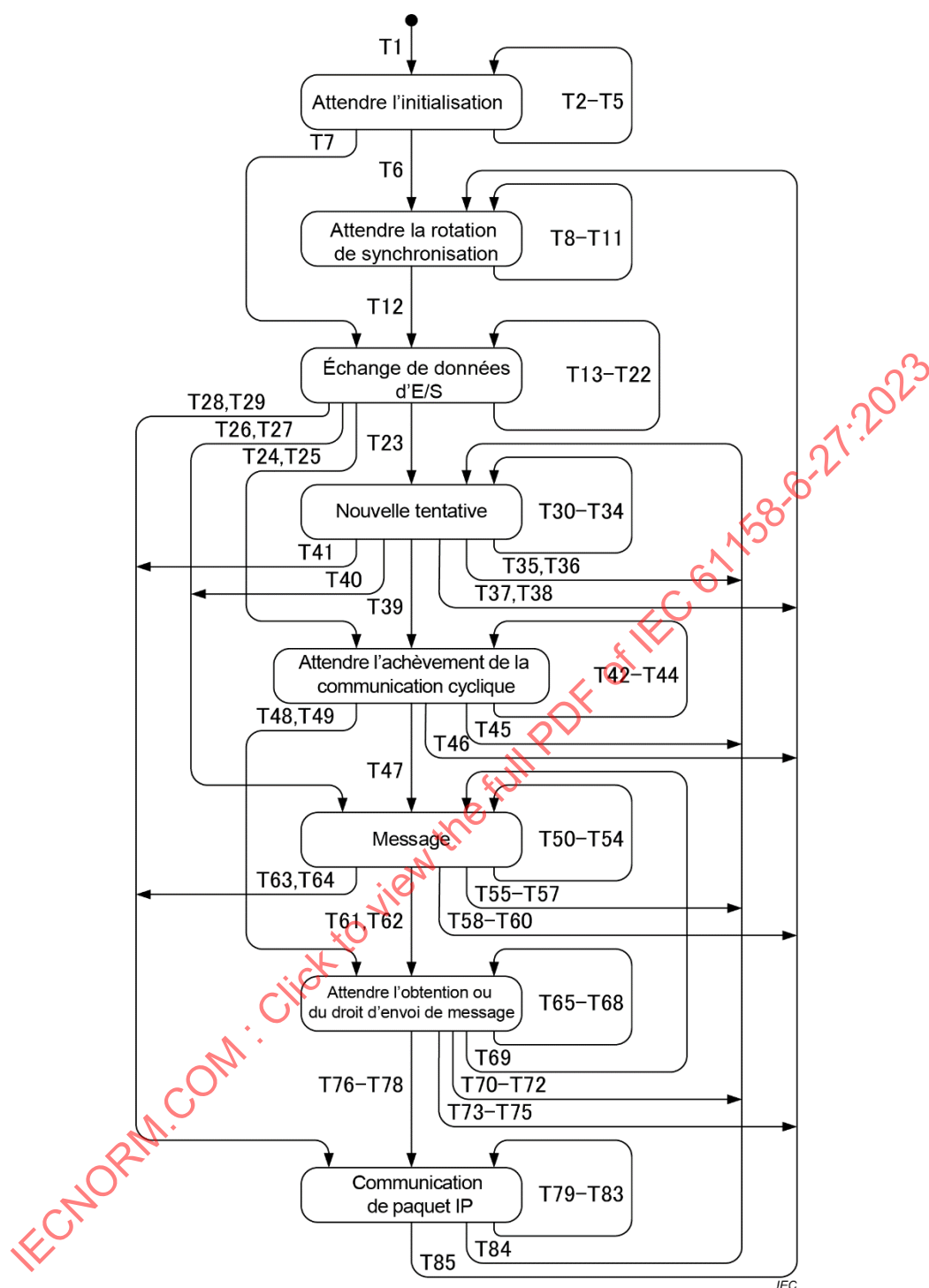


Figure 11 – Schéma de transition d'états de la CTC de l'appareil principal

Tableau 14 – Table de transition d'états de la CTC de l'appareil principal

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
1	Tous	Mise sous tension ou CTC_Reset.req / néant	Attendre l'initialisation
2	Attendre l'initialisation	CTC_Set_Par.req(Par_ID, Val) / CTC_SET_PAR(Par_ID, Val, Result); CTC_Set_Par.cnf(Result);	Attendre l'initialisation
3	Attendre l'initialisation	CTC_Get_Par.req(Par_ID) / CTC_GET_PAR(Par_ID, Val, Result); CTC_Get_Par.cnf(Result, Par_ID, Val);	Attendre l'initialisation
4	Attendre l'initialisation	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Attendre l'initialisation
5	Attendre l'initialisation	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Attendre l'initialisation
6	Attendre l'initialisation	CTC_Start.req [CHECK_SYNC_RIGHT() = "SLV"] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Attendre la notification de synchronisation
7	Attendre l'initialisation	CTC_Start.req [CHECK_SYNC_RIGHT() = "OWN"] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Échange de données d'E/S
8	Attendre la notification de synchronisation	EXPIRED_TIMER(T(Tcycle)) = "True" / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Attendre la notification de synchronisation
9	Attendre la notification de synchronisation	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if(Result == FCMD_SYNC) { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tsync)); RESTART_TIMER(T(Tipst)); }	Attendre la notification de synchronisation
10	Attendre la notification de synchronisation	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Attendre la notification de synchronisation
11	Attendre la notification de synchronisation	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU) CTC-READ-DATA.cnf(Result, SDU)	Attendre la notification de synchronisation
12	Attendre la notification de synchronisation	EXPIRED_TIMER(T(Tsync)) = "True" / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI"); { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Transfert des données d'entrée

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
13	Échange de données d'E/S	EXPIRED_TIMER(T(Tcycle)) = "True" / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Échange de données d'E/S
14	Échange de données d'E/S	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "LO") && (T(Trspst) < V(Trspst)) && (Rcv_sts = "OK")] / EXEC_IP_SEG(Port_num, RcvIPDLSDU, None);	Échange de données d'E/S
15	Échange de données d'E/S	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "LO") && (T(Trspst) >= V(Trspst)) && (Rcv_sts = "OK")] / Result = CHECK_PDU_CYCR(DLSDU, V(Nslave)); if(Result == "OK") { V(Nrcv) = V(Nrcv) + 1; PUT_RCVCHK_LIST(V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts); }	Échange de données d'E/S
16	Réception des données d'E/S	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "LO") && (T(Trspst) >= V(Trspst)) && (Rcv_sts <> "OK")] / CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	Échange de données d'E/S
17	Échange de données d'E/S	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "HI") && (Rcv_sts = "OK")] / Result = CHECK_PDU_CYCR(DLSDU, V(Nslave)); if(Result == "OK") { V(Nrcv) = V(Nrcv) + 1; PUT_RCVCHK_LIST(V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts); }	Échange de données d'E/S
18	Réception des données d'E/S	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_TRANS_MODE() = "HI") && (Rcv_sts <> "OK")] / CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	Échange de données d'E/S
19	Échange de données d'E/S	EXPIRED_TIMER(T(Trspst)) = "True" / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp));	Échange de données d'E/S
20	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) = 0) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	Échange de données d'E/S
21	Échange de données d'E/S	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Échange de données d'E/S
22	Échange de données d'E/S	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Échange de données d'E/S

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
23	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) > 0) && (V(Nrcv_max) > V(Nrcv))] / for((V(Nslave) = GET_RCVCHK_LIST()) <> 0) { PUT_RETRY_LIST(V(Nslave)); PUT_RCVCHK_LIST(V(Nslave)); } V(Tretry) = CALC_RETIRE(); DLSDU = BUILD_PDU_RETREQ(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tretry), V(Tretry)); V(Nretry) = V(Nmax_retry) – 1;	Nouvelle tentative
24	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) > 0) && (V(Nrcv_max) >= V(Nrcv)) && (V(Nsmst) > 0)] / none	Attendre l'achèvement de la communication cyclique
25	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [V(Nmax_retry) = 0 && V(Nsmst) > 0] / none	Attendre l'achèvement de la communication cyclique
26	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) > 0) && (V(Nrcv_max) >= V(Nrcv)) && (V(Nsmst) = 0) && (CHECK_MSG_EN() <> 0)] / none	Message
27	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) = 0) && (V(Nsmst) = 0) && (CHECK_MSG_EN() <> 0)] / none	Message
28	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [((V(Nmax_retry) > 0) && (V(Nrcv_max) >= V(Nrcv))) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0)] / none	Communication de paquet IP
29	Échange de données d'E/S	EXPIRED_TIMER(T(Tresp)) = "True" [(V(Nmax_retry) = 0) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0)] / none	Communication de paquet IP
30	Nouvelle tentative	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_CYCR(DLSDU, V(Nslave)); if(Result == "OK") { RMV_RETRY_LIST(V(Nslave)); } STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	Nouvelle tentative
31	Nouvelle tentative	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts <> "OK"] / CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts);	Nouvelle tentative
32	Nouvelle tentative	EXPIRED_TIMER(T(Tretry)) = "True" [(V(Nretry) <> 0) && (GET_RETRY_LIST() <> 0) && (CMP_RTIM(T(Tcycle), CALC_RETIRE()) > 0)] / V(Tretry) = CALC_RETIRE(); DLSDU = BUILD_PDU_RETREQ(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tretry), V(Tretry)); V(Nretry) = V(Nretry) – 1;	Nouvelle tentative
33	Nouvelle tentative	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Nouvelle tentative
34	Nouvelle tentative	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Nouvelle tentative

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
35	Nouvelle tentative	EXPIRED_TIMER(T(Tretry)) = "True" [(CHECK_SYNC_RIGHT() = "OWN") && ((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0))) && ((V(Nsmst) = 0) && (CHECK_MSG_EN() = 0)) && (CHECK_IP_EN() = 0)] / none	Échange de données d'E/S
36	Nouvelle tentative	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() = "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() = "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Échange de données d'E/S
37	Nouvelle tentative	EXPIRED_TIMER(T(Tretry)) = "True" [(CHECK_SYNC_RIGHT() = "SLV") && ((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0)) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	Attendre la notification de synchronisation
38	Nouvelle tentative	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Attendre la notification de synchronisation
39	Nouvelle tentative	EXPIRED_TIMER(T(Tretry)) = "True" [((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0)) && (V(Nsmst) > 0))] / none	Attendre l'achèvement de la communication cyclique
40	Nouvelle tentative	EXPIRED_TIMER(T(Tretry)) = "True" [((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0)) && (V(Nsmst) = 0) && (CHECK_MSG_EN() <> 0))] / none	Message
41	Nouvelle tentative	EXPIRED_TIMER(T(Tretry)) = "True" [((V(Nretry) = 0) (GET_RETRY_LIST() = 0) (CMP_RTIM(T(Tcycle), CALC_RETIME()) <= 0)) && (V(Nsmst) = 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0))] / none	Communication de paquet IP
42	Attendre l'achèvement de la communication cyclique	SRC_Rcv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / DESTROY_PDU_CYCR(DLSDU);	Attendre l'achèvement de la communication cyclique
43	Attendre l'achèvement de la communication cyclique	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Attendre l'achèvement de la communication cyclique

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
44	Attendre l'achèvement de la communication cyclique	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Attendre l'achèvement de la communication cyclique
45	Attendre l'achèvement de la communication cyclique	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() = "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() = "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Echange de données d'E/S
46	Attendre l'achèvement de la communication cyclique	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Attendre la notification de synchronisation
47	Attendre l'achèvement de la communication cyclique	EXPIRED_TIMER(T(Tmsgst)) = "True" [(CHECK_MSG_RIGHT() <> 0) && (CHECK_MSG_EN() <> 0)] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
48	Attendre l'achèvement de la communication cyclique	EXPIRED_TIMER(T(Tmsgst)) = "True" [(CHECK_MSG_RIGHT() <> 0) && (CHECK_MSG_EN() = 0)] / DLSDU = BUILD_PDU_TOKEN(V(Nnext_add), Msg); SRC_Send_Frame.req(Port_sel, DLSDU);	Attendre l'obtention du droit d'envoi de message
49	Attendre l'achèvement de la communication cyclique	EXPIRED_TIMER(T(Tmsgst)) = "True" [CHECK_MSG_RIGHT() = 0] / none	Attendre l'obtention du droit d'envoi de message
50	Message	EXPIRED_TIMER(T(Tmsgst)) = "True" [CHECK_MSG_RIGHT() <> 0] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
51	Message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / EXEC_MSGC_SEG(Result, DLSDU, None);	Message
52	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) > 0)] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
53	Message	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Message

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
54	Message	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Message
55	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "OWN") && (CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	Échange de données d'E/S
56	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "OWN") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_IP_EN() = 0)] / none	Échange de données d'E/S
57	Message	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Échange de données d'E/S
58	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "SLV") && (CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	Attendre la notification de synchronisation
59	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_SYNC_RIGHT() = "SLV") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_IP_EN() = 0)] / none	Attendre la notification de synchronisation
60	Message	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Attendre la notification de synchronisation
61	Message	EXPIRED_TIMER(T(Tmsgst)) = "True" [CHECK_MSG_RIGHT() = 0] / none	Attendre l'obtention du droit d'envoi de message
62	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) > 0)] / DLSDU = BUILD_PDU_TOKEN(V(Nnext_add), Msg); SRC_Send_Frame.req(Port_sel, DLSDU);	Attendre l'obtention du droit d'envoi de message
63	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() <> 0)] / none	Communication de paquet IP

