IEC 61158-4-8

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 4-8: Data-link layer protocol specification – Type 8 elements

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

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

About IEC publications

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

- Catalogue of IEC publications: www.iec.ch/searchpub

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 It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

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

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 61158-4-8

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 4-8: Data-link layer protocol specification – Type 8 elements

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE XF

ICS 35.100.20; 25.040.40

ISBN 2-8318-9435-2

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 4-8: Data-link layer protocol specification – Type 8 elements

FOREWORD

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

NOTE Use of some of the associated protocol types is restricted by their intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a particular data-link layer protocol type to be used with physical layer and application layer protocols in Type combinations as specified explicitly in the IEC 61784 series. Use of the various protocol types in other combinations may require permission from their respective intellectual-property-right holders.

IEC draws attention to the fact that it is claimed that compliance with this standard may involve the use of patents as follows, where the [xx] notation indicates the holder of the patent right:

Type 8 and possibly other Types:

DE 41 00 629 C1 [PxC] Steuer- und Datenübertragungsanlage

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

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

[PxC]: Phoenix Contact GmbH & Co. KG
Referat Patente / Patent Department
Postfach 1341
D-32819 Blomberg
Germany

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

International Standard IEC 61158-4-8 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This first edition and its companion parts of the IEC 61158-4 subseries cancel and replace IEC 61158-4:2003. This edition of this part constitutes an editorial revision.

This edition of IEC 61158-4 includes the following significant changes from the previous edition:

a) deletion of the former Type 6 fieldbus, and the placeholder for a Type 5 fieldbus data link layer, for lack of market relevance;

b) addition of new types of fieldbuses;

c) division of this part into multiple parts numbered -4-1, -4-2, …, -4-19.

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

FDIS	Report on voting
65C/474/FDIS	65C/485/RVD

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

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

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

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

NOTE The revision of this standard will be synchronized with the other parts of the IEC 61158 series.

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

INTRODUCTION

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

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

a) as a guide for implementors and designers;

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

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

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 4-8: Data-link layer protocol specification – Type 8 elements

1 Scope

1.1 General

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

This protocol provides a highly-optimized means of interchanging fixed-length input/output data and variable-length segmented messages between a single master device and a set of slave devices interconnected in a loop (ring) topology. The exchange of input/output data is totally synchronous by configuration, and is unaffected by the messaging traffic.

Devices are addressed implicitly by their position on the loop. The determination of the number, identity and characteristics of each device can be configured, or can be detected automatically at start-up.

1.2 Specifications

This standard specifies

a) procedures for the timely transfer of data and control information from one data-link user entity to a peer user entity, and among the data-link entities forming the distributed data-link service provider;

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

1.3 Procedures

The procedures are defined in terms of

a) the interactions between peer DL-entities (DLEs) through the exchange of fieldbus 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 a Ph-service provider in the same system through the exchange of Ph-service primitives.

1.4 Applicability

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

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

1.5 Conformance

This standard also specifies conformance requirements for systems implementing these procedures. This 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 standard. 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 (Ed.4.0), *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-3-8, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 3-8: Data link service definition – Type 8 elements*

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

3 Terms, definitions, symbols and abbreviations

For the purposes of this standard, the following terms, definitions, symbols and abbreviations 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-address [7498-3]

3.1.2 DL-address-mapping [7498-1]

3.1.3 DL-connection [7498-1]

3.1.4 DL-connection-end-point [7498-1]

3.1.5 DL-connection-end-point-identifier [7498-1]

3.1.6 DL-data-source [7498-1]

3.1.7 DL-name [7498-3]

3.1.8 DL-protocol [7498-1]

3.1.9 DL-protocol-connection-identifier [7498-1]

3.1.10 DL-protocol-control-information [7498-1]

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

3.1.12 DL-service-connection-identifier [7498-1]

3.1.13 DL-service-data-unit [7498-1]

3.1.14 DL-user-data [7498-1]

3.1.15 layer-management [7498-1]

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

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

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

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

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

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

3.1.22 Ph-interface-control-information [7498-1]

3.1.23 Ph-interface-data [7498-1]

3.1.24 primitive name [7498-3]

3.1.25 reset [7498-1]

3.1.26 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 confirm (primitive);
requestor.deliver (primitive)

3.2.2 DL-service-primitive;
primitive

3.2.3 DL-service-provider

3.2.4 DL-service-user

3.2.5 indication (primitive)
acceptor.deliver (primitive)

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

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

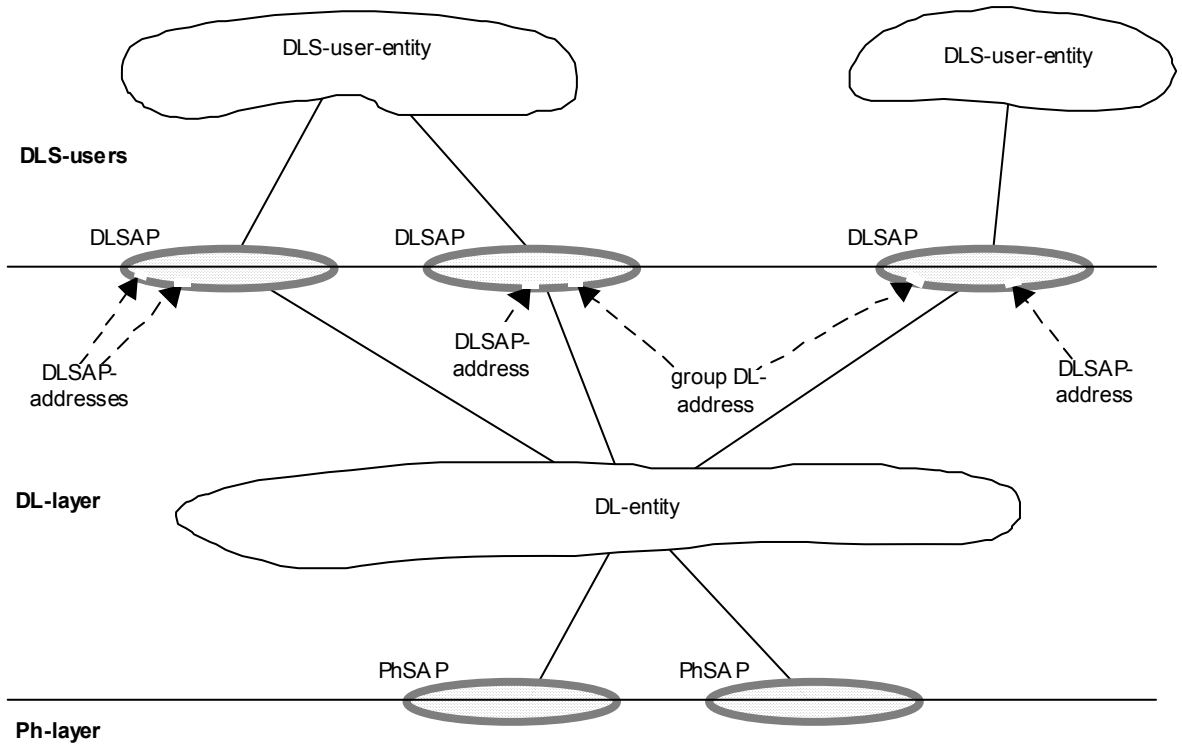
3.3 Common terms and definitions

NOTE This subclause contains the common terms and definitions used by Type 8.

3.3.1
link, local link
single DL-subnetwork in which any of the connected DLEs may communicate directly, without any intervening DL-relaying, whenever all of those DLEs that are participating in an instance of communication are simultaneously attentive to the DL-subnetwork during the period(s) of attempted communication

3.3.2
DLSAP
distinctive point at which DL-services are provided by a single DL-entity to a single higher-layer entity.

NOTE This definition, derived from ISO/IEC 7498-1, is repeated here to facilitate understanding of the critical distinction between DLSAPs and their DL-addresses. (See Figure 1.)



NOTE 1 DLSAPs and PhSAPs are depicted as ovals spanning the boundary between two adjacent layers.

NOTE 2 DL-addresses are depicted as designating small gaps (points of access) in the DLL portion of a DLSAP.

NOTE 3 A single DL-entity may have multiple DLSAP-addresses and group DL-addresses associated with a single DLSAP.

Figure 1 – Relationships of DLSAPs, DLSAP-addresses and group DL-addresses

3.3.3
DL(SAP)-address
either an individual DLSAP-address, designating a single DLSAP of a single DLS-user, or a group DL-address potentially designating multiple DLSAPs, each of a single DLS-user

NOTE This terminology is chosen because ISO/IEC 7498-3 does not permit the use of the term DLSAP-address to designate more than a single DLSAP at a single DLS-user.

3.3.4
extended link
DL-subnetwork, consisting of the maximal set of links interconnected by DL-relays, sharing a single DL-name (DL-address) space, in which any of the connected DL-entities may communicate, one with another, either directly or with the assistance of one or more of those intervening DL-relay entities

NOTE An extended link may be composed of just a single link.

3.3.5
frame
denigrated synonym for DLPDU

3.3.6
receiving DLS-user
DL-service user that acts as a recipient of DL-user-data

NOTE A DL-service user can be concurrently both a sending and receiving DLS-user.

3.3.7
sending DLS-user
DL-service user that acts as a source of DL-user-data

3.4 Additional Type 8 definitions

3.4.1 bus coupler

PhL entity which includes or excludes Ph-segments into or from the network

3.4.2 device

slave or master

3.4.3 device code

two octets which characterize the properties of a slave

3.4.4 DLPDU cycle

transaction initiated from the master in which user data or identification/status information is sent to all slaves and – within the same cycle - received from all slaves

3.4.5 IN data

data received by the master and sent by the slaves

3.4.6 master

DL-entity controlling the data transfer on the network and initiating the media access of the slaves by starting the DLPDU cycle

3.4.7 OUT data

data sent by the master and received by the slaves

3.4.8 parameter channel

acyclic transmission path using a client/server communication model

3.4.9 process data channel

conveyance path allowing a very efficient, high-speed and cyclic transmission of process-relevant data, between slaves and master

3.4.10 receive update memory

memory area containing the data, which was received from the network

3.4.11 ring segment

group of slaves in consecutive order

3.4.12 ring segment level

nesting level number of a ring segment

3.4.13 slave

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

3.4.14 transmit update memory

memory area containing the data to be sent across the network

3.4.15 update time

time which passes between two consecutive starts of DLPDU cycles used for data transfer

3.5 Symbols and abbreviations

3.5.1 Type 8 reference model terms

BLL	basic link layer
BLLSDU	BLL service data unit
BLL_TSDU	BLL transmit service data unit
BLL_RSDU	BLL receive service data unit
MACSDU	MAC service data unit
PDL	peripherals data link
PDLSDU	PDL service data unit
PhMS	Ph-management service

3.5.2 Local variables, timers, counters and queues

add_wait	See Table 15
BLL_access_control	See Table 31
bus_timeout	See Table 31
C_{cerr}	See Table 22
C_{conf}	See Table 22
C_{cycle}	See Table 22
C_{req_retry}	See Table 22
C_{swa}	See Table 22
configuration_valid	See Table 31
loopback_word (LBW)	See Table 41
max_dlsdu_size_from_req	See Table 15
max_dlsdu_size_from_res	See Table 15
max_receiving_queue_depth	See Table 15
max_sending_queue_depth	See Table 15
max_spa_retry	See Table 15
max_swa_count	See Table 15
start_bus_cycle	See Table 15
time_timeout	See Table 41
trigger_mode	See Table 15
update_time	See Table 31

3.5.3 DLPDU classes

DATA	data	See Table 20
FCB_SET	frame count bit	See Table 20
IDL	idle	See Table 20
RWA	read word again	See Table 20
SPA	send parameter with acknowledge	See Table 20
SVA	send value with acknowledge	See Table 20
SWA	send word again	See Table 20

3.5.4 Miscellaneous

AT	application triggered
BAC	basic access control

CO	confirmation
CRC	cyclic redundancy check
DL-Ph	Data Link-Physical (interface)
DLI	DL-interface
DSAP	destination service access point
FCB	frame count bit
FCT	function
FMS	fieldbus message specification
GSM	get shared memory
IN	input
L_status	link status
LBW	loopback word
lsb	least significant bit
M, (m)	mandatory
msb	most significant bit
NT	network triggered
O, (o)	optional
OUT	output
PDL	peripherals data Link
PM	protocol machine
PNM1	peripherals network management of Layer 1
PNM2	peripherals network management of Layer 2
PSM	put shared memory
RUM	receive update memory
S	selection
SM	state machine
TUM	transmit update memory

4 DL-protocol

4.1 Overview

The DLL is modelled as a Four-Level model (see Figure 2).

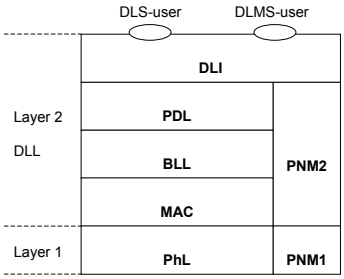


Figure 2 – Data Link Layer Entity

4.2 DL-service Interface (DLI)

4.2.1 General

The Data Link service Interface (DLI) provides service primitives to the DLS-user and DLMS-user (see Figure 3).

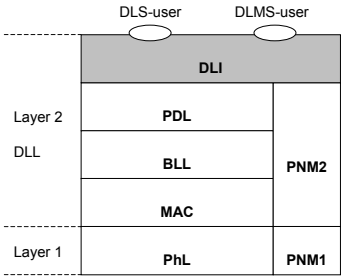


Figure 3 – Location of the DLI in the DLL

The DLI translates and issues the primitives received from the DLS-/DLMS-user to the local PDL and PNM2 interface. It also translates and issues the primitives received from the local PDL or PNM2 interface and delivers it to the DLS-/DLMS-user.

The DLI protocol has only a single state called “ACTIVE”.

4.2.2 Primitive definitions

4.2.2.1 General

Table 1 and Table 2 show the primitives exchanged between DLS-/DLMS-user and DLI.

4.2.2.2 Primitives exchanged between DLS-/DLMS-user and DLI

Table 1 – Primitives issued by DLS-/DLMS-user to DLI

Primitive name	Source	Associated parameters	Functions
DL-PUT request	DLS-user	Buffer DL-identifier, DLS-user-data	Requests the DLE to write a DLSDU into the transmit buffer
DL-GET request	DLS-user	Buffer DL-identifier	Requests the DLE to read a DLSDU from the receive buffer
DL-DATA request	DLS-user	DLCEP DL-identifier, DLS-user-data	Requests the DLE to write a DLSDU into the send queue
DLM-RESET request	DLS-user	(<none>)	Requests the DLE to execute a reset.
DLM-SET-VALUE request	DLMS-user	Variable-name, Desired-value	Requests the DLE to overwrite a local variable
DLM-GET-VALUE request	DLMS-user	Variable-name	This Primitive is issued to request the DLL to read the content of a local variable
DLM-GET-CURRENT-CONFIGURATION request	DLMS-user	Desired-configuration	Requests the DLE to read out the current configuration of the DL-subnetwork.
DLM-GET-ACTIVE-CONFIGURATION request	DLMS-user	(<none>)	Requests the DLE to read out the active configuration of the DL-subnetwork
DLM-SET_ACTIVE_CONFIGURATION request	DLMS-user	Active-configuration	Requests the DLE to execute a certain active configuration of the DL-subnetwork

Table 2 – Primitives issued by DLI to DLS-/DLMS-user

Primitive name	Source	Associated parameters
DL-PUT confirm	DLI	Status
DL-GET confirm	DLI	Status, DLS-user-data
DL-BUFFER-RECEIVED indication	DLI	Status
DL-DATA confirm	DLI	Status
DL-DATA indication	DLI	DLCEP DL-identifier, DLS-user-data
DLM-RESET confirm	DLI	Status
DLM-EVENT indication	DLI	Event-identifier, Additional-information
DLM-SET-VALUE confirm	DLI	Status
DLM-GET-VALUE confirm	DLI	Status, Additional-information
DLM-GET-CURRENT-CONFIGURATION confirm	DLI	Status, Additional-information
DLM-GET-ACTIVE-CONFIGURATION confirm	DLI	Status, Additional-information
DLM-SET-ACTIVE-CONFIGURATION confirm	DLI	Status, Additional-information

4.2.2.3 Parameters of DLS-/DLMS-User and DLI primitives

All parameters used in the primitives exchanged between the DLS-/DLMS-user and the DLI are specified in IEC 61158-3-8.

4.2.3 DLI State Tables

4.2.3.1 General

Figure 4 show a state transition diagram of the DLI.

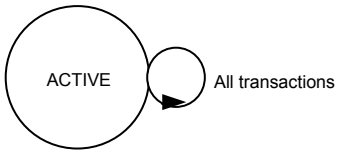


Figure 4 – State transition diagram of DLI

The transitions of the DLI protocol are specified in Table 3 and Table 4. Service primitive names are mixed-case with underscores ("_") replacing dashes ("-"), and with a dot-separated suffix indicating the underlying type of primitive: request, confirm or indication.

Table 3 – DLI state table – sender transactions

#	Current state	Event Action	Next state
S1	ACTIVE	DL_Put.request PSM.request{ offset := GetOffset(Buffer DL-identifier) length := "length of DLS-user-data" data := DLS-user-data }	ACTIVE
S2	ACTIVE	DL_Get.request GSM.request{ offset := GetOffset(Buffer DL-identifier) length := GetLength(Buffer DL-identifier) }	ACTIVE

S3	ACTIVE	DL_Data.request{
		
		PDL_Data_Ack.request{
		rem_add := GetRemAdd (DLCEP DL-identifier)
		DLSDU := DLS-user-data }
			ACTIVE

S4	ACTIVE	DLM_Reset.request PNM2_Reset.request{ }	ACTIVE
S5	ACTIVE	DLM_Set_Value.request PNM2_Set_Value.request { variable_name := Variable-name, desired_value := Desired-value }	ACTIVE
S6	ACTIVE	DLM_Get_Value.request PNM2_Get_Value.request{ variable_name := Variable-name }	ACTIVE
S7	ACTIVE	DLM_Get_Current_Configuration.request PNM2_Get_Current_Configuration.request{ network_configuration := Desired Configuration }	ACTIVE
S8	ACTIVE	DLM_Get_Active_Configuration.request PNM2_Get_Active_Configuration.request{ }	ACTIVE
S9	ACTIVE	DLM_Set_Active_Configuration.request PNM2_Set_Active_Configuration.request{ active_configuration := Active-configuration }	ACTIVE

Table 4 – DLI state table – receiver transactions

#	Current state	Event Action	Next state
R1	ACTIVE	PSM.confirm DL_Put.confirm{ Status := status }	ACTIVE
R2	ACTIVE	GSM.confirm DL_Get.confirm{ Status := status, DLS-user-data := data }	ACTIVE
R3	ACTIVE	Buffer_Received.indication DL_Buffer_Received.indication{ Status := status }	ACTIVE
R4	ACTIVE	PDL_Data_Ack.confirm DL_Data.confirm{ Status := L_status }	ACTIVE
R5	ACTIVE	PDL_Data_Ack.indication DL_Data.indication{ DLCEP DL-identifier := GetDlsUserId(local_add), DLS-user-data := DLSDU }	ACTIVE
R6	ACTIVE	PNM2_Reset.confirm DLM_Reset.confirm{ Status := M_status }	ACTIVE
R7	ACTIVE	PNM2_Event.indication DLM_Event.indication { Event-identifier := event, Additional-information := add_info }	ACTIVE
R8	ACTIVE	PNM2_Set_Value.confirm DLM_Set_Value.confirm { Status := M_status}	ACTIVE
R9	ACTIVE	PNM2_Get_Value.confirm DLM_Get_Value.confirm{ Status := M_status Current-Value := current_value }	ACTIVE
R10	ACTIVE	PNM2_Get_Current_Configuration.confirm DLM_Get_Current_Configuration.confirm{ Status := status, Current-configuration := current_configuration }	ACTIVE
R11	ACTIVE	PNM2_Get_Active_Configuration.confirm DLM_Get_Active_Configuration.confirm{ Status := status, Active-configuration := active_configuration }	ACTIVE
R12	ACTIVE	PNM2_Set_Active_Configuration.confirm DLM_Set_Active_Configuration.confirm{ Status := status, Additional-information := add_info }	ACTIVE

4.2.3.2 Functions used by DLI

The functions used by DLI are given in Table 5 to Table 8. The details of these functions is not specified by of this standard. These functions use information which was stored by the local DL-management when establishing the DLCs.

