EtherCAT Specification – Part 4

Data Link Layer protocols specification

Document: ETG.1000.4 S (R) V1.0.4

Nomenclature:

ETG-Number	ETG.1000.4
Type	S (Standard)
State	R (Release)
Version	V1.0.4

Created by: ETG
Contact: info@ethercat.org
Date 15.09.2017

EtherCAT Technology Group

Trademarks and Patents

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany. Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Disclaimer

The documentation has been prepared with care. The technology described is, however, constantly under development. For that reason the documentation is not in every case checked for consistency with performance data, standards or other characteristics. In the event that it contains technical or editorial errors, we retain the right to make alterations at any time and without warning. No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Copyright

DOCUMENT HISTORY

Version	Comment
1.0	Adopted from IEC-Standard 61158-4 Type 12 for ETG use only!
1.0.1	Corrections and clarifications according to ETG1000_ES_D_V0i6_EcatSpecErrata.doc
1.0.2	Corrections and clarifications according to ETG1000_V1i0i2_ES_R_V0i7_EcatSpecErrata.doc and IEC SC65C MT 9 editorial comments
1.0.3	Corrections and clarifications according to ETG1000_V1i0i2_ES_R_V0i8_EcatSpecErrata.doc
1.0.4	Corrections and clarifications according to ETG1000_V1i0i3_ES_R_V0i9_EcatSpecErrata.doc

CONTENTS

Figures

Tables

1 Scope

1.1 Scope of this standard and accordance to IEC Standards

The ETG.1000 series specifies the EtherCAT Technology within the EtherCAT Technology group. It is devided in the following parts:

- ETG.1000.2: Physical Layer service definition and protocol specification
- ETG.1000.3: Data Link Layer service definition
- ETG.1000.4: Data Link Layer protocol specification
- ETG.1000.5: Application Layer service definition
- ETG.1000.6: Application Layer protocol specification

These parts are leant on the corresponding parts of the IEC 61158 series Type 12. EtherCAT is named Type 12 in IEC 61158 to avoid the usage of brand names.

1.2 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.

1.3 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.

1.4 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.

1.5 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.

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

IEC 61158-2:2010, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-3-12, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 3-12: Data link service definition – Type 12 elements*

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

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, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet*

ISO/IEC 9899, *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*; 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 and 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 [7498-1]

3.1.2 DL-protocol [7498-1]

3.1.3 DL-protocol-data-unit [7498-1]

3.1.4 (N)-entity [7498-1]
DL-entity
Ph-entity

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

3.1.6 (N)-layer [7498-1]
DL-layer (N=2)
Ph-layer (N=1)

3.1.7 (N)-service [7498-1]
DL-service (N=2)
Ph-service (N=1)

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

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

3.1.10 peer-entities [7498-1]

3.1.11 Ph-interface-data [7498-1]

3.1.12 primitive name [7498-3]

3.1.13 reassembling [7498-1]

3.1.14 recombining [7498-1]

3.1.15 reset [7498-1]

3.1.16 routing [7498-1]

3.1.17 segmenting [7498-1]

3.1.18 sequencing [7498-1]

3.1.19 splitting [7498-1]

3.1.20 systems-management [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 part of ETG.1000, 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. A single DL-entity may have multiple group DL-addresses associated with a single DLSAP. 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 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 EtherCAT definitions

3.4.1

application
function or data structure for which data is consumed or produced [IEC 61158-5:2003]

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. 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.) [IEC 61158-2:2003]

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 "media" is used as the plural of medium.

3.4.22

message
ordered series of octets intended to convey information

NOTE 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 [derived from IEC 61158-2]

3.4.25

object
abstract representation of a particular component within a device

NOTE 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 EtherCAT symbols and abbreviations