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
64	Message	EXPIRED_TIMER(T(Tmsg)) = "True" [(CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_IP_EN() <> 0)] / none	Communication de paquet IP
65	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [CHECK_PDU_TOKEN(TokenType, DLSDU) <> 0] / EXEC_MSGS_SEG(Result, DLSDU, None);	Attendre l'obtention du droit d'envoi de message
66	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) <> 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) > 0)] / DESTROY_PDU_CYCR(DLSDU);	Attendre l'obtention du droit d'envoi de message
67	Attendre l'obtention du droit d'envoi de message	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Attendre l'obtention du droit d'envoi de message
68	Attendre l'obtention du droit d'envoi de message	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Attendre l'obtention du droit d'envoi de message
69	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_MSG_EN() <> 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) > 0)] / V(Tmsg) = CALC_MSGTIME(); EXEC_MSGC_SEG(OK, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsg), V(Tmsg));	Message
70	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "OWN") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	Envoi de données de sortie d'E/S
71	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "OWN") && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	Envoi de données de sortie d'E/S
72	Attendre l'obtention du droit d'envoi de message	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() == "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() == "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Échange de données d'E/S

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
73	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "SLV") && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() = 0)] / none	Envoi de données de sortie d'E/S
74	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_SYNC_RIGHT() = "SLV") && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() = 0)] / none	Envoi de données de sortie d'E/S
75	Attendre l'obtention du droit d'envoi de message	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Attendre la notification de synchronisation
76	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CHECK_MSG_EN() = 0) && (CMP_RTIM(T(Tcycle), CALC_MSGTIME()) <= 0) && (CHECK_IP_EN() <> 0)] / none	Communication de paquet IP
77	Attendre l'obtention du droit d'envoi de message	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(CHECK_PDU_TOKEN(TokenType, DLSDU) = 0) && (CMP_RTIM(T(Tcycle), CALC_TOKENTIME(Node_ID)) <= 0) && (CHECK_MSG_EN() = 0) && (CHECK_IP_EN() <> 0)] / none	Communication de paquet IP
78	Attendre l'obtention du droit d'envoi de message	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF) if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	Communication de paquet IP
79	Communication de paquet IP	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF) if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	Communication de paquet IP
80	Communication de paquet IP	EXPIRED_TIMER(T(Tip)) = "True" [CHECK_IP_PKT() <> 0] / if(CMP_RTIM(T(Tcycle), CALC_IPTIME()) > 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	Communication de paquet IP
81	Communication de paquet IP	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, RcvIPDLSDU) [Rcv_sts = "OK"] / EXEC_IP_SEG(Port_sel, RcvIPDLSDU, None);	Communication de paquet IP
82	Communication de paquet IP	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Communication de paquet IP

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
83	Communication de paquet IP	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Communication de paquet IP
84	Communication de paquet IP	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_SYNC_CYCC(); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tmsgst), V(Tmsgst)); START_TIMER(T(Tipst), V(Tipst)); if(CHECK_TRANS_MODE() = "HI") { SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tresp), V(Tresp)); } if(CHECK_TRANS_MODE() = "LO") { SRC_Repeat_Sel.req(SandF, SandF); START_TIMER(T(Trspst), V(Trspst)); } V(Nrcv) = 0; V(Nrcv_max) = GET_OUTPUT_NODENUM(); SET_RCVCHK_LIST();	Échange de données d'E/S
85	Communication de paquet IP	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Attendre la notification de synchronisation

8.2.2.2.2 Machine de protocole CTC de l'appareil subordonné

Le schéma de transition d'états CTC de l'appareil subordonné est tel que représenté à la Figure 12 et le tableau de transition d'états CTC de l'appareil subordonné est tel que représenté dans le Tableau 15.

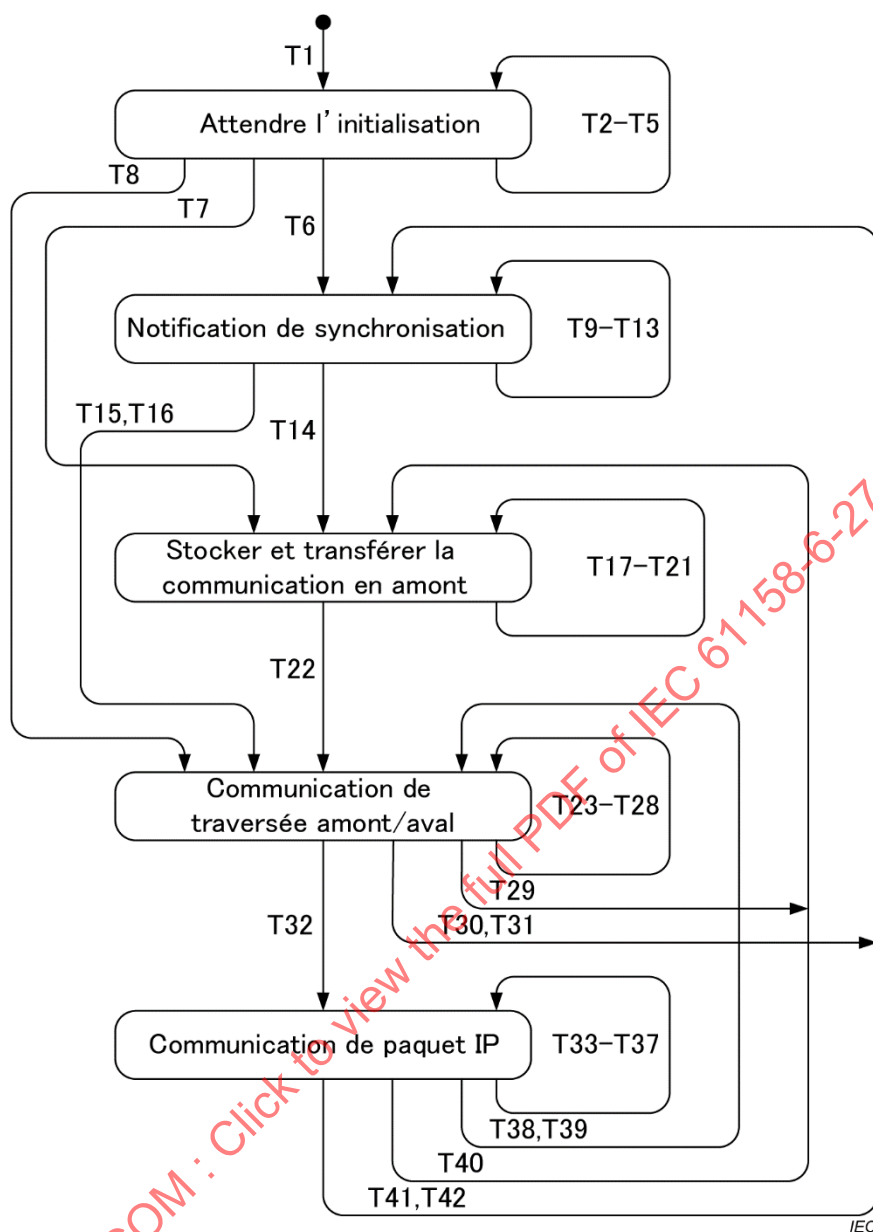


Figure 12 – Schéma de transition d'états de la CTC de l'appareil subordonné

Tableau 15 – Tableau de transition d'états de la CTC de l'appareil subordonné

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
1	Tous	Mise sous tension ou CTC_Reset.req / néant	Attendre l'initialisation
2	Attendre l'initialisation	CTC_Set_Par.req(Par_ID, Val) / CTC_SET_PAR(Par_ID, Val, Result); CTC_Set_Par.cnf(Result);	Attendre l'initialisation
3	Attendre l'initialisation	CTC_Get_Par.req(Par_ID) / CTC_GET_PAR(Par_ID, Val, Result); CTC_Get_Par.cnf(Result, Par_ID, Val);	Attendre l'initialisation
4	Attendre l'initialisation	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Attendre l'initialisation
5	Attendre l'initialisation	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Attendre l'initialisation
6	Attendre l'initialisation	CTC_Start.req [(CHECK_SYNC_RIGHT() = "OWN") (CHECK_SYNC_RIGHT() = "SLV")] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Notification de synchronisation
7	Attendre l'initialisation	CTC_Start.req [(CHECK_SYNC_RIGHT () <> "OWN") && (CHECK_SYNC_RIGHT () <> "SLV") && (MST1_TRANS_MODE () = "LO")] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Stocker et transférer la communication en amont
8	Attendre l'initialisation	CTC_Start.req [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI")] / START_TIMER(T(Tcycle), V(Tcycle)); CTC_Start.cnf;	Communication de traversée amont/aval
9	Notification de synchronisation	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Notification de synchronisation
10	Notification de synchronisation	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel,DLSDU); SRC_Repeat_Sel.req(CandT,CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Notification de synchronisation
11	Notification de synchronisation	SRC_Recv_Frame.ind (Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if(Result == "sync") { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tsync)); RESTART_TIMER(T(Tipst)); }	Notification de synchronisation
12	Notification de synchronisation	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU) CTC-WRITE-DATA.cnf(Result)	Notification de synchronisation
13	Notification de synchronisation	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Notification de synchronisation
14	Notification de synchronisation	EXPIRED_TIMER(T(Tsync)) = "True" [MST1_TRANS_MODE() = "LO"] / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end));	Stocker et transférer la communication en amont

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
15	Notification de synchronisation	EXPIRED_TIMER(T(Tsync)) = "True" [(MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() <> 0)] / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel,DLSDU); SRC_Repeat_Sel.req(CandT,CandT); SET_SEND_TIMING();	Amont/aval Communication de traversée amont/aval
16	Notification de synchronisation	EXPIRED_TIMER(T(Tsync)) = "True" [(MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() = 0)] / SRC_Repeat_Sel.req(CandT, CandT); SET_SEND_TIMING();	Communication de traversée amont/aval
17	Stocker et transférer la communication en amont	EXPIRED_TIMER(T(Tcycle)) = "True" / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end)); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Stocker et transférer la communication en amont
18	Stocker et transférer la communication en amont	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Port_num = 0] / Result = CHECK_PDU_FCMD(DLSDU) domain = CHECK_PDU_SRC(DLSDU) if(Result == "sync") { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tc_end)); RESTART_TIMER(T(Tipst)); } if(BELONG_1ST_DOMAIN() <> 0) { STORE_PDU_CYCC(domain, V(Nslave), DLSDU, Rcv_sts) }	Stocker et transférer la communication en amont
19	Stocker et transférer la communication en amont	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Port_num <> 0] / EXEC_IP_SEG(Port_num, RcvIPDLSDU, None);	Stocker et transférer la communication en amont
20	Stocker et transférer la communication en amont	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Stocker et transférer la communication en amont
21	Stocker et transférer la communication en amont	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Stocker et transférer la communication en amont
22	Stocker et transférer la communication en amont	EXPIRED_TIMER(T(Tc_end)) = "True" / if(BELONG_1ST_DOMAIN() <> 0) { CTC-Event.ind(CTC_Ev_Tcycle); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel, DLSDU); } SRC_Repeat_Sel.req(CandT,CandT);	Communication de traversée amont/aval
23	Communication de traversée amont/aval	SEND_TIMING_EVENT(mst) = "True" / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel,DLSDU);	Communication de traversée amont/aval

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
24	Communication de traversée amont/aval	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if(Result == FCMD_SYNC) { RESTART_TIMER(T(Tcycle)); RESTART_TIMER(T(Tipst)); if(BELONG_1ST_DOMAIN() <> 0) { STORE_PDU_CYCC(V(Nslave), DLSDU, Rcv_sts) } } else if(Result == FCMD_OUTPUT) { STORE_PDU_CYCC(V(Nslave), DLSDU, Rcv_sts) } else if(Result == FCMD_INPUT) { CHECK_PDU_CYCR(DLSDU, V(Nslave)); STORE_PDU_CYCR(V(Nslave), DLSDU, Rcv_sts) } else if(Result == FCMD_REQ_SEND) { Result = CHECK_RETRY(DLSDU, Rcv_sts, Port_num); if(Result == "TRUE") { DLSDU = BUILD_PDU_CYCR(); SRC_Send_Frame.req(Port_sel, DLSDU); } } else if(Result == "FCMD_MSG") { EXEC_MSGS_SEG(Result, DLSDU, DLSDU); }</pre>	Communication de traversée amont/aval
25	Communication de traversée amont/aval	<pre> EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() <> 0)] / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();</pre>	Communication de traversée amont/aval
26	Communication de traversée amont/aval	<pre> EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() = 0)] / START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();</pre>	Amont/aval Communication de traversée amont/aval
27	Communication de traversée amont/aval	<pre> CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);</pre>	Communication de traversée amont/aval
28	Communication de traversée amont/aval	<pre> CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);</pre>	Amont/aval Communication de traversée amont/aval
29	Amont/aval Communication de traversée amont/aval	<pre> EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "LO")] / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end)); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();</pre>	Stocker et transférer la communication en amont
30	Amont/aval Communication de traversée amont/aval	<pre> EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));</pre>	Notification de synchronisation

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
31	Amont/aval Communication de traversée amont/aval	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Notification de synchronisation
32	Amont/aval Communication de traversée amont/aval	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF); if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	Communication de paquet IP
33	Communication de paquet IP	EXPIRED_TIMER(T(Tipst)) = "True" / SRC_Repeat_Sel.req(SandF, SandF); if(CHECK_IP_PKT() <> 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	Communication de paquet IP
34	Communication de paquet IP	EXPIRED_TIMER(T(Tip)) = "True" [CHECK_IP_PKT() <> 0] / if(CMP_RTIM(T(Tcycle), CALC_IPTIME()) > 0) { V(Tip) = CALC_IPTIME(); EXEC_IP_SEG(Port_sel, None, DLSDU); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tip), V(Tip)); }	Communication de paquet IP
35	Communication de paquet IP	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / EXEC_IP_SEG(Port_sel, DLSDU, None);	Communication de paquet IP
36	Communication de paquet IP	CTC-WRITE-DATA.req(SAP_ID, SDU) / Result = SET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-WRITE-DATA.cnf(Result);	Communication de paquet IP
37	Communication de paquet IP	CTC-READ-DATA.req(SAP_ID) / Result = GET_CYC_DATA(SAP_ID, Node_ID, SDU); CTC-READ-DATA.cnf(Result, SDU);	Communication de paquet IP
38	Communication de paquet IP	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() <> 0)] / CTC-Event.ind(CTC_Ev_Tsync); DLSDU = BUILD_PDU_CYCR(mst); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Amont/aval Communication de traversée amont/aval
39	Communication de paquet IP	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "HI") && (BELONG_1ST_DOMAIN() = 0)] / START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Amont/aval Communication de traversée amont/aval
40	Communication de paquet IP	EXPIRED_TIMER(T(Tcycle)) = "True" [(CHECK_SYNC_RIGHT() <> "OWN") && (CHECK_SYNC_RIGHT() <> "SLV") && (MST1_TRANS_MODE() = "LO")] / SRC_Repeat_Sel.req(CandT, SandF); START_TIMER(T(Tc_end), V(Tc_end)); START_TIMER(T(Tipst), V(Tipst)); SET_SEND_TIMING();	Stocker et transférer la communication en amont

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
41	Communication de paquet IP	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Notification de synchronisation
42	Communication de paquet IP	EXPIRED_TIMER(T(Tcycle)) = "True" [CHECK_SYNC_RIGHT() = "SLV"] / SRC_Repeat_Sel.req(CandT, CandT); START_TIMER(T(Tsync), V(Tsync)); START_TIMER(T(Tipst), V(Tipst));	Notification de synchronisation

8.2.2.2.3 Liste des macros de la CTC

Le Tableau 16 indique la liste des macros utilisées pour la transition d'états de la CTC.