Table 5 – Function GetOffset

Name	GetOffset	**Used in**	DLI
Input	Buffer DL-identifier	**Output**	Offset address
Function	Returns a value that can unambiguously identify the offset address from the transmit buffer		

Table 6 – Function GetLength

Name	GetLength	**Used in**	DLI
Input	Buffer DL-identifier	**Output**	Length of data
Function	Returns the size of the DLSDU which can be held by the buffer named by Buffer DL-identifier.		

Table 7 – Function GetRemAdd

Name	GetRemAdd	**Used in**	DLI
Input	DLCEP DL-identifier	**Output**	Remote address
Function	Returns a value that can unambiguously identify the remote address from the remote device		

Table 8 – Function GetDlsUserId

Name	GetDlsUserId	**Used in**	DLI
Input	Local address	**Output**	DLCEP DL-identifier
Function	Returns a value that can unambiguously identify the DLCEP DL-identifier from the DLS user		

4.3 Peripherals data link (PDL)

4.3.1 Location of the PDL in the DLL

The Peripherals Data Link (PDL) is part of the Data Link Layer and uses the Basic Link Layer. Figure 5 shows its location.

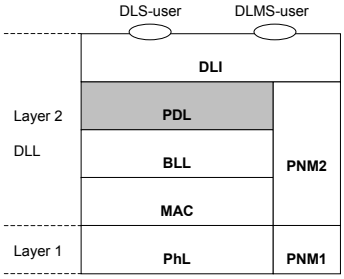


Figure 5 – Location of the PDL in the DLL

By means of the PDL layer each slave can establish a communication link with the master (see Figure 6).

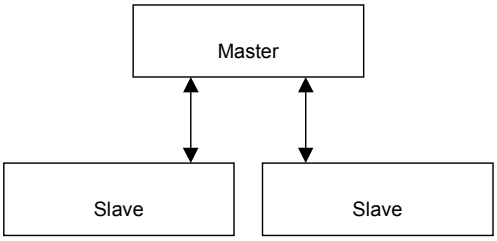


Figure 6 – PDL connection between slave and master

4.3.2 Functionality of the PDL

The PDL performs the following tasks.

— Processing of PDL_Data_Ack service

— Conversion of the non-cyclic PDL_Data_Ack service to cyclic BLL_Data services and vice versa

— Conversion of several DLSDUs of the PDL_Data_Ack.request primitives into a PDLSDU of the BLL_Data.request primitive

— Implementation of two trigger_modes within the PDL *(bus master only)*

— Control of the local PDL protocol machine(s)

— Update of the receive update memory and starting of the PDL protocol machines after a PDLSDU which was received from the BLL has been accepted,

— Generation of a PDLSDU from the transmit update memory as well as by means of the PDL protocol machines and transfer of this PDLSDU to be sent to the BLL

— Implementation of a direct access for PDL-user to the PDL receive and transmit update memory.

NOTE A PDLSDU of the master contains all cyclic data via PSM service to be transmitted in a data cycle and PDL message segments. The PDLSDU of a slave is a subset of the PDLSDU of the master and contains only the cyclic data to be transmitted in one data cycle and the PDL message segment of this slave

The PDL translates these functions by means of the four following protocol machines.

— PDL base protocol machine

— PDL protocol machine

— TRANSMIT protocol machine

— RECEIVE protocol machine.

4.3.3 DLI-PDL interface

4.3.3.1 General

The PDL provides service primitives for the PDL-user (see Figure 7).

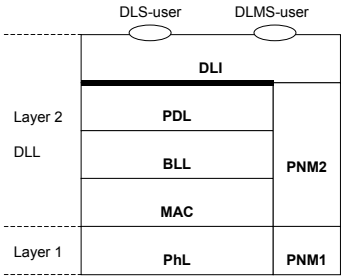


Figure 7 – Interface between PDL-user (DLI) and PDL in the layer model

4.3.3 describes the data transmission services which are available to the PDL-user, together with their service primitives and their associated parameters. These PDL services are mandatory.

4.3.3.2 Overview of the services

4.3.3.2.1 Available services

The following service for data transfer shall be available to the PDL-user:

— Send Parameter with Acknowledge (PDL_Data_Ack).

Furthermore, the PDL-user can use the following services to directly access the update memory.

— **P**ut **S**hared **M**emory (PSM)

— **G**et **S**hared **M**emory (GSM).

Figure 8 shows an overview of the services of the PDL.

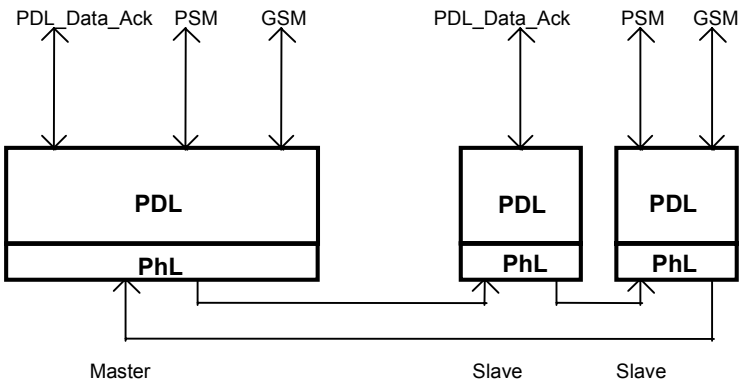


Figure 8 – Overview of the PDL services

4.3.3.2.2 Send parameter with acknowledge (PDL_Data_Ack)

This service allows a local PDL-user to send user data (DLSDU) to a single remote PDL-user. The remote PDL transfers the DLSDU to its PDL-user, provided that the DLSDU was received without errors. The local PDL-user receives a confirmation on the receipt or non-receipt of the DLSDU of the remote PDL.

The PDL_Data_Ack service shall only be used to transfer data from a queue.

Service primitives:

— PDL_Data_Ack.request
— PDL_Data_Ack.indication
— PDL_Data_Ack.confirm

4.3.3.2.3 Put shared memory (PSM)

This service allows a PDL-user to write data of a certain length into the transmit update memory. The BLL shall transmit this data in the next bus cycle.

Service primitives:

— PSM.request
— PSM.confirm

4.3.3.2.4 Get shared memory (GSM)

This service allows a PDL-user to read data of a certain length from the receive update memory.

Service primitives:

— GSM.request
— GSM.confirm

4.3.3.2.5 Buffer received (Buffer_Received)

The PDL uses this service to indicates the local PDL-user, that the contents of

Transmit Update Memory is transmitted, and the contents of

Receive Update Memory is updated with new received data.

Service primitive:

Buffer_Received.indication

4.3.3.3 Overview of the interactions

The services are provided by several service primitives (beginning with PDL_...). In order to request a service, the PDL-user uses a request primitive. A confirmation primitive is returned to the PDL-user after the service has been completed. The arrival of a service request is indicated to the remote PDL-user by means of an indication primitive.

Figure 9, Figure 10, Figure 11 and Figure 12 show the sequences of service primitives to handle the data transfer between master and slave:

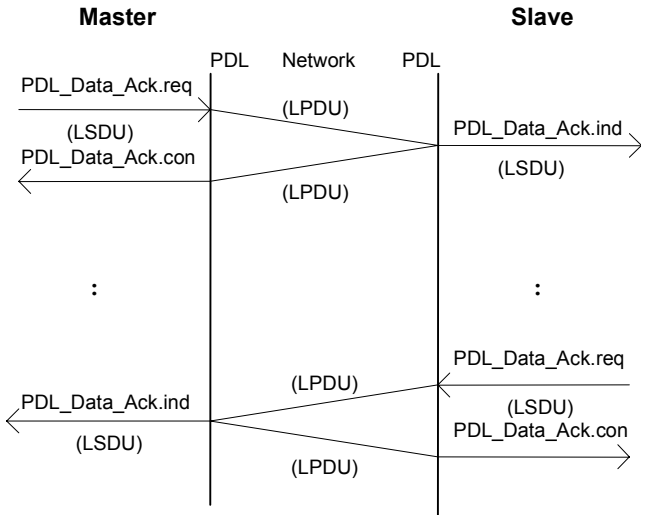


Figure 9 – PDL_Data_Ack service between master and only one slave

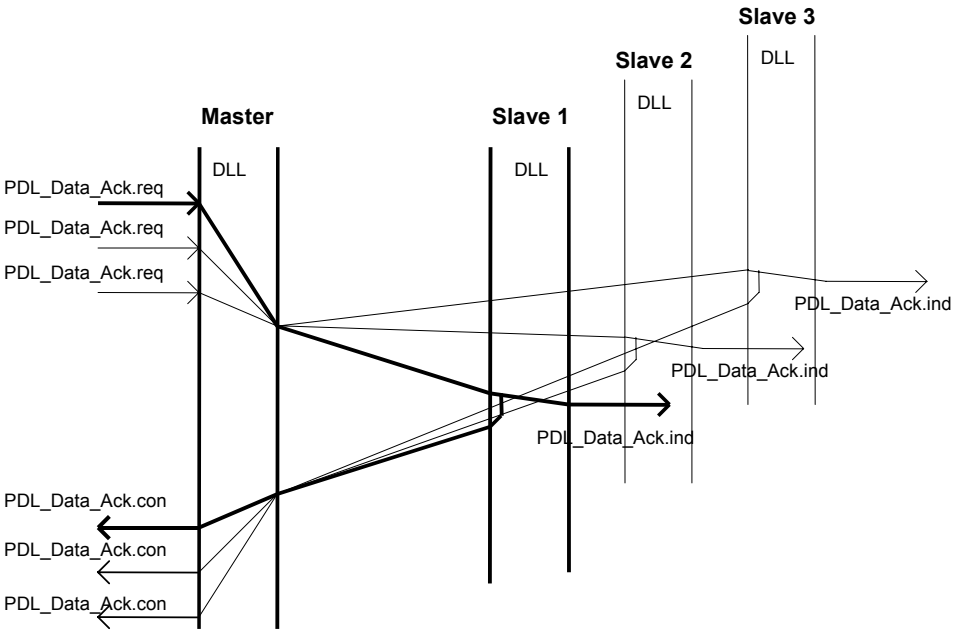


Figure 10 – Parallel processing of PDL_Data_Ack services

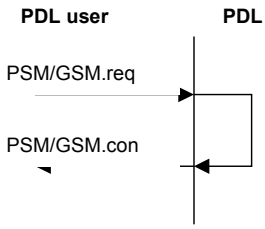


Figure 11 – PSM and GSM service for buffer access

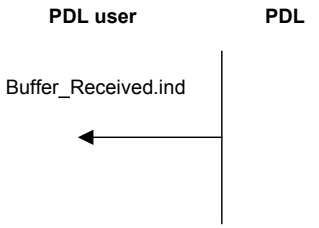


Figure 12 – Buffer_Received service to indicate successful data transfer

4.3.3.4 Formal description of the services and parameters

4.3.3.4.1 PDL_Data_Ack Service

Table 9 shows the parameters of the PDL_Data_Ack service.

Table 9 – PDL_Data_Ack

Parameter name	Request	Indication	Confirm
Argument	M	M	
rem_add	M		
local_add		M	
DLSDU	M	M(=)	
Result			M
rem_add			M
L_status			M

rem_add:

The rem_add parameter defines the PDL address of the remote device. The rem_add corresponds to the physical position of the device in the ring.

local_add:

The local_add parameter conveys the PDL address of the device where the PDL_Data_Ack service was invoked.

D**LSDU:**

The DLSDU parameter contains the PDL-user data to be transmitted.

L_status:

The L_status parameter indicates the success or failure of the preceding PDL_Data_Ack.request. The following values are defined for this parameter in Table 10:

Table 10 – PDL_Data_Ack L_status values

Value	Meaning
OK	Positive acknowledgement, service executed successfully
RR	Negative acknowledgement, resources of the remote PDL not available or insufficient
LR	Resources of the local PDL not available or insufficient
NA	No or not a plausible response (acknowledge response) from the remote device
DS	PDL layer not synchronized at the moment
IV	Invalid parameter in the request call

4.3.3.4.2 PSM service

Table 11 shows the parameters of the PSM service.

Table 11 – PSM

Parameter name	Request	Confirm
Argument	M	
offset	M	
length	M	
data	M	
Result(+)		S
Result(-)		S
error_type		M

offset:

This parameter specifies the offset address, beginning from the start address of the PDL transmit update memory**,** where the data should be written.

length:

This parameter specifies the amount of the data, which should be written into the PDL transmit update memory of layer 2.

data:

This parameter conveys the data, which should be written into the PDL transmit update memory of the layer 2.

error_type:

This parameter indicates the reason, why the service could not be executed successfully.

Possible errors are:

IV Invalid parameters in the request call
Data to write into the transmit update memory are not allowed, because the given parameter(s) of offset and/or length is/are invalid.

4.3.3.4.3 GSM service

Table 12 shows the parameters of the GSM service.

Table 12 – GSM

Parameter name	Request	Confirm
Argument	M	
offset	M	
length	M	
Result(+)		S
data		M
Result(-)		S
error_type		M

offset:

This parameter specifies the offset address, beginning from the start address of the PDL receive update memory, from where the data should be read.

length:

This parameter specifies the amount of the data, which should be read from the PDL receive update memory.

data:

This parameter conveys the data, which was read from the PDL receive update memory.

error_type:

This parameter indicates the reason, why the service could not be executed successfully. Possible error sources:

IV	Invalid parameters in the request call Data to be read from the receive update memory are not allowed, because the given parameter(s) of offset and/or length is/are invalid.

4.3.3.5 Detailed description of the interactions

4.3.3.5.1 Send parameter with acknowledge (PDL_Data_Ack)

The local PDL-user prepares a DLSDU which is transmitted by a PDL_Data_Ack.request primitive to the local PDL. The PDL accepts this service request and tries to send the DLSDU to the requested remote PDL. The local PDL sends a confirmation to its PDL-user with the PDL_Data_Ack.confirm primitive, which indicates a correct or incorrect data transfer.

Before the local PDL sends a confirmation to its user, a confirmation from the remote PDL is mandatory. If this confirmation is not received within the timeout period $T_{TO\_SPA\_ACK}$, the local PDL retries to send the DLSDU to the remote PDL. If the confirmation does not come after the Nth repetition (max_retry_count), then the local PDL sends a negative confirmation to its user.

If the data message was received without errors, the remote PDL transfers the DLSDU with a PDL_Data_Ack.indication primitive through the PDL-user interface.

The coding of the DLSDU is described in 4.3.5.3. Figure 13 shows the data flow between PDL-user, PDL and BLL for a PDL_Data_Ack service:

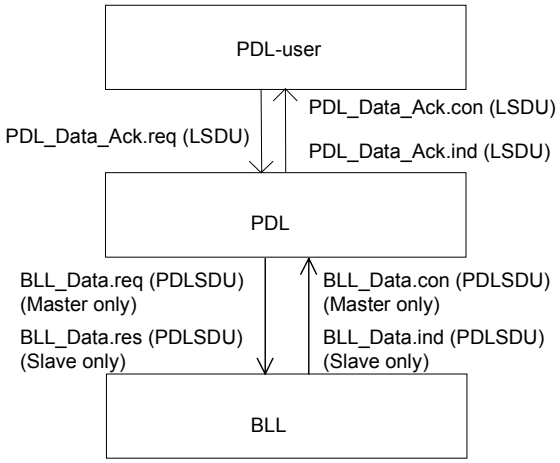


Figure 13 – Data flow between PDL-user, PDL and BLL of a PDL_Data_Ack service

4.3.3.5.2 Put shared memory (PSM)

The PDL-user uses this service to write user data directly to the transmit update memory. The service is locally processed after the PSM.request primitive has arrived. The PDL communicates the successful processing of the service to its PDL-user by means of a PSM.confirm primitive (immediate confirmation).

4.3.3.5.3 Get shared memory (GSM)

The PDL-user uses this service to read user data directly from the PDL receive update memory. The service is locally processed after the GSM.request primitive has arrived. The PDL communicates the successful processing of the service to the PDL-user by means of a GSM.confirm primitive (immediate confirmation).

4.3.4 PDL-PNM2 interface

4.3.4.1 General

This subclause defines the administrative PDL management services which are available to the PNM2, together with their service primitives and the associated parameters.

The PDL management is a part of the PDL that provides the management functions of the PDL requested by the PNM2. The PDL management handles the initialization, monitoring and error recovery in the PDL. Figure 14 shows the interface between PDL and PNM2 in the layer model.

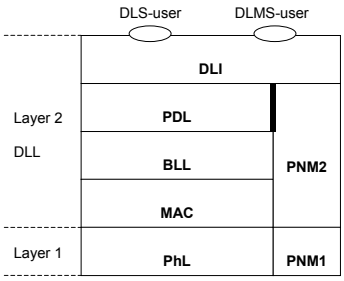


Figure 14 – Interface between PDL and PNM2 in the layer model

The service interface between PDL and PNM2 provides the following functions.

— Reset of the PDL protocol machine.

— Request and change of the current operating parameters of the PDL protocol machine.

— Indication of unexpected events, errors and status changes which occurred or are detected in the PDL.

4.3.4.2 Overview of the services

4.3.4.2.1 Available services

The PDL makes the following services available to the PNM2:

— Reset PDL,

— Set Value PDL or Get Value PDL,

— Event PDL.

The PDL services are described with service primitives (beginning with PDL_...).

4.3.4.2.2 Reset PDL

The PNM2 uses this required service to reset the PDL. Upon execution, the PNM2 receives a confirmation.

Service primitives:

— PDL_Reset.request

— PDL_Reset.confirm

4.3.4.2.3 Set Value PDL

The PNM2 uses this optional service to set new values to the PDL variables. Upon completion, the PNM2 receives a confirmation from the PDL whether the defined variables are assumed with the new value.

Service primitives:

— PDL_Set_Value.request

— PDL_Set_Value.confirm

4.3.4.2.4 Get Value PDL

The PNM2 uses this optional service to read the actual value of the PDL variables. The current value of the defined variable is transmitted with the confirmation from the PDL.

Service primitives:

— PDL_Get_Value.request

— PDL_Get_Value.confirm

4.3.4.2.5 Event PDL

The PDL uses this required service to inform the PNM2 about certain detected events or errors in the PDL.

Service primitive:

— PDL_Event.indication

4.3.4.3 Overview of the interactions

Figure 15 and Figure 16 show the time relations of the service primitives:

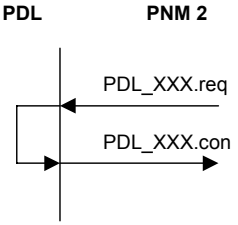


Figure 15 – Reset, Set Value and Get Value PDL services

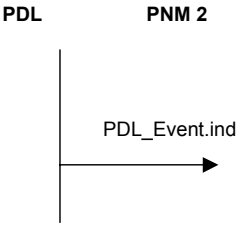


Figure 16 – Event PDL service

4.3.4.4 Detailed definition of the services and interactions

4.3.4.4.1 PDL_Reset

The PDL_Reset service is mandatory. The PNM2 transmits a PDL_Reset.request primitive to reset the PDL protocol machine (see Table 13).