AL	Application layer
AoE	Automation Device Specification (ADS) over EtherCAT
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 EtherCAT services
CSMA/CD	Carrier sense multiple access with collision detection
DC	Distributed clocks
DCSM	DC state machine
DHSM	(DL) PDU handler state machine
EtherCAT	Prefix for DL services and protocols
E²PROM	Electrically erasable programmable read only memory
EoE	Ethernet tunneled over EtherCAT services
ESC	EtherCAT slave controller
FCS	frame check sequence
FMMU	Fieldbus memory management unit
FoE	File access with EtherCAT 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)
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 ETG.1000.3. The service specification defines the services that are provided by the EtherCAT 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
EtherCAT 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	0: last EtherCAT PDU 1: EtherCAT 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. DWORD is a four octet double word.

```
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 EtherCAT DL/PDI shows the access right to this element. R means read access right, W means write access right. If neither EtherCAT 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	0: no acknowledge 1: acknowledge (must 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 unless 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.

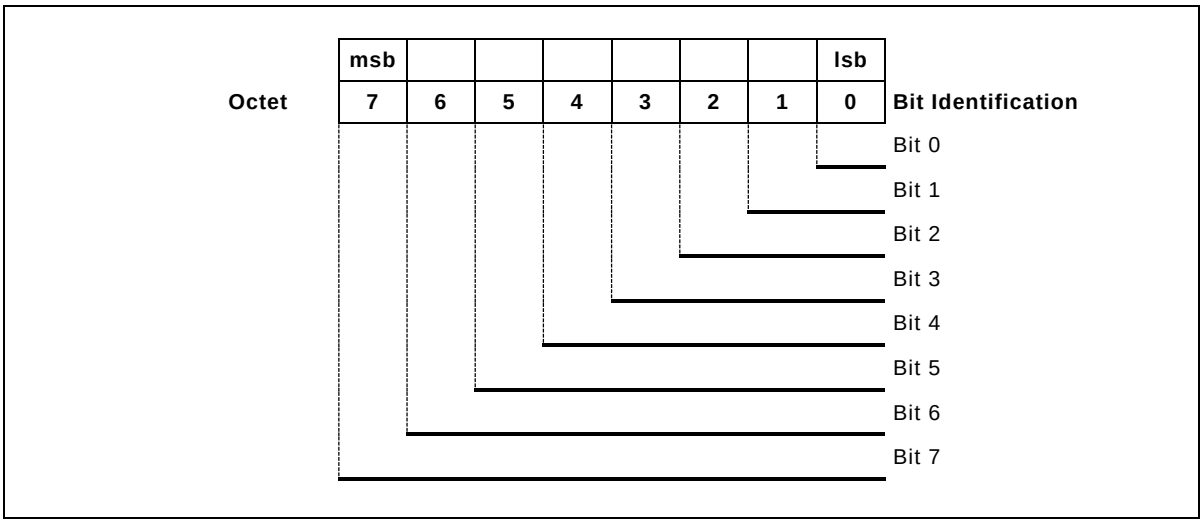


Figure 2 – Common structure of specific fields

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 1: 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

EtherCAT 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, an EtherCAT 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 EtherCAT 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 EtherCAT 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 an EtherCAT 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 EtherCAT DL can be implemented using standard Ethernet controllers without direct processing. The influence of the forwarding mechanism implementation on communication performance is detailed in IEC 61784-2.

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

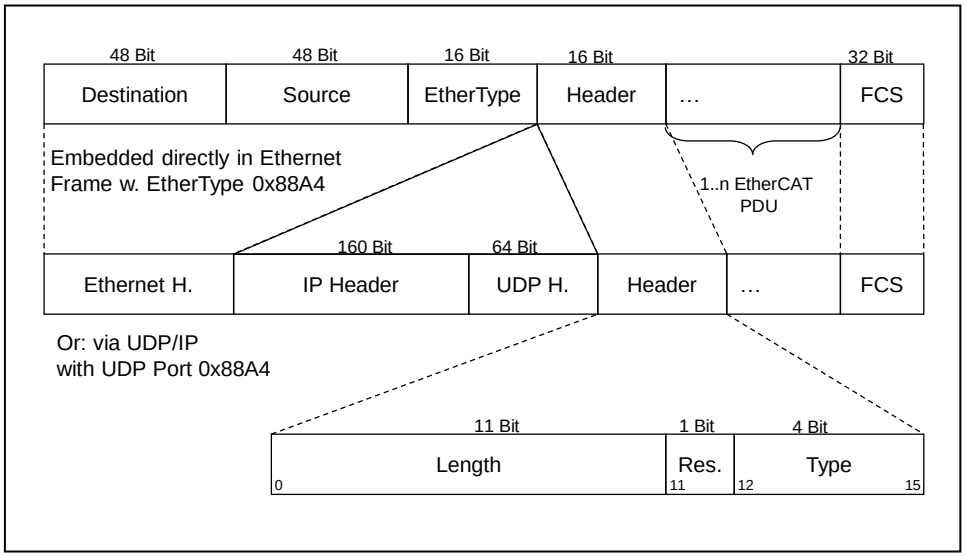


Figure 3 – Frame structure

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 EtherCAT 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 EtherCAT segment.

NOTE Further addressing details are given in the data-link layer service definition(ETG.1000.3)

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 EtherCAT PDUs. The EtherCAT PDUs are packed without gaps. The frame is terminated with the last EtherCAT PDU, unless the frame size is less than 64 octets, in which case the frame is 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 EtherCAT 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 an EtherCAT 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.

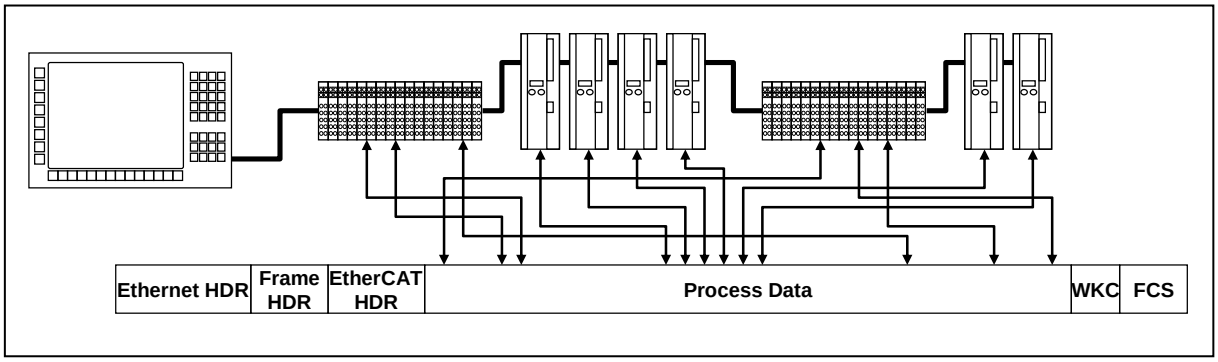


Figure 4 – Mapping of data in a frame

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 EtherCAT PDU. The master can assemble completely sorted logical process images via a single EtherCAT 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. EtherCAT 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

EtherCAT 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 an EtherCAT PDU, the addressed slave increments a working counter (WKC) positioned at the end of each EtherCAT PDU. Analyzing the working counter allows the master to check if the expected number of nodes has processed the corresponding EtherCAT PDU.

4.6 Node reference model

4.6.1 Mapping onto OSI basic reference model

EtherCAT 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 EtherCAT 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 EtherCAT DL data-link layer or the EtherCAT DL Application layer. Likewise, features common to users of the Fieldbus Application layer may be provided by the EtherCAT DL Application layer to simplify user operation, as noted in Figure 5.

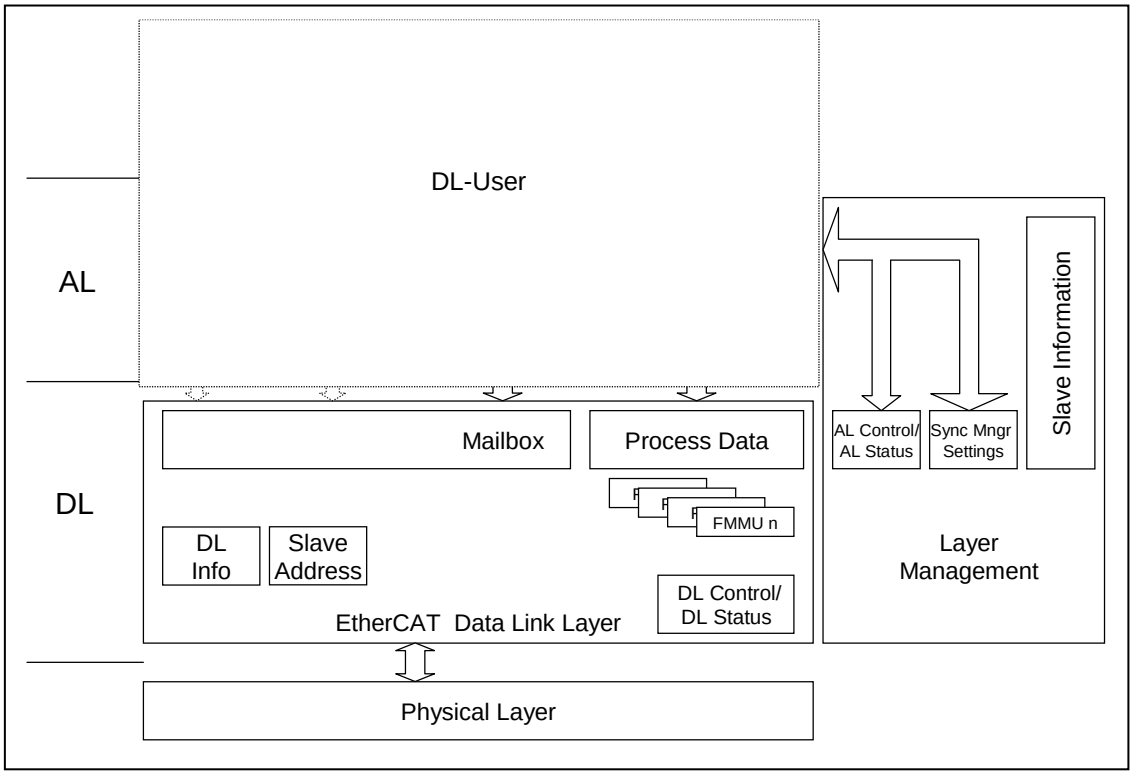


Figure 5 – Slave node reference model

4.6.2 Data-link Layer features

The data link layer provides basic time critical support for data communications among devices connected via EtherCAT 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

An EtherCAT Ethernet frame contains one or several EtherCAT PDUs (as shown in Figure 6), each addressing individual devices and/or memory areas. The EtherCAT frame is recognized by the combination of the EtherType 0x88A4[1] and the corresponding EtherCAT 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 EtherCAT frame header. Fragmentation of IP packets will be ignored. The UDP checksum shall be set to 0 by Slaves and could be

[1] The EtherType 0x88A4 was assigned for EtherCAT by the IEEE Registration Authority.

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

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

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

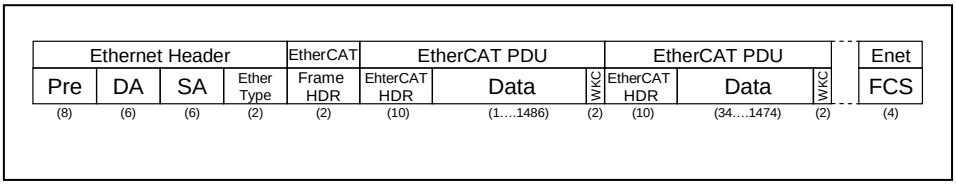


Figure 6 – EtherCAT PDUs embedded in Ethernet frame

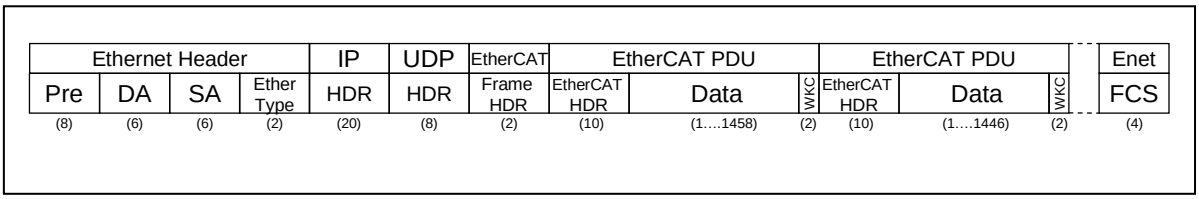


Figure 7 – EtherCAT PDUs embedded in UDP/IP

5 Frame structure

5.1 Frame coding principles

EtherCAT DL uses a standard ISO/IEC 8802-3 Ethernet frame structure for transporting EtherCAT PDUs. The PDUs may alternatively be sent via UDP/IP. The EtherCAT 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 EtherCAT 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

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

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

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

and, performing two's complement arithmetic,

Integern(*b*) = - Integern(^b) - 1 if $b_{n-1} = 1$

Note that 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 Ethernet DLPDU structure

5.3.1 EtherCAT frame inside an Ethernet frame

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

Table 9 – EtherCAT 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 (EtherCAT)
	EtherCAT 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 EtherCAT frame inside a UDP datagram