Tableau 16 – Liste des macros de la CTC

Nom de la fonction	Paramètre		Valeur de retour	Action
	Entrée	Sortie		
CHECK_SYNC_RIGHT	Néant	Néant	SM_ID	Notifier l'ID de nœud lorsqu'un appareil principal autre que le 1er appareil principal est le gestionnaire de synchronisation. Renvoyer l'ID propre à son propre nœud ou à un autre appareil subordonné (y compris le 2e appareil principal).
START_TIMER	Tim_ID, Val	Néant	Néant	Démarrer le temporisateur spécifié par Tim_ID à la valeur de temporisation spécifiée par Val. Ce temporisateur exécute des opérations de recharge automatique.
RESTART_TIMER	Tim_ID	Néant	Néant	Corriger le temporisateur spécifié par Tim_ID.
EXPIRED_TIMER	Tim_ID	Néant	Néant	Notifier à Tim_ID que le temporisateur est expiré.
CALC_RETIME	Néant	Néant	Retry_Tim	Calculer le temps de surveillance de la réponse de nouvelle tentative en fonction de la liste des nouvelles tentatives et du numéro de nœud de nouvelle tentative.
CALC_MSGTIME	Néant	Néant	Msg_Tim	Calculer le temps de surveillance de la réponse de message en fonction de la taille des données de message.
CALC_TOKENTIME	Node_ID	Néant	Token_Tim	Calculer le temps pour transmettre le paquet de jeton au nœud spécifié par Node_ID.
CALC_IPTIME	Néant	Néant	IP_Tim	Calculer le temps de transmission IP en fonction de la taille des données IP.

Nom de la fonction	Paramètre		Valeur de retour	Action
	Entrée	Sortie		
CMP_RTIM	Tim_ID1, Tim_ID2	Néant	Résultat	Comparer la valeur actuelle de Tim_ID1 et Tim_ID2. Ensuite, si le temps jusqu'à la temporisation de Tim_ID1 est supérieur ou égal à Tim_ID2, stocker 1 comme résultat.
SET_CYC_DATA	SAP_ID, Node_ID, SDU	Néant	Résultat	Mettre à jour le tampon pour l'échange de données E/S dans la CTC spécifiée par SAP_ID et Node_ID dans la SDU. Renvoyer le résultat indiquant si la mise à jour s'est terminée normalement.
GET_CYC_DATA	SAP_ID, Node_ID	SDU	Résultat	Stocker les données dans le tampon pour l'échange de données E/S dans la CTC spécifiée par SAP_ID et Node_ID dans SDU. Renvoyer le résultat indiquant si le stockage s'est terminé normalement.
BUILD_PDU_SYNC	Néant	Néant	DLSDU	Créer une DLSDU où sont stockées les données de sortie.
BUILD_PDU_CYCC	Néant	Néant	DLSDU	Créer la DLSDU où sont stockées les notifications de synchronisation et les données de sortie.
BUILD_PDU_SYNC_CYCC	Néant	Néant	DLSDU	Créer une DLSDU où sont stockées les demandes de retransmission.
BUILD_PDU_RETREQ	Néant	Néant	DLSDU	Créer une DLSDU où sont stockées les données de sortie.
BUILD_PDU_TOKEN	Node_ID TokenType	Néant	DLSDU	Créer DLSDU où sont stockés les jetons spécifiés par TokenType.
CHECK_PDU_CYCR	DLSDU	AXIS_NUM	Résultat	Détecter le numéro d'axe source, AXIS_NUM, de la DLSDU reçue, comparer et vérifier avec les paramètres d'axe, et renvoyer le résultat en tant que résultat.
CHECK_PDU_FCMD	DLSDU	Néant	Résultat	Détecter le code d'instruction de tête de la DLSDU reçue et renvoyer le résultat en tant que résultat. De plus, stocker le numéro d'axe de la source en tant que AXIS_NUM.
STORE_PDU_CYCR	AXIS_NUM, Rcv_sts, DLSDU	Néant	Néant	Stocker la DLSDU dans le tampon de données reçu de AXIS_NUM en fonction du statut de réception, Rcv_sts.
DESTROY_PDU_CYCR	DLSDU	Néant	Néant	Détruire la DLSDU spécifiée.
SET_RCVCHK_LIST	Néant	Néant	Néant	Créer la liste de contrôle de réception des données d'entrée.
PUT_RCVCHK_LIST	AXIS_NUM	Néant	Néant	Enregistrer l'axe du numéro d'axe spécifié par AXIS_NUM comme reçu normalement dans la liste de contrôle de réception des données d'entrée.

Nom de la fonction	Paramètre		Valeur de retour	Action
	Entrée	Sortie		
GET_RCVCHK_LIST	Néant	Néant	AXIS_NUM	Renvoyer le plus petit numéro parmi les numéros d'axe décrits comme n'étant pas reçus normalement dans la liste de contrôle de réception des données d'entrée en tant que valeur de retour.
PUT_RETRY_LIST	AXIS_NUM	Néant	Néant	Enregistrer le numéro d'axe, AXIS_NUM, dans la liste des nouvelles tentatives.
RMV_RETRY_LIST	AXIS_NUM	Néant	Néant	Supprimer le numéro d'axe, AXIS_NUM, de la liste des nouvelles tentatives.
GET_OUTPUT_NODENUM	Néant	Néant	Néant	Obtenir le numéro de nœud d'envoi des données de sortie.
EXEC_MSGC_SEG	Rcv_sts, RcvDLSDU	SndDLSDU	Néant	Il s'agit d'un message pour la fonction de commande de nœud primaire. Lors de la réception du message, l'état de réception est stocké dans Rcv_sts et les données reçues sont stockées dans RcvDLSDU. Lors de l'envoi d'un message, le message d'envoi stocké dans SndDLSDU est envoyé.
EXEC_MSGS_SEG	Rcv_sts, RcvDLSDU	SndDLSDU	Néant	Il s'agit d'un message pour la fonction de commande de nœud secondaire. Lors de la réception du message, l'état de réception est stocké dans Rcv_sts et les données reçues sont stockées dans RcvDLSDU. Lors de l'envoi d'un message, le message d'envoi stocké dans SndDLSDU est envoyé.
EXEC_IP_SEG	Port_sel, RcvIPDLSDU	SndIPDLSDU	Néant	Stocker le paquet reçu au port par Port_sel dans RcvIPDLSDU. Ou bien, envoyer le paquet, SndIPDLSDU, du port par Port_sel.
CHECK_MSG_EN	Néant	Néant	Enable_flag	Notifier si la bande de messages est définie et si l'utilisateur DLS émet une demande de message
CHECK_MSG_RIGHT	Néant	Néant	right_flag	Notifier si son propre nœud possède le droit d'envoyer le message
CHECK_IP_EN	Néant	Néant	Enable_flag	Notifier si la bande de communication IP est définie
CHECK_TRANS_MODE	Néant	Néant	Trans_Mode	Notifier si la communication cyclique de sa propre bande est ou non un mode de transfert élevé.
MST1_TRANS_MODE	Néant	Néant	Trans_Mode	Notifier si la communication cyclique du 1er domaine de commande est un mode de transfert élevé.
CHECK_PDU_TOKEN	TokenType, DLSDU	Néant	Résultat	Vérifier si la DLSDU reçue est ou non le jeton spécifié par TokenType et notifier le résultat en tant que résultat.

Nom de la fonction	Paramètre		Valeur de retour	Action
	Entrée	Sortie		
CHECK_IP_PKT	Néant	Néant	Résultat	Vérifier si le paquet IP à envoyer existe ou non et notifier le résultat en tant que résultat.
BELONG_1ST_DOMAIN	Néant	Néant	Résultat	Notifier si son propre nœud appartient ou non au 1er domaine de commande.
BUILD_PDU_CYCR	MST_NUM	Néant	DLSDU	Créer la DLSDU dans laquelle sont stockées les données à entrer dans l'appareil principal spécifié par MST_NUM.
SET_SEND_TIMING	Néant	Néant	Néant	Définir le cadencement pour envoyer une réponse pour chaque 2e domaine de commande auquel il appartient.
SEND_TIMING_EVENT	MST_NUM	Néant	Néant	Notifier l'occurrence d'un événement d'envoi dans le domaine de commande que commande MST_NUM.
STORE_PDU_CYCC	AXIS_NUM, Rcv_sts, DLSDU	Néant	Néant	Détecter les données devant être sorties de la DLSDU vers son propre numéro d'axe, AXIS_NUM, en fonction du statut de réception. RCV_sts, et stocker dans le tampon de données reçu.

8.2.2.2.4 Interface CTC-FDC

Les échanges de primitives entre la FDC et la CTC sont représentés ci-dessous. Le Tableau 17 est pour le cas qui a été émis par la FDC et le Tableau 18 pour le cas qui a été émis par la CTC.

Tableau 17 – Primitives et paramètres pour l'interface FDC émise par la FDC

Primitive	Source	Paramètre	Fonction
CTC-WRITE-DATA.req	FDC	SAP_ID, FDCSDU	Demande d'écriture de données d'envoi
CTC-READ-DATA.req	FDC	SAP_ID	Demande de lecture de données de réception
CTC-SDA.req	FDC	SAP_ID, Node_ID, Length, FDCSDU	Demande de données d'envoi avec confirmation de transmission
CTC-SDN.req	FDC	SAP_ID, Node_ID, Length, FDCSDU	Demande de données d'envoi sans confirmation de transmission
CTC-GET-STATUS.req	FDC	Sts_ID	Demande de référence de paramètre FSMUL

Tableau 18 – Primitives et paramètres pour l'interface FSM émise par la CTC

Primitive	Source	Paramètre
CTC-WRITE-DATA.cnf	CTC	Résultat
CTC-READ-DATA.cnf	CTC	Résultat, FDCSDU
CTC-SDA.ind	CTC	Node_ID, Length, FDCSDU
CTC-SDA.cnf	CTC	Résultat
CTC-SDN.ind	CTC	Node_ID, Length, FDCSDU
CTC-SDN.cnf	CTC	Résultat
CTC-GET-STATUS.cnf	CTC	Résultat, Cur_Status
CTC-EVENT.ind	CTC	Event_ID

8.2.2.2.3 Couche sous-jacente de gestion système de terrain

8.2.2.2.3.1 Machine de protocole FSM

Voir 8.3.

8.2.2.2.3.2 Primitives échangées entre FSM et FSMUL

Les échanges de primitives entre la FSM et la FSMUL sont représentés ci-dessous. Le Tableau 19 est pour le cas qui a été émis par la FSM et le Tableau 20 pour le cas qui a été émis par la FSMUL.

Tableau 19 – Liste des primitives et des paramètres (source FSM)

Primitive	Source	Paramètre
FSMUL-RESET.req	FSM	
FSMUL-SET-VALUE.req	FSM	Var_ID, Val
FSMUL-GET-VALUE.req	FSM	Var_ID
FSMUL-DETECT.req	FSM	
FSMUL-RECOG.req	FSM	
FSMUL-MEAS-DELAY.req	FSM	
FSMUL-ASYNC.req	FSM	
FSMUL-CONFIG.req	FSM	
FSMUL-START_SYNC.req	FSM	
FSMUL-START.req	FSM	
FSMUL-WDLYCNFG.req	FSM	
FSMUL-CLR-ERR.req	FSM	Err_Status

Tableau 20 – Liste des primitives et des paramètres (source FSMUL)

Primitive	Source	Paramètre
FSMUL-RESET.cnf	FSMUL	Résultat
FSMUL-SET-VALUE.cnf	FSMUL	Résultat
FSMUL-GET-VALUE.cnf	FSMUL	Result, Val
FSMUL-DETECT.cnf	FSMUL	
FSMUL-RECOG.cnf	FSMUL	
FSMUL-MEAS-DELAY.cnf	FSMUL	Résultat
FSMUL-MEAS-DELAY.ind	FSMUL	Delay_time
FSMUL-CONFIG.cnf	FSMUL	
FSMUL-START-SYNC.cnf	FSMUL	
FSMUL-START-SYNC.ind	FSMUL	
FSMUL-START.cnf	FSMUL	Résultat
FSMUL-START.ind	FSMUL	Com_Mode, Cycle_time, Max_Delay, TM_unit
FSMUL-CLR-ERR.cnf	FSMUL	Résultat
FSMUL-EVENT.ind	FSMUL	Event_ID

8.2.2.2.3.3 Paramètres utilisés avec les primitives FSM

Voir 6.3.1.2.3 de l'IEC 61158-5-27 pour les paramètres de la primitive de service.