Table 13 – PDL_Reset

Parameter name	Request	Confirm
Argument	M	
Result(+)		M

4.3.4.4.2 PDL_Set_Value

The PDL_Set_Value service is optional. The PNM2 transfers a PDL_Set_Value.request primitive to the PDL to set a defined PDL variable with a desired value. After receipt of this primitive, the PDL tries to select the variable and to set the new value. Upon execution, the PDL transfers a PDL_Set_Value.confirm primitive to the PNM2 (see Table 14).

Table 14 – PDL_Set_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
desired_value	M	
Result(+)		M

variable_name:

This parameter defines the PDL variable which is set to a new value.

desired_value:

This parameter declares the new value for the PDL variable.

Table 15 provides information on which PDL variable may be set to which new value.

Table 15 – PDL variables

Name of PDL variable	Value range	Default
max_spa_retry	0,2,4,6, … 14	14
max_swa_count	0 … 255	128
add_wait	1, 2, 3, 4	4
start_bus_cycle	ON, OFF	OFF
trigger_mode	network_triggered (NT), application_triggered (AT)	NT
max_dlsdu_size_from_req	1 … 256 (see note)	256
max_dlsdu_size_from_res	1 … 256 (see note)	256
max_receiving_queue_depth	1 … 256 (see note)	256
max_sending_queue_depth	1 … 256 (see note)	256
NOTE Only for PDL_Data_Ack services and each link.		

4.3.4.4.3 PDL_Get_Value

The PDL_Get_Value service is optional. The PNM2 transfers a PDL_Get_Value.request primitive to the PDL to read out the current value of a defined PDL variable. After the PDL has received this primitive, it tries to select the defined variable and to transfer its current value to the PNM2 by means of a PDL_Get_Value.confirm primitive (see Table 16).

Table 16 – PDL_Get_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
Result(+)		M
current_value		M

variable_name:

This parameter defines the PDL variable, whose value should be read.

current_value:

This parameter contains the desired value of this PDL variable.

Only those PDL variables can be read, which can also be written by the service PDL_Set_Value.request.

4.3.4.4.4 PDL_Event

The PDL_Event service is mandatory. The PDL transfers a PDL_Event.indication primitive to the PNM2 to inform it about detected events or errors in the PDL (see Table 17).

Table 17 – PDL_Event

Parameter name	Indication
Argument	M
event	M

event:

This parameter defines the value of the detected event or error cause in the PDL according to Table 18:

Table 18 – Events

Name	Meaning	Mandatory/optional
PDL_cycle_end	The receive update memory was updated, and the contents of the transmit update memory are transmitted to the BLL	O

4.3.5 Data transfer procedures from a queue

4.3.5.1 Bus access and data transfer mechanism

4.3.5.1.1 Synchronization cycle

Before starting of data transfer between the master and the slave(s), the PDL layers on all devices shall start with a synchronization cycle. In this cycle, a synchronization message resets the frame count bit flags in all devices to a defined value. In addition, the master started with the transmit of configure data to all slaves. After receiving the new configure data, all slaves shall initialize themselves with the new received configure values.

The frame count bits prevent a multiplication of messages at the confirming and/or responding device (responder), as these would cause the loss of positive acknowledgements.

A synchronization cycle only takes place for one communication relationship, that is, between the PDL protocol machine of the master and the PDL protocol machine of a slave. A synchronization cycle is initiated in the following cases:

— after a hardware reset,

— after a reset of the PDL layer by the PDL-user,

— after the detection of protocol errors,

— after a multiple data cycle error (max_swa_count time expired), and

— after a multiple SPA_acknowledge_timeout (the SPA acknowledge timeout occurred max_spa_retry-times).

In the first two cases the buffers and queues of the protocol layer for sending and receiving of messages are cleared from the concerned devices. Thus, all requests, confirmations and indication stored in these buffers are lost. In the remote device, however, no buffers are cleared. After the synchronization cycle this device tries to transmit the interrupted send message again.

In all other cases no buffers are cleared in any device. Upon a successful synchronization, both devices re-try to carry out orders of the application which have not yet been completed.

4.3.5.1.2 SVA message

Upon a successful synchronization and before interrupted messages are sent again, the master sends a SVA Message ("Send Value with Acknowledge") to the slave. The SVA Message transfers variables for the parameterisation of the PDL protocol machine.

The SVA Message transmits max_swa_count. The max_swa_count variable has a default value of 128 and can be parameterized by means of PDL_Set_Value. The slave accepts this value as its own max_swa_count.

The max_swa_count variable shall be transferred. In addition, other variables may be specified.

4.3.5.1.3 Frame count bit

The frame count bit (FCB) prevents a multiplication of messages at the confirming and/or responding device (responder), as this would cause the loss of positive acknowledgements.

If a positive acknowledgement is lost for whatever reason, the requester tries to sent the previous message again. When this message has already been correctly received by the responder, this is indicated by an unchanged FCB. In this case the responder again sends the acknowledgement to the requester, directly after the receipt of the first message segment. Then, the requester stops the repeated sending.

If a new message is to be sent the FCB shall be changed. To ensure that the requester FCB (transmit FCB) and the responder FCB (receive FCB) of the remote device have the same initial value after the initialization of the layer 2 and after protocol errors, there will be a synchronization with FCB_SET messages. The FCBs are set to '1' if the synchronization was successful.

There is a FCB pair for both transmission directions (one transmit and one receive FCB for each direction) (see Figure 17).

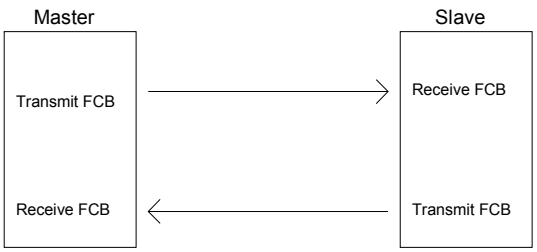


Figure 17 – Transmit and receive FCBs on the master and slave sides

4.3.5.1.4 Data transmission of bus cycles with errors

If a data cycle error occurs during the transmission of a SPA or SVA Message, the queue is not completely transmitted again. The transmission is rather continued with the queue parts that follow the error.

The master responds to cycle errors. Thus, a distinction is to be made between the two transmission directions **master → slave** and **slave → master**. If a cycle error occurs, this does not have any influence on the PDL protocol machine.

1) Data transmission: master → slave

If the master detects a data cycle error while queue is transmitted, the transmission shall be repeated from the error onwards. In this case the master communicates the error to the slave by means of a SWA Message.

Figure 18 and Figure 19 clarify the transmission master → slave with SWA Message. The numbering corresponds to the time sequence:

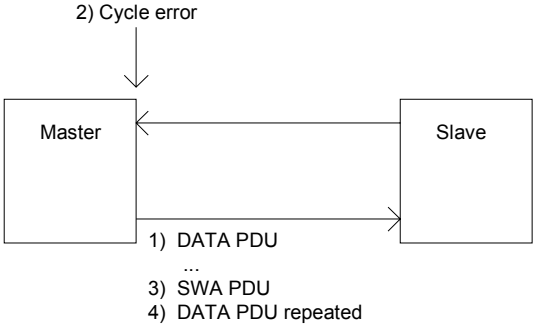


Figure 18 – Data transmission master → slave with SWA Message

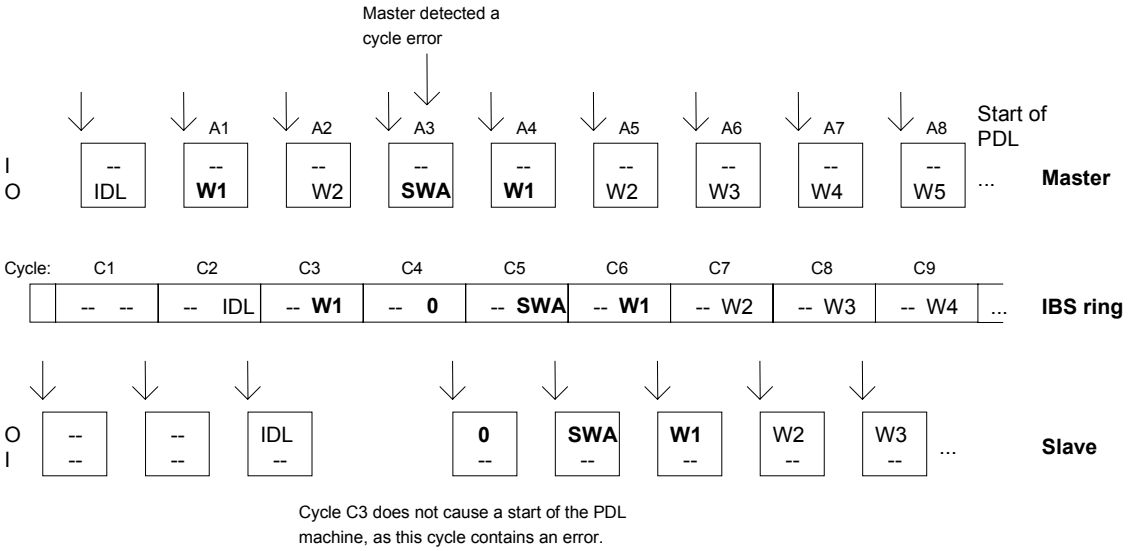


Figure 19 – Time sequence of the data transmission master → slave with SWA Message

2) Data transmission: slave → master:

If the master detects a cycle error when it receives a PDU, the slave will be announced immediately from the master by means of a RWA PDU. The slave shall confirm this RWA PDU with a SWA PDU, before the DATA PDU is sent again. The master uses the SWA PDU to mark the beginning of repeated data transmission.

An outstanding data transmission sequence from master → slave is interrupted during the RWA Message transfer and after the exception handling the data transmission can be continued.

Figure 20 and Figure 21 clarify the transmission slave → master with RWA Message or SWA Message:

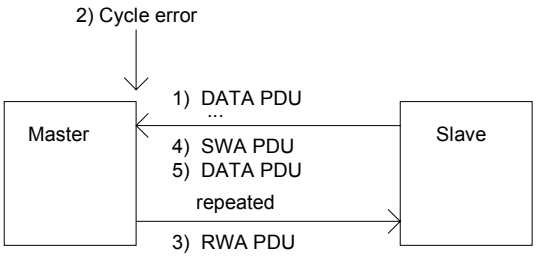


Figure 20 – Data transmission slave → master with SWA/RWA Message

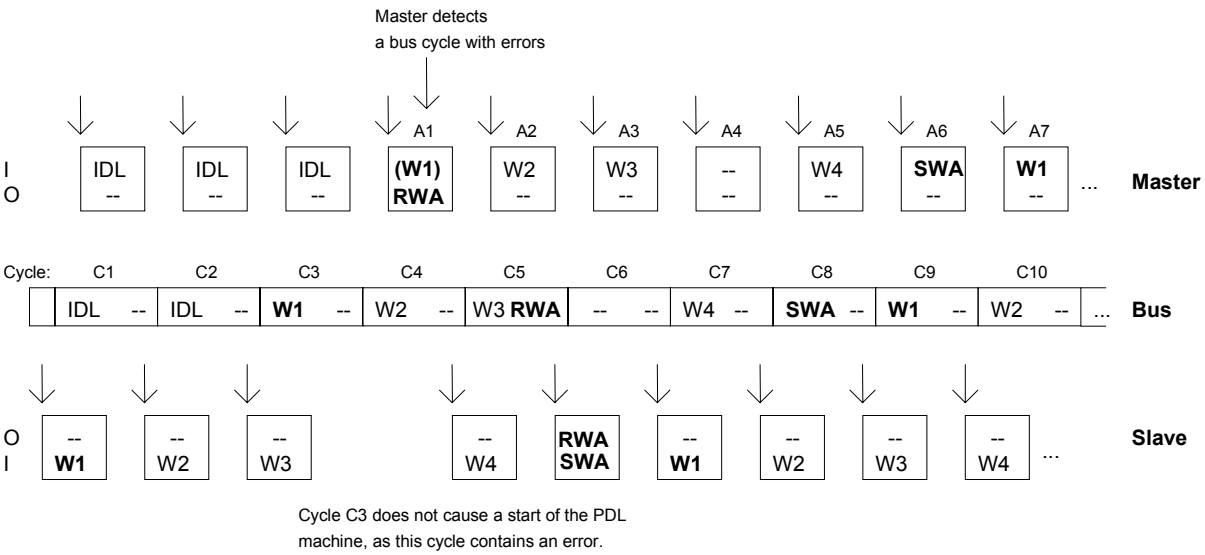


Figure 21 – Time sequence of the data transmission slave → master with SWA/RWA Message

4.3.5.2 Description of the time sequences

Master:

After each data cycle the PDL protocol machine is started once in the master for each slave in the ring having a parameter channel.

Among others, the protocol machine knows the following parameters:

Input parameters:

— The message segment which has been received during the last intact data cycle from the slave.

— The information whether a bus cycle error occurred during the last data cycle.

Output parameters:

— The message segment which is to be sent in the next cycle to the slave.

In Figure 22 these parameters are shown for each PDL protocol call A1…An, where I0…In identify the receive data for the master and the send data for the slave. Accordingly, O0…On are send data of the master and receive data of the slave.

Slave:

After the completion of a data cycle the PDL protocol machine is started on the slave, if the slave determined that the master did not send an IDL PDU and/or there is an outstanding send data request on the slave. IDL PDU are transmitted whenever there is no further user data are to be transmit. The last received message can be read and/or one outstanding send message can be prepared in the PDL protocol machine. The PDL pass the PDLPDU to the BLL for transmitting within the next data cycle to the master. The third line of the representation shows the starts of the PDL protocol machine of the slave (A1…A9) and the associated send and receive messages.

Bus:

In Figure 22, the middle row shows the cycles (C1…C9) and the messages which are transmitted in these cycles from the slave to the master and vice versa.

The transmission of a message from the TRANSMIT protocol machine of the master to the RECEIVE protocol machine of the slave requires two cycles, and the transmission from the TRANSMIT protocol machine of the slave to the RECEIVE protocol machine of the master requires three cycles.The TRANSMIT and RECEIVE protocol machines are components of the PDL protocol machine.

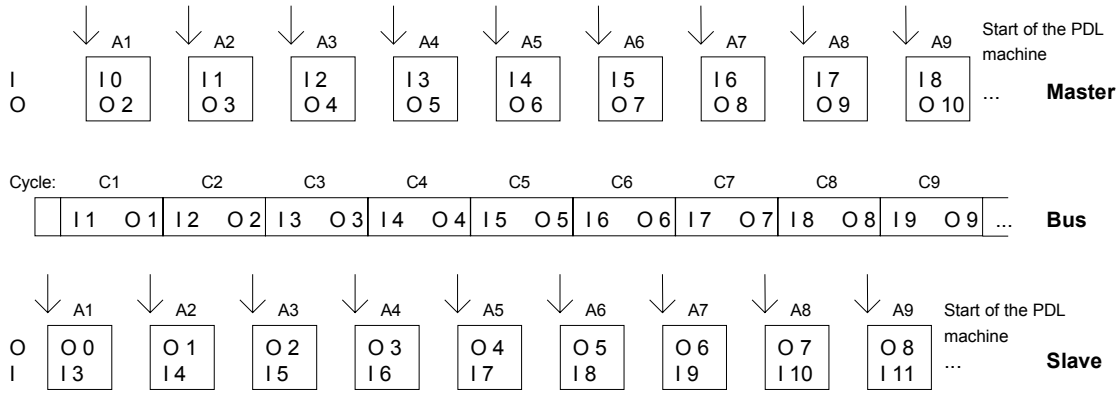


Figure 22 – Allocation of actions of the PDL protocol machines and data cycles

In the slave, the start of the PDL protocol machine for the sending and receiving of PDLPDU shall be completed before a further cycle end was indicated. Otherwise received data can be lost. The DLPDU cycle time depends with the respect from the number of slaves and from the data width of each slave which are connected to the DL-subnetwork.

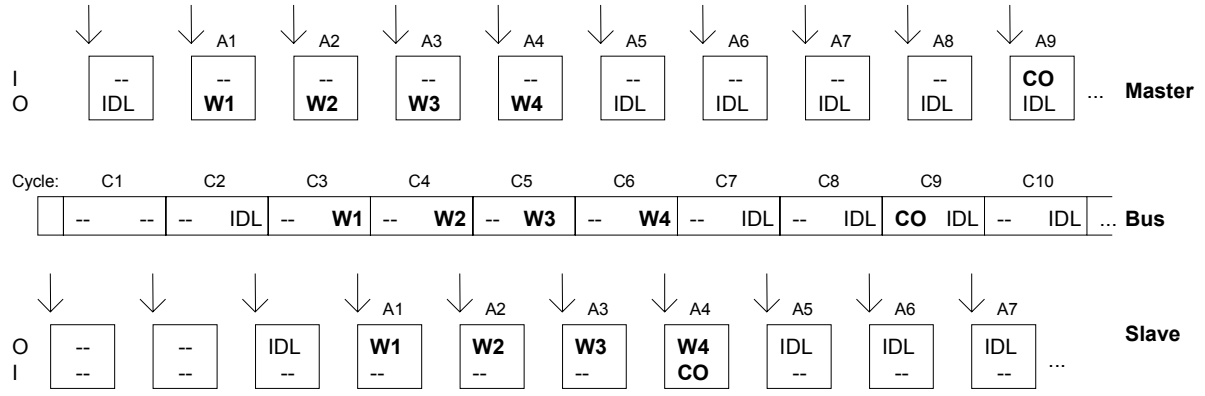


Figure 23 – Message transmission: master → slave

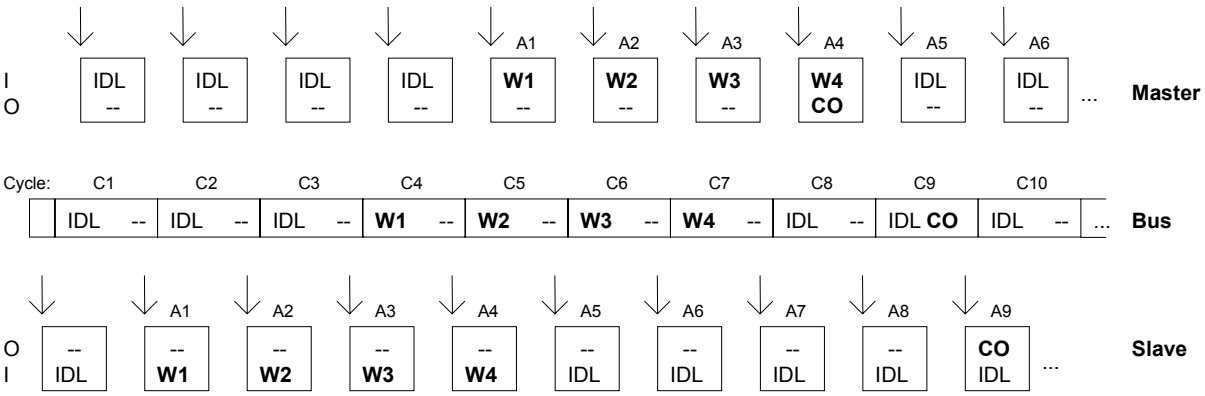


Figure 24 – Message transmission: slave → master

Figure 23 and Figure 24 show that at least DIST = 5 data cycles are needed between the sending of the last message, until a confirmation (CO) for this message is received.