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

Table 10 – EtherCAT 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) not checked within EtherCAT segment
	TotalLength	Unsigned16	(IP total length of service) - not checked within EtherCAT segment
	Identification	Unsigned16	(IP identification packet for fragmented service) - not checked within EtherCAT segment
	Flags	BYTE	Should not be fragmented (IP flags – they will not be considered but a fragmentation of EtherCAT frame will result in an error) - not checked within EtherCAT segment
	Fragments	BYTE	Should be 0 (IP fragment number - fragmentation of EtherCAT frame will result in an error) - not checked within EtherCAT segment
	Ttl	BYTE	(IP time to live – only checked at routers) - not checked within EtherCAT segment
	Protocol	BYTE	0x11 (IP sub-protocol – this value is reserved for UDP)
	Header checksum	Unsigned16	(IP header checksum) - not checked within EtherCAT segment
	Source IP address	BYTE[4]	(IP source address of the originator) - not checked within EtherCAT segment
	Destination IP address	BYTE[4]	(IP destination address of the target of the frame – within a EtherCAT segment usually a multicast address as an individual address requires the Address Resolution Protocol ARP) - not checked within EtherCAT segment
UDP	Src port	WORD	(UDP Source Port) - not checked within EtherCAT segment
	Dest port	WORD	0x88A4 (UDP Destination Port)
	Length	WORD	(UDP length of frame) - not checked within EtherCAT segment
	Checksum	WORD	(UDP checksum of frame) – will be set to 0 for EtherCAT frames but without checking
	EtherCAT frame		specified in 5.3.3
	Ethernet 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 EtherCAT DL-protocol

5.3.3 EtherCAT frame structure

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

Table 11 – EtherCAT frame structure containing EtherCAT PDUs

Frame part	Data field	Data type	Value/description
EtherCAT Frame	Length	Unsigned11	Length of this frame (minus 2 octets)
	Reserved	Unsigned1	0
	Type	Unsigned4	Protocol Type = EtherCAT DLPDUs (0x01)
	EtherCAT PDU 1		specified in 5.4
	…		specified in 5.4
	EtherCAT PDU n		specified in 5.4

Table 12 – EtherCAT frame structure containing network variables

Frame part	Data field	Data type	Value/description
EtherCAT frame	Length	Unsigned11	Length of this frame (minus 2 octets)
	reserved	Unsigned1	0
	Type	Unsigned4	Protocol type = network variables (0x04)
Publisher header	PubID	BYTE[6]	Publisher ID
	CntNV	Unsigned16	Number of Network variables contained in this EtherCAT frame
	CYC	Unsigned16	Cycle Number of the publisher side
	reserved	BYTE[2]	0x00, 0x00
	Network variable 1		specified in 5.4.4
	…		specified in 5.4.4
	Network variable n		specified in 5.4.4

Table 13 – EtherCAT frame structure containing mailbox

Frame part	Data field	Data type	Value/description
EtherCAT frame	Length	Unsigned11	Length of this frame (minus 2 octets)
	reserved	Unsigned1	0
	Type	Unsigned4	Protocol type = mailbox (0x05)
	Mailbox		specified in 5.6

5.4 EtherCAT DLPDU structure

5.4.1 Read

5.4.1.1 Overview

With the read services a master reads data from register or memory of one or several slaves. The working counter shall be incremented by each slave if at least one of the addressed attributes 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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 several slaves. The working counter is incremented if the addressed attribute is present and 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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 Overview

With the read write services, a master reads data from register or memory of one or several slaves and writes data to register or memory of one or several slaves. The working counter shall be incremented by each slave if at least one of the addressed attributes is present for read and in addition if at least one part of the data can be written.

5.4.3.2 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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.

5.4.3.4 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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.

5.4.3.5 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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 additionally incremented by one if a piece of data was successfully read.

5.4.3.6 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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.

5.4.3.7 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
EtherCAT 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	0: last EtherCAT PDU in EtherCAT frame 1: EtherCAT PDU in EtherCAT 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 EtherCAT 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.

5.4.4 Attributes access

For the read or write services to addressed attributes, the following restrictions are defined:

- a master can read and write defined attributes (see clause 6).
- a master shall write reserved bits of attributes with the value Zero.
- a slave shall return the value Zero for read access to reserved bits of attributes.
- a master shall ignore the value read from reserved bits of attributes.
- a master should not write to reserved attributes (neither as a separate access nor as part of an access to defined attributes).
- a master may not read from reserved attributes (neither as a separate access nor as part of an access to defined attributes).

NOTE: The last two rules are defined to handle legacy EtherCAT Slave Controllers in a compatible way.

Reserved attributes are defined as

- attributes that are not specified or specified as reserved,
- attributes that are defined as optional and not implemented in the slave controller (ESC),
- for write-access: attributes that are defined as read-only, or
- for read access: attributes that are defined as write-only

NOTE: Reserved bits and attributes according to the specification at the time of master/slave implementation.

NOTE: A master can check the availability of attributes by an individual read access of the attribute and checking the WKC=1.

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 EtherCAT mailbox structure

