IEC 61158-4-12

Edition 3.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

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

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 4-12: Spécification du protocole de la couche liaison de données – Éléments de type 12

IEC 61158-4-12:2014-08(en-fr)

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

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

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

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

IEC Catalogue - webstore.iec.ch/catalogue
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub
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 It also gives information on projects, replaced and withdrawn publications.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 also once a month by email.

Electropedia - www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 14 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
More than 55 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

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: csc@iec.ch.

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.

Catalogue IEC - webstore.iec.ch/catalogue
Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub
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 aussi une fois par mois par email.

Electropedia - www.electropedia.org
Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 14 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
Plus de 55 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

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

IEC 61158-4-12

Edition 3.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

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

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 4-12: Spécification du protocole de la couche liaison de données – Éléments de type 12

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XG

ICS 25.040.40; 35.100.20; 35.110

ISBN 978-2-8322-1724-5

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 4-12: Data-link layer protocol specification – Type 12 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.

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 IEC 61784-1 and IEC 61784-2.

International Standard IEC 61158-4-12 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2010. This edition constitutes a technical revision. The main changes with respect to the previous edition are listed below:

- bug fixes and
- editorial improvements.

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

FDIS	Report on voting
65C/762/FDIS	65C/772/RVD

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

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

A list of all the parts of the IEC 61158 series, published 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 publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

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

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

NTRODUCTION

This part of IEC 61158 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 61158-1:2013.

The data-link protocol provides the data-link service by making use of the services available from the physical layer. The primary aim of this standard is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer data-link entities (DLEs) 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:

a) as a guide for implementors and designers;

b) for use in the testing and procurement of equipment;

c) as part of an agreement for the admittance of systems into the open systems environment;

d) as a refinement to the understanding of time-critical communications within OSI.

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

NOTE Use of some of the associated protocol types is restricted by their 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 particular data-link layer protocol type to be used with physical layer and application layer protocols in Type combinations as specified explicitly in the profile parts. Use of the various protocol types in other combinations may require permission from their respective intellectual-property-right holders.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning Type 12 elements and possibly other types given as follows:

EP 1 590 927 B1	[BE]	Koppler für ein Netzwerk mit Ringtopologie und ein auf Ethernet basierten Netzwerk
EP 1 789 857 B1	[BE]	Datenübertragungsverfahren und automatisierungssystem zum Einsatz eines solchen Datenübertragungsverfahrens
DE 102007017835.4	[BE]	Paketvermittlungsvorrichtung und lokales Kommunikationsnetz mit einer solchen Paketvermittlungsvorrichtung
EP 1 456 722 B1	[BE]	Datenübertragungsverfahren, serielles Bussystem und Anschalteinheit für einen passiven Busteilnehmer

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

The holder of these patent rights has assured the IEC that he/she is willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of these patent rights is registered with IEC. Information may be obtained from:

[BE]: Beckhoff Automation GmbH
Eiserstraße 5
33415 Verl,
Germany

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

ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 4-12: Data-link layer protocol specification – Type 12 elements

1 Scope

1.1 General

The data-link layer provides basic time-critical messaging communications between devices in an automation environment.

This protocol provides communication opportunities to all participating data-link entities

a) in a synchronously-starting cyclic manner, and

b) in a cyclic or acyclic asynchronous manner, as requested each cycle by each of those data-link entities.

Thus this protocol can be characterized as one which provides cyclic and acyclic access asynchronously but with a synchronous restart of each cycle.

1.2 Specifications

This standard specifies

a) procedures for the transfer of data and control information from one data-link user entity to one or more user entity;

b) the structure of the DLPDUs used for the transfer of data and control information by the protocol of this standard, and their representation as physical interface data units.

1.3 Procedures

The procedures are defined in terms of

a) the interactions between DL-entities (DLEs) through the exchange of DLPDUs;

b) the interactions between a DL-service (DLS) provider and a DLS-user in the same system through the exchange of DLS primitives;

c) the interactions between a DLS-provider and the MAC services of ISO/IEC 8802-3.

1.4 Applicability

These procedures are applicable to instances of communication between systems which support time-critical communications services within the data-link layer of the OSI reference model, and which require the ability to interconnect in an open systems interconnection environment.

Profiles provide a simple multi-attribute means of summarizing an implementation's capabilities, and thus its applicability to various time-critical communications needs.

1.5 Conformance

This standard also specifies conformance requirements for systems implementing these procedures. This part of this standard does not contain tests to demonstrate compliance with such requirements.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. 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 IEC 61784-1 and IEC 61784-2 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 61158-3-12, *Industrial communication networks – Fieldbus specifications – Part 3-12: Data-link layer service definition – Type 12 elements*

IEC 61588, *Precision clock synchronization protocol for networked measurement and control systems*

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

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

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*

IEEE 802.1Q, *IEEE Standard for Local and metropolitan Area Networks – Virtual Bridged Local Area Networks,* available at <http://www.ieee.org>

IETF RFC 768, *User Datagram Protocol (UDP),* available at <http://www.ietf.org>

IETF RFC 791, *Internet protocol DARPA internet program protocol specification*, available at <http://www.ietf.org>

3 Terms, definitions, symbols, abbreviations and conventions

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

3.1 Reference model terms and definitions

This standard is based in part on the concepts developed in ISO/IEC 7498-1 and ISO/IEC 7498-3, and makes use of the following terms defined therein.

3.1.1 DL-duplex-transmission [ISO/IEC 7498-1]

3.1.2 DL-protocol [ISO/IEC 7498-1]

3.1.3 DL-protocol-data-unit [ISO/IEC 7498-1]

3.1.4 **(N)-entity** [ISO/IEC 7498-1]
DL-entity
Ph-entity

3.1.5 **(N)-interface-data-unit** [ISO/IEC 7498-1]
DL-service-data-unit (N=2)
Ph-interface-data-unit (N=1)

3.1.6 **(N)-layer** [ISO/IEC 7498-1]
DL-layer (N=2)
Ph-layer (N=1)

3.1.7 **(N)-service** [ISO/IEC 7498-1]
DL-service (N=2)
Ph-service (N=1)

3.1.8 **(N)-service-access-point** [ISO/IEC 7498-1]
DL-service-access-point (N=2)
Ph-service-access-point (N=1)

3.1.9 **(N)-service-access-point-address** [ISO/IEC 7498-1]
DL-service-access-point-address (N=2)
Ph-service-access-point-address (N=1)

3.1.10 **peer-entities** [ISO/IEC 7498-1]

3.1.11 **Ph-interface-data** [ISO/IEC 7498-1]

3.1.12 **primitive name** [ISO/IEC 7498-3]

3.1.13 **reassembling** [ISO/IEC 7498-1]

3.1.14 **recombining** [ISO/IEC 7498-1]

3.1.15 **reset** [ISO/IEC 7498-1]

3.1.16 **routing** [ISO/IEC 7498-1]

3.1.17 **segmenting** [ISO/IEC 7498-1]

3.1.18 **sequencing** [ISO/IEC 7498-1]

3.1.19 **splitting** [ISO/IEC 7498-1]

3.1.20 **systems-management** [ISO/IEC 7498-1]

3.2 Service convention terms and definitions

This standard also makes use of the following terms defined in ISO/IEC 10731 as they apply to the data-link layer:

3.2.1 **asymmetrical service**

3.2.2 **confirm (primitive);**
requestor.deliver (primitive)

3.2.3 **deliver (primitive)**

3.2.4 **DL-service-primitive;**
primitive

3.2.5 **DL-service-provider**

3.2.6 **DL-service-user**

3.2.7 indication (primitive)
acceptor.deliver (primitive)

3.2.8 request (primitive);
requestor.submit (primitive)

3.2.9 requestor

3.2.10 response (primitive);
acceptor.submit (primitive)

3.2.11 submit (primitive)

3.2.12 symmetrical service

3.3 Common terms and definitions

NOTE Many definitions are common to more than one protocol Type; they are not necessarily used by all protocol Types.

For the purpose of this document, the following definitions also apply:

3.3.1
frame
denigrated synonym for DLPDU

3.3.2
group DL-address
DL-address that potentially designates more than one DLSAP within the extended link

Note 1 to entry: A single DL-entity may have multiple group DL-addresses associated with a single DLSAP.

Note 2 to entry: A single DL-entity also may have a single group DL-address associated with more than one DLSAP.

3.3.3
node
single DL-entity as it appears on one local link

3.3.4
receiving DLS-user
DL-service user that acts as a recipient of DLS-user-data

Note 1 to entry: A DL-service user can be concurrently both a sending and receiving DLS-user.

3.3.5
sending DLS-user
DL-service user that acts as a source of DLS-user-data

3.4 Additional Type 12 definitions

3.4.1
application
function or data structure for which data is consumed or produced

[SOURCE: IEC 61158-5-12, 3.3.1]

3.4.2
application objects
multiple object classes that manage and provide a run time exchange of messages across the network and within the network device

3.4.3
basic slave
slave device that supports only physical addressing of data

3.4.4
bit
unit of information consisting of a 1 or a 0

Note 1 to entry: This is the smallest data unit that can be transmitted.

3.4.5
client
1) object which uses the services of another (server) object to perform a task

2) initiator of a message to which a server reacts

3.4.6
connection
logical binding between two application objects within the same or different devices

3.4.7
cyclic
events which repeat in a regular and repetitive manner

3.4.8
cyclic redundancy check
CRC
residual value computed from an array of data and used as a representative signature for the array

3.4.9
data
generic term used to refer to any information carried over a Fieldbus

3.4.10
data consistency
means for coherent transmission and access of the input- or output-data object between and within client and server

3.4.11
device
physical entity connected to the fieldbus composed of at least one communication element (the network element) and which may have a control element and/or a final element (transducer, actuator, etc.)

[SOURCE: IEC 61158-2, 3.1.13]

3.4.12
distributed clocks
method to synchronize slaves and maintain a global time base

3.4.13
error
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.4.14
event
instance of a change of conditions

3.4.15
fieldbus memory management unit
function that establishes one or several correspondences between logical addresses and physical memory

3.4.16
fieldbus memory management unit entity
single element of the fieldbus memory management unit: one correspondence between a coherent logical address space and a coherent physical memory location

3.4.17
full slave
slave device that supports both physical and logical addressing of data

3.4.18
interface
shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics as appropriate

3.4.19
master
device that controls the data transfer on the network and initiates the media access of the slaves by sending messages and that constitutes the interface to the control system

3.4.20
mapping
correspondence between two objects in that way that one object is part of the other object

3.4.21
medium
cable, optical fibre, or other means by which communication signals are transmitted between two or more points

Note 1 to entry: "media" is used as the plural of medium.

3.4.22
message
ordered series of octets intended to convey information

Note 1 to entry: Normally used to convey information between peers at the application layer.

3.4.23
network
set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.4.24
node
end-point of a link in a network or a point at which two or more links meet

[SOURCE: IEC 61158-2, 3.1.31, with some wording adjustment]

3.4.25
object
abstract representation of a particular component within a device

Note 1 to entry: An object can be

1) an abstract representation of the capabilities of a device. Objects can be composed of any or all of the following components:

   a) data (information which changes with time);

   b) configuration (parameters for behavior);

   c) methods (things that can be done using data and configuration).

2) a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior.

3.4.26
process data
data object containing application objects designated to be transferred cyclically or acyclically for the purpose of processing

3.4.27
server
object which provides services to another (client) object

3.4.28
service
operation or function than an object and/or object class performs upon request from another object and/or object class

3.4.29
slave
DL-entity accessing the medium only after being initiated by the preceding slave or the master

3.4.30
Sync manager
collection of control elements to coordinate access to concurrently used objects

3.4.31
Sync manager channel
single control elements to coordinate access to concurrently used objects

3.4.32
switch
MAC bridge as defined in IEEE 802.1D

3.5 Common symbols and abbreviations

NOTE Many symbols and abbreviations are common to more than one protocol Type; they are not necessarily used by all protocol Types.

DL-	Data-link layer (as a prefix)
DLC	DL-connection
DLCEP	DL-connection-end-point
DLE	DL-entity (the local active instance of the data-link layer)
DLL	DL-layer
DLPCI	DL-protocol-control-information
DLPDU	DL-protocol-data-unit

DLM	DL-management
DLME	DL-management Entity (the local active instance of DL-management)
DLMS	DL-management Service
DLS	DL-service
DLSAP	DL-service-access-point
DLSDU	DL-service-data-unit
FIFO	First-in first-out (queuing method)
OSI	Open systems interconnection
Ph-	Physical layer (as a prefix)
PhE	Ph-entity (the local active instance of the physical layer)
PhL	Ph-layer
QoS	Quality of service

3.6 Additional Type 12 symbols and abbreviations

AL	Application layer
DLSDU	Data-link protocol data unit
APRD	Auto increment physical read
APRW	Auto increment physical read write
APWR	Auto increment physical write
ARMW	Auto increment physical read multiple write
BRD	Broadcast read
BRW	Broadcast read write
BWR	Broadcast write
CAN	Controller area network
CoE	CAN application protocol over Type 12 services
CSMA/CD	Carrier sense multiple access with collision detection
DC	Distributed clocks
DCSM	DC state machine
DHSM	(DL) PDU handler state machine
Type 12	Prefix for DL services and protocols
E²PROM	Electrically erasable programmable read only memory
EoE	Ethernet tunneled over Type 12 services
ESC	Type 12 slave controller
FCS	frame check sequence
FMMU	Fieldbus memory management unit
FoE	File access with Type 12 services
FPRD	Configured address physical read
FPRW	Configured address physical read write
FPWR	Configured address physical write
FRMW	Configured address physical read multiple write
HDR	Header
ID	Identifier
IP	Internet protocol
LAN	Local area network
LRD	Logical memory read
LRW	Logical memory read write

LWR	Logical memory write
MAC	Media access control
MDI	Media dependent interface (specified in ISO/IEC 8802-3)
MDX	Mailbox data exchange
MII	Media independent interface (specified in ISO/IEC 8802-3)
PDI	Physical device interface (a set of elements that allows access to DL services from the DLS-user)
PDO	Process data object
PHY	Physical layer device (specified in ISO/IEC 8802-3)
PNV	Publish network variable
RAM	Random access memory
RMSM	Resilient mailbox state machine
Rx	Receive
SDO	Service data object
SII	Slave information interface
SIISM	SII state machine
SyncM	Synchronization manager
SYSM	Sync manager state machine
TCP	Transmission control protocol
Tx	Transmit
UDP	User datagram protocol
WKC	Working counter

3.7 Conventions

3.7.1 General concept

The services are specified in IEC 61158-3-12. The service specification defines the services that are provided by the Type 12 DL. The mapping of these services to ISO/IEC 8802-3 is described in this international Standard.

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

3.7.1.1 Abstract syntax conventions

The DL syntax elements related to PDU structure are described as shown in the example of Table 1.

Frame part denotes the element that will be replaced by this reproduction.

Data field is the name of the elements.

Data Type denotes the type of the terminal symbol.

Value/Description contains the constant value or the meaning of the parameter.

Table 1 – PDU element description example

Frame part	Data Field	Data Type	Value/Description
Type 12 xxx	CMD	Unsigned8	0x01
	IDX	Unsigned8	Index
	ADP	Unsigned16	Auto Increment Address
	ADO	Unsigned16	Physical Memory Address
	LEN	Unsigned11	Length of data of YYY in octets
	Reserved	Unsigned4	0x00
	NEXT	Unsigned1	0x00: last Type 12 PDU 0x01: Type 12 PDU follows
	IRQ	Unsigned16	Reserved for future use
	YYY		next element
	WKC	Unsigned16	Working Counter

The attribute types are described in C language notations (ISO/IEC 9899) as shown in Figure 1. BYTE and WORD are elements of type unsigned char and unsigned short.

```
typedef struct
{
  Unsigned8     Type;
  Unsigned8     Revision;
  Unsigned16    Build;
  Unsigned8     NoOfSuppFmmuChannels;
  Unsigned8     NoOfSuppSyncManChannels;
  Unsigned8     RamSize;
  Unsigned8     Reserved1;
  unsigned      FmmuBitOperationNotSupp:  1;
  unsigned      Reserved2:                7;
  unsigned      Reserved3:                8;
} TDLINFORMATION;
```

Figure 1 – Type description example

The attributes itself are described in a form as shown in Table 2.

Parameter describes a single element of the attribute.

Physical address denotes the location in physical address space.

Data Type denotes the type of this element.

Access type Type 12 DL/PDI shows the access right to this element. R means read access right, W means write access right. If neither Type 12 DL nor PDI has write access, this variable will be initialised and maintained by DL itself.

Value/Description contains the constant value and/or the meaning of the parameter.

Table 2 – Example attribute description

Parameter	Physical Address	Data Type	Access type	Access Type PDI	Value/Description
State	0x0120	Unsigned4	RW	R	0x01: Init Request 0x02: Pre-Operational Request 0x03: Bootstrap Mode Request 0x04: Safe Operational Request 0x08: Operational Request
Acknowledge	0x0120	Unsigned1	RW	R	0x00: no acknowledge 0x01 acknowledge (shall be a positive edge)
Reserved	0x0120	Unsigned3	RW	R	0x00
Application Specific	0x0121	Unsigned8	RW	R	

3.7.1.2 Convention for the encoding of reserved bits and octets

The term "reserved" may be used to describe bits in octets or whole octets. All bits or octets that are reserved should be set to zero at the sending side and shall not be tested at the receiving side except it is explicitly stated or if the reserved bits or octets are checked by a state machine.

The term "reserved" may also be used to indicate that certain values within the range of a parameter are reserved for future extensions. In this case the reserved values should not be used at the sending side and shall not be tested at the receiving side except it is explicitly stated or if the reserved values are check by a state machine.

3.7.1.3 Conventions for the common coding s of specific field octets

DLSDUs may contain specific fields that carry information in a primitive and condensed way. These fields shall be coded in the order according to Figure 2.

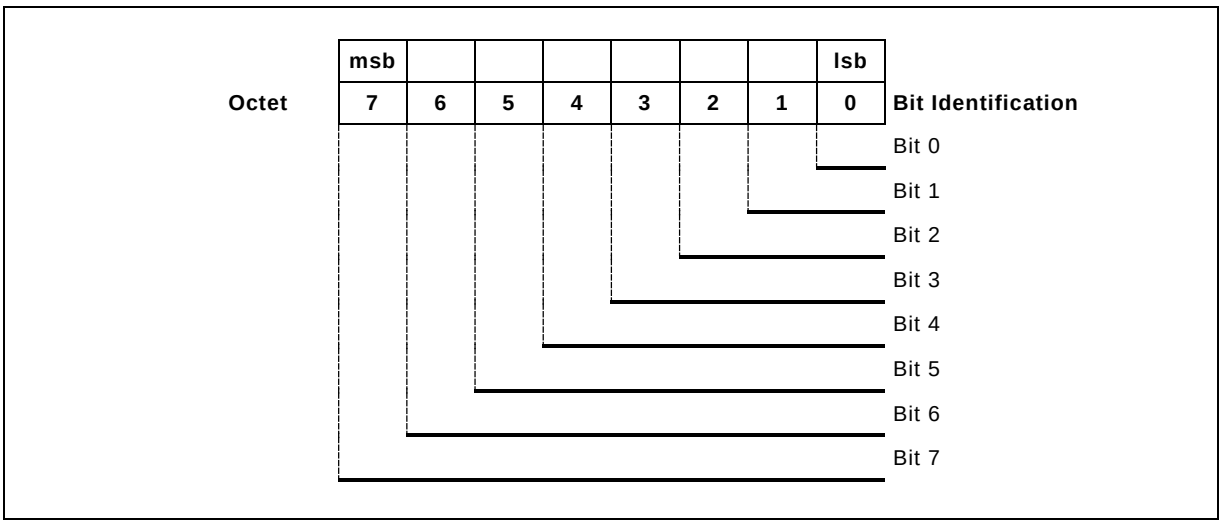


Figure 2 – Common structure of specific fields

Bits may be grouped as group of bits. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. Alias names may be used for each bit or group of bits or they may be marked as

reserved. The grouping of individual bits shall be in ascending order without gaps. The values for a group of bits may be represented as binary, decimal or hexadecimal values. This value shall only be valid for the grouped bits and can only represent the whole octet if all 8 bits are grouped. Decimal or hexadecimal values shall be transferred in binary values so that the bit with the highest number of the group represents the msb concerning the grouped bits.

EXAMPLE Description and relation for the specific field octet

Bit 0: reserved.

Bit 1-3: Reason_Code The decimal value 2 for the Reason_Code means general error.

Bit 4-7: shall always set to one.

The octet that is constructed according to the description above looks as follows:

(msb) Bit 7 = 1,

Bit 6 = 1,

Bit 5 = 1,

Bit 4 = 1,

Bit 3 = 0,

Bit 2 = 1,

Bit 1 = 0,

(lsb) Bit 0 = 0.

This bit combination has an octet value representation of 0xf4.

3.7.2 State machine conventions

The protocol sequences are described by means of State Machines.

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

The textual listing of the state transitions is structured as follows, see also Table 3.

– The first column contains the name of the transition.
– The second column in define the current state.
– The third column contains an optional event followed by Conditions starting with a “/” as first line character and finally followed by the actions starting with a “=>” as first line character.
– The last column contains the next state.

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

The layout of a Machine description is shown in Table 3. The meaning of the elements of a State Machine Description are shown in Table 4.

Table 3 – State machine description elements

#	Current state	Event /Condition => Action	Next state

Table 4 – Description of state machine elements

Description element	Meaning
Current state Next state	Name of the given states.
#	Name or number of the state transition.
Event	Name or description of the event.
/Condition	Boolean expression. The preceding “\” is not part of the condition.
=> Action	List of assignments and service or function invocations. The preceding “=>” is not part of the action.

The conventions used in the state machines are shown in Table 5.

Table 5 – Conventions used in state machines

Convention	Meaning
=	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.
axx	A parameter name if a is a letter. EXAMPLE Identifier = reason means value of a 'reason' parameter is assigned to a parameter called 'Identifier.'
"xxx"	Indicates fixed visible string. EXAMPLE Identifier = "abc" means value "abc" is assigned to a parameter named 'Identifier.'
nnn	if all elements are digits, the item represents a numerical constant shown in decimal representation
0xnn	if all elements nn are digits, the item represents a numerical constant shown in hexadecimal representation
==	A logical condition to indicate an item on the left is equal to an item on the right.
<	A logical condition to indicate an item on the left is less than the item on the right.
>	A logical condition to indicate an item on the left is greater than the item on the right.
!=	A logical condition to indicate an item on the left is not equal to an item on the right.
&&	Logical "AND"
||	Logical "OR"
!	Logical "NOT"
+ - * /	Arithmetic operators
;	Separator of expressions

Readers are strongly recommended to refer to the subclauses for the attribute definitions, the local functions, and the FDL-PDU definitions to understand protocol machines. It is assumed

that readers have sufficient knowledge of these definitions and they are used without further explanations.

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

4 Overview of the DL-protocol

4.1 Operating principle

Type 12 DL is a Real Time Ethernet technology that aims to maximize the utilization of the full duplex Ethernet bandwidth. Medium access control employs the Master/Slave principle, where the master node (typically the control system) sends the Ethernet frames to the slave nodes, which extract data from and insert data into these frames.

From an Ethernet point of view, a Type 12 segment is a single Ethernet device, which receives and sends standard ISO/IEC 8802-3 Ethernet frames. However, this Ethernet device is not limited to a single Ethernet controller with downstream microprocessor, but may consist of a large number of Type 12 slave devices. These process the incoming frames directly and extract the relevant user data, or insert data and transfer the frame to the next slave device. The last Type 12 slave device within the segment sends the fully processed frame back, so that it is returned by the first slave device to the master as response frame.

This procedure utilizes the full duplex mode of Ethernet: both communication directions are operated independently. Direct communication without switch between a master device and a Type 12 segment consisting of one or several slave devices may be established.

4.2 Topology

The topology of a communication system is one of the crucial factors for the successful application in automation. The topology has significant influence on the cabling effort, diagnostic features, redundancy options and hot-plug-and-play features.

The star topology commonly used for Ethernet leads to enhanced cabling effort and infrastructure costs. Especially for automation applications a line or tree topology is preferable.

The slave node arrangement represents an open ring bus. At the open end, the master device sends frames, either directly or via Ethernet switches, and receives them at the other end after they have been processed. All frames are relayed from the first node to the next ones. The last node returns the PDU back to the master. Utilizing the full duplex capabilities of Ethernet, the resulting topology is a physical line.

Branches, which in principle are possible anywhere, can be used to enhance the line structure into a tree structure from. A tree structure supports very simple wiring; individual branches, for example, can branch into control cabinets or machine modules, while the main line runs from one module to the next.

4.3 Frame processing principles

In order to achieve maximum performance, the Ethernet frames should be processed directly “on the fly”. If it is implemented this way, the slave node recognizes relevant commands and executes them accordingly while the frames are already passed on.

NOTE 1 Type 12 DL can be implemented using standard Ethernet controllers without direct processing. The influence of the forwarding mechanism implementation on communication performance is detailed in the profile parts.

The nodes have an addressable memory that can be accessed with read or write services, either each node consecutively or several nodes simultaneously. Several Type 12 PDUs can be embedded within an Ethernet frame, each PDU addressing a cohesive data section. As shown in Figure 3, the Type 12 PDUs are either transported:

a) directly in the data area of the Ethernet frame,

b) within the data section of a UDP datagram transported via IP.

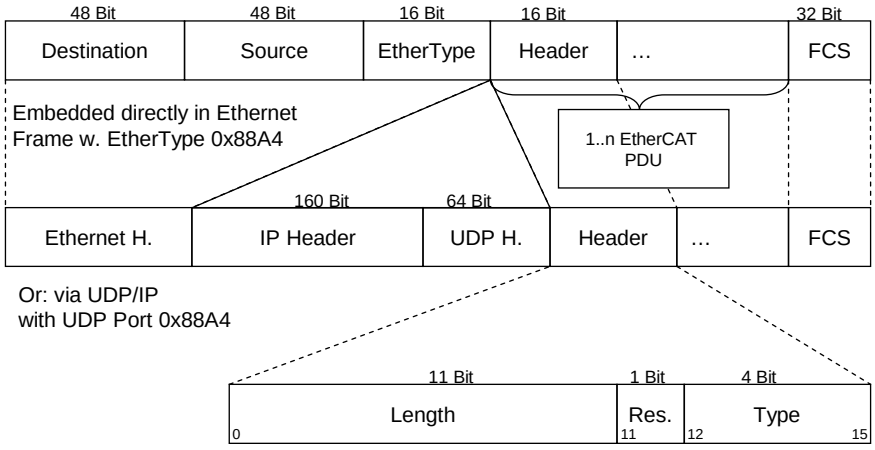


Figure 3 – Frame structure

Variant a) is limited to one Ethernet subnet, since associated frames are not relayed by routers. For machine control applications this usually does not represent a constraint. Multiple Type 12 segments can be connected to one or several switches. The Ethernet MAC address of the first node within the segment is used for addressing the Type 12 segment.

NOTE 2 Further addressing details are given in the data-link layer service definition (see IEC 61158-3-12).

Variant b) via UDP/IP generates a slightly larger overhead (IP and UDP header), but for less time-critical applications such as building automation it allows using IP routing. On the master side any standard UDP/IP implementation can be used.

4.4 Data-link layer overview

Several nodes can be addressed individually via a single Ethernet frame carrying several Type 12 PDUs. The Type 12 PDUs are packed without gaps. The frame is terminated with the last Type 12 PDU, unless the frame size is less than 64 octets, in which case the frame will be padded to 64 octets in length.

Compared with one frame per node this leads to a better utilization of the Ethernet bandwidth. However, for e.g. a 2 channel digital input node with just 2 bit of user data, the overhead of a single Type 12 PDU is still excessive.

Therefore the slave nodes may also support logical address mapping. The process data can be inserted anywhere within a logical address space. If a Type 12 PDU is sent that contains read or write services for a certain process image area located at the corresponding logical address, instead of addressing a particular node, the nodes insert the data at or extract the data from the right place within the process data, as noted in Figure 4.

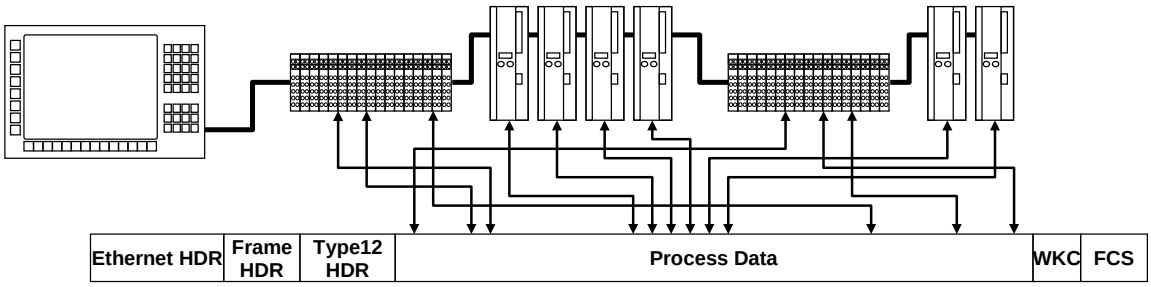


Figure 4 – Mapping of data in a frame

All other nodes that also detect an address match with the process image also insert their data, so that many nodes can be addressed simultaneously with a single Type 12 PDU. The master can assemble completely sorted logical process images via a single Type 12 PDU. Additional mapping is no longer required in the master, so that the process data can be assigned directly to the different control tasks. Each task can create its own process image and exchange it within its own timeframe. The physical order of the nodes is completely arbitrary and is only relevant during the first initialization phase.

The logical address space is 2^{32} Bytes = 4 GByte. Type 12 DL can be considered to be a serial backplane for automation systems that enables connection to distributed process data for both large and very small automation devices. Using a standard Ethernet controller and a standard Ethernet cable, a very large number of I/O channels without practical restrictions on the distribution can be connected to automation devices, which can be accessed with high bandwidth, minimum delay and near-optimum usable data rate. At the same time, devices such as fieldbus scanners can be connected as well, thus preserving existing technologies and standards.

4.5 Error detection overview

Type 12 DL checks by the Ethernet frame check sequence (FCS) whether a frame was transmitted correctly. Since one or several slaves modify the frame during the transfer, the FCS is recalculated by each slave. If a slave detects a checksum error, the slave does not repair the FCS but flags the master by incrementing the error counter, so that a fault can be located precisely.

When reading data from or writing data to a Type 12 PDU, the addressed slave increments a working counter (WKC) positioned at the end of each Type 12 PDU. Analyzing the working counter allows the master to check if the expected number of nodes has processed the corresponding Type 12 PDU.

4.6 Node reference model

4.6.1 Mapping onto OSI basic reference model

Type 12 DL is described using the principles, methodology and model of ISO/IEC 7498 Information processing systems — Open Systems Interconnection — Basic Reference Model (OSI). The OSI model provides a layered approach to communications standards, whereby the layers can be developed and modified independently. The Type 12 DL specification defines functionality from top to bottom of a full OSI stack, and some functions for the users of the stack. Functions of the intermediate OSI layers, layers 3 – 6, are consolidated into either the Type 12 DL data-link layer or the Type 12 DL Application layer. Likewise, features common to users of the Fieldbus Application layer may be provided by the Type 12 DL Application layer to simplify user operation, as noted in Figure 5.

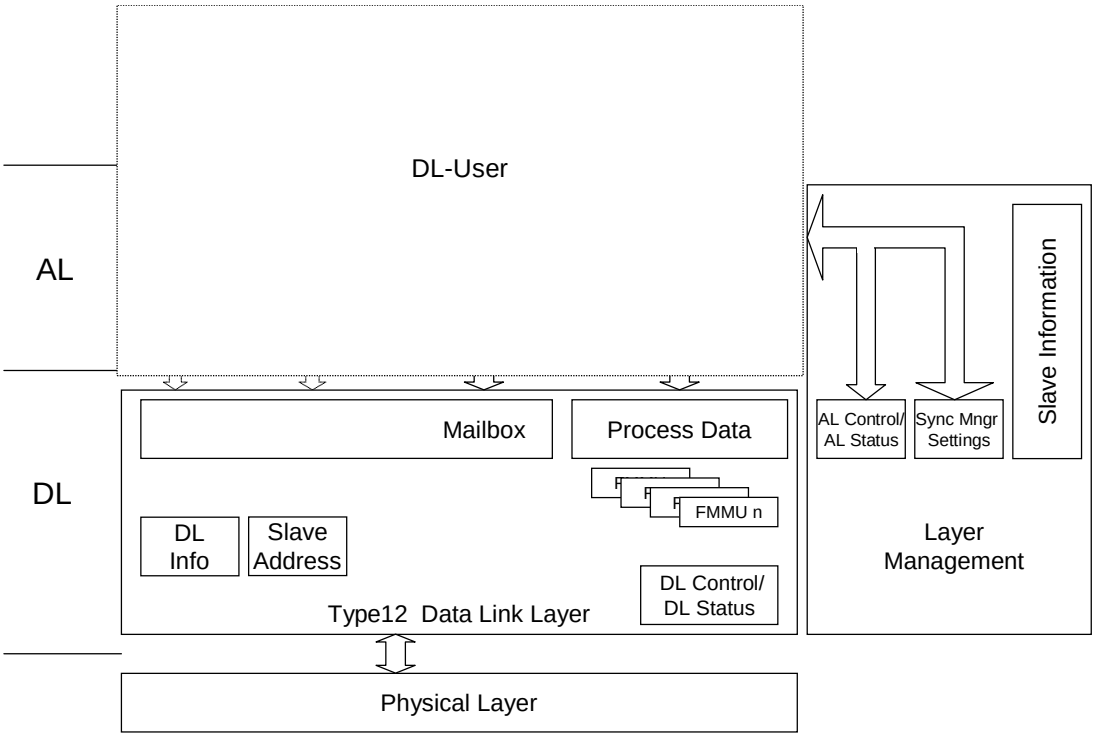


Figure 5 – Slave node reference model

4.6.2 Data-link Layer features

The data link layer provides basic time critical support for data communications among devices connected via Type 12 DL. The term “time-critical” is used to describe applications having 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.

The data link layer has the task to compute, compare and generate the frame check sequence and provide communications by extracting data from and/or including data into the Ethernet frame. This is done depending on the data link layer parameters which are stored at pre-defined memory locations. The application data is made available to the application layer in physical memory, either in a mailbox configuration or within the process data section.

4.7 Operation overview

4.7.1 Relation to ISO/IEC 8802-3

This part specifies data link layer services in addition to those specified in ISO/IEC 8802-3.

4.7.2 Frame structure

A Type 12 Ethernet frame contains one or several Type 12 PDUs (as shown in Figure 6), each addressing individual devices and/or memory areas. The Type 12 frame is recognized by the combination of the EtherType 0x88A4[1] and the corresponding Type 12 frame header or, when transported via UDP/IP according to IETF RFC 791/IETF RFC 768 (as shown in Figure 7) by the Destination UDP port 34980=0x88A4[2] and the Type 12 frame header. Fragmentation of IP packets will be ignored. The UDP checksum may be set to 0 by Slaves and could be ignored.

---

1 The EtherType 0x88A4 was assigned for Type 12 (EtherCAT) by the IEEE Registration Authority.

2 The UDP Port 34980 was assigned for Type 12 (EtherCAT) by the Internet Assigned Numbers Authority (IANA).

No check on IP type of service, IP header checksum, IP packet length and UDP length is required.

Each Type 12 PDU consists of a Type 12 header, the data area and a subsequent counter area (working counter), which is incremented by all nodes that were addressed by the Type 12 PDU and have exchanged associated data.

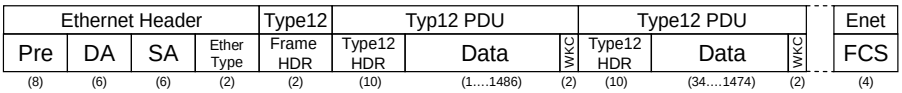


Figure 6 – Type 12 PDUs embedded in Ethernet frame

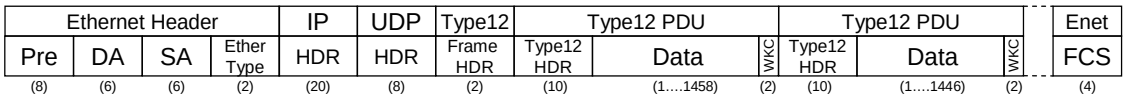


Figure 7 – Type 12 PDUs embedded in UDP/IP

5 Frame structure

5.1 Frame coding principles

Type 12 DL uses a standard ISO/IEC 8802-3 Ethernet frame structure for transporting Type 12 PDUs. The PDUs may alternatively be sent via UDP/IP. The Type 12 specific protocol parts are identical in both cases.

5.2 Data types and encoding rules

5.2.1 General description of data types and encoding rules

To be able to exchange meaningful data, the format of this data and its meaning have to be known by the producer and consumer(s). This specification models this by the concept of data types.

The encoding rules define the representation of values of data types and the transfer syntax for the representations. Values are represented as bit sequences. Bit sequences are transferred in sequences of octets (bytes). For numerical data types the encoding is little endian style as shown in Table 6.

The data types and encoding rules shall be valid for the DL services and protocols as well as for the AL services and protocols specified. The encoding rules for the Ethernet frame are specified in ISO/IEC 8802-3. The DLSDU of Ethernet is an octet string. The transmission order within octets depends upon MAC and PhL encoding rules.

5.2.2 Transfer syntax for bit sequences

For transmission across Type 12 DL a bit sequence is reordered into a sequence of octets. Hexadecimal notation is used for octets as specified in ISO/IEC 9899. Let $b = b_0 \ldots b_{n-1}$ be a bit sequence. Denote k a non-negative integer such that $8(k - 1) < n \leq 8k$. Then b is transferred in k octets assembled as shown in Table 6. The bits b_i, $i \geq n$ of the highest numbered octet are do not care bits.

Octet 1 is transmitted first and octet k is transmitted last. Hence the bit sequence is transferred as follows across the network (transmission order within an octet is determined by ISO/IEC 8802-3):

$b_7, b_6, ..., b_0, b_{15}, ..., b_8, ...$

Table 6 – Transfer Syntax for bit sequences

Octet number	1.	2.	k.
	b_7 .. b_0	b_{15} .. b_8	b_{8k-1} .. b_{8k-8}

EXAMPLE

Bit 9	...	Bit 0
10b	0001b	1100b
0x2	0x1	0xC
		= 0x21C

The bit sequence b = b_0 .. b_9 = 0011 1000 01_b represents an Unsigned10 with the value 0x21C and is transferred in two octets: First 0x1C and then 0x02.

5.2.3 Unsigned Integer

Data of basic data type Unsignedn has values in the non-negative integers. The value range is 0, ..., 2^n-1. The data is represented as bit sequences of length *n*. The bit sequence

$b = b_0 ... b_{n-1}$

is assigned the value

$$Unsignedn(b) = b_{n-1}\times 2^{n-1} + ... + b_1\times 2^1 + b_0\times 2^0$$

The bit sequence starts on the left with the least significant byte.

EXAMPLE The value 266 = 0x10A with data type Unsigned16 is transferred in two octets, first 0x0A and then 0x01.

The Unsignedn data types are transferred as specified in Table 7. Unsigned data types as Unsigned1 to Unsigned7 and Unsigned 9 to Unsigned15 will be used too. In this case the next element will start at the first free bit position as denoted in 3.7.1.

Table 7 – Transfer syntax for data type Unsignedn

octet number	1.	2.	3.	4.	5.	6.	7.	8.
Unsigned8	$b_7..b_0$							
Unsigned16	$b_7..b_0$	$b_{15}..b_8$						
Unsigned32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Unsigned64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.2.4 Signed Integer

Data of basic data type Integern has values in the integers. The value range is from -2^{n-1} to $2^{n-1}-1$. The data is represented as bit sequences of length n. The bit sequence

$$b = b_0 \,..\, b_{n-1}$$

is assigned the value

$$\text{Integern}(b) = b_{n-2}\times 2^{n-2} + \ldots + b_1\times 2^1 + b_0\times 2^0 \text{ if } b_{n-1} = 0$$

and, performing two's complement arithmetic,

$$\text{Integern}(b) = -\ \text{Integern}(\hat{}b) - 1 \text{ if } b_{n-1} = 1$$

NOTE The bit sequence starts on the left with the least significant bit.

EXAMPLE The value –266 = 0xFEF6 with data type Integer16 is transferred in two octets, first 0xF6 and then 0xFE.

The Integern data types are transferred as specified in Table 8. Integer data types as Integer1 to Integer7 and Integer9 to Integer15 will be used too. In this case the next element will start at the first free bit position as denoted in 3.7.1.

Table 8 – Transfer syntax for data type Integern

Octet number	1.	2.	3.	4.	5.	6.	7.	8.
Integer8	$b_7..b_0$							
Integer16	$b_7..b_0$	$b_{15}..b_8$						
Integer32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Integer64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.2.5 Octet String

The data type OctetString*length* is defined below; *length* is the length of the octet string.

ARRAY [length] OF Unsigned8 OctetString*length*

5.2.6 Visible String

The data type VisibleString*length* is defined below. The admissible values of data of type *VISIBLE_CHAR* are 0_h and the range from 0x20 to 0x7E. The data are interpreted as 7-bit coded characters. *length* is the length of the visible string.

Unsigned8 VISIBLE_CHAR

ARRAY [length] OF VISIBLE_CHAR VisibleString*length*

There is no 0x0 necessary to terminate the string.

5.3 DLPDU structure

5.3.1 Type 12 frame inside an Ethernet frame

The frame structure in consists of the following data entries as specified in Table 9.

Table 9 – Type 12 frame inside an Ethernet frame

Frame part	Data field	Data type	Value/description
Ethernet	Dest MAC	BYTE[6]	Destination MAC Address as specified in ISO/IEC 8802-3
	Src MAC	BYTE[6]	Source MAC Address as specified in ISO/IEC 8802-3
(optional)	VLAN Tag	BYTE[4]	0x81, 0x00 and two bytes Tag Control Information as specified in IEEE 802.1Q
	Ether Type	BYTE[2]	0x88, 0xA4 (Type 12)
	Type 12 frame		specified in 5.3.3
	Padding	BYTE[n]	shall be inserted if DL PDU is shorter than 64 octets as specified in ISO/IEC 8802-3
Ethernet FCS	FCS	Unsigned32	Standard Ethernet Checksum coding as specified in ISO/IEC 8802-3

5.3.2 Type 12 frame inside a UDP datagram

The frame structure in consists of the following data entries as specified in Table 10.

Table 10 – Type 12 frame inside an UDP PDU

Frame part	Data field	Data type	Value/description
Ethernet	Dest MAC	BYTE[6]	See Table 9
	Src MAC	BYTE[6]	See Table 9
(optional)	VLAN Tag	BYTE[4]	See Table 9
	Ether Type	BYTE[2]	0x08, 0x00 (IP)
IP	VersionHL	BYTE	0x45 (IP Version(4) header length (5*4 octets))
	Service	BYTE	0x00 (IP Type of service)
	TotalLength	Unsigned16	(IP total length of service) - not checked within Type 12 segment
	Identification	Unsigned16	(IP identification packet for fragmented service) - not checked within Type 12 segment
	Flags	BYTE	(IP flags – they will not be considered but a fragmentation of Type 12 frame will result in an error) - not checked within Type 12 segment
	Fragments	BYTE	(IP fragment number - fragmentation of Type 12 frame will result in an error) - not checked within Type 12 segment
	Ttl	BYTE	(IP time to live – only checked at routers) - not checked within Type 12 segment
	Protocol	BYTE	0x11 (IP sub-protocol – this value is reserved for UDP)
	Header checksum	Unsigned16	(IP header checksum) - not checked within Type 12 segment
	Source IP address	BYTE[4]	(IP source address of the originator) - not checked within Type 12 segment
	Destination IP address	BYTE[4]	(IP destination address of the target of the frame – within a Type 12 segment usually a multicast address as an individual address requires the Address Resolution Protocol ARP) - not checked within Type 12 segment
UDP	Src port	WORD	(UDP Source Port) - not checked within Type 12 segment
	Dest port	WORD	0x88A4 (UDP Source Port)
	Length	WORD	(UDP length of frame)) - not checked within Type 12

Frame part	Data field	Data type	Value/description
			segment
	Checksum	WORD	(UDP checksum of frame) – will be set to 0 for Type 12 frames but without checking
	Type 12 frame		specified in 5.3.3
	Padding	BYTE[n]	shall be inserted if DL PDU is shorter than 64 octets as specified in ISO/IEC 8802-3
Ethernet FCS	FCS	Unsigned32	Standard Ethernet Checksum coding as specified in ISO/IEC 8802-3

NOTE 1 IP packet structure and coding requirements are as specified in IETF RFC 791.

NOTE 2 The ordering of octets in multi-octet values is encoded differently in IETF protocols (see IETF RFC 768 and RFC 791) than it is within the Type 12 DL-protocol.

5.3.3 Type 12 frame structure

The Type 12 frame structure in shall consist one of the structures specified in Table 11, Table 12 and Table 13.

Table 11 – Type 12 frame structure containing Type 12 PDUs

Frame part	Data field	Data type	Value/description
Type 12 Frame	Length	unsigned11	Length of this frame (minus 2 octets)
	Reserved	unsigned 1	0
	Type	unsigned4	Protocol Type = Type 12 DLPDUs (0x01)
	Type 12 PDU 1		specified in 5.4
	…		specified in 5.4
	Type 12 PDU n		specified in 5.4

Table 12 – Type 12 frame structure containing network variables

Frame part	Data field	Data type	Value/description
Type 12 frame	Length	unsigned11	Length of this frame (minus 2 octets)
	reserved	unsigned 1	0
	Type	unsigned4	Protocol type = network variables (0x04)
Publisher header	PubID	BYTE[6]	Publisher ID
	CntNV	Unsigned16	Number of Network variables contained in this Type 12 frame
	CYC	Unsigned16	Cycle Number of the publisher side
	reserved	BYTE[2]	0x00, 0x00
	Network variable 1		Specified in 5.5
	…		Specified in 5.5
	Network variable n		Specified in 5.5

Table 13 – Type 12 frame structure containing mailbox

Frame part	Data field	Data type	Value/description
Type 12 frame	Length	unsigned11	Length of this frame (minus 2 octets)
	reserved	unsigned 1	0
	Type	unsigned4	Protocol type = mailbox (0x05)
	Mailbox		Specified in 5.6

5.4 Type 12 DLPDU structure

5.4.1 Read

5.4.1.1 Overview

With the read services a master reads data to memory of one or many slaves. The working counter shall be incremented by each slave if at least one of the addressed attribute is present.

5.4.1.2 Auto increment physical read (APRD)

The auto increment physical read (APRD) coding is specified in Table 14. Each slave increments the address ADP. The slave that receives an auto-increment address with value zero executes the requested read operation.

Table 14 – Auto increment physical read (APRD)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x01 (command APRD)
	IDX	Unsigned8	Index
	ADP	WORD	Auto increment address
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service and shall not be changed by the slave.

ADP

Each slave shall increment this parameter and the slave that receives this parameter with a value of zero shall perform the read access.

NOTE That means, the parameter contains the negative position of the slave in the logical loop beginning with 0 at the master side (e.g. –7 means that seven slaves are between the master and the addressed slave). At the confirmation this parameter contains the value of the request incremented by the number of transited slave devices.

ADO

This parameter shall contain the start address in the physical memory of the slave where the data to be read is stored

LEN

This parameter shall contain the size in octets of the data to be read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the read data if the access is valid at the addressed slaves site. Otherwise the value send out with the request remains unchanged.

WKC

This parameter shall be incremented by one if the data was successfully read.

5.4.1.3 Configured address physical read (FPRD)

The configured address physical read (FPRD) coding is specified in Table 15.

Table 15 – Configured address physical read (FPRD)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x04 (command FPRD)
	IDX	Unsigned8	Index
	ADP	WORD	Configured station address or configured station alias
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

The slave which has the value of D_address as station address or station address alias shall execute a read action.

ADO

This parameter shall contain the start address in the physical memory of the slave where the data to be read is stored.

LEN

This parameter shall contain the size in octets of the data to be read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the read data if the access is valid at the addressed slaves site. Otherwise the value send out with the request remains unchanged.

WKC

This parameter shall be incremented by one if the data was successfully read.

5.4.1.4 Broadcast read (BRD)

The broadcast read (BRD) coding is specified in Table 16.

Table 16 – Broadcast read (BRD)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x07 (command BRD)
	IDX	Unsigned8	Index
	ADP	WORD	Parameter incremented by 1 at each station forwarding BRD PDU
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

This parameter shall be incremented by one at each slave.

ADO

This parameter shall contain the start address in the physical memory where the data to be read is stored. Each slave who supports the requested physical memory area (physical memory address and length) shall respond to this service.

LEN

This parameter shall contain the size in octets of the data to be read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the read data collected before entry or default values of the master. This parameter shall contain the result of the bitwise-OR operation between the parameter data of the request and the addressed data in the slave.

WKC

This parameter shall be incremented by one by all slaves which made the bitwise-OR of the requested data.

5.4.1.5 Logical read (LRD)

The logical read (LRD) coding is specified in Table 17. The slave copies only data to the parameter data that are mapped by an FMMU entity from the logical address space to a physical address.

Table 17 – Logical read (LRD)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x0A (command LRD)
	IDX	Unsigned8	Index
	ADR	DWORD	Logical address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	reserved for future use
	DATA	OctetString LEN	Data, structure as specified by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADR

This parameter shall contain the start address in the logical memory where the data to be read is located. All slaves which have one or more address matches of the requested logical memory area (logical memory address and length) in their FMMU entities shall map the requested data to the data parameter as described by the FMMU entity settings and increment the working counter.

LEN

This parameter shall contain the size in octets of the data to be read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

On confirm this parameter specifies the data read from the device. Each slave which detects an address match of the requested logical memory area puts the data of the corresponding physical memory area in the correct part of this parameter.

WKC

This parameter shall be incremented by one by all slaves which detect an address match of the requested logical memory area.

5.4.2 Write

5.4.2.1 Overview

With the write services a master writes data to register or memory of one or many slaves. The working counter is incremented if the addressed attribute is present. The working counter can be incremented by one if at least one part of the data can be written.

5.4.2.2 Auto increment physical write (APWR)

The auto increment physical write (APWR) coding is specified in Table 18. Each slave increments the address. The slave that receives a zero value at auto-increment address parameter will execute the requested write operation.

Table 18 – Auto Increment physical write (APWR)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x02 (command APWR)
	IDX	Unsigned8	Index
	ADP	WORD	Auto increment address
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

The slave will be addressed by its position in the segment. Each slave shall increment this parameter, the slave who receives the value zero of this parameter shall respond to this service.

NOTE That means, the parameter contains the negative position of the slave in the logical ring beginning with 0 at the master side (e.g. -7 means 7 slaves are between master and the addressed slave). At the confirmation this parameter contains the value of the request incremented by the number of transited slave devices.

ADO

This parameter shall contain the start address in the physical memory of the slave where the data to be written is stored.

LEN

This parameter shall contain the size in octets of the data to be written.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written.

WKC

This parameter shall be incremented by one if the data can be successfully written.

5.4.2.3 Configured address physical write (FPWR)

The configured address physical write (FPWR) coding is specified in Table 19.

Table 19 – Configured address physical write (FPWR)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x05 (command FPWR)
	IDX	Unsigned8	Index
	ADP	WORD	Configured station address or configured station alias
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

The slave which has the value of D_address as station address or station address alias shall execute a write action.

ADO

This parameter shall contain the start address in the physical memory of the slave where the data to be written is stored.

LEN

This parameter shall contain the size in octets of the data to be written.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written.

WKC

This parameter shall be incremented by one if the data was successfully written.

5.4.2.4 Broadcast write (BWR)

The broadcast write (BWR) coding is specified in Table 20.

Table 20 – Broadcast write (BWR)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x08 (command BWR)
	IDX	Unsigned8	Index
	ADP	WORD	Parameter incremented by 1 at each station forwarding BWR PDU
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

This parameter shall be incremented by one at each slave.

ADO

This parameter shall contain the start address in the physical memory where the data to be written is stored. Each slave who supports the requested physical memory area (physical memory address and length) shall respond to this service.

LEN

This parameter shall contain the size in octets of the data to be written.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written.

WKC

This parameter shall be incremented by one by all slaves which write data in their physical memory.

5.4.2.5 Logical write (LWR)

The logical write (LWR) coding is specified in Table 21. The slave copies only data to the memory or register that are mapped by an FMMU entity from the logical address space to a physical address.

Table 21 – Logical write (LWR)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x0B (command LWR)
	IDX	Unsigned8	Index
	ADR	DWORD	Logical address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	reserved for future use
	DATA	OctetString LEN	Data, structure as specified by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADR

This parameter shall contain the start address in the logical memory where the data to be written is located. All slaves which have one or more address matches of the requested logical memory area (logical memory address and length) in their FMMUs shall respond to this service.

LEN

This parameter shall contain the size in octets of the data to be written.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written. Each slave which detects an address match of the requested logical memory area will put the data of the correct part of this parameter in the corresponding physical memory area.

WKC

This parameter shall be incremented by one by all slaves who detect an address match of the requested logical memory area and if the data was successfully written

5.4.3 Read write

5.4.3.1 Auto increment physical read write (APRW)

The optional auto increment physical read write (APRW) coding is specified in Table 22. Each slave increments the address. The slave that receives a zero value at auto-increment address parameter will execute the requested operation.

Table 22 – Auto increment physical read write (APRW)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x03 (command APRW)
	IDX	Unsigned8	Index
	ADP	WORD	Auto increment address
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

The slave will be addressed by its position in the segment. Each slave shall increment this parameter, the slave who receives the value zero of this parameter shall respond to this service.

NOTE That means, the parameter contains the negative position of the slave in the logical ring beginning with 0 at the master side (e.g. -7 means 7 slaves are between master and the addressed slave). At the confirmation this parameter contains the value of the request incremented by the number of transited slave devices.

ADO

This parameter shall contain the start address in the physical memory of the slave where data to be read and written is stored.

LEN

This parameter shall contain the size in octets of the data to be written and read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written and the data read from the addressed slave if the service can be executed successfully.

WKC

This parameter shall be incremented by two if the data was successfully written and additionally by one if the data was successfully read.

5.4.3.2 Configured address physical read write (FPRW)

The optional configured address physical read write (FPRW) coding is specified in Table 23.

Table 23 – Configured address physical read write (FPRW)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x06 (command FPRW)
	IDX	Unsigned8	Index
	ADP	WORD	Configured station address or configured station alias
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

The slave which has the value of D_address as station address or station address alias shall execute a read action followed by a write action.

ADO

This parameter shall contain the start address in the physical memory of the slave where the data to be read and written is stored.

LEN

This parameter shall contain the size in octets of the data to be written and read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written and the data read from the addressed slave if the service can be executed successfully.

WKC

This parameter shall be incremented by two if the data was successfully written and additionally by one if the data was successfully read.

5.4.3.3 Broadcast read write (BRW)

The optional broadcast read write (BRW) coding is specified in Table 24.

Table 24 – Broadcast read write (BRW)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x09 (command BRW)
	IDX	Unsigned8	Index
	ADP	WORD	Parameter incremented by 1 at each station forwarding BRW PDU
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working Counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

This parameter shall be incremented by one at each slave.

ADO

This parameter shall contain the start address in the physical memory where the data to be read and written is stored. Each slave who supports the requested physical memory area (physical memory address and length) shall respond to this service.

LEN

This parameter shall contain the size in octets of the data to be written and read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data before entry and will be the written. A read operation is performed before write. This parameter shall contain the result of the bitwise-OR operation between the parameter data of the request and the addressed data in the slave.

WKC

This parameter shall be incremented by two by all slaves if the data was successfully written and additionally by one by all slaves which made the bitwise-OR of the requested data.

5.4.3.4 Logical read write (LRW)

The optional logical read write (LRW) coding is specified in Table 25. A slave device can retrieve data with this service (write operation) and put data with this service (read operation). The slave will copy in or out only data to or from the parameter data that are mapped by an FMMU entity from the logical address space to a physical address. . It is highly recommended to support this command for better system performance.

Table 25 – Logical read write (LRW)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x0C (command LRW)
	IDX	Unsigned8	Index
	ADR	DWORD	Logical address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	Reserved for future use
	DATA	OctetString LEN	Data, structure as specified by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADR

This parameter shall contain the start address in the logical memory where the data to be read or written is located. All slaves which have one or more address matches of the requested logical memory area (logical memory address and length) in their FMMU shall respond to this service.

LEN

This parameter shall contain the size in octets of the data to be written and read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written. Each slave who detects an address match of the requested logical memory area will put the data of the correct part of this parameter in the corresponding physical memory area. With the confirmation this parameter shall contain the read data. Each slave who detects an address match of the addressed logical memory area will put the data of the corresponding physical memory area in the correct part of this parameter.

WKC

This parameter shall be incremented by each slave by two if a piece of data was successfully written and additional incremented by one if a piece of data was successfully read.

5.4.3.5 Auto increment physical read multiple write (ARMW)

The auto increment physical read multiple write (ARMW) coding is specified in Table 26.

Table 26 – Auto increment physical read multiple write (ARMW)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x0D (command ARMW)
	IDX	Unsigned8	Index
	ADP	WORD	Auto increment or register address
	ADO	WORD	Physical memory or register address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

The slave will be addressed by its position in the segment. Each slave shall increment this parameter, the slave who receives the value zero of this parameter shall execute a read action – the other slaves shall execute a write action.

NOTE That means, the parameter contains the negative position of the slave in the logical ring beginning with 0 at the master side (e.g. -7 means 7 slaves are between master and the addressed slave). At the confirmation this parameter contains the value of the request incremented by the number of transited slave devices.

ADO

This parameter shall contain the start address in the physical memory of the slave where data to be read and written is stored.

LEN

This parameter shall contain the size in octets of the data to be written and read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written and the data read from the addressed slave if the service can be executed successfully.

WKC

This parameter shall be incremented by one by each slave if the data was successfully read or written.

5.4.3.6 Configured address physical read multiple write (FRMW)

The configured address physical read multiple write (FRMW) coding is specified in Table 27.

Table 27 – Configured address physical read multiple write (FRMW)

Frame part	Data field	Data type	Value/description
Type 12 PDU	CMD	Unsigned8	0x0E (command FRMW)
	IDX	Unsigned8	Index
	ADP	WORD	Configured station address or configured station alias
	ADO	WORD	Physical memory address
	LEN	Unsigned11	Length of the DATA data field
	reserved	Unsigned3	0x00
	C	Unsigned1	Circulating frame 0: Frame is not circulating, 1: Frame has circulated once
	NEXT	Unsigned1	0x00: last Type 12 PDU in Type 12 frame 0x01: Type 12 PDU in Type 12 frame follows
	IRQ	WORD	External event
	DATA	OctetString LEN	Data, structure as specified in 5.6, Clause 6 or by DLS-user
	WKC	WORD	Working counter

CMD

The parameter Command shall contain the service command.

IDX

The parameter Index is the local identifier in the master of the service; it shall not be changed by the slave.

ADP

The slave which has the value of D_address as station address or station address alias shall execute a read action - the other slaves shall execute a write action.

ADO

This parameter shall contain the start address in the physical memory of the slave where data to be read and written is stored.