Delays while confirmations may be also taken into the account, so the protocol need additional cycles for waiting of a confirmation. This number of additional cycles is stored in the variable "add_wait". On the master side the variable add_wait can be parameterized with the PDL_Set_Value (value range: 1 to 4). The contents of the variable add_wait is transmitted from the master to the slave within the FCB_SET PDU while the PDL protocol machines are synchronized.

4.3.5.3 Coding of the messages

4.3.5.3.1 Overview

The message length for a slave with the parameter channel consists at least one octet for FCT and additional the length of the data 1, 3 or 7 octets, depending from the size of the message for transmission.

4.3.5.3.2 Structure of the code octet

4.3.5.3.2.1 General

Figure 25 shows the structure of a code octet, Table 19 shows the L_status encoding and Table 20 the FCT code.

B16 B15 B14	B13 B12 B11	B10	B9
L_status	FCT code	FCB	IDL

Figure 25 – Code octet of a PDLPDU

Table 19 – Encoding of the L_status

L_status (B16 B15 B14)	Generation (Local/Remote)	Meaning
0 0 0	–	No confirmation
0 0 1	L / R	Acknowledgement positive
0 1 0	R	Acknowledgement negative, no resource available (buffer full)
0 1 1	L / R	Acknowledgement negative, multiple data cycle error (transmission via the bus only from the master to the slave)
1 0 0	R	Acknowledgement positive (repeated)
1 0 1	R	Acknowledgement negative, no resource available (repeated)
1 1 0	L	Acknowledgement negative, acknowledge_timeout
1 1 1	R	FCB_SET confirmation, the FCT code equals 0; FCB = 1!

Table 20 – FCT code (PDLPDU-Types)

Function code	PDL-PDU Types	Meaning
0 0 0	IDL PDU	The IDL PDU do not contain any information (except for a FCB_SET confirmation).
0 0 1	DATA PDU	The DATA PDU contains segments of user data or PDL variables.
0 1 0	SPA PDU	The SPA PDU defines the start of a new queue data transmission and contains at least the length information of DLSDU in octets-1 and segments of user data.
0 1 1	SVA PDU	The SVA PDU defines the start of a new variable data transmission and contains at least the length information of variable data in octets-1 and possibly PDL variables.
1 0 0	Don't spare	Reserved
1 0 1	RWA PDU	RWA PDU (Receive Word Again): The data octets of the message contain at least the amount of successfully received data octets+1 and possibly DLSDU data or PDL variables.
1 1 0	SWA PDU	SWA PDU (Send Word Again): The data octets of the message contain at least the amount of successfully sent data octets + 1 and possibly DLSDU data or PDL variables.
1 1 1	FCB_SET PDU	The FCB_SET PDU contains L_status = 0 and FCB = 1, for an FCB_SET request from the master to the slave. The first data octet of the message contains user data for the PDL protocol machine of the slave.

4.3.5.3.2.2 FCB

Frame count bit : 0/1, alternating bit.
The FCB is only relevant within the start segment of SPA Message and SVA Messages.

4.3.5.3.2.3 Idle message

If the IDL-Bit (see Figure 25; Bit 9) equals 0, the message does not contain any information and is called IDLE message.

4.3.5.3.3 Messages with a size of one word

4.3.5.3.3.1 General

Figure 26 shows the structure of a message with a size of one word.

Code octet								**Data octet**							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
X	X	X	X	X	X	X	0	X	X	X	X	X	X	X	X

Figure 26 – Structure of a message with a size of one word

4.3.5.3.3.2 Call messages

1) **SPA Message** (PDLPDU-Type for transmission of queued user data):

In the L_status of the code octets, a confirmation for a SPA Message of the remote device can be transmitted at the same time (see Figure 27).

Code octet								**Data octet**							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
X	X	X	0	1	0	FCB	1	n-1: DLSDU length in octets-1: $1 \leq = n \leq 256$							
X	X	X	0	0	1	X	1	1st DLSDU octet							
X	X	X	0	0	1	X	1	2nd DLSDU octet							
X	X	X	0	0	1	X	1	Nth DLSDU octet							

Figure 27 – Structure of a SPA Message

2) **SVA Message** (DLPDU-Type for transmission of configuration data):

After a confirmed FCB_SET, further variables are transmitted from the master to the slave with the 'Send Value with Acknowledge' message. On the master side these variables can be set with the PDL_Set_Value.request and are also valid for the slave after the transmission of the SVA Message. The number of variable octets-1 is transmitted in the data octet of the start segment.

The swa_count variable shall be transferred. In addition, further variables may be specified (see Figure 28).

Code octet								Data octet							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
X	X	X	0	1	1	FCB	1	n-1: number of variable octets-1							
X	X	X	0	0	1	X	1	Swa_count (mandatory)							
X	X	X	0	0	1	X	1	2nd variable octet (optional)							
								…							
X	X	X	0	0	1	X	1	Nth variable octet (optional)							

Figure 28 – Structure of a SVA Message

3) **FCB_SET Message:**

This message is used to synchronize the PDL protocol machines of a parameter channel on the master and on the slave, that is, to set the FCBs to a defined value (see 4.3.5.1.3).

In the first data octet, the master also transmits the protocol specific parameters which have been set with the PDL_Set_Value service (see Figure 29).

Code octet								Data octet							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
0	0	0	1	1	1	1	1	Coding see below							

Figure 29 – Structure of a FCB_SET Message

4.3.5.3.3.3 Control segments

1) **RWA Message**:

With the RWA (Read Word Again) message the master indicates to the slave during a SPA Message that the slave has received one message without errors. In the first data octet the RWA Message contains the number of DLSDU octets which have been received without errors (see Figure 30).

Code octet								Data octet							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
X	X	X	1	0	1	X	1	k Number of correctly received DLSDU octets during the current SPA Messages							

Figure 30 – Structure of a RWA Message

2) **SWA Message**:

For the SWA Message a difference between master and slave has to be observed, as only the master can immediately detect a bus cycle with errors (see Figure 31).

— When a SPA Message is sent from the slave to the master, its identifies the beginning of repeated data transmission. Here, the SPA Message is sent after a RWA Message.

— If the master detects a data cycle with errors when a SPA or SVA Message is sent to the slave, the transmission is repeated from the error onwards. This error-1 = number of PDU sent without errors is communicated to the slave by means of a SWA Message.

Code octet								Data octet							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
X	X	X	1	1	0	X	1	k Number of data octets sent without errors							
X	X	X	0	0	1	X	1	(k+1)th data octet							
X	X	X	0	0	1	X	1	(k+2)th data octet							
								...							
X	X	X	0	0	1	X	1	N-th data octet							

Figure 31 – Structure of a SWA Message

4.3.5.3.3.4 Response messages

1) **Data confirmation** (confirmation for SPA or SVA Messages):

The higher three bits of the code octet contain the confirmation for the call messages described above. This confirmation can also be transmitted together with a message start segment of SPA or SVA, control or user data segment (exception: FCB_SET PDU) (see Figure 32).

Code octet								Data octet							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
L_status			X	X	X	X	1	X	X	X	X	X	X	X	X

Figure 32 – Structure of a confirmation for SPA or SVA Messages

2) **FCB_SET confirmation:**

The FCB_SET confirmation acknowledges a FCB_SET request (see Figure 33).

Code octet								Data octet							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
1	1	1	0	0	0	1	1	Coding see below							

Figure 33 – Structure of a FCB_SET as confirmation

Coding of the data octet for FCB_SET Message as request and confirmation

In the data octet of the FCB_SET as request or the FCB_SET as confirmation, the master transmits the protocol specific parameters which have been set with the PDL_Set_Value service (see Figure 34):

Code octet								Data octet							
B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1
X	X	X	X	X	X	X	X			a)			b)		c)

Figure 34 – Structure of the data octet for FCB_SET as requests and confirmations

a) **B8-B6: err_max**
The contents of the variable err_max should be multiplied with 2, accordingly to get the number of attempts for sending a same message

b) **B5-B3: Reserved**

c) **B2-B1: add_wait**
The variable add_wait-1 gives the number of cycles that is additionally waited for a confirmation.

4.3.5.3.4 Queue data with a segment size of more than one word

Example: Queue data size = 2 words, data transmission with SWA Message after a cycle error (see Figure 35).

Code octet							1st data octet	2nd data octet	3rd data octet							
X	X	X	1	1	0	X	1	k number of message octets sent without an error	(k+1)th message octet	(k+2)th message octet						
X	X	X	0	0	1	X	1	(k+3)th message octet	(k+4)th message octet	(k+5)th message octet						
X	X	X	1	1	0	X	1	(N-1)th message octet	N-th message octet	X	X	X	X	X	X	X	X

Figure 35 – Structure of a message with a size of more than one word

Number of segments to be transmitted for a message (DLSDU):

Calculation: $\mathbf{G_m(N) = (N-1)/m + 1}$

where

N is the user data quantity (including the number of octets sent/received without errors or the data length)

M is the number of data octets per segment (PDU)

$\mathbf{G_m(N)}$ is the number of PDU.

NOTE (N-1) / m is a not rounding integer division

Example:

A queue data with 13 octets is to be transmitted. N = 14, including the data length. The segment size is two words, that is, three data octets are available in the segment (m = 3):

$G_3(14) = (14-1) / 3 + 1 = 4 + 1 = 5$

4.3.6 PDL protocol machines

4.3.6.1 PDL base protocol machine

4.3.6.1.1 Description of the states

The PDL base protocol machine provides the functionality of the PDL and has the following five states:

4.3.6.1.2 PDL_INIT

State after power on. This state is only left when all required PDL protocol machines have been generated by means of the communication relationship list entries and the PDL receive and transmit update memories are initialized.

4.3.6.1.3 PDL_RECEIVE_UPDATE

In this state the PDL takes the user data out of the PDLSDU which was received from the BLL and updates the PDL receive update memory of layer 2.

4.3.6.1.4 PDL_PM_ACTIVE

With the transition into this state all protocol machines are initiated at the same time. This state shall only be left after all protocol machines have been processed.

4.3.6.1.5 PDL_TRANSMIT_UPDATE

In this state the PDL takes the user data out of the transmit update memory of layer 2, generates the user data part of the PDLSDU to be sent to the BLL and transmits the PDLSDU with a BLL_Data.request (master side) or BLL_Data.response (slave side) primitive to the BLL.

4.3.6.1.6 PDL_WAIT_FOR_PDLSDU

In this state the PDL base protocol machine (see Figure 36) waits for a BLL_Data.confirm(master side) or BLL_Data.indication (slave side) primitive of the BLL.

In all states (except PDL_INIT) the PDL_Data_Ack.request primitives, which are passed from the PDL-user to the PDL, are processed as follows:

First, the service request is locally confirmed by means of a PDL_Data_Ack.confirm primitive. Then at the receiving side, the PDL generates a PDL_Data_Ack.indication primitive that is passed on to the PDL-user, which can be identified by the remote address.

A reset generally causes a transition to the PDL_INIT state.

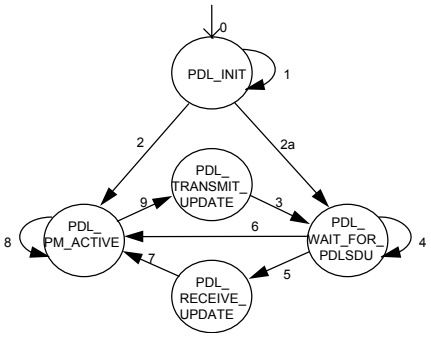


Figure 36 – PDL base protocol machine

The state transitions from every state (except PDL_INIT) to the same state when a PDL_Data_Ack.request primitive arrives from the PDL-user are not shown to simplify matters. However, they are listed in the state transition table.

4.3.6.1.7 Description of the transitions

Table 21 shows the state transitions of the PDL base protocol machine.

Table 21 – State transitions of the PDL base protocol machine

Initial state **event** \ condition ⇒ action	**Transition**	**Follow-up state**
After Power on	**0**	**PDL_INIT**
PDL_INIT \ initialization of the PDL protocol machines and the update memories are not yet completed	**1**	**PDL_INIT**
PDL_INIT ***(master side only)*** \ initialization completed ⇒ start of all PDL protocol machines	**2**	**PDL_PM_ACTIVE**
PDL_INIT ***(slave side only)*** \ initialization completed	**2a**	**PDL_WAIT_FOR_PDLSDU**
PDL_TRANSMIT_UPDATE ⇒ copy of user data from TUM to the PDLSDU AND send PDLSDU to BLL with BLL_Data.request/response	**3**	**PDL_WAIT_FOR_PDLSDU**
PDL_WAIT_FOR_PDLSDU \ no BLL_Data.confirm/indication received	**4**	**PDL_WAIT_FOR_PDLSDU**
PDL_WAIT_FOR_PDLSDU BLL_Data.confirm/indication received \ update_info == OK	**5**	**PDL_RECEIVE_UPDATE**
PDL_WAIT_FOR_PDLSDU BLL_Data.confirm received \ update_info == NOK ⇒ start of all PDL protocol machines ⇒ if trigger_mode == AT, then start_bus_cycle = OFF	**6**	**PDL_PM_ACTIVE**
PDL_RECEIVE_UPDATE ⇒ copy user data from PDLSDU to RUM ⇒ start all PDL protocol machines ⇒ if trigger_mode == AT, then start_bus_cycle = OFF	**7**	**PDL_PM_ACTIVE**
PDL_PM_ACTIVE \not all PDL protocol machines are stopped OR (***master side only)*** start_bus_cycle == OFF	**8**	**PDL_PM_ACTIVE**
PDL_PM_ACTIVE \ all PDL protocol machines are stopped AND (***master side only)*** start_bus_cycle == ON	**9**	**PDL_TRANSMIT_UPDATE**
Any_state PDL_Reset.request received ⇒ PDL_Reset.confirm		**PDL_INIT**
Any_state PDL_Data_Ack.request (..,rem_add,..) ⇒ PDL_Data_Ack.confirm ⇒ PDL_Data_Ack.indication to PDL-user with rem_add		**same_state**
Any_state all PDL management services ⇒ process and confirm the services		**same_state**

4.3.6.2 PDL protocol machine

4.3.6.2.1 Overview

For each slave with a parameter channel in the ring, the PDL base protocol machine of the master manages a PDL protocol machine. The PDL base protocol machine of a slave with a parameter channel, however, has only one PDL protocol machine for this parameter channel.

A PDL protocol machine processes all PDL_Data_Ack services which are transmitted via the parameter channel to the PDL and has, in particular, the following tasks:

Connection:

— Establishing the PDL connection to the remote device.

Sending side:

— Testing the send requests (PDL_Data_Ack.request) for plausibility
— Process the send requests
— Confirming the send requests (PDL_Data_Ack.confirm)
— Segmentation of the DLSDU to data segments
— Error detection while data transfer
— Passing the Message segments and confirmations with the BLL_Data.request/response to the Basic Link Layer (BLL).

Receiving side:

— Receiving confirmations and data segments with the BLL_Data.confirm
— Re-assembling the data segments to DLSDU
— Transferring received DLSDU to the PDL-user with PDL_Data_Ack.indication.

Figure 37 explains the PDL protocol machine in general for a master and a slave. The differences between master and slave are given. Each PDL protocol machine controls and uses a TRANSMIT and a RECEIVE protocol machine.

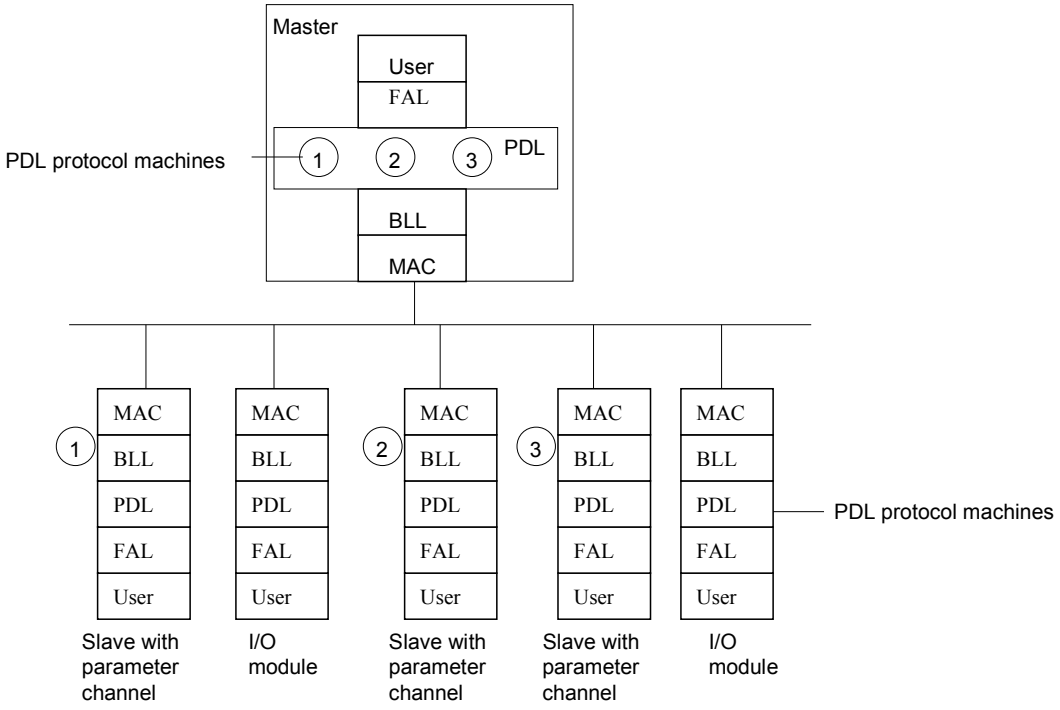


Figure 37 – Locations of the PDL and the PDL protocol machines in the master and slaves

4.3.6.2.2 Counters and flags in the protocol machines

4.3.6.2.2.1 Counters

The counters described in the following are used in the PDL protocol machines for the parameter channel (see Table 22):

Table 22 – Counters of the PDL protocol machines

Counter	Description	Used in protocol machine
Ccycle	Counter for the number of data cycles since the start signal of a SPA or SVA Message has been sent. This counter checks the correctness of repeated confirmations.	TRANSMIT
Ccerr	Counter for the number of defective data cycles after another while a message is sent.	TRANSMIT
Cconf	Counter for the number of cycles during which a confirmation is awaited.	PDL protocol and TRANSMIT
Creq_retry	Counter for the attempts to send messages when confirmations were lost due to defective data cycles.	TRANSMIT
CSWA	Counter for the number of cycles during which a SWA Message is awaited.	RECEIVE

4.3.6.2.2.2 Flags

connection

The PDL protocol machine manages a flag called "connection". This flag is the same as DISCONN, when the PDL protocol machine of the master is not synchronized with that of a

slave. In this case, the master or slave continuously sends synchronization messages until one message is confirmed by the remote device. Then connection == READY is set, that is, the PDL protocol machines are synchronized (see Table 23).