8.2.2.3 Machine de protocole de transfert de PTP (PTP_FW)

Les concepts conformes à la norme IEEE 802.1AS doivent être appliqués. La présente partie de la spécification définit l'utilisation de la norme IEEE 802.1AS.

8.2.3 Ponts virtuels

Les concepts conformes à la norme IEEE 802.1Q doivent être appliqués. La présente partie de la spécification définit l'utilisation de la norme IEEE 802.1Q.

8.2.4 Suite IP

Les concepts conformes aux normes IETF RFC 768, IETF RFC 791, IETF RFC 826 et IETF RFC 1112 doivent être appliqués. La présente partie de la spécification définit l'utilisation des normes IETF RFC 791, IETF RFC 826 et IETF RFC 1112.

8.2.5 Machine de protocole de correspondance de DLL (DMPM)

8.2.5.1 Généralités

Le protocole de correspondance de DLL est constitué de plusieurs machines de protocole:

- Le traitement des services invoqués par d'autres entités de la couche Application;
- Transfert des trames RT et PTP (RT_FW, PTP_FW);

NOTE 1 Le transfert des trames normalisées est conforme à la norme IEEE 802.1D et n'est donc pas spécifié ici.

- Correspondance vers l'interface de commande d'accès au support (MMAC) et fonctions d'envoi/réception pour les messages de synchronisation;

NOTE 2 L'invocation de MAC est conforme à l'ISO/IEC/IEEE 8802-3 et n'est donc pas spécifiée ici.

- PTP est une fonction d'horodatage fournie par le matériel dans l'interface PHY-MAC pour réaliser la norme IEEE Std 802.1AS. Cette mise en œuvre est facultative.

L'interface entre le gestionnaire de service, les machines de transfert et la correspondance à la commande d'accès au support (MMAC) est constituée d'un ensemble de files d'attente et de données globales.

La Figure 13 représente la structuration des machines de protocole pour la DMPM d'un appareil.

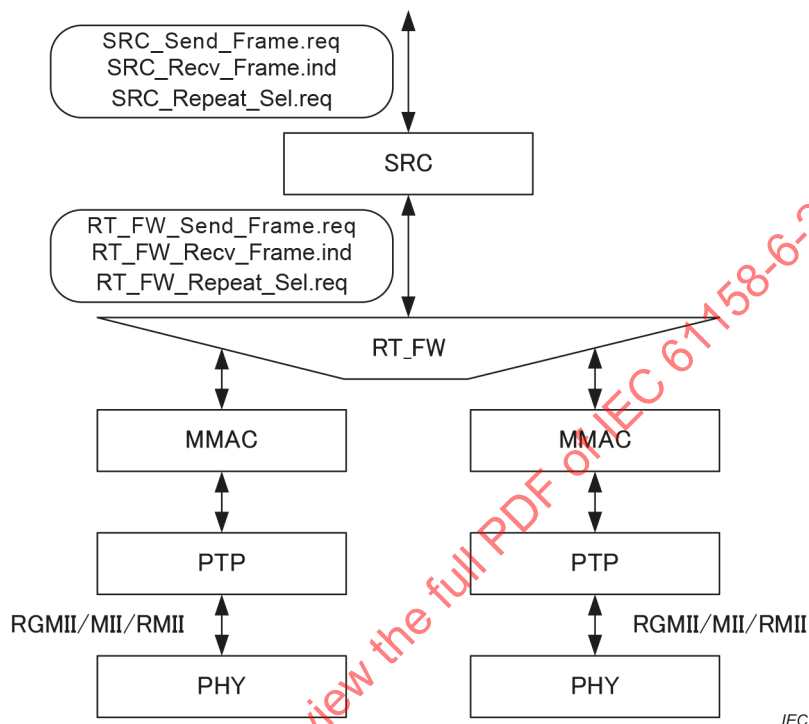


Figure 13 – Structure des machines de protocole à l'intérieur de la DMPM (pont)

8.2.5.2 Couche sous-jacente de commande d'émission et réception (SRC)

8.2.5.2.1 Machine de protocole SRC

Le schéma de transition d'états de la SRC est tel que représenté à la Figure 14 et la table de transition d'états de la SRC est tel que représentée dans le Tableau 21.

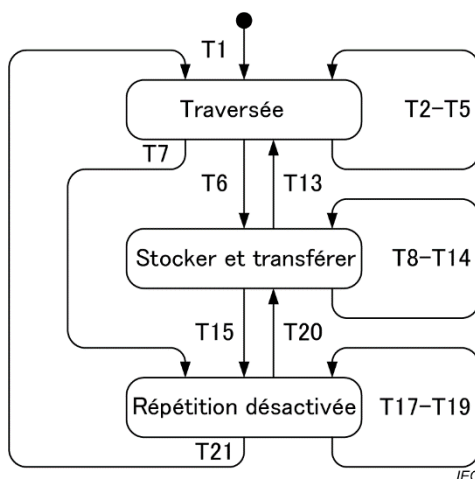


Figure 14 – Schéma de transition d'états de la SRC

Tableau 21 – Table de transition d'états de la SRC

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
1	Tous	Mise sous tension / néant	Traversée
2	Traversée	SRC_Send_Frame.req(Port_sel, DLSDU) / RT_FW_Send_Frame.ind(Port_sel, DLSDU)	Traversée
3	Traversée	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "OWN") && (CHECK_LT(DLSDU) = "RTPDU")] / RT_FW_Send_Frame.ind(Sed_sts, DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Traversée
4	Traversée	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "COM") && (CHECK_FCMD(DLSDU) = "SYNC")] / RT_FW_Send_Frame.ind(Port_sel, DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU); COR_SYNC_TIMER(V(Trcv), V(TS_fsync));	Traversée
5	Traversée	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, DA, SA, DLSDU) [!((CHECK_ADDR(DA) = "OWNDOM") (CHECK_LT(DLSDU) = "RT_PDU")) && !((CHECK_ADDR(DA) = "COM") && (CHECK_FCMD(DLSDU) = "SYNC"))] / RT_FW_Send_Frame.ind(Port_sel, DLSDU);	Traversée
6	Traversée	SRC_Repeat_Sel.req(SandF, SandF) / RT_FW_Repeat_Sel.req(SandF, SandF);	Stocker et transférer
7	Traversée	SRC_Repeat_Sel.req(OFF, OFF) / RT_FW_Repeat_Sel.req(OFF, OFF);	Répétition désactivée
8	Stocker et transférer	SRC_Send_Frame.req(Port_sel, DLSDU) / RT_FW_Send_Frame.ind(Port_sel, DLSDU);	Stocker et transférer
9	Stocker et transférer	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "BC") (CHECK_ADDR_TABLE(DA) = "True")] / STORE_SANDF(DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Stocker et transférer
10	Stocker et transférer	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [CHECK_ADDR_TABLE(DA) = "True"] / SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Stocker et transférer
11	Stocker et transférer	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [CHECK_ADDR_TABLE(SA) = "True"] / STORE_SANDF(DLSDU); SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU);	Stocker et transférer
12	Stocker et transférer	RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) <> "BC") && (CHECK_ADDR_TABLE(DA) <> "True") && (CHECK_ADDR_TABLE(SA) <> "True")] / STORE_SANDF(DLSDU);	Stocker et transférer
13	Stocker et transférer	REPEAT_STS(SandF) = "True" [CHECK_SANDF() <> 0] / while((CHECK_SANDF() <> 0) && (CMP_RTIM(T(Tcycle), (V(Tim) + CALC_SANDF())) > 0) { GET_SANDF(SedSDU); RT_FW_Send_Frame.ind(Sed_sts, SedSDU); }	Stocker et transférer

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
14	Stocker et transférer	<pre> REPEAT_STS(SandF) = "True" [CHECK_IP_PKT() <> 0] / while((CHECK_IP_PKT() <> 0) && (CMP_RTIM(T(Tcycle), (V(Tim) + CALC_IP_PKT()) > 0)) { GET_IP_PKT(DLSDU); RT_FW_Send_Frame.ind(Sed_sts, DLSDU); } </pre>	Stocker et transférer
15	Stocker et transférer	<pre> SRC_Repeat_Sel.req(CandT, CandT) / RT_FW_Repeat_Sel.req(CandT, CandT); </pre>	Traversée
16	Stocker et transférer	<pre> SRC_Repeat_Sel.req(OFF, OFF) / RT_FW_Repeat_Sel.req(OFF, OFF); </pre>	Répétition désactivée
17	Répétition désactivée	<pre> SRC_Send_Frame.req(Port_sel, DLSDU) / RT_FW_Send_Frame.ind(Port_sel, DLSDU); </pre>	Répétition désactivée
18	Répétition désactivée	<pre> RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, Trcv, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "MC_dom") && (CHECK_LT(DLSDU) = "RT_PDU")] / SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU); </pre>	Répétition désactivée
19	Répétition désactivée	<pre> RT_FW_Recv_Frame.ind(Port_num, Rcv_sts, DA, SA, DLSDU) [(CHECK_ADDR(DA) = "COM") && (CHECK_FCMD() = "SYNC")] / SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU); COR_SYNC_TIMER(V(Trcv), V(TS_fsync)); </pre>	Répétition désactivée
20	Répétition désactivée	<pre> SRC_Repeat_Sel.req(SandF, SandF) / RT_FW_Repeat_Sel.req(SandF, SandF); </pre>	Stocker et transférer
21	Répétition désactivée	<pre> SRC_Repeat_Sel.req(CandT, CandT) / RT_FW_Repeat_Sel.req(CandT, CandT); </pre>	Traversée

8.2.5.2.2 Liste des macros de la SRC

Le Tableau 22 indique la liste des macros utilisées pour la transition d'états de la SRC.

Tableau 22 – Liste des macros de la SRC

Nom de la fonction	Paramètre		Valeur de retour	Action
	Entrée	Sortie		
CHACK_ADDR	Addr	Néant	Résultat	Renvoyer COM lorsque Addr est l'adresse de multidiffusion commune, renvoyer MC lorsqu'il s'agit de l'adresse de multidiffusion au nœud voisin, renvoyer OWN lorsqu'il s'agit de l'adresse de multidiffusion pour son propre domaine de commande, renvoyer DOM lorsqu'il s'agit de l'adresse de multidiffusion pour un autre domaine de commande et renvoyer OTH dans d'autres cas.
CHACK_LT	DLSDU	Néant	Résultat	Vérifier le LT de la DLSDU et renvoyer la RTPDU pour RT-PDU et renvoyer OTH dans d'autres cas.
CHECK_FCMD	DLSDU	Néant	Résultat	Vérifier _FCMD Code de la DLSDU, et renvoyer le code d'instruction pour tout _PrimaryFCMD et renvoyer OTH dans d'autres cas.
COR_SYNC_TIME	V(Trcv), V(TS_fsync)	Néant	Néant	Corriger le temporisateur de synchronisation en utilisant Trcv et TS_fsync.
CHECK_ADDR_TABLE	Addr	Néant	Résultat	Vérifier si Addr est enregistré dans la table d'adresses. S'il est enregistré, renvoyer "True" S'il n'est pas enregistré, renvoyer "False"
STORE_SANDF	DLSDU	Néant	Néant	Stocker la DLSDU dans SANDF.
CHECK_SANDF	Néant	Néant	Résultat	Renvoyer le nombre de SDU stockées dans SANDF.
CALC_SANDF	Néant	Néant	Résultat	Calculer et renvoyer le temps exigé pour transmettre la première SDU stockée dans SANDF.
GET_SANDF	Néant	DLSDU	Néant	Renvoyer la première DLSDU stockée dans SANDF.
CMP_RTIM	Tim_ID1, Tim_ID2	Néant	Résultat	Comparer la valeur actuelle de Tim_ID1 et Tim_ID2. Ensuite, si le temps de Tim_ID1 jusqu'à la temporisation est égal ou supérieur à Tim_ID2, stocker 1 en tant que résultat.
CHECK_IP_PKT	Néant	Néant	Résultat	Renvoyer le nombre de SRCSDU stockées dans IP_PKT.
CALC_IP_PKT	Néant	Néant	Résultat	Calculer et renvoyer le temps exigé pour transmettre la première SRCSDU stockée dans IP_PKT.
GET_IP_PKT	Néant	DLSDU	Néant	Renvoyer la première DLSDU stockée dans IP_PKT.

8.2.5.2.3 Liste des fonctions de la SRC

Le Tableau 23 indique les fonctions utilisées pour la SRC et les descriptions.

Tableau 23 – Liste des fonctions de la SRC

Nom	Signification
RT_FW_Send_Frame	Envoyer une trame en spécifiant le numéro de port à envoyer. Plusieurs numéros de port peuvent être spécifiés.
RT_FW_Recv_Frame	Renvoyer le numéro de port reçu, le statut, l'heure reçue, l'adresse de destination, l'adresse source et la trame reçue lors de la réception de la trame.
RT_FW_Repeat_Sel	Choisir la méthode de transfert de trame reçue pour chaque port.

8.2.5.2.4 Interface SRC/CTC

8.2.5.2.4.1 Vue d'ensemble

La SRC fournit les services suivants à la CTC.

- Service d'envoi de (SRC_Send_Frame);
- Service de réception de trame (SRC_Recv_Frame);
- Service de répétition de la sélection (SRC_Repeat_Sel.req).

8.2.5.2.4.2 Définitions détaillées des primitives et paramètres de service

Le Tableau 24 indique les primitives et les paramètres qui sont échangés entre la SRC et la CTC. Les détails de ces paramètres sont présentés dans les paragraphes suivants.