LEN

This parameter shall contain the size in octets of the data to be written and read.

C

This Parameter shall indicate that the frame has circulated in the network and shall not be forwarded.

NEXT

This parameter shall specify if there is another Type 12 PDU in the frame.

IRQ

This parameter shall contain the External event (see Table 38) masked by the External event mask (see Table 39).

DATA

This parameter shall contain the data to be written and the data read from the addressed slave if the service can be executed successfully.

WKC

This parameter shall be incremented by one by all slaves if the data was successfully read or written.

5.5 Network variable structure

The network variable coding is specified in Table 28.

Table 28 – Network variable

Frame part	Data field	Data type	Value/description
Network variable	Index	Unsigned16	Index to a DLS-user object
	HASH	Unsigned16	Hash algorithm over the data structure of the data to detect changes
	LEN	Unsigned16	Length
	Q	Unsigned16	Quality
	DATA	OctetString [LEN]	Data, structure as specified by DLS-user

5.6 Type 12 mailbox structure

The mailbox coding is specified in Table 29. The mailbox encoding shall be used in conjunction with Type 12 mailbox memory elements or as coding for data structures conveying mailboxes via Ethernet DL or via IP.

Table 29 – Mailbox

Frame part	Data field	Data type	Value/description
Mailbox	Length	Unsigned16	Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client or data are transmitted outside the target Type 12 segment
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority ... 0x03: highest priority
	Type	Unsigned4	0x00: error(ERR) 0x01: reserved 0x02: Ethernet over Type 12 (EoE) 0x03: CAN application protocol over Type 12 (CoE) 0x04: File Access over Type 12 (FoE) 0x05: Servo Drive profile over Type 12 (SoE) 0x06 -0x0e: reserved 0x0f: vendor specific
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1. The Slave shall increment the Cnt value for each new mailbox service, the Master shall check this for detection of lost mailbox services. The Master shall change (should increment) the Cnt value. The slave shall check this for detection of a write repeat service. The Slave shall not check the sequence of the Cnt value. The master and the slave Cnt values are independent
	reserved	Unsigned1	0x00
	Service Data	OctetString [Length]	Mailbox Service Data

The encoding of Service Data in case of an error reply is specified in Table 30.

Table 30 – Error Reply Service Data

Frame part	Data field	Data type	Value/description
Service Data	Type	Unsigned16	0x01: Mailbox Command
	Detail	Unsigned16	0x01: MBXERR_SYNTAX Syntax of 6 octet Mailbox Header is wrong 0x02: MBXERR_UNSUPPORTEDPROTOCOL The Mailbox protocol is not supported 0x03: MBXERR_INVALIDCHANNEL Channel Field contains wrong value (a slave can ignore the channel field) 0x04: MBXERR_SERVICENOTSUPPORTED the service in the Mailbox protocol is not supported 0x05: MBXERR_INVALIDHEADER The mailbox protocol header of the mailbox protocol is wrong (without the 6 octet mailbox header) 0x06: MBXERR_SIZETOOSHORT

Frame part	Data field	Data type	Value/description
			length of received mailbox data is too short 0x07: MBXERR_NOMOREMEMORY Mailbox protocol cannot be processed because of limited ressources 0x08: MBXERR_INVALIDSIZE the length of data is inconsistent

6 Attributes

6.1 Management

6.1.1 DL Information

The DL information registers contain type, version and supported resources of the slave controller (ESC).

Parameter

Type

This parameter shall contain the type of the slave controller.

Revision (major revision)

This parameter shall contain the revision of the slave controller.

Build (minor revision)

This parameter shall contain the build number of the slave controller.

Number of supported FMMU entities

This parameter shall contain the number of supported FMMU entities of the slave controller.

Number of supported sync manager channels

This parameter shall contain the number of supported sync manager channels (or entities) of the slave controller.

RAM size

This parameter shall contain the RAM size in Kbyte supported by the slave controller (smaller size than an even number will be rounded down).

Port descriptor

Port 0 Physical Layer

This parameter should indicate the physical layer used for this port.

Port 1 Physical Layer

This parameter should indicate the physical layer used for this port.

Port 2 Physical Layer

This parameter should indicate the physical layer used for this port.

Port 3 Physical Layer

This parameter should indicate the physical layer used for this port.

Features supported

FMMU bit operation not supported

This parameter shall indicate whether the FMMU in the slave controller supports bit operations operations without restrictions or with documented restrictions (e.g. only bitwise mapping on specific memory areas).

This feature bit does not affect mappability of SM.WriteEvent flag (MailboxIn)

DC supported

This parameter is set to 1 if at least distributed clock receive times are supported.

DC range

This parameter shall indicate the clock value range (0: 32 bit/1:64 bit).

Low jitter EBUS

This parameter shall indicate that the low jitter feature is available.

Enhanced link detection EBUS

This parameter shall indicate that the enhanced link detection is availablefor EBUS ports.

Enhanced link detection MII

This parameter shall indicate that the enhanced link detection is availablefor MII ports.

Separate Handling of FCS errors

This parameter shall indicate that the errors induced by another type 12 slave will be counted separately.

Enhanced DC Sync Activation

This parameter shall indicate that enhanced DC Sync Activation is available.

LRW not supported

This parameter shall indicate that LRW is not supported.

BRW, APRW, FPRW not supported

This parameter shall indicate that BRW, APRW, FPRW is not supported.

Special FMMU/Sync manager configuration

This parameter shall indicate that a special FMMU/Sync manager configuration is used:

FMMU 0 is used for RxPDO (no bit mapping)

FMMU 1 is used for TxPDO (no bit mapping)

FMMU 2 is used for Mailbox write event bit of Sync manager 1 (FMMU bit operation is supported for this bit)

Sync manager 0 is used for write mailbox

Sync manager 1 is used for read mailbox

Sync manager 2 is used as Buffer for RxPDO

Sync manager 3 is used as Buffer for TxPDO

The attribute types of DL information are described in Figure 8.

```
typedef struct
{
  BYTE          Type;
  BYTE          Revision;
  WORD          Build;
  BYTE          NoOfSuppFmmuEntities;
  BYTE          NoOfSuppSyncManChannels;
  BYTE          RamSize;
  BYTE          PortDescr;
  unsigned      FmmuBitOperationNotSupp:  1;
  unsigned      Reserved2:                1;
  unsigned      DCSupp:                   1;
  unsigned      DCRange:                  1;
  unsigned      LowJEBUS:                 1;
  unsigned      EnhLDEBUS:                1;
  unsigned      EnhLDMII:                 1;
  unsigned      FCSsERR:                  1;
  unsigned      EnhancedDcSyncAct:        1;
  unsigned      NotSuppLRW:               1;
  unsigned      NotSuppBAFRW:             1;
  unsigned      sFMMUSyMC:                1;
  unsigned      Reserved4:                4;
} TDLINFORMATION;
```

Figure 8 – DL information type description

The DL Information coding is specified in Table 31.

Table 31 – DL information

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Type	0x0000	BYTE	R	R	
Revision	0x0001	BYTE	R	R	
Build	0x0002	WORD	R	R	
Number of supported FMMU entities	0x0004	BYTE	R	R	0x0001-0x0010
Number of supported Sync Manager channels	0x0005	BYTE	R	R	0x0001-0x0010
RAM Size	0x0006	BYTE	R	R	RAM size in koctet means 1024 octets (1-60)
Port0 Descriptor	0x0007	unsigned2	R	R	optional 00: Not implemented 01: Not configured 10: EBUS 11: MII/RMII
Port1 Descriptor	0x0007	unsigned2	R	R	optional 00: Not implemented 01: Not configured 10: EBUS 11: MII/RMII
Port2 Descriptor	0x0007	unsigned2	R	R	optional 00: Not implemented 01: Not configured 10: EBUS 11: MII/RMII
Port3 Descriptor	0x0007	unsigned2	R	R	optional 00: Not implemented 01: Not configured 10: EBUS 11: MII/RMII
FMMU Bit Operation Not Supported	0x0008	unsigned1	R	R	0: bit operation supported 1: bit operation not

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
					supported This feature bit does not affect mappability of SM.WriteEvent flag (MailboxIn)
Reserved	0x0008	unsigned1	R	R	
DC Supported	0x0008	unsigned1	R	R	0: DC not supported 1: DC supported
DC Range	0x0008	unsigned1	R	R	0: 32 bit 1: 64 bit for system time, system time offset and receive time time processing unit
Low Jitter EBUS	0x0008	unsigned1	R	R	0: not available 1: available
Enhanced Link Detection EBUS	0x0008	unsigned1	R	R	0: not available 1: available
Enhanced Link Detection MII	0x0008	unsigned1	R	R	0: not available 1: available
Separate Handling of FCS errors	0x0008	unsigned1	R	R	0: not active 1: active, Frames with modified FCS (additional nibble) should be counted separately in RX-Error Previous counter
Enhanced DC Sync Activation	0x0009	unsigned1	R	R	0: not available 1: available This feature refers to registers 0x981[7:3], 0x0984
LRW not supported	0x0009	unsigned1	R	R	0: LRW supported 1: LRW not supported
BRW, APRW; FPRW not supported	0x0009	unsigned1	R	R	0: BRW, APRW; FPRW supported 1: BRW, APRW; FPRW not supported
Special FMMU Syc manager configuration	0x0009	unsigned1	R	R	0: not active 1: active, FMMU 0 is used for RxPDO (no bit mapping) FMMU 1 is used for TxPDO (no bit mapping) FMMU 2 is used for Mailbox write event bit of Sync manager 1 Sync manager 0 is used for write mailbox Sync manager 1 is used for read mailbox Sync manager 2 is used as Buffer for incoming data Sync manager 3 is used as Buffer for outgoing data
Reserved	0x0009	unsigned4	R	R	

6.1.2 Station address

The configured station address register contains the station address of the slave which will be set to activate the FPRD, FPRW, FRMW and FPWR service in the slave controller.

Parameter

Configured station address

This parameter shall contain the configured station address of the slave controller which is set up by the master at start up.

Configured station alias

This parameter shall contain the configured station alias of the slave controller which is set up by DL-user at start up.

The attribute types of station address are described in Figure 9

```
typedef struct
{
  WORD          ConfiguredStationAddress;
  WORD          ConfiguredStationAlias;
} TFIXEDSTATIONADDRESS;
```

Figure 9 – Address type description

The station address coding is specified in Table 32.

Table 32 – Configured station address

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Configured Station Address	0x0010	WORD	RW	R	
Configured Station Alias	0x0012	WORD	RW	RW	Initialized with SII word 4

6.1.3 DL control

The DL control register is used to control the operation of the DL ports of the slave controller by the master.

Parameter

Forwarding rule

This parameter shall enable direct forwarding or restricted forwarding. Restricted forwarding will destroy non Type 12 frames.

Temporary loop control

This optional parameter enables temporary use of the loop control parameters written in the same frame for about one second. After this timeout, the original Loop control settings are restored automatically.

Loop control port 0

This parameter shall contain the information if there is an automatic activation of the port in case of a physical link or if the port is opened and or closed by commands of the master.

Loop control port 1

This parameter shall contain the information if there is an automatic activation of the port in case of a physical link or if the port is opened and or closed by commands of the master.

Loop control port 2

This parameter shall contain the information if there is an automatic activation of the port in case of a physical link or if the port is opened and or closed by commands of the master.

Loop control port 3

This parameter shall contain the information if there is an automatic activation of the port in case of a physical link or if the port is opened and or closed by commands of the master.

Transmit buffer size

This optional parameter should be used to optimize the delay within a station. If this station and its neighbours have a stable rate of transmitting, this parameter may be reduced. The default settings are determined by the required clock accuracy of ISO/IEC 8802-3.

Low jitter EBUS

This optional parameter indicates that the reduction of frame forwarding jitter for EBUS is enabled.

Enable alias address

This optional parameter should be used to enable the alias name.

The attribute types of DL Control are described in Figure 10.

```
typedef struct
{
  unsigned      ForwardingRule:         1;
  unsigned      TemporaryLoopControl:   1;
  unsigned      Reserved0:              6;
  unsigned      LoopControlPort0:       2;
  unsigned      LoopControlPort1:       2;
  unsigned      LoopControlPort2:       2;
  unsigned      LoopControlPort3:       2;
  unsigned      TxBufferSize:           3;
  unsigned      LowJitterEBUS:          1;
  unsigned      Reserved1:              4;
  unsigned      EnableAliasAddress:     1;
  unsigned      Reserved2:              7;
} TDLCONTROL;
```

Figure 10 – DL control type description

The DL Control coding is specified in Table 33.

Table 33 – DL control

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Forwarding rule	0x0100	Unsigned1	RW	R	0: EtherCAT frames are processed, Non-EtherCAT frames are forwarded without processing, SOURCE_MAC[1] may be set to 1 – locally administered address 1: EtherCAT frames are processed, Non-EtherCAT frames are destroyed, SOURCE_MAC[1] shall be set to 1 – locally administered address
Temporary Loop control	0x0100	Unsigned1	RW	R	0:permanent setting 1: temporary use of Loop Control Settings for ~1 second
reserved	0x0100	Unsigned6	RW	R	0x00

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Loop control port 0	0x0101	Unsigned2	RW	R	0: Auto => closed at “link down”, open with “link up” 1: Auto close => closed at “link down”, open with writing 1 after “link up” (or receiving a valid Ethernet frame at the closed port) 2: Always open 3: Always closed
Loop control port 1	0x0101	Unsigned2	RW	R	0: Auto => closed at “link down”, open with “link up” 1: Auto close => closed at “link down”, open with writing 1 1 after “link up” (or receiving a valid Ethernet frame at the closed port) 2: Always open 3: Always closed
Loop control port 2	0x0101	Unsigned2	RW	R	0: Auto => closed at “link down”, open with “link up” 1: Auto close => closed at “link down”, open with writing 1 1 after “link up” (or receiving a valid Ethernet frame at the closed port) 2: Always open 3: Always closed
Loop control port 3	0x0101	Unsigned2	RW	R	0: Auto => closed at “link down”, open with “link up” 1: Auto close => closed at “link down”, open with writing 1 1 after “link up” (or receiving a valid Ethernet frame at the closed port) 2: Always open 3: Always closed
TransmitBufferSize	0x0102	Unsigned3	RW	R	Buffer between preparation and send. Send will be if buffer is half full (7).
Low Jitter EBUS	0x0102	Unsigned1	RW	R	0: not active 1: active
reserved	0x0102	Unsigned4	RW	R	0x00
EnableAliasAddress	0x0103	Unsigned1	RW	R	0: Disable the station alias address 1: Enable the station alias address
reserved	0x0103	Unsigned7	RW	R	0x00
NOTE Loop open means sending over this port and waiting for a reaction at the receiving port is enabled – the received data will be forwarded to the peer port. Loop closed means that data, that should be forwarded are directly mirrored and thus they will be forwarded to the peer port. A closed port will discard all received data.					

6.1.4 DL status

The DL status register is used to indicate the state of the DL ports and the state of the interface between DL-user and DL.

Parameter

DL-user operational

This parameter shall contain the information if a DL-user is connected to the process data interface of the slave controller.

DL-user watchdog status

This parameter shall contain the status of the process data interface watchdog.

Extended link detection

This parameter shall contain the status of the activation of the extended link detection.

Link status port 0

This parameter indicates physical link on this port.

Link status port 1

This parameter indicates physical link on this port.

Link status port 2

This parameter indicates physical link on this port.

Link status port 3

This parameter indicates physical link on this port.

Loop back port 0

This parameter indicates forwarding on the same port i.e. loop back.

Signal detection port 0

This parameter indicates if there is a signal detected on Rx-Port.

Loop back port 1

This parameter indicates forwarding on the same port i.e. loop back.

Signal detection port 1

This parameter indicates if there is a signal detected on Rx-Port.

Loop back port 2

This parameter indicates forwarding on the same port i.e. loop back.

Signal detection port 2

This parameter indicates if there is a signal detected on Rx-Port.

Loop back port 3

This parameter indicates forwarding on the same port i.e. loop back.

Signal detection port 3

This parameter indicates if there is a signal detected on Rx-Port.

The attribute types of DL Status are described in Figure 11.

```
typedef struct
{
  unsigned      PdiOperational:           1;
  unsigned      DLSuserWatchdogStatus:        1;
  unsigned      ExtendedLinkDetection:    1;
  unsigned      Reserved1:                1;
  unsigned      LinkStatusPort0:          1;
  unsigned      LinkStatusPort1:          1;
  unsigned      LinkStatusPort2:          1;
  unsigned      LinkStatusPort3:          1;
  unsigned      LoopStatusPort0:          1;
  unsigned      SignalDetectionPort0:     1;
  unsigned      LoopStatusPort1:          1;
  unsigned      SignalDetectionPort1:     1;
  unsigned      LoopStatusPort2:          1;
  unsigned      SignalDetectionPort2:     1;
  unsigned      LoopStatusPort3:          1;
  unsigned      SignalDetectionPort3:     1;
} TDLSTATUS;
```

Figure 11 – DL status type description

The DL Status coding is specified in Table 34.

Table 34 – DL status

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
DLS-user operational	0x0110	Unsigned1	R	R	0x00: DLS-user not operational 0x01: DLS-user operational
DLS-user watchdog status	0x0110	Unsigned1	R	R	0x00: DLS-user watchdog expired 0x01: DLS-user watchdog not expired
Extended link detection	0x0110	Unsigned1	R	R	0: Deactivated 1: Activated for at least one port
Reserved	0x0110	Unsigned1	R	R	0x00
Link status port 0	0x0110	Unsigned1	R	R	0x00: no physical link on this port 0x01: physical link on this port
Link status port 1	0x0110	Unsigned1	R	R	0x00: no physical link on this port 0x01: physical link on this port
Link status port 2	0x0110	Unsigned1	R	R	0x00: no physical link on this port 0x01: physical link on this port
Link status port 3	0x0110	Unsigned1	R	R	0x00: no physical link on this port 0x01: physical link on this port
Loop status port 0	0x0111	Unsigned1	R	R	0x00: loop not active 0x01: loop active
Signal detection port 0	0x0111	Unsigned1	R	R	0x00: signal not detected on RX-port

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
					0x01: signal detected on RX-port
Loop status port 1	0x0111	Unsigned1	R	R	0x00: loop not active 0x01: loop active
Signal detection port 1	0x0111	Unsigned1	R	R	0x00: signal not detected on RX-port 0x01: signal detected on RX-port
Loop status port 2	0x0111	Unsigned1	R	R	0x00: loop not active 0x01: loop active
Signal detection port 2	0x0111	Unsigned1	R	R	0x00: signal not detected on RX-port 0x01: signal detected on RX-port
Loop status port 3	0x0111	Unsigned1	R	R	0x00: loop not active 0x01: loop active
Signal detection port 3	0x0111	Unsigned1	R	R	0x00: signal not detected on RX-port 0x01: signal detected on RX-port

6.1.5 DLS-user specific registers

6.1.5.1 DL-user control register

Figure 12 shows the primitives between master, DL and DL-user in case of a successful write sequence to the DL-user control register (R1).

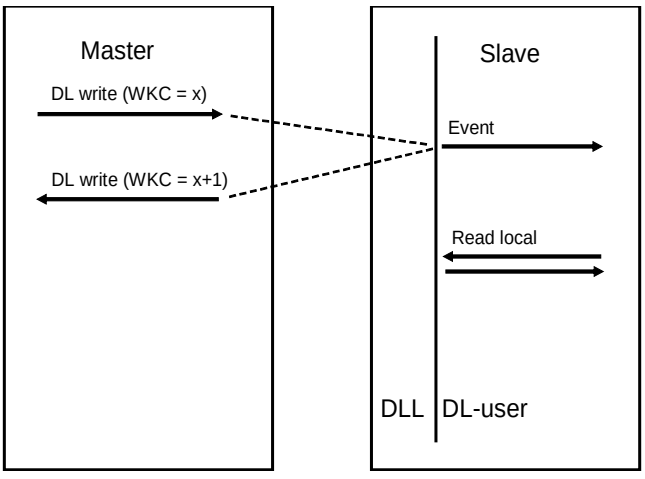


Figure 12 – Successful write sequence to DL-user control register

The master sends a write service with the working counter (WKC = x), the DL (slave controller) of the slave write the received data in the register area, increments the working counter (WKC = x + 1) and generates an event and the DL-user reads the control register. If the control register is not read out, the next write to this register will be ignored (is not changed).

The control register is used to pass control information from the master to the slave.

6.1.5.2 DL-user status register

Figure 13 shows the primitives between master, DL and DL-user in case of a successful read sequence to the DL-user status register (R3).

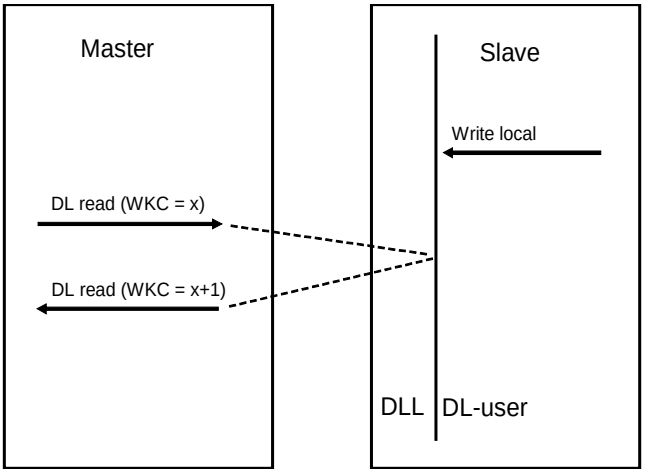


Figure 13 – Successful read sequence to the DL-user status register

The DL-user of the slave writes the DL-user status register locally. The master sends a read service with the working counter (WKC = x), the DL (slave controller) of the slave sends the data from the register area and increments the working counter (WKC = x + 1).

6.1.5.3 DL-user specific registers

There is a set of DL-user specific registers R2, R4 to R8. The meaning of the contents is defined by DL-user.

6.1.5.4 DL-user attributes

The DLS-user specific register structure and access type is described in Table 35.

Table 35 – DLS-user specific registers

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
DLS-user R1	0x0120	Unsigned8	RW	R	0x01
DLS-user R2	0x0121	Unsigned8	RW	R	0x00
DLS-user R3	0x0130	Unsigned8	R	RW	0x01
DLS-user R4	0x0131	Unsigned8	R	RW	0x00
reserved	0x0132	Unsigned16			
DLS-user R6	0x0134	Unsigned16	R	RW	0x00
DLS-user R7	0x0140	Unsigned8	R	R	0x00
Copy	0x0141	Unsigned1	R	R	0x00: no specific action 0x01: Copy DLS-user R1 to DLS-user R3
DLS-user R9	0x0141	Unsigned7	R	R	0x00
DLS-user R10	0x0142	Unsigned16	R	R	0x00
DLS-user R8	0x0150	Unsigned32	R	R	0x00

6.1.6 Event parameter

The event registers are used to indicate an event to the DL-user. The event shall be acknowledged if the corresponding event source is read. The events can be masked.

Parameter

DL-user Event

DL-user R1 Chg

This parameter is set if an write service to the DL-user control register is invoked and is reset when the DL-user control register is read local.

DC Event 0

This parameter is set if DC Event 0 is active.

DC Event 1

This parameter is set if DC Event 1 is active.

DC Event 2

This parameter is set if DC Event 2 is active.

Sync manager change event

This parameter is set if a write service to the Sync manager area occurs and will be reset if the DL-user reads out the event register.

Sync manager channel access events (0 to 15)

This parameter is set if an write service to an application memory area configured as write by the master or an read service to an application memory area configured as read by the master is received and will be reset when the application memory area will be read (read local) or written (write local).

DL-user Event Mask

If the corresponding attribute is set the DL-user event will be enabled and disabled otherwise.

External Event

DC Event 0

This parameter is set if DC Event 0 is active.

DL Status Chg

This parameter is set if the DL Status register is changed and is reset when a Type 12 read to DL Status register is invoked.

DL-user R3 Chg

This parameter is set if a write local service to the DL-user status register is invoked and is reset when a Type 12 read to DL-user status register is invoked.

Sync manager channel access events (0 to 7)

This parameter is set if an write service to an application memory area configured as write by the slave or an read service to an application memory area configured as read by the slave is received and will be reset when the application memory area will be read or written by Type 12 services.

Event Event Mask

If the corresponding attribute is set the external event will be enabled and disabled otherwise.

The Event structure as seen by the DLS-user and access type is described in Table 36.

Table 36 – DLS-user event

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
DLS-user R1 chg	0x0220	unsigned1	R	r	0x00: no event active 0x01: event active R1 was written)
DC event 0	0x0220	unsigned1	R	r	0x00: no event active 0x01 event active (at least one latch event occurred)
DC event 1	0x0220	unsigned1	R	r	0x00: sync signal is not active 0x01 sync signal is active
DC event 2	0x0220	unsigned1	R	r	0x00: sync signal is not active 0x01 sync signal is active
Sync manager change event	0x0220	unsigned1	R	r	0x00: no event active 0x01: event active (one or more sync manager channels were changed)
EEPROM Emulation	0x0220	unsigned1	R	r	0x00: No command pending 0x01: EEPROM command pending
DLE specific	0x0220	Unsigned2	R	r	0x00:
Sync manager channel 0 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 1 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 2 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 3 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 4 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 5 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 6 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Sync manager channel 7 event	0x0221	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 8 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 9 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 10 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 11 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 12 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 13 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 14 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
Sync manager channel 15 event	0x0222	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed)
DLE specific	0x0223	Unsigned8	R	r	0x00

The DLS-user Event Mask is related to DLS-user Event and coding is specified Table 37.

Table 37 – DLS-user event mask

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Event mask	0x0204	array [0..31] of unsigned1	R	rw	For each element: 0: disable event 1: enable event

The Event structure as seen by remote partner and access type is described in Table 38. The external event is mapped to IRQ parameter of all Type 12 PDUs accessing this slave. If an event is set and the associated mask is set the corresponding bit in the IRQ parameter of a PDU is set.

Table 38 – External event

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
DC Event 0	0x0210	unsigned1	R	r	0x00: no event active 0x01: DC event 0 active
reserved	0x0210	unsigned1	R	r	0x00
DL Status change	0x0210	unsigned1	R	r	0x00: no event active 0x01: DL status register was changed active
R3 Chg	0x0210	unsigned1	R	r	0x00: no event active 0x01: event active (R3 was written)
Sync manager channel 0 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Sync manager channel 1 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Sync manager channel 2 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Sync manager channel 3 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Sync manager channel 4 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Sync manager channel 5 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Sync manager channel 6 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Sync manager channel 7 event	0x0211	unsigned1	R	r	0x00: no event active 0x01: event active (sync manager channel was accessed by slave)
Reserved	0x0211	Unsigned4	R	r	0x00

The External Event Mask is related to External Event and coding is specified in Table 39.

Table 39 – External event mask

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Event mask	0x0200	array [0..15] of unsigned1	RW	r	For each element: 0: disable event 1: enable event

6.2 Statistics

6.2.1 RX error counter

The RX error counter registers contain information about physical layer errors and frame errors (e.g. length, FCS). All counters will be cleared if one counter is written. The counting is stopped when the maximum value of a counter (255) is reached.

Parameter

Port 0 physical layer error count

This parameter counts the occurrences of RX errors at the physical layer.

Port 0 frame error count

This parameter counts the occurrences of frame errors (including RX errors within frame).

Port 1 physical layer error count

This parameter counts the occurrences of RX errors at the physical layer.

Port 1 frame error count

This parameter counts the occurrences of frame errors (including RX errors within frame).

Port 2 physical layer error count

This parameter counts the occurrences of RX errors at the physical layer.

Port 2 frame error count

This parameter counts the occurrences of frame errors (including RX errors within frame).

Port 3 physical layer error count

This parameter counts the occurrences of RX errors at the physical layer.

Port 3 frame error count

This parameter counts the occurrences of frame errors (including RX errors within frame).

NOTE The frames will be processed during forwarding procedure. Thus, an RX error or frame error will occur at any stations beyond the erroneous station simultaneously. The master will obtain a true picture by subtracting the counts of the previous port.

The attribute types of RX Error Counter are described in Figure 14.

```
typedef struct
{
  Unsigned8          FrameErrorCountPort0;
  Unsigned8          PhyErrorCountPort0;
  Unsigned8          FrameErrorCountPort1;
  Unsigned8          PhyErrorCountPort1;
  Unsigned8          FrameErrorCountPort2;
  Unsigned8          PhyErrorCountPort2;
  Unsigned8          FrameErrorCountPort3;
  Unsigned8          PhyErrorCountPort3;
} TRXERRORCOUNTER;
```

Figure 14 – RX error counter type description

The RX Error Counter coding is specified in Table 40.

Table 40 – RX error counter

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Frame error count port 0	0x0300	unsigned8	RW	--	A write to one counter will reset all counters
Physical error count port 0	0x0301	unsigned8	RW	--	A write to one counter will reset all counters
Frame error count port 1	0x0302	unsigned8	RW	--	A write to one counter will reset all counters
Physical error count port 1	0x0303	unsigned8	RW	--	A write to one counter will reset all counters
Frame error count port 2	0x0304	unsigned8	RW	--	A write to one counter will reset all counters
Physical error count port 2	0x0305	unsigned8	RW	--	A write to one counter will reset all counters
Frame error count port 3	0x0306	unsigned8	RW	--	A write to one counter will reset all counters
Physical error count port 3	0x0307	unsigned8	RW	--	A write to one counter will reset all counters

6.2.2 Lost link counter

The optional lost link counter registers contain information about link down sequences. All counters will be cleared if one counter is written. The counting is stopped when the maximum value of a counter (255) is reached.

Parameter

Port 0 lost link count

This parameter counts the occurrences of link down.

Port 1 lost link count

This parameter counts the occurrences of link down.

Port 2 lost link count

This parameter counts the occurrences of link down.

Port 3 lost link count

This parameter counts the occurrences of link down.

The attribute types of Lost Link Counter are described in Figure 15.

```
typedef struct
{
  Unsigned8          LostLinkCountPort0;
  Unsigned8          LostLinkCountPort1;
  Unsigned8          LostLinkCountPort2;
  Unsigned8          LostLinkCountPort3;
} TLOSTLINKCOUNTER;
```

Figure 15 – Lost link counter type description

The Lost Link Counter coding is specified in Table 41.

Table 41 – Lost link counter

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Lost link count port 0	0x0310	unsigned8	RW	R	A write to one counter will reset all link lost counters
Lost link count port 1	0x0311	unsigned8	RW	R	A write to one counter will reset all link lost counters
Lost link count port 2	0x0312	unsigned8	RW	R	A write to one counter will reset all link lost counters
Lost link count port 3	0x0313	unsigned8	RW	R	A write to one counter will reset all link lost counters

6.2.3 Additional counter

The optional previous error counter registers contain information about error frames that indicate a problem on the predecessor links. As frames with error have a specific type of checksum this could be detected and reported. All counters will be cleared if one counter is written. The counting is stopped when the maximum value of a counter (255) is reached.

Parameter

Port 0 previous error count

This parameter counts the occurrences of errors detected by predecessor.

Port 1 previous error count

This parameter counts the occurrences of errors detected by predecessor.

Port 2 previous error count

This parameter counts the occurrences of errors detected by predecessor.

Port 3 previous error count

This parameter counts the occurrences of errors detected by predecessor.

The optional wrong Type 12 frame counter counts frames with i.e. wrong datagram structure. Counter will be cleared if one of the counters is written. The counting is stopped when the maximum value of a counter (255) is reached.

Parameter

Wrong Type 12 frame counter

This parameter counts the occurrences of wrong Type 12 frames.

The optional local problem counter counts occurrence of local problems. Counter will be cleared if the counter is written. The counting is stopped when the maximum value of a counter (255) is reached.

Parameter

Local problem counter

This parameter counts the occurrences of communication problems within a slave.

The attribute types of Additional Counter are described in Figure 16.

```
typedef struct
{
  Unsigned8          PreviousErrCountPort0;
  Unsigned8          PreviousErrCountPort1;
  Unsigned8          PreviousErrCountPort2;
  Unsigned8          PreviousErrCountPort3;
  Unsigned8          MalformatErrorCount;
  Unsigned8          LocalProblemCount;
} ADDCOUNTER;
```

Figure 16 – Additional counter type description

The Additional Counter coding is specified in Table 42.

Table 42 – Additional counter

Parameter	Physical Address	Data Type	Access type	Access Type PDI	Value/Description
Previous Error Count Port 0	0x0308	unsigned8	RW	R	A write to one counter will reset all counters
Previous Error Count Port 1	0x0309	unsigned8	RW	R	A write to one counter will reset all counters
Previous Error Count Port 2	0x030A	unsigned8	RW	R	A write to one counter will reset all counters
Previous Error Count Port 3	0x030B	unsigned8	RW	R	A write to one counter will reset all counters
Malformat frame Count	0x030C	unsigned8	RW	R	A write to this counter will reset this counter
Local Problem Count	0x030D	unsigned8	RW	R	A write to this counter will reset this counter

6.3 Watchdogs

6.3.1 Watchdog divider

The system clock of the slave controller is divided by the watchdog divider.

Parameter

Watchdog divider

This parameter shall contain the number of 40 ns intervals (minus 2) that represents the basic watchdog increment. (default value is 100 µs = 2 498).

The attribute type of watchdog divider is described in Figure 17.

```
typedef struct
{
  WORD          WatchdogDivider;
} TWATCHDOGDIVIDER;
```

Figure 17 – Watchdog divider type description

The Watchdog Divider coding is specified in Table 43.

Table 43 – Watchdog divider

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Watchdog divider	0x0400	WORD	RW	R	40 ns intervals used for other watchdog timers

6.3.2 DLS-user watchdog

The DL-user is monitored with the value of the DL-user watchdog. Each access from the DL-user to the slave controller shall reset this watchdog.

Parameter

DL-user watchdog

This parameter shall contain the watchdog to monitor the DL-user (default value 1000 with watchdog divider 100 µs means 100 ms watchdog).

The attribute type of DLS-user watchdog is described in Figure 18.

```
typedef struct
{
  WORD          DLSuserWatchdog;
} TDLUSERWATCHDOG;
```

Figure 18 – DLS-user Watchdog divider type description

The DLS-user watchdog coding is specified in Table 44.

Table 44 – DLS-user watchdog

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
DLS-user watchdog	0x0410	WORD	RW	R	

6.3.3 Sync manager watchdog

Each Sync manager entity is monitored with the value of the sync manager watchdog. Each write access to the DL-user memory area configured in the sync manager shall reset this watchdog if the watchdog option is enabled by this sync manager.

Parameter

Sync manager watchdog

This parameter shall contain the watchdog to monitor the Sync manager.

The attribute type of sync manager watchdog is described in Figure 19.

```
typedef struct
{
  WORD          SyncManChannelWatchdog;
} TSYNCMANCHANNELWATCHDOG;
```

Figure 19 – Sync manager watchdog type description

The sync manager watchdog coding is specified in Table 45.

Table 45 – Sync manager channel watchdog

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Sync manager watchdog	0x0420	WORD	RW	R	

6.3.4 Sync manager watchdog status

The status of each Sync manager watchdog is included in the Sync manager watchdog status.

Parameter

Sync manager watchdog status

This parameter shall contain the watchdog status of all Sync manager watchdogs.

The attribute types of sync manager watchdog status are described in Figure 20.

```
typedef struct
{
  unsigned      SyncManChannelWdStatus:      1;
  unsigned      Reserved:                   15;
} TSYNCMANCHANNELWDSTATUS;
```

Figure 20 – Sync manager watchdog status type description

The sync manager watchdog status encoding is specified in Table 46.

Table 46 – Sync manager watchdog Status

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Sync manager channel watchdog status	0x0440	Unsigned1	R	R	There is only one WD for all Sync managers 0: WD expired 1: WD active or not enabled
reserved	0x0440	Unsigned15	R	R	

6.3.5 Watchdog counter

The expiration of Watchdog is counted in this optional parameter.

Parameter

Sync manager watchdog counter

This parameter counts the expiration of all Sync manager watchdogs.

DL-user watchdog counter

This parameter counts the expiration of DL-user watchdogs.

The attribute types of watchdog counter are described in Figure 21.

```
typedef struct
{
  Unsigned8          SyncMWDCounter;
  Unsigned8          PDIWDCounter;
} WDCOUNTER;
```

Figure 21 – Watchdog counter type description

The watchdog counter coding is specified in Table 47.

Table 47 – Watchdog counter

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Sync manager WD count	0x0442	unsigned8	RW	R	A write will reset the watchdog counters
PDI WD count	0x0443	unsigned8	RW	R	A write will reset the watchdog counters

6.4 Slave information interface

6.4.1 Slave information interface area

The Slave Information Interface Area coding is DLS-user specific.

6.4.2 Slave information interface access

The attribute types of Slave Information Interface Access are described in Figure 22.

```
typedef struct
{
  unsigned      Owner:          1;
  unsigned      Lock:           1;
  unsigned      Reserved1:      6;
  unsigned      AccPDI:         1;
  unsigned      Reserved2:      7;
} TSIIACCESS;
```

Figure 22 – Slave information interface access type description

The Slave Information Interface Access coding is specified in Table 48.

Table 48 – Slave information interface access

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Owner	0x0500	Unsigned1	RW	R	0: Type 12 DL 1: PDI
Lock	0x0500	Unsigned1	RW	R	Reset Access to SII 0: no action 1: cancel access Setting this bit will reset Register 501.0
reserved	0x0500	Unsigned6	RW	R	
Access PDI	0x0501	Unsigned1	R	RW	0: no access 1: PDI access active
reserved	0x0501	Unsigned7	R	RW	

6.4.3 Slave information interface control/status

With the slave information interface control/status register the read or write operation to the slave information interface is controlled.

Parameter

Slave information interface assign

This parameter shall contain the information about assignment of interface to DL or DL-user.

Reset slave information interface access

This parameter resets access to slave information interface.

Slave information interface access

This parameter shall contain the information about slave information interface activity.

Slave information interface read size

This parameter shall contain the information about the number of octets (4 or 8) that can be read with one command.

Slave information interface write access

This parameter shall contain the information, if a write access to the slave information interface is allowed.

Slave information interface address algorithm

This parameter shall contain the information, if the protocol to the slave information interface contains one or two address octets.

Read operation

This parameter will be written from the master to start the read operation of 32 bits/64 bits in the slave information interface. This parameter will be read from the master to check if the read operation is finished.

Write operation

This parameter will be written from the master to start the write operation of 16 bits in the slave information interface. This parameter will be read from the master to check if the write operation is finished. There is no consistence guarantee for write operation. A break down during write can produce inconsistent values and should be avoided.

Reload operation

This parameter will be written from the master to start the reload operation of the first 128 bits in the slave information interface. This parameter will be read from the master to check if the reload operation is finished.

SII error

This parameter shall contain the information the read access of the SII parameter needed at start up failed.

Error command

This parameter shall contain the information if the last access to the slave information interface was successful.

Busy

This parameter contains the information if an access operation is ongoing.

The attribute types of Slave Information Interface Control/Status are described in Figure 23.

```
typedef struct
{
  unsigned      WriteAccess:              1;
  unsigned      Reserved1:                4;
  unsigned      EEPROM_Emulation          1;
  unsigned      ReadSize:                 1;
  unsigned      AddressAlgorithm:         1;
  unsigned      ReadOperation:            1;
  unsigned      WriteOperation:           1;
  unsigned      ReloadOperation:          1;
  unsigned      CheckSErrDLu:             1;
  unsigned      DeviceInfoError:          1;
  unsigned      CommandError:             1;
  unsigned      WriteError:               1;
  unsigned      Busy:                     1;
} TSIICONTROL;
```

Figure 23 – Slave information interface control/status type description

The Slave Information Interface Control/Status coding is specified in Table 49.

Table 49 – Slave information interface control/status

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
SII write access	0x0502	Unsigned1	RW	R	0x00: only read access to SII 0x01: read and write access to SII
reserved	0x0502	Unsigned4	R	R	0x00
EEPROM emulation	0x0502	Unsigned1	R	R	0x00: Normal operation (I²C interface used) 0x01: PDI emulates EEPROM (I²C not used)
SII Read Size	0x0502	Unsigned1	R	R	0x00: 4 octet read with one transaction 0x01: 8 octet read with one transaction
SII Address Algorithm	0x0502	Unsigned1	R	R	0x00: 1 octet used as address 0x01: 2 octets used as address
Read operation	0x0503	Unsigned1	RW	RW	0x00: no read operation requested (parameter write) or read operation not busy (parameter read) 0x01: read operation requested (parameter write) or read operation busy (parameter read) To start a new read operation there shall be a positive edge on this parameter
Write operation	0x0503	Unsigned1	RW	RW	0x00: no write operation requested (parameter write) or write operation not busy (parameter read) 0x01: write operation requested (parameter write) or write operation busy (parameter read) To start a new write operation there shall be a positive edge on this parameter

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Reload operation	0x0503	Unsigned1	RW	RW	0x00: no reload operation requested (parameter write) or reload operation not busy (parameter read) 0x01: reload operation requested (parameter write) or reload operation busy (parameter read) To start a new reload operation there shall be a positive edge on this parameter
Checksum Error	0x0503	Unsigned1	R	R	0x00: no checksum error loading DL-user information at startup 0x01: checksum error while reading at startup
Device info error	0x0503	Unsigned1	R	R	0x00: no error on reading Device Information at start-up 0x01: error on reading Device Information
Command error	0x0503	Unsigned1	R	R (W)	0x00: no error on last command 0x01: error on last command PDI Write only in SII emulation mode
Write error	0x0503	Unsigned1	R	R	0x00: no error on last write operation 0x01: error on last write operation
Busy	0x0503	Unsigned1	R	R	0x00: operation is finished 0x01: operation is ongoing

6.4.4 Actual slave information interface address

The actual slave information interface address register contains the actual address in the slave information interface which is accessed by the next read or write operation (by writing the slave information interface control/status register).

Parameter

Address

This parameter shall contain the address of the 16 bit word which is accessed by the next read or write operation.

The attribute type of slave information interface address is described in Figure 24.

```
typedef struct
{
  DWORD          SIIAddress;
} TSIIADDRESS;
```

Figure 24 – Slave information interface address type description

The actual slave information interface address coding is specified in Table 50.

Table 50 – Actual slave information interface address

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Address	0x0504	DWORD	RW	RW	16-Bit word address

6.4.5 Actual slave information interface data

The actual slave information interface Data register contains the data (16 bit) to be written in the slave information interface with the next write operation or the read data (32 bit/64 bit) with the last read operation.

Parameter

Data

The master will write this parameter with the data (16 bit) to be written in the slave information interface with the next write operation. The master will receive the last read data (32 bit/64 bit) from the slave information interface when reading this parameter.

The attribute type of slave information interface data is described in Figure 25.

```
typedef struct
{
  DWORD          SIIData;
} TSIIDATA;
```

Figure 25 – Slave information interface data type description

The actual slave information interface data coding is specified in Table 51.

Table 51 – Actual slave information interface data

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Data	0x0508	DWORD	RW	RW	For the write operation only the lower 16 Bit (0x508-0x509) will be used

6.5 Media independent interface (MII)

6.5.1 MII control/status

The MII management contains a set of optional attributes. With the MII control/status register the read or write operation to the MII is controlled.

Parameter

MII write access

This parameter shall contain the information, if a write access to the MII is allowed. Read should be always enabled if MII management is supported.

Address offset

This parameter shall contain the information about the offset between port number and MII address.

Read operation

This parameter will be written from the master to start the read operation of 16 bits in the MII. This parameter will be read from the master to check if the read operation is finished.

Write operation

This parameter will be written from the master to start the write operation of 16 bits in the MII. This parameter will be read from the master to check if the write operation is finished. There is no consistence guarantee for write operation. A break down during write can produce inconsistent values and should be avoided omission critical operations.

Error command

This parameter shall contain the information if the last access to the MII was successful.

Busy

This parameter contains the information if an access operation is ongoing.

The attribute types of MII control/status are described in Figure 26.

```
typedef struct
{
  unsigned      WriteAccess:            1;
  unsigned      Reserved1:              6;
  unsigned      PHYoffset:              1;
  unsigned      ReadOperation:          1;
  unsigned      WriteOperation:         1;
  unsigned      Reserved2:              4;
  unsigned      WriteError:             1;
  unsigned      Busy:                   1;
} TMIICONTROL;
```

Figure 26 – MII control/status type description

The MII control/status coding is specified in Table 52.

Table 52 – MII control/status

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Write access	0x0510	Unsigned1	RW	R	0x00: only read access to MII 0x01: read and write access to MII
Access PDI	0x0510	Unsigned1	R	R	0x00: Only ECAT 0x01: PDI access possible
Link Detection via MII management interface	0x0510	Unsigned1	R	R	0x00: Not active 0x01: Active
PHYoffset	0x0510	Unsigned5	R	R	0x00 (Default) offset to be added to MII address Set up by local configuration

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Read operation	0x0511	Unsigned1	RW	R	0x00: no read operation requested (parameter write) or read operation not busy (parameter read) 0x01: read operation requested (parameter write) or read operation busy (parameter read) To start a new read operation there shall be a positive edge on this parameter
Write operation	0x0511	Unsigned1	RW	R	0x00: no write operation requested (parameter write) or write operation not busy (parameter read) 0x01: write operation requested (parameter write) or write operation busy (parameter read) To start a new write operation there shall be a positive edge on this parameter
reserved	0x0511	Unsigned3	R	R	0x00
Read error	0x0511	Unsigned1	R	R	0x00: no error on last read operation 0x01: error on last read operation
Write error	0x0511	Unsigned1	R	R	0x00: no error on last write operation 0x01: error on last write operation
Busy	0x0511	Unsigned1	R	R	0x00: operation is finished 0x01: operation is ongoing

6.5.2 Actual MII address

The actual MII address register contains the actual address in the MII register of the slave which is accessed by the next read or write operation (by writing the MII control/status register).

Parameter

Address PHY

This parameter shall contain the address of the PHY which is accessed by the next read or write operation.

Address PHY register

This parameter shall contain the address of the PHY register which is accessed by the next read or write operation. PHY registers can be found in Clause 22 of ISO/IEC 8802-3:2000.

The attribute types of MII address are described in Figure 27.

```
typedef struct
{
  Byte          PHYAddress;
  Byte          RegAddress;
} TMIIADDRESS;
```

Figure 27 – MII address type description

The actual MII address coding is specified in Table 53.

Table 53 – Actual MII address

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Address PHY	0x0512	Unsigned8	RW	RW	Address of the PHY (0-63)
Address register	0x0513	Unsigned8	RW	RW	Address of the PHY Registers

6.5.3 Actual MII data

The actual MII data register contains the data (16 bit) to be written in the MII with the next write operation or the read data (16 bit) with the last read operation.

Parameter

Data

The master will write this parameter with the data to be written in the MII with the next write operation. The master will receive the last read data from the MII when reading this parameter.

The attribute type of MII data is described in Figure 28.

```
typedef struct
{
  Word          MIIData;
} TMIIDATA;
```

Figure 28 – MII data type description

The actual MII data coding is specified in Table 54.

Table 54 – Actual MII data

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Data	0x0514	Unsigned16	RW	RW	

6.5.4 MII access

The optional MII access registers mangages the MII access from ECAT and from PDI.

Parameter

Access MII

The control of the MII management

Access State

The register reflects the current access state

Access Reset

Reset Access State register

The attribute type of MII access is described in Figure 29.

```
typedef struct
{
  unsigned      MIIAccess:              1;
  unsigned      Reserved1:              7;
  unsigned      MIIAccessState:         1;
  unsigned      MIIAccessReset:         1;
  unsigned      Reserved2:              6;
} TMIIAccess;
```

Figure 29 – MII access type description

The MII access coding is specified in Table 55.

Table 55 – MII access

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
MII Access	0x0516	Unsigned1	RW	R	0: PDI control possible 1: No PDI control
reserved	0x0516	Unsigned7	R	R	
Access State	0x0517	Unsigned1	R	RW	0: ECAT access active 1: PDI access active
Access Reset	0x0517	Unsigned1	RW	R	0: no action 1: reset 0x0517.0
reserved	0x0517	Unsigned6	R	R	

6.6 Fieldbus memory management unit (FMMU)

6.6.1 General

The fieldbus memory management unit (FMMU) converts logical addresses into physical addresses by the means of internal address. Thus, FMMUs allow one to use logical addressing for data segments that span several slave devices: one DLPDU addresses data within several arbitrarily distributed devices. The FMMUs optionally support bit wise mapping. A DLE may contain several FMMU entities. Each FMMU entity maps one cohesive logical address space to one cohesive physical address space.

The FMMU consists of up to 16 entities. Each entity describes one memory translation between the logical memory of the Type 12 communication network and the physical memory of the slave.

Figure 30 shows an example mapping of logical address 0x14711.3 to 0x14712.0 to memory-octet 0xF01.1 to 0xF01.6.

NOTE The representation of bit values from left as the least significant bit to right as most significant bit does not imply an ordering scheme on transmission line.

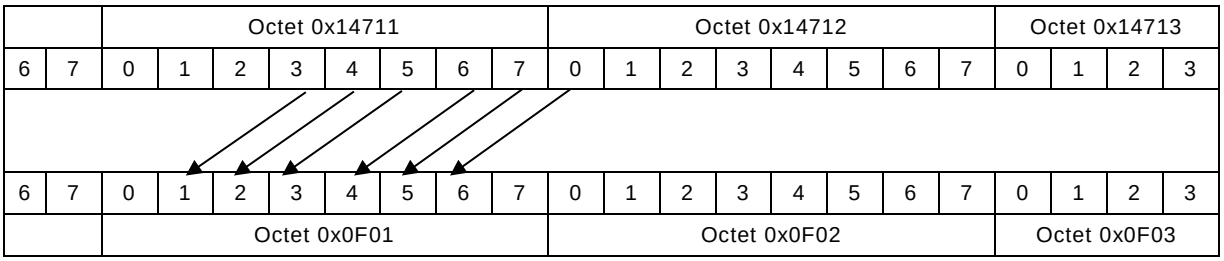


Figure 30 – FMMU mapping example

6.6.2 FMMU attributes

Parameter

Logical start address

This parameter shall contain the start address in octets in the logical memory area of the memory translation.

Logical start bit

This parameter shall contain the bit offset of the logical start address.

Logical end bit

This parameter shall contain the bit offset of the logical end address.

Physical start address

This parameter shall contain the start address in octets in the physical memory area of the memory translation.

Physical start bit

This parameter shall contain the bit offset of the physical start address.

Length

This parameter shall contain the size in octets of the memory translation from the first byte to the last byte in the logical address space (Length is 2 for the mapping).

Read enable

This parameter shall contain the information if a read operation (physical memory is source, logical memory is destination) is enabled.

Write enable

This parameter shall contain the information if a write operation (logical memory is source, physical memory is destination) is enabled.

Enable

This parameter shall contain the information if the memory translation is active or not.

The attribute types of FMMU entity are described in Figure 31.

```
typedef struct
{
  DWORD         LogicalStartAddress;
  WORD          Length;
  unsigned      LogicalStartBit:        3;
  unsigned      Reserved1:              5;
  unsigned      LogicalEndBit:          3;
  unsigned      Reserved2:              5;
  WORD          PhysicalStartAddress;
  unsigned      PhysicalStartBit:       3;
  unsigned      Reserved3:              5;
  unsigned      ReadEnable:             1;
  unsigned      WriteEnable:            1;
  unsigned      Reserved4:              6;
  unsigned      Enable:                 1;
  unsigned      Reserved5:              7;
  unsigned      Reserved6:              8;
  WORD          Reserved7;
} TFMMU;
```

Figure 31 – FMMU entity type description

A FMMU entity is specified in Table 56. Table 57 shows the FMMU structure.

Table 56 – Fieldbus memory management unit (FMMU) entity

Parameter	relative address (offset)	Data type	Access type	Access type PDI	Value/description
Logical start address	0x0000	DWORD	RW	R	
Length	0x0004	WORD	RW	R	
Logical start bit	0x0006	Unsigned3	RW	R	
reserved	0x0006	Unsigned5	RW	R	0x00
Logical end bit	0x0007	Unsigned3	RW	R	
reserved	0x0007	Unsigned5	RW	R	0x00
Physical start address	0x0008	WORD	RW	R	
Physical start bit	0x000A	Unsigned3	RW	R	
reserved	0x000A	Unsigned5	RW	R	0x00
Read enable	0x000B	Unsigned1	RW	R	0x00: entity will be ignored for read service 0x01: entity will be used for read service
Write enable	0x000B	Unsigned1	RW	R	0x00: entity will be ignored for write service 0x01: entity will be used for write service
reserved	0x000B	Unsigned6	RW	R	0x00
Enable	0x000C	Unsigned1	RW	R	0x00: entity not active 0x01: entity active
reserved	0x000C	Unsigned7	RW	R	0x00
reserved	0x000D	Unsigned24	R	R	0x0000

Table 57 – Fieldbus memory management unit (FMMU)

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
FMMU entity 0	0x0600	TFMMU	RW	R	
FMMU entity 1	0x0610	TFMMU	RW	R	
FMMU entity 2	0x0620	TFMMU	RW	R	
FMMU entity 3	0x0630	TFMMU	RW	R	
FMMU entity 4	0x0640	TFMMU	RW	R	
FMMU entity 5	0x0650	TFMMU	RW	R	
FMMU entity 6	0x0660	TFMMU	RW	R	
FMMU entity 7	0x0670	TFMMU	RW	R	
FMMU entity 8	0x0680	TFMMU	RW	R	
FMMU entity 9	0x0690	TFMMU	RW	R	
FMMU entity 10	0x06A0	TFMMU	RW	R	
FMMU entity 11	0x06B0	TFMMU	RW	R	
FMMU entity 12	0x06C0	TFMMU	RW	R	
FMMU entity 13	0x06D0	TFMMU	RW	R	
FMMU entity 14	0x06E0	TFMMU	RW	R	
FMMU entity 15	0x06F0	TFMMU	RW	R	

6.7 Sync manager

6.7.1 Sync manager overview

The sync manager controls the access to the DL-user memory. Each channel defines a consistent area of the DL-user memory.

There are two ways of data exchange between master and PDI:

- Handshake mode (mailbox): one entity fills data in and cannot access the area until the other entity reads out the data.
- Buffered mode: the interaction between both producer of data and consumer of data is uncorrelated – each entity expects access at any time, always providing the consumer with the newest data.

The Handshake mode is implemented with one buffer: an interrupt or a status flag indicates whether a buffer is empty or full.

The interchange of a buffer is valid only if the FCS of the frames that carries the read or writes command is valid. The principle of interaction is shown in Figure 32.

The actions of exchange buffers are coupled on the first octet and on the last octet:

- writing data in the first octet enables writing to the buffer if buffer is empty
- the buffer state will be set to full by writing the last octet of the buffer
- reading data out of the first octet prepares buffer for reading
- the buffer state will be set to empty by reading out the last octet of the buffer

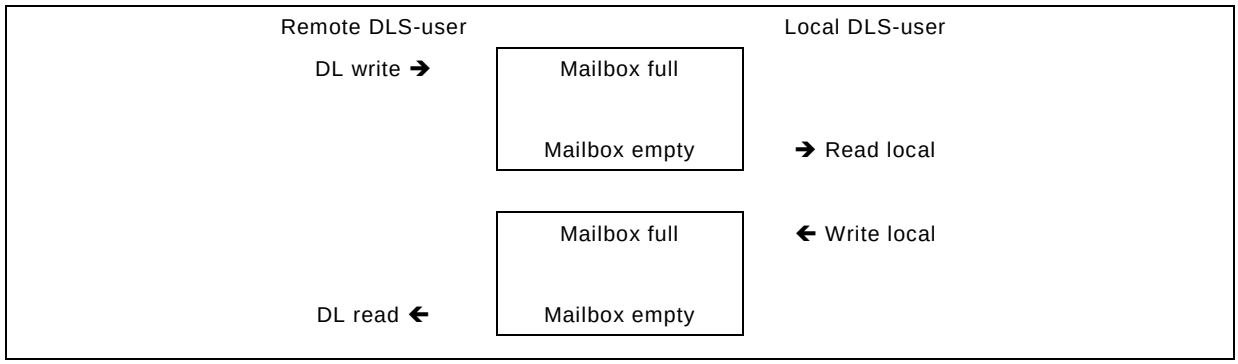


Figure 32 – SyncM mailbox interaction

If a mailbox is full, it cannot be written again until it is read out (i.e. last octet of mailbox will be read out). It does not matter how long it takes to read out the data (there may be timing constraints at the layers above).

For cyclic data there is a different concept implemented to ensure consistency and availability of data. This is accomplished by a set of buffers, which allows writing and reading data simultaneously without interference. Two buffers are allocated to the sender and to the receiver; a spare buffer helps as intermediate store.

This means that, in this mode, the buffers need to be triplicated. Figure 33 demonstrates a configuration with start address of 0x1000 and length of 0x100. The other buffers are virtually not available. Access is done always with addresses in the range of buffer 1. Reading the last octet or writing the last octet results in an automatic buffer exchange (from DL side only if the frame with the buffer data is received correctly).

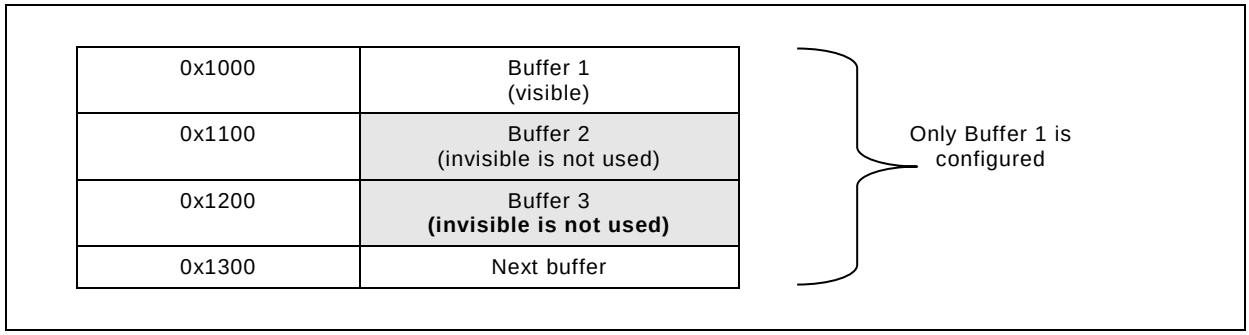


Figure 33 – SyncM buffer allocation

Figure 34 shows the principle of sync manager buffer interaction.