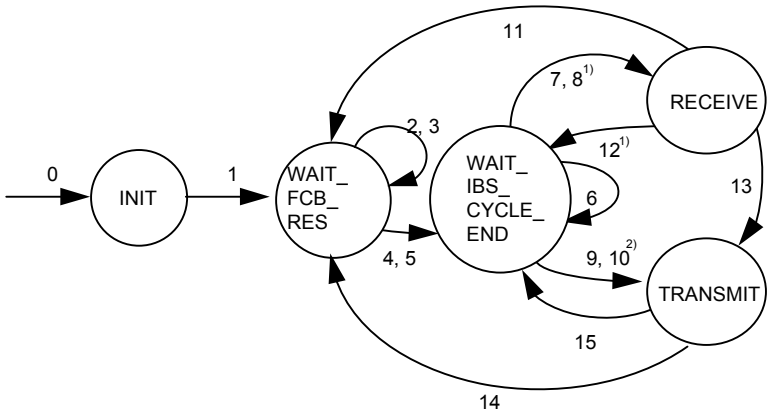
Table 23 – Meaning of the "connection" flag

Connection	Description
DISCONN	The PDL protocol machines are not synchronized
READY	The PDL protocol machines are synchronized. The PDL layer is ready to transmit a message

4.3.6.2.3 Description of the states

4.3.6.2.3.1 Overview

Figure 38 shows the PDL protocol machine.



1) Master only

2) Slave only

Figure 38 – PDL protocol machine

Transitions as a result of a received PDL_Reset.request primitive always go from every state to the INIT state. These transitions are not shown individually but are described by the transition 0.

4.3.6.2.3.2 INIT

Initialization of the PDL protocol machine

4.3.6.2.3.3 WAIT_FCB_RES

Synchronization

4.3.6.2.3.4 WAIT_IBS_CYCLE_END

The PDL protocol machine is synchronized. This state controls whether the RECEIVE and/or the TRANSMIT protocol machine is started.

4.3.6.2.3.5 RECEIVE

The RECEIVE protocol machine is started.

4.3.6.2.3.6 TRANSMIT

The TRANSMIT protocol machine is started.

4.3.6.2.4 Description of the transitions

The PDL protocol machine is started by the PDL base protocol machine when a data cycle has been completed and new received data (PDLSDU) from the BLL is available. If the received data has been processed, and new transmit data is not available, the PDL protocol machine stops itself. In the stopped state, the protocol machine does not respond to events. Only a PDL_Reset.request causes a reset of the protocol machine at any time (see Table 24).

Table 24 – State transitions of the PDL protocol machine

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
PDL_Reset.request received ⇒ initialize PDL protocol machine, reject non-processed PDL_Data_Ack services	**0 Reset**	**INIT**
INIT initialization of the PDL protocol machine completed C_{conf} = 0, stop PDL protocol machine	**1 TFCB_Req1**	**WAIT_FCB_RES**
WAIT_FCB_RES C_{conf} ≤ DIST+add_wait \neither FCB_SET request (slave side only) sent nor FCB_SET confirmation received ⇒ increment C_{conf}, stop PDL protocol machine	**2 Wait1**	**WAIT_FCB_RES**
WAIT_FCB_RES C_{conf} > DIST+add_wait \neither FCB_SET request (slave side only) nor FCB_SET confirmation received ⇒ send FCB_SET request, C_{conf} = 0, stop PDL protocol machine	**3 TFCB_Req2**	**WAIT_FCB_RES**
WAIT_FCB_RES FCB_SET request received ⇒ send FCB_SET confirmation, set receive and transmit FCB-Fag, connection = READY, reset TRANSMIT and RECEIVE protocol machine, ***master side only:*** send enable of SVA PDU (see TRANSMIT protocol machine), stop PDL protocol machine	**4 RFCB_Req1**	**WAIT_IBS_CYCLE_END**
WAIT_FCB_RES FCB_SET confirmation received ⇒set receive and transmit FCB, connection = READY, reset TRANSMIT and RECEIVE protocol machine., ***master side only:*** send enable SVA PDU (see TRANSMIT protocol machine), stop PDL protocol machine	**5 RFCB_Conf**	**WAIT_IBS_CYCLE_END**

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
WAIT_IBS_CYCLE_END FCB_SET as request received ⇒ send FCB_SET as confirmation, set receive and transmit FCB, connection = READY, reset TRANSMIT and RECEIVE protocol machines, ***master side only***: send enable SVA Message (see TRANSMIT protocol machine), stop PDL protocol machine	**6 RFCB_Req2**	**WAIT_IBS_CYCLE_END**
WAIT_IBS_CYCLE_END receipt of a confirmation and/or a message segments **except** FCB_SET request and FCB_SET confirmation, (**slave side only**), no RWA PDU ⇒ start RECEIVE protocol machine	**7 Recv_Req**	**RECEIVE**
WAIT_IBS_CYCLE_END ***(master side only)*** status of RECEIVE protocol machine == WAIT_SWA ⇒ start RECEIVE protocol machine	**8 Wait_SWA**	**RECEIVE**
WAIT_IBS_CYCLE_END receipt of an IDLE message ⇒ start TRANSMIT protocol machine	**9 Send_Req**	**TRANSMIT**
WAIT_IBS_CYCLE_END ***(slave side only)*** RWA PDU received ⇒ start TRANSMIT protocol machine (to send a SWA PDU)	**10 Recv_RWA**	**TRANSMIT**
RECEIVE RECEIVE protocol machine stopped AND connection == DISCONN ⇒ send FCB_SET request, C_{conf} = 0, stop PDL protocol machine	**11 TFCB_Req3**	**WAIT_FCB_RES**
RECEIVE ***(master side only)*** RECEIVE protocol machine stopped AND connection == READY AND RECEIVE protocol machine has sent RWA PDU ⇒ stop PDL protocol machine	**12 TM_Disable**	**WAIT_IBS_CYCLE_END**
RECEIVE RECEIVE protocol machine stopped AND connection == READY AND RECEIVE protocol machine ***(master side only)*** has not sent a RWA PDU ⇒ start TRANSMIT protocol machine	**13 Recv_OK2**	**TRANSMIT**
TRANSMIT TRANSMIT protocol machine stopped AND connection == DISCONN ⇒ send FCB_SET request, C_{conf} = 0, stop PDL protocol machine	**14 TFCB_Req4**	**WAIT_FCB_RES**
TRANSMIT TRANSMIT protocol machine stopped AND connection == READY ⇒ stop PDL protocol machine	**15 Transm_OK**	**WAIT_IBS_CYCLE_END**

4.3.6.3 TRANSMIT protocol machine

4.3.6.3.1 Description of the states

4.3.6.3.1.1 Overview

Figure 39 shows the TRANSMIT protocol machine.

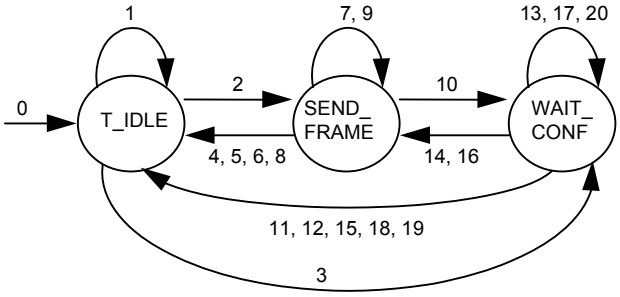


Figure 39 – TRANSMIT protocol machine

Transitions as a result of a PDL_Reset primitive always go from every state to the T_IDLE state. These transitions are not shown individually but are described by the transition 0.

4.3.6.3.1.2 T_IDLE

No message is being sent.

4.3.6.3.1.3 SEND_FRAME

A message is being sent.

4.3.6.3.1.4 WAIT_CONF

A message was sent. The confirmation of the remote device is being waited for.

4.3.6.3.2 Description of the transitions

The TRANSMIT protocol machine is started by the higher-level PDL protocol machine. Then the TRANSMIT protocol machine merely carries out one transition and stops itself. In the table which describes the state transitions the 'stop TRANSMIT protocol machine' action was not included for all transitions to simplify matters. In the stopped state the protocol machine does not respond to events. Only a PDL_Reset.request causes a reset of the protocol machine at any time (see Table 25).

Table 25 – State transitions of the TRANSMIT protocol machine

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
PDL_Reset.request ⇒ reset TRANSMIT protocol machine	**0 Reset**	**T_IDLE**
T_IDLE there is no send enable for a SVA PDU (*master only*) and no PDL_Data_Ack.request ⇒ no action (waiting)	**1 Wait1**	**T_IDLE**
T_IDLE send enable for SVA PDU (*master only*) or PDL_Data_Ack.request \PDU shall be transmitted with more than one segment ⇒ enter start segment of the SPA/SVA PDU, $C_{cycle} = C_{cerr} = C_{req\_reply} = 0$	**2 Request1**	**SEND_FRAME**
T_IDLE send enable for SVA PDU (*master only*) or PDL_Data_Ack.request \PDU can be transmitted with one segment ⇒ enter start segment of the SPA/SVA PDU, $C_{cycle} = C_{cerr} = C_{req\_reply} = 0$, $C_{conf} = 0$	**3 Request2**	**WAIT_CONF**
SEND_FRAME repeated confirmation for SPA PDU received (repeated positive or repeated queue full) \DIST ≤ C_{cycle} ≤ DIST+add_wait ⇒ change transmit FCB, sent SPA confirmation	**4 Recv_Conf1**	**T_IDLE**
SEND_FRAME repeated confirmation for SVA PDU received (repeated positive or repeated queue full) \DIST ≤ C_{cycle} ≤ DIST+add_wait ⇒ change transmit FCB	**5 Recv_Conf2**	**T_IDLE**
SEND_FRAME repeated confirmation for SPA or SVA PDU received (repeated positive or repeated queue full) \C_{cycle} < DIST OR C_{cycle} > DIST+add_wait ⇒ connection = DISCONN	**6 Disconn1**	**T_IDLE**
SEND_FRAME last data cycle contained errors \C_{cerr} ≤ max_swa_count ⇒ increment C_{cerr} , enter SWA PDU, increment C_{cycle}	**7 Cycl_Err1**	**SEND_FRAME**
SEND_FRAME last data cycle contained errors \C_{cerr} > max_swa_count ⇒ connection = DISCONN	**8 Mult_Err1**	**T_IDLE**
SEND_FRAME last data cycle completed without errors \more than one data segment to be sent ⇒ enter DATA PDU, increment C_{cycle}	**9 Send_Segm1**	**SEND_FRAME**
SEND_FRAME last data cycle completed without errors \last data segment to be sent ⇒ enter DATA PDU, $C_{conf} = 0$	**10 Send_Segm2**	**WAIT_CONF**

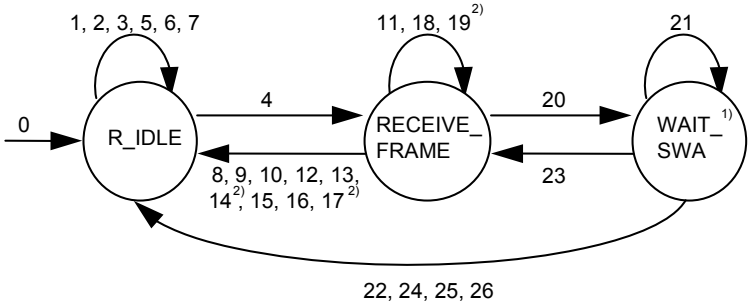
Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
WAIT_CONF confirmation SPA received (positive, repeated pos., queue full or repeated queue full) ⇒ change transmit FCB, enter SPA PDU	**11 Recv_Conf3**	**T_IDLE**
WAIT_CONF confirmation for SVA PDU received (positive, repeated pos., queue full or repeated queue full) ⇒ change transmit FCB	**12 Recv_Conf4**	**T_IDLE**
WAIT_CONF no confirmation received \C_{conf} > DIST+add_wait AND $C_{req\text{-}retry}$ ≤ max_req_retry AND queued data can be transmitted with one segment ⇒ increment C_{req_retry} , enter start segment of SPA or SVA PDU, $C_{cycle} = C_{cerr} = 0$, $C_{conf} = 0$	**13 Timeout1**	**WAIT_CONF**
WAIT_CONF no confirmation received \C_{conf} > DIST+add_wait AND C_{req_retry} ≤ max_req_retry AND queued data shall be transmitted with more than one segment ⇒ increment C_{req_retry} , enter start segment of the SPA or SVA PDU, $C_{cycle} = 0$, $C_{cerr} = 0$	**14 Timeout2**	**SEND_FRAME**
WAIT_CONF no confirmation received \C_{conf} > DIST+add_wait AND C_{req_retry} > max_req_retry ⇒ connection = DISCONN	**15 Mult_TO**	**T_IDLE**
WAIT_CONF last data cycle contained errors \C_{cerr} ≤ max_swa_count AND more than one data segment is to be sent repeatedly ⇒ increment C_{cerr} , enter SWA PDU	**16 Cycle_Err2**	**SEND_FRAME**
WAIT_CONF last data cycle contained errors \C_{cerr} ≤ max_swa_count AND SWA PDU can accept all data which is to be sent repeatedly ⇒ increment C_{cerr}, enter SWA PDU in PDLSDU, $C_{conf} = 0$	**17 Cycle_Err3**	**WAIT_CONF**
WAIT_CONF last data cycle contained errors \C_{cerr} > max_swa_count ⇒ connection = DISCONN	**18 Mult_Err2**	**T_IDLE**
WAIT_CONF repeated confirmation for SPA or SVA PDU received (repeated positive or repeated queue full) \C_{cycle} < DIST OR C_{cycle} > DIST+add_wait ⇒ connection = DISCONN	**19 Disconn2**	**T_IDLE**
WAIT_CONF no queue received \C_{conf} ≤ DIST+add_wait ⇒ increment C_{conf} and C_{cycle}	**20 Wait2**	**WAIT_CONF**

4.3.6.4 RECEIVE Protocol Machine

4.3.6.4.1 Description of the states

4.3.6.4.1.1 Overview

Figure 40 shows the RECEIVE protocol machine.



1) The WAIT_SWA state as well as the transitions from and to WAIT_SWA are for the master only
2) For a slave only

Figure 40 – RECEIVE protocol machine

Transitions as a result of PDL_Reset.request primitive always go from every state to the R_IDLE state. These transitions are not shown individually but are described by the transition 0.

4.3.6.4.1.2 R_IDLE

No message is being received.

4.3.6.4.1.3 SEND_FRAME

A message is being received.

4.3.6.4.1.4 WAIT_SWA (master side only)

A RWA PDU was sent. The responding SWA PDU is being waited for.

4.3.6.4.2 Description of the transitions

The RECEIVE protocol machine is started by the higher-level PDL protocol machine. Then, the RECEIVE protocol machine merely carries out one transition and stops itself. In Table 26 which describes the state transitions, the stop 'RECEIVE protocol machine' action was not included for all transitions to simplify matters. In the stopped state the protocol machine does not respond to events. Only a PDL_Reset.request causes a reset of the protocol machine at any time.

Table 26 – State transitions of the RECEIVE protocol machine

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
PDL_Reset.request received ⇒ reset RECEIVE protocol machine	**0 Reset**	**R_IDLE**

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
R_IDLE no SVA (*slave only*) or SPA PDU received ⇒ no action (waiting)	**1 Wait1**	**R_IDLE**
R_IDLE SVA (*slave only*) or SPA PDU received \receive FCB ≠ await FCB AND no confirmation has yet been sent for this receive FCB ⇒ connection = DISCONN	**2 Disconn1**	**R_IDLE**
R_IDLE SVA (*slave only*) or SPA PDU received \receive FCB ≠ await FCB AND a confirmation has already been sent for this receive FCB ⇒ enter repeated confirmation in PDLSDU (repeated positive or repeated queue full)	**3 Rep_Recept1**	**R_IDLE**
R_IDLE SVA (*slave only*) or SPA PDU received \start segment does not contain the complete message ⇒ accept data octets	**4 RStart_Sgm1**	**RECEIVE_FRAME**
R_IDLE SPA PDU received \start segment does not contain a complete message AND memory available ⇒ accept data octets, enter positive confirmation in PDLSDU, change receive FCB, PDL_Data_Ack.indication to the PDL-user	**5 RStart_Sgm2**	**R_IDLE**
R_IDLE SPA PDU received \start segment contains complete message AND no more memory available ⇒ enter negative confirmation (queue full) in PDLSDU, change receive FCB	**6 RStart_Sgm3**	**R_IDLE**
R_IDLE (*slave only)* SVA PDU received \start segment contains complete message ⇒ accept data octets, enter positive confirmation in PDLSDU, change receive FCB	**7 RStart_Sgm4**	**R_IDLE**
RECEIVE_FRAME no DATA PDU, no SWA PDU and no SVA (*slave only*) or SPA PDU received ⇒ stop the sending of a message	**8 Segm_Err1**	**R_IDLE**
RECEIVE_FRAME SVA (*slave only*) or SPA PDU received \receive FCB ≠ FCB in start segment AND no confirmation was sent for the received PDU ⇒ connection = DISCONN	**9 Disconn2**	**R_IDLE**
RECEIVE_FRAME SVA (*slave only*) or SPA PDU received \receive FCB ≠ await FCB in start segment AND a confirmation has already been sent for the PDU ⇒ enter repeated confirmation in PDLSDU (repeated positive or repeated queue full)	**10 Rep_Recept2**	**R_IDLE**
RECEIVE_FRAME SVA (*slave only*) or SPA PDU received \receive FCB == await FCB in start segment AND start segment does not contain the complete message ⇒ accept data octets	**11 RStart_Sgm5**	**RECEIVE_FRAME**

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
RECEIVE_FRAME SPA PDU received \receive FCB == await FCB in start segment AND start segment contains a complete message AND memory available ⇒ accept data octets, enter positive confirmation in PDLSDU, change receive FCB, PDL_Data_Ack.indication to the PDL-user	**12 RStart_Sgm6**	**R_IDLE**
RECEIVE_FRAME SPA PDU received \receive FCB == await FCB in start segment AND start segment contains a complete message AND no more memory available ⇒ enter negative confirmation (queue full) in PDLSDU, change receive FCB	**13 RStart_Sgm7**	**R_IDLE**
RECEIVE_FRAME (*slave only*) SVA PDU received \receive FCB == await FCB in start segment AND start segment contains a complete message ⇒ accept data octets, enter positive confirmation in PDLSDU, change receive FCB	**14 RStart_Sgm8**	**R_IDLE**
RECEIVE_FRAME DATA PDU received \last data segment of SAP AND memory available ⇒ accept data octets, enter positive confirmation in PDLSDU, change receive FCB, PDL_Data_Ack.indication to PDL-user	**15 RData_Sgm1**	**R_IDLE**
RECEIVE_FRAME DATA PDU received \last data segment of a SPA PDU AND no more memory available ⇒ enter negative confirmation (queue full) in PDLSDU, change receive FCB	**16 RData_Sgm2**	**R_IDLE**
RECEIVE_FRAME (*slave only*) DATA PDU received \last data segment of SVA ⇒ accept data octets, enter positive confirmation in PDLSDU, change receive FCB	**17 RData_Sgm3**	**R_IDLE**
RECEIVE_FRAME DATA PDU received \not last data segment of a SVA (*slave only)* or SPA PDU ⇒ accept data octets	**18 RData_Sgm4**	**RECEIVE_FRAME**
RECEIVE_FRAME *(slave only)* SWA PDU received ⇒ accept data octets (observe new position in the data flow)	**19 RSWA_Sgm1**	**RECEIVE_FRAME**
RECEIVE_FRAME *(master only)* last data cycle contained errors ⇒ enter RWA PDU in PDLSDU (correct position in the message), C_{SWA} = 0	**20 Cycle_Err**	**WAIT_SWA**
WAIT_SWA *(master only)* no SWA PDU received \C_{SWA} ≤ DIST+add_wait ⇒ increment C_{SWA}	**21 WAIT2**	**WAIT_SWA**
WAIT_SWA *(master only)* no SWA PDU received \C_{SWA} > DIST+add_wait	**22 Time_Out**	**R_IDLE**

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
WAIT_SWA ***(master only)*** SWA PDU received \SWA PDU does not contain the last data ⇒ accept data octets (observe new position in the data flow)	**23 RSWA_Sgm2**	**RECEIVE_FRAME**
WAIT_SWA ***(master only)*** SWA PDU received \SWA PDU contains the last data of a SPA AND memory is available ⇒ accept data octets (observe new position in the data flow), enter positive confirmation in PDLSDU, change receive FCB, PDL_Data_Ack.indication to PDL-user	**24 RSWA_Sgm3**	**R_IDLE**
WAIT_SWA ***(master only)*** SWA PDU received \SWA PDU contains the last data of SPA AND no more memory available ⇒ enter negative confirmation (queue full) in PDLSDU, change receive FCB	**25 RSWA_Sgm4**	**R_IDLE**
WAIT_SWA ***(master only)*** SWA PDU received \SWA PDU contains the last data of SVA ⇒ accept data octets (observe new position in the data flow), enter positive confirmation in PDLSDU, change receive FCB	**26 RSWA_Sgm5**	**R_IDLE**

4.4 Basic Link Layer (BLL)

4.4.1 Functionality of the BLL

The Basic Link Layer is the component of a device that is responsible for the controlled bus access.