Tableau 24 – Primitives et paramètres pour l'interface SRC-CTC

Primitives	Source	Paramètre
SRC_Send_Frame.req	CTC	Port_sel, DLSDU
SRC_Send_Frame.cnf	SRC	Résultat
SRC_Recv_Frame.ind	SRC	Rcv_sts, Length, DLSDU

a) SRC_Send_Frame

Le Tableau 25 indique les primitives et les paramètres du service d'envoi de trame.

Tableau 25 – Primitives et paramètres d'envoi de trame

SRC_Send_Frame Nom de paramètre	Demande	Confirmation
	Entrée	Sortie
Port_sel	O	
DLSDU	O	
Résultat		O

b) SRC_Recv_Frame

Le Tableau 26 indique les primitives et les paramètres du service de réception de trame.

Tableau 26 – Primitives et paramètres de réception de trame

SRC_Recv_Frame Nom de paramètre	Indication
	Sortie
Rcv_Sts	O
Length	O
DLSDU	O

c) SRC_Repeat_Sel

Tableau 27 – Primitives et paramètres du service de répétition de la sélection

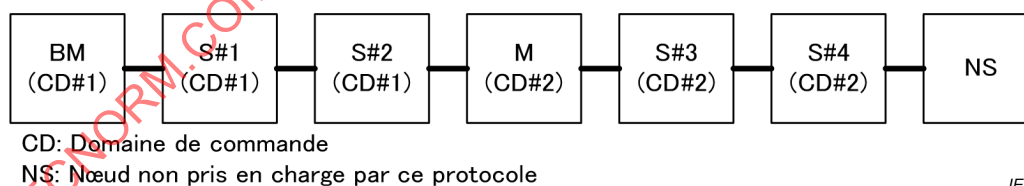
SRC_Repeat_Sel Nom de paramètre	Demande
	Entrée
Select_port1	O
Select_port2	O

8.3 Machine de protocole de gestion d'appareil de terrain (FSM PM)

8.3.1 Vue d'ensemble

Le protocole d'interface entre la FSMUL et la FSM est décrit dans ce paragraphe. La FSMUL fournit des services à la FSM pour les stations d'initialisation de réseau.

Dans le présent paragraphe, la question de mise en œuvre et les questions individuelles sont intentionnellement exclues. La Figure 15 représente un exemple de configuration de réseau. Dans cet exemple de configuration de réseau, l'appareil principal ayant la fonction appareil principal de bande (BM) (le 1er appareil principal), l'appareil subordonné N° 1 (S N° 1), l'appareil subordonné N° 2 (S N° 2), l'appareil principal sans la fonction appareil principal de bande (M), (le 2e appareil principal), l'appareil subordonné N° 3 (S N° 3), l'appareil subordonné N° 4 (S N° 4) et un nœud non pris en charge par ce protocole (NS) sont connectés en série à partir d'une extrémité. Alors que BM, S N° 1 et S N° 2 appartiennent au 1er domaine de commande, M, S N° 3 et S N° 4 appartiennent au 2e domaine de commande.



IEC

Figure 15 – Exemple de configuration de réseau

La Figure 16 représente les procédures d'initialisation du réseau lors du démarrage du système. Comme représenté à la Figure 16, certaines des procédures peuvent être ignorées.

L'initialisation du réseau avant le démarrage de la communication cyclique doit être mise en œuvre par l'appareil principal ayant la fonction d'appareil principal de bande (BM). L'appareil principal ayant la fonction d'appareil principal de bande commande l'état de connexion de tous les nœuds du réseau. Par conséquent, l'appareil principal ayant la fonction d'appareil principal de bande a besoin soit d'enregistrer les informations d'appareil de connexion avant le démarrage du système, soit de collecter les informations d'appareil de connexion en détectant les nœuds de connexion. En utilisant l'exemple de configuration de réseau dans la Figure 15, la séquence avec chaque procédure dans la Figure 16 est telle que décrite ci-dessous. Le cas avec BM comme appareil principal de synchronisation (SM) est tel que décrit en 8.3.3.1.1 et le cas avec l'appareil principal autre que le BM comme appareil principal de synchronisation (SM) est tel que décrit en 8.3.3.1.2. Il convient de noter que tous les nœuds connectés au réseau doivent être transférés par le mode stocker et transférer après la mise sous tension (P-ON) jusqu'à la "commutation du mode de transfert" à la Figure 16.

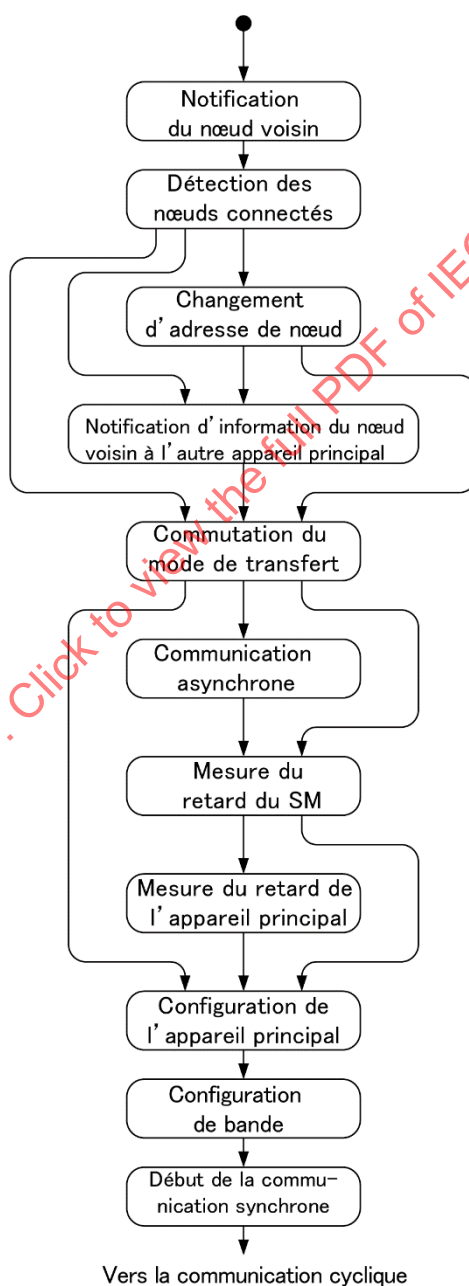


Figure 16 – Exemple de configuration de réseau

8.3.2 Reconnaissance et configuration de base

8.3.2.1 Notification de nœud connecté

La Figure 17 représente la séquence de notification du nœud voisin à exécuter lors de la mise sous tension des nœuds du réseau.

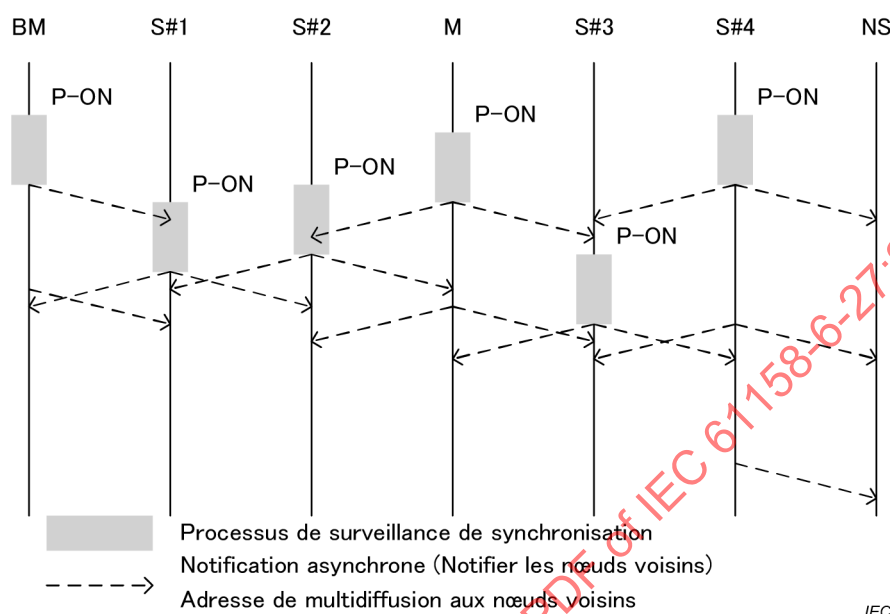


Figure 17 – Séquence de notification du nœud voisin

Tous les nœuds pris en charge par ce protocole doivent surveiller l'existence/l'absence de paquet de synchronisation pendant une certaine durée après la mise sous tension (P-ON). Le temps de surveillance (recommandé) doit être de 500 ms. Dans les cas où un paquet de synchronisation n'est pas détecté, une notification asynchrone [type d'événement: notification de nœud voisin] doit être multidiffusée à partir de tous les ports] (voir 4.3.4.10). Dans ce cas, l'adresse de multidiffusion du nœud voisin doit être spécifiée comme destination (voir Tableau 4). Dans les cas où un paquet de synchronisation est détecté, l'état passe à "attendre le début de la communication cyclique" sans envoyer de notification asynchrone [type d'événement: notification du nœud voisin] (voir 8.3.5 pour les étapes suivantes).

Ce processus pour notifier les nœuds voisins doit être exécuté pour chaque port des nœuds. Dans les cas où un port ne reçoit pas de notification de nœud voisin de la part des autres nœuds, la notification de nœud voisin doit être envoyée continuellement au port concerné à certains intervalles. Les intervalles (recommandés) d'envoi continu doivent être inférieurs ou égaux à 200 ms.

Dans les cas où un port reçoit la notification de nœud voisin d'un autre port, l'envoi de la notification de nœud voisin vers le port doit être arrêté et "l'adresse MAC source" de la notification de nœud voisin reçue doit être mémorisée en tant qu'information de nœud voisin pour le port.

8.3.2.2 Notification de nœud connecté

Après un certain temps à partir de la notification du nœud voisin décrite dans le paragraphe précédent, le BM envoie une notification asynchrone [type d'événement: notification de démarrage] à d'autres nœuds (voir 4.3.4.10) en utilisant l'adresse de multidiffusion commune (voir Figure 18). Il est recommandé de fixer le temps écoulé entre l'exécution de la notification de nœud voisin décrite dans le paragraphe précédent et l'envoi de la notification de démarrage, de sorte que tous les nœuds puissent envoyer la notification de nœud voisin en tenant compte de l'anomalie dans le cadencement de démarrage pour tous les nœuds. Comme ce temps dépend du système, il ne doit pas être spécifié dans le présent document.

Les autres nœuds qui ont reçu la notification de démarrage du BM doivent mémoriser l'adresse source (adresse de nœud de BM) et le port reçu comme port en amont. Comme décrit en tant que S N° 4 à la Figure 18, dans les cas où la notification de nœud voisin est envoyée continuellement de l'étape précédente au nœud non pris en charge par le présent protocole, l'envoi de la notification de nœud voisin doit être arrêté lorsque la notification de démarrage est reçue et les informations de nœud voisin pour son propre nœud doivent être confirmées. Ensuite, un paquet spécifié par ce protocole ne doit pas être envoyé au port auquel est connecté un nœud non pris en charge par ce protocole.

Chaque nœud (autre que le BM) qui a reçu la notification de démarrage doit renvoyer une notification asynchrone [type d'événement: réponse de notification de démarrage] au BM (voir 4.3.4.10). Dans l'étape précédente, l'intervalle entre la réception de la notification du nœud voisin d'un autre nœud et l'envoi de la notification de démarrage côté BM doit être de 1 s (recommandé). De plus, le cadencement pour chaque nœud pour émettre une réponse après la réception de la notification de démarrage doit se situer dans les 100 μ s à partir de la réception. Les paramètres de notification de démarrage (type de fonctionnement, fanion pris en charge par le SM, fanion de retard de l'appareil principal terminé (uniquement pour l'appareil principal), temps de retard du SM et informations relatives au nœud voisin) doivent être mis à la réponse de notification de démarrage (voir 4.3.4.10). Le BM doit mémoriser l'adresse source (adresse de nœud de chaque nœud) stockée dans la notification de démarrage et les paramètres de notification de démarrage.

Le BM doit attendre l'arrivée de la notification de démarrage de chaque nœud après l'envoi de la notification de démarrage pour le temps d'attente de détection. Ce temps d'attente de détection doit être de 2 ms (recommandé).

Dans les cas où le BM possède à l'avance les paramètres de notification de démarrage de tous les nœuds, un mécanisme de vérification avec les paramètres de notification de démarrage obtenus à partir de la réponse de notification de démarrage est mis en œuvre.