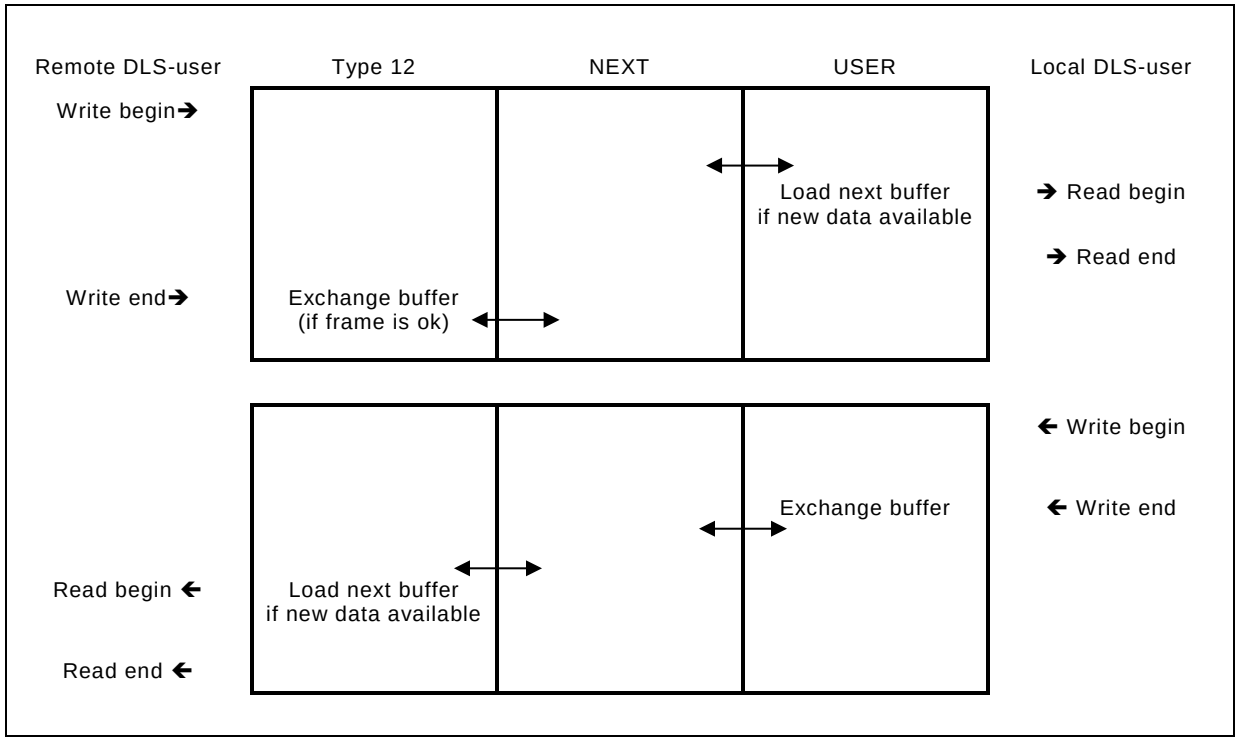


Figure 34 – SyncM buffer interaction

This scheme allows access to a buffer independent of the read or write frequencies. Therefore, the slave can be implemented independent of the speed of the master.

Figure 35 shows an example interaction with a read mailbox error.

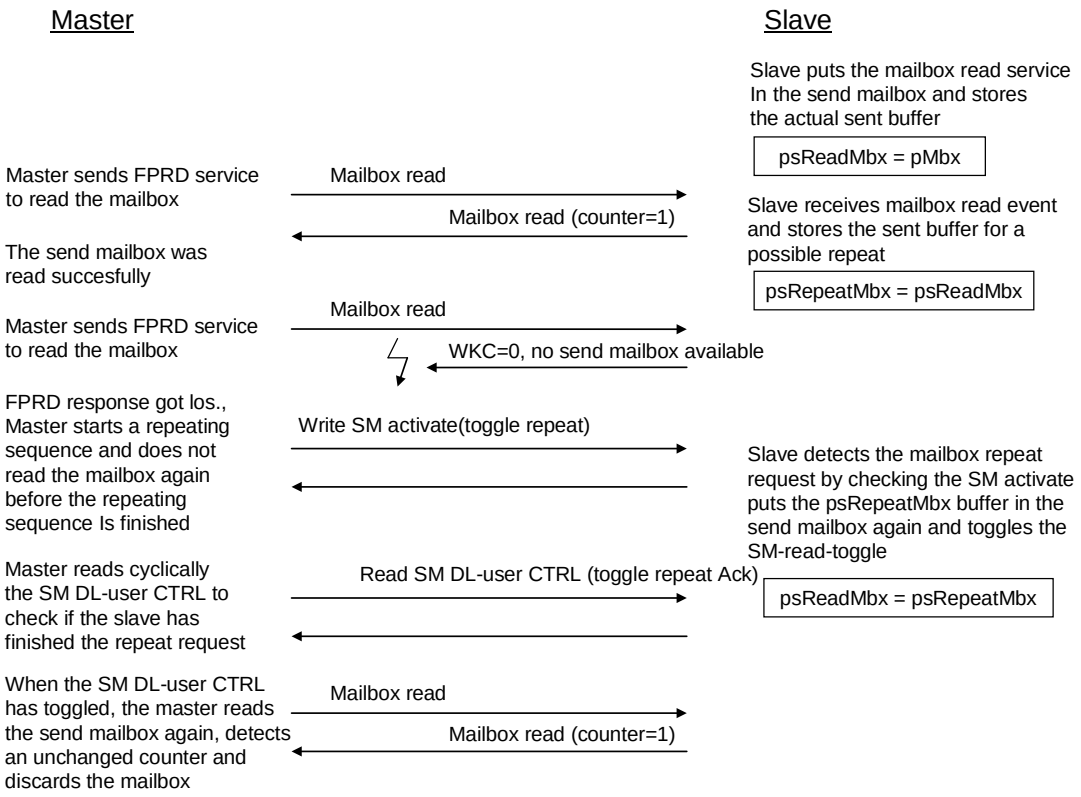


Figure 35 – Handling of write/read toggle with read mailbox

The toggle bits are used to resynchronize mailbox communication if a Type 12 DLPDU is lost, i.e. a previously lost read-mailbox entry will be loaded again.

6.7.2 Sync Manager Attributes

Parameter

Physical start address

This parameter shall contain the start address in octets in the physical memory of the consistent DL-user memory area.

Length

This parameter shall contain the size in octets of the consistent DL-user memory area.

Operation mode

This parameter shall contain the information if the consistent DL-user memory area is of mailbox access type or buffered access type.

Direction

This parameter shall contain the information if the consistent DL-user memory area is read or written by the master.

Ecat Event enable

This parameter shall contain the information if an event is generated if there is new data available in the consistent DL-user memory area which was written by the master (direction write) or if the new data from the DL-user was read by the master (direction read).

DLS-user Event enable

This parameter shall contain the information if an event is generated if there is new data available in the consistent DL-user memory area which was written by DLS-user or if the new data from the Master was read by the DLS-user.

Watchdog trigger enable

This optional parameter shall contain the information if the monitoring of an access to the consistent DL-user memory area is enabled.

Write event

This parameter shall contain the information if the consistent DL-user memory (direction write) has been written by the master and the event enable parameter is set.

Read event

This parameter shall contain the information if the consistent DL-user memory (direction read) has been read by the master and the event enable parameter is set.

Mailbox access type state

This parameter shall contain the state (buffer read, buffer written) of the consistent DL-user memory if it is of mailbox access type.

Buffered access type state

This optional parameter shall contain the state (buffer number, locked) of the consistent DL-user memory if it is of buffered access type.

Read buffer state

This optional parameter indicates the current read buffer state

Write buffer state

This optional parameter indicates the current write buffer state

Channel enable

This parameter shall contain the information if the sync manager channel is active.

Repeat

A change in this parameter indicates a repeat request. This is primarily used to repeat the last mailbox interactions.

DC Event 0 with Type 12 write

This optional parameter shall contain the information if the DC 0 Event shall be invoked in case of a Type 12 write.

DC Event 0 with local write

This optional parameter shall contain the information if the DC 0 Event shall be invoked in case of a local write.

Channel enable PDI

This parameter shall contain the information if the sync manager channel is active.

Repeat Ack

A change in this parameter indicates a repeat request acknowledge. After setting the value of Repeat in the parameter repeat acknowledge.

The attribute types of a sync manager channel are described in Figure 36.

```
typedef struct
{
  WORD          PhysicalStartAddress;
  WORD          Length;
  unsigned      OperationMode:          2;
  unsigned      Direction:              2;
  unsigned      EcatEventEnable:        1;
  unsigned      DLSuserEventEnable:     1;
  unsigned      WatchdogEnable:         1;
  unsigned      Reserved2:              1;
  unsigned      WriteEvent:             1;
  unsigned      ReadEvent:              1;
  unsigned      Reserved3:              1;
  unsigned      mailboxState:           1;
  unsigned      bufferState:            2;
  unsigned      ReadBufferState:        1;
  unsigned      WriteBufferState:       1;
  unsigned      ChannelEnable:          1;
  unsigned      Repeat:                 1;
  unsigned      Reserved5:              4;
  unsigned      DCEvent0wBusw:          1;
  unsigned      DCEvent0wlocw:          1;
  unsigned      ChannelEnablePDI:       1;
  unsigned      RepeatAck:              1;
  unsigned      Reserved6:              6;
} TSYNCMAN;
```

Figure 36 – Sync manager channel type description

A sync manager channel is specified in Table 58.

Table 59 shows the sync manager structure.

Table 58 – Sync manager channel

Parameter	relative address (offset)	Data type	Access type	Access type PDI	Value/description
Physical start address	0x0000	WORD	RW	R	
Length	0x0002	WORD	RW	R	
Buffer type	0x0004	Unsigned2	RW	R	0x00: buffered 0x02: mailbox
Direction	0x0004	Unsigned2	RW	R	0x00: area shall be read from the master 0x01: area shall be written by the master
ECAT event enable	0x0004	Unsigned1	RW	R	0x00: event is not active 0x01: event is active
DLS-user event enable	0x0004	Unsigned1	RW	R	0x00: DLS-user event is not active 0x01: DLS-user event is active
Watchdog enable	0x0004	Unsigned1	RW	R	0x00: watchdog disabled 0x01: watchdog enabled
reserved	0x0004	Unsigned1	RW	R	0x00
Write event	0x0005	Unsigned1	R	R	0x00: no write event 0x01: write event
Read event	0x0005	Unsigned1	R	R	0x00: no read event 0x01: read event

Parameter	relative address (offset)	Data type	Access type	Access type PDI	Value/description
reserved	0x0005	unsigned1	R	R	0x00
Mailbox state	0x0005	Unsigned1	R	R	0x00: mailbox empty 0x01: mailbox full
Buffered state	0x0005	Unsigned2	R	R	0x00: first buffer 0x01: second buffer 0x02: third buffer 0x03: buffer locked
Read buffer state	0x0005	Unsigned1	R	R	0x00: read buffer is not open 0x01: read buffer is open
Write buffer state	0x0005	Unsigned1	R	R	0x00: write buffer is not open 0x01: write buffer is open
Channel enable	0x0006	Unsigned1	RW	R	0x00: channel disabled 0x01: channel enabled
Repeat	0x0006	Unsigned1	RW	R	
reserved	0x0006	Unsigned4	RW	R	0x00
DC Event 0 with Bus access	0x0006	Unsigned1	RW	R	0x00: no Event 0x01: DC Event if master completes buffer access
DC Event 0 with local access	0x0006	Unsigned1	RW	R	0x00: no Event 0x01: DC Event if DL-user completes buffer access
Channel enable PDI	0x0007	Unsigned1	R	RW	0x00: channel enabled 0x01: channel disabled
RepeatAck	0x0007	Unsigned1	R	RW	shall follow repeat after data recovery
reserved	0x0007	Unsigned6	R	RW	0x00

Table 59 – Sync manager Structure

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Sync manager channel 0	0x0800	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 1	0x0808	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 2	0x0810	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 3	0x0818	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 4	0x0820	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 5	0x0828	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 6	0x0830	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 7	0x0838	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 8	0x0840	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 9	0x0848	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 10	0x0850	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 11	0x0858	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 12	0x0860	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 13	0x0868	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 14	0x0870	TSYNCMAN	RW	R	Last Byte PDI writeable
Sync manager channel 15	0x0878	TSYNCMAN	RW	R	Last Byte PDI writeable

The Sync Manager channels shall be used in the following way:

- Sync Manager channel 0: mailbox write
- Sync Manager channel 1: mailbox read
- Sync Manager channel 2: process data write (may be used for process data read if no process data write supported)
- Sync Manager channel 3: process data read

If mailbox is not supported, it shall be used in the following way:

- Sync Manager channel 0: process data write (may be used for process data read if no process data write supported)
- Sync Manager channel 1: process data read

6.8 Distributed clock

6.8.1 General

DC is used for very precise timing requirements and for using timing signals that can be generated independent of the communication cycle. Systems with not so high requirements on synchronization may be synchronized by sharing a service (preferable LRW or LRD or LWR) or using the same Ethernet frame for access to buffers.

6.8.2 Delay measurement

Delay measurement needs time stamping information which is related to a single frame. The slave just provides means for time stamping, the calculation of the delay is the task of the master.

Parameter

Receive time port 0

This parameter shall contain the receiving time of a special datagram's beginning on port 0. The special datagram shall be a write access to this parameter. The receiving time of this frame will be written in this parameter at the end of this datagram if the receiving was correctly. Additionally the latch for the receive time port 1, 2 and 3 registers will be enabled for the same datagram.

Receive time port 1

This parameter shall contain the receiving time of a special datagram's beginning on port 1. The special datagram shall be a write access to the receive time port 0 register. The receiving time of this frame will be written in this parameter at the end of this datagram if the receiving was correctly.

Receive time port 2

This parameter shall contain the receiving time of a special datagram's beginning on port 2. The special datagram shall be a write access to the receive time port 0 register. The receiving time of this frame will be written in this parameter at the end of this datagram if the receiving was correctly.

Receive time port 3

This parameter shall contain the receiving time of a special datagram's beginning on port 3. The special datagram shall be a write access to the receive time port 0 register. The receiving time of this frame will be written in this parameter at the end of this datagram if the receiving was correctly.

6.8.3 Local time parameter

The local time parameter contains the local system time and parameter for the control loop which are dedicate to implement a control loop for coordinating the local system time with a global time.

Parameter

Local system time

This parameter shall contain the local system time latched when a datagram is received. A write access to this parameter shall start a comparison of the latched local system time with the written reference system time. The result of this comparison shall be an input of the PLL for the local system time.

System time offset

This parameter shall contain the offset between the local system time and the global time.

System time transmission delay

This parameter shall contain the transmission delay from the slave controller with the reference system time to the local slave controller.

System time difference

This parameter shall contain the result of the last compare between local system time and time of last write minus system time offset and minus system time transmission delay.

Control loop parameters

This implementation specific parameters shall contain the setting parameters for the local system time control loop.

6.8.4 DL-user time parameter

The DL-user time parameter contains the local time parameter DC user P1 to P12 for DL-user.

NOTE The meaning of the parameters is not defined in this scope. The access rights are specified in IEC 61158-5-12.

6.8.5 DC attributes

The attribute types of distributed clock delay measurement is included in local time parameter which is described in Figure 37.

```
typedef struct
{
  DWORD         ReceiveTimePort0;
  DWORD         ReceiveTimePort1;
  DWORD         ReceiveTimePort2;
  DWORD         ReceiveTimePort3;
  UINT64        LocalSystemTime;
  BYTE          Reserved2[8];
  UINT64        SystemTimeOffset;
  DWORD         SystemTimeTransmissionDelay;
  DWORD         SystemTimeDifference;
  WORD          ControlLoopParameter1;
  WORD          ControlLoopParameter2;
  WORD          ControlLoopParameter3;
  BYTE          Reserved3[74];
} TDCTRANSMISSION;
```

Figure 37 – Distributed clock local time parameter type description

The distributed clock local time parameter is specified in Table 60.

Table 60 – Distributed clock local time parameter

Parameter	Physical address (offset)	Data type	Access type	Access type PDI	Value/description
Receive time port 0	0x0900	DWORD	R	R	A write access latches the local time (in ns) at receive begin (start first element of preamble) on Port 0 of this PDU in this parameter (if the PDU was received correctly) and enables the latch of Port 1 - 3
Receive time port 1	0x0904	DWORD	R	R	Local time (in ns) at receive begin on Port 1 when a PDU containing a write access to Receive time port 0 register was received correctly
Receive time port 2	0x0908	DWORD	R	R	Local time (in ns) at receive begin on Port 1 when a PDU containing a write access to Receive time port 0 register was received correctly
Receive time port 3	0x090C	DWORD	R	R	Local time (in ns) at receive begin on Port 1 when a PDU containing a write access to Receive time port 0 register was received

Parameter	Physical address (offset)	Data type	Access type	Access type PDI	Value/description
					correctly
System time	0x0910	UINT64	RW	R	A write access compares the latched local system time (in ns) at receive begin at the processing unit of this PDU with the written value (lower 32 bit; if the PDU was received correctly), the result will be the input of DC PLL
Receive time processing unit	0x0918	UINT64	RW	R	Local time (in ns) at receive begin at the processing unit of a PDU containing a write access to Receive time port 0 (if the PDU was received correctly)
System time offset	0x0920	UINT64	RW	R	Offset between the local time (in ns) and the local system time (in ns)
System time transmission delay	0x0928	DWORD	RW	R	Offset between the reference system time (in ns) and the local system time (in ns)
System time difference	0x092C	DWORD	RW	R	Bit 30..0: Mean difference between local copy of System Time and received System Time values Bit 31: 0: Local copy of System Time greater than or equal received System Time 1: Local copy of System Time smaller than received System Time
Control Loop Parameter 1	0x0930	WORD	R(W)	R(W)	Implementation Specific
Control Loop Parameter 2	0x0932	WORD	R	R	Implementation Specific
Control Loop Parameter 3	0x0934	WORD	R(W)	R(W)	Implementation Specific

The Distributed Clock DLS-user parameter encoding is described in Table 61.

Table 61 – Distributed clock DLS-user parameter

Parameter	Physical address (offset)	Data type	Access type	Access type PDI	Value/description
reserved	0x0980	BYTE	RW	R	0
DC user P1	0x0981	BYTE	RW	R	Implementation Specific
DC user P2	0x0982	Unsigned16	R	R	Implementation Specific
DC user P13	0x0983	BYTE	R	R	Implementation Specific
DC user P14	0x0984	BYTE	R	R	Implementation Specific
reserved	0x0985	BYTE[8]	RW	R	
DC user P3	0x098E	Unsigned16	R	R	Implementation Specific
DC user P4	0x0990	DWORD	RW	R	Implementation Specific
reserved	0x0994	BYTE[12]	R	R	
DC user P5	0x09A0	DWORD	RW	R	Implementation Specific
DC user P6	0x09A4	DWORD	RW	R	Implementation Specific
DC user P7	0x09A8	Unsigned16	RW	R	Implementation Specific
reserved	0x09AA	BYTE[4]	R	R	
DC user P8	0x09AE	Unsigned1	R	R	Implementation Specific
DC user P9	0x09B0	DWORD	R	R	Implementation Specific
reserved	0x09B4	BYTE[4]	R	R	
DC user P10	0x09B8	DWORD	R	R	Implementation Specific
reserved	0x09BC	BYTE[4]	R	R	
DC user P11	0x09C0	DWORD	R	R	Implementation Specific
reserved	0x09C4	BYTE[4]	R	R	
DC user P12	0x09C8	DWORD	R	R	Implementation Specific
reserved	0x09CC	BYTE[4]	R	R	

7 DL-user memory

7.1 Overview

After reset, when DLS-user is operational, memory can be used in principle from communication and from local DL-user without any restrictions and there is a communication possible via this area. But there is no consistent handling of data possible via that mechanism.

With SYNC manager, it is possible to use the memory area in a coordinated fashion. Because the SYNC manager is established by the master, a slave does not use this area that is dedicated to communication.

The following two coordinated ways of communication are supported.

- A buffered mode which allows consistent reading and writing in both directions – three memory areas are needed to support that. The local update rate and the communication cycle can be set up independently.
- A mailbox mode with a single buffer that enables interlocked communication. One entity (communication or DL-user) fills in the data and the memory area is locked until the other entity will read out the data.

7.2 Mailbox access type

7.2.1 Mailbox transfer

Mailbox transfer services are described from the point of master regarding the direction (write means write of data from the master and read means read out of data by the master) and from the slave regarding the service description. The interaction includes a handshake procedure, i.e. the master has to wait for an action of the slave after issuing a service request and vice versa.

The data-link layer specifies resilient services for reading and writing of one shot data.

With the write service a master (client) requests a change in a memory area of the slave. A write service will be acknowledged if the addressed slave is available and the write mailbox is empty. A sequence count is present to detect duplicates. Consecutive writes with the same sequence count value will be indicated only once.

The read update data will be stored until there is a read data indication for the next read update.

7.2.2 Write access from master

Figure 38 shows the primitives between master, DLL and DL-user in case of a successful write sequence.

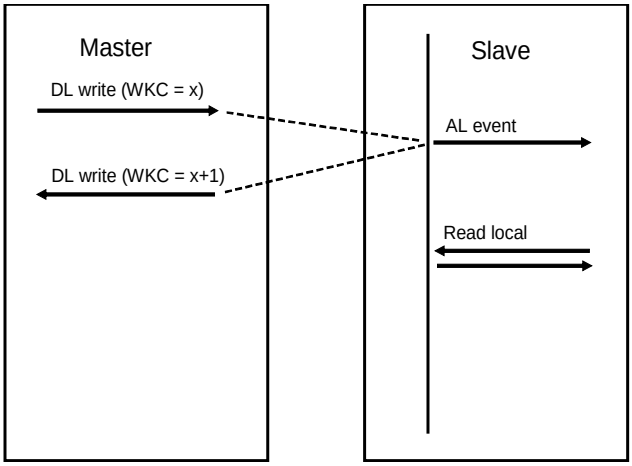


Figure 38 – Successful write sequence to mailbox

The master sends a write service with the working counter (WKC = x), the DLL (slave controller) of the slave write the received data in the DL-user memory area, increments the working counter (WKC = x + 1) and generates an event. The corresponding sync manager channel locks the DL-user memory area until it will be read by the DL-user. The master receives a successful write response because the WKC was incremented. The DL-user reads the DL-user memory area and the corresponding sync manager channel unlocks the DL-user memory area so that it can be written again by the master.

Figure 39 shows the primitives between master, DLL and DL-user in case of a bad write sequence.

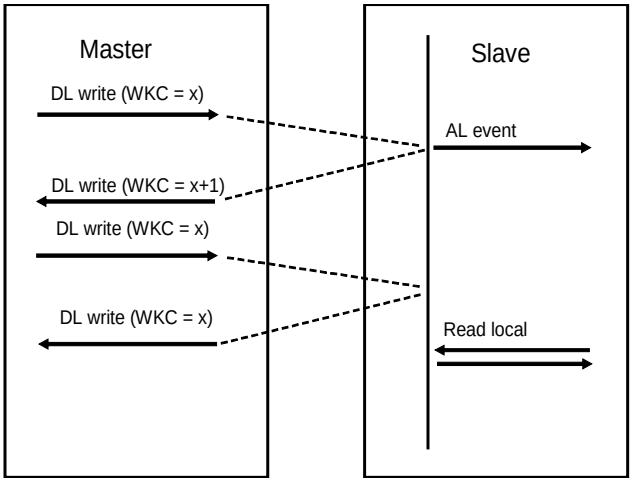


Figure 39 – Bad write sequence to mailbox

The master sends a write service with the working counter (WKC = x), the DLL (slave controller) of the slave write the received data in the DL-user memory area, increments the working counter (WKC = x + 1) and generates an event. The corresponding sync manager channel locks the DL-user memory area until it will be read from the DL-user. The master receives a successful write response because the WKC was incremented. Before the DL-user reads the DL-user memory area the master writes the same area again with the working counter (WKC = x). Because the DL-user memory area is still locked, the DLL of the slave will ignore the received data and will not increment the working counter. The master receives a bad write response because the WKC was not incremented. Later the DL-user reads the DL-user memory area and the corresponding sync manager channel unlocks the DL-user memory area so that it can be written again by the master.

7.2.3 Read access from master

Figure 40 shows the primitives between master, DLL and DL-user in case of a successful read sequence.

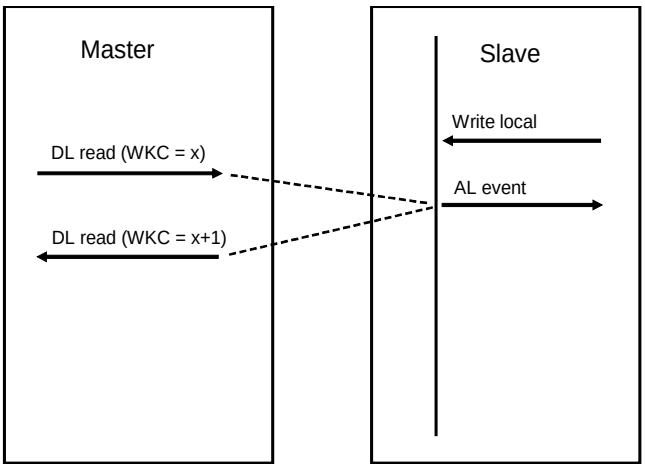


Figure 40 – Successful read sequence to mailbox

The DL-user updates the DL-user memory area. The corresponding sync manager channel locks the DL-user memory area until it will be read from the master. The master sends a read service with the working counter (WKC = x), the DLL (slave controller) of the slave sends the data of the DL-user memory area, increments the working counter (WKC = x + 1) and generates an event to the DL-user. The master receives a successful read response because the WKC was incremented. The corresponding sync manager channel unlocks the DL-user memory area that it can be written by the DL-user again.

Figure 41 shows the primitives between master, DLL and DL-user in case of a bad read sequence.

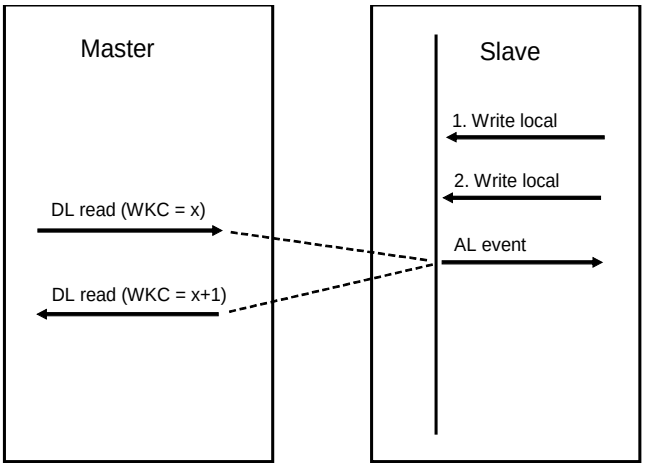


Figure 41 – Bad read sequence to mailbox

The DL-user updates the DL-user memory area (1. write Local). The corresponding sync manager channel locks the DL-user memory area until it will be read from the master. The DL-user updates the DL-user memory area again (2. write Local), but this update will be ignored by the corresponding sync manager channel because the old data was not read by the master. When the master sends a read request with the working counter (WKC = x), the DLL (slave controller) of the slave sends the data of the DL-user memory area, increments the working counter (WKC = x + 1) and generates an event to the DL-user. The master receives a successful read response because the WKC was incremented. The corresponding sync manager channel now unlocks the DL-user memory area so that it can be written again by the DL-user.

7.3 Buffered access type

7.3.1 Write access from master

Figure 42 shows the primitives between master, DLL and DL-user in case of a write sequence. The example shows a fast master with a slave reacting at a slower rate.

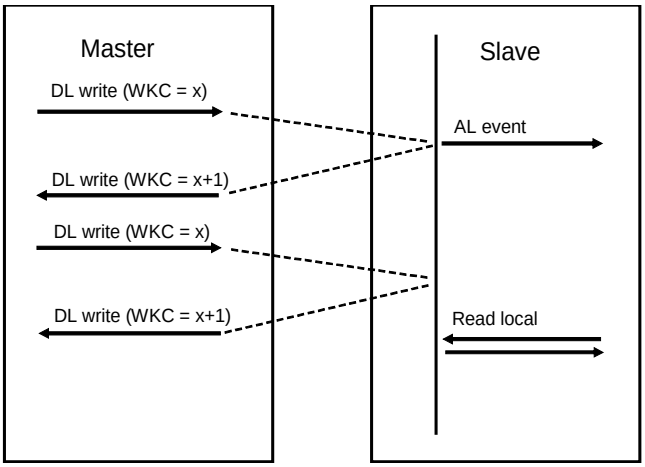


Figure 42 – Successful write sequence to buffer

The master sends a write request with the working counter (WKC = x), the DLL (slave controller) of the slave writes the received data in the DL-user memory area, increments the working counter (WKC = x + 1) and generates an event to the DL-user. The master receives a

successful write response because the WKC was incremented. Before the DL-user reads the DL-user memory area the master writes the same area again with the working counter (WKC = x). Because the buffered access type DL-user memory area is never locked, the DLL of the slave overwrites the received data in the DL-user memory area, increments the working counter (WKC = x + 1) and generates again an event to the DL-user. The master receives a successful write response because the WKC was incremented. Later the DL-user reads the DL-user memory area.

7.3.2 Read access from master

Figure 43 shows the primitives between master, DLL and DL-user in case of a successful read sequence. The slave updates data several times before the master reads out the data.

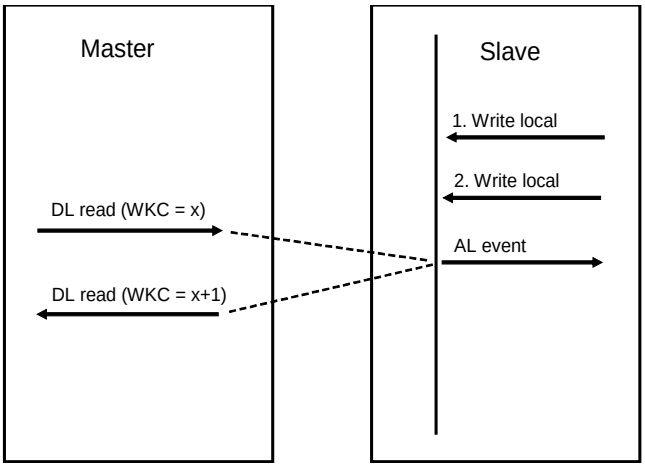


Figure 43 – Successful read sequence to buffer

The DL-user updates the DL-user memory area (1. write Local). The DL-user updates the DL-user memory area again with new values (2. write Local), because buffered type DL-user memory areas will never be locked, the corresponding sync manager channel overwrites the old data. Then the master sends a read service with the working counter (WKC = x), the DLL (slave controller) of the slave sends the data of the DL-user memory area, increments the working counter (WKC = x + 1) and generates an event to the DL-user. The master receives a successful read response because the WKC was incremented.

8 Type 12: FDL protocol state machines

8.1 Overview of slave DL state machines

Figure 44 illustrates the general structure of the DL of a slave by showing its state machines and their interaction.

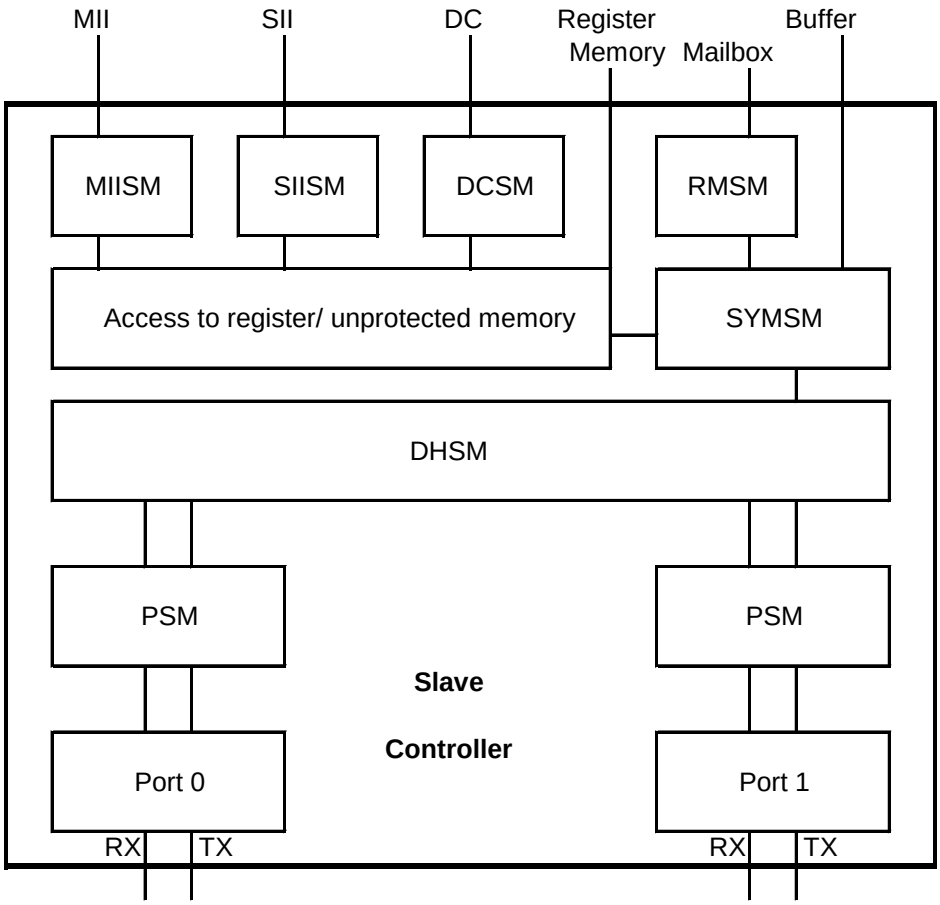


Figure 44 – Structuring of the protocol machines of an slave

8.2 State machine description

8.2.1 Port state machine (PSM)

The PSM co-ordinate the underlying port state machines used for processing of MAC frames and for passing it octet by octet to the PDU handler. There exists one state machine for each DL of the two or more DL interfaces of a slave named as ports. There is no explicit state machine for ports as it follows the rules defined for ports in ISO/IEC 8802-3 with the exceptions:

a) the message is passed at an octet by octet base instead of passing the whole frame at the DL interface.

b) if a port has no link a Tx.req primitive will result in an Rx.ind primitive (provided the port is in automatic mode or the loop is closed by a command)

Additionally, the statistic counters as defined in IEC 61158-3-12 will be handled by PSM.

8.2.2 PDU handler state machine (DHSM)

The DHSM will process the Ethernet frames by splitting it up to individual Type 12 PDUs at the primary port and the Receive Time 0 write request at the secondary port and map it to the individual registers or to the SYSM(sync manager state machine) or to the DCSM (DC state machine). The FMMU as a mapping of the global address space to the physical addressing, the activation of the SIISM and MIISM by register access are also located to the DHSM. A more detailed specification of DHSM can be found in Clause A.1.

8.2.3 Synch manager state machine (SYSM)

Synch manager state machine will handle the memory areas covered by SynchM as mailboxes and buffers. The mailbox services are forwarded to a state machine that handles retries (resilient mailbox state machine – RMSM). There exists a SYSM for each sync manager. The access to the memory is passed from SYSM to SYSM as long as no SYSM is activated for this address. If no SYSM is activated for a specific memory address a request to a memory area or register is done. Some specific access rules apply to registers – they are described at the register attributes in IEC 61158-3-12. A more detailed specification of SYSM can be found in Clause A.2.

8.2.4 Resilient mailbox state machine (RMSM)

The mandatory RMSM is responsible for mailbox retries in the read mailbox and checking of sequence numbers in the write mailbox. A retry of mailbox write is a write to the mailbox with the same sequence number.

The retry mechanism of read mailbox uses the Repeat and RepeatAck parameter of the Sync Manager channel. A toggle in the Repeat parameter triggers the slave to retry the last read. Clause A.3 describes the read mailbox behaviour.

8.2.5 SII state machine (SIISM)

8.2.5.1 Slave information interface access flow charts

SIISM is responsible for access to SII. There are read, write and reload operations specified for this interface. The master can activate this operation by following the specific procedural sequence.

8.2.5.2 Read operation

Figure 45 shows the flow of a read operation.

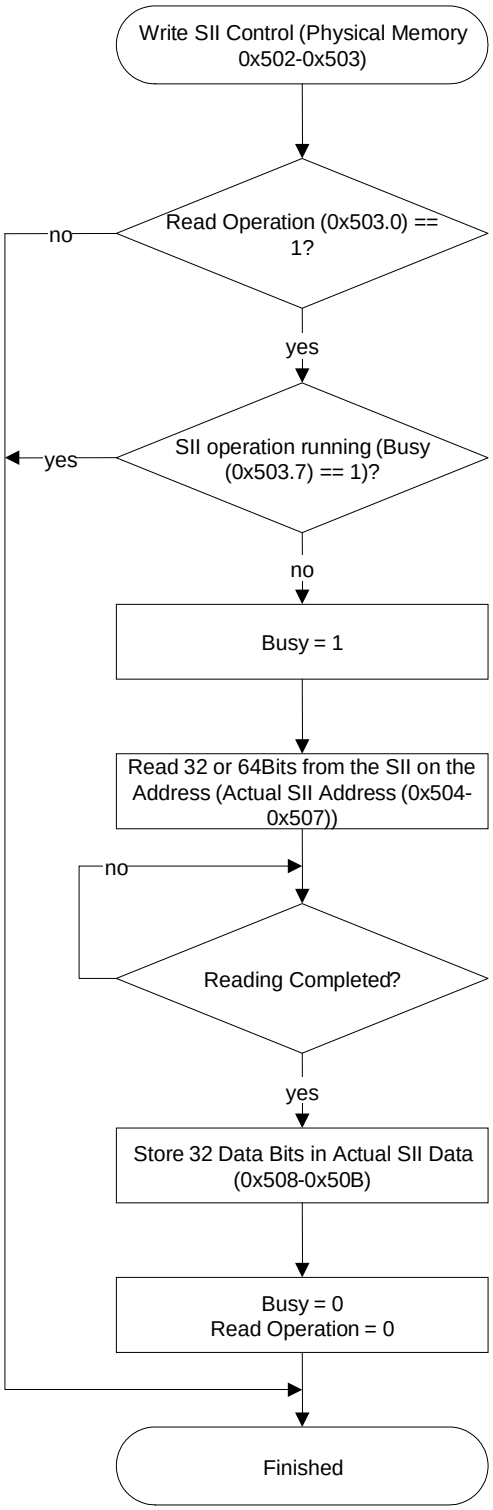


Figure 45 – Slave information interface read operation

8.2.5.3 Write operation

Figure 46 shows the flow of a write operation.

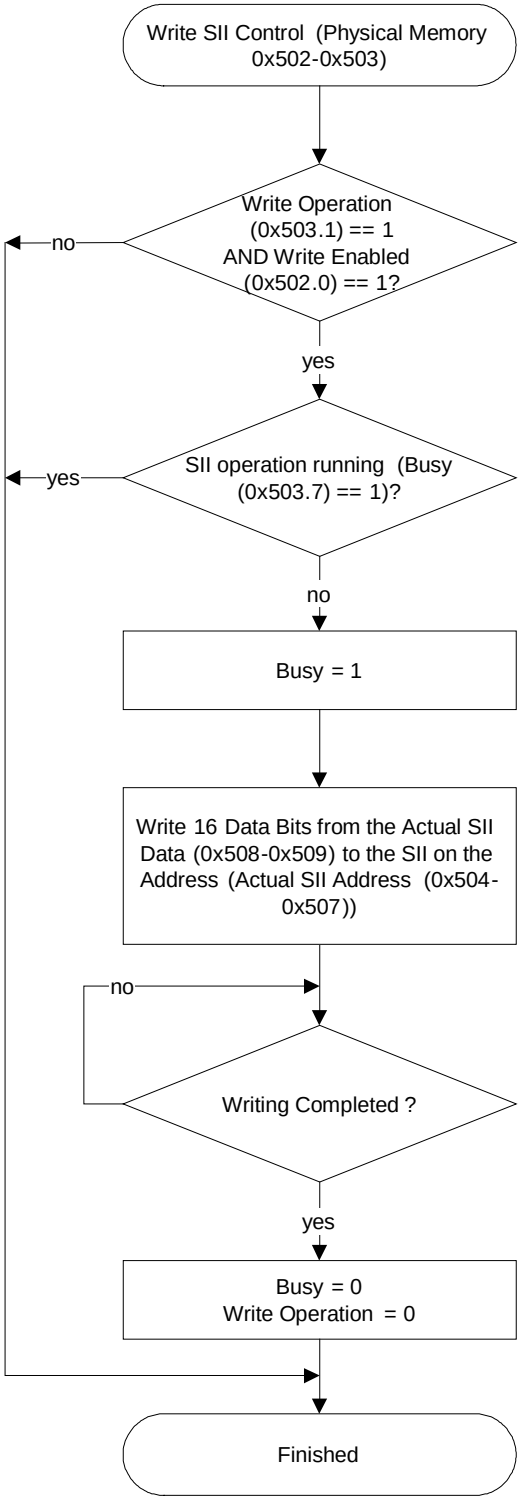


Figure 46 – Slave information interface write operation

8.2.5.4 Reload operation

Figure 47 shows the flow of a reload operation.

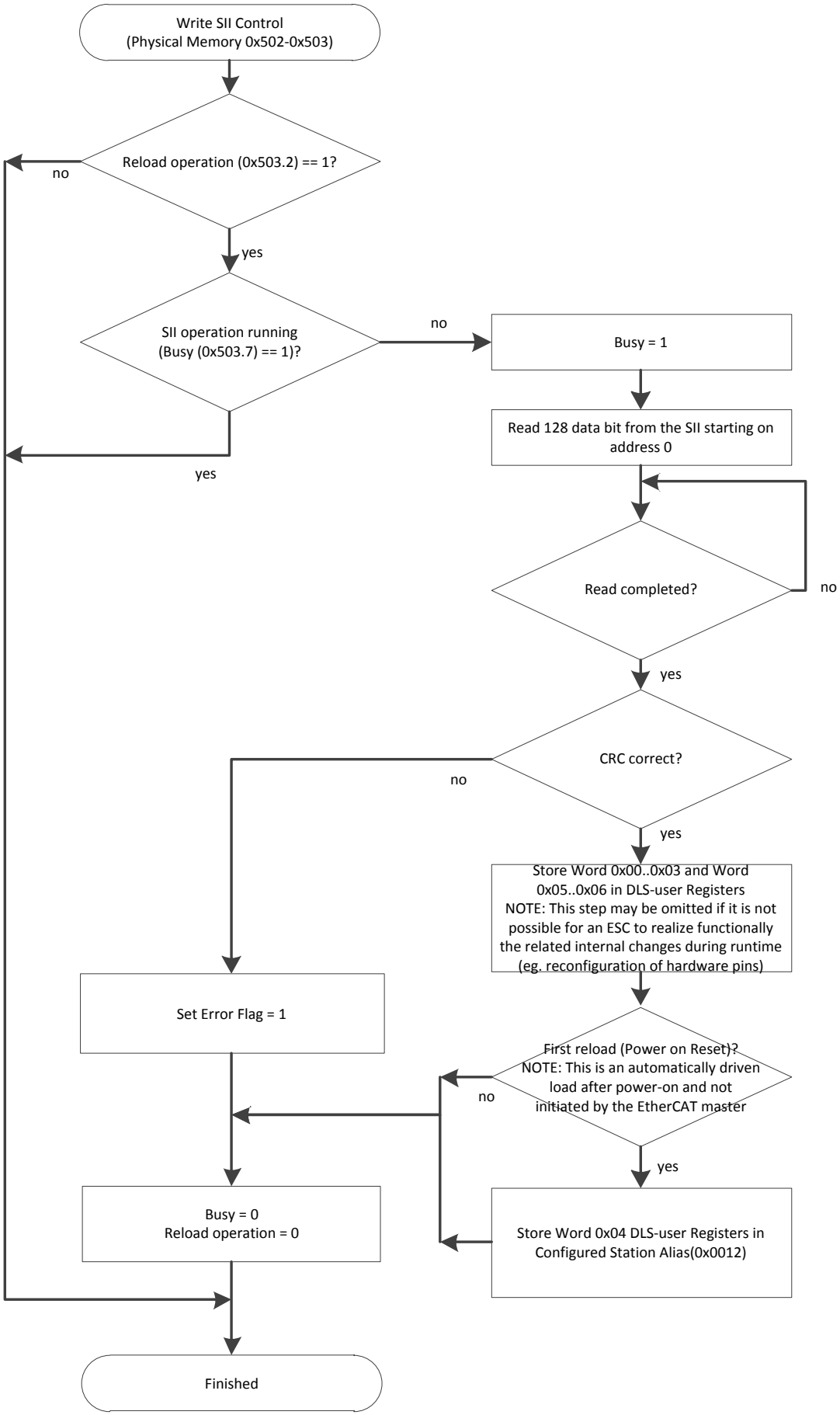


Figure 47 – Slave information interface reload operation

8.2.6 MII state machine (MIISM)

MIISM is responsible for access to MII (the media independent interface according to ISO/IEC 8802-3). The flow follows the structure as specified in 8.2.5 with distinct addresses for command, address and data buffer.

8.2.7 DC state machine (DCSM)

8.2.7.1 Description of the DC structure

DCSM handles the coordination of the local clock and the local clocks synchronization and time stamping capabilities. The DC registers are described in IEC 61158-3-12.

Distributed clocks enable all slave devices to have the same time. The first slave device within the segment that contains a clock is the clock reference. Its clock is used to synchronize the slave clocks of the other slave devices and of the master device. The master device sends a synchronisation PDU at certain intervals (as required in order to avoid the slave clock diverging beyond application specific limits), in which the slave device containing the reference clock enters its current time. The slave devices with slave clocks then read the time from the same PDU with ARMW service. Due to the logical ring structure this is possible since the reference clock is located before the slave clocks in the segment.

Since each slave introduces a small delay in the outgoing and return direction (within the device and also on the physical link), the propagation delay time between reference clock and the respective slave clock shall be considered during the synchronisation of the slave clocks. For measuring the propagation delay, the master device sends a broadcast write to a special address (the receive time register of port 0), which causes each slave device to save the time when the PDU was received (or its local clock time) in the outgoing direction and on the way back. The master can read these saved times and set up a delay register accordingly.

Definition of a reference clock
One slave will be used as a reference clock. The reference clock is the first clock between master and all the slaves to be synchronized. Reference clock distributes its clock cyclically with ARMW or FRMW command. The reference clock is adjustable from a “global” reference clock – such as IEC 61588.

Precondition
There is no mechanism to calculate the residence time. Thus, no significant jitter of residence time is allowed for all Type 12 slave devices.

Key features:

- Drift compensation to Reference Clock
- Propagation delay measurement

  Each slave controller measures the delay between the two directions of a frame

  Master calculates the propagation delays between all slaves
- Offset compensation to Reference Clock (System Time)

  Same absolute system time in all devices (jitter below 1 µs)

Figure 48 gives a structural overview of the DC element. It assumes a local clock rate of 100 MHz but a clock resolution of 1ns. This allows to adjust the local clock in small steps to the global clock rate and avoids jumps in the time scale.

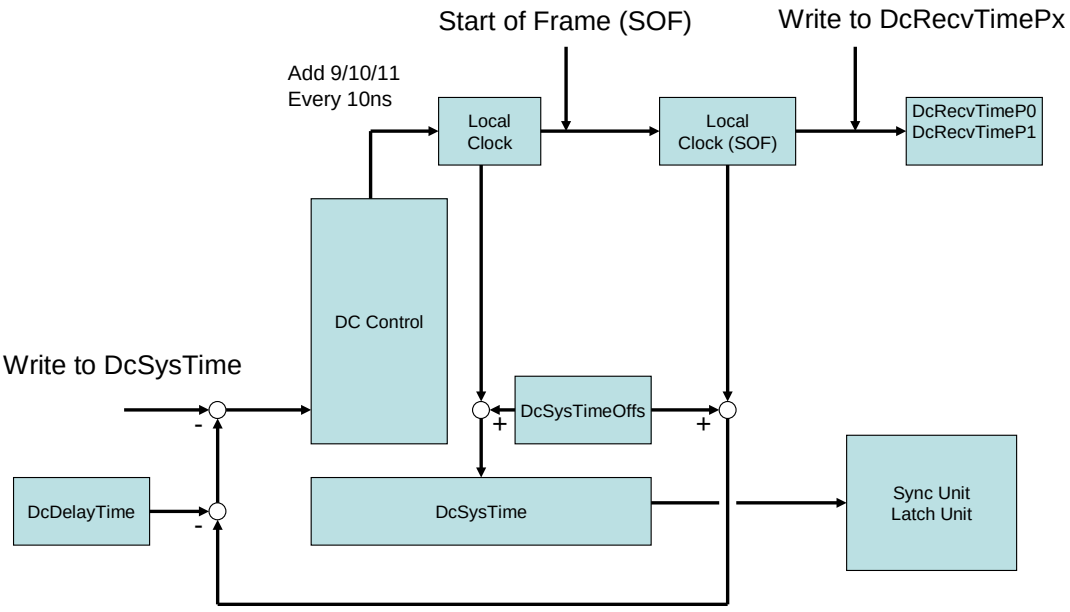


Figure 48 – Distributed clock

SOF is the beginning of the preamble of the Ethernet frame.

The local clock is incremented by 10 every 10 ns and depending on the drift of the clock by 9 or 11 on a regular time base specified in DC-control (for drift compensation).

The SysTimeOffset allows adaptation without changing the free running local clock. The delay as the second offset is used to compensate the delays from reference clock to slave clock.

External synchronisation is accomplished by mechanisms specified in IEC 61588. Any device with external communication interfaces may contain a boundary clock. The slave with the master clock is synchronized to the boundary clock. A Type 12 segment shall have only one active boundary clock at any time in order to meet the IEC 61588 topology requirements.

DC is used for very precise timing requirements. Systems with synchronisation needs in the range of 10 µs and higher or with other means of delay compensation may be synchronized by sharing a Type 12 PDUs accessing write buffers of the devices to be synchronized.

8.2.7.2 Delay measurement

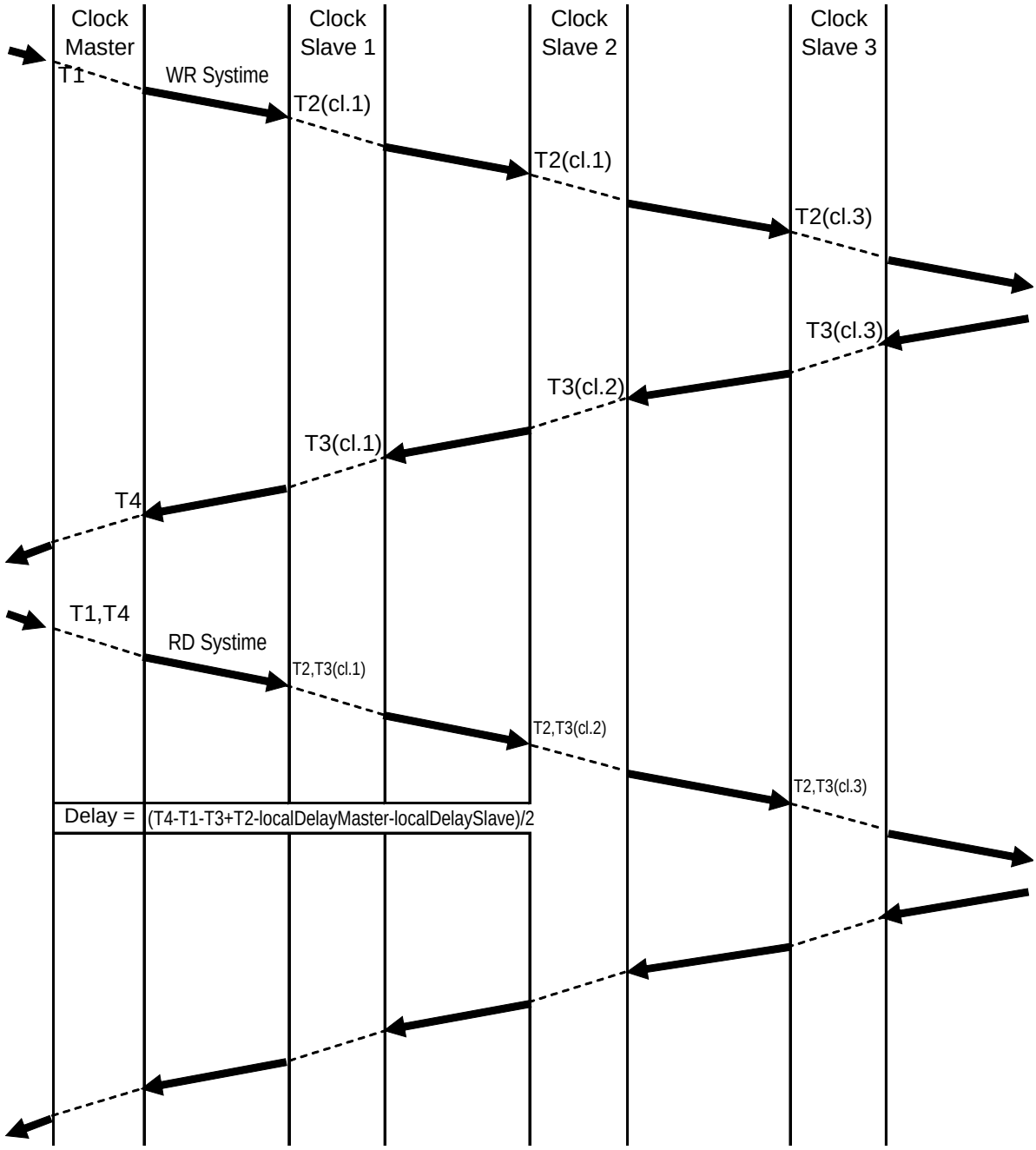


Figure 49 – Delay measurement sequence

Figure 49 shows the principle of delay measurement.

T1(of the clock master) and T2(of the various slave clocks 1,2,3 here denoted as cl.1, cl.2 and cl.3) refers to Receive time Port 0, T3(of the various slave clocks 1,2,3 here denoted as cl.1, cl.2 and cl.3) and T4 refers to Receive time Port 1. T2=T3 for the last slave clock. This model assumes symmetric connection i.e. the path from A to B is as long as the path from B to A. Different line delays and different propagation delays in the Ethernet PHYs may result in an constant time offset.

Annex A
(informative)

Type 12: Additional specifications on DL-Protocol state machines

NOTE 1 This annex specifies a number of finite state machines used by the DLE to provide its low-level and high-level protocol functions. This specification is complementary to the textual specification in the body of this standard; in case of conflict the requirements of the textual specification take precedence.

NOTE 2 The finite state machine descriptions given here are necessarily less than a complete description of an implementation. Additional requirements and considerations are found in the textual specification.

A.1 DHSM

A.1.1 Primitive definitions

A.1.1.1 Primitive exchanged between PSM and DHSM

Table A.1 shows primitives issued by DHSM to the PSM.

Table A.1 – Primitives issued by DHSM to PSM

Primitive name	Associated parameters
Tx request	Port, Byte, Data

Table A.2 shows primitives issued by the PSM to the DHSM.

Table A.2 – Primitives issued by PSM to DHSM

Primitive name	Associated parameters
Rx indication	Port, Byte, Data

A.1.1.2 Parameters of PSM Primitives

Table A.3 shows all parameters used with primitives between the DHSM and the PSM.

Table A.3 – Parameters used with primitives exchanged between DHSM and PSM

Parameter name	Description
Port	Identifier of the local port, starting with 0 as the primary port
Byte	Identifier of the octet received/transmitted as specified in Table A.4
Data	Value of the octet received/transmitted

Table A.4 – Identifier for the octets of a Ethernet frame

Parameter name	Description
0	first octet of Preamble
1	further octet of Preamble
2	first octet of DA
3	further octet of DA
4	first octet of SA
5	further octet of SA
6	first octet of VLAN
7	further octet of VLAN
8	first octet of Ethertype
9	second octet of Ethertype
10	First Octet of Ethernet SDU
10+ n	(n+1)-th Octet of Ethernet SDU
0xffff (END)	end of frame with correct FCS
0xfffe (ERR)	abort frame with error

State machine description

There exist exactly one DHSM per slave device.

The DHSM forms the interface between remote interaction and local memory. Each frame octets will be passed from port to port by DHSM.

If a Type 12 command is recognized an interaction with the SYM state machines will be issued if the local memory is addressed. The local actions will be invoked if the indication was issued at port 0. The only local action issued on the other ports is the time stamping of incoming frames.

This state machine describes the interpretation of Ethernet frames with a specific real time Ethertype or with a specific UDP destination port. Specific Type 12 handling as detection of circulating frames, incrementing auto increment address, updating of WKC and FCS will be done by DHSM.

The Error Handling is described at a logical level. For better localization of erroneous links the station detecting an error will forward with the damaged FCS a 4 bit physical symbol which would cause an alignment error. This alignment error in combination with a FCS which is inverted in the last 2 bits is a signal that the problem occurred on a different link. Each detected error causes an additional entry in the appropriate statistics attributes.

Local Constants
LASTP
Identifier of the last Ethernet port. Ports are numbered from 0 to LASTP.

END
Indicator of end of Ethernet frame.

ERR
Indicator of end of Ethernet frame due to an error condition.

Local Variables
CMD
Command identifier of a Type 12 PDU.

Etype1
First octet of Ethertype.

Length
Length of a Type 12 PDU.

MF
Indicates last Type 12 PDU in an Ethernet frame.

RxTimeLatch[0..LASTP]
Length of a Type 12 PDU.

AdL
First address octet of an Type 12 PDU.

AdH
Second address octet of a Type 12 PDU.

DaL
Third address octet of a Type 12 PDU.

DaH
Forth address octet of a Type 12 PDU.

LeL
First length octet of a Type 12 PDU.

LeH
Second length octet of a Type 12 PDU.

wkc
Local additive factor to be added to WKC in Type 12 PDU.

RData
Local data octet of memory element.

WData
Data octet of a Type 12 PDU to be written.

Overflow
Indicates overflow of an addition of two octets treated as unsigned integer.

State table nomenclature

The standard suffixes ".req", ".cnf" and ".ind" are used to indicate the request, confirm and indication primitives, respectively.

DHSM table

The DHSM State table is shown in Table A.5.