The mailbox coding is specified in Table 29. The mailbox encoding shall be used in conjunction with EtherCAT 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 EtherCAT segment
	Channel	Unsigned6	0x00 (Reserved)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x00: error(ERR) 0x01: ADS over EtherCAT (AoE) 0x02: Ethernet over EtherCAT (EoE) 0x03: CAN application protocol over EtherCAT (CoE) 0x04: File Access over EtherCAT (FoE) 0x05: Servo profile over EtherCAT (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	0
	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 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 0x09: MBXERR_SERVICEINWORK Mailbox service already in use

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 available for EBUS ports.

Enhanced link detection MII

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

Separate Handling of FCS errors

This parameter shall indicate that the errors induced by another EtherCAT 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      NoSupportReservedRegister: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	0x01-0x10
Number of supported Sync Manager channels	0x0005	BYTE	R	R	0x01-0x10
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 supported This feature bit does not affect mappability of SM.WriteEvent flag (MailboxIn)
NoSupportReservedRegister	0x0008	Unsigned1	R	R	1: shall only be used for legacy reasons. Reserved registers may not be written, reserved registers may not be read when out of register area (refer to documentation of specific slave controller (ESC) 0: no restriction in register access
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 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

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
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	--	--	

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 EtherCAT 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 modification, The source MAC address is not changed for any frame 1: EtherCAT frames are processed, non-EtherCAT frames are destroyed, The source MAC address is changed by the Processing Unit for every frame (SOURCE_MAC[1] is 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	--	--	
Loop control port 0	0x0101	Unsigned2	RW	R	0: Auto => closed at “link down”, open with “link up” 1: Auto close => loop closed at “link down”, opened with writing 01 after “link up” (or receiving a valid Ethernet frame at the closed port) 2: Always open 3: Always closed

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Loop control port 1	0x0101	Unsigned2	RW	R	0: Auto => closed at “link down”, open with “link up” 1: Auto close => loop closed at “link down”, opened with writing 01 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 => loop closed at “link down”, opened with writing 01 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 => loop closed at “link down”, opened with writing 01 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	--	--	
EnableAliasAddress	0x0103	Unsigned1	RW	R	0: Disable the station alias address 1: Enable the station alias address
reserved	0x0103	Unsigned7	--	--	
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	0: DLS-user not operational 1: DLS-user operational
DLS-user watchdog status	0x0110	Unsigned1	R	R	0: DLS-user watchdog expired 1: DLS-user watchdog not expired
Extended link detection	0x0110	Unsigned1	R	R	0: Deactivated 1: Activated for at least one port
Reserved	0x0110	Unsigned1	--	--	
Link status port 0	0x0110	Unsigned1	R	R	0: no physical link on this port 1: physical link on this port
Link status port 1	0x0110	Unsigned1	R	R	0: no physical link on this port 1: physical link on this port
Link status port 2	0x0110	Unsigned1	R	R	0: no physical link on this port 1: physical link on this port
Link status port 3	0x0110	Unsigned1	R	R	0: no physical link on this port 1: physical link on this port
Loop status port 0	0x0111	Unsigned1	R	R	0: loop not active 1: loop active
Signal detection port 0	0x0111	Unsigned1	R	R	0: signal not detected on RX-port 1: signal detected on RX-port
Loop status port 1	0x0111	Unsigned1	R	R	0: loop not active 1: loop active
Signal detection port 1	0x0111	Unsigned1	R	R	0: signal not detected on RX-port 1: signal detected on RX-port
Loop status port 2	0x0111	Unsigned1	R	R	0: loop not active 1: loop active
Signal detection port 2	0x0111	Unsigned1	R	R	0: signal not detected on RX-port 1: signal detected on RX-port
Loop status port 3	0x0111	Unsigned1	R	R	0: loop not active 1: loop active
Signal detection port 3	0x0111	Unsigned1	R	R	0: signal not detected on RX-port 1: 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).

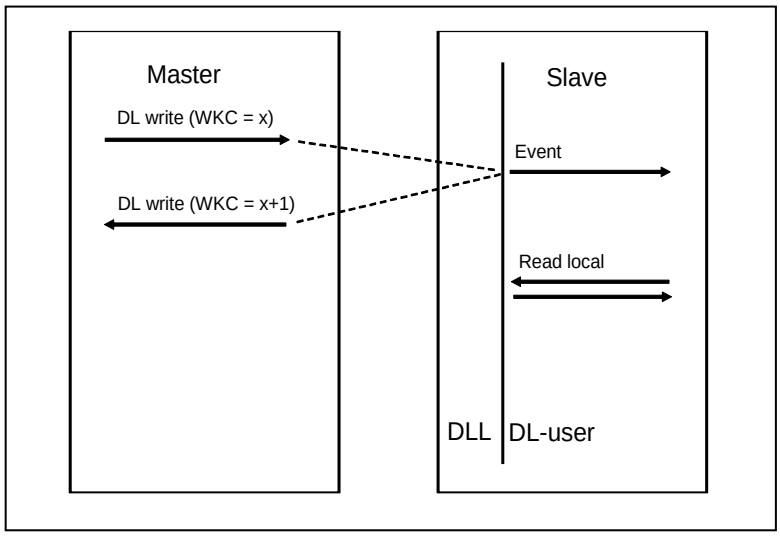


Figure 12 – Successful write sequence to DL-user control register

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).

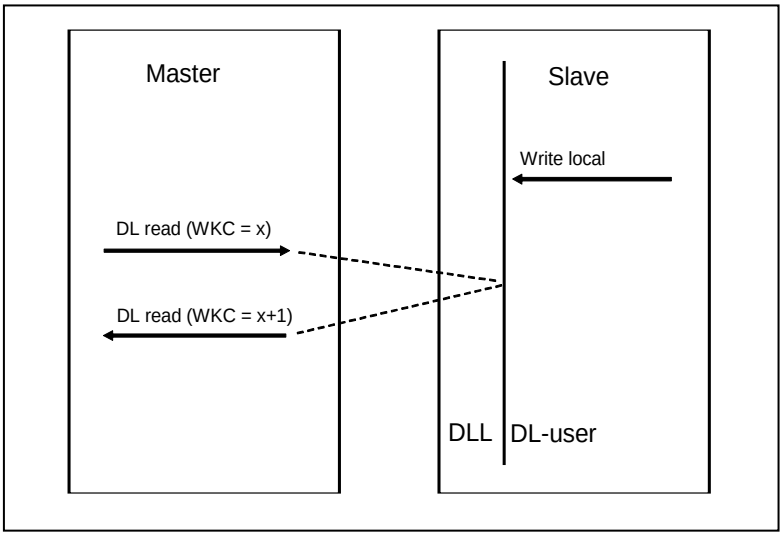


Figure 13 – Successful read sequence to the DL-user status register

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
DLS-user R5	0x0132	Unsigned16	R	RW	0x00
DLS-user R6	0x0134	Unsigned16	R	RW	0x00
DLS-user R7	0x0140	Unsigned8	R	R	0x00
Copy	0x0141	Unsigned1	R	R	0: no specific action 1: Copy DLS-user R1 to DLS-user R3
DLS-user R9	0x0141	Unsigned7	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 a 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 a write service to an application memory area configured as write by the master or a 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 an EtherCAT 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 an EtherCAT read to DL-user status register is invoked.

Sync manager channel access events (0 to 7)

This parameter is set if a write service to an application memory area configured as write by the slave or a 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 EtherCAT services.

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	0: no event active 1: event active R1 was written
DC event 0	0x0220	Unsigned1	R	R	0: no event active 1. event active (at least one latch event occurred)
DC event 1	0x0220	Unsigned1	R	R	0: sync signal is not active 1. sync signal is active