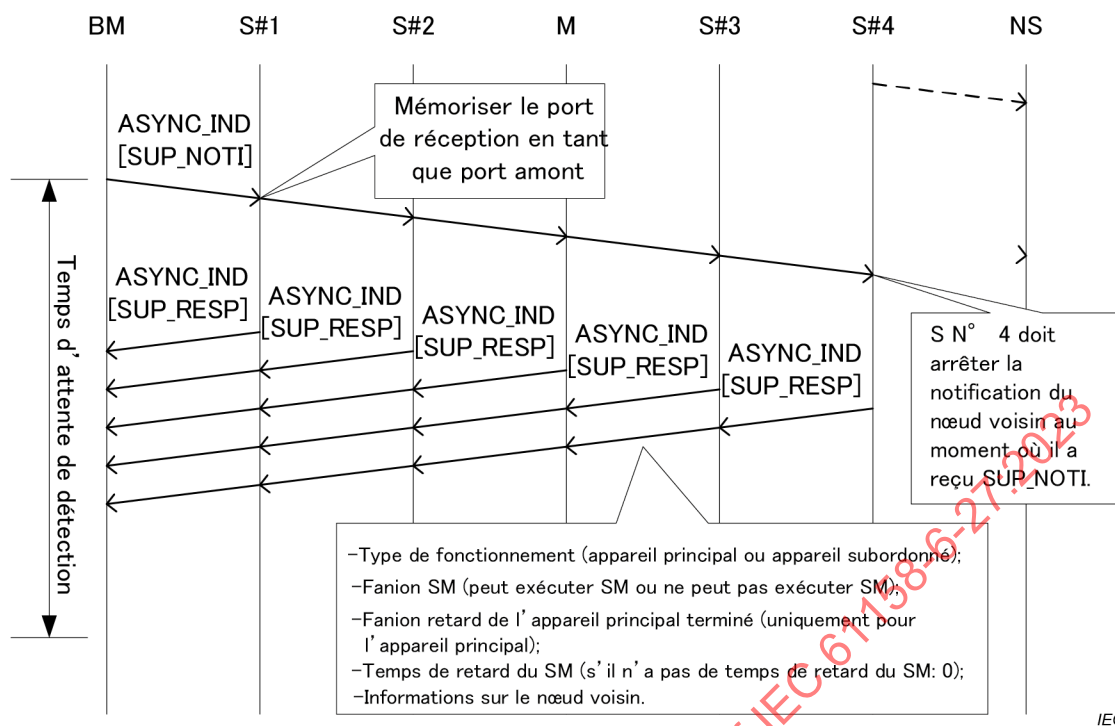


Figure 18 – Séquence de détection de nœud connecté

Les informations relatives au nœud voisin doivent être une liste des informations obtenues dans la séquence de notification du nœud connecté à partir du paquet de notification du nœud voisin (adresse MAC) pour chaque port. Un exemple est représenté à la Figure 19. Dans les cas où le commutateur existe entre les nœuds comme représenté à la Figure 19, plusieurs informations (adresse MAC) existent dans un port.

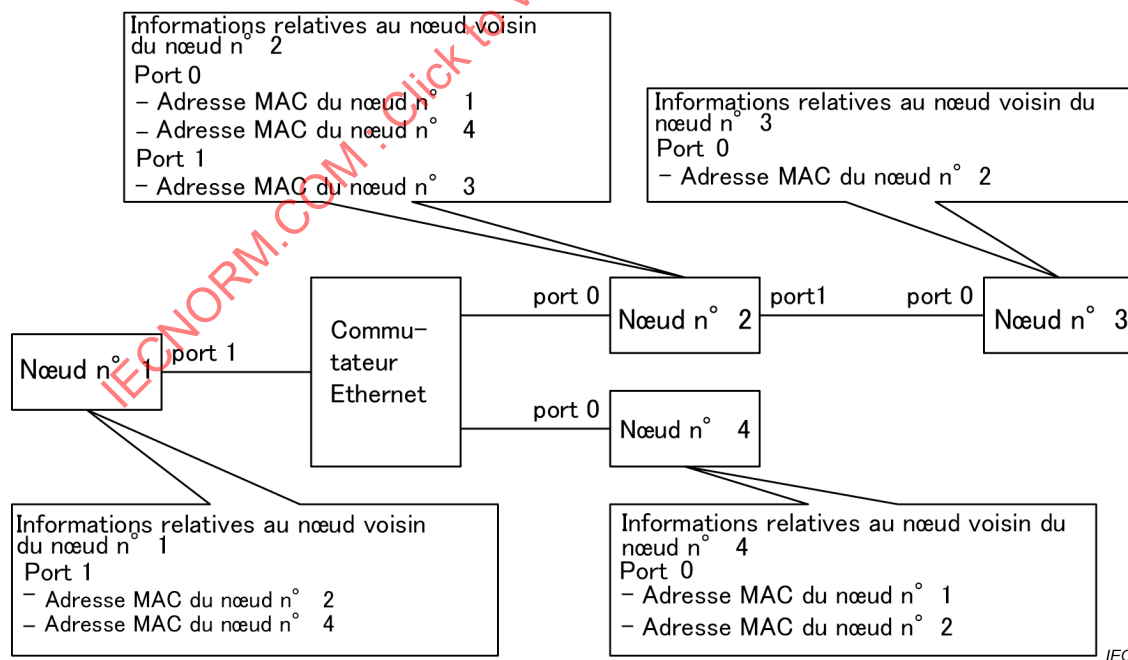


Figure 19 – Exemple d'informations relatives au nœud voisin

8.3.3 Démarrage du système

8.3.3.1 Machine de protocole de l'appareil principal

8.3.3.1.1 Appareil principal avec SM

Comme pour l'appareil principal avec fonction SM lors de l'initialisation du réseau, la Figure 20 représente le schéma de transition d'états FSMUL et le Tableau 28 la table de transition d'états FSMUL.

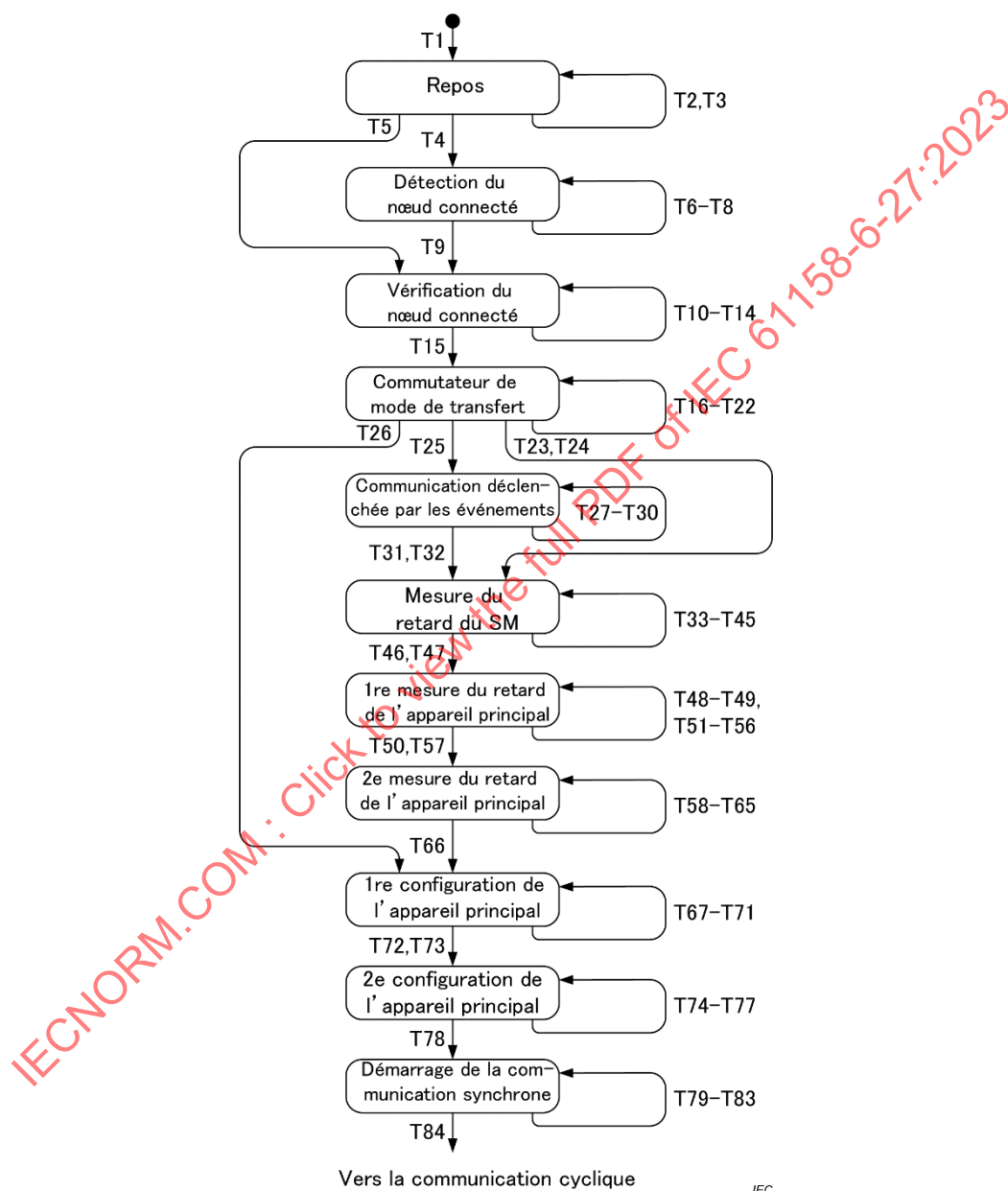


Figure 20 – Schéma de transition d'états FSMUL pour l'appareil principal avec fonction SM

Tableau 28 – Table de transition d'états FSMUL pour l'appareil principal avec fonction SM

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
1	Tous	Mise sous tension ou FSMUL-RESET.req / FSMUL-RESET.cnf; CTC_Reset.req; SRC_Reset.req;	Repos
2	Repos	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Repos
3	Repos	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Repos
4	Repos	FSMUL-DETECT.req / = BUILD_PDU_ASYNC_IND(V(Nmul_add), SUP_NOTI); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; START_TIMER(T(Twait), V(Tset_up));	Détection du nœud connecté
5	Repos	FSMUL-RECOG.req / V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	Vérification du nœud connecté
6	Détection du nœud connecté	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) / Result = CHECK_PDU_FCMD(DLSDU); if (Result == FCMD_ASYNC_IND) && (DLSDU.EventType == SUP_RESP) { SET_NODE_INFO(V(Niomap), MAC_Address, DLSDU.SMA); SET_NODE_INFO(V(Niomap), Node_Address, DLSDU.SA); V(Niomap)++; }	Détection du nœud connecté
7	Détection du nœud connecté	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Détection du nœud connecté
8	Détection du nœud connecté	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Détection du nœud connecté
9	Détection du nœud connecté	EXPIRED_TIMER(T(Twait)) = "True" / V(Nmax_nodes) = V(Niomap); V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); FSMUL-DETECT.cnf; DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Vérification du nœud connecté

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
10	Vérification du nœud connecté	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) < V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Nstatus)); V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	Vérification du nœud connecté
11	Vérification du nœud connecté	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) < V(Nmax_nodes)] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Vérification du nœud connecté
12	Vérification du nœud connecté	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) >= V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Nstatus)) } } </pre>	Vérification du nœud connecté
13	Vérification du nœud connecté	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Vérification du nœud connecté
14	Vérification du nœud connecté	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Vérification du nœud connecté
15	Vérification du nœud connecté	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) >= V(Nmax_nodes)] / FSMUL-RECOG.cnf; DLSDU = BUILD_PDU_CDO_WRITE(V(Nmulti_add), Req, trans_mode, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; </pre>	Commutateur de mode de transfert
16	Commutateur de mode de transfert	<pre> SRC_Send_Frame.cnf() [V(Niomap) == 0] / V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, trans_mode, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Commutateur de mode de transfert

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
17	Commutateur de mode de transfert	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) < V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Trans_Mode, V(Ntrans_mode)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Trans_Mode, V(Ntrans_mode)); V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), V(Niomap), Node_Address) DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, trans_mode, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	Commutateur de mode de transfert
18	Commutateur de mode de transfert	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) < V(Nmax_nodes)] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, trans_mode, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Commutateur de mode de transfert
19	Commutateur de mode de transfert	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niomap) >= V(Nmax_nodes)] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Ntrans_mode)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), Status, V(Ntrans_mode)); } Result = CHECK_TRANS_MODE(); if(Result == "OK") { FSMUL-SW-TRANS-MODE.cnf; } DLSDU = BUILD_PDU_CDO_WRITE(V(Nmulti_add), Req, Trans_Mode, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; } </pre>	Commutateur de mode de transfert
20	Commutateur de mode de transfert	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) >= V(Nmax_nodes)] / DLSDU = BUILD_PDU_CDO_WRITE(V(Nmulti_add), Req, Trans_Mode, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; </pre>	Commutateur de mode de transfert
21	Commutateur de mode de transfert	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Commutateur de mode de transfert
22	Commutateur de mode de transfert	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Commutateur de mode de transfert
23	Commutateur de mode de transfert	<pre> FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "OWN"] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; START_TIMER(T(Twait), V(Tresp)); </pre>	Mesure du retard SM (SM de son propre nœud)

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
24	Commutateur de mode de transfert	FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "SLV"] / DLSDU = BUILD_PDU_ASYNC_IND(V(Nsm_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 'FF'H; START_TIMER(T(Twait), V(Tresp));	Mesure du retard SM (SM d'un autre nœud)
25	Commutateur de mode de transfert	FSMUL-ASYNC.req / néant	Communication déclenchée par les événements
26	Commutateur de mode de transfert	FSMUL-CONFIG.req / néant	1re configuration de l'appareil principal
27	Communication déclenchée par les événements	CTC-SDN.req(SAP_ID, Node_ID, Length, SDU) / DLSDU = BUILD_PDU_EVD(Node_ID, Length, SDU); Len = GET_LEN(DLSDU); SRC_Send_Frame.req(Port_sel, Node_ID, Len, DLSDU); CTC-SDN.cnf("OK");	Communication déclenchée par les événements
28	Communication déclenchée par les événements	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Node_ID = GET_DA(DLSDU); Len = GET_LEN(DLSDU); DLSDU = GET_DATA(DLSDU); CTC-SDN.ind(SAP_ID, Node_ID, Len, SDU);	Communication déclenchée par les événements
29	Communication déclenchée par les événements	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Communication déclenchée par les événements
30	Communication déclenchée par les événements	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Communication déclenchée par les événements
31	Communication déclenchée par les événements	FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "OWN"] / SDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 0; START_TIMER(T(Twait), V(Tresp));	Mesure du retard SM (SM de son propre nœud)
32	Communication déclenchée par les événements	FSMUL-MEAS-DELAY.req [CHECK_SYNC_RIGHT() = "SLV"] / DLSDU = BUILD_PDU_ASYNC_IND(V(Nsm_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap) = 'FF'H; START_TIMER(T(Twait), V(Tresp));	Mesure du retard SM (SM d'un autre nœud)
33	Mesure du retard SM (SM de son propre nœud)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) <= V(Nmax_nodes)] / V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_REQ_SEND(V(Nnode_add), MEAS_DLY, V(Nmy_add), V(Nmy_mac_add)); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	Mesure du retard SM (SM de son propre nœud)
34	Mesure du retard SM (SM de son propre nœud)	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == REQ) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), None); GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Resp, V(Treq_s), V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); V(Niomap)++; }	Mesure du retard SM (SM de son propre nœud)