In the case of the master it makes the BLL_Data service available at its interface to the PDL. This service allows to run specific data cycles and to exchange data between PDL and BLL. In the slave the BLL ensures that received data is passed to the PDL and that new data are to be sent is accepted from the PDL (see Figure 41).

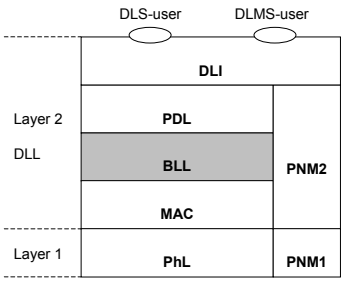


Figure 41 – Location of the BLL in the DLL

The BLL can be parameterized and reset via the interface to the PNM2.

The Basic Link Layer of the master is subdivided into the "BLL operating protocol machine" and "BLL-BAC protocol machine". A slave has only a simplified BLL operating protocol machine.

4.4.2 PDL-BLL interface

4.4.2.1 General

4.4.2 describes the BLL_Data data transmission service, which is available to the PDL, with its service primitives and the associated parameters. The BLL_Data service is mandatory. Figure 42 shows the interface between PDL and BLL in the layer model.

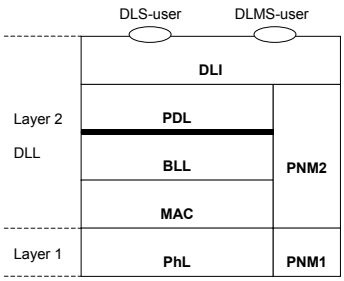


Figure 42 – Interface between PDL and BLL in the layer model

4.4.2.2 Overview of the service and interactions

4.4.2.2.1 BLL_Data

The BLL makes the BLL_Data service available to the PDL. With this service and a PDLSDU the PDL of the master transfers the OUT data within a data cycle to the slaves and receives simultaneous all IN data from the slaves with a PDLSDU. The OUT and IN data are separated with respect to time, that is, the OUT and IN data which are sent or received with a service call, need not belong to one and the same data cycle. Thus, PDL and PhL can operate independently of each other.

The slave behavior is similar to the master:
The BLL of a slave provides the new received OUT data to the PDL by means of indication. The PDL transmits the IN data for sending within next data cycle to the BLL by means of a response. The IN data will be sent in one of the next bus cycles over the physical medium to the master.

The BLL_Data service is provided by using four service primitives. The master uses a request primitive to request a service. A confirmation primitive is returned to the master after the service has been executed. The BLL sends new IN data with the indication primitive to the PDL. The PDL responds to this indication with a response primitive.

Service primitives:

— BLL_Data.request (master side only)

— BLL_Data.confirm (master side only)

— BLL_Data.indication (slave side only)

— BLL_Data.response (slave side only).

4.4.2.3 Overview of the interactions

Figure 43 shows the time relations of the primitives for the BLL_Data service:

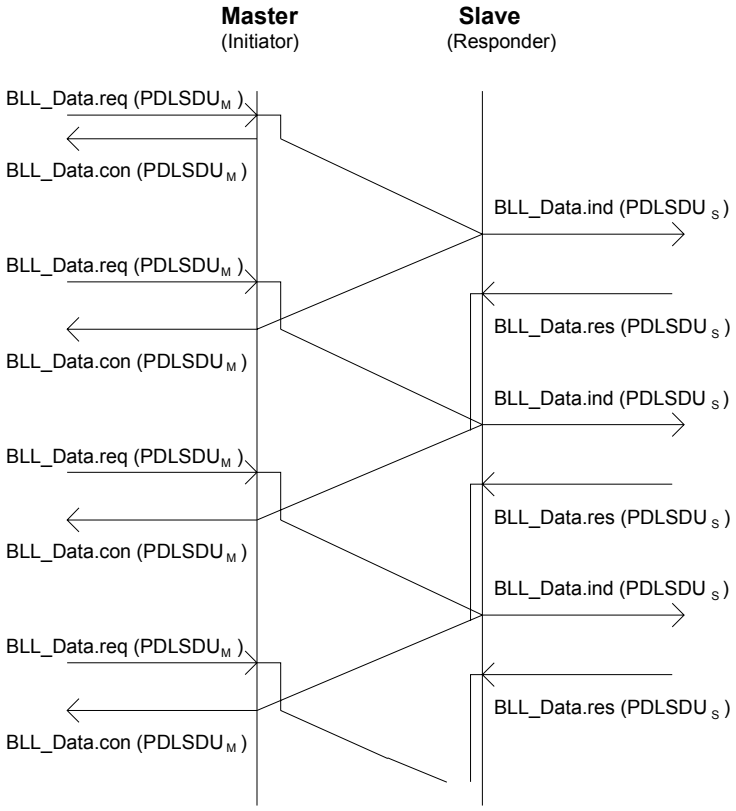


Figure 43 – BLL_Data service

4.4.2.4 Detailed definitions of the services and interactions

4.4.2.4.1 BLL_Data

The BLL_Data service is mandatory.

Using BLL_Data.request (master side only), the PDL of the master shall use this service primitive for sending of a PDLSDU within next data cycle. The PDLSDU shall contain all data which is to be transmitted over the bus in a data cycle. If the BLL of the master received new data, it passes this data as a PDLSDU to the PDL using a BLL_Data.confirm. The update_info parameter contains the information whether the data is valid. The received data is not valid when a data cycle contained errors. The BLL immediately confirms a BLL_Data.request by means of a BLL_Data.confirm with result (-), provided that no valid bus configuration exists or the BLL cannot accept further OUT data owing to a shortage of resources.

The BLL of a slave transfers new received data as a PDLSDU to the PDL using the BLL_Data.indication primitive. The BLL does not get any received data when there are any errors in the data cycles and accordingly does not generate a BLL_Data.indication. Using a BLL_Data.res primitive, the PDL of the slaves sends new transmit data in a PDLSDU to the BLL (see Table 27).

Table 27 – BLL_Data

Parameter name	Request	Indication	Response	Confirm
Argument	M	M		
PDLSDU	M	M		
Result(+)			M	S
PDLSDU			M	M
update_info				M
Result(-)				S
error_code				M

argument:

The argument contains the service-specific parameters of the service call.

PDLSDU:

Request:
The PDLSDU parameter contains the OUT data to all slaves, which is to be transferred in one data cycle. The BLL passes the data on to the subordinate MAC layer.

Indication:
The PDLSDU parameter contains the OUT data which was received in the last data cycle without errors.

result(+):

This parameter indicates that the service was executed successfully.

Confirmation:
This parameter contains the IN data which the master received in the last data cycle.

Response:
The PDLSDU contains the IN data of a slave which is to be transmitted in a data cycle.

update_info:

This parameter describes the validity of the IN data. Possible codes are:

a) OK — the PDLSDU contains valid IN data.

b) NOK — the PDLSDU does not contain any valid IN data.

result(–):

This parameter indicates that the service could not be executed successfully.

error_code:

This parameter indicates the reason why the service could not be executed successfully. Possible error codes are:

STATE_CONFLICT
The PDL sent a BLL_Data.request, although no valid bus configuration exists.

NO_RESRC
The PDL sent a BLL_Data.request, although the BLL is not ready to accept new OUT data.

4.4.3 PNM2-BLL interface

4.4.3.1 General

The management of the BLL is the part of the BLL that provides the management functionality of the BLL requested by the PNM2. The management of the BLL handles the initialization, the monitoring, and the error recovery in the BLL.

4.4.3 defines the administrative BLL management services which are available to the PNM2, together with their service primitives and associated parameters. Figure 44 shows the interface between PNM2 and BLL in the layer model.

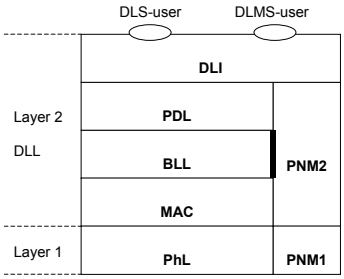


Figure 44 – Interface between PNM2 and BLL in the layer model

The service interface between PNM2 and BLL provides the following functions.

— Reset of the BLL

— Request and change of the current operating parameters of the BLL

— Indication of unexpected events, errors, and status changes which occurred or were detected in the BLL.

4.4.3.2 Overview of the services

4.4.3.2.1 Available services

The BLL makes the following services available to the PNM2.

— BLL_ID (acquire bus configuration)

— Reset BLL

— Set Value BLL

— Get Value BLL

— Event BLL.

The BLL services are described with the primitives (beginning with BLL_…).

4.4.3.2.2 BLL_ID

The BLL (master side only) makes the required BLL_ID service available to the PNM2. With this service and a BLLSDU the PNM2 of the master transfers the control codes for an identification cycle to the slaves and receives all device codes of an identification cycle from the slaves with a BLLSDU.

Service primitives:

— BLL_ID.request (master only)

— BLL_ID.confirm (master only)

4.4.3.2.3 BLL_Reset

The PNM2 uses this required service to reset the BLL. Upon execution of the service the PNM2 receives a confirmation.

Service primitives:

— BLL_Reset.request

— BLL_Reset.confirm

4.4.3.2.4 BLL_Set_Value

The PNM2 uses this optional service to set a new value to the variables of the BLL. Upon completion, the PNM2 receives a confirmation from the BLL whether the defined variables accepted the new value.

Service primitives:

— BLL_Set_Value.request

— BLL_Set_Value.confirm

4.4.3.2.5 BLL_Get_Value

The PNM2 uses this optional service to read the variables of the BLL. The current value of the defined variable is transmitted in the response of the BLL.

Service primitives:

— BLL_Get_Value.request,

— BLL_Get_Value.confirm.

4.4.3.2.6 BLL_Event

The BLL uses this required service to inform the PNM2-user about certain events or errors in the BLL.

Service primitive:

— BLL_Event.indication

4.4.3.3 Overview of the interactions

Figure 45 and Figure 46 show the time relations of the service primitives:

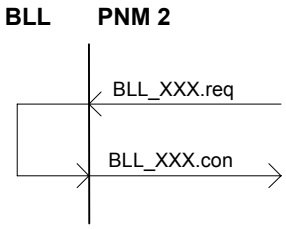


Figure 45 – Reset, Set Value and Get Value BLL services

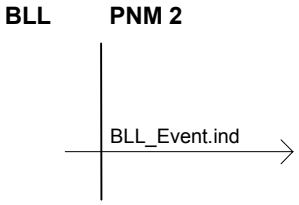


Figure 46 – Event BLL service

4.4.3.4 Detailed definitions of the services and interactions

4.4.3.4.1 BLL_ID (master side only)

Using a BLL_ID.request, PNM2 transfers a BLLSDU to the BLL. The BLL shall initiate an identification cycle when it receives this request. The BLLSDU shall contain all data which is to be transmitted over the bus in an identification cycle. If the BLL of the master received new received data in an identification cycle, it passes this data as a BLLSDU to the PNM2 using a BLL_ID.confirm. The received data is not valid when an identification cycle contained errors (see Table 28).

Table 28 – BLL_Data

Parameter name	Request	Confirm
Argument	M	
SDU	M	
Result(+)		S
SDU		M
Result(-)		S
error_code		M

argument:

The argument contains the service-specific parameters of the service call.

SDU:

Request: The SDU parameter contains the control codes to all slaves, which is to be transferred in one identification cycle. The BLL passes the data on to the subordinate MAC layer.

result(+):

This parameter indicates that the service was executed successfully.

Confirmation: This parameter contains the device codes which the master received in the last identification cycle.

result(-):

This parameter indicates that the service could not be executed successfully.

error_code:

This parameter indicates the reason why the service could not be executed successfully.

4.4.3.4.2 BLL_Reset

The BLL_Reset service is mandatory. The PNM2 transfers a BLL_Reset.request to the BLL to reset it (see Table 29).

Table 29 – BLL_Reset

Parameter name	Request	Confirm
Argument	M	
Result(+)		M

4.4.3.4.3 BLL_Set_Value

The BLL_Set_Value service is mandatory. The PNM2 transfers a BLL_Set_Value.request primitive to the BLL to set a defined BLL variable to a desired value. After receipt of this primitive, the BLL tries to select the variable and to set the new value. Upon completion, the BLL transmits a BLL_Set_Value.confirm primitive to the PNM2 (see Table 30).

Table 30 – BLL_Set_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
desired_value	M	
Result(+)		M

variable_name:

This parameter defines the BLL variable which is set to a new value.

desired_value:

This parameter declares the new value for the BLL variable.

Table 31 provides information on which BLL variables may be set to which new values.

Table 31 – BLL variables

Name of BLL variable	Value range	Default
update_time T_{UP}	$(0..2^{15})$ * 0,1 ms	
bus_timeout T_{TO_BUS}	$(0..2^{15})$ * 1 ms	
Configuration_valid	true, false	false
BLL_access_control	locked, req_to_lock, unlocked	locked

4.4.3.4.4 BLL_Get_Value

The BLL_Get_Value service is optional. The PNM2 transfers a BLL_Get_Value.request primitive to the BLL to read out the current value of a specified BLL variable. After receipt of this primitive, the BLL tries to select the specified variable and transmit its current value to the PNM2 with a BLL_Get_Value.confirm primitive (see Table 32).

Table 32 – BLL_Get_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
Result(+)		M
current_value		M

variable_name:

This parameter specifies the BLL variable the value of which is to be read out.

desired_value:

This parameter contains the read-out value of the BLL variable.

The BLL variables to be read are exactly those variables that can be written to with BLL_Set_Value.

4.4.3.4.5 BLL_Event

The BLL_Event service is mandatory. The BLL transfers a BLL_Event.indication primitive to the PNM2 to inform it about important events or errors in the BLL (see Table 33 and Table 34).

Table 33 – BLL_Event

Parameter name	Indication
Argument	M
event	M

event:

This parameter specifies the event which occurred or the error source in the BLL and can assume the following values:

Table 34 – BLL_Event

Name	Meaning	Mandatory / optional
BLL_bus_timeout	There is a bus timeout t_{TO_BUS}, that is, the time between two data cycles completed without errors was too long. The BLL declared the bus configuration invalid.	O
BLL_update_timeout	The was an update timeout before the next data cycle was started.	O
BLL_cycle_error	Identifies cycles with errors, BLL interrupts data cycles, waits for the enabling of PNM2 by means of the BLL_Set_Value (variable: BLL_access_control = unlocked)	M

4.4.4 Protocol machines of the BLL

4.4.4.1 BLL protocol machines of the master

4.4.4.1.1 Overview

The BLL operating protocol machine of the master receives the OUT data of the higher-level PDL layer and passes it to the BLL-BAC-PM. The BLL-BAC-PM then starts a data cycle which is controlled by a timer. Moreover, after a data cycle the BLL operating protocol machine receives the IN data from the BLL-BAC-PM and passes it to the higher-level PDL. It is capable of initiating the next data cycle while the PDL is still processing the data of the last cycle. Another functionality is the monitoring of the bus timeout t_{O_BUS}. After a data cycle with errors, existing PDLSDUs in the BLL are rejected owing to the method of functioning of the PDL protocol machines.

In the master, the BLL-BAC-PM (BAC: 'Basic Access Control') ensures that the update_time t_{UP} is kept for data cycles. For this, it passes, initiated by a timer, BAC_Cycle.requests from the BLL operating protocol machine to the MAC layer as MAC_Cycle.requests. Furthermore, the BLL_Access_Control variable can be used to interrupt the starting of bus cycles by the PNM2 in order to start ID cycles there. As the diagnostic application will possibly run an identification cycle after a data cycle with errors, the BLL-BAC-PM sets the BLL_Access_Control variable after a data cycle error automatically to locked. Upon completion of the ID cycle, the PNM2 shall again enable the starting of the data cycles with a BLL_Set_Value service.

4.4.4.1.2 BLL-internal functions

The BLL-MAC-PM makes the BAC_Cycle service available to the BLL operating protocol machine of the master. In the BLL-MAC-PM the service is mapped onto the MAC_Cycle service of the MAC sublayer. The structure of the BAC_Cycle service exactly corresponds to that of the MAC_Cycle service.

The difference between the two services is that with the BAC_Cycle a bus cycle is not immediately started after the service call, but the cycle start is synchronized with the clock of a timer. The PNM2 can also prevent the start of the cycle with the BLL_Set_Value service (BLL_access_control variable).

4.4.4.1.3 BAC_Reset

After a BLL_Reset, the operating protocol machine resets the BAC-PM with the BAC_Reset service. The service is immediately confirmed.

4.4.4.1.4 BLL operating protocol machine

Figure 47 shows the BLL operating protocol machine of a master.

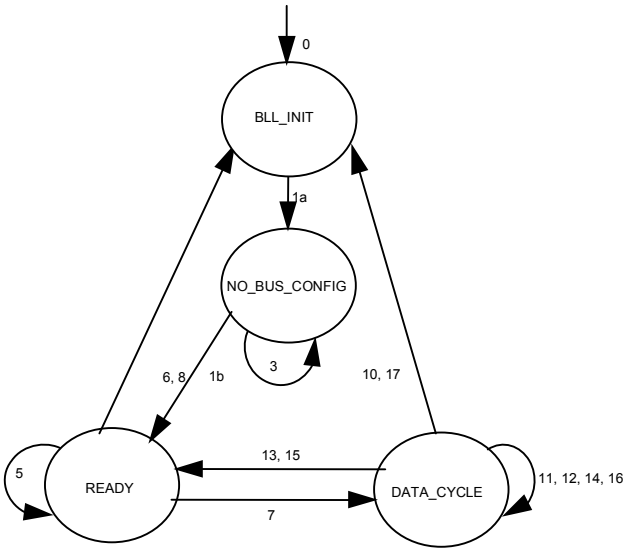


Figure 47 – BLL operating protocol machine of the master

Transitions as a result of BLL_Reset primitive always go from every state to the NO_BUS_CONFIG state. These transitions are not specified individually but are described with the transition 0.

States of the BLL operating protocol machine of the master

4.4.4.1.4.1 BLL_INIT

The Basic Link Layer, including the BLL_access_control and configuration_valid variables, is initialized and/or reset.

4.4.4.1.4.2 NO_BUS_CONFIG

There is no valid active bus configuration. No data cycles can be run.

4.4.4.1.4.3 READY

There is no valid active bus configuration. A data cycle can be initiated with a BLL_Data.request.

4.4.4.1.4.4 DATA_CYCLE

A data cycle was initiated with a BLL_Data.request.

Table 35 describes the state transitions.