DC event 2	0x0220	Unsigned1	R	R	0: sync signal is not active 1. sync signal is active
Sync manager change event	0x0220	Unsigned1	R	R	0: no event active 1: event active (one or more Sync manager channels were changed)
EEPROM Emulation	0x0220	Unsigned1	R	R	0: No command pending 1: EEPROM command pending
DLE specific	0x0220	Unsigned2	R	R	0x00
Sync manager channel 0 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 1 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 2 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 3 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 4 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 5 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 6 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 7 event	0x0221	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Sync manager channel 8 event	0x0222	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 9 event	0x0222	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 10 event	0x0222	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 11 event	0x0222	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 12 event	0x0222	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 13 event	0x0222	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 14 event	0x0222	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed)
Sync manager channel 15 event	0x0222	Unsigned1	R	R	0: no event active 1: 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 in 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 EtherCAT 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	0: no event active 1: DC event 0 active
Reserved	0x0210	Unsigned1	--	--	
DL Status change	0x0210	Unsigned1	R	R	0: no event active 1: DL status register was changed active
R3 or R4 Chg	0x0210	Unsigned1	R	R	0: no event active 1: event active (R3 or R4 was written)
Sync manager channel 0 event	0x0210	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed by slave)
Sync manager channel 1 event	0x0210	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed by slave)
Sync manager channel 2 event	0x0210	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed by slave)
Sync manager channel 3 event	0x0210	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed by slave)
Sync manager channel 4 event	0x0211	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed by slave)
Sync manager channel 5 event	0x0211	Unsigned1	R	R	0x000 no event active 1: event active (Sync manager channel was accessed by slave)
Sync manager channel 6 event	0x0211	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed by slave)
Sync manager channel 7 event	0x0211	Unsigned1	R	R	0: no event active 1: event active (Sync manager channel was accessed by slave)
Reserved	0x0211	Unsigned4	--	--	

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 for each counter whose maximum value (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 of the group
Physical error count port 0	0x0301	Unsigned8	RW	--	A write to one counter will reset all counters of the group
Frame error count port 1	0x0302	Unsigned8	RW	--	A write to one counter will reset all counters of the group
Physical error count port 1	0x0303	Unsigned8	RW	--	A write to one counter will reset all counters of the group
Frame error count port 2	0x0304	Unsigned8	RW	--	A write to one counter will reset all counters of the group
Physical error count port 2	0x0305	Unsigned8	RW	--	A write to one counter will reset all counters of the group
Frame error count port 3	0x0306	Unsigned8	RW	--	A write to one counter will reset all counters of the group
Physical error count port 3	0x0307	Unsigned8	RW	--	A write to one counter will reset all counters of the group

6.2.2 Lost link counter

The optional lost link counter registers contain information about link down sequences. All lost link counters will be cleared if one counter is written. The counting is stopped for each counter whose maximum value (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 lost link counters
Lost link count port 1	0x0311	Unsigned8	RW	R	A write to one counter will reset all lost link counters
Lost link count port 2	0x0312	Unsigned8	RW	R	A write to one counter will reset all lost link counters
Lost link count port 3	0x0313	Unsigned8	RW	R	A write to one counter will reset all lost link 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 previous error counters will be cleared if one counter is written. The counting is stopped for each counter whose maximum value (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 Malformat Frame counter counts frames with i.e. wrong datagram structure. The counter will be cleared if the counter is written. The counting is stopped when the maximum value (255) is reached.

Parameter

Malformat frame counter

This parameter counts the occurrences of wrong EtherCAT frames.

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

Parameter

Local problem counter

This parameter counts the occurrences of communication problems within a slave. The counter will be cleared if the counter is written. The counting is stopped when the maximum value (255) is reached.

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 Previous Error counters
Previous Error Count Port 1	0x0309	Unsigned8	RW	R	A write to one counter will reset all Previous Error counters
Previous Error Count Port 2	0x030A	Unsigned8	RW	R	A write to one counter will reset all Previous Error counters
Previous Error Count Port 3	0x030B	Unsigned8	RW	R	A write to one counter will reset all Previous Error 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 40ns intervals (minus 2) that represents the basic watchdog increment (default value is 100µs = 2498).

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

Parameter

DLS-user watchdog

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

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

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

Figure 18 – DLS-user Watchdog 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	--	--	

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: EtherCAT DL 1: PDI
Lock	0x0500	Unsigned1	RW	R	Reset Access to SII 0: no action 1: cancel access Setting this bit will reset Register 0x0501.0