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
35	Mesure du retard SM (SM de son propre nœud)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) > V(Nmax_nodes)] / V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Mesure du retard du SM
36	Mesure du retard SM (SM d'un autre nœud)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) = 'FF'H] / DLSDU = BUILD_PDU_ASYNC_IND(V(Nsm_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	Mesure du retard SM (SM d'un autre nœud)
37	Mesure du retard SM (SM d'un autre nœud)	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Niomap) = 'FF'H)] / Result = CHECK_PDU_FCMD(DLSDU) if(Result == FCMD_SYNC) { GET_SYNC_DATA(DLSDU, V(Tsyn_s)); GET_RECV_TIME(V(Tsyn_r)); V(Niomap) = 0; DLSDU = BUILD_PDU_MEAS_DLY(V(Nsm_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); }	Mesure du retard SM (SM d'un autre nœud)
38	Mesure du retard SM (SM d'un autre nœud)	EXPIRED_TIMER(T(Twait)) = "True" [V(Niomap) = 0] / DLSDU = BUILD_PDU_MEAS_DLY(V(Nmy_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	Mesure du retard SM (SM d'un autre nœud)
39	Mesure du retard SM (SM d'un autre nœud)	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && V(Niomap) = 0] / Result = CHECK_PDU_FCMD(DLSDU) && if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == RESP) (DLSDU.SA == V(Nsm_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = ((Treq_r - Tsyn_s) - (Treq_s - Tsyn_r)) / 2; SET_NODE_INFO(V(Nmst), Delay, V(Ndelay)); V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_REQ_SEND(V(Nnode_add), MEAS_DLY, V(Nmy_add), V(Nmy_mac_add)); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Treq_intvl)); }	Mesure du retard SM (SM d'un autre nœud)
40	Mesure du retard SM (SM d'un autre nœud)	EXPIRED_TIMER(T(Twait)) = "True" [V(Nmax_nodes) > V(Niomap) > 0] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_REQ_SEND(V(Nnode_add), MEAS_DLY, V(Nmy_add), V(Nmy_mac_add)); SRC_Send_Frame.req(Port_sel, DLSDU); if(V(Niomap) <> V(Nmax_nodes)) { START_TIMER(T(Twait), V(Treq_intvl)); }	Mesure du retard SM (SM d'un autre nœud)

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
41	Mesure du retard SM (SM d'un autre nœud)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Niormap) >= V(Nmax_nodes))] / Result = CHECK_PDU_FCMD(DLSDU) if((Result == FCMD_ASYNC_IND) && (DLSDU.EventType == MEAS_DLY)) { V(Niormap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts)); } </pre>	Mesure du retard du SM
42	Mesure du retard du SM	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Niormap) < V(Nmax_nodes)] / Result := CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Status, V(Nstatus)); V(Niormap)++; V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	Mesure du retard du SM
43	Mesure du retard du SM	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niormap) < V(Nmax_nodes)] / DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	Mesure du retard du SM
44	Mesure du retard du SM	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Mesure du retard du SM
45	Mesure du retard du SM	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Mesure du retard du SM
46	Mesure du retard du SM	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(V(Niormap) >= V(Nmax_nodes)) && (CHECK_SYNC_RIGHT() = "OWN")] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Status, V(Nstatus)); V(Ndmn1) = 1; V(Niormap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, delay_time, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	1re mesure du retard de l'appareil principal (SM de son propre nœud)

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
47	Mesure du retard du SM	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(V(Niormap) >= V(Nmax_nodes)) && (CHECK_SYNC_RIGHT() = "SLV")] / Result:= CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Status, V(Nstatus)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Status, V(Nstatus)); DLSDU = BUILD_PDU_REQ_SEND(V(Nsm_add), SYNC, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Ndmn1) = 'FF'H; START_TIMER(T(Twait), V(Tresp)); } } </pre>	1re mesure du retard de l'appareil principal (SM d'un autre nœud)
48	1re mesure du retard de l'appareil principal (SM de son propre nœud)	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Ndmn1) < V(Nmax_dmn1)] / Result = CHECK_PDU_FCMD(DLSDU) if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Delay, V(Ndelay)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Delay, V(Ndelay)); V(Ndmn1)++; V(Niormap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, delay_time, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } } </pre>	1re mesure du retard de l'appareil principal (SM de son propre nœud)
49	1re mesure du retard de l'appareil principal (SM de son propre nœud)	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niormap) < V(Nmax_nodes)] / DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, status, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1re mesure du retard de l'appareil principal (SM de son propre nœud)
50	1re mesure du retard de l'appareil principal (SM de son propre nœud)	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [V(Ndmn1) >= V(Nmax_dmn1)] / Result = CHECK_PDU_FCMD(DLSDU) if((Result == FCMD_READ_CDO) && (DLSDU.HS == ACK) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_READ_DATA(DLSDU, 1, Delay, V(Ndelay)); if(Result == "OK") { SET_NODE_INFO(V(Niormap), Delay, V(Ndelay)); V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); } } </pre>	2e mesure du retard de l'appareil principal
51	1re mesure du retard de l'appareil principal (SM d'un autre nœud)	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Niormap) = 'FF'H] / DLSDU = BUILD_PDU_REQ_SEND(V(Nsm_add), SYNC, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1re mesure du retard de l'appareil principal (SM d'un autre nœud)

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
52	1re mesure du retard de l'appareil principal (SM d'un autre nœud)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Ndmn1) = 'FF'H)] / Result = CHECK_PDU_FCMD(DLSDU); if(Result == FCMD_SYNC) { RESTART_TIMER(Tcycle); V(Ndmn1) = 1; V(Niormap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } </pre>	2e mesure du retard de l'appareil principal (SM d'un autre nœud)
53	1re mesure du retard de l'appareil principal (SM d'un autre nœud)	<pre> EXPIRED_TIMER(T(Twait)) = "True" [V(Ndmn1) < V(Nmax_dmn1)] / DLSDU = BUILD_PDU_MEAS_DLY(V(Niormap), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	2e mesure du retard de l'appareil principal (SM d'un autre nœud)
54	1re mesure du retard de l'appareil principal (SM d'un autre nœud)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Ndmn1) < V(Nmax_dmn1))] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == RESP) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = (Treq_r – Treq_s) / 2; SET_NODE_INFO(V(Nmst), Delay, V(Ndelay)); V(Ndmn1)++; V(Niormap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn1)); V(Nnode_add) = GET_NODE_INFO(V(Niormap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); } </pre>	2e mesure du retard de l'appareil principal (SM d'un autre nœud)
55	1re mesure du retard de l'appareil principal	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	1re mesure du retard de l'appareil principal
56	1re mesure du retard de l'appareil principal	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	1re mesure du retard de l'appareil principal
57	1re mesure du retard de l'appareil principal (SM d'un autre nœud)	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (V(Ndmn1) >= V(Nmax_dmn1))] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == RESP) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = (Treq_r – Treq_s) / 2; SET_NODE_INFO(V(Nmst), Delay, V(Ndelay)); V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	2e mesure du retard de l'appareil principal

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
58	2e mesure du retard de l'appareil principal	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_WRITE_CDO) && (DLSDU.HS == REQ) && (DLSDU.SA == V(Nnode_add))) { Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if((Result == "OK") && (CDO_ID == mst2_uport)) { SET_CDO_DATA(CDO_ID, Port_num); Result = 0; DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Ack, CDO_ID, Result); SRC_Send_Frame.req(Port_sel, DLSDU); } } </pre>	2e mesure du retard de l'appareil principal
59	2e mesure du retard de l'appareil principal	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_ASYNC_IND) && (DLSDU.EventType == MEAS_DLY) && (DLSDU.SA == V(Nnode_add))) { DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	2e mesure du retard de l'appareil principal
60	2e mesure du retard de l'appareil principal	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU) && (DLSDU.SA == V(Nnode_add)); if(Result == FCMD_SYNC) { RESTART_TIMER(Tcycle) } </pre>	2e mesure du retard de l'appareil principal
61	2e mesure du retard de l'appareil principal	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Result = CHECK_PDU_FCMD(DLSDU); if((Result == FCMD_MEAS_DLY) && (DLSDU.HS == REQ) && (DLSDU.SA == V(Nnode_add))) { GET_DELAY_DATA(DLSDU, V(Treq_s), None); GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Resp, V(Treq_s), V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	2e mesure du retard de l'appareil principal
62	2e mesure du retard de l'appareil principal	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (Result = FCMD_TOKEN) && (DLSDU.TokenType == MST_MEAS_DLY) && (DLSDU.SA == V(Nnode_add)) && (V(Nmst2_index) < V(Nmst2_index_max))] / V(Nmst2_index)++; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	2e mesure du retard de l'appareil principal
63	2e mesure du retard de l'appareil principal	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType == MST_MEAS_DLY) && (DLSDU.SA == V(Nnode_add)) && (V(Nmst2_index) >= V(Nmst2_index_max))] / FSMUL-MEAS-DELAY.cnf; </pre>	2e mesure du retard de l'appareil principal
64	2e mesure du retard de l'appareil principal	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	2e mesure du retard de l'appareil principal

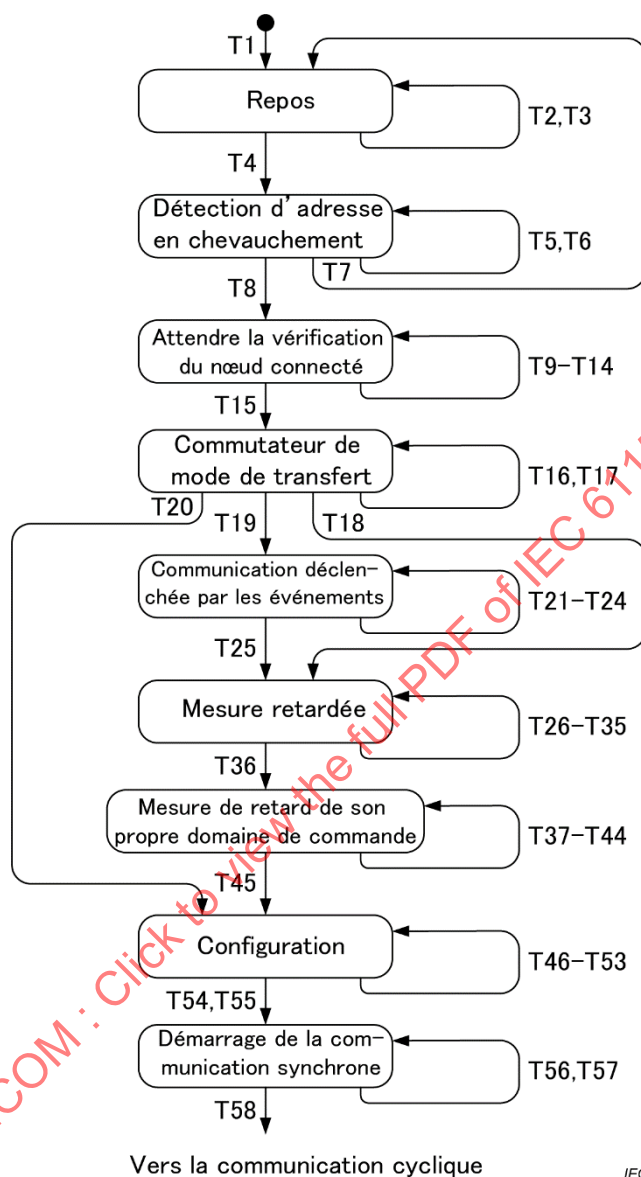
N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
65	2e mesure du retard de l'appareil principal	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	2e mesure du retard de l'appareil principal
66	2e mesure du retard de l'appareil principal	FSMUL-CONFIG.req / V(Ncnf_index) = 0; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	1re configuration de l'appareil principal
67	1re configuration de l'appareil principal	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / Result = GET_CDO_READ_DATA(DLSDU, 1, V(Ncdo_id), V(Ncdo_data)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), V(Ncdo_id), V(Ncdo_data)); V(Ncnf_index)++; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = SET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	1re configuration de l'appareil principal
68	1re configuration de l'appareil principal	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / V(Ncnf_index)++; GET_CONFIG_CDO(V(Ncnf_index), V(Ncdo_id), V(Ncdo_data), V(Nrw), V(Niomap)); V(Nnode_add) = SET_NODE_INFO(V(Niomap), Node_Address); if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp));	1re configuration de l'appareil principal

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
69	1re configuration de l'appareil principal	<pre> EXPIRED_TIMER(T(Twait)) = "True" / if(V(Nrw) == READ) { DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, V(Ncdo_id), none, none); } else { DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, V(Ncdo_id), V(Ncdo_data)); } SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp)); </pre>	1re configuration de l'appareil principal
70	1re configuration de l'appareil principal	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	1re configuration de l'appareil principal
71	1re configuration de l'appareil principal	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	1re configuration de l'appareil principal
72	1re configuration de l'appareil principal	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) >= V(Ncnf_index_max))] / Result = GET_CDO_READ_DATA(DLSDU, 1, V(Ncdo_id), V(Ncdo_data)); if(Result == "OK") { SET_NODE_INFO(V(Niomap), V(Ncdo_id), V(Ncdo_data)); V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), CONFIG); SRC_Send_Frame.req(Port_sel, DLSDU); } </pre>	2e configuration de l'appareil principal
73	1re configuration de l'appareil principal	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ncnf_index) < V(Ncnf_index_max))] / V(Nmst2_index) = 1; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), CONFIG); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	2e configuration de l'appareil principal
74	2e configuration de l'appareil principal	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType = CONFIG) && (DLSDU.SA = V(Nnode_add)) && (V(Nmst2_index) < V(Nmst2_index_max))] / V(Nmst2_index)++; V(Nnode_add) = GET_MST2_NODE_ADD(V(Nmst2_num)); DLSDU = BUILD_PDU_TOKEN(V(Nnode_add), MST_MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	2e configuration de l'appareil principal
75	2e configuration de l'appareil principal	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType = MST_MEAS_DLY) && (DLSDU.SA = V(Nnode_add)) && (V(Nmst2_index) >= V(Nmst2_index_max))] / FSMUL-CONFIG.cnf; </pre>	2e configuration de l'appareil principal
76	2e configuration de l'appareil principal	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	2e configuration de l'appareil principal
77	2e configuration de l'appareil principal	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	2e configuration de l'appareil principal