The following events may occur in each state and shall be taken into consideration.

— Change of the value of the configuration_valid variable (The PNM2 can change this value with BLL_Set_Value).

— BLL_Data.request.

— BAC_Cycle.confirm (if a BAC_Cycle.request was sent before).

— Time t_1 expired, that is, the bus timeout t_{TO_BUS} is exceeded.

— BLL_Reset.request.

Receive- and transmit BLLSDU:

In the BLL operating protocol machine a distinction is made between receive and transmit BLLSDU. The receive BLLSDU (BLL_RSDU) contains the data which was received by the MAC sublayer after a data cycle, while the transmit BLLSDU (BLL_TSDU) contains all data which was sent to the MAC sublayer prior to a data cycle. As BLL_RSDU is not necessarily passed on to the PDL immediately after the cycle end, the BLL_RSDU is buffered together with a $SDU\_status_{BLL\_RSDU}$ in the BLL:

$SDU\_status_{BLL\_RSDU}$

a) OK — there is valid input data since a data cycle was completed without errors.

b) NOK — there is no valid input data since a data cycle has not yet been run or because the last data cycle contained errors.

Table 35 – State transitions of the BLL operating protocol machine of the master

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
After power on	**0**	**BLL_INIT**
BLL_INIT ⇒ Initialize operating PM, reject BLL_TSDUs and BLL_RSDUs which have not yet been processed	**1a**	**NO_BUS_CONFIG**
NO_BUS_CONFIG configuration_valid == true (edge) ⇒ generate BLL_RSDU with $SDU\_status_{BLL\_RSDU}$ = NOK	**1b**	**READY**
NO_BUS_CONFIG BLL_Data.request ⇒ BLL_Data.confirm (-) with error_code = STATE_CONFLICT	**3**	**NO_BUS_CONFIG**
READY configuration_valid == false (edge)	**6**	**BLL_INIT**
READY BLL_Data.request ⇒ accept PDLSDU as BLL_TSDU , BLL_Data.confirm(+) with PDLSDU = BLL_RSDU and update_info = $SDU\_status_{BLL\_RSDU}$, BAC_Cycle.request with BLLSDU = BLL_TSDU	**7**	**DATA_CYCLE**
READY timer T_1 expired ⇒ BLL_Event.indication with event = BLL_bus_timeout	**8**	**BLL_INIT**
DATA_CYCLE configuration_valid == false (edge)	**10**	**BLL_INIT**
DATA_CYCLE BLL_Data.request \still resources available ⇒ accept PDLSDU as BLL_TSDU	**11**	**DATA_CYCLE**
DATA_CYCLE BLL_Data.request \no more resources available ⇒ BLL_Data.confirm (-) with error_code = NO_RESRC	**12**	**DATA_CYCLE**

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
DATA_CYCLE BAC_Cycle.confirm with BLLSDU and result == OK \no further BLL_TSDU available ⇒ set timer T_1 to value T_{TO_BUS}, buffer BLLSDU as BLL_RSDU with $SDU\_status_{BLL\_RSDU}$ = OK	**13**	**READY**
DATA_CYCLE BAC_Cycle.confirm with BLLSDU and result == OK \further BLL_TSDU available ⇒ set timer T_1 to the value T_{TO_BUS}, BLL_Data.confirm(+) with PDLSDU = BLLSDU and update_info = OK, BAC_Cycle.request with BLLSDU = BLL_TSDU	**14**	**DATA_CYCLE**
DATA_CYCLE configuration_valid == false (edge)	**15**	**READY**
DATA_CYCLE BLL_Data.request \still resources available ⇒ accept PDLSDU as BLL_TSDU	**16**	**DATA_CYCLE**
DATA_CYCLE BLL_Data.request \no more resources available ⇒ BLL_Data.confirm (-) with error_code = NO_RESRC	**17**	**BLL_INIT**
DATA_CYCLE BAC_Cycle.confirm with BLLSDU and result == OK \no further BLL_TSDU available ⇒ set timer T_1 to value T_{TO_BUS}, buffer BLLSDU as BLL_RSDU with $SDU\_status_{BLL\_RSDU}$ = OK		**same_state**
DATA_CYCLE BAC_Cycle.confirm with BLLSDU and result == OK \further BLL_TSDU available ⇒ set timer T_1 to the value T_{TO_BUS}, BLL_Data.confirm(+) with PDLSDU = BLLSDU and update_info = OK, BAC_Cycle.request with BLLSDU = BLL_TSDU		**BLL_INIT**

4.4.4.1.5 BLL-BAC protocol machine

Figure 48 shows the BLL-BAC protocol machine.

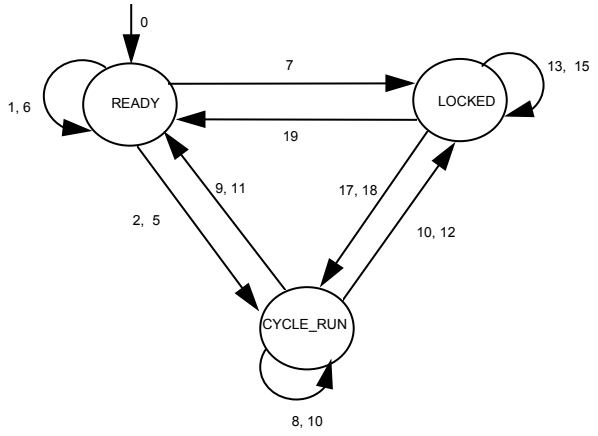


Figure 48 – BLL-BAC protocol machine

Transitions as a result of a BLL_Reset.request primitive always go from every state to the READY state. These transitions are not shown individually but are described by the transition 0.

States of the BLL-BAC-PM (protocol machine)

4.4.4.1.5.1 READY

The initiating of bus cycles is enabled. No cycle is run because the BLL operating protocol machine and diagnostic application did not send a BAC_Cycle.request or because the update time t_{UP} has not yet expired. t_{UP} is considered for data cycles only.

4.4.4.1.5.2 CYCLE_RUN

The initiating of bus cycles is enabled. A bus cycle is running. In this state the protocol machine waits for the end of the bus cycle.

4.4.4.1.5.3 LOCKED

The initiating of bus cycles is locked. However, a running bus cycle can still be completed. If there is a BAC_Cycle.request in this state, the BLLSDU transferred with the request is buffered. Only after the enabling of the bus cycles will a new bus cycle be started. Only one BLLSDU can be buffered.

Table 36 describes the state transitions.

The following events may occur and shall be taken into consideration.

— BAC_Cycle.request.
— MAC_Cycle.confirm (if a MAC_Cycle.request was sent before).
— Timer T_2, that is, the update_time T_{UP} expired.
— Change of the BLL_access_control variable.
— BAC_Reset.request.

Table 36 – State transitions of the BLL-BAC protocol machine

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
After power on ⇒ reset BAC-PM	**0**	**READY**
READY BAC_Cycle.request (data_cycle, BLLSDU) \T_2 started and not yet expired ⇒ retain BLLSDU	**1**	**READY**
READY BAC_Cycle.request (data_cycle, BLLSDU) \T_2 expired or not started ⇒ MACSDU = BLLSDU, MAC_Cycle.request (data_cycle, MACSDU), set T_2 to T_{UP} , start T_2	**2**	**CYCLE_RUN**
READY T_2 expired \BAC_Cycle request already received but not yet executed ⇒ MACSDU = BLLSDU, MAC_Cycle.request (data_cycle, MACSDU), set T_2 to T_{UP}, start T_2	**5**	**CYCLE_RUN**

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
READY T_2 expired \no pending BAC_Cycle service available ⇒ BLL_Event.indication with event = BLL_update_timeout	**6**	**READY**
READY BLL_access_control == req_to_lock (edge)	**7**	**LOCKED**
CYCLE_RUN BAC_Cycle.request ⇒ BAC_Cycle.confirm with result = NO	**8**	**CYCLE_RUN**
CYCLE_RUN MAC_Cycle.confirm with result == OK ⇒ BAC_Cycle.confirm with result = OK to BLL operating protocol machine	**9**	**READY**
CYCLE_RUN MAC_Cycle.confirm with result == NO ⇒ BAC_Cycle.confirm with result = NO to BLL operating protocol machine, BLL_access_control == locked, BLL_Event: BLL_cycle_error	**10**	**LOCKED**
CYCLE_RUN T_2 expired ⇒ BLL_Event.indication with event = BLL_update_timeout	**11**	**CYCLE_RUN**
CYCLE_RUN BLL_access_control == req_to_lock (edge)	**12**	**LOCKED**
LOCKED BAC_Cycle.request (data_cycle, BLLSDU) \no pending BAC_Cycle service available ⇒ retain BLLSDU	**13**	**LOCKED**
LOCKED MAC_Cycle.confirm with result == OK/NO ⇒ BAC_Cycle.confirm with result == OK/NO to BLL operating protocol machine, BLL_access_control = locked, Result == NO → BLL_Event.indication with event = BLL_cycle_error	**15**	**LOCKED**
LOCKED BLL_access_control == unlocked (edge) \bus cycle not yet been completed	**17**	**CYCLE_RUN**
LOCKED BLL_access_control == unlocked (edge) \bus cycle completed, but BAC_Cycle service is still pending ⇒ MACSDU = BLLSDU, MAC_Cycle.request (data_cycle, MACSDU)	**18**	**CYCLE_RUN**
LOCKED BLL_access_control == unlocked (edge) \bus cycle completed AND no BAC_Cycle service is pending	**19**	**READY**
any_state BAC_Reset.request ⇒ reset BAC-PM, BAC_Reset.con		**READY**

4.4.4.2 BLL protocol machine of the slave

4.4.4.2.1 Overview

The BLL-BAC protocol machine does not exist in the slave, since only the master can initiate bus cycles. The BLL operating protocol machine of the slave is greatly simplified. It only passes the data from the MAC to the PDL and vice versa.

4.4.4.2.2 BLL operating protocol machine

Figure 49 shows the BLL operating protocol machine of a slave.

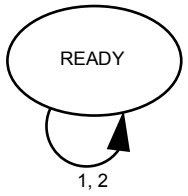


Figure 49 – BLL operating protocol machine of the slave

States of the BLL operating protocol machine of the slave

4.4.4.2.2.1 READY

The BLL operating protocol machine is ready to accept new output data in form of a MACSDU from the MAC and passes it to the PDL as a BLLSDU. IN data is also passed from PDL (BLLSDU) to the MAC (MACSDU). The BLL is only responsible for the transmission of data by means of data cycles.

Table 37 describes the state transitions.

The following events may occur in every state and shall be taken into consideration:

— BLL_Data.res

— MAC_Cycle_ind

The MAC_Cycle_ind is a combination of the following interactions at the MAC ↔ MAC-user interface.

a) MAC_Data.indication (Data_Cycle).

b) MAC_Get_Data.request (Data_Receive).

c) MAC_Get_Data.confirm (Data_Receive, OK, MACSDU).

Table 37 – State transitions of the BLL operating protocol machine of the slave

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
READY MAC_Cycle_ind with MACSDU ⇒ BLLSDU = MACSDU, BLL_Data.indication (BLLSDU)	**1**	**READY**
READY BLL_Data.res (BLLSDU) ⇒ MACSDU = BLLSDU, MAC_Cycle_res (MACSDU)	**2**	**READY**

MAC_Cycle_res is a combination of the following interactions at the MAC-MAC-user interface:

1) MAC_Put_Data.request (Data_Transmit, MACSDU)
2) MAC_Put_Data.confirm (Data_Transmit, OK).

4.5 Medium Access Control (MAC)

4.5.1 Location of the MAC in the DLL

The Medium Access Control (MAC) is the lowest sublayer of the Data Link Layer (DLL) and is based on the PhL (PhL) (see Figure 50).

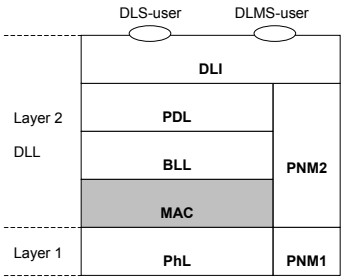


Figure 50 – Location of the MAC in the DLL

As shown in Figure 51, the PhL is also subdivided into several sublayers, the functionality of which follows from Figure 51 as well. The sublayer is called MAC-user and corresponds to the BLL sublayer.

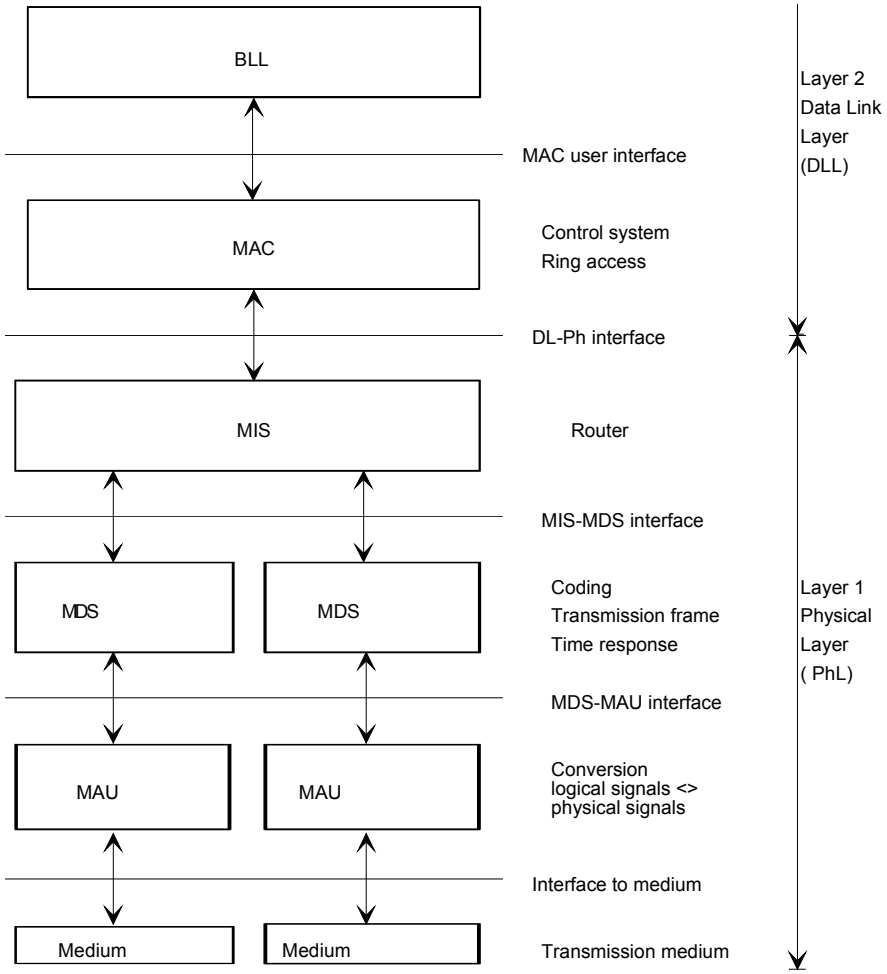


Figure 51 – Model details of layers 1 and 2

4.5.2 Functionality of the MAC

The MAC sublayer controls the access to the transmission medium and ensures that the received and transmitted user data is checked in the form of a 16-bit CRC polynomial.

The MAC sublayer transmits the data, which are received the MAC-user, in one DLPDU cycle as a sequence of binary data units via the DL-Ph interface to the PhL. At the same time the MAC sublayer receives data units via the DL-Ph interface from the PhL. If the PhL has received these data units without errors it makes them available to the MAC-user. It is always the master that initiates a DLPDU cycle, so that a difference is to be made between the active MAC sublayer of a master and the passive MAC sublayer of the slave.

A DLPDU cycle consists of a data sequence which may follow a check sequence (see Figure 52 and Figure 53).

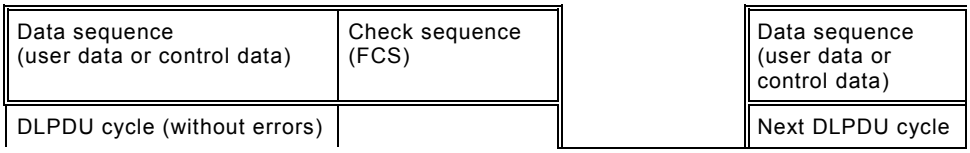


Figure 52 – DLPDU cycle of a data sequence without errors

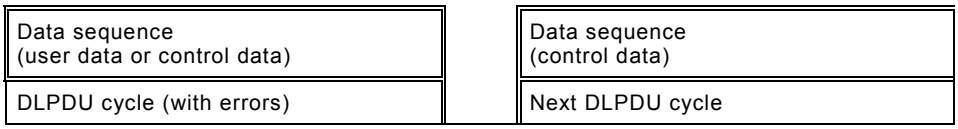


Figure 53 – DLPDU cycle of a data sequence with errors

A data sequence may either include a data cycle for the transmission of user data from the process data channel and, if applicable, from the parameter channel, or an identification cycle for the transmission of data for configuration and error diagnostics. If the data sequence was transmitted without any errors, a check sequence follows which is initiated by the master. This check sequence first transmits the CRC polynomial (checksum) of the data transmitted in the data sequence before, followed by the receive status (checksum status). Should there be an error in the data sequence, and this sequence is followed by another data sequence, the check sequence is omitted and another data sequence is sent, which shall be part of an identification cycle.

The Medium Access Control (MAC) sublayer is a part of the DLL and controls the secured data transmission between the devices over the transmission medium. It transmits the MACSDU from the MAC-user, generates the accordingly DLPDU and transmits it via the DL-Ph interface to the PhL. Conversely, it receives a DLPDU via the DL-Ph interface, and generates the MACSDU from it and transmits it to the MAC-user.

To secure the data transmission, the MAC sublayer generates the checksum of the DLPDU to be transmitted in the form of a CRC polynomial and transmits this sum via the DL-Ph interface to the PhL. Conversely, the MAC sublayer generates the checksum of a received DLPDU in the form of a CRC polynomial and compares it with the received checksum.

4.5.3 Master

4.5.3.1 DLPDU Structure

A distinction is made between the following DLPDU formats: **data sequence DLPDU** and **check sequence DLPDU**.

4.5.3.1.1 Data Sequence DLPDU

The MAC sublayer of a master shall generate the data sequence DLPDU according to Figure 54 by adding the loopback word (LBW) to the MACSDU. The DLPDU thus generated is transmitted from left to right to the PhL in the form of PhIDUs so that the LBW is transmitted first, followed by the MACSDU.

When the number of the data bits transmitted by the BLL cannot be divided by eighth without a remainder, the loopback word (LBW) shall be preceded by extra bits of any contents. The number of these extra bits is 8 bits minus the number of remainder bits.

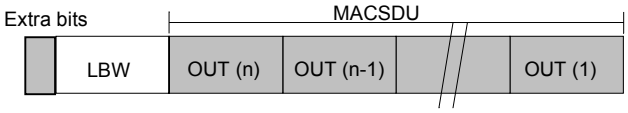


Figure 54 – Data sequence DLPDU transmitted by the master

Conversely, the MAC sublayer of a master shall remove the LBW from a data sequence DLPDU received according to Figure 55 and compare it with the LBW and the extra bits of the last data sequence DLPDU transmitted to the PhL. If both words are identical, the MAC sublayer shall transmit the received MACSDU via the MAC-user interface to the MAC-user. The data sequence DLPDU is received from left to right, starting with the MACSDU.