reserved	0x0500	Unsigned6	--	--	
Access PDI	0x0501	Unsigned1	R	RW	0: no access 1: PDI access active
reserved	0x0501	Unsigned7	--	--	

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 write access

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

Slave information interface assign

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

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 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 that 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	0: only read access to SII 1: read and write access to SII
reserved	0x0502	Unsigned4	--	--	
EEPROM emulation	0x0502	Unsigned1	R	R	0: Normal operation (DL interfaces to SII) 1: DL-user emulates SII
SII Read Size	0x0502	Unsigned1	R	R	0: 4 octet read with one transaction 1: 8 octet read with one transaction
SII Address Algorithm	0x0502	Unsigned1	R	R	0: 1 octet used as address 1: 2 octets used as address
Read operation	0x0503	Unsigned1	RW	RW	0: no read operation requested (parameter write) or read operation not busy (parameter read) 1: read operation requested (parameter write) or read operation busy (parameter read) To start a new read operation there must be a positive edge on this parameter
Write operation	0x0503	Unsigned1	RW	RW	0: no write operation requested (parameter write) or write operation not busy (parameter read) 1: write operation requested (parameter write) or write operation busy (parameter read) To start a new write operation there must be a positive edge on this parameter
Reload operation	0x0503	Unsigned1	RW	RW	0: no reload operation requested (parameter write) or reload operation not busy (parameter read) 1: reload operation requested (parameter write) or reload operation busy (parameter read) To start a new reload operation there must be a positive edge on this parameter
Checksum Error	0x0503	Unsigned1	R	R	0: no checksum error loading DL-user information at startup 1: checksum error while reading at startup

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Device info error	0x0503	Unsigned1	R	R	0: no error on reading Device Information at start-up 1: error on reading Device Information
Command error	0x0503	Unsigned1	R	R(W)	0: no error on last command 1: error on last command PDI Write only in SII emulation mode
Write error	0x0503	Unsigned1	R	R	0: no error on last write operation 1: error on last write operation
Busy	0x0503	Unsigned1	R	R	0: operation is finished 1: operation is ongoing

6.4.4 Slave information interface address

The slave information interface address register contains the 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 slave information interface address coding is specified in Table 50.

Table 50 – Slave information interface address

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Address	0x0504	DWORD	RW	RW	32-Bit word address Only the lower 16 Bit (0x0504-0x0505) will be used.

6.4.5 Slave information interface data

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

Table 51 – 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	0: only read access to MII 1: read and write access to MII
Access PDI	0x0510	Unsigned1	R	R	0: Only ECAT 1: PDI access possible
Link Detection via MII management interface	0x0510	Unsigned1	R	R	0: Not active 1: Active
PHYoffset	0x0510	Unsigned5	R	R	0x00 (Default) offset to be added to MII address. Set up by local configuration
Read operation	0x0511	Unsigned1	RW	R	0: no read operation requested (parameter write) or read operation not busy (parameter read) 1: read operation requested (parameter write) or read operation busy (parameter read) To start a new read operation there must be a positive edge on this parameter
Write operation	0x0511	Unsigned1	RW	R	0: no write operation requested (parameter write) or write operation not busy (parameter read) 1: write operation requested (parameter write) or write operation busy (parameter read) To start a new write operation there must be a positive edge on this parameter
reserved	0x0511	Unsigned3	--	--	

Parameter	Physical address	Data type	Access type	Access type PDI	Value/description
Read error	0x0511	Unsigned1	R	R	0: no error on last read operation 1: error on last read operation
Write error	0x0511	Unsigned1	R	R	0: no error on last write operation 1: error on last write operation
Busy	0x0511	Unsigned1	R	R	0: operation is finished 1: operation is ongoing

6.5.2 MII address

The MII address register contains the 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 ISO/IEC 8802-3 clause 22.

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 MII address coding is specified in Table 53.

Table 53 – 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 MII data

The 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 MII data coding is specified in Table 54.

Table 54 – 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 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	--	--	
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	--	--	

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 EtherCAT 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 the most significant bit does not imply an ordering scheme on transmission line.

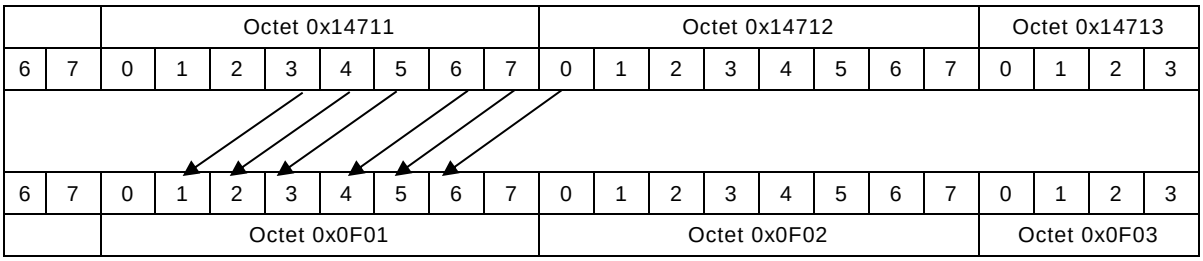


Figure 30 – FMMU mapping example

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

An 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	--	--	
Logical end bit	0x0007	Unsigned3	RW	R	
reserved	0x0007	Unsigned5	--	--	
Physical start address	0x0008	WORD	RW	R	
Physical start bit	0x000A	Unsigned3	RW	R	
reserved	0x000A	Unsigned5	--	--	
Read enable	0x000B	Unsigned1	RW	R	0: entity will be ignored for read service 1: entity will be used for read service
Write enable	0x000B	Unsigned1	RW	R	0: entity will be ignored for write service 1: entity will be used for write service
reserved	0x000B	Unsigned6	--	--	
Enable	0x000C	Unsigned1	RW	R	0: entity not active 1: entity active
reserved	0x000C	Unsigned7	--	--	
reserved	0x000D	Unsigned24	--	--	

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 write command is valid. The principle of interaction is shown in Figure 32.