Table A.5 – DHSM state table

#	Current state	Event /condition ⇒action	Next state
1	ETH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ETH
2	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 0 => RxTimeLatch[Port] = CT Port = 1 Tx.req(Port,Byte,Data)	ETH
3	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 1 => Port = 1 Tx.req(Port,Byte,Data)	ETH
4	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 2 => Port = 1 INIT_FCS(Data) Tx.req(Port,Byte,Data)	ETH
5	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 3 => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
6	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 4 => Port = 1 Data = Data | 2 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
7	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 4 && Byte < 8 => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
8	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 8 => Port = 1 Etype1 = Data UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
9	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data== 0xA4 && Etype1== 0x88) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECAT
10	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data== 0x00 && Etype1== 0x08) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIP

#	Current state	Event /condition ⇒action	Next state
11	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data != 0xA4 || Etype1 != 0x88) && (Data != 0x00 || Etype1 != 0x08) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
12	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 9 => Port = 1 Tx.req(Port,Byte,Data)	ETH
13	EIP	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIP
14	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte < 10 => Port = 1 Byte = ERR Tx.req(Port,Byte,Data)	ETH
15	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && (Byte == END || Byte == ERR) => Port = 1 Tx.req(Port,Byte,Data)	ETH
16	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 && Data == 0x45 => Port = 1 Tx.req(Port,Byte,Data)	EIP
17	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 && Data !=0x45 => Port = 1 Tx.req(Port,Byte,Data)	ETH
18	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 10 && Byte < 19 => Port = 1 Tx.req(Port,Byte,Data)	EIP
19	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 19 && Data == 0x11 => Port = 1 Tx.req(Port,Byte,Data)	EIP
20	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 19 && Data !=0x11 => Port = 1 Tx.req(Port,Byte,Data)	ETH
21	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 19 && Byte < 32 => Port = 1 Tx.req(Port,Byte,Data)	EIP
22	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 32 && Data == 0x88 => Port = 1 Tx.req(Port,Byte,Data)	EIP

#	Current state	Event /condition ⇒action	Next state
23	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 32 && Data != 0x88 => Port = 1 Tx.req(Port,Byte,Data)	ETH
24	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 33 && Data == 0xA4 => Port = 1 Tx.req(Port,Byte,Data)	EIP
25	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 33 && Data != 0xA4 => Port = 1 Tx.req(Port,Byte,Data)	ETH
26	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 33 && Byte < 36 => Port = 1 Tx.req(Port,Byte,Data)	EIP
27	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 36 => Port = 1 Data = 0 Tx.req(Port,Byte,Data)	EIP
28	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 37 => Port = 1 Data = 0 Tx.req(Port,Byte,Data)	ECAT
29	ECAT	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECAT
30	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte < 10 => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
31	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 => Len = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECAT
32	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 11&& (Data & 0xf0 == 0x10) => Len = Len + (Data*256 & 0x0f) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECMD
33	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 11&& (Data & 0xf0 != 0x10) => Port = 1 Tx.req(Port,Byte,Data)	ETH

#	Current state	Event /condition ⇒action	Next state
34	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
35	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
36	ECMD	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECMD
37	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 => CMD = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIDX
38	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
39	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
40	EIDX	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIDX
41	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 => Cmd = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADL
42	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	Current state	Event /condition ⇒action	Next state
43	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
44	EADL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EADL
45	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == BRD, BWR, BRW, APRD, APWR, APRW,ARMW => AdL = Data Data = Data + 1 Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADH
46	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == other => AdL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADH
47	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
48	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
49	EADH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EADH
50	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == BRD, BWR, BRW => AdH = Data if AdL == 0xff then Data = Data + 1 Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL

#	Current state	Event /condition ⇒action	Next state
51	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == APRD, APWR, APRW,ARMW => AdH = Data if AdL == 0xff then Data = Data + 1 if Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
52	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == FPRD, FPWR, FPRW,FRMW => AdH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
53	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == LRD, LWR, LRW => AdH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
54	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == other => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
55	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
56	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
57	EDAL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDAL
58	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 => DaL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAH
59	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	Current state	Event /condition ⇒action	Next state
60	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
61	EDAH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDAH
62	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 => DaH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ELEL
63	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
64	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
65	ELEL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ELEL
66	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 => LeL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ELEH
67	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
68	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	Current state	Event /condition ⇒action	Next state
69	ELEH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ELEH
70	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && (!Closed[Port] || Data & 0x40 == 0) => if Closed[Port] then LeH = Data | 0x40 else LeH = Data MF = LeH & 0x80 Length = LeL + 256 * (LeH & 0x0f) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIRL
71	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && (Closed[Port] && Data & 0x40 != 0) => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
72	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
73	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
74	EIRL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIRL
75	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 => if Ena then Data == Data | (EventH & EventMskH) Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIRH
76	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	Current state	Event /condition ⇒action	Next state
77	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
78	EIRH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIRH
79	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length == 0 => wkc = 0 Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
80	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 && !Ena => wkc = 0 Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTI
81	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 && Ena => wkc = 0 Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
82	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
83	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
84	EDTI	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDTI
85	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length == 0 => Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL

#	Current state	Event /condition ⇒action	Next state
86	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 => Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTI
87	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
88	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
89	EDTA	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDTA
90	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && R_FMMUMATCH(LOGADD) => MemoryAddress = RFMMUMAP (LOGADD) WKC = 0 WData= Data Read.ind (Address, Data, WKC)	WSYLR
91	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && W_FMMUMATCH(LOGADD) => MemoryAddress = WFMMUMAP (LOGADD) WKC = 0 RData= Data Write.ind (Address, Data, WKC)	WSYLW
92	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && N_FMMUMATCH(LOGADD) && Length == 0 => INC(LOGADD) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
93	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && N_FMMUMATCH(LOGADD) && Length != 0 => Length -- INC(LOGADD) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
94	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDPR => WKC = 0 RData= Data Read.ind (Address, Data, WKC)	WSYPR

#	Current state	Event /condition ⇒action	Next state
95	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDPW => WKC = 0 WData= Data Write.ind (Address, Data, WKC)	WSYPW
96	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
97	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
98	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /W_FMMUMATCH(LOGADD) => wkc = wkc | WKC MemoryAddress = WFMMUMAP (LOGADD) RData = Data Data = WData WKC = 0 Write.ind (Address, Data, WKC)	WSYLW
99	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /!W_FMMUMATCH(LOGADD) && Length == 0 => wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
100	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /!W_FMMUMATCH(LOGADD) && Length != 0 => Length -- INC(LOGADD) wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
101	WSYLW	**Write.rsp (MemoryAddress, Data, WKC)** /Length == 0 => INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
102	WSYLW	**Write.rsp (MemoryAddress, Data, WKC)** /Length != 0 => Length -- INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA

#	Current state	Event /condition ⇒action	Next state
103	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /CMDPW => wkc = wkc | WKC if OR then Data = WData | Data RData = Data Data = WData WKC = 0 Write.ind (Address, Data, WKC)	WSYPW
104	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /!CMDPW && Length == 0 => if OR then Data = WData | Data wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
105	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /!CMDPW && Length != 0 => Length -- if OR then Data = WData | Data wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
106	WSYPW	**Write.rsp (MemoryAddress, Data, WKC)** /Length == 0 => INC(LOGADD) wkc = wkc | (WKC * CMDPRMW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
107	WSYPW	**Write.rsp (MemoryAddress, Data, WKC)** /Length != 0 => Length -- INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
108	EWKL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EWKL
109	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 => Overflow, Data = Data + wkc Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKH
110	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	Current state	Event /condition ⇒action	Next state
111	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
112	EWKH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECMD
113	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && MF => Data = Data + Overflow Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
114	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && !MF => Data = Data + Overflow Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EFCS1
115	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
116	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
117	EFCS1	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS1
118	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS1 Port 1 Tx.req(Port,Byte,Data)	EFCS2
119	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	Current state	Event /condition ⇒action	Next state
120	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
121	EFCS2	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS2
122	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS2 Port 1 Tx.req(Port,Byte,Data)	EFCS3
123	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
124	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
125	EFCS3	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS3
126	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS3 Port 1 Tx.req(Port,Byte,Data)	EFCS4
127	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
128	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	Current state	Event /condition ⇒action	Next state
129	EFCS4	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS4
130	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS4 Port 1 Tx.req(Port,Byte,Data)	EFCS5
131	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
132	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
133	EFCS5	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS5
134	EFCS5	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = END Success = TRUE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
135	EFCS5	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte != END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

Functions

The DHSM Functions are summarized in Table A.6.

Table A.6 – DHSM function table

Function name	Operations
INIT_FCS(Data)	Initiate FCS with 0xFFFFFFFF
UPD_FCS(Data)	Updates FCS according to ISO/IEC 8802-3
CMDL	Cmd == LRD, LRW, LWR
CMDPR	Cmd == BRD, BRW || (Cmd == ARD, ARW, ARMW && AdH, AdL == 0) || (Cmd == FRD, FRW, FRMW && AdH, AdL==ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled)
CMDPW	Cmd == BWR; BRW || (Cmd == AWR, ARW && (AdH, AdL-1) == 0) || (Cmd == FWR, FRW && AdH, AdL==ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled) || (Cmd == ARMW && (AdH, AdL-1) != 0) || (Cmd == FRMW && AdH, AdL != ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled)
CMDLRW	if Cmd == LRW then 2 else 1
CMDRMW	if Cmd == ARMW,FRMW then 0 else 1
RFMMUMAP (LOGADD)	Map LOGADD to Address in memory space using read FMMU
WFMMUMAP (LOGADD)	Map LOGADD to Address in memory space using write FMMU
R_FMMUMATCH (LOGADD)	LOGADD can be mapped onto a read to an address in memory && Cmd == LRD, LRW
W_FMMUMATCH (LOGADD)	LOGADD can be mapped onto a write to an address in memory && Cmd == LWR, LRW
N_FMMUMATCH (LOGADD)	!W_FMMUMATCH (LOGADD) && !R_FMMUMATCH (LOGADD)
OR	Cmd == BRD, BRW
INC(LOGADD)	Overflow, AdL = AdL+ 1 Overflow, AdH= AdH + Overflow Overflow, DaL= DaL + Overflow Overflow, DaH= DaH + Overflow

A.2 SYSM

A.2.1 Primitive definition

A.2.1.1 Primitive exchanged between DHSM and SYSM

Table A.7 shows primitives issued by SYSM to the DHSM.

Table A.7 – Primitives issued by SYSM to DHSM

Primitive name	Associated parameters
Read response	Address, Data, WKC
Write response	Address, Data, WKC

Table A.8 shows primitives issued by the DHSM to the SYSM.

Table A.8 – Primitives issued by DHSM to SYSM

Primitive name	Associated parameters
Read indication	Address, Data, WKC
Write indication	Address, Data, WKC
Terminate indication	Success

A.2.1.2 Primitive exchanged between DL-User and SYSM

Table A.9 shows primitives issued by the DL-User to the SYSM.

Table A.9 – Primitives issued by DL-User to SYSM

Primitive name	Associated parameters
DL-Read local request	Address, Length
DL-Write local request	Address, Length, Data

Table A.10 shows primitives issued by SYSM to the DL-User.

Table A.10 – Primitives issued by SYSM to DL-User

Primitive name	Associated parameters
DL-Read local confirmation	L-Status, Data
DL-Write local confirmation	L-Status

NOTE Local events are not modeled as they do not have impact to SYSM.

A.2.1.3 Parameters of DHSM Primitives

Table A.11 shows all parameters used with primitives between the SYSM and the DHSM.

Table A.11 – Parameters used with primitives exchanged between SYSM and DHSM

Parameter name	Description
WKC	Working counter of local access
Address	Identifier of the physical memory area
Data	Value of the octet received/transmitted

State machine description

There exist a SYSM for each sync manager channel. The abstract model is that Type 12 Read/Write indication primitives and Read/Write Local request primitives are passed to the first SYSM and executed if there is an address match or passed to the next SYSM. If no SYSM respond to the service primitive a physical memory handler will execute the service request if the memory or register is present and the service type is enabled for this register. A register write access will be done if the Ethernet frame is parsed successfully by DHSM. Read access to the first octet of word or double word registers are executed in an atomic way, i.e. reading the first octet will freeze the value for further access.

There are buffered types and mailbox type SYSM. The local requests are modelled in that way, that the read/write access is completely within the boundary or completely outside the boundary of a sync manager area.

The SM events are generated when the associated sync manager channel register area is written.

Local variables

eact
Activation of a buffer by master.

uact
Activation of a buffer by DLS-user.

Terminate
Indicates the Termination of a mailbox or buffer transaction.

User
Contains user buffer.

Buffer
Contains buffer used for communication.

Next
Contains buffer filled for next use.

Free
Contains free buffer.

p1, p2, p3
Memory location of the three buffer, first is located as specified by SM, others are filled on the following locations.

act
Contains activity code of a mailbox.

State table nomenclature

The standard suffixes ".req", ".cnf" and ".ind" are used to indicate the request, confirm and indication primitives, respectively.

SYSM table

The SYSM State Table is shown in Table A.12.

Table A.12 – SYSM state table

#	Current State	Event /Condition =>Action	Next State

1	any state	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 0 && SM.Direction == 0
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		eact, uact = FALSE
		Terminate = FALSE
		next=NIL,buffer=p0,user=p1,free=p2
		SM.bufferedState=3
		if (AL Event Enable) then Enable SM Event (Toggle)
			BR-IDLE

2	any state	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 0 && SM.Direction == 1
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		eact, uact = FALSE
		Terminate = FALSE
		next=NIL, buffer =p0,user=p1,free=p2
		SM.bufferedState=3
		if (AL Event Enable) then Enable SM Event (Toggle)
			BW-IDLE

3	any state	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 2 && SM.Direction == 0
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		Terminate = FALSE
		act = 0
		SM.mailboxState=0
		if (AL Event Enable) then Enable SM Event (Toggle)
			MR-IDLE

4	any state	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 2 && SM.Direction == 1
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		Terminate = FALSE
		act = 0
		SM.mailboxState=0
		if (AL Event Enable) then Enable SM Event (Toggle)
			MW-IDLE

5	any state	**SM Event** /(!SM.ChannelEnable || SM.ChannelDisable) || SM.Buffer Type == 1,3 =>	OFF
6	OFF	**DL-Write Local.req (Address, Length, Data)** => pass next	OFF
7	OFF	**DL-Read Local.req (Address, Length)** => pass next	OFF
8	OFF	**Write.ind (Address, Data, WKC)** => pass next	OFF
9	OFF	**Read.ind (Address, Data, WKC)** => pass next	OFF
10	OFF	**Terminate.ind(Success)** => pass next	OFF

#	Current State	Event /Condition =>Action	Next State
11	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && !uact => (user. Address) = Data L-Status = OK uact = TRUE SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	BR-IDLE
12	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && uact => (user. Address) = Data L-Status = OK SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	BR-IDLE
13	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && !uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
14	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK uact = FALSE SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
15	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /NA&&AE && !uact => L-Status = WRNGSEQ DL-Write Local.cnf (L-Status)	BR-IDLE
16	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&E && uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK uact = FALSE SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
17	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&NE && uact => (user. Address) = Data L-Status = OK DL-Write Local.cnf (L-Status)	BR-IDLE
18	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BR-IDLE
19	BR-IDLE	**DL-Read Local.req (Address, Length)** => pass next	BR-IDLE

#	Current State	Event /Condition =>Action	Next State
20	BR-IDLE	**Write.ind (Address, Data, WKC)** => pass next	BR-IDLE
21	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && !eact => if next != NIL then free = buffer, buffer = next, next = NIL Data = (buffer. Address) eact = TRUE WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
22	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && eact => Data = (buffer. Address) WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
23	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && !eact => if next != NIL then free = buffer, buffer = next, next = NIL Data = (buffer. Address) WKC = WKC +1 eact = TRUE Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
24	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && eact => /*Buffer exchange possible*/ Data = (buffer. Address) WKC = WKC +1 Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
25	BR-IDLE	**Read.ind (Address, Data, WKC)** /NA&&AE && !eact => Read.rsp (Address, Data, WKC)	BR-IDLE
26	BR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&E && eact => Data = (buffer. Address) WKC = WKC +1 Terminate = TRUE Read.rsp (Address, Data, WKC)	BR-IDLE
27	BR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&NE && eact => Data = (buffer. Address) WKC = WKC +1 Read.rsp (Address, Data, WKC)	BR-IDLE
28	BR-IDLE	**Read.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BR-IDLE

#	Current State	Event /Condition =>Action	Next State
29	BR-IDLE	**Terminate.ind(Success)** /Success && Terminate => eact = FALSE Terminate = FALSE SM.ReadEvent = 1 pass next	BR-IDLE
30	BR-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && eact) then eact = FALSE pass next	BR-IDLE
31	BR-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE eact = FALSE pass next	BR-IDLE
32	BW-IDLE	**DL-Write Local.req (Address, Length, Data)** => pass next	BW-IDLE
33	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && !uact => if next != NIL then free = user, user = next, next = NIL Data = (user. Address) L-Status = OK uact = TRUE SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
34	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && uact => Data = (user. Address) L-Status = OK SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
35	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && !uact => if next != NIL then free = user, user = next, next = NIL Data = (user. Address) L-Status = OK SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
36	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && uact => Data = (user. Address) L-Status = OK uact = FALSE SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
37	BW-IDLE	**DL-Read Local.req (Address, Length)** /NA&&AE && !uact => L-Status = WRNGSEQ DL-Read Local.cnf (L-Status, Data)	BW-IDLE

#	Current State	Event /Condition =>Action	Next State
38	BW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&E && uact => Data = (user. Address) L-Status = OK uact = FALSE SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
39	BW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&NE && uact => Data = (user. Address) L-Status = OK DL-Read Local.cnf (L-Status, Data)	BW-IDLE
40	BW-IDLE	**DL-Read Local.req (Address, Length)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BW-IDLE
41	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && !eact => (buffer. Address) = data eact = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
42	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && eact => (buffer. Address) = data WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
43	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && !eact => (buffer. Address) = data WKC = WKC +1 eact = TRUE Terminate = TRUE SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
44	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && eact => (buffer. Address) = data Terminate = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
45	BW-IDLE	**Write.ind (Address, Data, WKC)** /NA&&AE && !eact => Write.rsp (Address, Data, WKC)	BW-IDLE
46	BW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&E && eact => (buffer. Address) = data WKC = WKC +1 Terminate = TRUE Write.rsp (Address, Data, WKC)	BW-IDLE

#	Current State	Event /Condition =>Action	Next State
47	BW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&NE && eact => (buffer. Address) = data WKC = WKC +1 Write.rsp (Address, Data, WKC)	BW-IDLE
48	BW-IDLE	**Write.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BW-IDLE
49	BW-IDLE	**Read.ind (Address, Data, WKC)** => pass next	BW-IDLE
50	BW-IDLE	**Terminate.ind(Success)** /Success && Terminate => eact = FALSE Terminate = FALSE if next == NIL then next = buffer, buffer = free, free = NIL else user <=> next SM.bufferedState=next buffer number SM.WriteEvent = 1 pass next	BW-IDLE
51	BW-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && eact) then eact = FALSE pass next	BW-IDLE
52	BW-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE eact = FALSE pass next	BW-IDLE
53	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && act ==0 => (user. Address) = Data act = 1 L-Status = OK SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	MR-IDLE
54	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&AE && act != 0 => L-Status = NODATA DL-Write Local.cnf (L-Status)	MR-IDLE
55	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && act == 0 => (user. Address) = Data act = 2 SM.mailboxState=1 L-Status = OK SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	MR-IDLE
56	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&AE && && act != 1 => L-Status = NODATA DL-Write Local.cnf (L-Status)	MR-IDLE

#	Current State	Event /Condition =>Action	Next State
57	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&E && act == 1 => (user. Address) = Data act = 2 SM.mailboxState=1 L-Status = OK SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	MR-IDLE
58	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&NE && act == 1 => (user. Address) = Data L-Status = OK DL-Write Local.cnf (L-Status)	MR-IDLE
59	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MR-IDLE
60	MR-IDLE	**DL-Read Local.req (Address, Length)** => pass next	MR-IDLE
61	MR-IDLE	**Write.ind (Address, Data, WKC)** => pass next	MR-IDLE
62	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && act ==2 => Data = (buffer. Address) act = 3 WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	MR-IDLE
63	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&AE && act != 2 => Read.rsp (Address, Data, WKC)	MR-IDLE
64	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && act == 2 => Data = (buffer. Address) WKC = WKC +1 act = 4 Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	MR-IDLE
65	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&AE && && act != 3 => Read.rsp (Address, Data, WKC)	MR-IDLE
66	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&E && act == 3 => Data = (buffer. Address) WKC = WKC +1 act = 4 Terminate = TRUE Read.rsp (Address, Data, WKC)	MR-IDLE
67	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&NE && act == 3 => Data = (buffer. Address) WKC = WKC +1 Read.rsp (Address, Data, WKC)	MR-IDLE

#	Current State	Event /Condition =>Action	Next State
68	MR-IDLE	**Read.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MR-IDLE
69	MR-IDLE	**Terminate.ind(Success)** /Success && Terminate => Terminate = FALSE act = 0 SM.mailboxState=0 SM.ReadEvent = 1 pass next	MR-IDLE
70	MR-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && act == 3) then act = 2 pass next	MR-IDLE
71	MR-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE act = 2 pass next	MR-IDLE
72	MW-IDLE	**DL-Write Local.req (Address, Length, Data)** => pass next	MW-IDLE
73	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && act ==3 => Data = (User. Address) L-Status = OK act = 4 SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	MW-IDLE
74	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&AE && act != 3 => L-Status = NODATA DL-Read Local.cnf (L-Status, Data)	MW-IDLE
75	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && act == 3 => Data = (User. Address) L-Status = OK act = 0 SM.mailboxState=0 SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	MW-IDLE
76	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&AE && && act != 4 => L-Status = NODATA DL-Read Local.cnf (L-Status, Data)	MW-IDLE
77	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&E && act == 4 => Data = (User. Address) L-Status = OK act = 0 SM.mailboxState=0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	MW-IDLE

#	Current State	Event /Condition =>Action	Next State
78	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&NE && act == 4 => Data = (User. Address) L-Status = OK DL-Read Local.cnf (L-Status, Data)	MW-IDLE
79	MW-IDLE	**DL-Read Local.req (Address, Length)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MW-IDLE
80	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && act ==0 => (buffer. Address) = Data act = 1 WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	MW-IDLE
81	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&AE && act != 0 => Write.rsp (Address, Data, WKC)	MW-IDLE
82	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && act == 0 => (buffer. Address) = Data act = 2 Terminate = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	MW-IDLE
83	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&AE && && act != 1 => Write.rsp (Address, Data, WKC)	MW-IDLE
84	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&E && act == 1 => (buffer. Address) = Data act = 2 Terminate = TRUE WKC = WKC +1 Write.rsp (Address, Data, WKC)	MW-IDLE
85	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&NE && act == 1 => (buffer. Address) = Data WKC = WKC +1 Write.rsp (Address, Data, WKC)	MW-IDLE
86	MW-IDLE	**Write.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MW-IDLE
87	MW-IDLE	**Read.ind (Address, Data, WKC)** => pass next	MW-IDLE

#	Current State	Event /Condition =>Action	Next State
88	MW-IDLE	**Terminate.ind(Success)** /Success && Terminate => Terminate = FALSE act = 3 SM.mailboxState=1 SM.WriteEvent = 1 pass next	MW-IDLE
89	MW-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && act == 1) then act = 0 pass next	MW-IDLE
90	MW-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE act = 0 pass next	MW-IDLE

Functions

The SYSM Functions are summarized in Table A.13.

Table A.13 – SYSM function table

Function name	Operations
A	Address == SM.PhysicalStartAddress
NA	Address > SM.PhysicalStartAddress
E	Address + Length == SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter
NE	Address + Length < SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter
AE	Address + Length =< SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter

A.3 RMSM

A.3.1 Primitive definitions

A.3.1.1 Primitive exchanged between SYSM and RMSM

Table A.14 shows primitive issued by the RMSM to the SYSM.

Table A.14 – Primitives issued by RMSM to SYSM

Primitive name	Associated parameters
DL-Write local request	Address, Length, Data

Table A.15 shows primitive issued by SYSM to the RMSM.

Table A.15 – Primitives issued by SYSM to RMSM

Primitive name	Associated parameters
DL-Write local confirmation	L-Status

NOTE Local events are not modeled as they do not have impact to RMSM.

A.3.1.2 Parameters of SYSM Primitives

Table A.16 shows all parameters used with primitives between the RMSM and the SYSM.

Table A.16 – Parameters used with primitives exchanged between RMSM and SYSM

Parameter name	Description
Address	Identifier of the physical memory area
Length	Length of the octets transmitted
Data	Value of the octet transmitted

State machine description

There exist a RMSM for each read mailbox sync manager channel.

The RMSM writes the read mailbox on user request. Only one user request (Read Upd) is accepted. If the data are read by the master the RMSM will store the data in an auxiliary buffer.

A change in the toggle will result in an restore of the old mailbox data (even if there was a new mailbox update in between).

Local variables

Tggl
Contains Toggle Flag at the last SM Event.

Boxstate
Signals State of the Box.

BackupBox
Storage of the mailbox PDU for backup purposes.

ActualBox
Virtual storage of the actual mailbox PDU.

SeqN
Contains the sequence number for the next service.

State table nomenclature

The standard suffixes ".req", ".cnf" and ".ind" are used to indicate the request, confirm and indication primitives, respectively.

RMSM table

The RMSM State Table is shown in Table A.17.

Table A.17 – RMSM state table

#	Current state	Event /condition ⇒action	Next state
1	OFF	**Enable SM Event (Toggle)** => Tggl=Toggle Boxstate = 0 SeqN = 0 BackupBox = ERR PDU with no Error(0,0,0,0)	IDLE
2	OFF	**Disable SM Event** =>	OFF
3	IDLE	**Enable SM Event (Toggle)** =>	IDLE
4	IDLE	**Disable SM Event** => Terminate Segmented Services	OFF
5	IDLE	**Toggle SM Event (Toggle)** /Tggl != Toggle =>	IDLE
6	IDLE	**Toggle SM Event (Toggle)** /Tggl == Toggle => ActualBox = BackupBox Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read Boxstate = 1 Tggl = Toggle DL-Write Local.req(Address, Length, Data) write SM status(Toggle)	SEND
7	IDLE	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** => ActualBox = Parameter from event service primitive Update (SeqN) Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read Boxstate = 0 DL-Write Local.req(Address, Length, Data)	SEND
8	IDLE	**DL-Write Local.cnf (L-Status)** => ignore	IDLE
9	SEND	**Enable SM Event (Toggle)** =>	IDLE
10	SEND	**Disable SM Event** => Terminate Segmented Services	OFF
11	SEND	**Toggle SM Event (Toggle)** /Tggl != Toggle =>	SEND
12	SEND	**Toggle SM Event (Toggle)**/Tggl == Toggle && Boxstate == 0=>ActualBox <=> BackupBox (Exchange) Address = SM1.PhysicalStartAddress Length = SM1.LengthData = encode Mailbox ReadBoxstate = 2DL-Write Local.req(Address, Length, Data)write SM status(Toggle)	SEND
13	SEND	**Toggle SM Event (Toggle)** /Tggl == Toggle && Boxstate == 1, 2 => write SM status(Toggle)	SEND

#	Current state	Event /condition ⇒action	Next state
14	SEND	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** /Boxstate == 1 => BackupBox = Parameter from event service primitive Update (SeqN) Boxstate = 2	SEND
15	SEND	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** /Boxstate == 0, 2 => invalid user sequence	SEND
16	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 0 => DL_status = L-Status BackupBox =ActualBox DL-Mailbox Read Upd.cnf (DL_status)	IDLE
17	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 1 => DL_status = L-Status DL-Mailbox Read Upd.cnf (DL_status)	IDLE
18	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 2 => ActualBox = BackupBox Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read DL_status = L-Status Boxstate = 0 DL-Mailbox Read Upd.cnf (DL_status) DL-Write Local.req(Address, Length, Data)	SEND

Functions

The RMSM Functions are summarized in Table A.18.

Table A.18 – RMSM function table

Function name	Operations
Update (SeqN)	if (SeqN < 7) then SeqN = SeqN + 1 else SeqN = 1

Bibliography

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

IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-5-12:2014, *Industrial communication networks – Fieldbus specifications – Part 5-12: Application layer service definition – Type 12 elements*

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

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

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

ISO/IEC/TR 8802-1, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 1: Overview of Local Area Network Standards*

ISO/IEC 15802-1, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Common specifications – Part 1: Medium Access Control (MAC) service definition*

ITU-T V.41, *Code-independent error-control system*

ANSI X3.66 (R1990), *Advanced data communication control procedures (ADCCP)*

IEEE 802.1D, *IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Bridges,* available at <http://www.ieee.org>

IETF RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets : MIB-II,* available at <http://www.ietf.org>

IETF RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types,* available at <http://www.ietf.org>

_____

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 4-12: Spécification du protocole de la couche liaison de données – Éléments de type 12

AVANT-PROPOS

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

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 CEI 61784-1 et la CEI 61784-2.

La Norme internationale CEI 61158-4-12 a été établie par le sous-comité 65C: Réseaux industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Cette troisième édition annule et remplace la deuxième édition parue en 2010. Elle constitue une révision technique. Les principales modifications par rapport à l'édition précédente sont énumérées ci-dessous:

- réparations des erreurs et
- améliorations rédactionnelles.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65C/762/FDIS	65C/772/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

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

Une liste de toutes les parties de la série CEI 61158, publiées sous le titre général *Réseaux de communications industriels – Spécifications des bus de terrain*, est disponible sur le site web de la CEI.

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

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

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

INTRODUCTION

La présente partie de la CEI 61158 appartient à la série de normes visant à faciliter l'interconnexion des composants du système d'automatisation. Elle est liée aux autres normes de la série telle que définie par le modèle de référence de bus de terrain «à trois couches» décrit dans la CEI 61158-1:2013.

Le protocole de liaison de données assure un service de liaison de données en s'appuyant sur les services offerts par la couche physique. La présente norme a pour principal objet de fournir un ensemble de règles de communication, exprimées sous la forme des procédures que doivent réaliser des entités de liaison de données homologues (DLE) au moment de la communication. Ces règles de communication ont pour vocation de fournir une base de développement stable visant à atteindre différents objectifs:

a) guider les développeurs et les concepteurs;

b) réaliser les essais et acquérir l'équipement;

c) dans le cadre d'un accord d'intégration des systèmes dans l'environnement de systèmes ouverts;

d) dans le cadre d'une meilleure compréhension des communications à contrainte de temps au sein de l'OSI.

La présente norme porte en particulier sur la communication et l'interfonctionnement des capteurs, des effecteurs et d'autres appareils d'automatisation. Grâce à la présente norme associée à d'autres normes des modèles de référence OSI ou de bus de terrain, des systèmes par ailleurs incompatibles peuvent fonctionner ensemble, quelle que soit leur combinaison.

NOTE L'utilisation de certains des types de protocoles associés est limitée par les détenteurs de leurs droits de propriété intellectuelle. Dans tous les cas, l'engagement visant à limiter l'abandon des droits de propriété intellectuelle prévus par les détenteurs de ces droits permet d'utiliser un type de protocole de couche de liaison de données particulier avec les protocoles de couche physique et de couche d'application dans les combinaisons de type, comme spécifié explicitement dans les parties relatives au profil. L'utilisation de différents types de protocole dans d'autres combinaisons peut impliquer d'obtenir l'autorisation auprès de leurs détenteurs de droit de propriété intellectuelle respectifs.

La Commission Électrotechnique Internationale (CEI) attire l'attention sur le fait qu'il est déclaré que la conformité avec le présent document peut impliquer l'utilisation de brevets relatifs aux éléments de Type 12 et éventuellement aux autres types présentés ci-dessous:

EP 1 590 927 B1	[BE]	Koppler für ein Netzwerk mit Ringtopologie und ein auf Ethernet basierten Netzwerk
EP 1 789 857 B1	[BE]	Datenübertragungsverfahren und automatisierungssystem zum Einsatz eines solchen Datenübertragungsverfahrens
DE 102007017835.4	[BE]	Paketvermittlungsvorrichtung und lokales Kommunikationsnetz mit einer solchen Paketvermittlungsvorrichtung
EP 1 456 722 B1	[BE]	Datenübertragungsverfahren, serielles Bussystem und Anschalteinheit für einen passiven Busteilnehmer

La CEI 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 la CEI qu'il consent à négocier des licences avec des demandeurs du monde entier, soit sans frais soit à des termes et conditions raisonnables et non discriminatoires. À ce propos, la déclaration du détenteur des droits de propriété est enregistrée à la CEI. Des informations peuvent être demandées à:

[BE]: Beckhoff Automation GmbH
Eiserstraße 5
33415 Verl,
Allemagne

L'attention est d'autre part attirée sur le fait que certains des éléments du présent document peuvent faire l'objet de droits de propriété autres que ceux qui ont été mentionnés ci-dessus. La CEI ne saurait être tenue pour responsable de l'identification de ces droits de propriété en tout ou partie.

L'ISO (www.iso.org/patents) et la CEI (http://patents.iec.ch) maintiennent des bases de données, consultables en ligne, des droits de propriété pertinents à leurs normes. Les utilisateurs sont encouragés à consulter ces bases de données pour obtenir l'information la plus récente concernant les droits de propriété.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 4-12: Spécification du protocole de la couche liaison de données – Éléments de type 12

1 Domaine d'application

1.1 Généralités

La couche de liaison de données assure les communications de messagerie à contrainte de temps de base entre les appareils d'un environnement d'automatisation.

Ce protocole offre des opportunités de communication à toutes les entités de liaison de données participantes

a) de manière cyclique et synchrone, et

b) de manière cyclique ou acyclique asynchrone, comme demandé par chaque cycle de chacune de ces entités de liaison de données.

Par conséquent, ce protocole peut se caractériser comme assurant un accès cyclique et acyclique asynchrone, mais avec un redémarrage synchrone de chaque cycle.

1.2 Spécifications

La présente norme spécifie

a) les procédures de transfert de données et d'informations de commande d'une entité utilisateur de liaison de données vers une ou plusieurs entités utilisateur;

b) la structure des DLPDU utilisées par le protocole de la présente norme pour le transfert des données et des informations de commande, et leur représentation sous forme d'unités de données d'interface physique.

1.3 Procédures

Les procédures sont définies en termes

a) d'interactions entre les entités DL (DLE) par l'échange de DLPDU;

b) d'interactions entre un fournisseur de service DL (DLS) et un utilisateur DLS au sein du même système par l'échange de primitives DLS;

c) d'interactions entre un fournisseur DLS et les services MAC de l'ISO/CEI 8802-3.

1.4 Applicabilité

Ces procédures s'appliquent aux instances de communication entre des systèmes qui prennent en charge des services de communications à contrainte de temps dans la couche de liaison de données du modèle de référence OSI, et qui peuvent être connectés dans un environnement d'interconnexion de systèmes ouverts.

Les profils sont un moyen simple à plusieurs attributs de récapituler les capacités d'une mise en œuvre, et donc son applicabilité en fonction des différents besoins de communications à contrainte de temps.

1.5 Conformité

La présente norme spécifie également les exigences de conformité relatives aux systèmes mettant en œuvre ces procédures. La présente partie de la norme ne comporte aucun essai visant à démontrer la conformité à ces exigences.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. 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 CEI 61158, ainsi que la CEI 61784-1 et la CEI 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.

CEI 61158-3-12, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 3-12: Définition des services de la couche liaison de données – Éléments de type 12*

IEC 61588, *Precision clock synchronization protocol for networked measurement and control systems* (disponible en anglais seulement)

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Le modèle de base*

ISO/CEI 7498-3, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Dénomination et adressage*

ISO/IEC 8802-3:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications* (disponible en anglais seulement)

ISO/IEC 9899, *Information technology – Programming Languages – C* (disponible en anglais seulement)

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base – Conventions pour la définition des services OSI*

IEEE 802.1Q, *IEEE Standard for Local and metropolitan Area Networks – Virtual Bridged Local Area Networks,* disponible à l'adresse <http://www.ieee.org>

IETF RFC 768, *User Datagram Protocol (UDP),* disponible à l'adresse <http://www.ietf.org>

IETF RFC 791, *Internet protocol DARPA internet program protocol specification*, disponible à l'adresse <http://www.ietf.org>

3 Termes, définitions, symboles, abréviations et conventions

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

3.1 Termes et définitions du modèle de référence

La présente norme repose en partie sur les concepts développés dans l'ISO/CEI 7498-1 et l'ISO/CEI 7498-3, et utilise les termes suivants.

3.1.1 transmission duplex DL [ISO/CEI 7498-1]

3.1.2 protocole DL [ISO/CEI 7498-1]

3.1.3 unité de données de protocole DL [ISO/CEI 7498-1]

3.1.4 entité (N) [ISO/CEI 7498-1]
entité DL
entité Ph

3.1.5 unité de données d'interface (N) [ISO/CEI 7498-1]
unité de données de service DL (N=2)
unité de données d'interface Ph (N=1)

3.1.6 couche (N) [ISO/CEI 7498-1]
couche DL (N=2)
couche Ph (N=1)

3.1.7 service (N) [ISO/CEI 7498-1]
service DL (N=2)
service Ph (N=1)

3.1.8 point d'accès au service (N) [ISO/CEI 7498-1]
point d'accès au service DL (N=2)
point d'accès au service Ph (N=1)

3.1.9 adresse de point d'accès au service (N) [ISO/CEI 7498-1]
adresse de point d'accès au service DL (N=2)
adresse de point d'accès au service Ph (N=1)

3.1.10 entités homologues [ISO/CEI 7498-1]

3.1.11 données d'interface Ph [ISO/CEI 7498-1]

3.1.12 nom de primitive [ISO/CEI 7498-3]

3.1.13 réassemblage [ISO/CEI 7498-1]

3.1.14 recombinaison [ISO/CEI 7498-1]

3.1.15 réinitialisation [ISO/CEI 7498-1]

3.1.16 routage [ISO/CEI 7498-1]

3.1.17 segmentation [ISO/CEI 7498-1]

3.1.18 séquencement [ISO/CEI 7498-1]

3.1.19 éclatement [ISO/CEI 7498-1]

3.1.20 gestion-systèmes [ISO/CEI 7498-1]

3.2 Termes et définitions de convention de service

La présente norme utilise également les termes définis dans l'ISO/CEI 10731 tels qu'ils s'appliquent à la couche de liaison de données:

3.2.1 service asymétrique

3.2.2 confirmation (primitive);
requestor.deliver (primitive)

3.2.3 deliver (primitive)

3.2.4 primitive de service DL;
primitive

3.2.5 fournisseur de service DL

3.2.6 utilisateur de service DL

3.2.7 indication (primitive)
acceptor.deliver (primitive)

3.2.8 request (primitive);
requestor.submit (primitive)

3.2.9 demandeur

3.2.10 response (primitive);
acceptor.submit (primitive)

3.2.11 submit (primitive)

3.2.12 service symétrique

3.3 Termes et définitions communs

NOTE Plusieurs définitions sont communes à plusieurs types de protocole; elles ne sont pas nécessairement utilisées par tous les types de protocoles.

Pour les besoins du présent document, les définitions suivantes s'appliquent également:

3.3.1
trame
synonyme déconseillé de DLPDU

3.3.2
adresse DL de groupe
adresse DL qui désigne potentiellement plusieurs DLSAP dans la liaison étendue

Note 1 à l'article: Une entité DL unique peut comporter plusieurs adresses DL de groupe associées à un seul DLSAP.

Note 2 à l'article: Une entité DL unique peut aussi avoir une seule adresse DL de groupe associée à plusieurs DLSAP.

3.3.3
nœud
entité DL unique telle qu'elle se présente sur une liaison locale

3.3.4
utilisateur DLS destinataire
utilisateur du service DL auquel sont destinées les données utilisateur DLS

Note 1 à l'article: Un utilisateur de service DL peut être à la fois un utilisateur DLS expéditeur et destinataire.

3.3.5
utilisateur DLS expéditeur
utilisateur du service DL à la source des données utilisateur DLS

3.4 Définitions Type 12 supplémentaires

3.4.1
application
fonction ou structure de données pour laquelle des données sont consommées ou produites

[SOURCE: CEI 61158-5-12, 3.3.1]

3.4.2
objets d'application
classes contenant plusieurs objets permettant de gérer et d'assurer l'échange dynamique des messages sur le réseau et au sein de l'appareil de réseau

3.4.3
esclave de base
appareil esclave ne prenant en charge que l'adressage physique des données

3.4.4
bit
unité d'informations composée de 1 ou de 0

Note 1 à l'article: Il s'agit de la plus petite unité de données pouvant être émise.

3.4.5
client
1) objet qui utilise les services d'un autre objet (serveur) pour exécuter une tâche

2) émetteur d'un message auquel un serveur réagit

3.4.6
connexion
liaison logique entre deux objets d'application d'appareils identiques ou différents

3.4.7
cyclique
événement qui se répète de manière régulière et répétitive

3.4.8
contrôle de redondance cyclique
CRC
valeur résiduelle calculée à partir d'une matrice de données et utilisée comme signature représentative de la matrice

Note 1 à l'article: L'abréviation «CRC» est dérivée du terme anglais développé correspondant «Cyclic Redundancy Check».

3.4.9
données
terme générique utilisé pour faire référence à des informations acheminées sur un bus de terrain

3.4.10
cohérence des données
moyen cohérent de transmission et d'accès de l'objet de données d'entrée ou de sortie entre un client et un serveur et à l'intérieur de ceux-ci

3.4.11
appareil
entité physique connectée au bus de terrain, composée d'au moins un élément de communication (élément de réseau) et qui peut comporter un élément de commande et/ou un élément final (transducteur, actionneur, etc.)

[SOURCE: CEI 61158-2, 3.1.13]

3.4.12
horloges distribuées
méthode de synchronisation d'esclaves et de maintien d'une base de temps globale

3.4.13
erreur
écart ou discordance entre une valeur ou une condition calculée, observée ou mesurée et la valeur ou la condition spécifiée ou théoriquement correcte

3.4.14
événement
instance d'un changement des conditions

3.4.15
unité de gestion de mémoire du bus de terrain FMMU
fonction permettant d'établir une ou plusieurs correspondances entre des adresses logiques et la mémoire physique

3.4.16
entité FMMU
élément unique de l'unité de gestion de mémoire du bus de terrain: correspondance entre un espace d'adresse logique cohérent et un emplacement de mémoire physique cohérent

3.4.17
esclave complet
appareil esclave prenant en charge l'adressage physique et logique des données

3.4.18
interface
frontière commune entre deux unités fonctionnelles, définie par des caractéristiques fonctionnelles, des caractéristiques de signal ou d'autres caractéristiques appropriées

3.4.19
maître
appareil qui contrôle le transfert de données sur le réseau et lance l'accès au support des esclaves en envoyant des messages et qui fait office d'interface avec le système de commande

3.4.20
mapping
correspondance entre deux objets de sorte que l'un d'eux fasse partie intégrante de l'autre

3.4.21
support
câble, fibre optique ou autres moyens de transmission des signaux de communication entre deux ou plusieurs points

Note 1 à l'article: support en anglais est "medium" dont le pluriel est "media".

3.4.22
message
série ordonnée d'octets destinée à communiquer des informations

Note 1 à l'article: En principe utilisé pour acheminer des informations entre des homologues au niveau de la couche d'application.

3.4.23
réseau
ensemble de nœuds connectés par un certain type de support de communication, y compris les répéteurs intercalés, les ponts, les routeurs et les passerelles de couche inférieure

3.4.24
nœud
point d'extrémité d'une liaison dans un réseau ou un point auquel deux ou plusieurs liaisons se rencontrent

[SOURCE: CEI 61158-2, 3.1.31, avec ajustement rédactionnel]

3.4.25
objet
représentation abstraite d'un composant particulier d'un appareil

Note 1 à l'article: Un objet peut être

1) une représentation abstraite des capacités d'un appareil. Les objets peuvent être composés de tout ou partie des composants suivants:

   a) données (informations évoluant dans le temps);

   b) configuration (paramètres de comportement);

   c) méthodes (actions qui peuvent être réalisées à l'aide de données et d'une configuration).

2) un ensemble de données (se présentant sous la forme de variables) et de méthodes (procédures) connexes d'utilisation de ces données qui ont clairement défini l'interface et le comportement.

3.4.26
données de processus
objet de données contenant des objets d'application destinés à être transférés de manière cyclique ou acyclique pour les besoins du traitement

3.4.27
serveur
objet qui offre des services à un autre objet (client)

3.4.28
service
opération ou fonction qu'un objet et/ou une classe d'objets exécute à la demande d'un autre objet et/ou une autre classe d'objets

3.4.29
esclave
entité DL accédant au support uniquement après avoir été lancée par l'esclave précédent ou par le maître

3.4.30
gestionnaire de synchronisation
ensemble d'éléments de commande visant à coordonner l'accès aux objets utilisés simultanément

3.4.31
canal de gestionnaire de synchronisation
élément de commande unique visant à coordonner l'accès aux objets utilisés simultanément

3.4.32
commutateur
pont MAC tel que défini dans l'IEEE 802.1D

3.5 Symboles et abréviations communs

NOTE Plusieurs symboles et abréviations sont communs à plusieurs types de protocole; elles ne sont pas nécessairement utilisées par tous les types de protocoles.

DL-	Préfixe désignant la couche liaison de donnée
DLC	DL connection (Connexion DL)
DLCEP	DL connection-end-point (Extrémité de connexion DL)
DLE	DL entity (Entité DL) (instance active locale de la couche de liaison de données)
DLL	DL-Layer (Couche DL)
DLPCI	DL protocol-control-information (Informations de commande de protocole DL)
DLPDU	DL protocol-data-unit (Unité de données de protocole DL)
DLM	DL management (Gestion DL)
DLME	DL-management Entity (Entité de gestion DL) (instance active locale de la gestion DL)
DLMS	DL-management Service (Service de gestion DL)
DLS	DL service (Service DL)
DLSAP	DL-service-access-point (Point d'accès au service DL)
DLSDU	DL-service-data-unit (Unité de données de service DL)
FIFO	First-in First-out (Premier entré, premier sorti) (méthode de mise en file d'attente)
OSI	Open systems interconnection (Interconnexion de systèmes ouverts)
Ph-	Physical layer (Préfixe désignant la couche physique)
PhE	Ph-entity (Entité Ph) (instance active locale de la couche physique)
PhL	Ph-layer (Couche physique)
QoS	Quality of service (Qualité de service)

3.6 Symboles et abréviations supplémentaires de Type 12

AL	Application layer (Couche d'application)
DLSDU	Data-link protocol data unit (Unité de données de protocole de liaison de données)
APRD	Auto increment physical read (Lecture physique à incrément automatique)
APRW	Auto increment physical read write (Lecture/écriture physiques à incrément automatique)
APWR	Auto increment physical write (Écriture physique à incrément automatique)
ARMW	Auto increment physical read multiple write (Écriture multiple/lecture physique à incrément automatique)
BRD	Broadcast read (Lecture de diffusion)
BRW	Broadcast read write (Lecture/écriture de diffusion)
BWR	Broadcast write (Écriture de diffusion)
CAN	Controller area network (Réseau local du contrôleur)
CoE	CAN application protocol over Type 12 services (Protocole d'application CAN sur services de Type 12)
CSMA/CD	Carrier sense multiple access with collision detection (Accès multiple par

	surveillance du signal et détection de collision)
DC	Distributed clocks (Horloges distribuées)
DCSM	DC state machine (diagramme d'états DC)
DHSM	(DL) PDU handler state machine (diagramme d'états du gestionnaire PDU (DL))
Type 12	Prefix for DL services and protocols (Préfixe des services et protocoles DL)
E²PROM	Electrically erasable programmable read only memory (Mémoire morte programmable effaçable électriquement)
EoE	Ethernet tunneled over Type 12 services (Tunnel Ethernet sur les services de Type 12)
ESC	Type 12 slave controller (Contrôleur d'esclave de Type 12)
FCS	Frame check sequence (Séquence de contrôle de trame)
FMMU	Fieldbus memory management unit (Unité de gestion de mémoire du bus de terrain)
FoE	File access with Type 12 services (Accès au fichier avec services de Type 12)
FPRD	Configured address physical read (Lecture physique de l'adresse configurée)
FPRW	Configured address physical read write (Écriture/lecture physique de l'adresse configurée)
FPWR	Configured address physical write (Écriture physique de l'adresse configurée)
FRMW	Configured address physical read multiple write (Écriture multiple/lecture physique de l'adresse configurée)
HDR	Header (En-tête)
ID	Identifier (Identificateur)
IP	Internet protocol (Protocole Internet)
LAN	Local area network (Réseau local)
LRD	Logical memory read (Lecture de mémoire logique)
LRW	Logical memory read write (Écriture/lecture de mémoire logique)
LWR	Logical memory write (Écriture de mémoire logique)
MAC	Media access control (Commande d'accès au support)
MDI	Media dependent interface (Interface dépendante du support (spécifiée dans l'ISO/CEI 8802-3))
MDX	Mailbox data exchange (Échange de données de boîte aux lettres)
MII	Media independent interface (Interface indépendante du support (spécifiée dans l'ISO/CEI 8802-3))
PDI	Physical device interface (Interface d'appareil physique (ensemble d'éléments permettant d'accéder aux services DL provenant de l'utilisateur DLS))
PDO	Process data object (Objet de données de processus)
PHY	Physical layer device (Appareil de couche physique (spécifié dans l'ISO/CEI 8802-3))
PNV	Publish network variable (Variable de réseau publique)
RAM	Random access memory (Mémoire vive)
RMSM	Resilient mailbox state machine (Diagramme d'états de boîte aux lettres résilient)
Rx	Receive (Recevoir)
SDO	Service data object (Objet de données de service)
SII	Slave information interface (Interface d'informations de l'esclave)
SIISM	SII state machine (diagramme d'états SII)
SyncM	Synchronization manager (Gestionnaire de synchronisation)
SYSM	Sync manager state machine (Diagramme d'états de gestionnaire de synchronisation)

TCP	Transmission control protocol (Protocole de contrôle de transmission)
Tx	Transmit (Émettre)
UDP	User datagram protocol (Protocole de datagramme utilisateur)
WKC	Working counter (Compteur en fonction)

3.7 Conventions

3.7.1 Concept général

Les services sont spécifiés dans la CEI 61158-3-12. La spécification de services définit les services qui sont fournis par la DL de Type 12. Le mapping de ces services avec l'ISO/CEI 8802-3 est décrit dans la présente norme internationale.

La présente norme utilise les conventions de description données dans l'ISO/CEI 10731.

3.7.1.1 Conventions de syntaxe abstraite

Les éléments de la syntaxe DL associés à la structure de la PDU sont décrits comme indiqué dans l'exemple du Tableau 1.

La colonne « Partie de trame» indique l'élément qui sera remplacé par cette reproduction.

La colonne «Champ de données» est le nom de l'élément.

La colonne «Type de données» indique le type de symbole terminal.

La colonne «Valeur/Description» contient la valeur constante ou la signification du paramètre.

Tableau 1 – Exemple de description d'élément PDU

Partie de trame	Champ de données	Type de données	Valeur/Description
Type 12 xxx	CMD	Unsigned8	0x01
	IDX	Unsigned8	Indice
	ADP	Unsigned16	Adresse à incrément automatique
	ADO	Unsigned16	Adresse de mémoire physique
	LEN	Unsigned11	Longueur des données de YYY, en octets
	Reserved (Réservé)	Unsigned4	0x00
	NEXT	Unsigned1	0x00: dernière PDU de Type 12 0x01: PDU de Type 12 suivante
	IRQ	Unsigned16	Réservé à un usage ultérieur
	YYY		Élément suivant
	WKC	Unsigned16	Compteur en fonction

Les types d'attribut sont décrits en langage C (ISO/CEI 9899) (voir Figure 1). BYTE et WORD sont des éléments de type unsigned char (caractère non signé) et unsigned short (court non signé).

```
typedef struct
{
  Unsigned8     Type;
  Unsigned8     Revision;
  Unsigned16    Build;
  Unsigned8     NoOfSuppFmmuChannels;
  Unsigned8     NoOfSuppSyncManChannels;
  Unsigned8     RamSize;
  Unsigned8     Reserved1;
  unsigned      FmmuBitOperationNotSupp:  1;
  unsigned      Reserved2:                7;
  unsigned      Reserved3:                8;
} TDLINFORMATION;
```

Figure 1 – Exemple de description de type

Les attributs eux-mêmes sont décrits sous la forme présentée dans le Tableau 2.

Le paramètre décrit un seul élément de l'attribut.

L'adresse physique indique l'emplacement dans l'espace d'adresses physiques.

L'attribut «Type de données» désigne le type de cet élément.

Le type d'accès DL/PDI Type 12 indique les droits d'accès à cet élément. R et W signifient respectivement «droits d'accès en lecture» et «droits d'accès en écriture». Si DL et PDI de Type 12 ne détiennent aucun droit d'accès en écriture, cette variable sera initialisée et maintenue par DL.

La colonne « Valeur/Description » contient la valeur constante et/ou la signification du paramètre.

Tableau 2 – Exemple de description d'attribut

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/Description
State (état)	0x0120	Unsigned4	RW	R	0x01: Demande Init 0x02: Demande pré-opérationnelle 0x03: Demande de mode d'amorçage 0x04: Demande opérationnelle de sécurité 0x08: Demande opérationnelle
Acknowledge (Acquittement)	0x0120	Unsigned1	RW	R	0x00: pas d'acquittement 0x01 acquittement (doit être un front positif)
Reserved (réservé)	0x0120	Unsigned3	RW	R	0x00
Application Specific (spécifique à l'application)	0x0121	Unsigned8	RW	R	

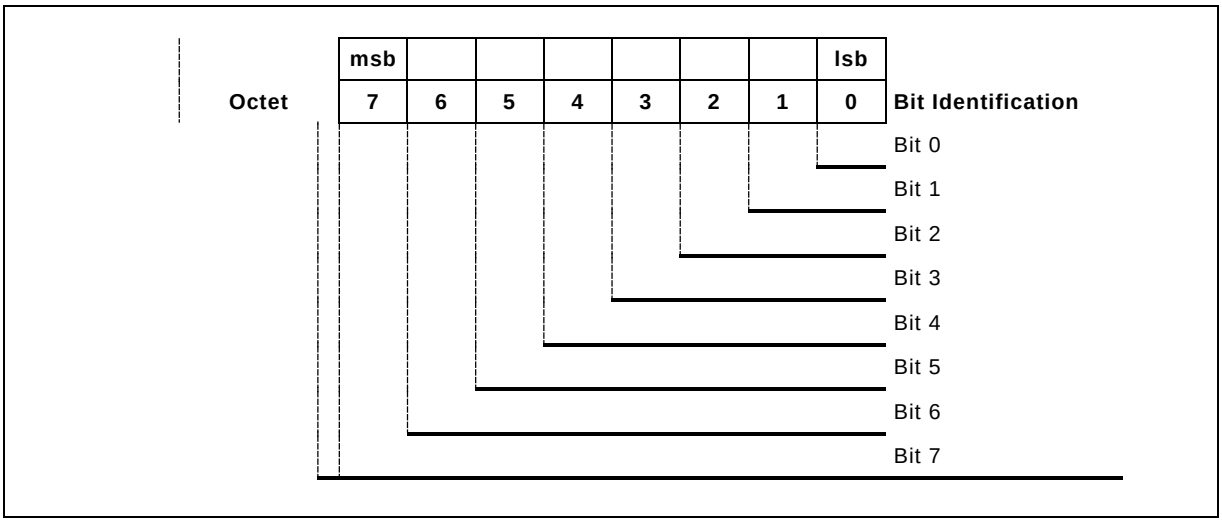
3.7.1.2 Convention de codage des bits et octets réservés

Le terme « reserved» (réservé) peut être utilisé pour décrire les bits en octets ou ensemble d'octets. Il convient d'attribuer la valeur zéro à tous les bits ou octets qui sont réservés côté émetteur. Ils ne doivent pas être soumis à essai côté destinataire, sauf si cela a été explicitement énoncé ou si les bits ou octets réservés sont contrôlés par un diagramme d'états.

Le terme « reserved» (réservé) peut également être utilisé pour indiquer que certaines valeurs dans la plage d'un paramètre sont réservées à des extensions ultérieures. Dans ce cas, il convient de ne pas utiliser les valeurs réservées côté émetteur. Elles ne doivent pas non plus être soumises à essai côté destinataire, sauf si cela a été explicitement énoncé ou si les valeurs réservées sont contrôlées par un diagramme d'états.

3.7.1.3 Conventions de codage commun d'octets de champs particuliers

Les DLSDU peuvent comporter des champs particuliers acheminant des informations de manière primitive et condensée. Ces champs doivent être codés dans l'ordre conformément à la Figure 2.



Légende

Anglais	Français
msb	Bit de poids fort
lsb	Bit de poids faible
Octet	Octet
Bit Identification	Identification de bit

Figure 2 – Structure commune des champs spécifiques

Les bits peuvent être rassemblés en groupe de bits. Chaque bit ou groupe de bits doit être adressé par son Identification de bit (Bit 0, Bits 1 à 4, par exemple). La position à l'intérieur de l'octet doit être conforme à la figure ci-dessus. Les noms d'alias peuvent être utilisés pour chaque bit ou groupe de bits ou être marqués comme étant réservés. Le regroupement de bits individuels doit être réalisé dans l'ordre croissant, sans espace. Les valeurs d'un groupe de bits peuvent être représentées sous forme binaire, décimale ou hexadécimale. Cette valeur doit uniquement être valide pour les bits groupés et ne peut représenter que l'ensemble de l'octet si les 8 bits sont regroupés. Les valeurs décimales ou hexadécimales doivent être transférées en valeurs binaires, de sorte que le bit portant le numéro le plus élevé du groupe représente le bit de poids fort (msb) des bits regroupés.

EXEMPLE Description et relation pour l'octet de champ particulier

Bit 0: Reserved (réservé)

Bit 1-3: Reason_Code La valeur décimale 2 de Reason_Code indique une erreur générale.

Bit 4-7: doit être toujours mis à un.

L'octet construit conformément à la description ci-dessus se présente comme suit:

(bit de poids fort) Bit 7 = 1,

Bit 6 = 1,

Bit 5 = 1,

Bit 4 = 1,

Bit 3 = 0,

Bit 2 = 1,

Bit 1 = 0,

(bit de poids faible) Bit 0 = 0.

Cette combinaison de bits comporte une représentation de valeur d'octet 0xf4.

3.7.2 Conventions du diagramme d'états

Les séquences de protocole sont décrites au moyen de diagrammes d'états.

Dans les diagrammes d'états, les états sont représentés par des cases et les transitions d'état par des flèches. Les noms d'états et de transitions du diagramme d'état correspondent aux noms indiqués dans la liste des transitions d'état.

La liste des transitions d'état est structurée comme suit, voir également le Tableau 3.

– La première colonne contient le nom de la transition.
– La deuxième colonne définit l'état en cours.
– La troisième colonne contient un événement facultatif suivi par les conditions commençant par une barre oblique (/) et se terminant par les actions commençant par =>.
– La dernière colonne contient l'état suivant.

Si l'événement se produit et les conditions sont remplies, la transition est lancée, c'est-à-dire que les actions sont exécutées et que le passage à l'état suivant a lieu.

Le Tableau 3 présente une description d'un diagramme d'états. La signification des éléments de la description d'un diagramme d'états est présentée dans le Tableau 4.

Tableau 3 – Éléments de description d'un diagramme d'états

#	État en cours	Événement /Condition => Action	État suivant

Tableau 4 – Description des éléments d'un diagramme d'états

Élément de description	Signification
État en cours État suivant	Nom des états donnés.
#	Nom ou numéro de la transition d'état.
Événement	Nom ou description de l'événement.
/Condition	Expression booléenne. Le signe \ qui précède ne fait pas partie de la condition.
=> Action	Liste des affectations et des appels de service ou de fonction. Le signe => qui précède ne fait pas partie de l'action.

Les conventions utilisées dans les diagrammes d'états sont présentées dans le Tableau 5.