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
78	2e configuration de l'appareil principal	FSMUL-START-SYNC.req / V(Niomap) = 0; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); if(V(Nnode_add) = V(Nsm_add)) { V(Nnode_add)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); } DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Démarrage de la communication synchrone
79	Démarrage de la communication synchrone	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = cyclic_start)) && (V(Niomap) < V(Nmax_nodes))] / V(Niomap)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); if(V(Nnode_add) == V(Nsm_add)) { V(Nnode_add)++; V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); } DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Démarrage de la communication synchrone
80	Démarrage de la communication synchrone	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = cyclic_start)) && (V(Niomap) >= V(Nmax_nodes))] / V(Nnode_add) = V(Nsm_add); DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Démarrage de la communication synchrone
81	Démarrage de la communication synchrone	EXPIRED_TIMER(T(Twait)) = "True" / DLSDU = BUILD_PDU_CDO_WRITE(V(Nnode_add), Req, cyclic_start, 1); SRC_Send_Frame.req(Port_sel, DLSDU); START_TIMER(T(Twait), V(Tresp_sts));	Démarrage de la communication synchrone
82	Démarrage de la communication synchrone	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Démarrage de la communication synchrone
83	Démarrage de la communication synchrone	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Démarrage de la communication synchrone
84	Démarrage de la communication synchrone	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nsm_add)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = cyclic_start))] / CTC_Start.req;	Vers la communication cyclique

8.3.3.1.2 Appareil principal sans SM

Comme pour l'appareil principal sans fonction SM lors de l'initialisation du réseau, la Figure 21 représente le schéma de transition d'états FSMUL et le Tableau 29 la table de transition d'états FSMUL.



IEC

Figure 21 – Schéma de transition d'états FSMUL pour l'appareil principal sans fonction SM

Tableau 29 – Table de transition d'états FSMUL pour l'appareil principal sans fonction SM

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
1	Tous	Mise sous tension ou FSMUL-RESET.req / FSMUL-RESET.cnf; CTC_Reset.req; SRC_Reset.req; SET_CDO_DATA(Status, IDLE);	Repos
2	Repos	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Repos
3	Repos	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Repos
4	Repos	FSMUL-START.req / RECEIVE_ALL_PKT(ON); START_TIMER(T(Twait), V(Tdt_same_add)); SET_CDO_DATA(Status, DT_ADD_ERR);	Détection d'adresse en chevauchement
5	Détection d'adresse en chevauchement	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Détection d'adresse en chevauchement
6	Détection d'adresse en chevauchement	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Détection d'adresse en chevauchement
7	Détection d'adresse en chevauchement	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (DLSDU.SA = V(Nmy_add))] / FSMUL-CLR-ERR.cnf(Address_error);	Détection d'adresse en chevauchement
8	Détection d'adresse en chevauchement	EXPIRED_TIMER(T(Twait)) = "True" / RECEIVE_ALL_PKT(OFF); SET_CDO_DATA(Status, WRECOG);	Attendre la vérification du nœud connecté
9	Attendre la vérification du nœud connecté	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_ASYNC_IND = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.EventType = SUP_NOTI)] / SET_CDO_DATA(Mst1_mac_add, DLSDU.SRC_MAC_ADD); SET_CDO_DATA(Mst1_node_add, DLSDU.SA); SET_CDO_DATA(Mst1_uport, Port_num); DLSDU = BUILD_PDU_ASYNC_IND(DLSDU.SA, SUP_RESP); SRC_Send_Frame.req(Port_num, DLSDU);	Attendre la vérification du nœud connecté
10	Attendre la vérification du nœud connecté	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) if(Result == "OK") { GET_CDO_DATA(CDO_ID, CDO_data) DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, CDO_ID, OK, CDO_data) SRC_Send_Frame.req(Port_num, DLSDU) if(CDO_ID == Status) { DLSDU = BUILD_PDU_CDO_WRITE(V(Nmul_next), REQ, Connect, 1); SRC_Send_Frame.req(Port_sel, DLSDU); } } }	Attendre la vérification du nœud connecté
11	Attendre la vérification du nœud connecté	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if((Result == "OK") && (CDO_ID == Connect)) { SET_CDO_DATA(GET_CDO_PORTID(Port_num), 1); }	Attendre la vérification du nœud connecté

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
12	Attendre la vérification du nœud connecté	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data); if(Result == "OK") { SET_CDO_DATA(CDO_ID, CDO_data); DLSDU = BUILD_PDU_CDO_WRITE(DLSDU.SA, ACK, CDO_ID, OK); SRC_Send_Frame.req(Port_num, DLSDU); if(CDO_ID == Connect) { SET_CDO_DATA(GET_CDO_PORTID(Port_num), 1); } } </pre>	Attendre la vérification du nœud connecté
13	Attendre la vérification du nœud connecté	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Attendre la vérification du nœud connecté
14	Attendre la vérification du nœud connecté	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Attendre la vérification du nœud connecté
15	Attendre la vérification du nœud connecté	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (CDO_ID = Trans_Mode)] / CHANGE_TRANS_MODE(); SET_CDO_DATA(Status, TRANS_MODE); </pre>	Commutateur de mode de transfert
16	Commutateur de mode de transfert	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Commutateur de mode de transfert
17	Commutateur de mode de transfert	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Commutateur de mode de transfert
18	Commutateur de mode de transfert	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode)] && FSMUL-MEAS-DELAY.req] / GET_CDO_DATA(Trans_Mode, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); V(Ndly_cnt) = 0; SET_CDO_DATA(Status, MEASURED_DLY); </pre>	Mesure différée
19	Commutateur de mode de transfert	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)] && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode)] && FSMUL-ASYNC.req] / GET_CDO_DATA(Trans_Mode, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); SET_CDO_DATA(Status, ASYNC); </pre>	Communication déclenchée par les événements

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
20	Commutateur de mode de transfert	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ)) && (GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none) = "OK") && (CDO_ID = Trans_Mode) && FSMUL-CONFIG.req] / GET_CDO_DATA(Trans_Mode, CDO_data); DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, Trans_Mode, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); SET_CDO_DATA(Status, WCONFIG);	Attendre la configuration
21	Communication déclenchée par les événements	CTC-SDN.req(SAP_ID, Node_ID, Length, DLSDU) / DLSDU = BUILD_PDU_EVD(Node_ID, Length, SDU); Len = GET_LEN(DLSDU); SRC_Send_Frame.req(Port_sel, Node_ID, Len, DLSDU); CTC-SDN.cnf("OK");	Communication déclenchée par les événements
22	Communication déclenchée par les événements	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [Rcv_sts = "OK"] / Node_ID = GET_DA(DLSDU); Len = GET_LEN(DLSDU); DLSDU = GET_DATA(DLSDU); CTC-SDN.ind(SAP_ID, Node_ID, Len, SDU);	Communication déclenchée par les événements
23	Communication déclenchée par les événements	FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result);	Communication déclenchée par les événements
24	Communication déclenchée par les événements	FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal);	Communication déclenchée par les événements
25	Communication déclenchée par les événements	FSMUL-MEAS-DELAY.req / V(Ndly_cnt) = 0; SET_CDO_DATA(Status, MEASURED_DLY);	Mesure différée
26	Mesure différée	SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_ASYNC_IND = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.EventType = MEAS_DLY))] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU);	Mesure différée
27	Mesure différée	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS = REQ)) && (V(Ndly_cnt) < V(Nmax_nodes))] / GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SA, RESP, DLSDU.TimeStampT3, V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); V(Ndly_cnt)++;	Mesure différée
28	Mesure différée	SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS = REQ)) && (V(Ndly_cnt) >= V(Nmax_nodes))] / GET_RECV_TIME(V(Treq_r)); DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SA, RESP, DLSDU.TimeStampT3, V(Treq_r)); SRC_Send_Frame.req(Port_sel, DLSDU); DLSDU = BUILD_PDU_ASYNC_IND(V(Nmst1_add), MEAS_DLY); SRC_Send_Frame.req(Port_sel, DLSDU); SET_CDO_DATA(Status, MEAS_DLY_CMP); V(Ndly_cnt) = 0;	Mesure différée

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
29	Mesure différée	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_SYNC = CHECK_PDU_FCMD(DLSDU))] / GET_CDO_DATA(Status, V(Nstatus)); if(V(Nstatus) == MEAS_DLY_CMP) { RESTART_TIMER(Tcycle); } else { GET_SYNC_DATA(DLSDU, V(Tsyn_s)); GET_RECV_TIME(V(Tsyn_r)); } </pre>	Mesure différée
30	Mesure différée	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_REQ_SEND = CHECK_PDU_FCMD(DLSDU) && (DLSDU.PollingType = MEAS_DLY))] / DLSDU = BUILD_PDU_MEAS_DLY(DLSDU.SM_ADD, REQ, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	Mesure différée
31	Mesure différée	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_MEAS_DLY = CHECK_PDU_FCMD(DLSDU) && (DLSDU.HS = RESP) && (DLSDU.SA = V(Nsm_add)))] => GET_DELAY_DATA(DLSDU, V(Treq_s), V(Treq_r)); V(Ndelay) = ((Treq_r - Tsyn_s) - (Treq_s - Tsyn_r)) / 2; SET_CDO_DATA(DELAY, V(Ndelay)); SET_CDO_DATA(Status, MEAS_DLY_CMP); </pre>	Mesure différée
32	Mesure différée	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_READ_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = REQ))] / Result = GET_CDO_READ_DATA(DLSDU, 1, CDO_ID, none); if(Result == "OK") { GET_CDO_DATA(CDO_ID, CDO_data) DLSDU = BUILD_PDU_CDO_READ(V(Nmst1_add), ACK, CDO_ID, OK, CDO_data); SRC_Send_Frame.req(Port_num, DLSDU); } </pre>	Mesure différée
33	Mesure différée	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_REQ_SEND = CHECK_PDU_FCMD(DLSDU) && (DLSDU.PollingType = SYNC))] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); </pre>	Mesure différée
34	Mesure différée	<pre> FSMUL-SET-VALUE.req(Var_ID, Val) / Result = SET_VALUE(Var_ID, Val); FSMUL-SET-VALUE.cnf(Result); </pre>	Mesure différée
35	Mesure différée	<pre> FSMUL-GET-VALUE.req(Par_ID) / Result = GET_VALUE(Var_ID, CurVal); FSMUL-GET-VALUE.cnf(Result, CurVal); </pre>	Mesure différée
36	Mesure différée	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_TOKEN = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.TokenType = MST_MEAS_DLY)] / V(Ndmn2) = 1; V(Niomap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Niomap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, mst2_uport, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); </pre>	Mesure de retard de son propre domaine de commande

N° T	État de la source	Événement (arguments) [conditions] / Action	État de la cible
37	Mesure de retard de son propre domaine de commande	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) < V(Nmax_dmn2))] / Result = GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) if(Result = "OK") { V(Ndmn2)++; V(Nimap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Nimap), Node_Address); DLSDU = BUILD_PDU_CDO_READ(V(Nnode_add), Req, mst2_uport, 1); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp)); }</pre>	Mesure de retard de son propre domaine de commande
38	Mesure de retard de son propre domaine de commande	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) >= V(Nmax_dmn2)) && (CHECK_SYNC_RIGHT() = "OWN")] / DLSDU = BUILD_PDU_SYNC(); SRC_Send_Frame.req(Port_sel, DLSDU); V(Ndmn2) = 1; V(Nimap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Nimap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp));</pre>	Mesure de retard de son propre domaine de commande
39	Mesure de retard de son propre domaine de commande	<pre> SRC_Recv_Frame.ind(Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_WRITE_CDO = CHECK_PDU_FCMD(DLSDU)) && (GET_CDO_WRITE_DATA(DLSDU, 1, CDO_ID, CDO_data) = "OK") && (DLSDU.HS = ACK) && (DLSDU.SA = V(Nnode_add)) && (V(Ndmn2) >= V(Nmax_dmn2)) && (CHECK_SYNC_RIGHT() = "SLV")] / DLSDU = BUILD_PDU_REQ_SEND(V(Nsm_add), SYNC, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp));</pre>	Mesure de retard de son propre domaine de commande
40	Mesure de retard de son propre domaine de commande	<pre> SRC_Recv_Frame.ind(Port_num, Rcv_sts, Length, DLSDU) [(Rcv_sts = "OK") && (FCMD_SYNC = CHECK_PDU_FCMD(DLSDU))] / RESTART_TIMER(Tcycle); V(Ndmn2) = 1; V(Nimap) = GET_DOMAIN_AXISNUM(N(my_add), V(dmn2)); V(Nnode_add) = GET_NODE_INFO(V(Nimap), Node_Address); DLSDU = BUILD_PDU_MEAS_DLY(V(Nnode_add), Req, none, none); SRC_Send_Frame.req(Port_sel, DLSDU); V(Nlast_frame) = DLSDU.FCMDCode; START_TIMER(T(Twait), V(Tresp));</pre>	Mesure de retard de son propre domaine de commande