The exchange buffers actions are linked to the first octet and to 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 the buffer for reading
- the buffer state will be set to empty by reading out the last octet of the buffer

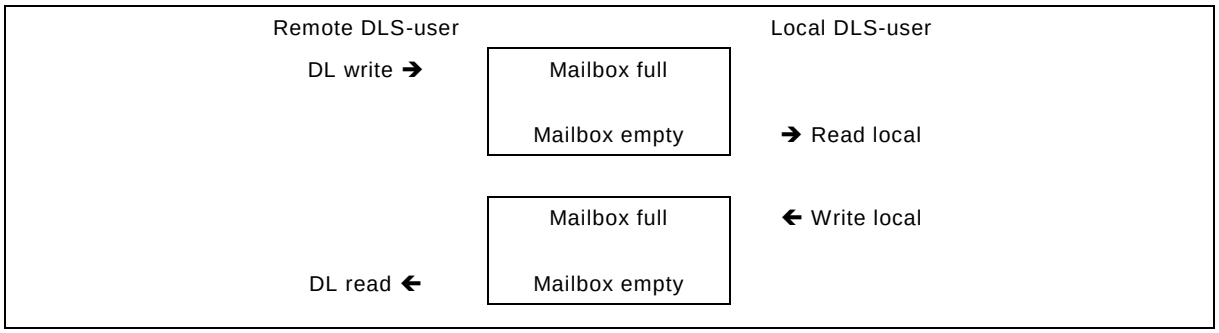
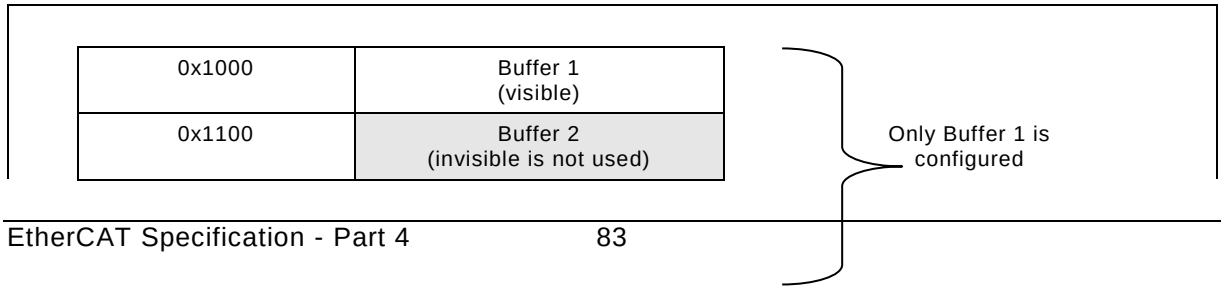


Figure 32 – SyncM mailbox interaction

If a mailbox is full, it can not be written again until it is read out (i.e. last octet of mailbox is 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 tripled. Figure 33 demonstrates a configuration with start address of 0x1000 and length of 0x100. The other buffers are invisible - access always uses 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).



0x1200	Buffer 3 (invisible is not used)
0x1300	Next buffer

Figure 33 – SyncM buffer allocation

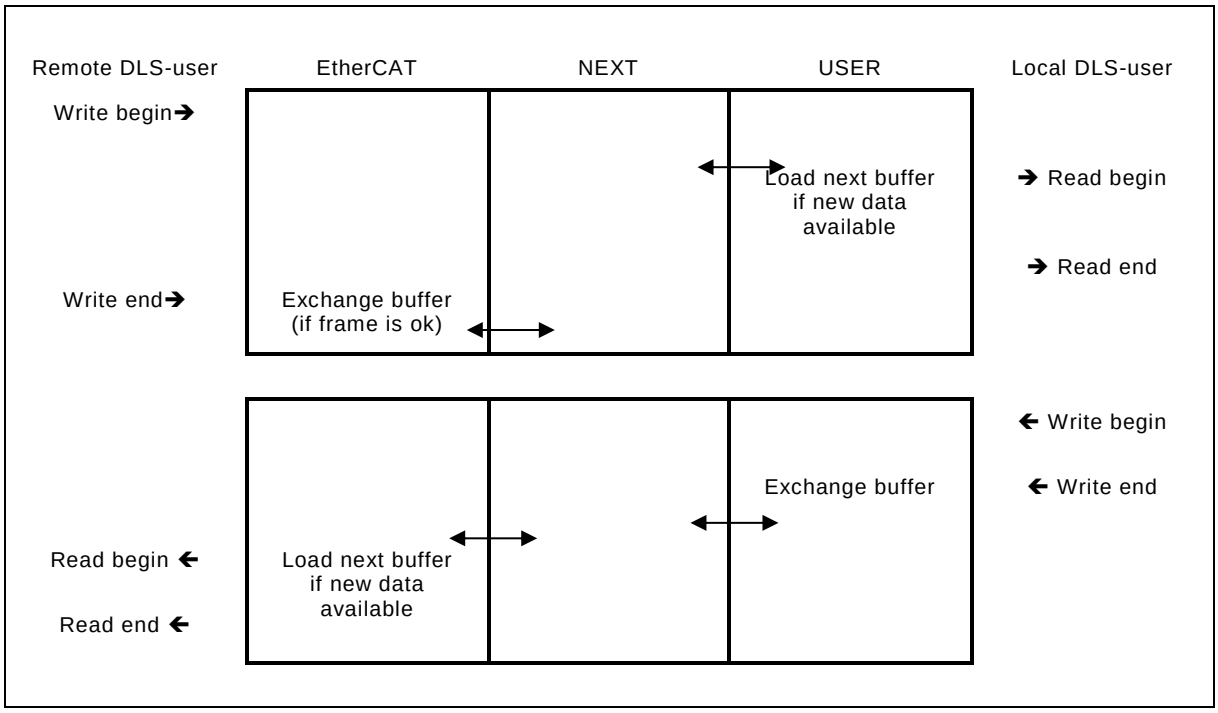


Figure 34 – SyncM buffer interaction

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.

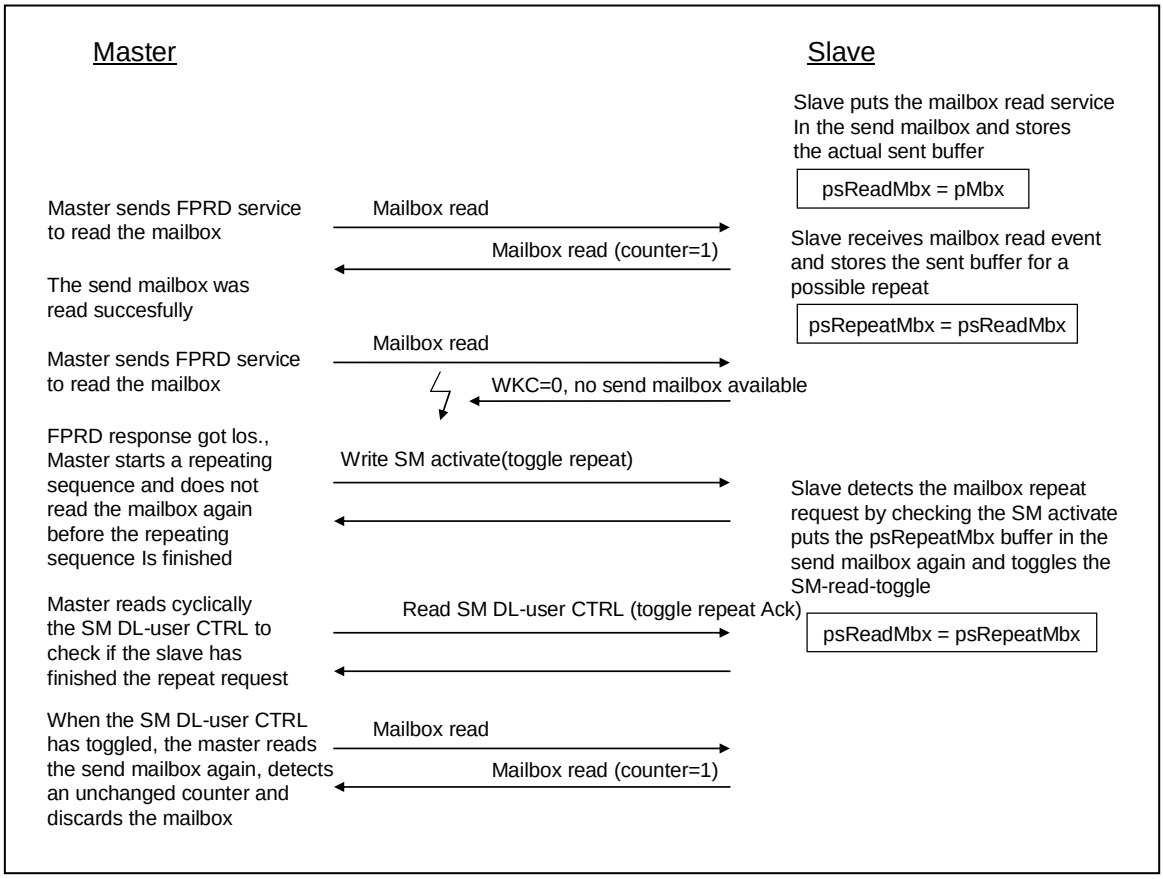


Figure 35 – Handling of write/read toggle with read mailbox

The toggle bits are used to resynchronize mailbox communication if a EtherCAT 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 read buffer state

Write buffer state

This optional parameter indicates the 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 EtherCAT write

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

Deactivate Channel 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	0: event is not active 1: event is active
DLS-user event enable	0x0004	Unsigned1	RW	R	0: DLS-user event is not active 1: DLS-user event is active
Watchdog enable	0x0004	Unsigned1	RW	R	0: watchdog disabled 1: watchdog enabled
reserved	0x0004	Unsigned1	--	--	
Write event	0x0005	Unsigned1	R	R	0: no write event 1: write event
Read event	0x0005	Unsigned1	R	R	0x00: no read event 1: read event

reserved	0x0005	unsigned1	--	--	
Mailbox state	0x0005	Unsigned1	R	R	0: mailbox empty 1: 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	0: read buffer is not open 1: read buffer is open
Write buffer state	0x0005	Unsigned1	R	R	0: write buffer is not open 1: write buffer is open
Channel enable	0x0006	Unsigned1	RW	R	0: channel disabled 1: channel enabled
Repeat	0x0006	Unsigned1	RW	R	
reserved	0x0006	Unsigned4	--	--	
DC Event 0 with Bus access	0x0006	Unsigned1	RW	R	0: no Event 1: DC Event if master completes buffer access
DC Event 0 with local access	0x0006	Unsigned1	RW	R	0: no Event 1: DC Event if DL-user completes buffer access
Deactivate Channel PDI	0x0007	Unsigned1	R	RW	0: channel enabled 1: channel disabled
RepeatAck	0x0007	Unsigned1	R	RW	shall follow repeat after data recovery
reserved	0x0007	Unsigned6	--	--	

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

NOTE 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 ETG.1000.5.

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 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)

Parameter	Physical address (offset)	Data type	Access type	Access type PDI	Value/description
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	--	--	
DC user P1	0x0981	BYTE	RW	R	Implementation Specific
DC user P2	0x0982	Unsigned16	R	R	Implementation Specific
DC user P13	0x0984	BYTE	R	R	Implementation Specific
DC user P14	0x0985	BYTE	R	R	Implementation Specific
reserved	0x0986	BYTE[8]	--	--	
DC user P3	0x098E	Unsigned16	R	R	Implementation Specific
DC user P4	0x0990	DWORD	RW	R	Implementation Specific
reserved	0x0994	BYTE[12]	--	--	
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]	--	--	
DC user P8	0x09AE	Unsigned16	R	R	Implementation Specific
DC user P9	0x09B0	DWORD	R	R	Implementation Specific
reserved	0x09B4	BYTE[4]	--	--	
DC user P10	0x09B8	DWORD	R	R	Implementation Specific
reserved	0x09BC	BYTE[4]	--	--	
DC user P11	0x09C0	DWORD	R	R	Implementation Specific
reserved	0x09C4	BYTE[4]	--	--	
DC user P12	0x09C8	DWORD	R	R	Implementation Specific
reserved	0x09CC	BYTE[4]	--	--	

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 view of the 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.

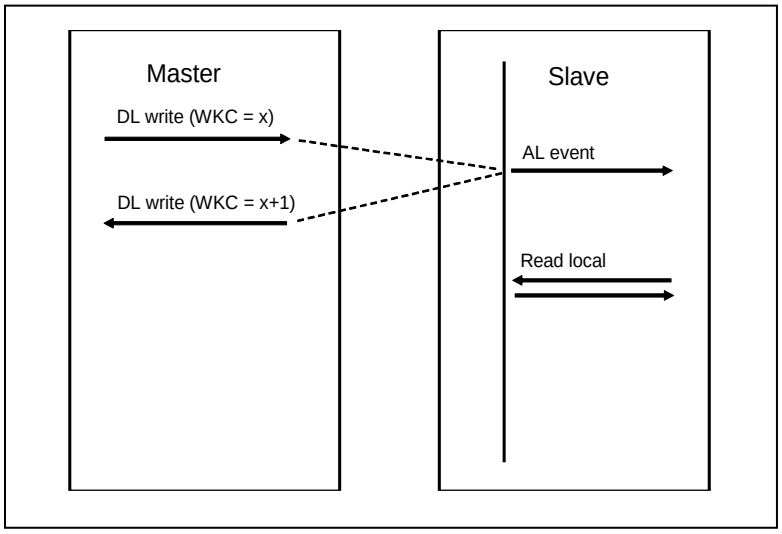


Figure 38 – Successful write sequence to mailbox

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 is 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.

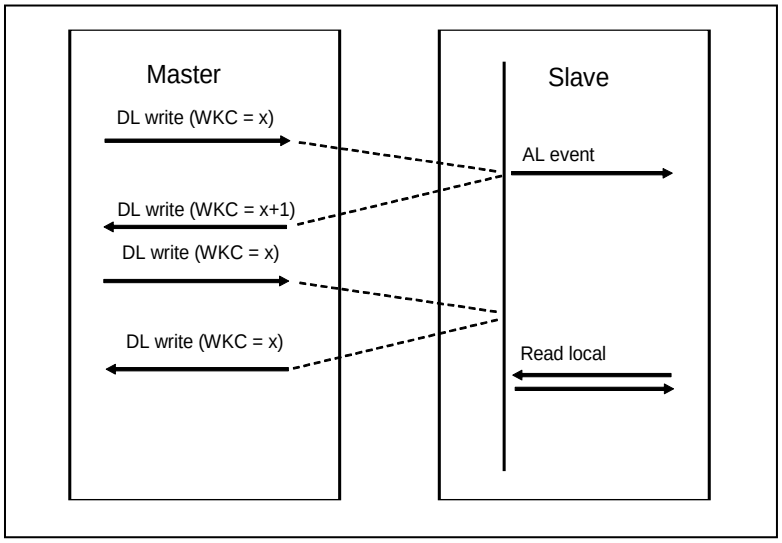


Figure 39 – Bad write sequence to mailbox

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 is 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.

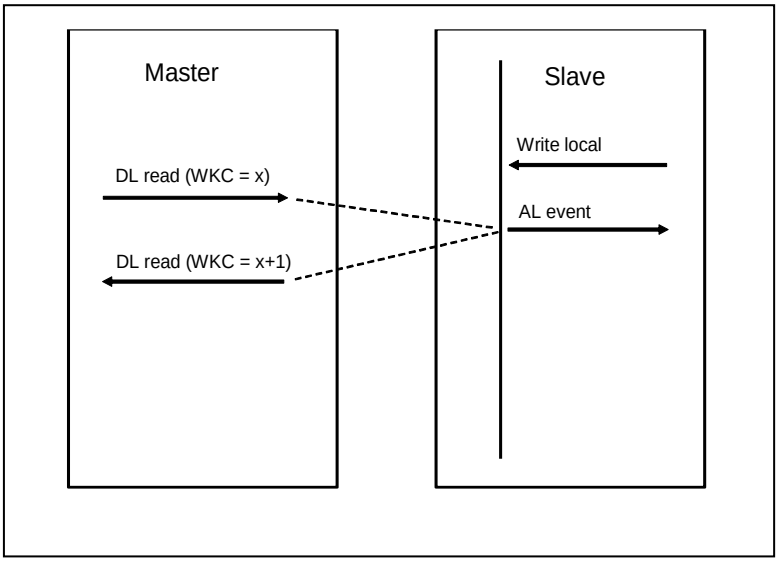


Figure 40 – Successful read sequence to mailbox

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.

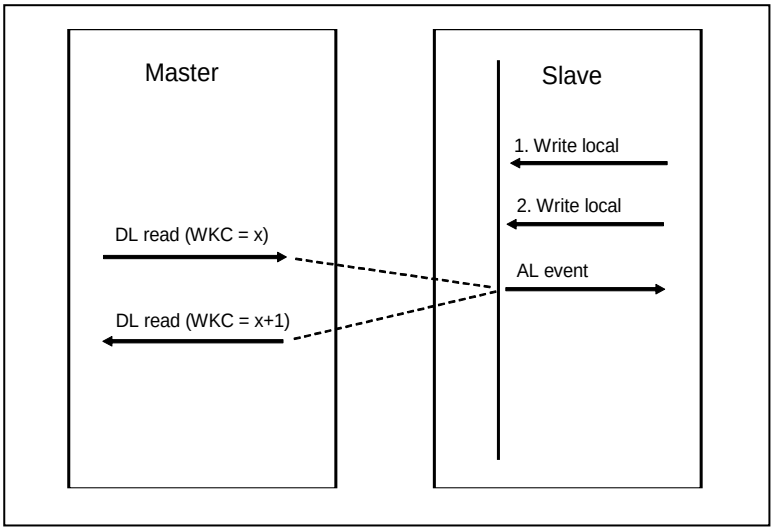


Figure 41 – Bad read sequence to mailbox

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.

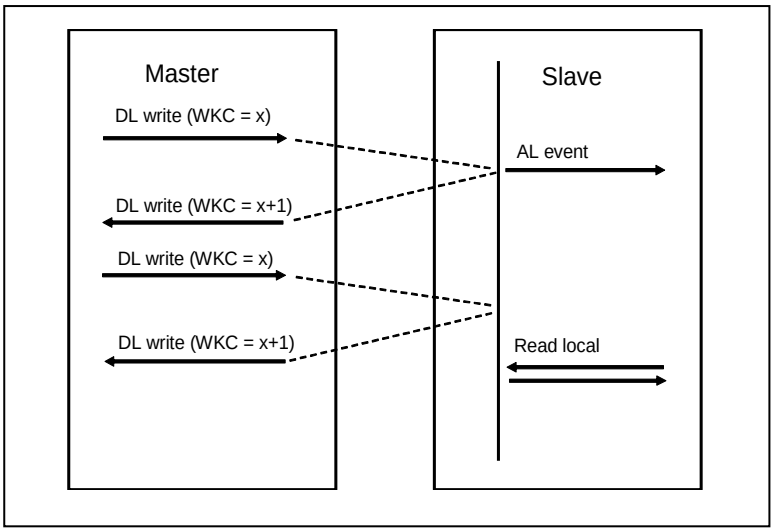


Figure 42 – Successful write sequence to buffer

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.

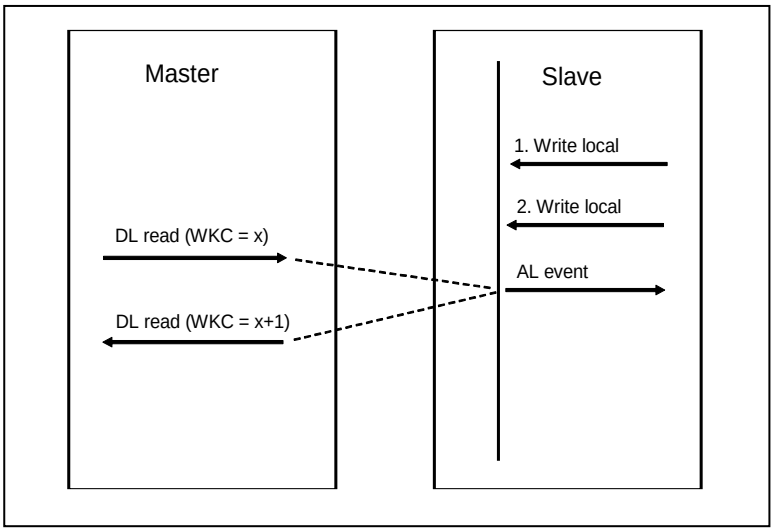


Figure 43 – Successful read sequence to buffer

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 EtherCAT: 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.

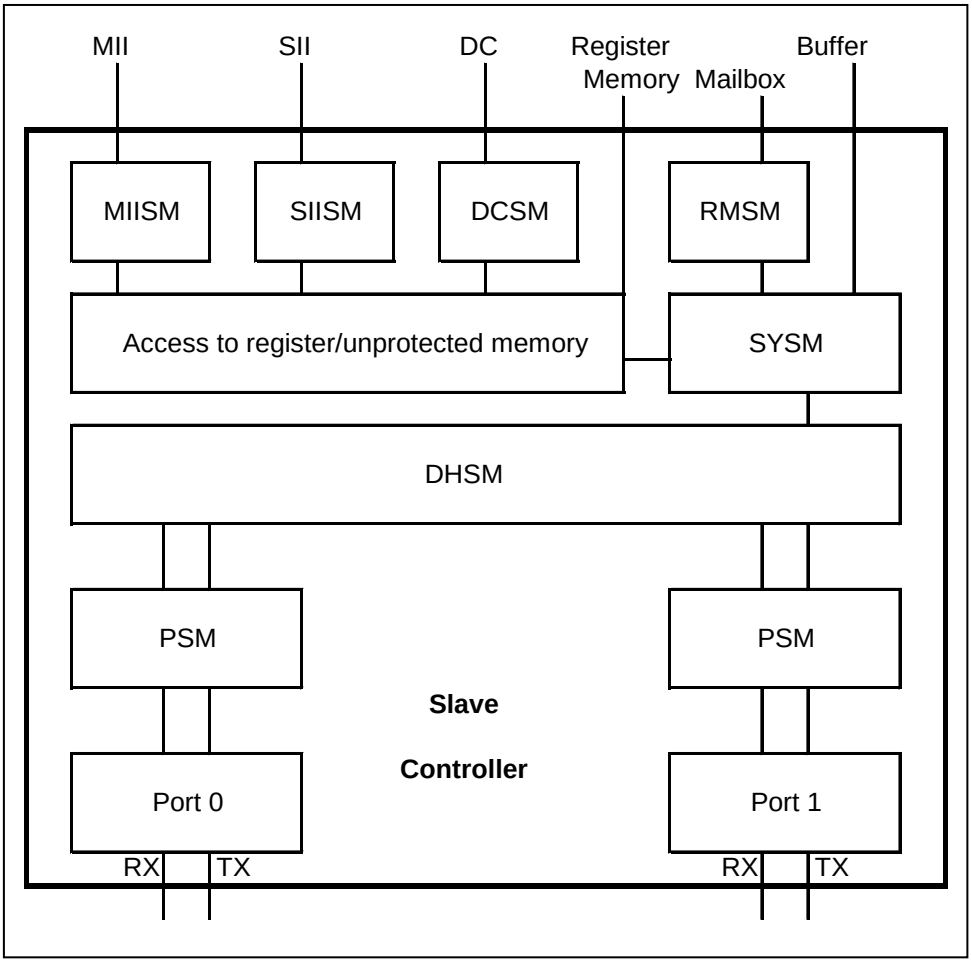


Figure 44 – Structuring of the protocol machines of an slave

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 ETG.1000.3 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 EtherCAT 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 Sync manager state machine (SYSM)

Sync manager state machine will handle the memory areas covered by Sync manager 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 ETG.1000.3. 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.

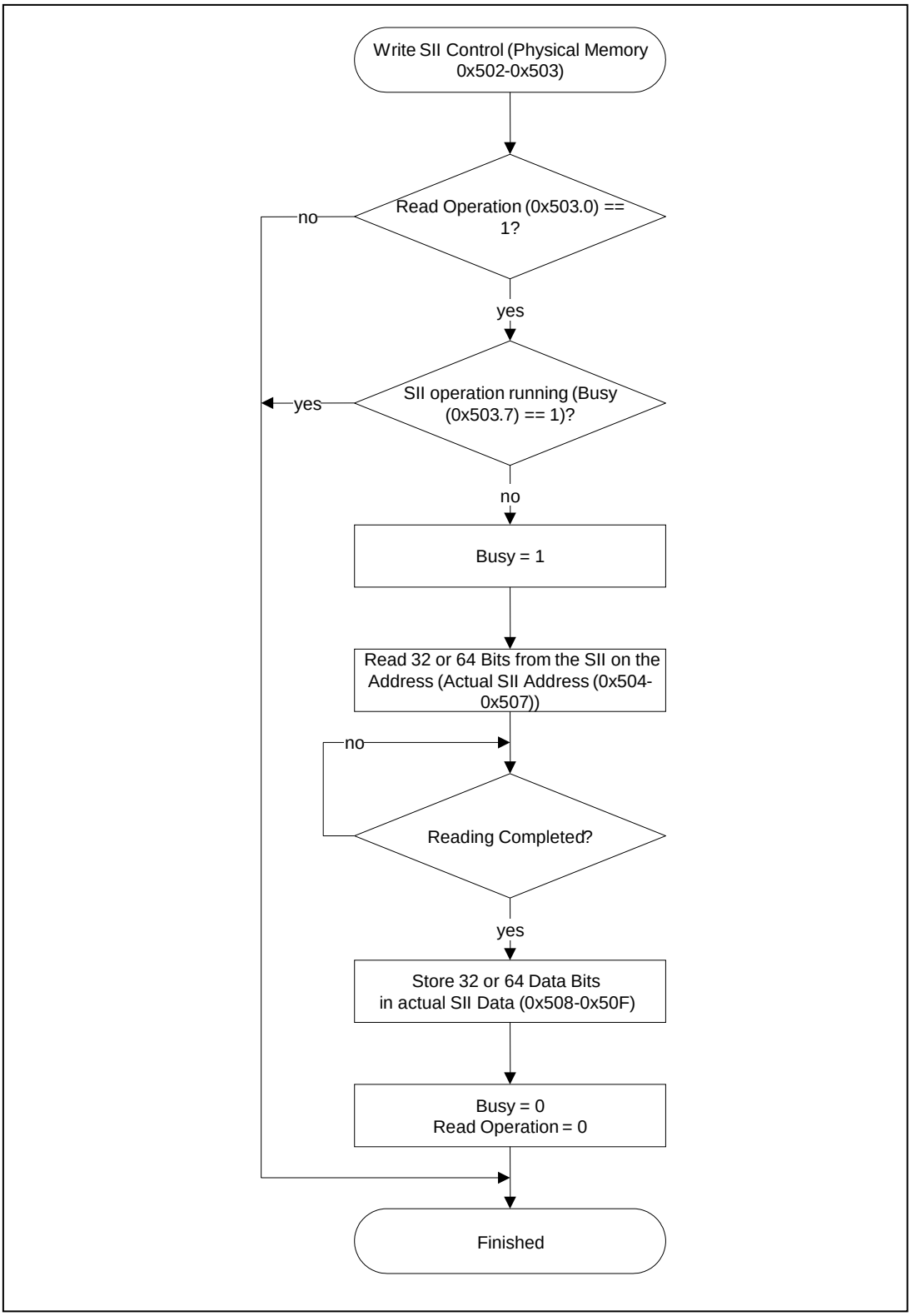


Figure 45 – Slave information interface read operation

8.2.5.3 Write operation

Figure 46 shows the flow of a write operation.

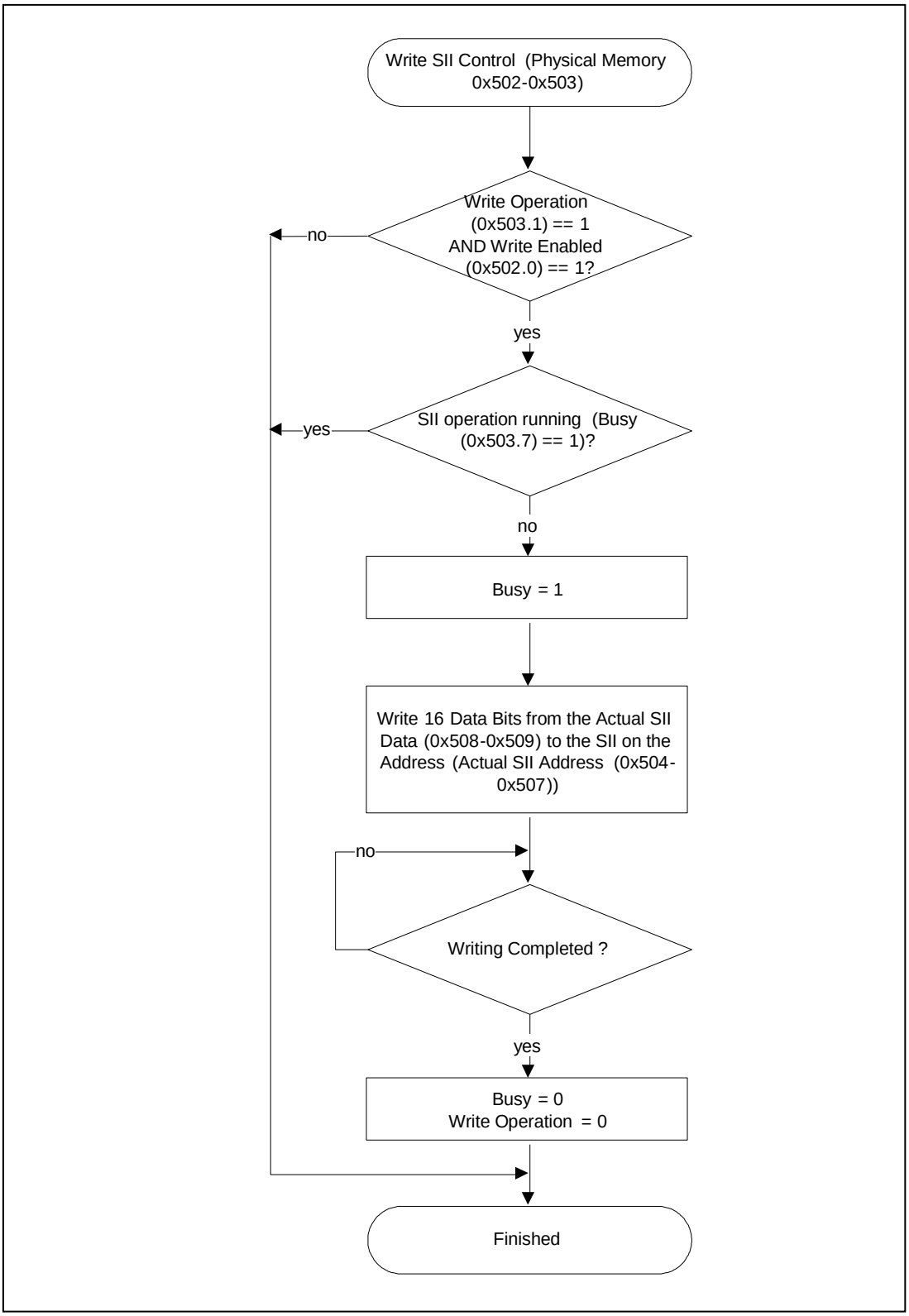


Figure 46 – Slave information interface write operation

8.2.5.4 Reload operation

Figure 47 shows the flow of a reload operation.

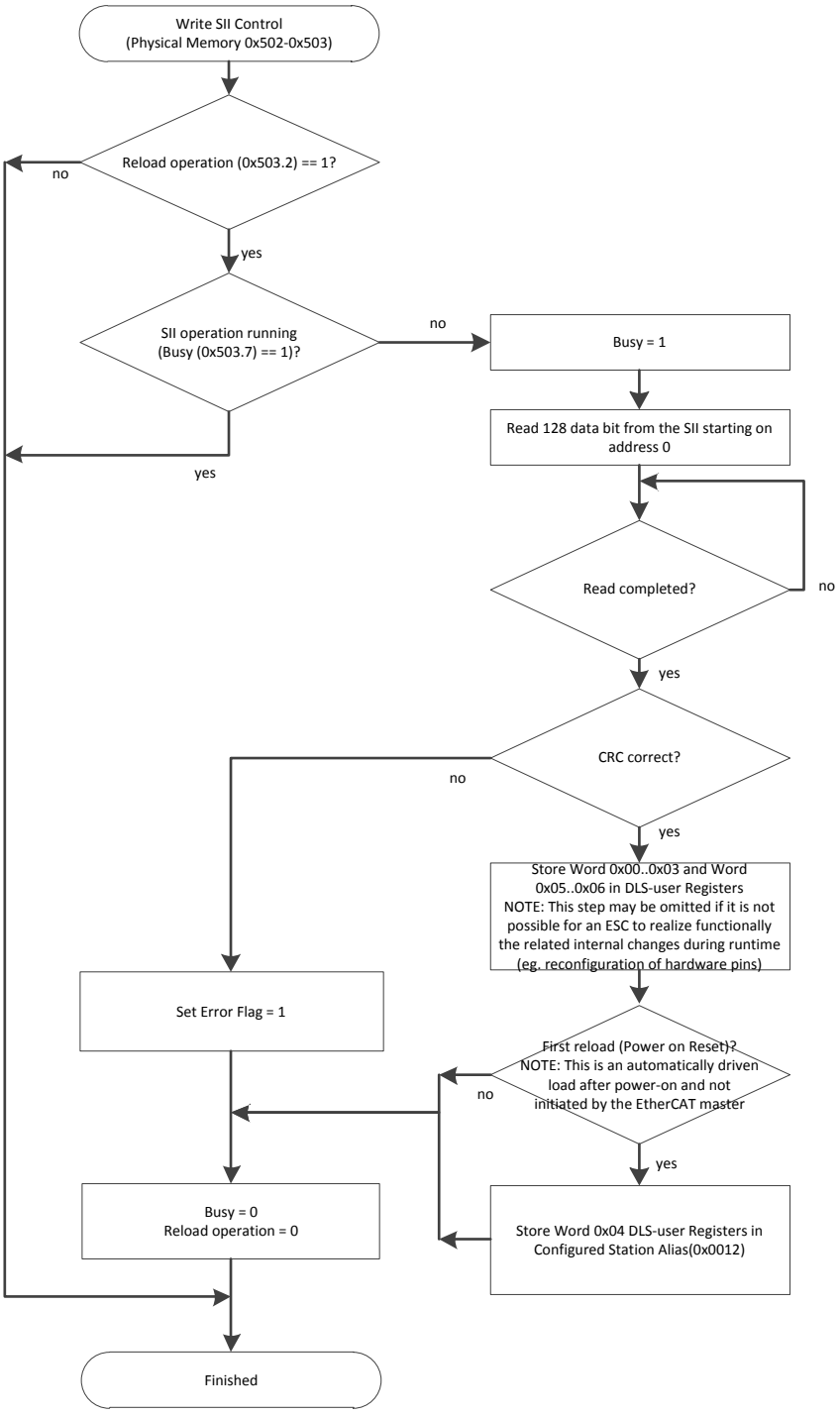


Figure 47 – Slave information interface reload operation

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 ETG.1000.3.

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 ISO/IEC 61588

Precondition