Tableau 5 – Conventions utilisées dans les diagrammes d'états

Convention	Signification
=	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.
axx	Nom de paramètre si a est une lettre. EXEMPLE Identifier = reason la valeur moyenne d'un paramètre reason est attribuée à un paramètre appelé 'Identifier'.
"xxx"	Indique une chaîne visible fixe. EXEMPLE Identifier = "abc" la valeur moyenne abc est attribuée à un paramètre appelé 'Identifier'.
nnn	Si tous les éléments sont des chiffres, l'entité représente une constante numérique affichée en représentation décimale
0xnn	Si tous les éléments nn sont des chiffres, l'entité représente une constante numérique affichée en représentation hexadécimale
==	Condition logique indiquant qu'une entité de gauche est égale à une entité de droite.
<	Condition logique indiquant qu'une entité de gauche est inférieure à une entité de droite.
>	Condition logique indiquant qu'une entité de gauche est supérieure à une entité de droite.
!=	Condition logique indiquant qu'une entité de gauche n'est pas égale à une entité de droite.
&&	AND (ET) Logique
||	OR (OU) Logique
!	NOT (NON) Logique
+ - * /	Opérateurs arithmétiques
;	Séparateur d'expressions

Il est vivement recommandé aux lecteurs de se reporter aux paragraphes relatifs aux définitions d'attribut, aux fonctions locales et aux définitions de FDL-PDU pour comprendre les diagrammes d'états. Les lecteurs sont censés avoir une connaissance suffisante de ces définitions pour les utiliser sans explications complémentaires.

D'autres constructions telles que définies en langage C (ISO/CEI 9899) peuvent être utilisées pour décrire les conditions et les actions.

4 Vue d'ensemble du protocole DL

4.1 Principe de fonctionnement

La DL de Type 12 est une technologie Ethernet en temps réel visant à optimiser l'utilisation de la largeur de bande Ethernet bidirectionnelle simultanée. La commande d'accès au support utilise le principe Maître/Esclave, dans lequel le nœud maître (en général le système de commande) envoie les trames Ethernet aux nœuds esclaves, lesquels extraient les données de ces trames et les y insèrent.

Du point de vue Ethernet, un segment Type 12 est un appareil Ethernet unique, qui reçoit et envoie des trames Ethernet conformes à l'ISO/CEI 8802-3. Toutefois, cet appareil Ethernet ne se limite pas à un seul contrôleur Ethernet doté d'un microprocesseur aval, mais peut être composé d'un grand nombre d'appareils esclaves Type 12. Cela permet de traiter les trames entrantes directement et d'extraire les données utilisateur pertinentes ou d'insérer les données et de transférer la trame vers l'appareil esclave suivant. Le dernier appareil esclave Type 12 du segment renvoie la trame totalement traitée, de manière à la renvoyer par le premier appareil esclave au maître sous la forme d'une trame de réponse.

Cette procédure permet d'utiliser le mode bidirectionnel simultané d'Ethernet: les deux directions de communication fonctionnent de manière indépendante. La communication directe sans commutateur entre un appareil maître et un segment Type 12 composé d'un ou de plusieurs appareils esclaves peut être établie.

4.2 Topologie

La topologie d'un système de communication est l'un des facteurs fondamentaux de la réussite d'une application en matière d'automatisation. La topologie a une influence significative sur l'effort de câblage, sur les fonctions de diagnostic, sur les options de redondance et sur les fonctions plug-and-play à chaud.

La topologie en étoile communément utilisée pour Ethernet implique un effort de câblage et des coûts d'infrastructures plus importants. Il est préférable d'utiliser une topologie linéaire ou arborescente, plus particulièrement pour les applications d'automatisation.

La disposition du nœud esclave représente un bus en anneau ouvert. A l'extrémité ouverte, l'appareil maître envoie des trames, directement ou par l'intermédiaire des commutateurs Ethernet, et les reçoit à l'autre extrémité après leur traitement. Toutes les trames sont relayées du premier nœud vers les nœuds qui suivent. Le dernier nœud renvoie la PDU vers le maître. Grâce aux capacités bidirectionnelles simultanées d'Ethernet, la topologie obtenue est une ligne physique.

Les branches, qui peuvent être en principe présentes dans toute la topologie, peuvent permettre d'améliorer la structure linéaire d'une structure arborescente. Une structure arborescente prend en charge un câblage très simple; Les branches individuelles, par exemple, peuvent être raccordées à des armoires de commande ou des modules de machine, la ligne principale passant d'un module à l'autre.

4.3 Principes de traitement de trame

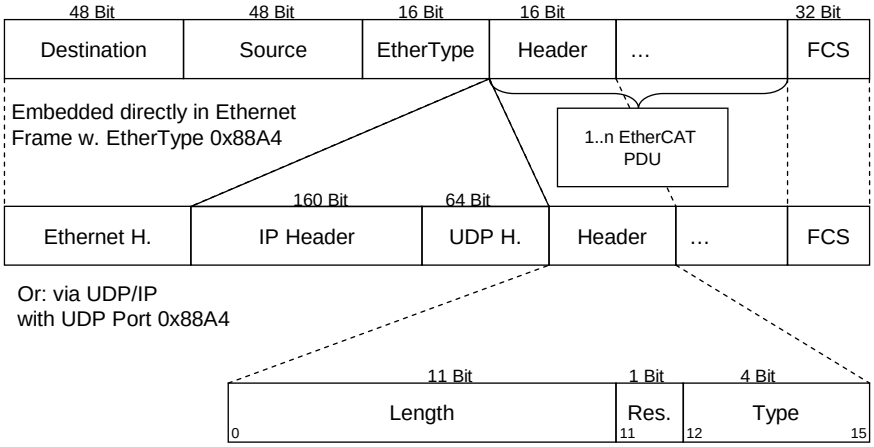
Pour obtenir des performances optimales, il convient de traiter les trames Ethernet directement « à la volée ». Si le nœud esclave est mis en œuvre de cette manière, il reconnaît les commandes correspondantes et les exécute en conséquence lors du passage des trames.

NOTE 1 La liaison de données Type 12 peut être mise en œuvre à l'aide de contrôleurs Ethernet normalisés sans traitement direct. L'influence de la mise en œuvre du mécanisme de transfert sur les performances de communication est présentée en détail dans les parties relatives au profil.

Les nœuds disposent d'une mémoire adressable accessible grâce aux services de lecture ou d'écriture, un nœud à la fois ou plusieurs nœuds simultanément. Plusieurs PDU Type 12 peuvent être intégrées à une trame Ethernet, chacune d'elle adressant une section de données de cohésion. Comme le montre la Figure 3, les PDU Type 12 sont soit transportées:

a) directement dans la zone de données de la trame Ethernet,

b) dans la section de données d'un datagramme UDP transporté via IP.



Légende

Anglais	Français
Embedded directly in Ethernet Frame w. EtherType 0x88A4	Intégré directement dans la trame Ethernet avec EtherType 0x88A4
IP Header	En-tête IP
Length	Longueur
Or: via UDP/IP with UDP Port 0x88A4	Ou: via UDP/IP avec port UDP 0x88A4

Figure 3 – Structure d'une trame

La variante a) se limite à un sous-réseau Ethernet, les trames associées n'étant pas relayées par les routeurs. En général, pour les applications de contrôle de machine, cela n'est pas une contrainte. Plusieurs segments Type 12 peuvent être connectés à un ou plusieurs commutateurs. L'adresse MAC Ethernet du premier nœud du segment est utilisée pour adresser le segment Type 12.

NOTE 2 De plus amples détails relatifs à l'adressage sont donnés dans la définition du service de couche de liaison de données (voir CEI 61158-3-12).

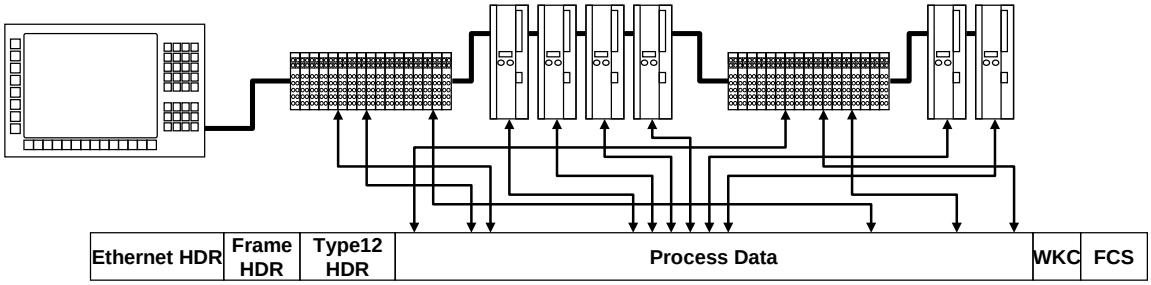
La variante b) via UDP/IP génère un surdébit sensiblement plus important (en-tête IP et UDP), mais pour les applications à moindre contrainte de temps (l'automatisation des bâtiments, par exemple), elle permet d'utiliser le routage IP. Du côté maître, une mise en œuvre UDP/IP normalisée peut être utilisée.

4.4 Vue d'ensemble de la couche de liaison de données

Plusieurs nœuds peuvent être adressés individuellement par l'intermédiaire d'une seule trame Ethernet transportant plusieurs PDU Type 12. Les PDU Type 12 sont compactées sans espaces. La trame se termine par la dernière PDU Type 12, sauf si la taille de trame est inférieure à 64 octets, auquel cas la trame sera remplie pour atteindre une longueur de 64 octets.

Par rapport à une trame par nœud, cela permet une meilleure utilisation de la largeur de bande Ethernet. Toutefois, pour un nœud d'entrée numérique à 2 canaux avec juste 2 bits de données utilisateur, par exemple, le surdébit d'une seule PDU Type 12 PDU reste excessif.

Par conséquent, les nœuds esclaves peuvent également prendre en charge le mapping d'adresse logique. Les données de processus peuvent être insérées n'importe où dans l'espace d'adresse logique. Si une PDU Type 12 est envoyée et qu'elle contient des services de lecture ou d'écriture pour une certaine zone d'image de processus à l'adresse logique correspondante, au lieu de l'adressage d'un nœud particulier, les nœuds insèrent les données dans le bon emplacement dans le processus de données ou les en extrait, comme indiqué à la Figure 4.



Légende

Anglais	Français
Ethernet HDR	HDR Ethernet
Frame HDR	Trame HDR
Process Data	Données de processus

Figure 4 – Mapping des données dans une trame

Tous les autres nœuds qui détectent également une adresse correspondant à l'image de processus insèrent également leurs données, de sorte que plusieurs nœuds peuvent être adressés simultanément avec une seule PDU Type 12. Le maître peut assembler les images de processus logique complètement triées par l'intermédiaire d'une seule PDU Type 12. Un mapping supplémentaire n'est plus requis dans le maître, de sorte que les données de processus peuvent être directement attribuées aux différentes tâches de commande. Chaque tâche peut créer sa propre image de processus et l'échanger dans son propre calendrier. L'ordre physique des nœuds est totalement arbitraire et uniquement pertinent lors de la première phase d'initialisation.

L'espace d'adresse logique est de 2^{32} octets = 4 Go. La liaison de données Type 12 peut être considérée comme un fond de panier série destiné aux systèmes d'automatisation, assurant la connexion aux données de processus distribuées pour de très petits et de volumineux appareils d'automatisation. Grâce à un contrôleur Ethernet et un câble Ethernet normalisés, un très grand nombre de canaux d'entrée/sortie ne faisant l'objet d'aucune restriction pratique sur la distribution peut être connecté aux appareils d'automatisation, accessibles avec une largeur de bande élevée, un délai minimal et une vitesse de transmission des données proche de l'optimale. Dans le même temps, des appareils tels que des scanneurs de bus de terrain peuvent être également connectés, permettant de préserver les technologies et normes existantes.

4.5 Vue d'ensemble de la détection des erreurs

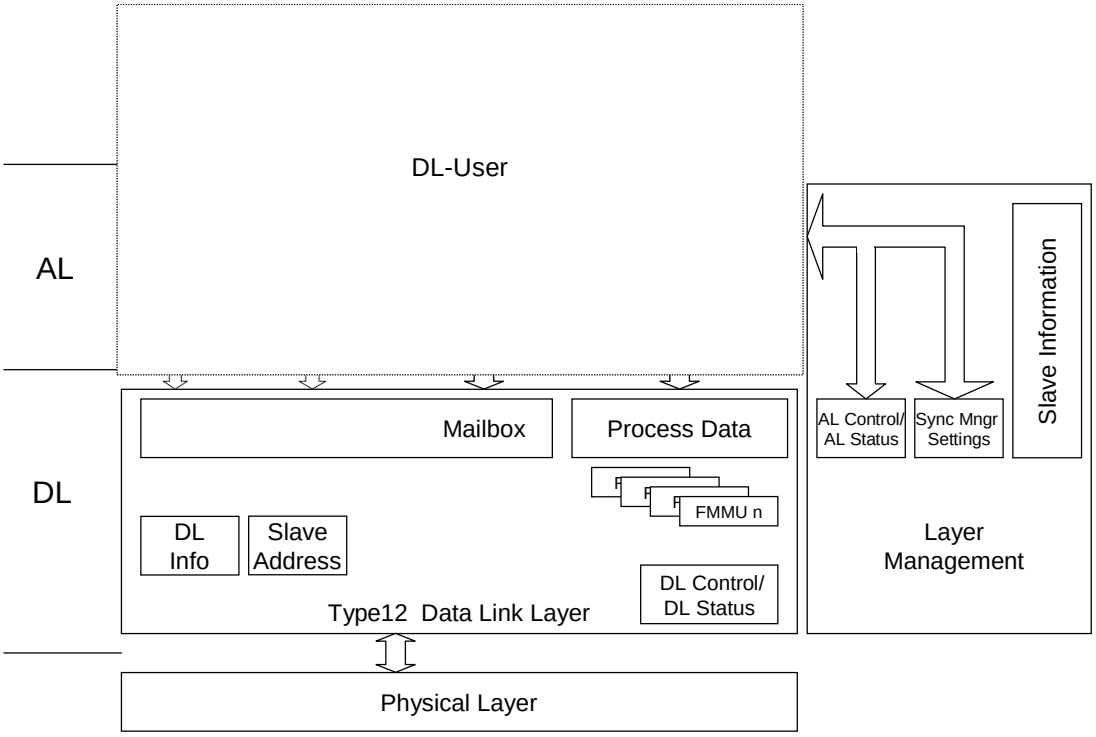
La liaison de données Type 12 s'appuie sur la séquence de contrôle de trame Ethernet (FCS) pour vérifier si une trame a été transmise correctement. Étant donné qu'un ou plusieurs esclaves modifient la trame pendant le transfert, la FCS est recalculée par chaque esclave. Si un esclave détecte une erreur de somme de contrôle, il ne répare pas la FCS mais balise le maître en incrémentant le compteur d'erreurs, de manière à pouvoir détecter précisément une anomalie.

Lors de l'écriture des données dans une PDU Type 12 ou de leur lecture, l'esclave adressé incrémente un compteur en fonction (WKC) placé à la fin de chaque PDU Type 12. L'analyse du compteur en fonction permet au maître de vérifier si le nombre prévu de nœuds a traité la PDU Type 12 correspondante.

4.6 Modèle de référence de nœud

4.6.1 Mapping du modèle de référence de base OSI

La liaison de données Type 12 est décrite à l'aide des principes, de la méthodologie et du modèle de l'ISO/CEI 7498 Systèmes de traitement de l'information — Interconnexion des systèmes ouverts — Modèle de référence de base (OSI). Le modèle OSI fournit une approche en couches des normes de communication, selon laquelle il est possible de modifier et développer les couches de manière indépendante. La spécification de la DL Type 12 définit la fonctionnalité de haut en bas d'une pile OSI complète, ainsi que certaines fonctions destinées aux utilisateurs de la pile. Les fonctions des couches OSI intermédiaires, les couches 3 à 6, sont consolidées dans la couche de liaison de données DL Type 12 ou dans la couche d'application DL Type 12. De même, les fonctions communes aux utilisateurs de la couche d'application de bus de terrain peuvent être fournies par la couche d'application DL Type 12 afin de simplifier l'utilisation, comme indiqué à la Figure 5.



Légende

Anglais	Français
Dl-User	Utilisateur de DL
Mailbox	Boîte aux lettres (Boîte d'échange)
Process data	Données de processus
Sync Mngr settings	Paramètres du gestionnaire de sync
Layer Management	Gestion de couche
Slave Information	Informations de l'esclave
Type 12 Data Link Layer	Couche de liaison de données Type 12
AL Control	Commande AL
AL Status	État AL
DL Control	Commande DL
DL Status	État DL
DL Info	Info DL
Slave Address	Adresse de l'esclave

Anglais	Français
Physical Layer	Couche physique

Figure 5 – Modèle de référence du nœud esclave

4.6.2 Caractéristiques de la couche liaison de données

La couche de liaison de données fournit le support de communication de données à contrainte de temps de base parmi les appareils connectés via la DL Type 12. Le terme «à contrainte de temps» est utilisé pour décrire les applications à fenêtre temporelle, dans laquelle une ou plusieurs actions spécifiées ont besoin d'être réalisées selon un certain niveau de certitude. Si les actions spécifiées ne sont pas réalisées dans la fenêtre temporelle, les applications demandant les actions risquent de connaître une défaillance, avec les risques que cela comporte pour les équipements, les installations et éventuellement la vie humaine.

La couche de liaison de données est chargée de calculer, de comparer et de générer la séquence de contrôle de trame, et d'assurer la communication en extrayant les données de la trame Ethernet et/ou en les y incluant. Cela est fonction des paramètres de la couche de liaison de données stockés dans des emplacements de mémoire préalablement définis. Les données d'application sont mises à disposition de la couche d'application dans la mémoire physique, soit dans une configuration de boîte aux lettres soit dans la section de données de processus.

4.7 Vue d'ensemble du fonctionnement

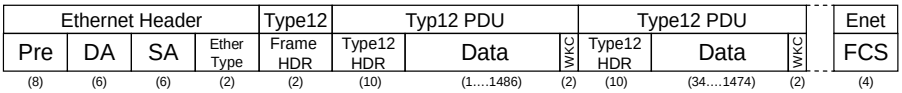
4.7.1 Relation avec l'ISO/CEI 8802-3

La présente partie spécifie les services de couche de liaison de données, outre ceux spécifiés dans l'ISO/IEC 8802-3.

4.7.2 Structure de trame

Une trame Ethernet Type 12 contient une ou plusieurs PDU Type 12 (comme montré à la Figure 6), chacune adressant des appareils et/ou des zones de mémoires individuels. La trame Type 12 est reconnue par la combinaison d'EtherType 0x88A4[1] et de l'en-tête de trame Type 12 correspondant ou, si elle est transportée via UDP/IP conformément à l'IETF RFC 791/IETF RFC 768 (comme montré à la Figure 7) par le port UDP de destination 34980=0x88A4[2] et l'en-tête de trame Type 12.La fragmentation des paquets IP est ignorée. Les esclaves peuvent attribuer la valeur 0 à la somme de contrôle UDP, qui peut être ignorée. Le type de service, la somme de contrôle de l'en-tête IP, la longueur de paquet IP et la longueur UDP ne font l'objet d'aucun contrôle.

Chaque PDU Type 12 est composée d'un en-tête Type 12, de la zone de données et d'une zone de compteur subséquente (compteur en fonction) qui est incrémentée par tous les nœuds adressés par la PDU Type 12 et dont les données associées ont été échangées.



Légende

Anglais	Français
Ethernet Header	En-tête Ethernet
Type 12 PDU	PDU Type 12

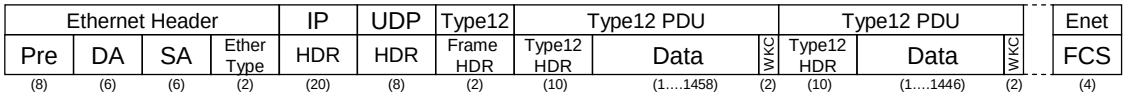
---

1 EtherType 0x88A4 a été attribué pour Type 12 (EtherCAT) par l'autorité d'enregistrement de l'IEEE.

2 Le port UDP 34980 a été attribué pour Type 12 (EtherCAT) par l'IANA (Internet Assigned Numbers Authority).

Anglais	Français
Frame HDR	Trame HDR
Data	Données
Type 12 HDR	HDR Type 12

Figure 6 – PDU Type 12 intégrées dans une trame Ethernet



Légende

Anglais	Français
Ethernet Header	En-tête Ethernet
Type 12 HDR	HDR Type 12
Frame HDR	Trame HDR
Data	Données
Type 12 PDU	PDU Type 12

Figure 7 – PDU Type 12 intégrées dans UDP/IP

5 Structure de trame

5.1 Principes de codage de trame

La DL Type 12 utilise une structure de trame Ethernet conforme à l'ISO/CEI 8802-3 pour le transport des PDU Type 12. Par ailleurs, les PDU peuvent être envoyées via UDP/IP. Les parties du protocole spécifique au Type 12 sont identiques dans les deux cas.

5.2 Types de données et règles de codage

5.2.1 Description générale des types de données et des règles de codage

Pour pouvoir échanger des données significatives, leurs format et signification doivent être connus du producteur et du/des consommateur(s). La présente spécification modélise cela grâce au concept de types de données.

Les règles de codage définissent la représentation des valeurs des types de données et la syntaxe de transfert pour les représentations. Les valeurs sont représentées sous la forme de séquences binaires. Les séquences binaires sont transférées en séquences d'octets. Pour les types de données numériques, le codage est au format petit-boutiste (voir Tableau 6).

Les types de données et les règles de codage doivent être valides pour les services et protocoles DL et AL spécifiés. Les règles de codage de la trame Ethernet sont spécifiées dans l'ISO/CEI 8802-3. La DLSDU d'Ethernet est une chaîne d'octets. L'ordre de transmission à l'intérieur des octets dépend des règles de codage MAC et PhL.

5.2.2 Syntaxe de transfert des séquences binaires

Pour les transmissions sur DL Type 12, une séquence binaire est réorganisée en séquence d'octets. La notation hexadécimale est utilisée pour les octets (voir l'ISO/CEI 9899). Soit la séquence binaire $b = b_0 \ldots b_{n-1}$. Soit k un entier non négatif tel que $8(k - 1) < n \leq 8k$. Alors b est transféré dans k octets assemblés comme indiqué dans le Tableau 6. Les bits b_i, $i \geq n$ de l'octet le plus élevé sont des bits sans importance.

L'octet 1 est transmis le premier, et l'octet *k* le dernier. Par conséquent, la séquence binaire est transférée comme suit sur le réseau (l'ordre de transmission à l'intérieur d'un octet est déterminé par l'ISO/IEC 8802-3):

$b_7, b_6, ..., b_0, b_{15}, ..., b_8, ...$

Tableau 6 – Syntaxe de transfert des séquences binaires

Numéro d'octet	1.	2.	k.
	b_7 .. b_0	b_{15} .. b_8	b_{8k-1} .. b_{8k-8}

EXEMPLE

Bit 9	...	Bit 0
10b	0001b	1100b
0x2	0x1	0xC
		= 0x21C

La séquence de bits b = b_0 .. La séquence binaire b_9 = 0011 1000 01_b représente un Unsigned10 avec la valeur 0x21C et est transférée dans deux octets: 0x1C d'abord, puis 0x02.

5.2.3 Entier non signé

Les données d'un type de données de base Unsignedn contiennent des valeurs dans les entiers non négatifs. La plage de valeurs est 0, ..., 2^n-1. Les données sont représentées sous la forme de séquences binaires de longueur *n*. La séquence binaire

$b = b_0 ... b_{n-1}$

comporte la valeur

$$\text{Unsignedn}(b) = b_{n-1}\times 2^{n-1} + ... + b_1\times 2^1 + b_0\times 2^0$$

La séquence binaire commence à gauche par l'octet de poids faible.

EXEMPLE La valeur 266 = 0x10A avec le type de données Unsigned16 est transférée dans deux octets, 0x0A d'abord, puis 0x01.

Les types de données Unsignedn sont transférés tels que spécifiés dans le Tableau 7. Les types de données non signées Unsigned1 à Unsigned7 et Unsigned 9 à Unsigned15 seront également utilisés. Dans ce cas, l'élément suivant commencera au premier bit libre (voir 3.7.1).

Tableau 7 – Syntaxe de transfert du type de données Unsignedn

Numéro d'octet	1.	2.	3.	4.	5.	6.	7.	8.
Unsigned8	$b_7..b_0$							
Unsigned16	$b_7..b_0$	$b_{15}..b_8$						
Unsigned32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Unsigned64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.2.4 Entier signé

Les données d'un type de données de base Integern contiennent des valeurs entières. La plage de valeurs s'étend de -2^{n-1} à $2^{n-1}-1$. Les données sont représentées sous la forme de séquences binaires de longueur *n*. La séquence binaire

$$b = b_0 \,..\, b_{n-1}$$

comporte la valeur

$$\text{Integern}(b) = b_{n-2}\times 2^{n-2} + \ldots + b_1\times 2^1 + b_0\times 2^0 \text{ if } b_{n-1} = 0$$

et, après deux calculs arithmétiques,

$$\text{Integern}(b) = -\text{ Integern}(\hat{}b) - 1 \text{ if } b_{n-1} = 1$$

NOTE La séquence binaire commence à gauche par le bit de poids faible.

EXEMPLE La valeur –266 = 0xFEF6 avec le type de données Integer16 est transférée dans deux octets, 0xF6 d'abord, puis 0xFE.

Les types de données Integern sont transférés tels que spécifiés dans le Tableau 8. Les types de données Integer (Entier) tels que Integer1 à Integer7 et Integer9 à Integer15 seront utilisés too. Dans ce cas, l'élément suivant commencera au premier bit libre (voir 3.7.1).

Tableau 8 – Syntaxe de transfert du type de données Integern

Numéro d'octet	1.	2.	3.	4.	5.	6.	7.	8.
Integer8	$b_7..b_0$							
Integer16	$b_7..b_0$	$b_{15}..b_8$						
Integer32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Integer64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.2.5 Chaîne d'octets

Le type de données OctetString*length* est défini ci-dessous; *length* est la longueur de la chaîne d'octets.

ARRAY [length] OF Unsigned8 OctetString*length*

5.2.6 Chaîne visible

Le type de données VisibleString*length* est défini ci-dessous. Les valeurs de données admises du type *VISIBLE_CHAR* sont 0_h et la plage est comprise entre 0x20 et 0x7E. Les données sont interprétées en caractères codés 7 bits. *length* est la longueur de la chaîne visible.

Unsigned8 VISIBLE_CHAR

ARRAY [length] OF VISIBLE_CHAR VisibleString*length*

Aucun 0x0 n'est nécessaire pour terminer la chaîne.

5.3 Structure DLPDU

5.3.1 Trame Type 12 à l'intérieur d'une trame Ethernet

La structure de trame est composée des entrées de données suivantes comme spécifié au Tableau 9.

Tableau 9 – Trame Type 12 à l'intérieur d'une trame Ethernet

Partie de trame	Champ de données	Type de données	Valeur/description
Ethernet	Dest MAC	BYTE[6]	Adresse MAC de destination (voir ISO/IEC 8802-3)
	Src MAC	BYTE[6]	Adresse MAC source (voir ISO/IEC 8802-3)
(facultatif)	Étiquette VLAN	BYTE[4]	0x81, 0x00 et TCI (Tag Control Information) à deux octets (voir IEEE 802.1Q)
	Ether Type	BYTE[2]	0x88, 0xA4 (Type 12)
	Trame Type 12		spécifiée en 5.3.3
	Remplissage	BYTE[n]	doit être inséré si la PDU DL comporte moins de 64 octets (voir ISO/CEI 8802-3)
Ethernet FCS	FCS	Unsigned32	Codage de somme de contrôle Ethernet normalisé (voir ISO/CEI 8802-3)

5.3.2 Trame Type 12 à l'intérieur d'un datagramme UDP

La structure de trame est composée des entrées de données suivantes comme spécifié au Tableau 10.

Tableau 10 – Trame Type 12 à l'intérieur d'une PDU UDP

Partie de trame	Champ de données	Type de données	Valeur/description
Ethernet	Dest MAC	BYTE[6]	Voir Tableau 9
	Src MAC	BYTE[6]	Voir Tableau 9
(facultatif)	Étiquette VLAN	BYTE[4]	Voir Tableau 9
	Ether Type	BYTE[2]	0x08, 0x00 (IP)
IP	VersionHL	BYTE	0x45 (IP Version(4) longueur d'en-tête (5*4 octets))
	Service	BYTE	0x00 (Type de service IP)
	TotalLength	Unsigned16	(longueur totale de service IP) - non contrôlée dans le segment Type 12
	Identification	Unsigned16	(paquet d'identification IP du service fragmenté) - non contrôlé dans le segment Type 12
	Flags (Fanions)	BYTE	(fanion IP – ils ne seront pas pris en compte, mais une fragmentation de la trame Type 12 génèrera une erreur) – non contrôlés dans le segment Type 12
	Fragments	BYTE	(numéro de fragment IP - la fragmentation de la trame Type 12 génèrera une erreur) – non contrôlé dans le segment Type 12
	Ttl	BYTE	(durée de vie IP – contrôlée uniquement au niveau des routeurs) - non contrôlée dans le segment Type 12
	Protocole	BYTE	0x11 (sous-protocole IP – cette valeur est réservée pour UDP)
	Somme de contrôle de	Unsigned16	(somme de contrôle de l'en-tête IP) - non contrôlée dans le segment Type 12

Partie de trame	Champ de données	Type de données	Valeur/description
	l'en-tête		
	Adresse IP source	BYTE[4]	(adresse IP source de l'auteur) - non contrôlée dans le segment Type 12
	Adresse IP de destination	BYTE[4]	(adresse IP de destination de la cible de la trame – en général, dans un segment Type 12, une adresse multidiffusion telle qu'une adresse individuelle requiert un protocole ARP (Address Resolution Protocol)) - non contrôlée dans le segment Type 12
UDP	Port Src	WORD	(Port source UDP) - non contrôlé dans le segment Type 12
	Port de destination	WORD	0x88A4 (Port source UDP)
	Longueur	WORD	(longueur UDP de la trame) - non contrôlée dans le segment Type 12
	Somme de contrôle	WORD	(somme de contrôle UDP de la trame) – sera mise à 0 pour les trames Type 12, mais sans contrôle
	Trame Type 12		spécifiée en 5.3.3
	Remplissage	BYTE[n]	doit être inséré si la PDU DL comporte moins de 64 octets (voir ISO/CEI 8802-3)
Ethernet FCS	FCS	Unsigned32	Codage de somme de contrôle Ethernet normalisé (voir ISO/CEI 8802-3)

NOTE 1 La structure de paquet IP et les exigences de codage sont spécifiées dans l'IETF RFC 791.

NOTE 2 L'ordre des octets dans des valeurs à plusieurs octets est codé différemment dans les protocoles IETF (voir IETF RFC 768 et RFC 791) que dans le protocole DL Type 12.

5.3.3 Structure de la trame Type 12

La structure de trame Type 12 doit être composée de l'une des structures spécifiées dans le Tableau 11, Tableau 12 et le Tableau 13.

Tableau 11 – Structure de trame Type 12 contenant des PDU Type 12

Partie de trame	Champ de données	Type de données	Valeur/description
Trame Type 12	Longueur	unsigned11	Longueur de cette trame (moins 2 octets)
	Reserved (réservé)	unsigned 1	0
	Type	unsigned4	Type de protocole = DLPDU Type 12 (0x01)
	PDU 1 Type 12		spécifiée en 5.4
	…		spécifiée en 5.4
	PDU n Type 12		spécifiée en 5.4

Tableau 12 – Structure de trame Type 12 contenant des variables de réseau

Partie de trame	Champ de données	Type de données	Valeur/description
Trame Type 12	Longueur	unsigned11	Longueur de cette trame (moins 2 octets)
	Reserved (réservé)	unsigned 1	0
	Type	unsigned4	Type de protocole = variables de réseau (0x04)
En-tête de l'éditeur	PubID	BYTE[6]	ID de l'éditeur
	CntNV	Unsigned16	Nombre de variables de réseau contenues dans cette trame Type 12
	CYC	Unsigned16	Numéro de cycle côté éditeur
	Reserved (réservé)	BYTE[2]	0x00, 0x00
	Variable de réseau 1		Spécifiée en 5.5
	…		Spécifiée en 5.5
	Variable de réseau n		Spécifiée en 5.5

Tableau 13 – Structure de trame Type 12 contenant une boîte aux lettres

Partie de trame	Champ de données	Type de données	Valeur/description
Trame Type 12	Longueur	unsigned11	Longueur de cette trame (moins 2 octets)
	Reserved (réservé)	unsigned 1	0
	Type	unsigned4	Type de protocole = boîte aux lettres (0x05)
	Boîte aux lettres		Spécifiée en 5.6

5.4 Structure DLPDU de type 12

5.4.1 Lecture

5.4.1.1 Vue d'ensemble

Grâce aux services de lecture, un maître lit les données dans la mémoire d'un ou de plusieurs esclaves. Le compteur en fonction doit être incrémenté par chaque esclave si au moins l'un des attributs adressés est présent.

5.4.1.2 Lecture physique à incrément automatique (APRD)

Le codage APRD (lecture physique à incrément automatique) est spécifié dans le Tableau 14. Chaque esclave incrémente l'ADP d'adresse. L'esclave qui reçoit une adresse à incrément automatique portant une valeur nulle exécute l'opération de lecture demandée.

Tableau 14 – Lecture physique à incrément automatique (APRD)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x01 (commande APRD)
	IDX	Unsigned8	Indice
	ADP	WORD	Adresse à incrément automatique
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître et ne doit pas être modifié par l'esclave.

ADP

Chaque esclave doit incrémenter ce paramètre, et l'esclave qui le reçoit avec une valeur nulle doit exécuter l'accès en lecture.

NOTE Cela signifie que le paramètre contient la position négative de l'esclave dans la boucle logique commençant par 0 côté maître (par exemple, –7 signifie que sept esclaves se trouvent entre le maître et l'esclave adressé). À la confirmation, ce paramètre contient la valeur de la demande incrémentée du nombre d'appareils esclaves transités.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à lire sont enregistrées

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données de lecture si l'accès est valide au niveau des esclaves adressés. Sinon, la valeur envoyée avec la demande ne change pas.

WKC

Ce paramètre doit être incrémenté de un si la lecture des données a abouti.

5.4.1.3 Lecture physique de l'adresse configurée (FPRD)

Le codage FPRD (lecture physique de l'adresse configurée) est spécifié dans le Tableau 15.

Tableau 15 – Lecture physique de l'adresse configurée (FPRD)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x04 (commande FPRD)
	IDX	Unsigned8	Indice
	ADP	WORD	Adresse de station configurée ou alias de station configuré
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

L'esclave dont la valeur de D_address fait office d'adresse de station ou d'alias de l'adresse de station doit exécuter une action de lecture.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à lire sont enregistrées.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données de lecture si l'accès est valide au niveau des esclaves adressés. Sinon, la valeur envoyée avec la demande ne change pas.

WKC

Ce paramètre doit être incrémenté de un si la lecture des données a abouti.

5.4.1.4 Lecture de diffusion (BRD)

Le codage BRD (lecture de diffusion) est spécifié dans le Tableau 16.

Tableau 16 – Lecture de diffusion (BRD)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x07 (commande BRD)
	IDX	Unsigned8	Indice
	ADP	WORD	Paramètre incrémenté de 1 au niveau de chaque station transférant la PDU BRD
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

Ce paramètre doit être incrémenté de un au niveau de chaque esclave.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique dans laquelle les données à lire sont enregistrées. Chaque esclave qui prend en charge la zone de mémoire physique demandée (adresse et longueur de la mémoire physique) doit répondre à ce service.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données de lecture collectées avant l'entrée ou les valeurs par défaut du maître. Ce paramètre doit contenir le résultat de l'opération OR au niveau du bit entre les données de paramètre de la demande et les données adressées dans l'esclave.

WKC

Ce paramètre doit être incrémenté de un par tous les esclaves qui ont créé l'opération OR au niveau du bit des données demandées.

5.4.1.5 Lecture logique (LRD)

Le codage LRD (lecture logique) est spécifié dans le Tableau 17. L'esclave copie uniquement les données dans les données de paramètre mappées par une entité FMMU entre l'espace d'adresse logique et une adresse physique.

Tableau 17 – Lecture logique (LRD)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x0A (commande LRD)
	IDX	Unsigned8	Indice
	ADR	DWORD	Adresse logique
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	réservé à un usage ultérieur
	DATA	OctetString LEN	Données, structure telle que spécifiée par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADR

Ce paramètre doit contenir l'adresse de début de la mémoire logique dans laquelle les données à lire sont placées. Tous les esclaves dont les entités FMMU contiennent une ou plusieurs correspondances d'adresse de la zone de mémoire logique demandée (adresse et longueur de mémoire logique) doivent mapper les données demandées avec le paramètre de données, comme indiqué par les paramètres de l'entité FMMU, puis incrémenter le compteur en fonction.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Dès confirmation, ce paramètre spécifie la lecture de données dans l'appareil. Chaque esclave qui détecte une correspondance d'adresse de la zone de mémoire logique demandée place les données de la zone de mémoire physique correspondante dans la partie appropriée de ce paramètre.

WKC

Ce paramètre doit être incrémenté de un par tous les esclaves qui détectent une correspondance d'adresse de la zone de mémoire logique demandée.

5.4.2 Écriture

5.4.2.1 Vue d'ensemble

Grâce aux services d'écriture, un maître écrit les données dans le registre ou la mémoire d'un ou de plusieurs esclaves. Le compteur en fonction est incrémenté si l'attribut adressé est présent. Le compteur en fonction peut être incrémenté de un si au moins une partie des données peut être écrite.

5.4.2.2 Écriture physique à incrément automatique (APWR)

Le codage APWR (écriture physique à incrément automatique) est spécifié dans le Tableau 18. Chaque esclave incrémente l'adresse. L'esclave qui reçoit un paramètre d'adresse à incrément automatique portant une valeur nulle exécutera l'opération d'écriture demandée.

Tableau 18 – Écriture physique à incrément automatique (APWR)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x02 (commande APWR)
	IDX	Unsigned8	Indice
	ADP	WORD	Adresse à incrément automatique
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

L'esclave sera adressé en fonction de sa position dans le segment. Chaque esclave doit incrémenter ce paramètre, et celui qui reçoit ce paramètre avec une valeur nulle doit répondre à ce service.

NOTE Cela signifie que le paramètre contient la position négative de l'esclave dans l'anneau logique commençant par 0 côté maître (par exemple, -7 signifie que 7 esclaves se trouvent entre le maître et l'esclave adressé). À la confirmation, ce paramètre contient la valeur de la demande incrémentée du nombre d'appareils esclaves transités.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à écrire sont enregistrées.

LEN

Ce paramètre doit contenir la taille, en octets, des données à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire.

WKC

Ce paramètre doit être incrémenté de un si l'écriture des données peut aboutir.

5.4.2.3 Écriture physique de l'adresse configurée (FPWR)

Le codage FPWR (écriture physique de l'adresse configurée) est spécifié dans le Tableau 19.

Tableau 19 – Écriture physique de l'adresse configurée (FPWR)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x05 (commande FPWR)
	IDX	Unsigned8	Indice
	ADP	WORD	Adresse de station configurée ou alias de station configuré
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

L'esclave dont la valeur de D_address fait office d'adresse de station ou d'alias de l'adresse de station doit exécuter une action d'écriture.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à écrire sont enregistrées.

LEN

Ce paramètre doit contenir la taille, en octets, des données à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire.

WKC

Ce paramètre doit être incrémenté de un si l'écriture des données a abouti.

5.4.2.4 Écriture de diffusion (BWR)

Le codage BWR (écriture de diffusion) est spécifié dans le Tableau 20.

Tableau 20 – Écriture de diffusion (BWR)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x08 (commande BWR)
	IDX	Unsigned8	Indice
	ADP	WORD	Paramètre incrémenté de 1 au niveau de chaque station transférant la PDU BWR
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

Ce paramètre doit être incrémenté de un au niveau de chaque esclave.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique dans laquelle les données à écrire sont enregistrées. Chaque esclave qui prend en charge la zone de mémoire physique demandée (adresse et longueur de la mémoire physique) doit répondre à ce service.

LEN

Ce paramètre doit contenir la taille, en octets, des données à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire.

WKC

Ce paramètre doit être incrémenté de un par tous les esclaves qui écrivent des données dans leur mémoire physique.

5.4.2.5 Écriture logique (LWR)

Le codage LWR (écriture logique) est spécifié dans le Tableau 21. L'esclave copie uniquement les données dans la mémoire ou le registre mappées par une entité FMMU entre l'espace d'adresse logique et une adresse physique.

Tableau 21 – Écriture logique (LWR)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x0B (commande LWR)
	IDX	Unsigned8	Indice
	ADR	DWORD	Adresse logique
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	réservé à un usage ultérieur
	DATA	OctetString LEN	Données, structure telle que spécifiée par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADR

Ce paramètre doit contenir l'adresse de début de la mémoire logique dans laquelle les données à écrire sont placées. Tous les esclaves dont les FMMU contiennent une ou plusieurs correspondances d'adresse de la zone de mémoire logique demandée (adresse et longueur de mémoire logique) doivent répondre à ce service.

LEN

Ce paramètre doit contenir la taille, en octets, des données à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire. Chaque esclave qui détecte une correspondance d'adresse de la zone de mémoire logique demandée placera les données de la partie appropriée de ce paramètre dans la zone de mémoire physique correspondante.

WKC

Ce paramètre doit être incrémenté de un par tous les esclaves qui détectent une correspondance d'adresse de la zone de mémoire logique demandée, et si les données ont été correctement écrites.

5.4.3 Lecture écriture

5.4.3.1 Lecture/écriture physiques à incrément automatique (APRW)

Le codage facultatif APRW (lecture/écriture physique à incrément automatique) est spécifié dans le Tableau 22. Chaque esclave incrémente l'adresse. L'esclave qui reçoit un paramètre d'adresse à incrément automatique portant une valeur nulle exécutera l'opération demandée.

Tableau 22 – Lecture/écriture physiques à incrément automatique (APRW)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x03 (commande APRW)
	IDX	Unsigned8	Indice
	ADP	WORD	Adresse à incrément automatique
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

L'esclave sera adressé en fonction de sa position dans le segment. Chaque esclave doit incrémenter ce paramètre, et celui qui reçoit ce paramètre avec une valeur nulle doit répondre à ce service.

NOTE Cela signifie que le paramètre contient la position négative de l'esclave dans l'anneau logique commençant par 0 côté maître (par exemple, -7 signifie que 7 esclaves se trouvent entre le maître et l'esclave adressé). À la confirmation, ce paramètre contient la valeur de la demande incrémentée du nombre d'appareils esclaves transités.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à lire et à écrire sont enregistrées.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire et à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire et les données lues de l'esclave adressé si l'exécution du service peut aboutir.

WKC

Ce paramètre doit être incrémenté de deux si les données ont été correctement écrites, puis de un si elles ont été correctement lues.

5.4.3.2 Écriture/lecture physiques de l'adresse configurée (FPRW)

Le codage facultatif FPRW (écriture/lecture physiques de l'adresse configurée) est spécifié dans le Tableau 23.

Tableau 23 – Écriture/lecture physiques de l'adresse configurée (FPRW)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x06 (commande FPRW)
	IDX	Unsigned8	Indice
	ADP	WORD	Adresse de station configurée ou alias de station configuré
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, 6 par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

L'esclave dont la valeur de D_address fait office d'adresse de station ou d'alias de l'adresse de station doit exécuter une action de lecture, suivie d'une action d'écriture.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à lire et à écrire sont enregistrées.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire et à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire et les données lues de l'esclave adressé si l'exécution du service peut aboutir.

WKC

Ce paramètre doit être incrémenté de deux si les données ont été correctement écrites, puis de un si elles ont été correctement lues.

5.4.3.3 Lecture/écriture de diffusion (BRW)

Le codage facultatif BRW (lecture/écriture de diffusion) est spécifié dans le Tableau 24.

Tableau 24 – Lecture/écriture de diffusion (BRW)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x09 (commande BRW)
	IDX	Unsigned8	Indice
	ADP	WORD	Paramètre incrémenté de 1 au niveau de chaque station transférant la PDU BRW
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, 6 par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

Ce paramètre doit être incrémenté de un au niveau de chaque esclave.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique dans laquelle les données à lire et à écrire sont enregistrées. Chaque esclave qui prend en charge la zone de mémoire physique demandée (adresse et longueur de la mémoire physique) doit répondre à ce service.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire et à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données précédant l'entrée et sera écrit. Une opération de lecture est exécutée avant l'opération d'écriture. Ce paramètre doit contenir le résultat de l'opération OR au niveau du bit entre les données de paramètre de la demande et les données adressées dans l'esclave.

WKC

Ce paramètre doit être incrémenté de deux par tous les esclaves si les données ont été correctement écrites, puis de un par tous les esclaves qui ont créé l'opération OR au niveau du bit des données demandées.

5.4.3.4 Lecture/écriture logique (LRW)

Le codage facultatif LRW (lecture/écriture logique) est spécifié dans le Tableau 25. Grâce à ce service, un appareil esclave peut extraire (opération d'écriture) et placer (opération de lecture) des données. L'esclave copie ou extrait uniquement les données entre les données de paramètre mappées par une entité FMMU de l'espace d'adresse logique vers une adresse physique. . Il est fortement recommandé de prendre en charge cette commande pour une meilleure performance du système.

Tableau 25 – Lecture/écriture logique (LRW)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x0C (commande LRW)
	IDX	Unsigned8	Indice
	ADR	DWORD	Adresse logique
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Réservé à un usage ultérieur
	DATA	OctetString LEN	Données, structure telle que spécifiée par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADR

Ce paramètre doit contenir l'adresse de début de la mémoire logique dans laquelle les données à lire ou à écrire sont placées. Tous les esclaves dont la FMMU contient une ou plusieurs correspondances d'adresse de la zone de mémoire logique demandée (adresse et longueur de mémoire logique) doivent répondre à ce service.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire et à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire. Chaque esclave qui détecte une correspondance d'adresse de la zone de mémoire logique demandée place les données de la partie appropriée de ce paramètre dans la zone de mémoire physique correspondante. Avec la confirmation, ce paramètre doit contenir les données lues. Chaque esclave qui détecte une correspondance d'adresse de la zone de mémoire logique adressée place les données de la zone de mémoire physique correspondante dans la partie appropriée de ce paramètre.

WKC

Ce paramètre doit être incrémenté de deux par chaque esclave si une partie des données a été correctement écrite, puis de un si une partie des données a été correctement lue.

5.4.3.5 Écriture multiple/lecture physique à incrément automatique (ARMW)

Le codage ARMW (écriture multiple/lecture physique à incrément automatique) est spécifié dans le Tableau 26.

Tableau 26 – Écriture multiple/lecture physique à incrément automatique (ARMW)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x0D (commande ARMW)
	IDX	Unsigned8	Indice
	ADP	WORD	Incrément automatique ou adresse de registre
	ADO	WORD	Mémoire physique ou adresse de registre
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

L'esclave sera adressé en fonction de sa position dans le segment. Chaque esclave doit incrémenter ce paramètre. Celui qui reçoit ce paramètre avec une valeur nulle doit exécuter une action de lecture, les autres doivent exécuter une action d'écriture.

NOTE Cela signifie que le paramètre contient la position négative de l'esclave dans l'anneau logique commençant par 0 côté maître (par exemple, -7 signifie que 7 esclaves se trouvent entre le maître et l'esclave adressé). À la confirmation, ce paramètre contient la valeur de la demande incrémentée du nombre d'appareils esclaves transités.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à lire et à écrire sont enregistrées.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire et à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire et les données lues de l'esclave adressé si l'exécution du service peut aboutir.

WKC

Ce paramètre doit être incrémenté de un par chaque esclave si les données ont été correctement lues ou écrites.

5.4.3.6 Écriture multiple/lecture physique de l'adresse configurée (FRMW)

Le codage FRMW (écriture multiple/lecture physique de l'adresse configurée) est spécifié dans le Tableau 27.

Tableau 27 – Écriture multiple/lecture physique de l'adresse configurée (FRMW)

Partie de trame	Champ de données	Type de données	Valeur/description
PDU Type 12	CMD	Unsigned8	0x0E (commande FRMW)
	IDX	Unsigned8	Indice
	ADP	WORD	Adresse de station configurée ou alias de station configuré
	ADO	WORD	Adresse de mémoire physique
	LEN	Unsigned11	Longueur du champ de données DATA
	Reserved (réservé)	Unsigned3	0x00
	C	Unsigned1	Trame en circulation 0: La trame ne circule pas, 1: La trame a circulé une fois
	NEXT	Unsigned1	0x00: dernière PDU Type 12 de la trame Type 12 0x01: la PDU Type 12 de la trame Type 12 suit
	IRQ	WORD	Événement externe
	DATA	OctetString LEN	Données, structure spécifiée en 5.6, à l'Article 6 ou par l'utilisateur DLS
	WKC	WORD	Compteur en fonction

CMD

Le paramètre Command doit contenir la commande du service.

IDX

Le paramètre Index est l'identificateur local du service dans le maître; il ne doit pas être modifié par l'esclave.

ADP

L'esclave dont la valeur de D_address fait office d'adresse de station ou d'alias de l'adresse de station doit exécuter une action de lecture.

ADO

Ce paramètre doit contenir l'adresse de début de la mémoire physique de l'esclave dans laquelle les données à lire et à écrire sont enregistrées.

LEN

Ce paramètre doit contenir la taille, en octets, des données à lire et à écrire.

C

Ce paramètre doit indiquer que la trame a circulé dans le réseau et ne doit pas être transférée.

NEXT

Ce paramètre doit spécifier l'éventuelle présence d'une autre PDU Type 12 dans la trame.

IRQ

Ce paramètre doit contenir l'événement externe (voir Tableau 38) caché par le masque d'événement externe (voir Tableau 39).

DATA

Ce paramètre doit contenir les données à écrire et les données lues de l'esclave adressé si l'exécution du service peut aboutir.

WKC

Ce paramètre doit être incrémenté de un par tous les esclaves si les données ont été correctement lues ou écrites.

5.5 Structure de variable de réseau

Le codage de la variable réseau est spécifié dans le Tableau 28.

Tableau 28 – Variable de réseau

Partie de trame	Champ de données	Type de données	Valeur/description
Variable de réseau	Indice	Unsigned16	Indice d'un objet d'utilisateur DLS
	HASH	Unsigned16	Algorithme de hachage de structure de données permettant de détecter les modifications
	LEN	Unsigned16	Longueur
	Q	Unsigned16	Qualité
	DATA	OctetString [LEN]	Données, structure telle que spécifiée par l'utilisateur DLS

5.6 Structure de boîte aux lettres de type 12

Le codage de la boîte aux lettres est spécifié dans le Tableau 29. Le codage de la boîte aux lettres doit être utilisé conjointement avec les éléments de mémoire de la boîte aux lettres Type 12 ou sous la forme d'un codage des structures de données acheminant les boîtes aux lettres via la DL Ethernet ou le protocole IP.

Tableau 29 – Boîte aux lettres

Partie de trame	Champ de données	Type de données	Valeur/description
Boîte aux lettres	Longueur	Unsigned16	Longueur des données du service de boîte aux lettres
	Adresse	WORD	Adresse de station de la source si un maître est client. Adresse de station de la destination si un esclave est client ou que les données sont transmises hors du segment Type 12 cible
	Canal	Unsigned6	0x00 (Réservé pour un usage futur)
	Priorité	Unsigned2	0x00: priorité la plus faible … 0x03: priorité la plus élevée
	Type	Unsigned4	0x00: error(ERR) 0x01: réservé 0x02: Ethernet sur Type 12 (EoE) 0x03: Protocole d'application CAN sur Type 12 (CoE) 0x04: Accès au fichier sur Type 12 (FoE) 0x05: Profil Servo commande sur Type 12 (SoE) 0x06 -0x0e: réservé 0x0f: spécifique au fournisseur
	Cnt	Unsigned3	Compteur des services de boîte aux lettres (0 est réservé, 1 est la valeur de début, la valeur qui suit 7 est 1. L'esclave doit incrémenter la valeur Cnt de chaque nouveau service de boîte aux lettres, le maître doit s'en assurer pour la détection des services perdus de boîte aux lettres. Le maître doit modifier la valeur Cnt (il convient qu'il l'incrémente). L'esclave doit s'en assurer pour la détection d'un service de répétition d'écriture. L'esclave ne doit pas vérifier la séquence de la valeur Cnt. Les valeurs Cnt du maître et de l'esclave sont indépendantes
	Reserved (réservé)	Unsigned1	0x00
	Données de service	OctetString [Length]	Données de service de boîte aux lettres

Le codage des données de service en cas de réponse d'erreur est spécifié dans le Tableau 30.

Tableau 30 – Données de service de réponse d'erreur

Partie de trame	Champ de données	Type de données	Valeur/description
Données de service	Type	Unsigned16	0x01: Commande de boîte aux lettres
	Détail	Unsigned16	0x01: MBXERR_SYNTAX La syntaxe de l'en-tête de boîte aux lettres à 6 octets est erronée 0x02: MBXERR_UNSUPPORTEDPROTOCOL Le protocole de boîte aux lettres n'est pas pris en charge 0x03: MBXERR_INVALIDCHANNEL Le champ relatif au canal contient une valeur incorrecte (un esclave peut ignorer ce champ) 0x04: MBXERR_SERVICENOTSUPPORTED Le service du protocole de boîte aux lettres n'est pas pris en charge 0x05: MBXERR_INVALIDHEADER L'en-tête du protocole de boîte aux lettres est incorrect (sans l'en-tête de boîte aux lettres à 6 octets) 0x06: MBXERR_SIZETOOSHORT La longueur des données de boîte aux lettres reçues est trop courte 0x07: MBXERR_NOMOREMEMORY Impossible de traiter le protocole de boîte aux lettres en raison de ressources limitées 0x08: MBXERR_INVALIDSIZE La longueur des données est incohérente

6 Attributs

6.1 Gestion

6.1.1 Informations DL

Les registres d'informations DL contiennent le type, la version et les ressources prises en charge du contrôleur d'esclave (ESC).

Paramètre

Type

Ce paramètre doit contenir le type de contrôleur d'esclave.

Revision (révision majeure)

Ce paramètre doit contenir la révision du contrôleur d'esclave.

Build (révision mineure)

Ce paramètre doit contenir le numéro de version du contrôleur d'esclave.

NoOfSuppFmmuEntities

Ce paramètre doit contenir le nombre d'entités FMMU prises en charge du contrôleur d'esclave.

NoOfSuppSyncManChannels

Ce paramètre doit contenir le nombre de canaux (ou d'entités) du gestionnaire de synchronisation pris en charge du contrôleur d'esclave.

RamSize

Ce paramètre doit contenir la taille de la mémoire RAM en Ko prise en charge par le contrôleur d'esclave (une taille inférieure à un nombre pair sera arrondie à la borne inférieure).

PortDescr

Port 0 Physical Layer

Il convient que ce paramètre indique la couche physique utilisée pour ce port.

Port 1 Physical Layer

Il convient que ce paramètre indique la couche physique utilisée pour ce port.

Port 2 Physical Layer

Il convient que ce paramètre indique la couche physique utilisée pour ce port.

Port 3 Physical Layer

Il convient que ce paramètre indique la couche physique utilisée pour ce port.

Fonctions prises en charge

FmmuBitOperationNotSupp

Ce paramètre doit indiquer si la FMMU dans le contrôleur d'esclave prend en charge les opérations binaires sans restrictions ou avec des restrictions documentées (seul le mapping au niveau du bit sur des zones de mémoire spécifiques, par exemple).

Le bit de cette fonction n'influe pas sur la possibilité de mapping du fanion SM.WriteEvent (MailboxIn)

DCSupp

Ce paramètre est mis à 1 si au moins les heures de réception de l'horloge distribuée sont prises en charge.

DCRange

Ce paramètre doit indiquer la plage de valeurs d'horloge (0: 32 bit/1:64 bit).

LowJEBUS

Ce paramètre doit indiquer que la fonction de faible gigue est disponible.

EnhLDEBUS

Ce paramètre doit indiquer que la détection de liaison améliorée est disponible pour les ports EBUS.

EnhLDMII

Ce paramètre doit indiquer que la détection de liaison améliorée est disponible pour les ports MII.

FCSsERR

Ce paramètre doit indiquer que les erreurs induites par un autre esclave de type 12 seront comptées séparément.

Enhanced DC Sync Activation

Ce paramètre doit indiquer que l'activation améliorée de la synchronisation de l'horloge distribuée est disponible.

NotSuppLRW

Ce paramètre doit indiquer que LRW n'est pas pris en charge.

NotSuppBAFRW

Ce paramètre doit indiquer que BRW, APRW et FPRW ne sont pas pris en charge.

sFMMUSyMC

Ce paramètre doit indiquer qu'une configuration particulière du FMMU/gestionnaire de synchronisation est utilisée:

FMMU 0 est utilisé pour RxPDO (pas de mapping de bit)

FMMU 1 est utilisé pour TxPDO (pas de mapping de bit)

FMMU 2 est utilisée pour le bit d'événement d'écriture de boîte aux lettres du gestionnaire de synchronisation 1 (l'opération sur bit de la FMMU est prise en charge pour ce bit)

Le gestionnaire de synchronisation 0 est utilisé pour la boîte aux lettres en écriture

Le gestionnaire de synchronisation 1 est utilisé pour la boîte aux lettres en lecture

Le gestionnaire de synchronisation 2 fait office de mémoire tampon pour RxPDO

Le gestionnaire de synchronisation 3 fait office de mémoire tampon pour TxPDO

Les types d'attribut des informations DL sont décrits à la Figure 8.

```
typedef struct
{
  BYTE          Type;
  BYTE          Revision;
  WORD          Build;
  BYTE          NoOfSuppFmmuEntities;
  BYTE          NoOfSuppSyncManChannels;
  BYTE          RamSize;
  BYTE          PortDescr;
  unsigned      FmmuBitOperationNotSupp:  1;
  unsigned      Reserved2:                1;
  unsigned      DCSupp:                   1;
  unsigned      DCRange:                  1;
  unsigned      LowJEBUS:                 1;
  unsigned      EnhLDEBUS:                1;
  unsigned      EnhLDMII:                 1;
  unsigned      FCSsERR:                  1;
  unsigned      EnhancedDcSyncAct:        1;
  unsigned      NotSuppLRW:               1;
  unsigned      NotSuppBAFRW:             1;
  unsigned      sFMMUSyMC:                1;
  unsigned      Reserved4:                4;
} TDLINFORMATION;
```

Figure 8 – Description du type d'informations DL

Le codage des informations DL est spécifié dans le Tableau 31.

Tableau 31 – Informations DL

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Type	0x0000	BYTE	R	R	
Revision (révision majeure)	0x0001	BYTE	R	R	
Build (révision mineure)	0x0002	WORD	R	R	
NoOfSuppFmmuEntities	0x0004	BYTE	R	R	0x0001-0x0010
NoOfSuppSyncManChannels	0x0005	BYTE	R	R	0x0001-0x0010
RAMSize	0x0006	BYTE	R	R	La taille de la mémoire vive en Ko signifie 1024 octets (1-60)
PortDescr	0x0007	unsigned2	R	R	facultatif 00: Non mis en œuvre 01: Non configuré 10: EBUS 11: MII/RMII
Port1 Descriptor	0x0007	unsigned2	R	R	facultatif 00: Non mis en œuvre 01: Non configuré 10: EBUS 11: MII/RMII

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Port2 Descriptor	0x0007	unsigned2	R	R	facultatif 00: Non mis en œuvre 01: Non configuré 10: EBUS 11: MII/RMII
Port3 Descriptor	0x0007	unsigned2	R	R	facultatif 00: Non mis en œuvre 01: Non configuré 10: EBUS 11: MII/RMII
FmmuBitOperationNotSupp	0x0008	unsigned1	R	R	0: opération sur bit prise en charge 1: opération sur bit non prise en charge Le bit de cette fonction n'influe pas sur la possibilité de mapping du fanion SM.WriteEvent (MailboxIn)
Reserved (réservé)	0x0008	unsigned1	R	R	
DCSupp	0x0008	unsigned1	R	R	0: DC non pris en charge 1: DC pris en charge
DCRange	0x0008	unsigned1	R	R	0: 32 bit 1: 64 bits pour le temps système, le décalage de temps système et le temps de réception au niveau de l'unité de traitement du temps
Low Jitter EBUS	0x0008	unsigned1	R	R	0: non disponible 1: disponible
Enhanced Link Detection EBUS	0x0008	unsigned1	R	R	0: non disponible 1: disponible
Enhanced Link Detection MII	0x0008	unsigned1	R	R	0: non disponible 1: disponible
Separate Handling of FCS errors	0x0008	unsigned1	R	R	0: inactif 1: actif, Il convient de compter séparément les trames avec une séquence FCS modifiée (quartet supplémentaire) dans le compteur précédent d'erreurs RX
Enhanced DC Sync Activation	0x0009	unsigned1	R	R	0: non disponible 1: disponible Cette caractéristique se rapporte aux registres 0x981[7:3], 0x0984
NotSuppLRW	0x0009	unsigned1	R	R	0: LRW pris en charge 1: LRW non pris en charge
BRW, APRW; NotSuppFPRW	0x0009	unsigned1	R	R	0: BRW, APRW; FPRW pris en charge 1: BRW, APRW; FPRW non pris en charge
sFMMUSyMC	0x0009	unsigned1	R	R	0: inactif 1: actif,

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
					FMMU 0 est utilisé pour RxPDO (pas de mapping de bit) FMMU 1 est utilisé pour TxPDO (pas de mapping de bit) FMMU 2 est utilisé pour le bit d'événement d'écriture de boîte aux lettres du gestionnaire de synchronisation 1 Le gestionnaire de synchronisation 0 est utilisé pour la boîte aux lettres en écriture Le gestionnaire de synchronisation 1 est utilisé pour la boîte aux lettres en lecture Le gestionnaire de synchronisation 2 fait office de mémoire tampon pour les données entrantes Le gestionnaire de synchronisation 3 fait office de mémoire tampon pour les données sortantes
Reserved (réservé)	0x0009	unsigned4	R	R	

6.1.2 Adresse de station

Le registre d'adresse de station configurée contient l'adresse de station de l'esclave qui sera définie pour activer les services FPRD, FPRW, FRMW et FPWR dans le contrôleur d'esclave.

Paramètre

ConfiguredStationAddress

Ce paramètre doit contenir l'adresse de station configurée du contrôleur d'esclave qui est définie par le maître au démarrage.

ConfiguredStationAlias

Ce paramètre doit contenir l'alias de station configuré du contrôleur d'esclave qui est défini par l'utilisateur de DL au démarrage.

Les types d'attribut de l'adresse de station sont décrits à la Figure 9.

```
typedef struct
{
  WORD          ConfiguredStationAddress;
  WORD          ConfiguredStationAlias;
} TFIXEDSTATIONADDRESS;
```

Figure 9 – Description du type d'adresse

Le codage de l'adresse de station est spécifié dans le Tableau 32.

Tableau 32 – Adresse de station configurée

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
ConfiguredStationAddress	0x0010	WORD	RW	R	
ConfiguredStationAlias	0x0012	WORD	RW	RW	Initialisé avec SII mot 4

6.1.3 Commande DL

Le registre de commande DL permet au maître de contrôler le fonctionnement des ports DL du contrôleur d'esclave.

Paramètre

ForwardingRule

Ce paramètre doit activer le transfert direct ou le transfert restreint. Le transfert restreint détruira les trames non Type 12.

TemporaryLoopControl

Ce paramètre facultatif permet l'utilisation temporaire des paramètres de contrôle de boucle écrits dans la même trame pour environ une seconde. À l'expiration de ce délai, les paramètres originaux de contrôle de boucle sont restaurés automatiquement.

LoopControlPort0

Ce paramètre doit contenir les informations indiquant si une activation automatique du port est prévue en cas de liaison physique ou si le port est ouvert et/ou fermé par les commandes du maître.

LoopControlPort1

Ce paramètre doit contenir les informations indiquant si une activation automatique du port est prévue en cas de liaison physique ou si le port est ouvert et/ou fermé par les commandes du maître.

LoopControlPort2

Ce paramètre doit contenir les informations indiquant si une activation automatique du port est prévue en cas de liaison physique ou si le port est ouvert et/ou fermé par les commandes du maître.

LoopControlPort3

Ce paramètre doit contenir les informations indiquant si une activation automatique du port est prévue en cas de liaison physique ou si le port est ouvert et/ou fermé par les commandes du maître.

TxBufferSize

Il convient d'utiliser ce paramètre facultatif pour optimiser le délai dans une station. Si la vitesse de transmission de cette station et de ses voisines est stable, ce paramètre peut être réduit. Les paramètres par défaut sont déterminés par la précision d'horloge requise de l'ISO/CEI 8802-3.

LowJEBUS

Ce paramètre facultatif indique que la réduction de la gigue de transfert de la trame pour EBUS est activée.

EnableAliasAddress

Il convient d'utiliser ce paramètre facultatif pour activer le nom d'alias.

Les types d'attribut de commande DL sont décrits à la Figure 10.

```
typedef struct
{
  unsigned      ForwardingRule:        1;
  unsigned      TemporaryLoopControl:  1;
  unsigned      Reserved0:             6;
  unsigned      LoopControlPort0:      2;
  unsigned      LoopControlPort1:      2;
  unsigned      LoopControlPort2:      2;
  unsigned      LoopControlPort3:      2;
  unsigned      TxBufferSize:          3;
  unsigned      LowJitterEBUS:         1;
  unsigned      Reserved1:             4;
  unsigned      EnableAliasAddress:    1;
  unsigned      Reserved2:             7;
} TDLCONTROL;
```

Figure 10 – Description du type de commande DL

Le codage de la commande DL est spécifié dans le Tableau 33.

Tableau 33 – Commande DL

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
ForwardingRule	0x0100	Unsigned1	RW	R	0: Les trames EtherCAT sont traitées, Les trames Non-EtherCAT sont transmises sans traitement, SOURCE_MAC[1] peut être mis à 1 – adresse localement administrée 1: Les trames EtherCAT sont traitées, Les trames Non-EtherCAT sont rejetées, SOURCE_MAC[1] doit être mis à 1 – adresse localement administrée
TemporaryLoopControl	0x0100	Unsigned1	RW	R	0: Configuration permanente 1: utilisation temporaire des paramètres de contrôle de boucle pour ~1 seconde
Reserved (réservé)	0x0100	Unsigned6	RW	R	0x00
LoopControlPort0	0x0101	Unsigned2	RW	R	0: Auto => fermé lors de «l'interruption de liaison» (link down), ouvert lors de « l'établissement de liaison » (link up) 1: Fermeture auto => fermé lors de «l'interruption de liaison», ouvert avec écriture 1 et «établissement de liaison» (link up) (ou réception d'une trame Ethernet valide au niveau du port fermé) 2: Toujours ouvert 3: Toujours fermé
LoopControlPort1	0x0101	Unsigned2	RW	R	0: Auto => fermé lors de «l'interruption de liaison» (link down), ouvert lors de «l'établissement de

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
					liaison» (link up) 1: Fermeture auto => fermé lors de «l'interruption de liaison», ouvert avec écriture de 1 1 et «établissement de liaison» (link up) (ou réception d'une trame Ethernet valide au niveau du port fermé) 2: Toujours ouvert 3: Toujours fermé
LoopControlPort2	0x0101	Unsigned2	RW	R	0: Auto => fermé lors de «l'interruption de liaison» (link down), ouvert lors de «l'établissement de liaison» (link up) 1: Fermeture auto => fermé lors de «l'interruption de liaison», ouvert avec écriture de 1 1 et «établissement de liaison» (link up) (ou réception d'une trame Ethernet valide au niveau du port fermé) 2: Toujours ouvert 3: Toujours fermé
LoopControlPort3	0x0101	Unsigned2	RW	R	0: Auto => fermé lors de «l'interruption de liaison» (link down), ouvert lors de «l'établissement de liaison» (link up) 1: Fermeture auto => fermé lors de «l'interruption de liaison», ouvert avec écriture de 1 1 et «établissement de liaison» (link up) (ou réception d'une trame Ethernet valide au niveau du port fermé) 2: Toujours ouvert 3: Toujours fermé
TransmitBufferSize	0x0102	Unsigned3	RW	R	Mémoire tampon entre la préparation et l'envoi. L'envoi aura lieu si la mémoire tampon est à moitié pleine (7).
Low Jitter EBUS	0x0102	Unsigned1	RW	R	0: inactif 1: actif
Reserved (réservé)	0x0102	Unsigned4	RW	R	0x00
EnableAliasAddress	0x0103	Unsigned1	RW	R	0: Désactiver l'adresse d'alias de station 1: Activer l'adresse d'alias de station
Reserved (réservé)	0x0103	Unsigned7	RW	R	0x00

NOTE Boucle ouverte s'entend souvent d'un envoi sur ce port et de l'attente d'une réaction du port destinataire – les données reçues seront transférées au port homologue. Boucle fermée s'entend souvent de données qu'il convient d'envoyer sont directement inversées, et donc transférées au port homologue. Un port fermé écartera toutes les données reçues.

6.1.4 État de DL

Le registre d'état de DL permet d'indiquer l'état des ports DL et de l'interface entre l'utilisateur de DL et la couche Liaison de Données.

Paramètre

DL-user operational

Ce paramètre doit contenir les informations si un utilisateur de DL est connecté à l'interface de données de processus du contrôleur d'esclave.

DLSuserWatchdogStatus

Ce paramètre doit contenir l'état du temporisateur de l'interface de données de processus.

ExtendedLinkDetection

Ce paramètre doit contenir l'état de l'activation de la détection de liaison étendue.

LinkStatusPort0

Ce paramètre indique la liaison physique sur ce port.

LinkStatusPort1

Ce paramètre indique la liaison physique sur ce port.

LinkStatusPort2

Ce paramètre indique la liaison physique sur ce port.

LinkStatusPort3

Ce paramètre indique la liaison physique sur ce port.

LoopBackPort0

Ce paramètre indique le transfert sur le même port, c'est-à-dire une boucle avec retour.

SignalDetectionPort0

Ce paramètre indique si un signal est détecté sur Rx-Port.

LoopBackPort1

Ce paramètre indique le transfert sur le même port, c'est-à-dire une boucle avec retour.

SignalDetectionPort1

Ce paramètre indique si un signal est détecté sur Rx-Port.

LoopBackPort2

Ce paramètre indique le transfert sur le même port, c'est-à-dire une boucle avec retour.

SignalDetectionPort2

Ce paramètre indique si un signal est détecté sur Rx-Port.

LoopBackPort3

Ce paramètre indique le transfert sur le même port, c'est-à-dire une boucle avec retour.

SignalDetectionPort3

Ce paramètre indique si un signal est détecté sur Rx-Port.

Les types d'attribut de l'état DL sont décrits à la Figure 11.

```
typedef struct
{
  unsigned      PdiOperational:           1;
  unsigned      DLSuserWatchdogStatus:        1;
  unsigned      ExtendedLinkDetection:    1;
  unsigned      Reserved1:                1;
  unsigned      LinkStatusPort0:          1;
  unsigned      LinkStatusPort1:          1;
  unsigned      LinkStatusPort2:          1;
  unsigned      LinkStatusPort3:          1;
  unsigned      LoopStatusPort0:          1;
  unsigned      SignalDetectionPort0:     1;
  unsigned      LoopStatusPort1:          1;
  unsigned      SignalDetectionPort1:     1;
  unsigned      LoopStatusPort2:          1;
  unsigned      SignalDetectionPort2:     1;
  unsigned      LoopStatusPort3:          1;
  unsigned      SignalDetectionPort3:     1;
} TDLSTATUS;
```

Figure 11 – Description du type d'état DL

Le codage de l'état DL est spécifié dans le Tableau 34.

Tableau 34 – État DL

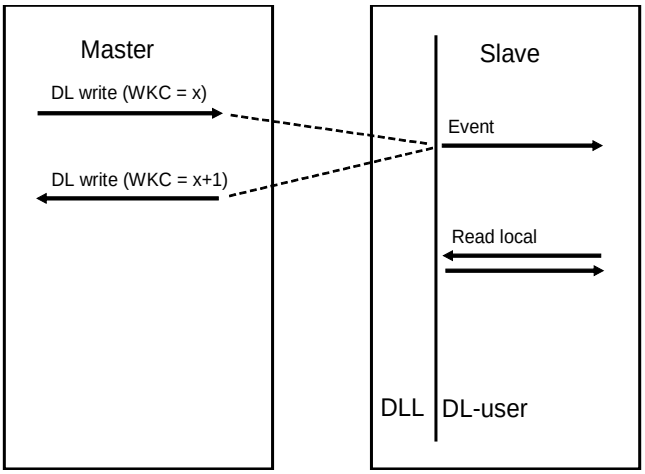
Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
DLS-user operational	0x0110	Unsigned1	R	R	0x00: Utilisateur DLS non opérationnel 0x01: Utilisateur DLS opérationnel
DLS-user watchdog status	0x0110	Unsigned1	R	R	0x00: Le temporisateur de l'utilisateur DLS a expiré 0x01: Le temporisateur de l'utilisateur DLS n'a pas expiré
ExtendedLinkDetection	0x0110	Unsigned1	R	R	0: Désactivé 1: Activé pour au moins un port
Reserved (réservé)	0x0110	Unsigned1	R	R	0x00
LinkStatusPort0	0x0110	Unsigned1	R	R	0x00: pas de liaison physique sur ce port 0x01: liaison physique sur ce port
LinkStatusPort1	0x0110	Unsigned1	R	R	0x00: pas de liaison physique sur ce port 0x01: liaison physique sur ce port
LinkStatusPort2	0x0110	Unsigned1	R	R	0x00: pas de liaison physique sur ce port 0x01: liaison physique sur ce port
LinkStatusPort3	0x0110	Unsigned1	R	R	0x00: pas de liaison physique sur ce port 0x01: liaison physique sur ce port
LoopStatusPort 0	0x0111	Unsigned1	R	R	0x00: boucle inactive 0x01: boucle active
SignalDetectionPort0	0x0111	Unsigned1	R	R	0x00: signal non détecté sur le port RX 0x01: signal détecté sur le port RX
LoopStatusPort 1	0x0111	Unsigned1	R	R	0x00: boucle inactive 0x01: boucle active
SignalDetectionPort1	0x0111	Unsigned1	R	R	0x00: signal non détecté sur le port RX 0x01: signal détecté sur le port RX
LoopStatusPort 2	0x0111	Unsigned1	R	R	0x00: boucle inactive 0x01: boucle active
SignalDetectionPort2	0x0111	Unsigned1	R	R	0x00: signal non détecté sur le port RX 0x01: signal détecté sur le port RX
LoopStatusPort 3	0x0111	Unsigned1	R	R	0x00: boucle inactive 0x01: boucle active
SignalDetectionPort3	0x0111	Unsigned1	R	R	0x00: signal non détecté

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
					sur le port RX 0x01: signal détecté sur le port RX

6.1.5 Registres spécifiques à l'utilisateur DLS

6.1.5.1 Registre de contrôle de l'utilisateur de DL

La Figure 12 illustre les primitives entre le maître, la DL et l'utilisateur de DL en cas de réussite de la séquence d'écriture dans le registre de commande de l'utilisateur de DL (R1).



Légende

Anglais	Français
Master	Maître
Slave	Esclave
DL write	Écriture DL
Event	Événement
Read local	Lecture locale
DL-User	Utilisateur de DL

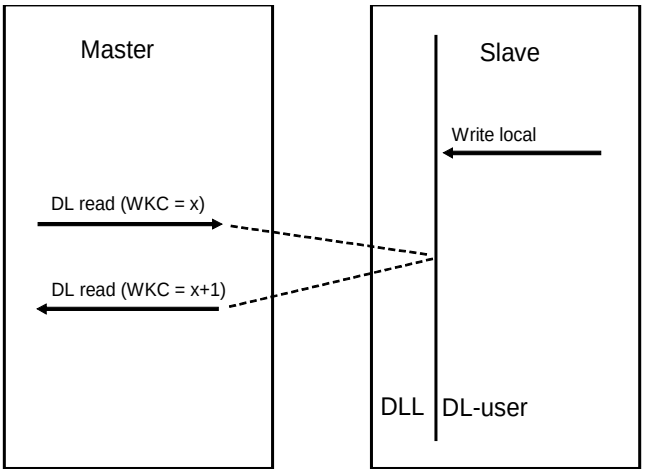
Figure 12 – Séquence d'écriture réussie dans le registre de commande de l'utilisateur de DL

Le maître envoie un service d'écriture avec le compteur en fonction (WKC = x), la DL (contrôleur d'esclave) de l'esclave écrit les données reçues dans la zone de registre, incrémente le compteur en fonction (WKC = x + 1) et génère un événement, puis l'utilisateur de DL lit le registre de commande. Si le registre de commande n'est pas lu, l'opération d'écriture suivante dans ce registre est ignorée (n'est pas modifiée).

Le registre de commande permet au maître de transmettre des informations de commande à l'esclave.

6.1.5.2 Registre d'état de l'utilisateur de DL

La Figure 13 illustre les primitives entre le maître, la DL et l'utilisateur de DL en cas de réussite de la séquence de lecture dans le registre d'états de l'utilisateur de DL (R3).



Légende

Anglais	Français
Master	Maître
Slave	Esclave
DL read	Lecture DL
Event	Événement
Write local	Écriture locale
DL-User	Utilisateur de DL

Figure 13 – Séquence de lecture réussie dans le registre d'état de l'utilisateur de DL

L'utilisateur de DL de l'esclave écrit en local dans le registre d'état de l'utilisateur de DL. Le maître envoie un service de lecture avec le compteur en fonction (WKC = x), la DL (contrôleur d'esclave) de l'esclave envoie les données depuis la zone de registre et incrémente le compteur en fonction (WKC = x + 1).

6.1.5.3 Registres spécifiques à l'utilisateur de DL

Il existe un ensemble de registres spécifiques à l'utilisateur de DL R2, R4 à R8. La signification du contenu est définie par l'utilisateur de DL.

6.1.5.4 Attributs de l'utilisateur de DL

La structure de registre spécifique à l'utilisateur DLS et le type d'accès sont décrits dans le Tableau 35.

Tableau 35 – Registres spécifiques à l'utilisateur DLS

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
DLS-user R1	0x0120	Unsigned8	RW	R	0x01
DLS-user R2	0x0121	Unsigned8	RW	R	0x00
DLS-user R3	0x0130	Unsigned8	R	RW	0x01
DLS-user R4	0x0131	Unsigned8	R	RW	0x00
reserved (réservé)	0x0132	Unsigned16			
DLS-user R6	0x0134	Unsigned16	R	RW	0x00
DLS-user R7	0x0140	Unsigned8	R	R	0x00
Copy	0x0141	Unsigned1	R	R	0x00: pas d'action particulière 0x01: Copier DLS-user R1 dans DLS-user R3
DLS-user R9	0x0141	Unsigned7	R	R	0x00
DLS-user R10	0x0142	Unsigned16	R	R	0x00
DLS-user R8	0x0150	Unsigned32	R	R	0x00

6.1.6 Paramètre d'événement

Les registres d'événement permettent d'indiquer un événement à l'utilisateur de DL. L'événement doit être acquitté en cas de lecture de la source d'événement correspondante. Les événements peuvent être masqués.

Paramètre

Événement de l'utilisateur de DL

DL-user R1 Chg

Une valeur est attribuée à ce paramètre si un service d'écriture dans le registre de commande de l'utilisateur de DL est appelé et il est réinitialisé si le registre de commande de l'utilisateur de DL est lu en local.

DC Event 0

Une valeur est attribuée à ce paramètre si DC Event 0 est actif.

DC Event 1

Une valeur est attribuée à ce paramètre si DC Event 1 est actif.

DC Event 2

Une valeur est attribuée à ce paramètre si DC Event 2 est actif.

Sync manager change event

Une valeur est attribuée à ce paramètre si un service d'écriture dans la zone du gestionnaire de synchronisation se produit. Une nouvelle valeur est attribuée si l'utilisateur de DL lit le registre d'événement.

Sync manager channel access events (0 à 15)

Une valeur est attribuée à ce paramètre en cas de réception d'un service d'écriture ou de lecture dans une zone de mémoire d'application configurée en écriture ou en lecture par le maître. Ce paramètre est réinitialisé si la zone de mémoire d'application est lue (lecture locale) ou écrite (écriture locale).

DL-user Event Mask

Si une valeur est attribuée à l'attribut correspondant, l'événement de l'utilisateur de DL sera activé, et désactivé dans le cas contraire.

Événement externe

DC Event 0

Une valeur est attribuée à ce paramètre si DC Event 0 est actif.

DL Status Chg

Une valeur est attribuée à ce paramètre si le registre d'état de DL est modifié. Ce paramètre est réinitialisé si une lecture de Type 12 dans le registre d'état DL est demandée.

DL-user R3 Chg

Une valeur est attribuée à ce paramètre si un service local d'écriture dans le registre d'état de l'utilisateur de DL est appelé. Ce paramètre est réinitialisé si une lecture de Type 12 dans le registre d'état de l'utilisateur de DL est appelée.

Sync manager channel access events (0 à 7)

Une valeur est attribuée à ce paramètre en cas de réception d'un service d'écriture ou de lecture dans une zone de mémoire d'application configurée en écriture ou en lecture par l'esclave. Ce paramètre est réinitialisé si la zone de mémoire d'application est lue ou écrite par les services Type 12.

Event Event Mask

Si une valeur est attribuée à l'attribut correspondant, l'événement externe est activé, et désactivé dans le cas contraire.

La structure d'événement telle que perçue par l'utilisateur DLS et le type d'accès sont décrits dans le Tableau 36.

Tableau 36 – Événement de l'utilisateur DLS

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
DLS-user R1 chg	0x0220	unsigned1	R	r	0x00: pas d'événement actif 0x01: l'événement actif R1 a été écrit
DC event 0	0x0220	unsigned1	R	r	0x00: pas d'événement actif 0x01 événement actif (au moins un événement de verrouillage s'est produit)
DC event 1	0x0220	unsigned1	R	r	0x00: le signal de synchronisation est inactif 0x01 le signal de synchronisation est actif
DC event 2	0x0220	unsigned1	R	r	0x00: le signal de synchronisation est inactif 0x01 le signal de synchronisation est actif
Sync manager change event	0x0220	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (un ou plusieurs canaux du gestionnaire de synchronisation ont été modifiés)
EEPROM Emulation	0x0220	unsigned1	R	r	0x00: Pas de commande en attente 0x01: Commande EEPROM en attente

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
DLE specific (spécifique à la DLE)	0x0220	Unsigned2	R	r	0x00:
Sync manager channel 0 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 1 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 2 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 3 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 4 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 5 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 6 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 7 event	0x0221	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 8 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Sync manager channel 9 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 10 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 11 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 12 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 13 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 14 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
Sync manager channel 15 event	0x0222	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé)
DLE specific (spécifique à la DLE)	0x0223	Unsigned8	R	r	0x00

Le masque d'événement de l'utilisateur DLS est associé à l'événement de l'utilisateur DLS et le codage est spécifié dans le Tableau 37.

Tableau 37 – Masque d'événement de l'utilisateur de DLS

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Event mask	0x0204	Array [0..31] of unsigned1	R	rw	Pour chaque élément: 0: désactiver l'événement 1: activer l'événement

La structure d'événement telle que perçue par le partenaire distant et le type d'accès sont décrits dans le Tableau 38. L'événement externe est mappé au paramètre IRQ de toutes les PDU Type 12 qui accèdent à cet esclave. Si un événement et le masque associé sont définis, le bit correspondant du paramètre IRQ d'une PDU l'est également.

Tableau 38 – Événement externe

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
DC Event 0	0x0210	unsigned1	R	r	0x00: pas d'événement actif 0x01: DC event 0 actif
reserved (réservé)	0x0210	unsigned1	R	r	0x00
DL Status change	0x0210	unsigned1	R	r	0x00: pas d'événement actif 0x01: Le registre d'état DL a été activé
R3 Chg	0x0210	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (R3 a été écrit)
Sync manager channel 0 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)
Sync manager channel 1 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)
Sync manager channel 2 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)
Sync manager channel 3 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)
Sync manager channel 4 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)
Sync manager channel 5 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Sync manager channel 6 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)
Sync manager channel 7 event	0x0211	unsigned1	R	r	0x00: pas d'événement actif 0x01: événement actif (le canal du gestionnaire de synchronisation a été accédé par l'esclave)
Reserved (réservé)	0x0211	Unsigned4	R	r	0x00

Le masque d'événement externe est associé à l'événement externe et le codage est spécifié dans le Tableau 39.

Tableau 39 – Masque d'événement externe

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Event mask	0x0200	Array [0..15] of unsigned1	RW	r	Pour chaque élément: 0: désactiver l'événement 1: activer l'événement

6.2 Statistiques

6.2.1 Compteur d'erreurs Rx

Les registres du compteur d'erreurs Rx contiennent des informations relatives aux erreurs de la couche physique et aux erreurs de trame (longueur, FCS, par exemple). Tous les compteurs sont annulés si l'un d'eux est écrit. Le comptage s'arrête lorsque la valeur maximale d'un compteur (255) est atteinte.

Paramètre

PhyErrorCountPort0

Ce paramètre compte les occurrences d'erreurs RX au niveau de la couche physique.

FrameErrorCountPort0

Ce paramètre compte les occurrences d'erreurs de trame (y compris les erreurs RX à l'intérieur d'une trame).

PhyErrorCountPort1

Ce paramètre compte les occurrences d'erreurs RX au niveau de la couche physique.

FrameErrorCountPort1

Ce paramètre compte les occurrences d'erreurs de trame (y compris les erreurs RX à l'intérieur d'une trame).

PhyErrorCountPort2

Ce paramètre compte les occurrences d'erreurs RX au niveau de la couche physique.

FrameErrorCountPort2

Ce paramètre compte les occurrences d'erreurs de trame (y compris les erreurs RX à l'intérieur d'une trame).

PhyErrorCountPort3

Ce paramètre compte les occurrences d'erreurs RX au niveau de la couche physique.

FrameErrorCountPort3

Ce paramètre compte les occurrences d'erreurs de trame (y compris les erreurs RX à l'intérieur d'une trame).

NOTE Les trames sont traitées pendant la procédure de transfert. Par conséquent, une erreur RX ou une erreur de trame se produira au niveau des stations en aval de la station erronée simultanément. Le maître obtiendra une image réelle en soustrayant les comptages du port précédent.

Les types d'attribut du compteur d'erreurs RX sont décrits à la Figure 14.

```
typedef struct
{
  Unsigned8          FrameErrorCountPort0;
  Unsigned8          PhyErrorCountPort0;
  Unsigned8          FrameErrorCountPort1;
  Unsigned8          PhyErrorCountPort1;
  Unsigned8          FrameErrorCountPort2;
  Unsigned8          PhyErrorCountPort2;
  Unsigned8          FrameErrorCountPort3;
  Unsigned8          PhyErrorCountPort3;
} TRXERRORCOUNTER;
```

Figure 14 – Description du type de compteur d'erreurs RX

Le codage du compteur d'erreurs Rx est spécifié dans le Tableau 40.

Tableau 40 – Compteur d'erreurs RX

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
FrameErrorCountPort0	0x0300	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs
PhyErrorCountPort0	0x0301	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs
FrameErrorCountPort1	0x0302	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs
PhyErrorCountPort1	0x0303	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs
FrameErrorCountPort2	0x0304	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs
PhyErrorCountPort2	0x0305	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs
FrameErrorCountPort3	0x0306	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs
PhyErrorCountPort3	0x0307	unsigned8	RW	--	Une écriture dans un compteur réinitialisera tous les compteurs

6.2.2 Compteur de liaisons perdues

Les registres facultatifs du compteur de liaisons perdues contiennent des informations relatives aux séquences d'interruption de liaison. Tous les compteurs sont annulés si l'un d'eux est écrit. Le comptage s'arrête lorsque la valeur maximale d'un compteur (255) est atteinte.

Paramètre

LostLinkCountPort0

Ce paramètre compte les occurrences d'interruption de liaison.

LostLinkCountPort1

Ce paramètre compte les occurrences d'interruption de liaison.

LostLinkCountPort2

Ce paramètre compte les occurrences d'interruption de liaison.

LostLinkCountPort3

Ce paramètre compte les occurrences d'interruption de liaison.

Les types d'attribut du compteur de liaisons perdues sont décrits à la Figure 15.

```
typedef struct
{
  Unsigned8          LostLinkCountPort0;
  Unsigned8          LostLinkCountPort1;
  Unsigned8          LostLinkCountPort2;
  Unsigned8          LostLinkCountPort3;
} TLOSTLINKCOUNTER;
```

Figure 15 – Description du type de compteur de liaisons perdues

Le codage du compteur de liaisons perdues est spécifié dans le Tableau 41.

Tableau 41 – Compteur de liaisons perdues

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
LostLinkCountPort0	0x0310	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs de liaisons perdues
LostLinkCountPort1	0x0311	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs de liaisons perdues
LostLinkCountPort2	0x0312	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs de liaisons perdues
LostLinkCountPort3	0x0313	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs de liaisons perdues

6.2.3 Compteur supplémentaire

Les registres du compteur d'erreurs précédentes facultatif contiennent des informations relatives aux trames d'erreur indiquant un problème sur les liaisons du prédécesseur. Les trames erronées présentant un type particulier de somme de contrôle, il est possible de les détecter et de les signaler. Tous les compteurs sont annulés si l'un d'eux est écrit. Le comptage s'arrête lorsque la valeur maximale d'un compteur (255) est atteinte.

Paramètre

PreviousErrCountPort0

Ce paramètre compte les occurrences d'erreurs détectées par le prédécesseur.

PreviousErrCountPort1

Ce paramètre compte les occurrences d'erreurs détectées par le prédécesseur.

PreviousErrCountPort2

Ce paramètre compte les occurrences d'erreurs détectées par le prédécesseur.

PreviousErrCountPort3

Ce paramètre compte les occurrences d'erreurs détectées par le prédécesseur.

Le compteur de trames Type 12 erronées facultatif dénombre les trames, c'est-à-dire avec une structure de datagramme erronée. Le compteur est annulé si l'un des compteurs est écrit. Le comptage s'arrête lorsque la valeur maximale d'un compteur (255) est atteinte.

Paramètre

Wrong Type 12 frame counter

Ce paramètre compte les occurrences de trames Type 12 incorrectes.

Le compteur de problème local facultatif dénombre les occurrences de problèmes locaux. Le compteur est annulé s'il est écrit. Le comptage s'arrête lorsque la valeur maximale d'un compteur (255) est atteinte.

Paramètre

Local problem counter

Ce paramètre compte les occurrences de problèmes de communication à l'intérieur d'un esclave.

Les types d'attribut du compteur supplémentaire sont décrits à la Figure 16.

```
typedef struct
{
  Unsigned8          PreviousErrCountPort0;
  Unsigned8          PreviousErrCountPort1;
  Unsigned8          PreviousErrCountPort2;
  Unsigned8          PreviousErrCountPort3;
  Unsigned8          MalformatErrorCount;
  Unsigned8          LocalProblemCount;
} ADDCOUNTER;
```

Figure 16 – Description du type de compteur supplémentaire

Le codage du compteur supplémentaire est spécifié dans le Tableau 42.

Tableau 42 – Compteur supplémentaire

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
PreviousErrCountPort0	0x0308	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs
PreviousErrCountPort1	0x0309	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs
PreviousErrCountPort2	0x030A	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs
PreviousErrCountPort3	0x030B	unsigned8	RW	R	Une écriture dans un compteur réinitialisera tous les compteurs
MalformatErrorCount	0x030C	unsigned8	RW	R	Une écriture dans ce compteur réinitialisera ce compteur
LocalProblemCount	0x030D	unsigned8	RW	R	Une écriture dans ce compteur réinitialisera ce compteur

6.3 Chiens de garde

6.3.1 Diviseur de chien de garde

L'horloge système du contrôleur d'esclave est séparée par le diviseur du chien de garde.

Paramètre

WatchdogDivider

Ce paramètre doit contenir le nombre d'intervalles de 40 ns (moins 2) représentant l'incrément de base du chien de garde. (la valeur par défaut est 100 µs = 2 498).

Le type d'attribut du diviseur du chien de garde est décrit à la Figure 17.

```
typedef struct
{
  WORD          WatchdogDivider;
} TWATCHDOGDIVIDER;
```

Figure 17 – Description du type de diviseur du chien de garde

Le codage du diviseur du chien de garde est spécifié dans le Tableau 43.

Tableau 43 – Diviseur du chien de garde

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
WatchdogDivider	0x0400	WORD	RW	R	Intervalles de 40 ns utilisés pour d'autres temporisateurs de chien de garde

6.3.2 Chien de garde de l'utilisateur de DLS

L'utilisateur de DL est surveillé avec la valeur du chien de garde de l'utilisateur de DL. Chaque accès de l'utilisateur de DL au contrôleur d'esclave doit réinitialiser ce chien de garde.

Paramètre

DLSuserWatchdog

Ce paramètre doit contenir le chien de garde afin de surveiller l'utilisateur de DL (la valeur par défaut 1000 avec le diviseur du chien de garde 100 µs indique un chien de garde de 100 ms).

Le type d'attribut du chien de garde de l'utilisateur de DLS est décrit à la Figure 18.

```
typedef struct
{
  WORD          DLSuserWatchdog;
} TDLUSERWATCHDOG;
```

Figure 18 – Description du type de diviseur du chien de garde de l'utilisateur de DLS

Le codage du chien de garde de l'utilisateur de DLS est spécifié dans le Tableau 44.

Tableau 44 – Chien de garde de l'utilisateur de DLS

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
DLSuserWatchdog	0x0410	WORD	RW	R	

6.3.3 Chien de garde du gestionnaire de synchronisation

Chaque entité de gestionnaire de synchronisation est contrôlée par la valeur du chien de garde du gestionnaire de synchronisation. Chaque accès en écriture à la zone de mémoire de l'utilisateur de DL configurée dans le gestionnaire de synchronisation doit réinitialiser ce chien de garde si l'option correspondante est activée par ce gestionnaire de synchronisation.

Paramètre

Sync manager watchdog

Ce paramètre doit contenir le chien de garde surveillant le gestionnaire de synchronisation.

Le type d'attribut du chien de garde du gestionnaire de synchronisation est décrit à la Figure 19.

```
typedef struct
{
  WORD          SyncManChannelWatchdog;
} TSYNCMANCHANNELWATCHDOG;
```

Figure 19 – Description du type du chien de garde du gestionnaire de synchronisation

Le codage du chien de garde du gestionnaire de synchronisation est spécifié dans le Tableau 45.

Tableau 45 – Chien de garde du canal du gestionnaire de synchronisation

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Sync manager watchdog	0x0420	WORD	RW	R	

6.3.4 État du chien de garde du gestionnaire de synchronisation

L'état de chaque chien de garde du gestionnaire de synchronisation est inclus dans l'état correspondant.

Paramètre

Sync manager watchdog status

Ce paramètre doit contenir l'état de chien de garde de tous les chiens de garde du gestionnaire de synchronisation.

Les types d'attribut de l'état du chien de garde du gestionnaire de synchronisation sont décrits à la Figure 20.

```
typedef struct
{
  unsigned      SyncManChannelWdStatus:      1;
  unsigned      Reserved:                   15;
} TSYNCMANCHANNELWDSTATUS;
```

Figure 20 – Description du type de l'état du chien de garde du gestionnaire de synchronisation

Le codage de l'état du chien de garde du gestionnaire de synchronisation est spécifié dans le Tableau 46.

Tableau 46 – État de chien de garde du gestionnaire de synchronisation

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Sync manager channel watchdog status	0x0440	Unsigned1	R	R	Il n'y a qu'un seul chien de garde pour tous les gestionnaires de synchronisation 0: Le chien de garde a expiré 1: Chien de garde actif ou non activé
Reserved (réservé)	0x0440	Unsigned15	R	R	

6.3.5 Compteur de chien de garde

L'expiration du chien de garde est décomptée dans ce paramètre facultatif.

Paramètre

Sync manager watchdog counter

Ce paramètre décompte l'expiration de tous les chiens de garde du gestionnaire de synchronisation.

DL-user watchdog counter

Ce paramètre décompte l'expiration des chiens de garde de l'utilisateur de DL.

Les types d'attribut du compteur de chien de garde sont décrits à la Figure 21.

```
typedef struct
{
  Unsigned8          SyncMWDCounter;
  Unsigned8          PDIWDCounter;
} WDCOUNTER;
```

Figure 21 – Description du type de compteur du chien de garde

Le codage du compteur de chien de garde est spécifié dans le Tableau 47.

Tableau 47 – Compteur du chien de garde

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Sync manager WD count	0x0442	unsigned8	RW	R	Une écriture réinitialisera compteurs de chien de garde
PDI WD count	0x0443	unsigned8	RW	R	Une écriture réinitialisera compteurs de chien de garde

6.4 Interface d'informations de l'esclave

6.4.1 Zone de l'interface d'informations de l'esclave

Le codage de la zone de l'interface d'informations de l'esclave est spécifique à l'utilisateur DLS.

6.4.2 Accès à l'interface d'informations de l'esclave

Les types d'attribut de l'accès à l'interface d'informations de l'esclave sont décrits à la Figure 22.

```
typedef struct
{
  unsigned      Owner:          1;
  unsigned      Lock:           1;
  unsigned      Reserved1:      6;
  unsigned      AccPDI:         1;
  unsigned      Reserved2:      7;
} TSIIACCESS;
```

Figure 22 – Description du type d'accès à l'interface d'informations de l'esclave

Le codage d'accès à l'interface d'informations de l'esclave est spécifié dans le Tableau 48.

Tableau 48 – Accès à l'interface d'informations de l'esclave

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Owner	0x0500	Unsigned1	RW	R	0: DL Type 12 1: PDI
Lock	0x0500	Unsigned1	RW	R	Réinitialiser l'accès à SII 0: aucune action 1: annuler l'accès L'attribution d'une valeur à ce bit réinitialise le registre 501.0
Reserved (réservé)	0x0500	Unsigned6	RW	R	
Access PDI	0x0501	Unsigned1	R	RW	0: aucun accès 1: accès PDI actif
Reserved (réservé)	0x0501	Unsigned7	R	RW	

6.4.3 Contrôle/état de l'interface d'informations de l'esclave

Le registre de contrôle/d'état de l'interface d'informations de l'esclave permet de contrôler les opérations de lecture ou d'écriture dans cette interface.

Paramètre

Slave information interface assign

Ce paramètre doit contenir les informations relatives à l'attribution de l'interface à la liaison de données ou à l'utilisateur de DL.

Reset slave information interface access

Ce paramètre réinitialise l'accès à l'interface d'informations de l'esclave.

Slave information interface access

Ce paramètre doit contenir les informations relatives à l'activité de l'interface d'informations de l'esclave.

Slave information interface read size

Ce paramètre doit contenir les informations relatives au nombre d'octets (4 ou 8) qui peuvent être lus avec une commande.

Slave information interface write access

Ce paramètre doit contenir des informations, si un accès en écriture à l'interface d'informations de l'esclave est autorisé.

Slave information interface address algorithm

Ce paramètre doit contenir des informations, si le protocole de l'interface d'informations de l'esclave contient un ou deux octets d'adresse.

ReadOperation

Ce paramètre sera écrit depuis le maître afin de lancer l'opération de lecture de 32 bits/64 bits dans l'interface d'informations de l'esclave. Ce paramètre sera lu depuis le maître afin de vérifier si l'opération de lecture est terminée.

WriteOperation

Ce paramètre sera écrit depuis le maître afin de lancer l'opération d'écriture de 16 bits dans l'interface d'informations de l'esclave. Ce paramètre sera lu depuis le maître afin de vérifier si l'opération d'écriture est terminée. L'opération d'écriture ne fait l'objet d'aucune garantie de cohérence. Une interruption pendant l'écriture peut générer des valeurs incohérentes et il convient de l'éviter.

ReloadOperation

Ce paramètre sera écrit depuis le maître afin de lancer l'opération de recharge des premiers 128 bits dans l'interface d'informations de l'esclave. Ce paramètre sera lu depuis le maître afin de vérifier si l'opération de recharge est terminée.

SII error

Ce paramètre doit contenir des informations si l'accès en lecture du paramètre SII au démarrage n'a pas abouti.

Error command

Ce paramètre doit contenir des informations si le dernier accès à l'interface d'informations de l'esclave a abouti.

Busy

Ce paramètre contient des informations si une opération d'accès est en cours.

Les types d'attribut du contrôle/ de l'état de l'interface d'informations de l'esclave sont décrits à la Figure 23.

```
typedef struct
{
  unsigned      WriteAccess:              1;
  unsigned      Reserved1:                4;
  unsigned      EEPROM_Emulation          1;
  unsigned      ReadSize:                 1;
  unsigned      AddressAlgorithm:         1;
  unsigned      ReadOperation:            1;
  unsigned      WriteOperation:           1;
  unsigned      ReloadOperation:          1;
  unsigned      CheckSErrDLu:             1;
  unsigned      DeviceInfoError:          1;
  unsigned      CommandError:             1;
  unsigned      WriteError:               1;
  unsigned      Busy:                     1;
} TSIICONTROL;
```

Figure 23 – Description du type de contrôle/d'état de l'interface d'informations de l'esclave

Le codage du contrôle/de l'état de l'interface d'informations de l'esclave est spécifié dans le Tableau 49.

Tableau 49 – Contrôle/état de l'interface d'informations de l'esclave

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
SII write access	0x0502	Unsigned1	RW	R	0x00: accès en lecture seule à SII 0x01: accès en lecture et écriture à SII
Reserved (réservé)	0x0502	Unsigned4	R	R	0x00
EEPROM emulation	0x0502	Unsigned1	R	R	0x00: Fonctionnement normal (interface I²C utilisée) 0x01: PDI émule EEPROM (interface I²C non utilisée)
SII Read Size	0x0502	Unsigned1	R	R	0x00: lecture de 4 octets avec une transaction 0x01: lecture de 8 octets avec une transaction
SII Address Algorithm	0x0502	Unsigned1	R	R	0x00: 1 octet utilisé en tant qu'adresse

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
					0x01: 2 octets utilisés en tant qu'adresse
ReadOperation	0x0503	Unsigned1	RW	RW	0x00: aucune opération de lecture demandée (écriture de paramètre) ou opération de lecture non occupée (lecture de paramètre) 0x01: opération de lecture demandée (écriture de paramètre) ou opération de lecture occupée (lecture de paramètre) Pour lancer une nouvelle opération de lecture, ce paramètre doit comporter un front positif
WriteOperation	0x0503	Unsigned1	RW	RW	0x00: aucune opération d'écriture demandée (écriture de paramètre) ou opération d'écriture non occupée (lecture de paramètre) 0x01: opération d'écriture demandée (écriture de paramètre) ou opération d'écriture occupée (lecture de paramètre) Pour lancer une nouvelle opération d'écriture, ce paramètre doit comporter un front positif
ReloadOperation	0x0503	Unsigned1	RW	RW	0x00: aucune opération de recharge demandée (écriture de paramètre) ou opération de recharge non occupée (lecture de paramètre) 0x01: opération de recharge demandée (écriture de paramètre) ou opération de recharge occupée (lecture de paramètre) Pour lancer une nouvelle opération de recharge, ce paramètre doit comporter un front positif
Checksum Error	0x0503	Unsigned1	R	R	0x00: aucune information relative à l'utilisateur de DL de chargement d'erreur de somme de contrôle au démarrage 0x01: erreur de somme de contrôle lors de la lecture au démarrage
DeviceInfoError	0x0503	Unsigned1	R	R	0x00: aucune erreur de lecture des informations de l'appareil au démarrage 0x01: erreur de lecture des informations de l'appareil
CommandError	0x0503	Unsigned1	R	R (W)	0x00: aucune erreur sur la dernière commande 0x01: erreur sur la dernière commande Écriture PDI uniquement en mode d'émulation SII

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
WriteError	0x0503	Unsigned1	R	R	0x00: aucune erreur sur la dernière opération d'écriture 0x01: erreur sur la dernière opération d'écriture
Busy	0x0503	Unsigned1	R	R	0x00: l'opération est terminée 0x01: l'opération est en cours

6.4.4 Adresse réelle de l'interface d'informations de l'esclave

Le registre de l'adresse réelle de l'interface d'informations de l'esclave contient l'adresse réelle de l'interface d'informations de l'esclave à laquelle l'opération de lecture ou d'écriture suivante accède (en écrivant dans le registre de contrôle/d'état de l'interface d'informations de l'esclave).

Paramètre

Address

Ce paramètre doit contenir l'adresse du mot de 16 bits à laquelle l'opération de lecture ou d'écriture suivante accède.

Le type d'attribut de l'adresse de l'interface d'informations de l'esclave est décrit à la Figure 24.

```
typedef struct
{
  DWORD          SIIAddress;
} TSIIADDRESS;
```

Figure 24 – Description du type de l'adresse d'interface d'informations de l'esclave

Le codage de l'adresse réelle de l'interface d'informations de l'esclave est spécifié dans le Tableau 50.

Tableau 50 – Adresse réelle de l'interface d'informations de l'esclave

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Address	0x0504	DWORD	RW	RW	Adresse du mot de 16 bits

6.4.5 Données réelles de l'interface d'informations de l'esclave

Le registre des données réelles de l'interface d'informations de l'esclave contient les données (16 bits) à écrire dans l'interface d'informations de l'esclave lors de l'opération d'écriture suivante ou les données de lecture (32 bits/64 bits) de la dernière opération de lecture.

Paramètre

Data

Le maître écrira ce paramètre avec les données (16 bits) à écrire dans l'interface d'informations de l'esclave lors de l'opération d'écriture suivante. Le maître recevra les dernières données de lecture (32 bits/64 bits) provenant de l'interface d'informations de l'esclave lors de la lecture de ce paramètre.

Le type d'attribut des données de l'interface d'informations de l'esclave est décrit à la Figure 25.

```
typedef struct
{
  DWORD          SIIData;
} TSIIDATA;
```

Figure 25 – Description du type des données d'interface d'informations de l'esclave

Le codage des données réelles de l'interface d'informations de l'esclave est spécifié dans le Tableau 51.

Tableau 51 – Données réelles de l'interface d'informations de l'esclave

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Data	0x0508	DWORD	RW	RW	Pour l'opération d'écriture, seuls les 16 bits inférieurs (0x508-0x509) sont utilisés

6.5 Interface indépendante du support (MII)

6.5.1 Contrôle/état MII

La gestion de l'interface MII contient un ensemble d'attributs facultatifs. Le registre de contrôle/d'état MII permet de contrôler l'opération de lecture ou d'écriture dans l'interface MII.

Paramètre

MII write access

Ce paramètre doit contenir des informations, si un accès en écriture à l'interface MII est autorisé. Il convient de toujours activer la lecture si la gestion de l'interface MII est prise en charge.

Address offset

Ce paramètre doit contenir des informations relatives au décalage entre le numéro de port et l'adresse MII.

ReadOperation

Ce paramètre est écrit depuis le maître afin de lancer l'opération de lecture de 16 bits dans l'interface MII. Ce paramètre sera lu depuis le maître afin de vérifier si l'opération de lecture est terminée.

WriteOperation

Ce paramètre est écrit depuis le maître afin de lancer l'opération d'écriture de 16 bits dans l'interface MII. Ce paramètre sera lu depuis le maître afin de vérifier si l'opération d'écriture est terminée. L'opération d'écriture ne fait l'objet d'aucune garantie de cohérence. Une interruption pendant l'écriture peut générer des valeurs incohérentes, et il convient d'éviter les opérations critiques d'omission.

Error command

Ce paramètre doit contenir des informations si le dernier accès à l'interface MII a abouti.

Busy

Ce paramètre doit contenir des informations si une opération d'accès est en cours.

Les types d'attribut de contrôle/d'état MII sont décrits à la Figure 26.

```
typedef struct
{
  unsigned      WriteAccess:              1;
  unsigned      Reserved1:                6;
  unsigned      PHYoffset:                1;
  unsigned      ReadOperation:            1;
  unsigned      WriteOperation:           1;
  unsigned      Reserved2:                4;
  unsigned      WriteError:               1;
  unsigned      Busy:                     1;
} TMIICONTROL;
```

Figure 26 – Description du type de contrôle/d'état MII

Le codage du contrôle/de l'état MII est spécifié dans le Tableau 52.

Tableau 52 – Contrôle/état MII

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
WriteAcess	0x0510	Unsigned1	RW	R	0x00: accès en lecture seule à l'interface MII 0x01: accès en lecture et écriture à l'interface MII
Access PDI	0x0510	Unsigned1	R	R	0x00: Seulement ECAT 0x01: Accès PDI possible
Link Detection via MII management interface	0x0510	Unsigned1	R	R	0x00: inactif 0x01: actif
PHYoffset	0x0510	Unsigned5	R	R	0x00 (valeur par défaut) décalage à ajouter à l'adresse MII Définie dans la configuration locale
ReadOperation	0x0511	Unsigned1	RW	R	0x00: aucune opération de lecture demandée (écriture de paramètre) ou opération de lecture non occupée (lecture de paramètre) 0x01: opération de lecture demandée (écriture de paramètre) ou opération de lecture occupée (lecture de paramètre) Pour lancer une nouvelle opération de lecture, ce paramètre doit comporter un front positif
WriteOperation	0x0511	Unsigned1	RW	R	0x00: aucune opération d'écriture demandée (écriture de paramètre) ou opération d'écriture non occupée (lecture de paramètre) 0x01: opération d'écriture demandée (écriture de paramètre) ou opération d'écriture occupée (lecture de paramètre) Pour lancer une nouvelle opération d'écriture, ce paramètre doit comporter un front positif
Reserved (réservé)	0x0511	Unsigned3	R	R	0x00

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
ReadError	0x0511	Unsigned1	R	R	0x00: aucune erreur sur la dernière opération de lecture 0x01: erreur sur la dernière opération de lecture
WriteError	0x0511	Unsigned1	R	R	0x00: aucune erreur sur la dernière opération d'écriture 0x01: erreur sur la dernière opération d'écriture
Busy	0x0511	Unsigned1	R	R	0x00: l'opération est terminée 0x01: l'opération est en cours

6.5.2 Adresse MII réelle

Le registre de l'adresse MII réelle contient l'adresse réelle du registre MII de l'esclave à laquelle l'opération de lecture ou d'écriture suivante accède (en écrivant dans le registre de contrôle/d'état de l'interface MII).

Paramètre

Address PHY

Ce paramètre doit contenir l'adresse de l'appareil de couche physique à laquelle l'opération de lecture ou d'écriture suivante accède.

Address PHY register

Ce paramètre doit contenir l'adresse du registre PHY à laquelle l'opération de lecture ou d'écriture suivante accède. Les registres PHY sont présentés à l'Article 22 de l'ISO/CEI 8802-3:2000.

Les types d'attribut de l'adresse MII sont décrits à la Figure 27.

```
typedef struct
{
  Byte          PHYAddress;
  Byte          RegAddress;
} TMIIADDRESS;
```

Figure 27 – Description du type d'adresse MII

Le codage de l'adresse réelle de l'interface MII est spécifié dans le Tableau 53.

Tableau 53 – Adresse MII réelle

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Address PHY	0x0512	Unsigned8	RW	RW	Adresse de l'appareil de couche physique (0-63)
RegAddress	0x0513	Unsigned8	RW	RW	Adresse des registres PHY

6.5.3 Données MII réelles

Le registre des données MII réelles contient les données (16 bits) à écrire dans l'interface MII lors de l'opération d'écriture suivante ou les données de lecture (16 bits) de la dernière opération de lecture.

Paramètre

Data

Le maître écrira ce paramètre avec les données à écrire dans l'interface MII lors de l'opération d'écriture suivante. Le maître recevra les dernières données de lecture provenant de l'interface MII lors de la lecture de ce paramètre.

Le type d'attribut des données MII est décrit à la Figure 28.

```
typedef struct
{
  Word          MIIData;
} TMIIDATA;
```

Figure 28 – Description du type de données MII

Le codage des données réelles de l'interface MII est spécifié dans le Tableau 54.

Tableau 54 – Données MII réelles

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Data	0x0514	Unsigned16	RW	RW	

6.5.4 Accès MII

Les registres d'accès MII facultatifs gèrent l'accès MII depuis ECAT et depuis PDI.

Paramètre

Access MII

Le contrôle de la gestion MII

Access State

Le registre reflète l'état d'accès en cours

Access Reset

Réinitialiser le registre d'état d'accès

Le type d'attribut de l'accès MII est décrit à la Figure 29.

```
typedef struct
{
  unsigned      MIIAccess:              1;
  unsigned      Reserved1:              7;
  unsigned      MIIAccessState:         1;
  unsigned      MIIAccessReset:         1;
  unsigned      Reserved2:              6;
} TMIIAccess;
```

Figure 29 – Description du type d'accès MII

Le codage d'accès à l'interface MII est spécifié dans le Tableau 55.

Tableau 55 – Accès MII

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
MII Access	0x0516	Unsigned1	RW	R	0: Le contrôle PDI est possible 1: pas de contrôle PDI
Reserved (réservé)	0x0516	Unsigned7	R	R	
Access State	0x0517	Unsigned1	R	RW	0: accès ECAT actif 1: accès PDI actif
Access Reset	0x0517	Unsigned1	RW	R	0: aucune action 1: réinitialiser 0x0517.0
Reserved (réservé)	0x0517	Unsigned6	R	R	

6.6 Unité de gestion de mémoire de bus de terrain (FMMU)

6.6.1 Généralités

L'unité de gestion de mémoire de bus de terrain (FMMU) permet de convertir des adresses logiques en adresses physiques au moyen d'une adresse interne. Ainsi, les FMMU permettent d'utiliser un adressage logique des segments de données qui couvrent plusieurs appareils esclaves: une DLPDU traite les données au sein de plusieurs appareils répartis arbitrairement. Les FMMU prennent en charge facultativement le mapping de bit. Une DLE peut contenir plusieurs entités FMMU. Chaque entité FMMU mappe un espace d'adresse logique de cohésion à un espace d'adresse physique de cohésion.

La FMMU est composée de 16 entités au maximum. Chaque entité décrit une translation de mémoire entre la mémoire logique du réseau de communication Type 12 et la mémoire physique de l'esclave.

La Figure 30 illustre un exemple de mapping de l'adresse logique 0x14711.3 à 0x14712.0 à l'octet de mémoire 0xF01.1 to 0xF01.6.

NOTE La représentation des valeurs de bit de la gauche (bit de poids faible) vers la droite (bit de poids fort) n'implique pas un schéma de mise en séquence sur la ligne de transmission,

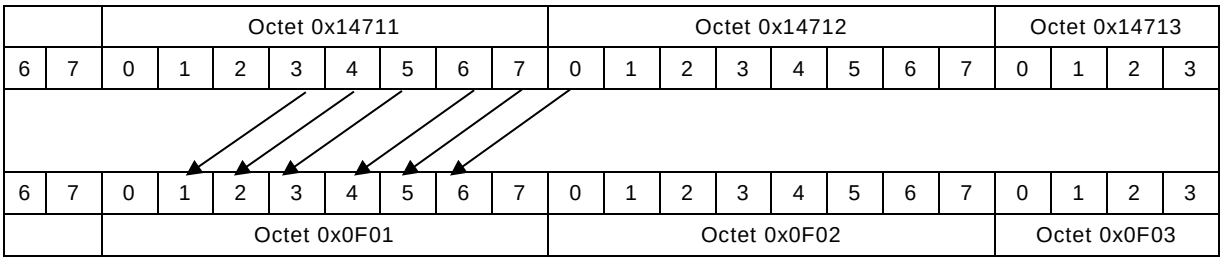


Figure 30 – Exemple de mapping FMMU

6.6.2 Attributs FMMU

Paramètre

LogicalStartAddress

Ce paramètre doit contenir l'adresse de début, en octets, dans la zone de mémoire logique de la translation de mémoire.

LogicalStartBit

Ce paramètre doit contenir le décalage de bit de l'adresse de début logique.

LogicalEndBit

Ce paramètre doit contenir le décalage de bit de l'adresse de fin logique.

PhysicalStartAddress

Ce paramètre doit contenir l'adresse de début, en octets, dans la zone de mémoire physique de la translation de mémoire.

PhysicalStartBit

Ce paramètre doit contenir le décalage de bit de l'adresse de début physique.

Length

Ce paramètre doit contenir la taille, en octets, de la translation de mémoire entre le premier et le dernier octet de l'espace d'adresse logique (la valeur 2 est attribuée à Length pour le mapping).

ReadEnable

Ce paramètre doit contenir des informations si une opération de lecture (la mémoire physique étant la source et la mémoire logique la destination) est activée.

WriteEnable

Ce paramètre doit contenir des informations si une opération d'écriture (la mémoire logique étant la source et la mémoire physique la destination) est activée.

Enable

Ce paramètre doit contenir des informations si la translation de mémoire est active ou pas.

Les types d'attribut de l'entité FMMU sont décrits à la Figure 31.

```
typedef struct
{
  DWORD         LogicalStartAddress;
  WORD          Length;
  unsigned      LogicalStartBit:         3;
  unsigned      Reserved1:               5;
  unsigned      LogicalEndBit:           3;
  unsigned      Reserved2:               5;
  WORD          PhysicalStartAddress;
  unsigned      PhysicalStartBit:        3;
  unsigned      Reserved3:               5;
  unsigned      ReadEnable:              1;
  unsigned      WriteEnable:             1;
  unsigned      Reserved4:               6;
  unsigned      Enable:                  1;
  unsigned      Reserved5:               7;
  unsigned      Reserved6:               8;
  WORD          Reserved7;
} TFMMU;
```

Figure 31 – Description du type d'entité FMMU

Une entité FMMU est spécifiée dans le Tableau 56. Le Tableau 57 présente la structure FMMU.