There is no mechanism to calculate the residence time. Thus, no significant jitter of residence time is allowed for all EtherCAT 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 100MHz 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.

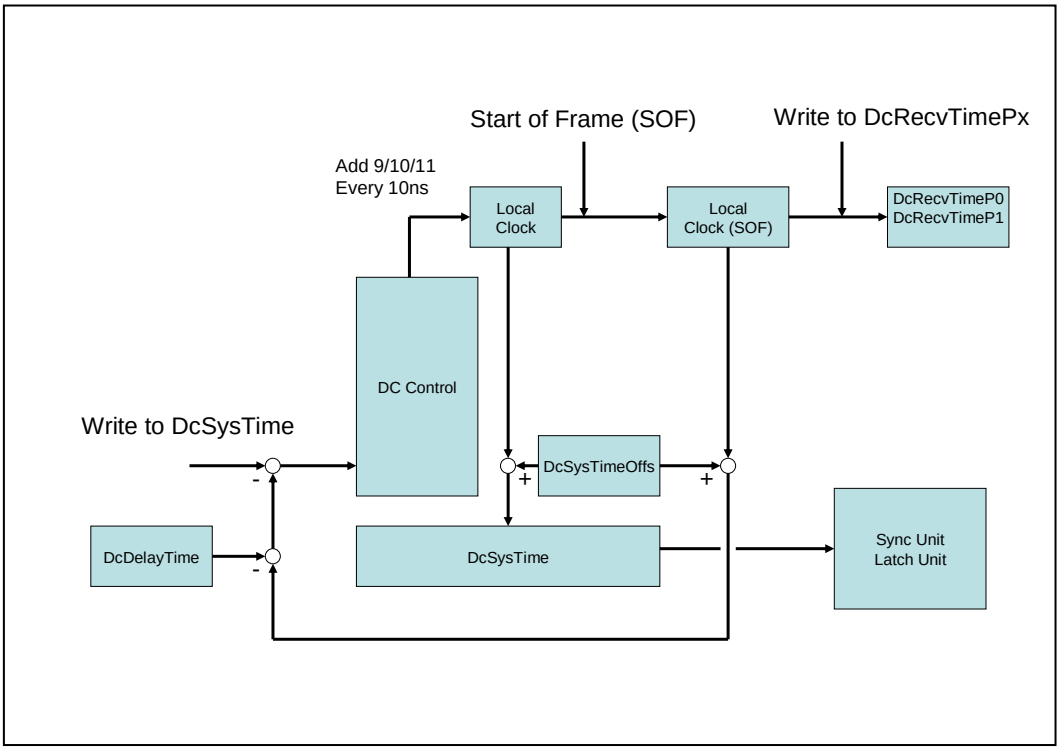


Figure 48 – Distributed clock

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. An EtherCAT 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 EtherCAT PDUs accessing write buffers of the devices to be synchronized.

8.2.7.2 Delay measurement

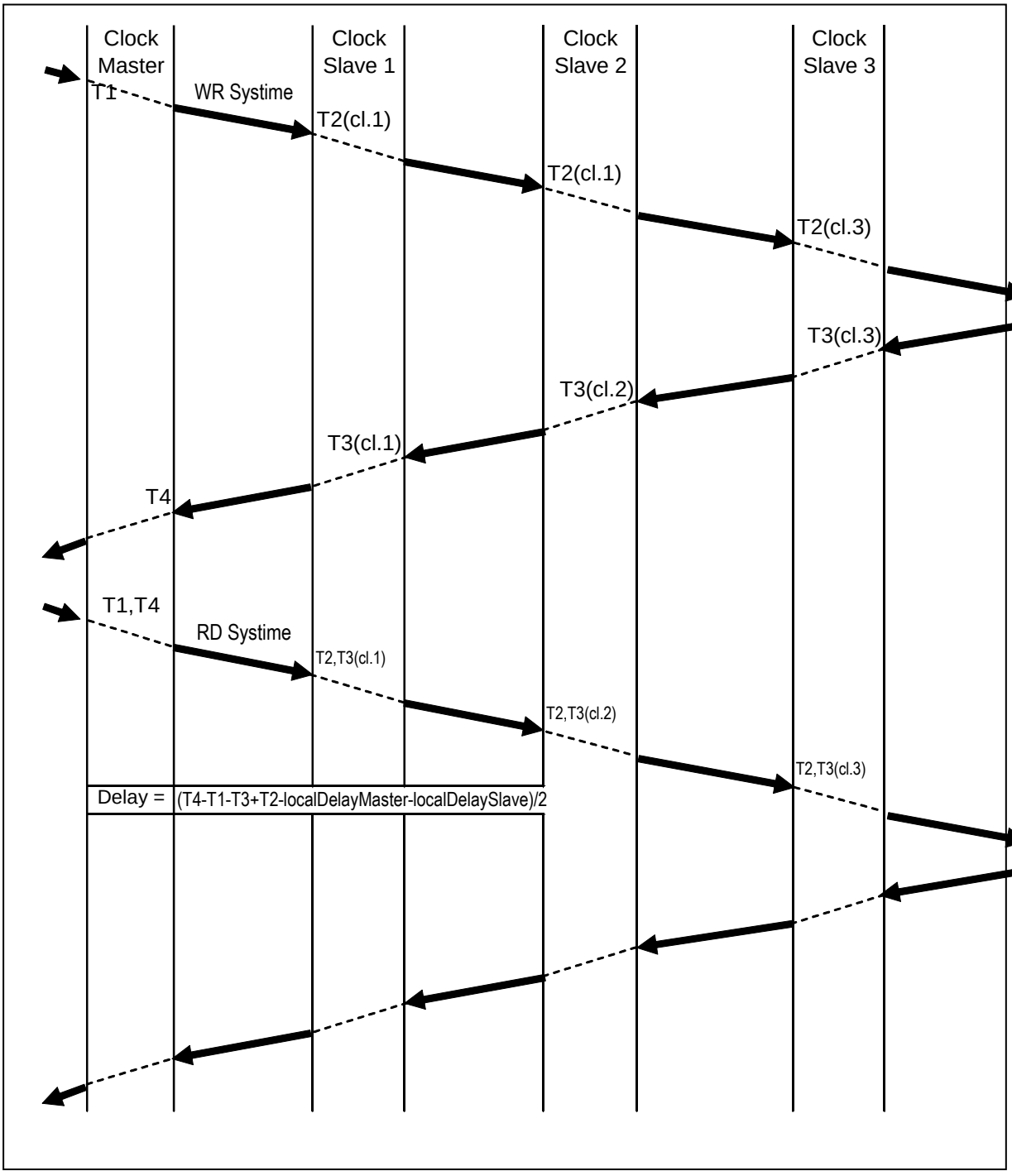


Figure 49 – Delay measurement sequence

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)
–
EtherCAT: 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

NOTE: The port sequence in the DHSM state table A.5 is given as 0->1->2->-3->0 for a 4-port device, but the frame processing in the slave controller (ESC) is in the order 0->3->1->2->0

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

A.1.2 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 EtherCAT 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 EtherCAT 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 an EtherCAT PDU.

Etype1
First octet of Ethertype.

Length
Length of an EtherCAT PDU.

MF
Indicates last EtherCAT PDU in an Ethernet frame.

RxTimeLatch[0..LASTP]
Latched time of last received frame per port.

AdL
First address octet of an EtherCAT PDU.

AdH
Second address octet of an EtherCAT PDU.

DaL
Third address octet of an EtherCAT PDU.

DaH
Fourth address octet of an EtherCAT PDU.

LeL
First length octet of an EtherCAT PDU.

LeH
Second length octet of an EtherCAT PDU.

wkc
Local additive factor to be added to WKC in EtherCAT PDU.

RData
Local data octet of memory element.

WData
Data octet of an EtherCAT 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.

A.1.3 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

#	Current state	Event /condition ⇒action	Next state
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 if 0x0100[0] == 1 then 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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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
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

Table A.5 *(continued)*

#	Current state	Event /condition ⇒action	Next state
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

A.1.4 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 == APRD, APRW, APRW && AdH, AdL == 0) || (Cmd == FPRD, FPRW, FPRMW && AdH, AdL==ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled)
CMDPW	Cmd == BWR; BRW || (Cmd == APWR, APRW && (AdH, AdL) == 0) || (Cmd == FPWR, FPRW && AdH, AdL==ConfiguredStationAddress, ConfiguredStationAlias if Alias enabled) || (Cmd == ARMW && (AdH, AdL) != 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

Function name	Operations
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

A.2.2 State machine description

There exists a SYSM for each Sync manager channel. The abstract model is that EtherCAT 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 buffers, 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.

A.2.3 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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
9	OFF	**Read.ind (Address, Data, WKC)** => pass next	OFF
10	OFF	**Terminate.ind(Success)** => pass next	OFF
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
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
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
28	BR-IDLE	**Read.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BR-IDLE
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
37	BW-IDLE	**DL-Read Local.req (Address, Length)** /NA&&AE && !uact => L-Status = WRNGSEQ DL-Read Local.cnf (L-Status, Data)	BW-IDLE
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
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
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
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
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
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
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
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
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

Table A.12 *(continued)*

#	Current State	Event /Condition =>Action	Next State
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
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

A.2.4 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

A.3.2 State machine description

There exists 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.

A.3.3 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

A.3.4 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 60559, *Binary floating-point arithmetic for microprocessor systems*

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

IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

IEC/TR 61158-1(Ed.2.0), *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

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

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

IEC 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 10646-1, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*

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*

ISO 8509, *Information processing systems – Open System Interconnection – Service Conventions*

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>

Internet Engineering Task Force (IETF), *Request for Comments (RFC)*:

RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets : MIB-II*

RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types*

ETG.1000.2 *EtherCAT Specification Part 2: Physical layer specification and service definition*

ETG.1000.3 *EtherCAT Specification Part 3: Data Link Layer service definition*

ETG.1000.4 *EtherCAT Specification Part 4: Data-link layer protocol specification*

ETG.1000.5 *EtherCAT Specification Part 5: Application layer service definition*

ETG.1000.6 *EtherCAT Specification Part 6: Application layer protocol specification*

ETG.1100 EtherCAT Specification Communication profiles

———————