Tableau 56 – Entité Unité de gestion de mémoire du bus de terrain Entité (FMMU)

Paramètre	Adresse relative (décalage)	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
LogicalStartAddress	0x0000	DWORD	RW	R	
Length	0x0004	WORD	RW	R	
LogicalStartBit	0x0006	Unsigned3	RW	R	
reserved (réservé)	0x0006	Unsigned5	RW	R	0x00

Paramètre	Adresse relative (décalage)	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
LogicalEndBit	0x0007	Unsigned3	RW	R	
reserved (réservé)	0x0007	Unsigned5	RW	R	0x00
PhysicalStartAddress	0x0008	WORD	RW	R	
PhysicalStartBit	0x000A	Unsigned3	RW	R	
Reserved (réservé)	0x000A	Unsigned5	RW	R	0x00
ReadEnable	0x000B	Unsigned1	RW	R	0x00: l'entité sera ignorée pour le service de lecture 0x01: l'entité sera utilisée pour le service de lecture
WriteEnable	0x000B	Unsigned1	RW	R	0x00: l'entité sera ignorée pour le service d'écriture 0x01: l'entité sera utilisée pour le service d'écriture
Reserved (réservé)	0x000B	Unsigned6	RW	R	0x00
Enable	0x000C	Unsigned1	RW	R	0x00: entité inactive 0x01: entité active
Reserved (réservé)	0x000C	Unsigned7	RW	R	0x00
Reserved (réservé)	0x000D	Unsigned24	R	R	0x0000

Tableau 57 – Unité de gestion de mémoire de bus de terrain (FMMU)

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
FMMU entity 0	0x0600	TFMMU	RW	R	
FMMU entity 1	0x0610	TFMMU	RW	R	
FMMU entity 2	0x0620	TFMMU	RW	R	
FMMU entity 3	0x0630	TFMMU	RW	R	
FMMU entity 4	0x0640	TFMMU	RW	R	
FMMU entity 5	0x0650	TFMMU	RW	R	
FMMU entity 6	0x0660	TFMMU	RW	R	
FMMU entity 7	0x0670	TFMMU	RW	R	
FMMU entity 8	0x0680	TFMMU	RW	R	
FMMU entity 9	0x0690	TFMMU	RW	R	
FMMU entity 10	0x06A0	TFMMU	RW	R	
FMMU entity 11	0x06B0	TFMMU	RW	R	
FMMU entity 12	0x06C0	TFMMU	RW	R	
FMMU entity 13	0x06D0	TFMMU	RW	R	
FMMU entity 14	0x06E0	TFMMU	RW	R	
FMMU entity 15	0x06F0	TFMMU	RW	R	

6.7 Gestionnaire de synchronisation

6.7.1 Vue d'ensemble du gestionnaire de synchronisation

Le gestionnaire de synchronisation contrôle l'accès à la mémoire de l'utilisateur de DL. Chaque canal définit une zone cohérente de cette mémoire.

Le maître et le PDI échangent des données de deux manières:

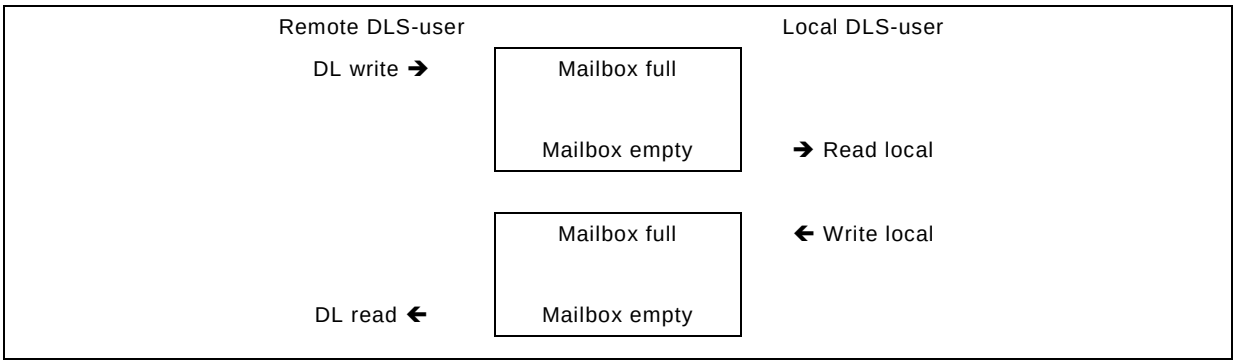
- Mode d'établissement de connexion (boîte aux lettres): une entité envoie les données et ne peut pas accéder à la zone tant que l'autre entité ne les a pas lues.
- Mode de mise en mémoire tampon: l'interaction entre le producteur et l'utilisateur des données n'est pas corrélée – chaque entité attend de pouvoir accéder à tout moment, en alimentant toujours l'utilisateur des données les plus récentes.

Le mode d'établissement de connexion est mis en œuvre avec une mémoire tampon: un fanion d'interruption ou d'état indique si une mémoire tampon est vide ou pleine.

La permutation d'une mémoire tampon est valide uniquement si la FCS des trames qui achemine la commande de lecture ou d'écriture est valide. Le principe de l'interaction est présenté à la Figure 32.

Les actions d'échange de mémoires tampons sont associées sur les premier et dernier octets:

- l'écriture de données sur le premier octet permet d'écrire dans la mémoire tampon, si elle est vide
- l'état de la mémoire tampon sera défini sur «plein» en écrivant le dernier octet de la mémoire tampon
- la lecture des données dans le premier octet permet de préparer la mémoire tampon à la lecture
- l'état de la mémoire tampon est défini sur «vide» en lisant le dernier octet de la mémoire tampon



Légende

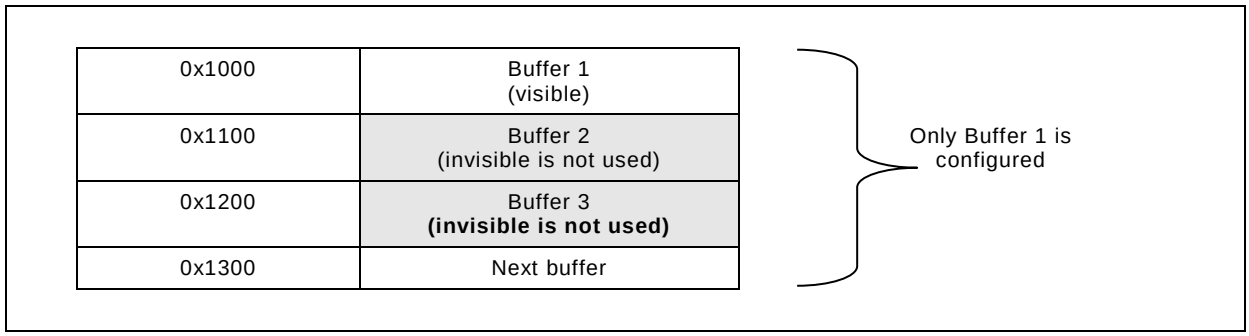
Anglais	Français
Remote DLS-user	Utilisateur DLS distant
Local DLS-user	Utilisateur DLS local
DL write	Écriture DL
DL read	Lecture DL
Mailbox full	Boîte aux lettres pleine
Mailbox empty	Boîte aux lettres vide
Write local	Écriture locale
Read local	Lecture locale

Figure 32 – Interaction de boîte aux lettres SyncM

Si une boîte aux lettres est pleine, il est impossible d'y écrire des données tant qu'elles n'ont pas été lues (c'est-à-dire que le dernier octet de la boîte aux lettres est lu). La durée de lecture des données n'a aucune importance (les couches supérieures peuvent faire l'objet de contraintes de temporisation).

Pour les données cycliques, un concept différent a été mis en œuvre pour assurer la cohérence et la disponibilité des données. Il s'agit d'un ensemble de mémoires tampons, permettant d'écrire et de lire des données simultanément sans interférence. Deux mémoires tampons sont allouées à l'émetteur et au destinataire; Une mémoire tampon de rechange fait office de magasin intermédiaire.

Cela signifie que dans ce mode, les mémoires tampons nécessitent d'être présentes en trois exemplaires. La Figure 33 illustre une configuration avec l'adresse de début 0x1000 et de longueur 0x100. Les autres mémoires tampons sont virtuellement non disponibles. L'accès a toujours lieu avec des adresses situées dans la plage de la mémoire tampon 1. La lecture ou l'écriture du dernier octet donne lieu à un échange automatique de mémoire tampon (du côté DL uniquement si la trame contenant les données de mémoire tampon est correctement reçue).

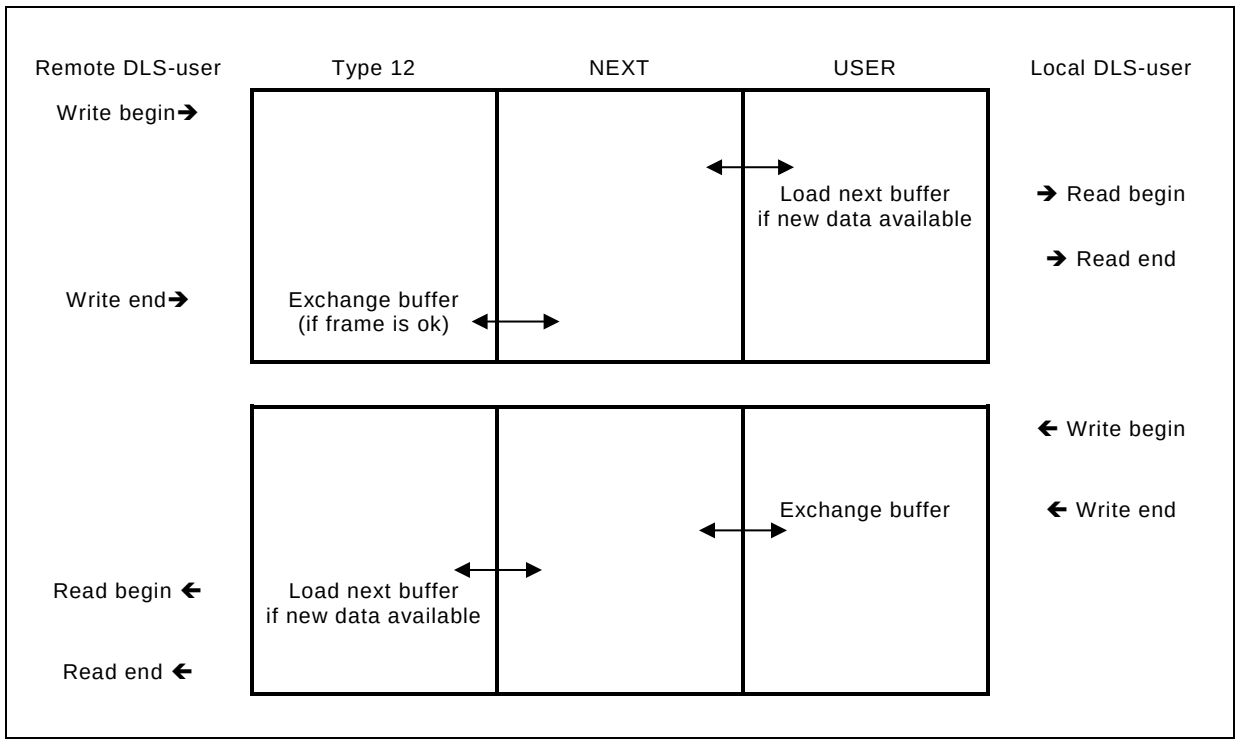


Légende

Anglais	Français
Buffer 1 (visible)	Mémoire tampon 1 (visible)
Buffer 2 (invisible is not used)	Mémoire tampon 2 (invisible si inutilisée)
Buffer 3 (invisible is not used)	Mémoire tampon 3 (invisible si inutilisée)
Next buffer	Mémoire tampon suivante
Only Buffer 1 is configured	Seule la mémoire tampon 1 est configurée

Figure 33 – Allocation de mémoire tampon SyncM

La Figure 34 présente le principe d'interaction avec mémoire tampon du gestionnaire de synchronisation.



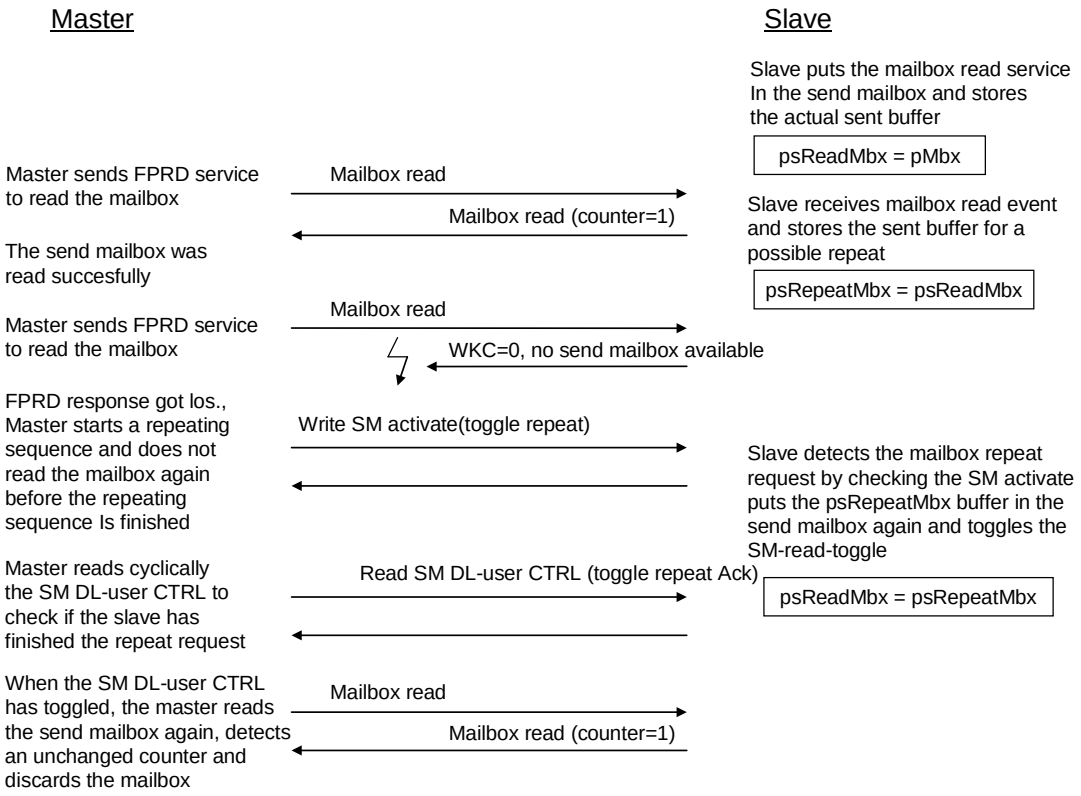
Légende

Anglais	Français
Remote DLS-user	Utilisateur DLS distant
Write begin	Début de l'écriture
Write end	Fin de l'écriture
Read begin	Début de la lecture
Read end	Fin de la lecture
Type 12	Type 12
NEXT	Suivant
USER	Utilisateur
Local DLS-user	Utilisateur DLS local
Exchange buffer (if frame is ok)	Échange de mémoire tampon (si la trame est ok)
Load next buffer if new data available	Charger la mémoire tampon suivante si les nouvelles données sont disponibles
Exchange buffer	Échange de mémoire tampon

Figure 34 – Interaction de mémoire tampon SyncM

Ce schéma permet d'accéder à une mémoire tampon, quelles que soient les fréquences de lecture et d'écriture. Par conséquent, l'esclave peut être mis en œuvre quelle que soit la vitesse du maître.

La Figure 35 présente un exemple d'interaction avec une erreur de boîte aux lettres en lecture.



Légende

Anglais	Français
Master	Maître
Slave	Esclave
Master sends FPRD service to read the mailbox	Le maître envoie le service FPRD pour lire la boîte aux lettres
The send mailbox was read successfully	La lecture de la boîte aux lettres d'envoi a abouti
FPRD response got lost. Master starts a repeating sequence and does not read the mailbox again before the repeating sequence is finished	La réponse FPRD a été perdue. Le maître démarre une séquence de répétition et ne relit pas la boîte aux lettres avant la fin de la séquence de répétition
Master reads cyclically the SM DL-user CTRL to check if the slave has finished the repeat request	Le maître lit le CTRL de l'utilisateur de DL SM de manière cyclique pour vérifier si l'esclave a terminé la demande de répétition
When the SM DL-user CTRL has toggled, the master reads the send mailbox again, detects an unchanged counter and discards the mailbox	Lors du basculement du CTRL de l'utilisateur de DL SM, le maître relit la boîte aux lettres d'envoi, détecte un compteur non modifié et ignore la boîte aux lettres
Slave puts the mailbox read service in the send mailbox and stores the actual sent buffer	L'esclave place le service de lecture de boîte aux lettres dans la boîte aux lettres d'envoi et stocke la mémoire tampon réelle envoyée
Slave receives mailbox read event and stores the sent buffer for a possible repeat	L'esclave reçoit l'événement de lecture de boîte aux lettres et stocke la mémoire tampon envoyée pour une éventuelle répétition
Slave detects the mailbox repeat request by checking the SM activate puts the psRepeatMbx buffer in the send mailbox again and toggles the SM-read-Toggle	L'esclave détecte la demande de répétition de boîte aux lettres en vérifiant l'activation SM, place de nouveau la mémoire tampon psRepeatMbx dans la boîte aux lettres d'envoi et bascule le SM-read-Toggle
Mailbox read	Lecture de boîte aux lettres
Write SM activate (toggle repeat)	Écriture de l'activation SM (répétition de basculement)

Anglais	Français
no send mailbox available	aucune boîte aux lettres d'envoi disponible
Read SM DI-user CTRL	Lecture du CTRL de l'utilisateur de DL SM

Figure 35 – Traitement du basculement écriture/lecture avec la boîte aux lettres en lecture

Les bits de basculement permettent de resynchroniser la communication de boîte aux lettres en cas de perte d'une DLPDU Type 12, c'est-à-dire qu'une entrée de boîte aux lettres en lecture perdue est de nouveau chargée.

6.7.2 Attributs du gestionnaire de synchronisation

Paramètre

PhysicalStartAddress

Ce paramètre doit contenir l'adresse de début, en octets, dans la mémoire physique de la zone de mémoire cohérente de l'utilisateur de DL.

Length

Ce paramètre doit contenir la taille, en octets, de la zone de mémoire cohérente de l'utilisateur de DL.

OperationMode

Ce paramètre doit contenir des informations si la zone de mémoire cohérente de l'utilisateur de DL est un type d'accès à la boîte aux lettres ou un type d'accès en mémoire tampon.

Direction

Ce paramètre doit contenir des informations si la zone de mémoire cohérente de l'utilisateur de DL est lue ou écrite par le maître.

EcatEventEnable

Ce paramètre doit contenir des informations si un événement est généré lorsque le maître a écrit (sens d'écriture) ou lu (sens de lecture) de nouvelles données disponibles dans la zone de mémoire cohérente de l'utilisateur de DL.

DLS-userEventEnable

Ce paramètre doit contenir des informations si un événement est généré lorsque de nouvelles données disponibles dans la zone de mémoire cohérente de l'utilisateur de DL étant écrites par l'utilisateur DLS ou si les nouvelles données depuis le maître ont été lues par l'utilisateur DLS.

WatchdogTriggerEnable

Ce paramètre facultatif doit contenir des informations si la surveillance d'un accès dans la zone de mémoire cohérente de l'utilisateur de DL est activée.

WriteEvent

Ce paramètre doit contenir des informations si le maître a écrit (sens d'écriture) dans la mémoire cohérente de l'utilisateur de DL et qu'une valeur a été attribuée au paramètre EventEnable.

ReadEvent

Ce paramètre doit contenir des informations si le maître a lu (sens de lecture) dans la mémoire cohérente de l'utilisateur de DL et qu'une valeur a été attribuée au paramètre EventEnable.

MailboxAccessTypeState

Ce paramètre doit contenir l'état (lecture/écriture de la mémoire tampon) de la mémoire cohérente de l'utilisateur de DL si son état permet d'accéder à la boîte aux lettres.

BufferedAccessTypeState

Ce paramètre facultatif doit contenir l'état (numéro de mémoire tampon, verrouillée) de la mémoire cohérente de l'utilisateur de DL s'il s'agit d'un type d'accès en mémoire tampon.

ReadBufferState

Ce paramètre facultatif indique l'état en cours de la mémoire tampon de lecture

WriteBufferState

Ce paramètre facultatif indique l'état en cours de la mémoire tampon d'écriture

ChannelEnable

Ce paramètre doit contenir des informations si le canal du gestionnaire de synchronisation est actif.

Repeat

Une modification de ce paramètre signale une demande de répétition. Il s'agit principalement de répéter les dernières interactions de boîte aux lettres.

DC Event 0 with Type 12 write

Ce paramètre facultatif doit contenir des informations si DC 0 Event doit être appelé en cas d'écriture de Type 12.

DC Event 0 with local write

Ce paramètre facultatif doit contenir des informations si DC 0 Event doit être appelé en cas d'écriture locale.

ChannelEnablePDI

Ce paramètre doit contenir des informations si le canal du gestionnaire de synchronisation est actif.

RepeatAck

Une modification de ce paramètre signale une demande d'acquittement. Une fois attribuée la valeur du paramètre Repeat, répéter l'acquittement.

Les types d'attribut d'un canal de gestionnaire de synchronisation sont décrits à la Figure 36.

```
typedef struct
{
  WORD          PhysicalStartAddress;
  WORD          Length;
  unsigned      OperationMode:          2;
  unsigned      Direction:              2;
  unsigned      EcatEventEnable:        1;
  unsigned      DLSuserEventEnable:     1;
  unsigned      WatchdogEnable:         1;
  unsigned      Reserved2:              1;
  unsigned      WriteEvent:             1;
  unsigned      ReadEvent:              1;
  unsigned      Reserved3:              1;
  unsigned      mailboxState:           1;
  unsigned      bufferState:            2;
  unsigned      ReadBufferState:        1;
  unsigned      WriteBufferState:       1;
  unsigned      ChannelEnable:          1;
  unsigned      Repeat:                 1;
  unsigned      Reserved5:              4;
  unsigned      DCEvent0wBusw:          1;
  unsigned      DCEvent0wlocw:          1;
  unsigned      ChannelEnablePDI:       1;
  unsigned      RepeatAck:              1;
  unsigned      Reserved6:              6;
} TSYNCMAN;
```

Figure 36 – Description du type de canal du gestionnaire de synchronisation

Un canal du gestionnaire de synchronisation est spécifié dans le Tableau 58.
Le Tableau 59 illustre la structure du gestionnaire de synchronisation.

Tableau 58 – Canal du gestionnaire de synchronisation

Paramètre	Adresse relative (décalage)	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
PhysicalStartAddress	0x0000	WORD	RW	R	
Length	0x0002	WORD	RW	R	
Buffer type	0x0004	Unsigned2	RW	R	0x00: mis en mémoire tampon 0x02: boîte aux lettres
Direction	0x0004	Unsigned2	RW	R	0x00: la zone doit être lue depuis le maître 0x01: la zone doit être écrite par le maître
ECATEventEnable	0x0004	Unsigned1	RW	R	0x00: l'événement n'est pas actif 0x01: l'événement est actif
DLSUserEventEnable	0x0004	Unsigned1	RW	R	0x00: l'événement de l'utilisateur DLS n'est pas actif 0x01: l'événement de l'utilisateur DLS est actif
WatchdogEnable	0x0004	Unsigned1	RW	R	0x00: chien de garde désactivé 0x01: chien de garde activé
Reserved (réservé)	0x0004	Unsigned1	RW	R	0x00
WriteEvent	0x0005	Unsigned1	R	R	0x00: aucun événement d'écriture 0x01: événement d'écriture

Paramètre	Adresse relative (décalage)	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
ReadEvent	0x0005	Unsigned1	R	R	0x00: aucun événement de lecture 0x01: événement de lecture
Reserved (réservé)	0x0005	unsigned1	R	R	0x00
MailboxState	0x0005	Unsigned1	R	R	0x00: boîte aux lettres vide 0x01: boîte aux lettres pleine
Buffered state	0x0005	Unsigned2	R	R	0x00: première mémoire tampon 0x01: deuxième mémoire tampon 0x02: troisième mémoire tampon 0x03: mémoire tampon verrouillée
ReadBufferState	0x0005	Unsigned1	R	R	0x00: la mémoire tampon de lecture n'est pas ouverte 0x01: la mémoire tampon de lecture est ouverte
WriteBufferState	0x0005	Unsigned1	R	R	0x00: la mémoire tampon d'écriture n'est pas ouverte 0x01: la mémoire tampon d'écriture est ouverte
ChannelEnable	0x0006	Unsigned1	RW	R	0x00: canal désactivé 0x01: canal activé
Repeat	0x0006	Unsigned1	RW	R	
Reserved (réservé)	0x0006	Unsigned4	RW	R	0x00
DC Event 0 with Bus access	0x0006	Unsigned1	RW	R	0x00: aucun événement 0x01: événement DC si le maître accède à la mémoire tampon
DC Event 0 with local access	0x0006	Unsigned1	RW	R	0x00: aucun événement 0x01: événement DC si l'utilisateur de DL accède à la mémoire tampon
ChannelEnablePDI	0x0007	Unsigned1	R	RW	0x00: canal activé 0x01: canal désactivé
RepeatAck	0x0007	Unsigned1	R	RW	doit suivre la répétition après la récupération des données
Reserved (réservé)	0x0007	Unsigned6	R	RW	0x00

Tableau 59 – Structure du gestionnaire de synchronisation

Paramètre	Adresse physique	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Sync manager channel 0	0x0800	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 1	0x0808	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 2	0x0810	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 3	0x0818	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 4	0x0820	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 5	0x0828	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 6	0x0830	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 7	0x0838	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 8	0x0840	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 9	0x0848	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 10	0x0850	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 11	0x0858	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 12	0x0860	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 13	0x0868	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 14	0x0870	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture
Sync manager channel 15	0x0878	TSYNCMAN	RW	R	Dernier octet PDI accessible en écriture

Les canaux du gestionnaire de synchronisation doivent être utilisés de la manière suivante:

- Sync Manager channel 0: écriture dans la boîte aux lettres
- Sync Manager channel 1: lecture dans la boîte aux lettres
- Sync Manager channel 2: écriture des données de processus (il peut être utilisé pour la lecture des données de processus si l'écriture des données de processus n'est pas prise en charge)
- Sync Manager channel 3: lecture des données de processus

Si la boîte aux lettres n'est pas prise en charge, elle doit être utilisée de la manière suivante:

- Sync Manager channel 0: écriture des données de processus (il peut être utilisé pour la lecture des données de processus si l'écriture des données de processus n'est pas prise en charge)
- Sync Manager channel 1: lecture des données de processus

6.8 Horloge distribuée

6.8.1 Généralités

L'horloge distribuée (DC) est utilisée pour répondre à des exigences de temporisation très précises et pour utiliser des signaux de synchronisation qui peuvent être générés indépendamment du cycle de communication. Les systèmes ne faisant pas l'objet d'exigences aussi strictes en matière de synchronisation peuvent être synchronisés en partageant un service (LRW, LRD ou LWR, de préférence) ou en utilisant la même trame Ethernet pour accéder aux mémoires tampons.

6.8.2 Mesure du délai

La mesure du délai s'appuie sur des informations de datation liées à une seule trame. L'esclave fournit les moyens de datation, le calcul du délai revenant au maître.

Paramètre

ReceiveTimePort 0

Ce paramètre doit contenir l'heure de réception du début d'un datagramme particulier sur le port 0. Ce datagramme doit être un accès en écriture à ce paramètre. L'heure de réception de cette trame est écrite dans ce paramètre à la fin de ce datagramme si la réception a abouti. De plus, le verrou des registres du port 1, 2 et 3 de l'heure de réception est activé pour le même datagramme.

ReceiveTimePort1

Ce paramètre doit contenir l'heure de réception du début d'un datagramme particulier sur le port 1. Ce datagramme doit être un accès en écriture au registre du port 0 de l'heure de réception. L'heure de réception de cette trame est écrite dans ce paramètre à la fin de ce datagramme si la réception a abouti.

ReceiveTimePort2

Ce paramètre doit contenir l'heure de réception du début d'un datagramme particulier sur le port 2. Ce datagramme doit être un accès en écriture au registre du port 0 de l'heure de réception. L'heure de réception de cette trame sera écrite dans ce paramètre à la fin de ce datagramme si la réception a abouti.

ReceiveTimePort3

Ce paramètre doit contenir l'heure de réception du début d'un datagramme particulier sur le port 3. Ce datagramme doit être un accès en écriture au registre du port 0 de l'heure de réception. L'heure de réception de cette trame sera écrite dans ce paramètre à la fin de ce datagramme si la réception a abouti.

6.8.3 Paramètre de temps local

Le paramètre de temps local contient le temps système local et le paramètre de la boucle de contrôle, qui visent à mettre en œuvre une boucle de contrôle pour la coordination du temps du système local avec un temps universel.

Paramètre

LocalSystemTime

Ce paramètre doit contenir le temps du système local verrouillé lors de la réception d'un datagramme. Un accès en écriture à ce paramètre doit lancer une comparaison du temps du système local verrouillé avec le temps système de référence écrit. Le résultat de cette comparaison doit être une entrée de la boucle à verrouillage de phase (PLL) pour le temps du système local.

SystemTimeOffset

Ce paramètre doit contenir le décalage entre le temps du système local et le temps universel.

SystemTimeTransmissionDelay

Ce paramètre doit contenir le délai de transmission du contrôleur d'esclave comportant le temps du système de référence et le contrôleur d'esclave local.

SystemTimeDifference

Ce paramètre doit contenir le résultat de la dernière comparaison entre le temps du système local et le temps de la dernière écriture moins le décalage du temps du système, moins le délai de transmission du temps du système.

Paramètres de la boucle de contrôle

Ces paramètres spécifiques à la mise en œuvre doivent contenir les paramètres de configuration pour la boucle de contrôle du temps système local.

6.8.4 Paramètre de temps de l'utilisateur de DL

Le paramètre de temps de l'utilisateur de DL contient l'utilisateur DC P1 à P12 du paramètre de temps local pour l'utilisateur de DL.

NOTE La signification des paramètres n'est pas définie dans le cadre de ce domaine d'application. Les droits d'accès sont spécifiés dans la CEI 61158-5-12.

6.8.5 Attributs DC

Les types d'attribut de mesure du délai de l'horloge distribuée sont inclus dans le paramètre de temps local décrit à la Figure 37.

```
typedef struct
{
  DWORD         ReceiveTimePort0;
  DWORD         ReceiveTimePort1;
  DWORD         ReceiveTimePort2;
  DWORD         ReceiveTimePort3;
  UINT64        LocalSystemTime;
  BYTE          Reserved2[8];
  UINT64        SystemTimeOffset;
  DWORD         SystemTimeTransmissionDelay;
  DWORD         SystemTimeDifference;
  WORD          ControlLoopParameter1;
  WORD          ControlLoopParameter2;
  WORD          ControlLoopParameter3;
  BYTE          Reserved3[74];
} TDCTRANSMISSION;
```

Figure 37 – Description du type de paramètre de temps local de l'horloge distribuée

Le paramètre de temps local de l'horloge distribuée est spécifié dans le Tableau 60.

Tableau 60 – Paramètre de temps local de l'horloge distribuée

Paramètre	Adresse physique (décalage)	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
ReceiveTimePort0	0x0900	DWORD	R	R	Un accès en écriture verrouille le temps local (en ns) au début de la réception (premier élément du préambule) sur le port 0 de cette PDU de ce paramètre (si la PDU a été correctement reçue) et active le verrou des ports 1 à 3
ReceiveTimePort1	0x0904	DWORD	R	R	Temps local (en ns) au début de la réception sur le port 1 lorsqu'une PDU contenant un accès en écriture dans le registre du port 0 de l'heure de réception a été correctement reçue
ReceiveTimePort2	0x0908	DWORD	R	R	Temps local (en ns) au début de la réception sur le port 1 lorsqu'une PDU contenant un accès en écriture dans le registre du port 0 de l'heure de réception a été correctement reçue
ReceiveTimePort3	0x090C	DWORD	R	R	Temps local (en ns) au début de la réception sur le port 1 lorsqu'une PDU contenant un accès en écriture dans le registre du port 0 de l'heure de réception a été correctement reçue
SystemTime	0x0910	UINT64	RW	R	Un accès en écriture compare le temps système local verrouillé (en ns) au début de la réception par l'unité de traitement de cette PDU à la valeur écrite (les 32 bits inférieurs; si la PDU a été correctement reçue), le résultat sera l'entrée de la PLL de l'horloge distribuée
ReceiveTimeProcessingUnit	0x0918	UINT64	RW	R	Temps local (en ns) au début de la réception par l'unité de traitement d'une PDU contenant un accès en écriture au port 0 de l'heure de réception (si la PDU a été correctement reçue)

Paramètre	Adresse physique (décalage)	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
SystemTimeOffset	0x0920	UINT64	RW	R	Décalage entre le temps local (en ns) et le temps système local (en ns)
SystemTimeTransmissionDelay	0x0928	DWORD	RW	R	Décalage entre le temps système de référence (en ns) et le temps système local (en ns)
SystemTimeDifference	0x092C	DWORD	RW	R	Bit 30..0: Différence moyenne entre la copie locale du temps système et les valeurs du temps système reçues Bit 31: 0: Copie locale du temps système supérieure ou égale au temps système reçu 1: Copie locale du temps système inférieur au temps système reçu
Control Loop (Paramètre 1)	0x0930	WORD	R(W)	R(W)	Spécifique à la mise en œuvre
Control Loop (Paramètre 2)	0x0932	WORD	R	R	Spécifique à la mise en œuvre
Control Loop (Paramètre 3)	0x0934	WORD	R(W)	R(W)	Spécifique à la mise en œuvre

Le codage du paramètre d'horloge distribuée de l'utilisateur DLS est décrit dans le Tableau 61.

Tableau 61 – Paramètre d'horloge distribuée de l'utilisateur DLS

Paramètre	Adresse physique (décalage)	Type de données	Type d'accès	PDI du type d'accès	Valeur/description
Reserved (réservé)	0x0980	BYTE	RW	R	0
DC user P1	0x0981	BYTE	RW	R	Spécifique à la mise en œuvre
DC user P2	0x0982	Unsigned16	R	R	Spécifique à la mise en œuvre
DC user P13	0x0983	BYTE	R	R	Spécifique à la mise en œuvre
DC user P14	0x0984	BYTE	R	R	Spécifique à la mise en œuvre
Reserved (réservé)	0x0985	BYTE[8]	RW	R	
DC user P3	0x098E	Unsigned16	R	R	Spécifique à la mise en œuvre
DC user P4	0x0990	DWORD	RW	R	Spécifique à la mise en œuvre
Reserved (réservé)	0x0994	BYTE[12]	R	R	
DC user P5	0x09A0	DWORD	RW	R	Spécifique à la mise en œuvre
DC user P6	0x09A4	DWORD	RW	R	Spécifique à la mise en œuvre
DC user P7	0x09A8	Unsigned16	RW	R	Spécifique à la mise en œuvre
Reserved (réservé)	0x09AA	BYTE[4]	R	R	
DC user P8	0x09AE	Unsigned1	R	R	Spécifique à la mise en œuvre
DC user P9	0x09B0	DWORD	R	R	Spécifique à la mise en œuvre
Reserved (réservé)	0x09B4	BYTE[4]	R	R	
DC user P10	0x09B8	DWORD	R	R	Spécifique à la mise en œuvre
Reserved (réservé)	0x09BC	BYTE[4]	R	R	
DC user P11	0x09C0	DWORD	R	R	Spécifique à la mise en œuvre
Reserved (réservé)	0x09C4	BYTE[4]	R	R	
DC user P12	0x09C8	DWORD	R	R	Spécifique à la mise en œuvre
Reserved (réservé)	0x09CC	BYTE[4]	R	R	

7 Mémoire de l'utilisateur de DL

7.1 Vue d'ensemble

Après la réinitialisation, lorsque l'utilisateur DLS est opérationnel, la mémoire peut être utilisée en principe sans restriction depuis la communication et depuis l'utilisateur de DL local, une communication étant possible par cette zone. Cependant, il n'existe aucun traitement cohérent de données possible grâce à ce mécanisme.

Avec le gestionnaire de synchronisation, il est possible d'utiliser la zone de mémoire de manière coordonnée. Étant donné que le gestionnaire de synchronisation est établi par le maître, un esclave n'utilise pas cette zone qui est dédiée à la communication.

Les deux moyens coordonnés de communication suivants sont pris en charge.

- Un mode de mise en mémoire tampon, permettant d'assurer la cohérence de lecture et d'écriture dans les deux directions, trois zones de mémoire étant nécessaires à sa prise en charge. La vitesse de mise à jour locale et le cycle de communication peuvent être configurés de manière indépendante.
- Un mode de boîte aux lettres, avec une seule mémoire tampon permettant une communication verrouillée. Une entité (communication ou utilisateur de DL) place les données, et la zone de mémoire est verrouillée tant que l'autre entité ne les a pas lues.

7.2 Type d'accès à la boîte aux lettres

7.2.1 Transfert de boîte aux lettres

Les services de transfert de boîte aux lettres sont décrits du point de vue du maître eu égard à la direction (écriture et lecture s'entendent respectivement de l'écriture des données depuis le maître et de la lecture des données par le maître) et du point de vue de l'esclave eu égard à la description du service. L'interaction inclut une procédure d'établissement de liaison, le maître devant attendre une action de l'esclave suite à une demande de service, et inversement.

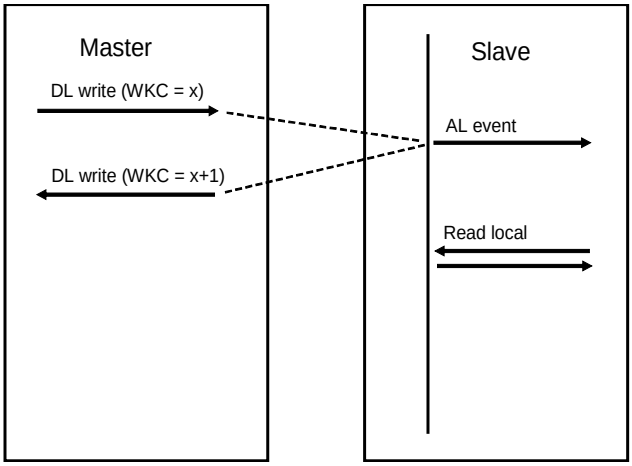
La couche de liaison de données spécifie des services résilients pour la lecture et l'écriture d'une salve de données.

Avec le service d'écriture, un maître (client) demande la modification d'une zone de mémoire de l'esclave. Un service d'écriture sera acquitté si l'esclave adressé est disponible et la boîte aux lettres en écriture est vide. Un numéro de séquence est prévu afin de détecter les doublons. Des écritures consécutives portant le même numéro de séquence ne seront indiquées qu'une seule fois.

Les données de mise à jour en lecture seront stockées tant que la lecture des données pour la mise à jour en lecture suivante n'est pas indiquée.

7.2.2 Accès en écriture depuis le maître

La Figure 38 illustre les primitives entre le maître, la DLL et l'utilisateur de DL en cas de réussite de la séquence d'écriture.



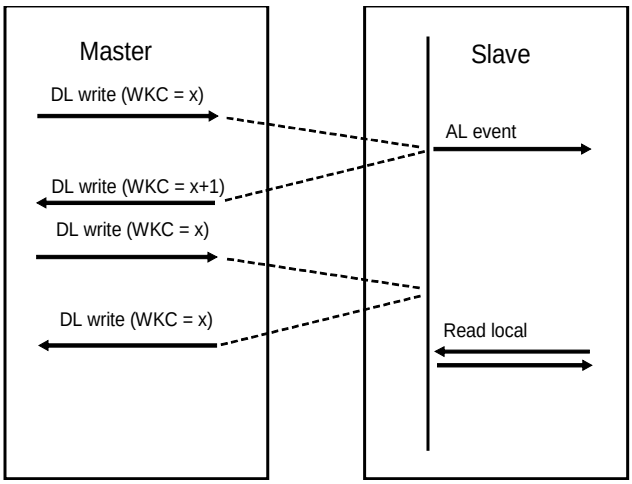
Légende

Anglais	Français
Master	Maître
Slave	Esclave
DL write	Écriture DL
AL event	Événement AL
Read local	Lecture locale

Figure 38 – Séquence d'écriture réussie dans la boîte aux lettres

Le maître envoie un service d'écriture avec le compteur en fonction (WKC = x), la couche de liaison de données (contrôleur d'esclave) de l'esclave écrit les données reçues dans la zone de mémoire de l'utilisateur de DL, incrémente le compteur en fonction (WKC = x + 1) et génère un événement. Le canal du gestionnaire de synchronisation correspondant verrouille la zone de mémoire de l'utilisateur de DL tant que l'utilisateur de DL ne l'a pas lue. Le maître reçoit une réponse d'écriture réussie, le compteur d'exploitation ayant été incrémenté. L'utilisateur de DL lit la zone de mémoire de l'utilisateur de DL, et le canal du gestionnaire de synchronisation correspondant déverrouille la zone de mémoire de l'utilisateur de DL de manière à permettre au maître d'y écrire de nouveau.

La Figure 39 illustre les primitives entre le maître, la DLL et l'utilisateur de DL en cas de séquence d'écriture erronée.



Légende

Anglais	Français
Master	Maître

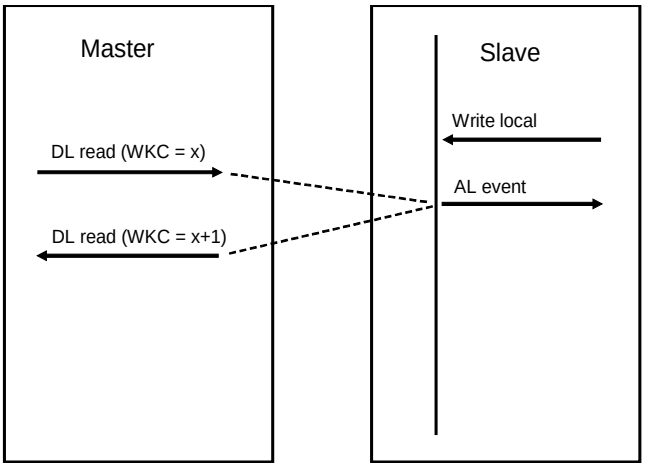
Anglais	Français
Slave	Esclave
DL write	Écriture DL
AL event	Événement AL
Read local	Lecture locale

Figure 39 – Séquence d'écriture erronée dans la boîte aux lettres

Le maître envoie un service d'écriture avec le compteur en fonction (WKC = x), la couche de liaison de données (contrôleur d'esclave) de l'esclave écrit les données reçues dans la zone de mémoire de l'utilisateur de DL, incrémente le compteur en fonction (WKC = x + 1) et génère un événement. Le canal du gestionnaire de synchronisation correspondant verrouille la zone de mémoire de l'utilisateur de DL tant que ce dernier ne l'a pas lue. Le maître reçoit une réponse d'écriture réussie, le compteur en fonction ayant été incrémenté. Avant que l'utilisateur de DL ne lise sa zone de mémoire, le maître écrit de nouveau dans la même zone avec le compteur en fonction (WKC = x). Étant donné que la zone de mémoire de l'utilisateur de DL est toujours verrouillée, la couche de liaison de données de l'esclave ignorera les données reçues et n'incrémentera pas le compteur en fonction. Le maître reçoit une réponse d'écriture erronée, le compteur en fonction n'ayant pas été incrémenté. Plus tard, l'utilisateur de DL lit sa zone de mémoire, que le canal du gestionnaire de synchronisation correspondant déverrouille de manière à permettre au maître d'y écrire de nouveau.

7.2.3 Accès en lecture depuis le maître

La Figure 40 illustre les primitives entre le maître, la DLL et l'utilisateur de DL en cas de réussite de la séquence de lecture.



Légende

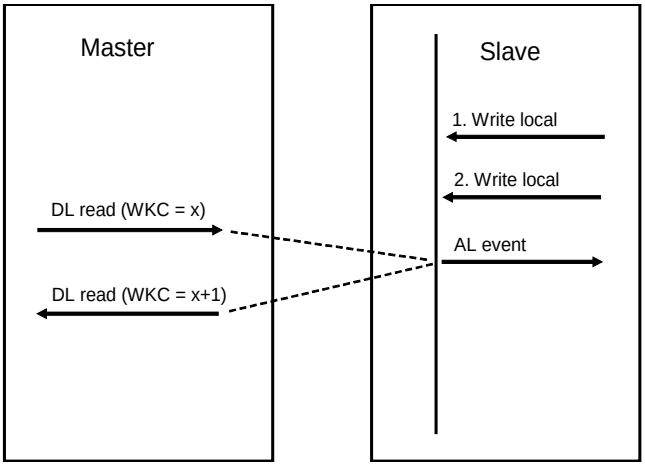
Anglais	Français
Master	Maître
Slave	Esclave
DL read	Lecture DL
AL event	Événement AL
Write local	Écriture locale

Figure 40 – Séquence de lecture réussie dans la boîte aux lettres

L'utilisateur de DL met à jour sa zone de mémoire. Le canal du gestionnaire de synchronisation correspondant verrouille la zone de mémoire de l'utilisateur de DL tant que le maître ne l'a pas lue. Le maître envoie un service de lecture avec le compteur en fonction (WKC = x), la couche de liaison de données (contrôleur d'esclave) de l'esclave envoie les données de la zone de mémoire de l'utilisateur de DL, incrémente le compteur en fonction

(WKC = x + 1) et génère un événement pour l'utilisateur de DL. Le maître reçoit une réponse de lecture réussie, le compteur en fonction ayant été incrémenté. Le canal du gestionnaire de synchronisation correspondant déverrouille la zone de mémoire de l'utilisateur de DL, de sorte que ce dernier soit en mesure d'y écrire de nouveau.

La Figure 41 illustre les primitives entre le maître, la DLL et l'utilisateur de DL en cas de séquence de lecture erronée.



Légende

Anglais	Français
Master	Maître
Slave	Esclave
DL read	Lecture DL
AL event	Événement AL
Write local	Écriture locale

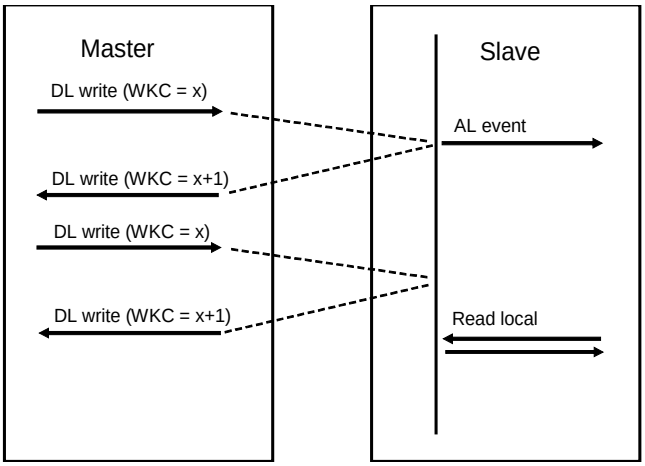
Figure 41 – Séquence de lecture erronée dans la boîte aux lettres

L'utilisateur de DL met à jour sa zone de mémoire (1. écriture locale). Le canal du gestionnaire de synchronisation correspondant verrouille la zone de mémoire de l'utilisateur de DL tant que le maître ne l'a pas lue. L'utilisateur de DL met de nouveau à jour sa zone de mémoire (2. écriture locale), mais elle sera ignorée par le canal du gestionnaire de synchronisation correspondant car le maître n'a pas lu les anciennes données. Lorsque le maître envoie une demande de lecture avec le compteur en fonction (WKC = x), la couche de liaison de données (contrôleur d'esclave) de l'esclave envoie les données de la zone de mémoire de l'utilisateur de DL, incrémente le compteur en fonction (WKC = x + 1) et génère un événement pour l'utilisateur de DL. Le maître reçoit une réponse de lecture réussie, le compteur en fonction ayant été incrémenté. Le canal du gestionnaire de synchronisation correspondant déverrouille la zone de mémoire de l'utilisateur de DL, de sorte que ce dernier soit en mesure d'y écrire de nouveau.

7.3 Type d'accès en mémoire tampon

7.3.1 Accès en écriture depuis le maître

La Figure 42 illustre les primitives entre le maître, la DLL et l'utilisateur de DL en cas d'une séquence d'écriture. L'exemple présente un maître rapide et un esclave au temps de réaction plus lent.



Légende

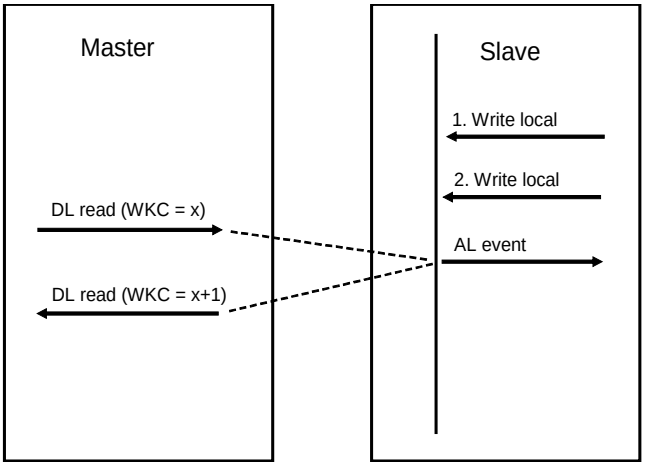
Anglais	Français
Master	Maître
Slave	Esclave
DL write	Écriture DL
AL event	Événement AL
Read local	Lecture locale

Figure 42 – Séquence d'écriture réussie dans la mémoire tampon

Le maître envoie une demande d'écriture avec le compteur en fonction (WKC = x), la couche de liaison de données (contrôleur d'esclave) de l'esclave écrit les données reçues dans la zone de mémoire de l'utilisateur de DL, incrémente le compteur en fonction (WKC = x + 1) et génère un événement pour l'utilisateur de DL. Le maître reçoit une réponse d'écriture réussie, le compteur en fonction ayant été incrémenté. Avant que l'utilisateur de DL ne lise sa zone de mémoire, le maître écrit de nouveau dans la même zone avec le compteur en fonction (WKC = x). La zone de mémoire de l'utilisateur du type d'accès en mémoire tampon n'étant jamais verrouillée, la couche de liaison de données de l'esclave écrase les données reçues dans la zone de mémoire de l'utilisateur de DL, incrémente le compteur en fonction (WKC = x + 1) et génère de nouveau un événement pour l'utilisateur de DL. Le maître reçoit une réponse d'écriture réussie, le compteur en fonction ayant été incrémenté. Plus tard, l'utilisateur de DL lit sa zone de mémoire.

7.3.2 Accès en lecture depuis le maître

La Figure 43 illustre les primitives entre le maître, la DLL et l'utilisateur de DL en cas de réussite de la séquence de lecture. L'esclave procède à plusieurs mises à jour des données avant que le maître ne les lise.



Légende

Anglais	Français
Master	Maître
Slave	Esclave
DL read	Lecture DL
AL event	Événement AL
Write local	Écriture locale

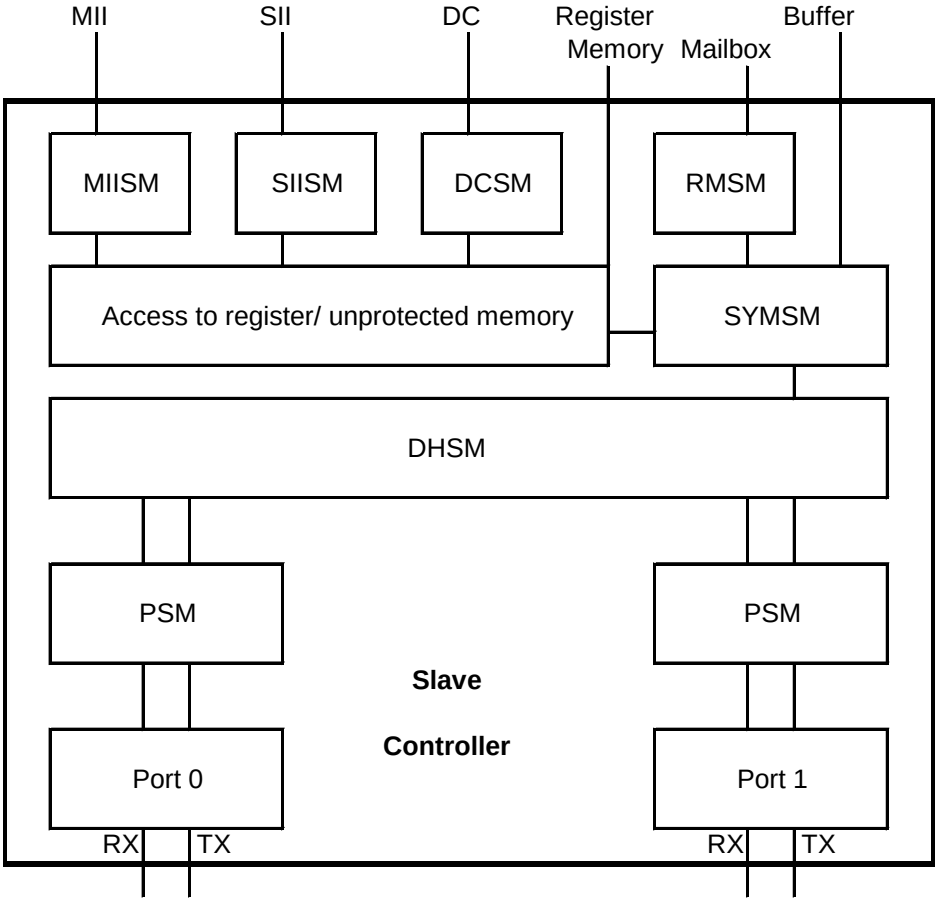
Figure 43 – Séquence de lecture réussie dans la mémoire tampon

L'utilisateur de DL met à jour sa zone de mémoire (1. écriture locale). Il met de nouveau à jour sa zone de mémoire avec de nouvelles valeurs (2. écriture locale), et comme les zones de mémoire de l'utilisateur de DL en mémoire tampon ne sont jamais verrouillées, le canal du gestionnaire de synchronisation correspondant écrase les anciennes données. Ensuite, le maître envoie un service de lecture avec le compteur en fonction (WKC = x), la couche de liaison de données (contrôleur d'esclave) de l'esclave envoie les données de la zone de mémoire de l'utilisateur de DL, incrémente le compteur en fonction (WKC = x + 1) et génère un événement pour l'utilisateur de DL. Le maître reçoit une réponse de lecture réussie, le compteur en fonction ayant été incrémenté.

8 Type 12: Diagrammes d'états du protocole FDL

8.1 Vue d'ensemble des diagrammes d'états DL esclaves

La Figure 44 illustre la structure générale de la DL d'un esclave en présentant ses diagrammes d'états et leurs interactions.



Légende

Anglais	Français
Register Memory	Mémoire du registre
Mailbox	Boîte aux lettres
Buffer	Mémoire tampon
Access to register/unprotected memory	Accès au registre/mémoire non protégée
Slave Controller	Contrôleur d'esclave

Figure 44 – Structure des diagrammes d'états de protocole d'un esclave

8.2 Description du diagramme d'états

8.2.1 Diagramme d'états de port (PSM)

Le PSM assure la coordination des diagrammes d'états de port sous-jacents utilisés pour le traitement des trames MAC et leur transfert octet par octet au gestionnaire PDU. Il existe un diagramme d'états par DL sur au moins deux interfaces DL d'un esclave nommé en fonction des ports. Il n'existe aucun diagramme d'états explicite pour les ports, puisqu'il suit les règles établies pour les ports de l'ISO/CEI 8802-3, à l'exception:

a) le message est transmis octet par octet, au lieu de transmettre l'ensemble de la trame au niveau de l'interface DL.

b) si un port ne dispose d'aucune liaison, une primitive Tx.req donnera lieu à une primitive Rx.ind (à condition que le port soit en mode automatique ou que la boucle soit fermée par une commande).

De plus, les compteurs statiques définis dans la CEI 61158-3-12 seront traités par PSM.

8.2.2 Diagramme d'états du gestionnaire PDU (DHSM)

Le DHSM traitera les trames Ethernet en les divisant en PDU Type 12 individuelles au niveau du port principal et en demande d'écriture Receive Time 0 au niveau du port secondaire, puis les mappe avec les registres individuels ou avec le diagramme d'états du gestionnaire de synchronisation (SYSM) ou le diagramme d'états DC (DCSM). La FMMU comme un mapping entre l'espace d'adresse global et l'adressage physique, l'activation du SIISM et du MIISM par l'accès au registre se trouvent également dans le DHSM. Le DHSM est présenté de manière plus détaillée à l'Article A.1.

8.2.3 Diagramme d'états du gestionnaire de synchronisation (SYSM)

Le diagramme d'états du gestionnaire de synchronisation traitera les zones de mémoire couvertes par SynchM (les boîtes aux lettres et les mémoires tampons, par exemple). Les services de boîte aux lettres sont transférés vers un diagramme d'états qui traite les nouvelles tentatives (le diagramme d'états de boîte aux lettres résilient – RMSM). Il existe un SYSM par gestionnaire de synchronisation. L'accès à la mémoire passe de SYSM en SYSM tant qu'aucun d'eux n'est activé pour cette adresse. Si aucun SYSM n'est activé pour une adresse mémoire spécifique, une demande est adressée à une zone de mémoire ou un registre. Certaines règles d'accès spécifiques s'appliquent aux registres. Elles sont présentées dans les attributs de registre de la CEI 61158-3-12. Le SYSM est présenté de manière plus détaillée à l'Article A.2.

8.2.4 Diagramme d'états de boîte aux lettres résilient (RMSM)

Le RMSM obligatoire est chargé des nouvelles tentatives de boîte aux lettres dans la boîte aux lettres en lecture et de vérifier les numéros de séquence dans la boîte aux lettres en écriture. Une nouvelle tentative d'écriture de boîte aux lettres est une écriture portant le même numéro de séquence.

Le mécanisme de nouvelle tentative de la boîte aux lettres en lecture utilise les paramètres Repeat et RepeatAck du canal du gestionnaire de synchronisation. Le basculement du paramètre Repeat amène l'esclave à tenter de nouveau la dernière lecture. L'Article A.3 décrit le comportement de la boîte aux lettres en lecture.

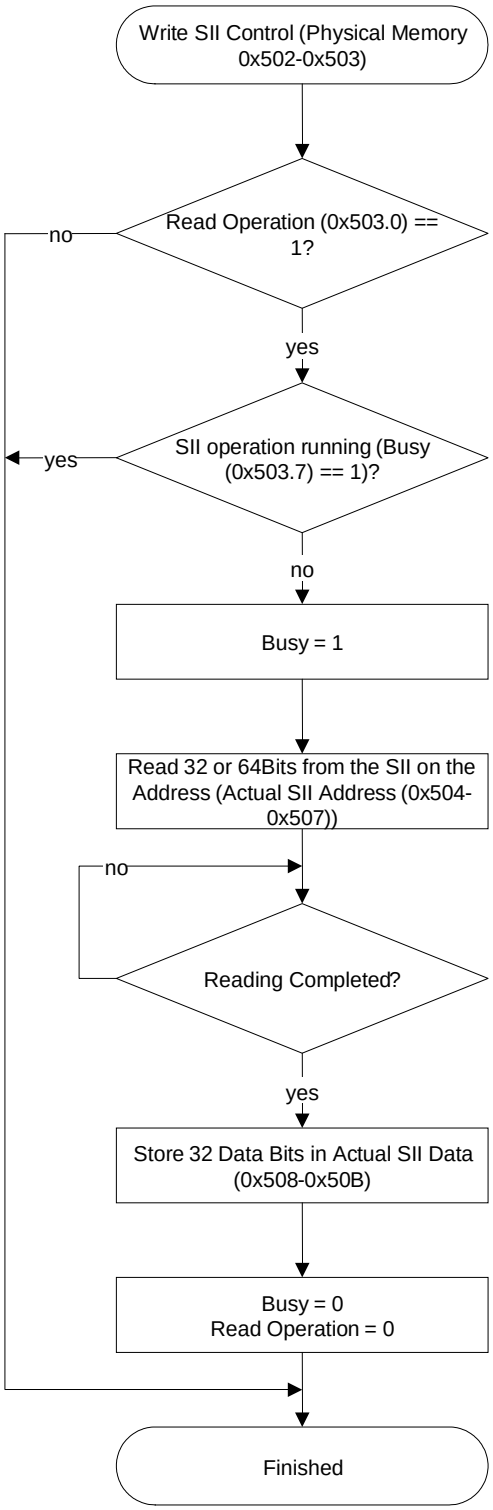
8.2.5 Diagramme d'états SII (SIISM)

8.2.5.1 Organigrammes d'accès à l'interface d'informations de l'esclave

Le SIISM est chargé de l'accès au SII. Des opérations de lecture, d'écriture et de recharge sont spécifiées pour cette interface. Le maître peut activer cette opération en suivant la séquence procédurale spécifique.

8.2.5.2 Read operation

La Figure 45 illustre le flux d'une opération de lecture.



Légende

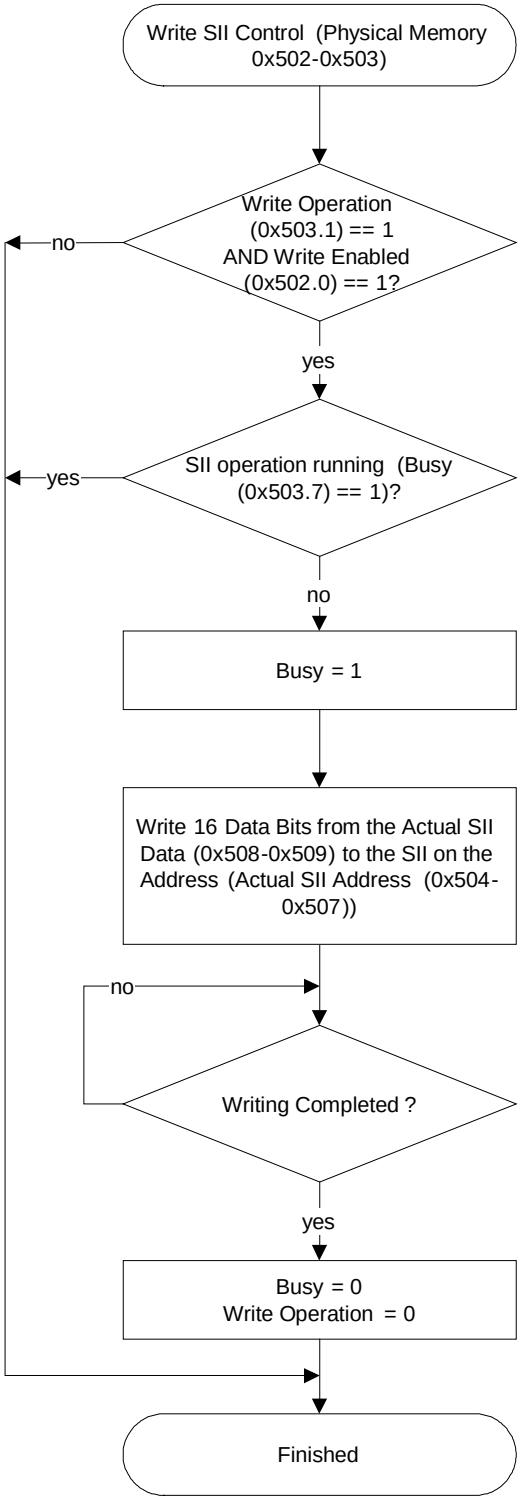
Anglais	Français
Write SII Control (Physical Memory 0x502-0x503)	Commande d'écriture SII (Mémoire physique 0x502-0x503)
Read Operation	Opération de lecture
SII operation running (Busy (0x503.7) == 1)?	Exécution de l'opération SII (Occupé (0x503.7) == 1)?
Busy = 1	Busy = 1
yes	oui
no	non

Anglais	Français
Read32 or 64bits from the SII on the Address (Actual SII Address (0x504-0x507))	Read32 ou 64bits à partir de SII sur l'adresse (adresse SII réelle (0x504-0x507))
Reading Completed?	Lecture terminée ?
Store 32 Data Bits in Actual SII Data	Stocker 32 bits de données dans les données SII réelles
Busy = 0 Read Operation = 0	Busy = 0 Opération de lecture = 0
Finished	Terminé

Figure 45 – Opération de lecture de l'interface d'informations de l'esclave

8.2.5.3 Write operation

La Figure 46 illustre le flux d'une opération d'écriture.



Légende

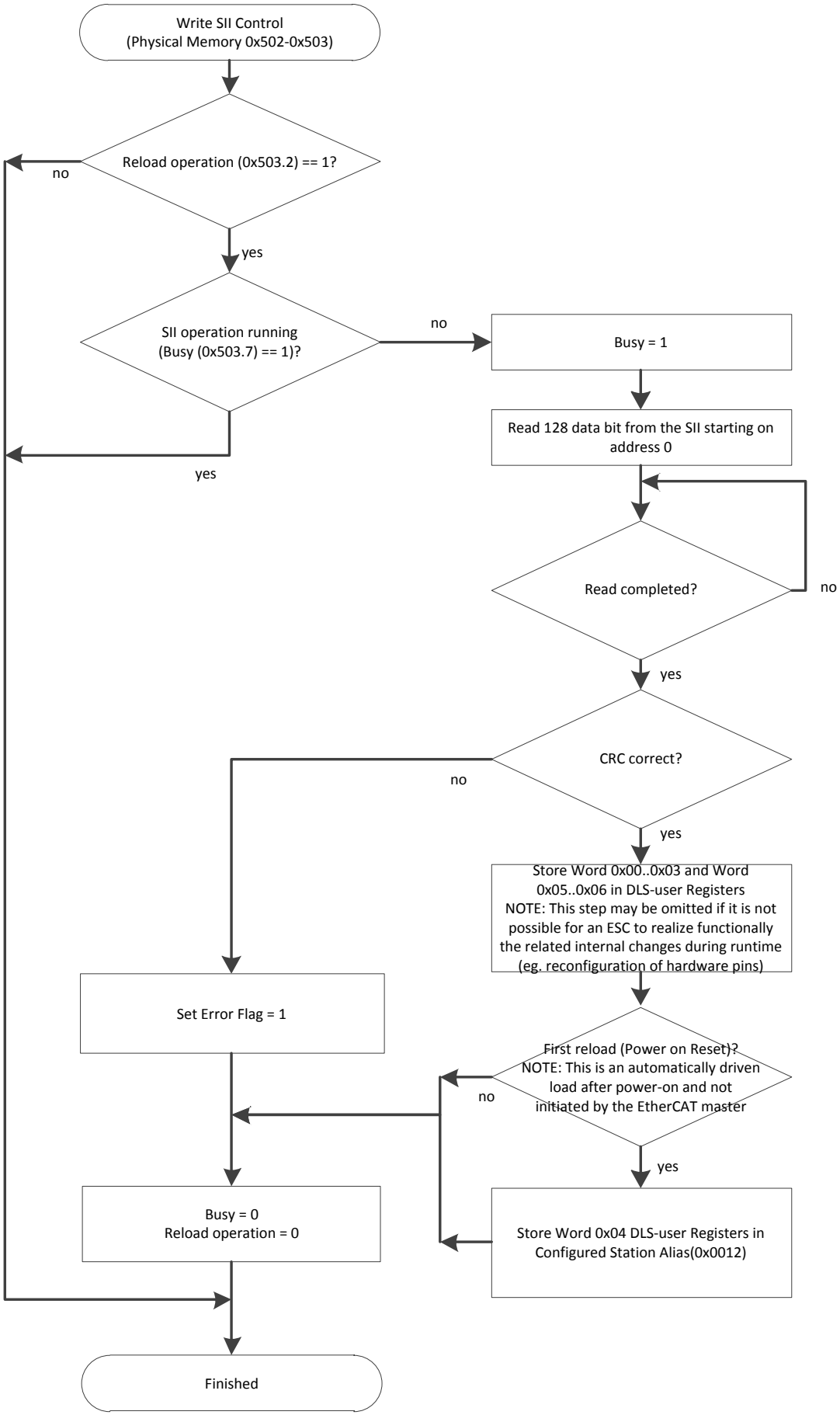
Anglais	Français
Write SII Control (Physical Memory 0x502-0x503)	Commande d'écriture SII (Mémoire physique 0x502-0x503)
Write Operation (0x503.1) == 1 AND Write Enabled	Opération d'écriture (0x503.1) == 1 ET Écriture activée
SII operation running (Busy (0x503.7) == 1)?	Exécution de l'opération SII (Occupé (0x503.7) == 1)?
Busy = 1	Busy = 1
yes	oui

Anglais	Français
no	non
Write 16 Data Bits from the Actual SII Data (0x508-0x509) to the SII on the Address (Actual SII Address (0x504-0x507))	Écriture de 16 bits de données à partir des données SII réelle (0x508-0x509) sur l'adresse (adresse SII réelle (0x504-0x507))
Writing Completed?	Écriture terminée ?
Busy = 0 Write Operation = 0	Busy = 0 Opération d'écriture = 0
Finished	Terminé

Figure 46 – Opération d'écriture de l'interface d'informations de l'esclave

8.2.5.4 Reload operation

La Figure 47 illustre le flux d'une opération de recharge.



Légende

Anglais	Français
Write SII Control (Physical Memory 0x502-0x503)	Commande d'écriture SII (Mémoire physique 0x502-0x503)
Reload Operation (0x503.2)	Opération de recharge (0x503.2)
SII operation running (Busy (0x503.7) == 1)?	Exécution de l'opération SII (Occupé (0x503.7) == 1)?
Busy = 1	Busy = 1
yes	oui
no	non
Read 128 Data Bits from SII starting on Address 0	Lecture de 128 bits de données de SII en commençant par l'adresse 0
Read Completed?	Lecture terminée ?
CRC correct?	CRC correct?
Set error flag = 1	Mettre le fanion d'erreur à 1
Busy = 0 Read Operation = 0	Busy = 0 Opération de lecture = 0
Finished	Terminé
Store word 0x00..0x03 and word 0x05..0x06 in DLS-user Registers NOTE This step may be omitted if it is not possible for an ESC to realize functionally the related internal changes during runtime (eg. Reconfiguraion of hardware pins)	Stocker le mot 0x00..0x03 et le mot 0x05..0x06 dans les registres de l'utilisateur DLS NOTE Cette étape peut être omise si un ESC n'est pas capable de réaliser fonctionnellement les changements internes liés au cours de l'exécution (par exemple Reconfiguraion des broches matérielles)
First reload (power on reset)? NOTE this is an automatically driven load after power-on and not initiated by the EtherCat master	Première recharge (remise à zéro à la mise sous tension)? NOTE il s'agit d'une charge commandée automatiquement après la mise sous tension et non initiée par le maître EtherCAT
Store Word 0x04 DLS-user Registers in Configured Station Alias (0x0012)	Stocker le mot 0x04 dans les registres de l'utilisateur DLS dans l'Alias de la station configurée (0x0012)

Figure 47 – Opération de recharge de l'interface d'informations de l'esclave

8.2.6 Diagramme d'états MII (MIISM)

Le MIISM est responsable de l'accès à l'interface MII (interface indépendante du support conformément à l'ISO/CEI 8802-3). Le flux suit la structure spécifiée en 8.2.5 avec des adresses distinctes pour la commande, l'adresse et la mémoire tampon de données.

8.2.7 Diagramme d'états DC (DCSM)

8.2.7.1 Description de la structure DC

Le DCSM assure la coordination de l'horloge locale, ainsi que la synchronisation et les fonctions de datation des horloges locales. Les registres DC sont décrits dans la CEI 61158-3-12.

Les horloges distribuées permettent à tous les appareils esclaves d'avoir la même heure. Le premier appareil esclave du segment contenant une horloge est l'horloge de référence. Son horloge permet de synchroniser les horloges esclaves des autres appareils esclaves et du maître. L'appareil maître envoie une PDU de synchronisation à certains intervalles (selon le cas, afin d'éviter que l'horloge esclave ne s'écarte des limites spécifiques à l'application), dans laquelle l'appareil esclave contenant l'horloge de référence entre son heure en cours. Les appareils esclaves dotés d'horloges esclaves lisent l'heure dans la PDU contenant le

service ARMW. Cela est possible en raison de la structure en anneau logique, étant donné que l'horloge de référence est placée avant les horloges esclaves dans le segment.

Étant donné que chaque esclave introduit un léger délai dans la direction sortante et de retour (à l'intérieur de l'appareil et sur la liaison physique), le délai de propagation entre l'horloge de référence et l'horloge esclave respective doit être pris en compte lors de la synchronisation des horloges esclaves. Pour mesurer le délai de propagation, l'appareil maître envoie une écriture de diffusion à une adresse particulière (le registre de temps de réception du port 0), ce qui amène chaque appareil esclave à sauvegarder l'heure de réception de la PDU (ou son temps horloge local) dans la direction sortante et sur le retour. Le maître peut lire ces heures sauvegardées et configurer un registre de délai en conséquence.

Définition d'une horloge de référence
Un esclave fait office d'horloge de référence. Il s'agit de la première horloge entre le maître et tous les esclaves à synchroniser. La distribution de l'horloge de référence est assurée de manière cyclique avec la commande ARMW ou FRMW. L'horloge de référence peut être ajustée à partir d'une horloge de référence «globale» (la CEI 61588, par exemple).

Condition préalable
Aucun mécanisme ne permet de calculer le temps de résidence. Par conséquent, aucune gigue significative du temps de résidence n'est admise pour tous les appareils esclaves Type 12.

Fonctions clés:

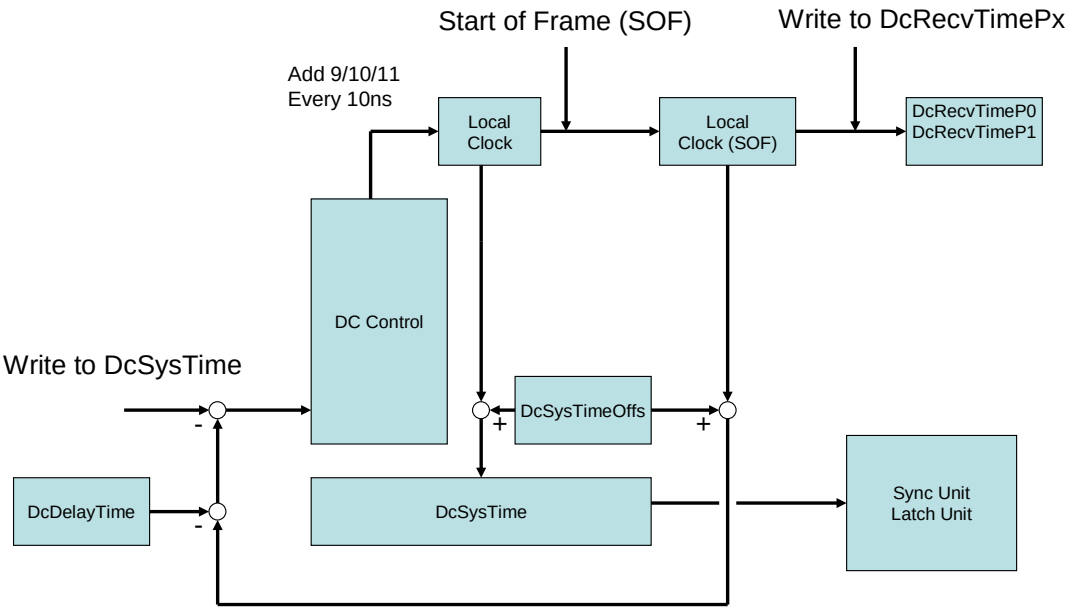
- Compensation de dérive par rapport à l'horloge de référence
- Mesure du délai de propagation

  Chaque contrôleur d'esclave mesure le délai entre les deux directions d'une trame

  Le maître calcule les délais de propagation entre tous les esclaves
- Compensation de décalage par rapport à l'horloge de référence (temps système)

  Temps système absolu identique dans tous les appareils (gigue inférieure à 1 µs)

La Figure 48 présente la structure de l'élément DC. Elle suppose une fréquence d'horloge locale de 100 MHz, mais une résolution d'horloge de 1 ns. Cela permet d'ajuster l'horloge locale par petits incréments jusqu'à la fréquence d'horloge globale et d'éviter les sauts dans l'échelle de temps.



Légende

Anglais	Français
Start of Frame	Début de trame
Add 9/10/11 Every 10ns	Ajouter 9/10/11 toutes les 10 ns
Local Clock	Horloge locale
DC Control	Commande DC
Write to DcSysTime	Écrire dans DcSysTime
Write to DcRecvTimePx	Écrire dans DcRecvTimePx
Sync Unit	Unité de sync
Latch Unit	Unité de verrouillage

Figure 48 – Horloge distribuée

Le préambule de la trame Ethernet commence par le début de trame (SOF).

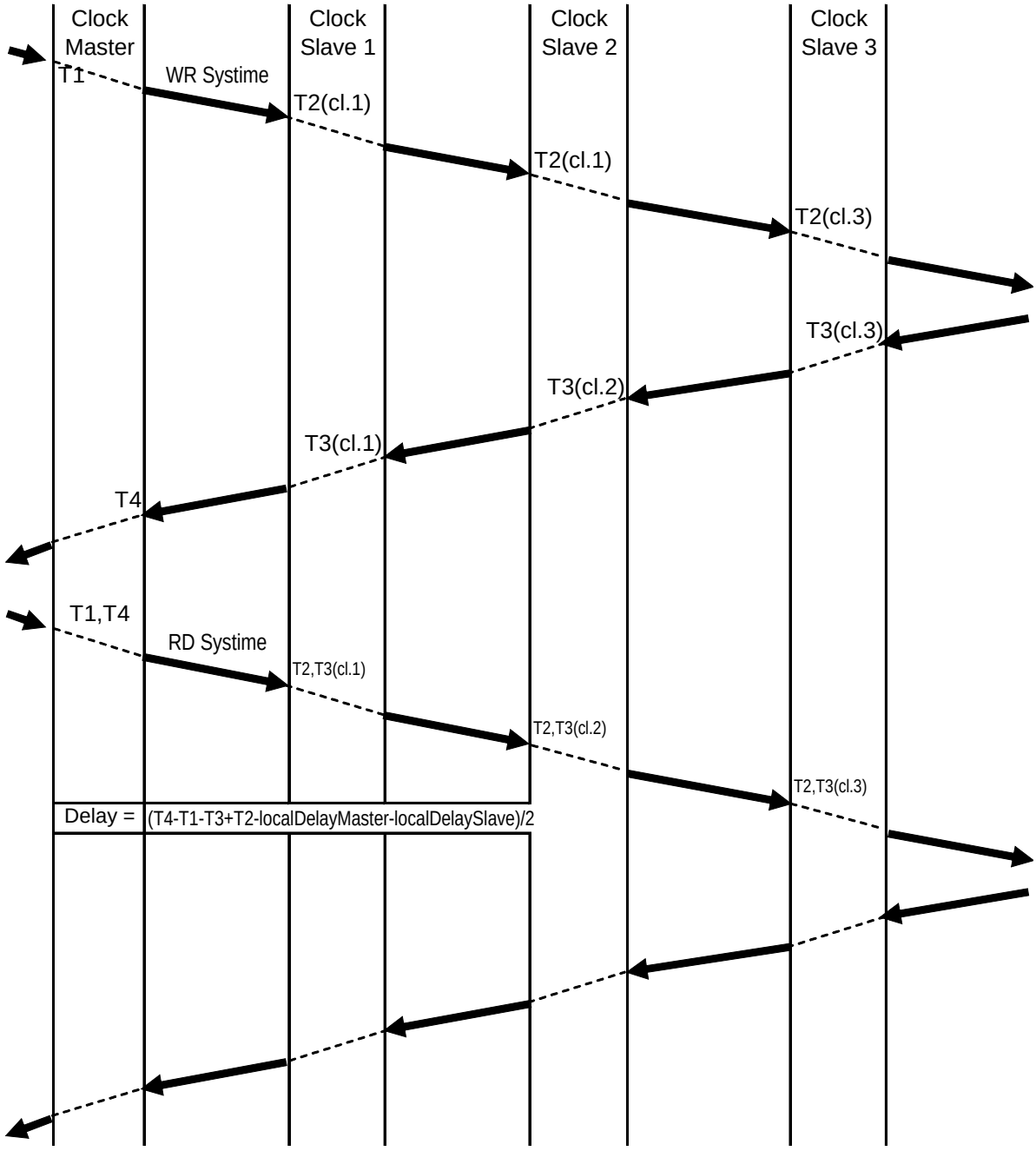
L'horloge locale est incrémentée de 10 toutes les 10 ns et en fonction de la dérive de l'horloge de 9 ou 11, selon un intervalle régulier spécifié dans la commande DC (pour la compensation de dérive).

SysTimeOffset permet une adaptation sans modifier l'horloge locale indépendante. En tant que deuxième décalage, le délai permet de compenser les délais de l'horloge de référence par rapport à l'horloge esclave.

La synchronisation externe est assurée par les mécanismes spécifiés dans la CEI 61588. Tous les appareils dotés d'interfaces de communication externes peuvent contenir une horloge «boundary clock». L'esclave doté de l'horloge maître est synchronisé par rapport à l'horloge «boundary clock». Un segment de Type 12 ne doit comporter à tout moment qu'une seule horloge «boundary clock» afin de satisfaire aux exigences de topologie de la CEI 61588.

L'horloge distribuée est utilisée pour répondre à des exigences de temporisation très précises. Les systèmes dont les besoins en synchronisation se situent dans la plage de 10 µs au moins ou qui sont dotés d'autres moyens de compensation de délai peuvent être synchronisés en partageant les PDU Type 12 accédant aux mémoires tampons en écriture des appareils à synchroniser.

8.2.7.2 Mesure du délai



Légende

Anglais	Français
Clock Master	Horloge maître
Clock Slave	Horloge esclave
Delay =	Délai =

Figure 49 – Séquence de mesure du délai

La Figure 49 illustre le principe de mesure du délai.

T1 (de l'horloge maître) et T2 (des horloges esclaves 1, 2 et 3, ici appelées cl.1, cl.2 et cl.3) font référence à la durée de réception du port 0, T3 (des horloges esclaves 1, 2 et 3, ici appelées cl.1, cl.2 et cl.3) et T4 fait référence à la durée de réception du port 1. T2 = T3 pour la dernière horloge esclave. Ce modèle suppose une connexion symétrique, c'est-à-dire que le chemin entre A et B est aussi long que le chemin entre B et A. Des retards de ligne et des délais de propagation différents dans les PHY Ethernet peuvent donner lieu à un décalage de temps constant.

Annexe A
(informative)

Type 12: Spécifications supplémentaires relatives aux diagrammes d'états de protocole DL

NOTE 1 La présente annexe spécifie un certain nombre de diagrammes d'états finis utilisés par la DLE pour assurer ses fonctions de protocole de niveaux inférieur et supérieur. Cette spécification vient en complément de la spécification textuelle présentée dans la présente norme; en cas de conflit, les exigences de la spécification textuelle l'emportent.

NOTE 2 Les descriptions du diagramme d'états finis données ici sont nécessairement moins exhaustives qu'une description complète d'une mise en œuvre. Des exigences et considérations supplémentaires sont disponibles dans la spécification textuelle.

A.1 DHSM

A.1.1 Définitions de primitive

A.1.1.1 Primitive échangée entre PSM et DHSM

Le Tableau A.1 représente primitives émises par DHSM au PSM.

Tableau A.1 – Primitives émises par DHSM au PSM

Nom de primitive	Paramètres associés
Tx request	Port, Byte, Data

Le Tableau A.2 représente les primitives émises par le PSM au DHSM.

Tableau A.2 – Primitives émises par le PSM au DHSM

Nom de primitive	Paramètres associés
Rx indication	Port, Byte, Data

A.1.1.2 Paramètres des primitives PSM

Le Tableau A.3 représente tous les paramètres utilisés avec les primitives entre le DHSM et le PSM.

Tableau A.3 – Paramètres utilisés avec les primitives échangées entre DHSM et PSM

Nom de paramètre	Description
Port	Identificateur du port local, commençant par 0 en tant que port principal
Byte	Identificateur de l'octet reçu/émis tel que spécifié dans le Tableau A.4
Data	Valeur de l'octet reçu/émis

Tableau A.4 – Identificateur des octets d'une trame Ethernet

Nom de paramètre	Description
0	Premier octet du préambule
1	Autres octets du préambule
2	Premier octet de DA
3	Autres octets de DA
4	Premier octet de SA
5	Autres octets de SA
6	Premier octet de VLAN
7	Autres octets de VLAN
8	Premier octet de Ethertype
9	Deuxième octet de Ethertype
10	Premier octet de SDU Ethernet
10+ n	(n+1)ème octet de SDU Ethernet
0xffff (END)	Fin de trame avec séquence de contrôle de trame correcte
0xfffe (ERR)	Annuler la trame erronée

Description du diagramme d'états

Il existe exactement un DHSM par appareil esclave.

Le DHSM assure l'interface entre l'interaction distante et la mémoire locale. Le DHSM transmet les octets de la trame d'un port à l'autre.

Si une commande Type 12 est reconnue, une interaction avec les diagrammes d'états SYM sera émise si la mémoire locale est adressée. Les actions locales seront appelées si l'indication a été émise au port 0. La seule action locale émise sur les autres ports est la datation des trames entrantes.

Ce diagramme d'états décrit l'interprétation des trames Ethernet comportant un Ethertype en temps réel spécifique ou un port de destination UDP spécifique. Les traitements Type 12 spécifiques (la détection des trames en circulation, l'incrémentation d'adresse à incrément automatique, la mise à jour du compteur en fonction et de la séquence de contrôle de trame, par exemple) seront assurés par le DHSM.

Le traitement des erreurs est décrit au niveau logique. Pour une meilleure localisation des liaisons erronées, la station qui détecte une erreur transmettra un symbole physique de 4 bits avec la FCS endommagée, ce qui provoquerait une erreur d'alignement. Cette erreur d'alignement, combinée à une FCS inversée dans les 2 derniers bits, signale que le problème s'est produit sur une liaison différente. Chaque erreur détectée génère une entrée supplémentaire dans les attributs statistiques appropriés.

Constantes Locales
LASTP
Identifiant du dernier port Ethernet. Les ports sont numérotés de 0 à LASTP.

END
Indicateur de fin d'une trame Ethernet.

ERR
Indicateur de fin d'une trame Ethernet suite à une condition d'erreur.

Variables locales
CMD
Identifiant de commande d'une PDU Type 12.

Etype1
Premier octet de Ethertype.

Length
Longueur d'une PDU Type 12.

MF
Indique la dernière PDU Type 12 d'une trame Ethernet.

RxTimeLatch[0..LASTP]
Longueur d'une PDU Type 12.

AdL
Premier octet d'adresse d'une PDU Type 12.

AdH
Deuxième octet d'adresse d'une PDU Type 12.

DaL
Troisième octet d'adresse d'une PDU Type 12.

DaH
Quatrième octet d'adresse d'une PDU Type 12.

LeL
Premier octet de longueur d'une PDU Type 12.

LeH
Deuxième octet de longueur d'une PDU Type 12.

wkc
Facteur supplémentaire local à ajouter au compteur en fonction d'une PDU Type 12.

RData
Octet de données local de l'élément de mémoire.

WData
Octet de données d'une PDU Type 12 à écrire.

Overflow
Indique un dépassement de deux octets supplémentaires traités comme un entier non signé.

Nomenclature de la table d'états

Les suffixes normalisés «.req», «.cnf» et «.ind» sont respectivement utilisés pour indiquer les primitives de requête, de confirmation et d'indication.

Table DHSM

La table d'états DHSM est présentée dans le Tableau A.5.

Tableau A.5 – Table d'états DHSM

#	État en cours	Événement /condition ⇒action	État suivant
1	ETH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ETH
2	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 0 => RxTimeLatch[Port] = CT Port = 1 Tx.req(Port,Byte,Data)	ETH
3	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 1 => Port = 1 Tx.req(Port,Byte,Data)	ETH
4	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 2 => Port = 1 INIT_FCS(Data) Tx.req(Port,Byte,Data)	ETH
5	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 3 => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
6	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 4 => Port = 1 Data = Data | 2 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
7	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 4 && Byte < 8 => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
8	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 8 => Port = 1 Etype1 = Data UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
9	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data== 0xA4 && Etype1== 0x88) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECAT
10	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data== 0x00 && Etype1== 0x08) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIP

#	État en cours	Événement /condition ⇒action	État suivant
11	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data != 0xA4 || Etype1 != 0x88) && (Data != 0x00 || Etype1 != 0x08) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
12	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 9 => Port = 1 Tx.req(Port,Byte,Data)	ETH
13	EIP	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIP
14	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte < 10 => Port = 1 Byte = ERR Tx.req(Port,Byte,Data)	ETH
15	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && (Byte == END || Byte == ERR) => Port = 1 Tx.req(Port,Byte,Data)	ETH
16	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 && Data == 0x45 => Port = 1 Tx.req(Port,Byte,Data)	EIP
17	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 && Data !=0x45 => Port = 1 Tx.req(Port,Byte,Data)	ETH
18	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 10 && Byte < 19 => Port = 1 Tx.req(Port,Byte,Data)	EIP
19	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 19 && Data == 0x11 => Port = 1 Tx.req(Port,Byte,Data)	EIP
20	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 19 && Data !=0x11 => Port = 1 Tx.req(Port,Byte,Data)	ETH
21	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 19 && Byte < 32 => Port = 1 Tx.req(Port,Byte,Data)	EIP
22	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 32 && Data == 0x88 => Port = 1 Tx.req(Port,Byte,Data)	EIP

#	État en cours	Événement /condition ⇒action	État suivant
23	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 32 && Data != 0x88 => Port = 1 Tx.req(Port,Byte,Data)	ETH
24	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 33 && Data == 0xA4 => Port = 1 Tx.req(Port,Byte,Data)	EIP
25	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 33 && Data != 0xA4 => Port = 1 Tx.req(Port,Byte,Data)	ETH
26	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 33 && Byte < 36 => Port = 1 Tx.req(Port,Byte,Data)	EIP
27	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 36 => Port = 1 Data = 0 Tx.req(Port,Byte,Data)	EIP
28	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 37 => Port = 1 Data = 0 Tx.req(Port,Byte,Data)	ECAT
29	ECAT	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECAT
30	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte < 10 => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
31	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 => Len = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECAT
32	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 11&& (Data & 0xf0 == 0x10) => Len = Len + (Data*256 & 0x0f) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECMD
33	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 11&& (Data & 0xf0 != 0x10) => Port = 1 Tx.req(Port,Byte,Data)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
34	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
35	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
36	ECMD	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECMD
37	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 => CMD = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIDX
38	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
39	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
40	EIDX	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIDX
41	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 => Cmd = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADL
42	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
43	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
44	EADL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EADL
45	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == BRD, BWR, BRW, APRD, APWR, APRW,ARMW => AdL = Data Data = Data + 1 Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADH
46	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == other => AdL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADH
47	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
48	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
49	EADH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EADH
50	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == BRD, BWR, BRW => AdH = Data if AdL == 0xff then Data = Data + 1 Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL

#	État en cours	Événement /condition ⇒action	État suivant
51	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == APRD, APWR, APRW,ARMW => AdH = Data if AdL == 0xff then Data = Data + 1 if Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
52	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == FPRD, FPWR, FPRW,FRMW => AdH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
53	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == LRD, LWR, LRW => AdH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
54	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == other => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
55	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
56	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
57	EDAL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDAL
58	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 => DaL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAH
59	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
60	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
61	EDAH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDAH
62	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 => DaH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ELEL
63	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
64	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
65	ELEL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ELEL
66	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 => LeL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ELEH
67	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
68	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
69	ELEH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ELEH
70	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && (!Closed[Port] || Data & 0x40 == 0) => if Closed[Port] then LeH = Data | 0x40 else LeH = Data MF = LeH & 0x80 Length = LeL + 256 * (LeH & 0x0f) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIRL
71	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && (Closed[Port] && Data & 0x40 != 0) => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
72	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
73	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
74	EIRL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIRL
75	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 => if Ena then Data == Data | (EventH & EventMskH) Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIRH
76	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
77	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
78	EIRH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIRH
79	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length == 0 => wkc = 0 Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
80	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 && !Ena => wkc = 0 Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTI
81	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 && Ena => wkc = 0 Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
82	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
83	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
84	EDTI	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDTI
85	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length == 0 => Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL

#	État en cours	Événement /condition ⇒action	État suivant
86	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 => Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTI
87	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
88	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
89	EDTA	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDTA
90	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && R_FMMUMATCH(LOGADD) => MemoryAddress = RFMMUMAP (LOGADD) WKC = 0 WData= Data Read.ind (Address, Data, WKC)	WSYLR
91	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && W_FMMUMATCH(LOGADD) => MemoryAddress = WFMMUMAP (LOGADD) WKC = 0 RData= Data Write.ind (Address, Data, WKC)	WSYLW
92	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && N_FMMUMATCH(LOGADD) && Length == 0 => INC(LOGADD) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
93	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && N_FMMUMATCH(LOGADD) && Length != 0 => Length -- INC(LOGADD) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
94	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDPR => WKC = 0 RData= Data Read.ind (Address, Data, WKC)	WSYPR

#	État en cours	Événement /condition ⇒action	État suivant
95	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDPW => WKC = 0 WData= Data Write.ind (Address, Data, WKC)	WSYPW
96	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
97	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
98	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /W_FMMUMATCH(LOGADD) => wkc = wkc | WKC MemoryAddress = WFMMUMAP (LOGADD) RData = Data Data = WData WKC = 0 Write.ind (Address, Data, WKC)	WSYLW
99	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /!W_FMMUMATCH(LOGADD) && Length == 0 => wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
100	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /!W_FMMUMATCH(LOGADD) && Length != 0 => Length -- INC(LOGADD) wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
101	WSYLW	**Write.rsp (MemoryAddress, Data, WKC)** /Length == 0 => INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
102	WSYLW	**Write.rsp (MemoryAddress, Data, WKC)** /Length != 0 => Length -- INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA

#	État en cours	Événement /condition ⇒action	État suivant
103	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /CMDPW => wkc = wkc | WKC if OR then Data = WData | Data RData = Data Data = WData WKC = 0 Write.ind (Address, Data, WKC)	WSYPW
104	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /!CMDPW && Length == 0 => if OR then Data = WData | Data wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
105	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /!CMDPW && Length != 0 => Length -- if OR then Data = WData | Data wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
106	WSYPW	**Write.rsp (MemoryAddress, Data, WKC)** /Length == 0 => INC(LOGADD) wkc = wkc | (WKC * CMDPRMW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
107	WSYPW	**Write.rsp (MemoryAddress, Data, WKC)** /Length != 0 => Length -- INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
108	EWKL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EWKL
109	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 => Overflow, Data = Data + wkc Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKH
110	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
111	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
112	EWKH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECMD
113	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && MF => Data = Data + Overflow Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
114	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && !MF => Data = Data + Overflow Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EFCS1
115	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
116	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
117	EFCS1	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS1
118	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS1 Port 1 Tx.req(Port,Byte,Data)	EFCS2
119	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
120	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
121	EFCS2	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS2
122	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS2 Port 1 Tx.req(Port,Byte,Data)	EFCS3
123	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
124	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
125	EFCS3	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS3
126	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS3 Port 1 Tx.req(Port,Byte,Data)	EFCS4
127	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
128	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

#	État en cours	Événement /condition ⇒action	État suivant
129	EFCS4	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS4
130	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS4 Port 1 Tx.req(Port,Byte,Data)	EFCS5
131	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
132	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
133	EFCS5	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS5
134	EFCS5	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = END Success = TRUE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
135	EFCS5	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte != END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

Fonctions

Les fonctions DHSM sont récapitulées dans le Tableau A.6.

Tableau A.6 – Table de fonctions DHSM

Nom de fonction	Opérations
INIT_FCS(Data)	Lance la séquence de contrôle de trame avec 0xFFFFFFFF
UPD_FCS(Data)	Met à jour la séquence de contrôle de trame conformément à l'ISO/CEI 8802-3
CMDL	Cmd == LRD, LRW, LWR
CMDPR	Cmd == BRD, BRW || (Cmd == ARD, ARW, ARMW && AdH, AdL == 0) || (Cmd == FRD, FRW, FRMW && AdH, AdL==ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled)
CMDPW	Cmd == BWR; BRW || (Cmd == AWR, ARW && (AdH, AdL-1) == 0) || (Cmd == FWR, FRW && AdH, AdL==ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled) || (Cmd == ARMW && (AdH, AdL-1) != 0) || (Cmd == FRMW && AdH, AdL != ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled)
CMDLRW	if Cmd == LRW then 2 else 1
CMDRMW	if Cmd == ARMW,FRMW then 0 else 1
RFMMUMAP (LOGADD)	Mapper LOGADD à l'adresse dans l'espace mémoire à l'aide de Read FMMU
WFMMUMAP (LOGADD)	Mapper LOGADD à l'adresse dans l'espace mémoire à l'aide de Write FMMU
R_FMMUMATCH (LOGADD)	LOGADD peut être mappé en lecture à une adresse de la mémoire && Cmd == LRD, LRW
W_FMMUMATCH (LOGADD)	LOGADD peut être mappé en écriture à une adresse de la mémoire && Cmd == LWR, LRW
N_FMMUMATCH (LOGADD)	!W_FMMUMATCH (LOGADD) && !R_FMMUMATCH (LOGADD)
OR	Cmd == BRD, BRW
INC(LOGADD)	Overflow, AdL = AdL+ 1 Overflow, AdH= AdH + Overflow Overflow, DaL= DaL + Overflow Overflow, DaH= DaH + Overflow

A.2 SYSM

A.2.1 Définition de primitive

A.2.1.1 Primitive échangée entre DHSM et SYSM

Le Tableau A.7 représente les primitives émises par DHSM au PSM.

Tableau A.7 – Primitives émises par SYSM au DHSM

Nom de primitive	Paramètres associés
Read response (Réponse de lecture)	Address, Data, WKC
Write response (Réponse d'écriture)	Address, Data, WKC

Le Tableau A.8 représente les primitives émises par le DHSM au SYSM.

Tableau A.8 – Primitives émises par DHSM au SYSM

Nom de primitive	Paramètres associés
Read indication (Indication de lecture)	Address, Data, WKC
Write indication (Indication d'écriture)	Address, Data, WKC
Terminate indication (Indication de fin)	Success

A.2.1.2 Primitive échangée entre utilisateur de DL et SYSM

Le Tableau A.9 représente les primitives émises par l'utilisateur de DL au SYSM.

Tableau A.9 – Primitives émises par l'utilisateur au SYSM

Nom de primitive	Paramètres associés
DL-Read local request (Demande locale de lecture DL)	Address, Length
DL-Write local request (Demande locale d'écriture DL)	Address, Length, Data

Le Tableau A.10 représente les primitives émises par le SYSM à l'utilisateur de DL.

Tableau A.10 – Primitives émises par SYSM à l'utilisateur de DL

Nom de primitive	Paramètres associés
DL-Read local confirmation (Confirmation locale de lecture DL)	L-Status, Data
DL-Write local confirmation (Confirmation locale d'écriture DL)	L-Status

NOTE Les événements locaux ne sont pas modélisés étant donné qu'ils n'ont aucun impact sur le SYSM.

A.2.1.3 Paramètres des primitives DHSM

Le Tableau A.11 représente tous les paramètres utilisés avec les primitives entre le SYSM et le DHSM.

Tableau A.11 – Paramètres utilisés avec les primitives échangées entre SYSM et DHSM

Nom de paramètre	Description
WKC	Compteur en fonction de l'accès local
Address	Identificateur de la zone de mémoire physique
Data	Valeur de l'octet reçu/émis

Description du diagramme d'états

Il existe un SYSM par canal du gestionnaire de synchronisation. Le modèle abstrait implique de transmettre les primitives d'indication Type 12 Read/Write et les primitives de demande Read/Write au premier SYSM, puis de les exécuter si une adresse correspond ou a été transmise au SYSM suivant. Si aucun SYSM ne répond à la primitive de service, un

gestionnaire de mémoire physique exécutera la demande de service si la mémoire ou le registre est présent(e) et si le type de service est activé pour ce registre. Un accès en écriture au registre sera assuré si le DHSM parvient à analyser la trame Ethernet. L'accès en lecture au premier octet du mot ou des registres à mot double est exécuté de manière atomique, c'est-à-dire que la lecture du premier octet gèlera la valeur pour les accès ultérieurs.

Il existe des types de SYSM mis en mémoire tampon et un type de boîte aux lettres. Les demandes locales sont modélisées de cette manière, à savoir que l'accès en lecture/écriture est totalement à l'intérieur ou à l'extérieur des limites d'une zone du gestionnaire de synchronisation.

Les événements du gestionnaire de synchronisation sont générés lorsque la zone de registre du canal du gestionnaire de synchronisation est écrite.

Variables locales

eact
Activation d'une mémoire tampon par le maître.

uact
Activation d'une mémoire tampon par l'utilisateur de DL.

Terminate
Indique la fin d'une transaction de boîte aux lettres ou de mémoire tampon.

User
Contient la mémoire tampon de l'utilisateur.

Buffer
Contient la mémoire tampon utilisée pour la communication.

Next
Contient la mémoire tampon remplie pour la prochaine utilisation.

Free
Contient la mémoire tampon vide.

p1, p2, p3
Emplacement de la mémoire des trois mémoires tampons, la première se trouvant à l'endroit spécifié par le gestionnaire de synchronisation, les autres étant placées dans les emplacements suivants.

act
Contient le code d'activité d'une boîte aux lettres.

Nomenclature de la table d'états

Les suffixes normalisés «.req», «.cnf» et «.ind» sont respectivement utilisés pour indiquer les primitives de requête, de confirmation et d'indication.

Table SYSM

La table d'états SYSM est montrée au Tableau A.12.

Tableau A.12 – Table d'états SYSM

#	État en cours	Événement /Condition =>Action	État suivant

1	tous les états	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 0 && SM.Direction == 0
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		eact, uact = FALSE
		Terminate = FALSE
		next=NIL,buffer=p0,user=p1,free=p2
		SM.bufferedState=3
		if (AL Event Enable) then Enable SM Event (Toggle)
			BR-IDLE

2	tous les états	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 0 && SM.Direction == 1
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		eact, uact = FALSE
		Terminate = FALSE
		next=NIL, buffer =p0,user=p1,free=p2
		SM.bufferedState=3
		if (AL Event Enable) then Enable SM Event (Toggle)
			BW-IDLE

3	tous les états	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 2 && SM.Direction == 0
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		Terminate = FALSE
		act = 0
		SM.mailboxState=0
		if (AL Event Enable) then Enable SM Event (Toggle)
			MR-IDLE

4	tous les états	**SM Event**
		/(SM.ChannelEnable && !SM.ChannelDisable) && SM.Buffer Type == 2 && SM.Direction == 1
		=>
		SM.Toggle = SM.Repeat
		if (SM.Watchdog enable) then start WD timer
		Terminate = FALSE
		act = 0
		SM.mailboxState=0
		if (AL Event Enable) then Enable SM Event (Toggle)
			MW-IDLE

5	tous les états	**SM Event** /(!SM.ChannelEnable || SM.ChannelDisable) || SM.Buffer Type == 1,3 =>	OFF
6	OFF	**DL-Write Local.req (Address, Length, Data)** => pass next	OFF
7	OFF	**DL-Read Local.req (Address, Length)** => pass next	OFF
8	OFF	**Write.ind (Address, Data, WKC)** => pass next	OFF
9	OFF	**Read.ind (Address, Data, WKC)** => pass next	OFF
10	OFF	**Terminate.ind(Success)** => pass next	OFF

#	État en cours	Événement /Condition =>Action	État suivant
11	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && !uact => (user. Address) = Data L-Status = OK uact = TRUE SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	BR-IDLE
12	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && uact => (user. Address) = Data L-Status = OK SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	BR-IDLE
13	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && !uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
14	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK uact = FALSE SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
15	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /NA&&AE && !uact => L-Status = WRNGSEQ DL-Write Local.cnf (L-Status)	BR-IDLE
16	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&E && uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK uact = FALSE SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
17	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&NE && uact => (user. Address) = Data L-Status = OK DL-Write Local.cnf (L-Status)	BR-IDLE
18	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BR-IDLE
19	BR-IDLE	**DL-Read Local.req (Address, Length)** => pass next	BR-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
20	BR-IDLE	**Write.ind (Address, Data, WKC)** => pass next	BR-IDLE
21	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && !eact => if next != NIL then free = buffer, buffer = next, next = NIL Data = (buffer. Address) eact = TRUE WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
22	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && eact => Data = (buffer. Address) WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
23	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && !eact => if next != NIL then free = buffer, buffer = next, next = NIL Data = (buffer. Address) WKC = WKC +1 eact = TRUE Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
24	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && eact => /*Buffer exchange possible*/ Data = (buffer. Address) WKC = WKC +1 Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
25	BR-IDLE	**Read.ind (Address, Data, WKC)** /NA&&AE && !eact => Read.rsp (Address, Data, WKC)	BR-IDLE
26	BR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&E && eact => Data = (buffer. Address) WKC = WKC +1 Terminate = TRUE Read.rsp (Address, Data, WKC)	BR-IDLE
27	BR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&NE && eact => Data = (buffer. Address) WKC = WKC +1 Read.rsp (Address, Data, WKC)	BR-IDLE
28	BR-IDLE	**Read.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BR-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
29	BR-IDLE	**Terminate.ind(Success)** /Success && Terminate => eact = FALSE Terminate = FALSE SM.ReadEvent = 1 pass next	BR-IDLE
30	BR-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && eact) then eact = FALSE pass next	BR-IDLE
31	BR-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE eact = FALSE pass next	BR-IDLE
32	BW-IDLE	**DL-Write Local.req (Address, Length, Data)** => pass next	BW-IDLE
33	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && !uact => if next != NIL then free = user, user = next, next = NIL Data = (user. Address) L-Status = OK uact = TRUE SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
34	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && uact => Data = (user. Address) L-Status = OK SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
35	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && !uact => if next != NIL then free = user, user = next, next = NIL Data = (user. Address) L-Status = OK SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
36	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && uact => Data = (user. Address) L-Status = OK uact = FALSE SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
37	BW-IDLE	**DL-Read Local.req (Address, Length)** /NA&&AE && !uact => L-Status = WRNGSEQ DL-Read Local.cnf (L-Status, Data)	BW-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
38	BW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&E && uact => Data = (user. Address) L-Status = OK uact = FALSE SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
39	BW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&NE && uact => Data = (user. Address) L-Status = OK DL-Read Local.cnf (L-Status, Data)	BW-IDLE
40	BW-IDLE	**DL-Read Local.req (Address, Length)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BW-IDLE
41	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && !eact => (buffer. Address) = data eact = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
42	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && eact => (buffer. Address) = data WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
43	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && !eact => (buffer. Address) = data WKC = WKC +1 eact = TRUE Terminate = TRUE SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
44	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && eact => (buffer. Address) = data Terminate = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
45	BW-IDLE	**Write.ind (Address, Data, WKC)** /NA&&AE && !eact => Write.rsp (Address, Data, WKC)	BW-IDLE
46	BW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&E && eact => (buffer. Address) = data WKC = WKC +1 Terminate = TRUE Write.rsp (Address, Data, WKC)	BW-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
47	BW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&NE && eact => (buffer. Address) = data WKC = WKC +1 Write.rsp (Address, Data, WKC)	BW-IDLE
48	BW-IDLE	**Write.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BW-IDLE
49	BW-IDLE	**Read.ind (Address, Data, WKC)** => pass next	BW-IDLE
50	BW-IDLE	**Terminate.ind(Success)** /Success && Terminate => eact = FALSE Terminate = FALSE if next == NIL then next = buffer, buffer = free, free = NIL else user <=> next SM.bufferedState=next buffer number SM.WriteEvent = 1 pass next	BW-IDLE
51	BW-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && eact) then eact = FALSE pass next	BW-IDLE
52	BW-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE eact = FALSE pass next	BW-IDLE
53	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && act ==0 => (user. Address) = Data act = 1 L-Status = OK SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	MR-IDLE
54	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&AE && act != 0 => L-Status = NODATA DL-Write Local.cnf (L-Status)	MR-IDLE
55	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && act == 0 => (user. Address) = Data act = 2 SM.mailboxState=1 L-Status = OK SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	MR-IDLE
56	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&AE && && act != 1 => L-Status = NODATA DL-Write Local.cnf (L-Status)	MR-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
57	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&E && act == 1 => (user. Address) = Data act = 2 SM.mailboxState=1 L-Status = OK SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	MR-IDLE
58	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&NE && act == 1 => (user. Address) = Data L-Status = OK DL-Write Local.cnf (L-Status)	MR-IDLE
59	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MR-IDLE
60	MR-IDLE	**DL-Read Local.req (Address, Length)** => pass next	MR-IDLE
61	MR-IDLE	**Write.ind (Address, Data, WKC)** => pass next	MR-IDLE
62	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && act ==2 => Data = (buffer. Address) act = 3 WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	MR-IDLE
63	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&AE && act != 2 => Read.rsp (Address, Data, WKC)	MR-IDLE
64	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && act == 2 => Data = (buffer. Address) WKC = WKC +1 act = 4 Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	MR-IDLE
65	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&AE && && act != 3 => Read.rsp (Address, Data, WKC)	MR-IDLE
66	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&E && act == 3 => Data = (buffer. Address) WKC = WKC +1 act = 4 Terminate = TRUE Read.rsp (Address, Data, WKC)	MR-IDLE
67	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&NE && act == 3 => Data = (buffer. Address) WKC = WKC +1 Read.rsp (Address, Data, WKC)	MR-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
68	MR-IDLE	**Read.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MR-IDLE
69	MR-IDLE	**Terminate.ind(Success)** /Success && Terminate => Terminate = FALSE act = 0 SM.mailboxState=0 SM.ReadEvent = 1 pass next	MR-IDLE
70	MR-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && act == 3) then act = 2 pass next	MR-IDLE
71	MR-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE act = 2 pass next	MR-IDLE
72	MW-IDLE	**DL-Write Local.req (Address, Length, Data)** => pass next	MW-IDLE
73	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && act ==3 => Data = (User. Address) L-Status = OK act = 4 SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	MW-IDLE
74	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&AE && act != 3 => L-Status = NODATA DL-Read Local.cnf (L-Status, Data)	MW-IDLE
75	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && act == 3 => Data = (User. Address) L-Status = OK act = 0 SM.mailboxState=0 SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	MW-IDLE
76	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&AE && && act != 4 => L-Status = NODATA DL-Read Local.cnf (L-Status, Data)	MW-IDLE
77	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&E && act == 4 => Data = (User. Address) L-Status = OK act = 0 SM.mailboxState=0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	MW-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
78	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&NE && act == 4 => Data = (User. Address) L-Status = OK DL-Read Local.cnf (L-Status, Data)	MW-IDLE
79	MW-IDLE	**DL-Read Local.req (Address, Length)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MW-IDLE
80	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && act ==0 => (buffer. Address) = Data act = 1 WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	MW-IDLE
81	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&AE && act != 0 => Write.rsp (Address, Data, WKC)	MW-IDLE
82	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && act == 0 => (buffer. Address) = Data act = 2 Terminate = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	MW-IDLE
83	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&AE && && act != 1 => Write.rsp (Address, Data, WKC)	MW-IDLE
84	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&E && act == 1 => (buffer. Address) = Data act = 2 Terminate = TRUE WKC = WKC +1 Write.rsp (Address, Data, WKC)	MW-IDLE
85	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&NE && act == 1 => (buffer. Address) = Data WKC = WKC +1 Write.rsp (Address, Data, WKC)	MW-IDLE
86	MW-IDLE	**Write.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MW-IDLE
87	MW-IDLE	**Read.ind (Address, Data, WKC)** => pass next	MW-IDLE

#	État en cours	Événement /Condition =>Action	État suivant
88	MW-IDLE	**Terminate.ind(Success)** /Success && Terminate => Terminate = FALSE act = 3 SM.mailboxState=1 SM.WriteEvent = 1 pass next	MW-IDLE
89	MW-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && act == 1) then act = 0 pass next	MW-IDLE
90	MW-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE act = 0 pass next	MW-IDLE

Fonctions

Les fonctions SYSM sont récapitulées dans le Tableau A.13.

Tableau A.13 – Table de fonctions SYSM

Nom de fonction	Opérations
A	Address == SM.PhysicalStartAddress
NA	Address > SM.PhysicalStartAddress
E	Address + Length == SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter
NE	Address + Length < SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter
AE	Address + Length =< SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter

A.3 RMSM

A.3.1 Définitions de primitives

A.3.1.1 Primitive échangée entre SYSM et RMSM

Le Tableau A.14 représente la primitive émise par le RMSM au SYSM.

Tableau A.14 – Primitives émises par RMSM au SYSM

Nom de primitive	Paramètres associés
DL-Write local request (Demande locale d'écriture DL)	Address, Length, Data

Le Tableau A.15 représente la primitive émise par le SYSM au RMSM.

Tableau A.15 – Primitives émises par SYSM au RMSM

Nom de primitive	Paramètres associés
DL-Write local confirmation (Confirmation locale d'écriture DL)	L-Status

NOTE Les événements locaux ne sont pas modélisés étant donné qu'ils n'ont aucun impact sur le RMSM.

A.3.1.2 Paramètres des primitives SYSM

Le Tableau A.16 représente tous les paramètres utilisés avec les primitives entre le RMSM et le SYSM.

Tableau A.16 – Paramètres utilisés avec les primitives échangées entre RMSM et SYSM

Nom de paramètre	Description
Address	Identificateur de la zone de mémoire physique
Length	Longueur des octets émis
Data	Valeur de l'octet émis

Description du diagramme d'états

Il existe un RMSM par canal du gestionnaire de synchronisation de la boîte aux lettres en lecture.

Le RMSM écrit la boîte aux lettres en lecture à la demande de l'utilisateur. Une seule demande d'utilisateur (Read Upd) est acceptée. Si le maître lit les données, le RMSM les stockera dans une mémoire tampon auxiliaire.

Un changement de basculement donnera lieu à une restauration des anciennes données de boîte aux lettres (même si la boîte aux lettres a été mise à jour entre temps).

Variables locales

Tggl
Contient le drapeau de basculement au dernier événement du gestionnaire de synchronisation.

Boxstate
Signale l'état de la boîte.

BackupBox
Stockage de la PDU de boîte aux lettres pour des besoins de sauvegarde.

ActualBox
Stockage virtuel de la PDU de boîte aux lettres réelle.

SeqN
Contient le numéro de séquence du service suivant.

Nomenclature de la table d'états

Les suffixes normalisés «.req», «.cnf» et «.ind» sont respectivement utilisés pour indiquer les primitives de requête, de confirmation et d'indication.

Table RMSM

La table d'états RMSM est présentée dans le Tableau A.17.

Tableau A.17 – Table d'états RMSM

#	État en cours	Événement /condition ⇒action	État suivant
1	OFF	**Enable SM Event (Toggle)** => Tggl=Toggle Boxstate = 0 SeqN = 0 BackupBox = ERR PDU with no Error(0,0,0,0)	IDLE
2	OFF	**Disable SM Event** =>	OFF
3	IDLE	**Enable SM Event (Toggle)** =>	IDLE
4	IDLE	**Disable SM Event** => Terminate Segmented Services	OFF
5	IDLE	**Toggle SM Event (Toggle)** /Tggl != Toggle =>	IDLE
6	IDLE	**Toggle SM Event (Toggle)** /Tggl == Toggle => ActualBox = BackupBox Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read Boxstate = 1 Tggl = Toggle DL-Write Local.req(Address, Length, Data) write SM status(Toggle)	SEND
7	IDLE	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** => ActualBox = Parameter from event service primitive Update (SeqN) Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read Boxstate = 0 DL-Write Local.req(Address, Length, Data)	SEND
8	IDLE	**DL-Write Local.cnf (L-Status)** => ignore	IDLE
9	SEND	**Enable SM Event (Toggle)** =>	IDLE
10	SEND	**Disable SM Event** => Terminate Segmented Services	OFF
11	SEND	**Toggle SM Event (Toggle)** /Tggl != Toggle =>	SEND
12	SEND	**Toggle SM Event (Toggle)**/Tggl == Toggle && Boxstate == 0=>ActualBox <=> BackupBox (Exchange) Address = SM1.PhysicalStartAddress Length = SM1.LengthData = encode Mailbox ReadBoxstate = 2DL-Write Local.req(Address, Length, Data)write SM status(Toggle)	SEND
13	SEND	**Toggle SM Event (Toggle)** /Tggl == Toggle && Boxstate == 1, 2 => write SM status(Toggle)	SEND

#	État en cours	Événement /condition ⇒action	État suivant
14	SEND	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** /Boxstate == 1 => BackupBox = Parameter from event service primitive Update (SeqN) Boxstate = 2	SEND
15	SEND	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** /Boxstate == 0, 2 => invalid user sequence	SEND
16	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 0 => DL_status = L-Status BackupBox =ActualBox DL-Mailbox Read Upd.cnf (DL_status)	IDLE
17	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 1 => DL_status = L-Status DL-Mailbox Read Upd.cnf (DL_status)	IDLE
18	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 2 => ActualBox = BackupBox Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read DL_status = L-Status Boxstate = 0 DL-Mailbox Read Upd.cnf (DL_status) DL-Write Local.req(Address, Length, Data)	SEND

Fonctions

Les fonctions RMSM sont récapitulées dans le Tableau A.18.

Tableau A.18 – Table de fonctions RMSM

Nom de fonction	Opérations
Update (SeqN)	if (SeqN < 7) then SeqN = SeqN + 1 else SeqN = 1

Bibliographie

CEI 61131-2, *Automates programmables – Partie 2: Exigences et essais des équipements*

CEI 61131-3, *Automates programmables – Partie 3: Langages de programmation*

CEI 61158-1:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 1: Vue d'ensemble et lignes directrices des séries CEI 61158 et CEI 61784*

CEI 61158-2:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 2: Spécification et définition des services de la couche physique*

CEI 61158-5-12:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-12: Définition des services de la couche application – Éléments de type 12*

CEI 61158-6-12, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-12: Spécification du protocole de la couche application – Éléments de type 12*

CEI 61784-1, *Réseaux de communication industriels – Profils – Partie 1: Profils de bus de terrain*

CEI 61784-2, *Réseaux de communication industriels – Profils – Partie 2: Profils de bus de terrain supplémentaires pour les réseaux en temps réel basés sur l'ISO/CEI 8802-3*

ISO/IEC/TR 8802-1, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 1: Overview of Local Area Network Standards* (disponible en anglais seulement)

ISO/IEC 15802-1, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Common specifications – Part 1: Medium Access Control (MAC) service definition* (disponible en anglais seulement)

UIT-T V.41, *Système de protection contre les erreurs indépendant du code utilisé*

ANSI X3.66 (R1990), *Advanced data communication control procedures (ADCCP)*

IEEE 802.1D, *IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Bridges,* disponible à l'adresse <http://www.ieee.org>

IETF RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II,* disponible à l'adresse <http://www.ietf.org>

IETF RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types,* disponible à l'adresse <http://www.ietf.org>

_____

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

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

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