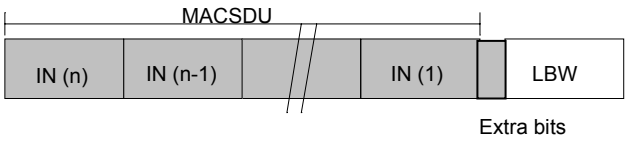


Figure 55 – Data sequence DLPDU received by the master

4.5.3.1.2 Check sequence DLPDU

For a secured transmission of the data sequence DLPDU, the MAC sublayer shall generate a check sequence DLPDU according to Figure 56 after the data sequence DLPDU was transmitted successfully. For this, the MAC sublayer generates a checksum for the transmitted data sequence DLPDU and transmits it together with the checksum status in a check sequence as a check sequence DLPDU via the DL-Ph interface to the PhL.

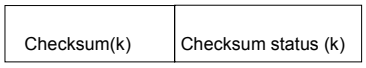


Figure 56 – Check sequence DLPDU

Conversely, the MAC sublayer of the master compare the checksum of a received check sequence DLPDU according to Figure 56 with the checksum calculated for the data sequence DLPDU that was received immediately before. Then it has to evaluate the checksum status of the received check sequence DLPDU. The result is communicated to the MAC-user.

4.5.3.2 Loopback word (LBW)

To secure the data transmission, the MAC shall generate a loopback word (LBW) with a structure as shown in Figure 57.

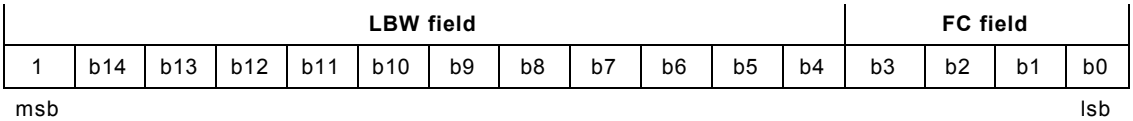


Figure 57 – Loopback word (LBW)

The loopback word is transmitted from right to left, starting with the least significant bit (lsb), and ending with the most significant bit (msb).

In every transmitted LBW the binary value of b3…b0 are used as frame counter (FC), with b3 as the msb and b0 as the lsb, is decremented by the value 1 compared with the value represented by b3…b0 of the last LBW, without considering a possible carry. The MAC-user

defines the values for b14…b4 via the management interface by means of the loopback word variable of the MAC.

4.5.3.3 Checksum

To secure the transmission of a data sequence DLPDU against errors, the MAC sublayer of a master generates an independent checksum in the form of a 16-bit FCS value of the data sequence DLPDU to be transmitted as well as of a received data sequence DLPDU. The FCS value is the division remainder resulting from a continuous division of the DLPDU, starting with the lsb, by the standard CCITT polynomial $X^{16} + X^{12} + X^5 + 1$ with pre- and post-division adjustment, all as specified in 4.5.3.3.1.

The checksum is transmitted in a check sequence DLPDU, beginning with the lsb and ending with the msb.

NOTE 2 The checksum may be generated synchronously with the transmission or receipt of the data sequence DLPDU.

NOTE 3 If a transmission error is detected when a data sequence DLPDU is received, the checksum for the received data sequence DLPDU is set to its initial value, L(X), as specified in 4.5.3.3.1.

4.5.3.3.1 Frame check sequence field

Within this subclause, any reference to bit *K* of an octet is a reference to the bit whose weight in a one-octet unsigned integer is 2^K.

NOTE This is sometimes referred to as “little endian” bit numbering.

Table 38 – FCS length and polynomial

Item	Value
n-k	16
G(X)	$X^{16} + X^{12} + X^5 + 1$ (notes 1, 2, 3)

NOTE 1 Code words D(X) constructed from this G(X) polynomial have Hamming distance 4 for lengths ≤ 4095 octets, and all errors of odd weight are detected.

NOTE 2 This G(X) polynomial is relatively prime to all, and is thus not compromised by any, of the primitive scrambling polynomials of the form $1 + X^{-j} + X^{-k}$ sometimes used in DCEs (modems). However, it is severely compromised by use of differential coding, which uses an encoding polynomial of $1 + X^{-1}$ (a factor of G(X)), and therefore it should be used with PhLs which do not employ differential coding.

NOTE 3 This is the same polynomial as specified in ISO/IEC 3309 (HDLC). However, the method of checking differs. As a consequence, the error detection properties implied by the Hamming distance apply only approximately to Type 8’s use.

For this standard, as in other International Standards (for example, ISO/IEC 3309, ISO/IEC 8802 and ISO/IEC 9314-2), DLPDU-level error detection is provided by calculating and appending a 16-bit frame check sequence (FCS) to the other DLPDU fields during transmission to form a "systematic code word"[1)] of length ***n*** consisting of ***k*** DLPDU message bits followed by ***n - k*** (equal to 16) redundant bits, and by checking on reception that the FCS field of the prior data sequence DLPDU is equal to that of the just-received check sequence DLPDU. The mechanism for this computation is as follows:

The generic form of the generator polynomial for this FCS construction is specified in equation (4). The specific polynomial for this DL-protocol is specified in Table 38.

The original message (that is, the DLPDU without an FCS), the FCS, and the composite message code word (the concatenated DLPDU and FCS) shall be regarded as vectors M(X),

1) W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes* (2nd edition), MIT Press, Cambridge, 1972.

F(X), and D(X), of dimension *k*, *n* - *k*, and *n*, respectively, in an extension field over GF(2). If the message bits are $m_1 \ldots m_k$ and the FCS bits are $f_{n-k-1} \ldots f_0$, where

$m_1 \ldots m_8$	form the first octet sent,
$m_{8N-7} \ldots m_{8N}$	form the *N*th octet sent,
$f_7 \ldots f_0$	form the last octet sent, and
m_1	is sent by the first PhL symbol(s) of the message and f_0 is sent by the last PhL symbol(s) of the message (not counting PhL framing information).

NOTE This "as transmitted" ordering is critical to the error detection properties of the FCS.

then the message vector M(X) shall be regarded to be

$$M(X) = m_1X^{k-1} + m_2X^{k-2} + \ldots + m_{k-1}X^1 + m_k \quad (1)$$

and the FCS vector F(X) shall be regarded to be

$$F(X) = f_{n-k-1}X^{n-k-1} + \ldots + f_0 \quad (2)$$
$$= f_{15}X^{15} + \ldots + f_0$$

The composite vector D(X), for the complete DLPDU, shall be constructed as the concatenation of the message and FCS vectors

$$D(X) = M(X)\,X^{n-k} + F(X) \quad (3)$$
$$= m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{n-k} + f_{n-k-1}X^{n-k-1} + \ldots + f_0$$
$$= m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{16} + f_{15}X^{15} + \ldots + f_0 \quad \text{(for the case of k = 15)}$$

The DLPDU presented to the PhL shall consist of an octet sequence in the specified order.

The redundant check bits $f_{n-k-1} \ldots f_0$ of the FCS shall be the coefficients of the remainder F(X), after division by G(X), of $L(X)\,(X^k + 1) + M(X)\,X^{n-k}$

where G(X) is the degree ***n-k*** generator polynomial for the code words

$$G(X) = X^{n-k} + g_{n-k-1}X^{n-k-1} + \ldots + 1 \quad (4)$$

and L(X) is the maximal weight (all ones) polynomial of degree ***n-k-1***

$$L(X) = (X^{n-k} + 1) / (X+1) = X^{n-k-1} + X^{n-k-2} + \ldots + X + 1 \quad (5)$$
$$= X^{15} + X^{14} + X^{13} + X^{12} + \ldots + X^2 + X + 1 \quad \text{(for the case of k = 15)}$$

That is,

$$F(X) = L(X)\,(X^k + 1) + M(X)\,X^{n-k} \ \text{(modulo } G(X)) \quad (6)$$

NOTE 1 The L(X) terms are included in the computation to detect initial or terminal message truncation or extension by adding a length-dependent factor to the FCS.

NOTE 2 As a typical implementation when n-k = 16, the initial remainder of the division is preset to all ones. The transmitted message bit stream is multiplied by X^{n-k} and divided (modulo 2) by the generator polynomial G(X), specified in equation (4). The ones complement of the resulting remainder is transmitted as the (n-k)-bit FCS, with the coefficient of X^{n-k-1} transmitted first.

4.5.3.4 Checksum status

The MAC sublayer of a master generates a checksum status as shown in Figure 58.

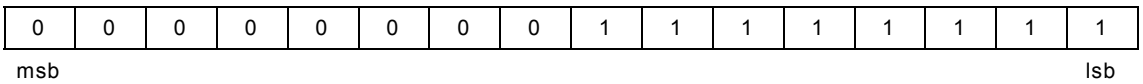


Figure 58 – Checksum status generated by the master

In a check sequence DLPDU transmission takes place immediately after the transmission of the checksum, starting with the lsb and ending with the msb.

In the same way the MAC sublayer of a master shall evaluate a checksum status received in a check sequence DLPDU as shown in Figure 59.

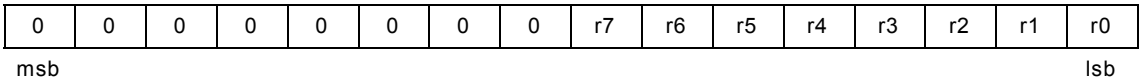


Figure 59 – Checksum status received by the master

For the logical binary values r7…r0 means: r7=r6=r5=r4=r3∧r2∧r1∧r0, where "∧" represents the bit-by-bit AND operator.

NOTE 1 If the expression r3∧r2∧r1∧r0 assumes the value logical 1, both the data sequence DLPDUs and the checksum between two devices was transmitted without errors.

NOTE 2 If the expression r3∧r2∧r1∧r0 assumes the value 0, then a transmission error occurred either when the data sequence DLPDUs were transmitted or the checksums between two devices.

The checksum status is received from right to left in a check sequence DLPDU immediately after the checksum. The lsb is transmitted first and the msb last.

4.5.3.5 Bus access control

The data transmission is described with separate protocol machines for the send and receive part.

4.5.3.5.1 Sender

Figure 60 shows the state transitions of the MAC sublayer for the transmission of a message in a data sequence (identification cycle or data cycle) from the master to the passive slaves as well as the mechanisms for the data transmission security (check sequence).

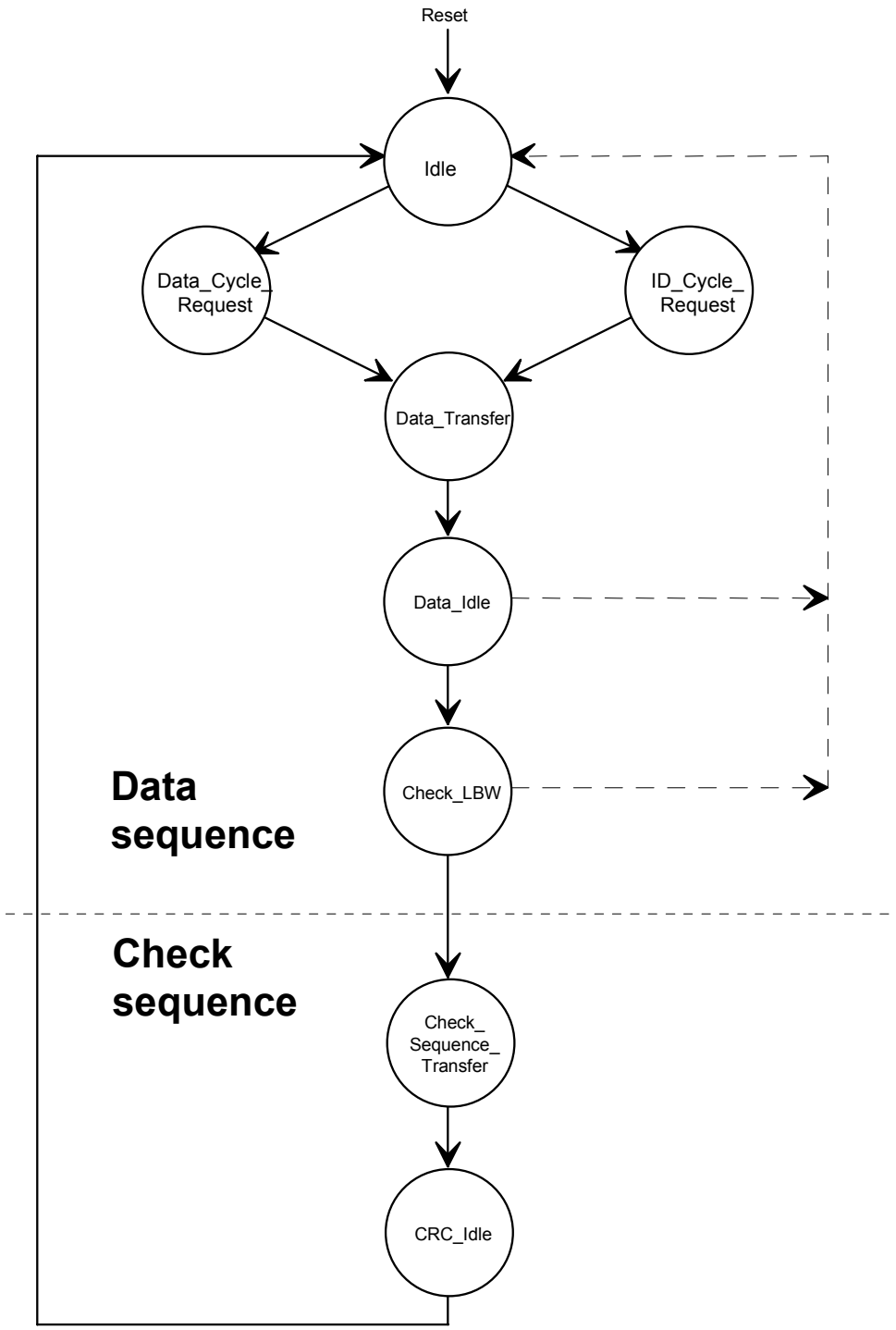


Figure 60 – MAC protocol machine of a master: transmission of a message

4.5.3.5.1.1 Data Sequence

4.5.3.5.1.1.1 Idle

In this state the MAC sublayer of a master shall wait for the request to start a DLPDU cycle through the MAC-user by means of a MAC_Data.request primitive.

If the MAC sublayer receives the request to start an identification cycle (cycle=ID_cycle) from the MAC-user by means of a MAC_Data.request primitive, it shall generate the MACSDU to be transmitted from the data which is available in the ID_transmit buffer for the transmission and assume the ID_cycle_request state. While data is accepted, any other access to the ID_transmit buffer is locked.

If the MAC sublayer receives the request to start a data cycle (cycle=data_cycle) from the MAC-user by means of a MAC_Data.request primitive, it shall generate the MACSDU to be transmitted from the data which is available in the data_transmit buffer for the transmission and assume the Data_Cycle_Request state. While data is accepted, any other access to the data_transmit buffer is locked.

4.5.3.5.1.1.2 ID_Cycle_Request

In this state the MAC sublayer shall first generate the data sequence DLPDU to be transmitted and then start an identification cycle via the DL-Ph interface by means of a Ph-DATA.request primitive (PhICI=start_ID_cycle). After the confirmation through the PhL by means of a Ph-DATA.confirm primitive, the MAC sublayer shall assume the Data_Transfer state.

4.5.3.5.1.1.3 Data_Cycle_Request

In this state the MAC sublayer shall first generate the data sequence DLPDU to be transmitted and then start a data cycle via the DL-Ph interface by means of a Ph-DATA.request primitive (PhICI=start_data_cycle). After the confirmation through the PhL by means of a Ph-DATA.confirm primitive, the MAC sublayer shall assume the Data_Transfer state.

4.5.3.5.1.1.4 Data_Transfer

In this state the MAC sublayer shall first transmit sequentially the data sequence DLPDU by means of the Ph-DATA.request primitives (PhICI=user_data) via the DL-Ph interface to the PhL. The MAC sublayer receives a confirmation for every Ph-DATA.request primitive through the PhL by means of a Ph-DATA.confirm primitive. At the same time the check sequence DLPDU is generated synchronously.

After a complete transmission of the data sequence DLPDU, the MAC sublayer shall assume the Data_Idle state.

4.5.3.5.1.1.5 Data_Idle

In this state the MAC sublayer requests a Ph-DATA.request (PhICI=User_Data_Idle) and waits until the data sequence DLPDU has been completely received. If the receive time monitoring circuit responded, since an identification or data cycle has been started and before the data sequence DLPDU has been completely received, the MAC sublayer shall terminate the data sequence with a corresponding MAC_Data.confirm primitive to the MAC-user and assume the Idle state.

After the data sequence DLPDU has been completely received, the MAC sublayer shall assume the Check_LBW state.

NOTE If a transmission error was detected during the data sequence, the MAC-user started with an identification cycle in the next DLPDU cycle.

4.5.3.5.1.1.6 Check_LBW

In this state, the MAC sublayer first checks whether a transmission error has been detected between the start of an identification or data cycle and the complete receipt of the data

sequence DLPDU. If this is the case, it shall end the data sequence with a corresponding MAC_Data.confirm primitive to the MAC-user and assume the Idle state.

If no receive error was detected, the received loopback word is compared with the loopback word transmitted from the MAC sublayer. If both words are identical, the MAC sublayer terminates the data sequence, generates the check sequence DLPDU and assumes the Check_Sequence_Transfer state.

If the received loopback word is not identical with the sent word, a transmission error occurred and the MAC sublayer shall terminate the data sequence with a corresponding MAC_Data.confirm primitive to the MAC-user and assume the Idle state.

4.5.3.5.1.2 Check sequence

4.5.3.5.1.2.1 Check_Sequence_Transfer

In this state, the MAC sublayer shall transmit the check sequence DLPDU sequentially via the DL-Ph interface to the PhL. The transmission starts with the checksum, which is transmitted by means of the Ph-DATA.request primitives (PhICI=CRC_data), followed by the checksum status which is transmitted by means of the Ph-DATA.request primitives (PhICI=CRC_status).

After the check sequence DLPDU has been completely transmitted the MAC sublayer shall assume the CRC_Idle state.

4.5.3.5.1.2.2 CRC_Idle

In this state the MAC sublayer requests a Ph-DATA.request (PhICI=CRC_Data_Idle) and waits until the check sequence DLPDU has been completely received. If the receive time monitoring circuit responded, since the transmission of the check sequence DLPDU started and before the complete receipt of the check sequence DLPDU, the MAC sublayer shall terminate the check sequence with a corresponding MAC_Data.confirm primitive to the MAC-user and assume the Idle state.

After the check sequence DLPDU has been completely received, the MAC sublayer checks whether a transmission error was detected in the check sequence. If no transmission error was detected, the MAC sublayer makes the received MACSDU available to the MAC-user in the ID_receive buffer, when the MACSDU was received in an identification cycle, or in the data_receive buffer, when the MACSDU was received in a data cycle. Afterwards, it completes the DLPDU cycle with a MAC_Data.confirm primitive (status=OK) to the MAC-user and assumes the Idle state. During the data transfer any other access to the corresponding buffer is locked.

If the check sequence detected a transmission error, the MAC sublayer completes the DLPDU cycle with a corresponding MAC_Data.confirm primitive to the MAC-user and assumes the Idle state.

NOTE If a transmission error was detected during the check sequence, the MAC-user started with an identification cycle in the next DLPDU cycle.

4.5.3.5.2 Receiver

The receive part of the MAC sublayer is described with two protocol machines: one protocol machine describes the receipt of a message, consisting of a data sequence and a check sequence, and the second protocol machine describes the identification of a data sequence as an identification or data cycle.

4.5.3.5.2.1 Message receiver

Figure 61 shows the state transitions in the MAC sublayer when a message is received in a data sequence (identification cycle or data cycle) by the master and the mechanisms for checking a secured data transmission (check sequence).

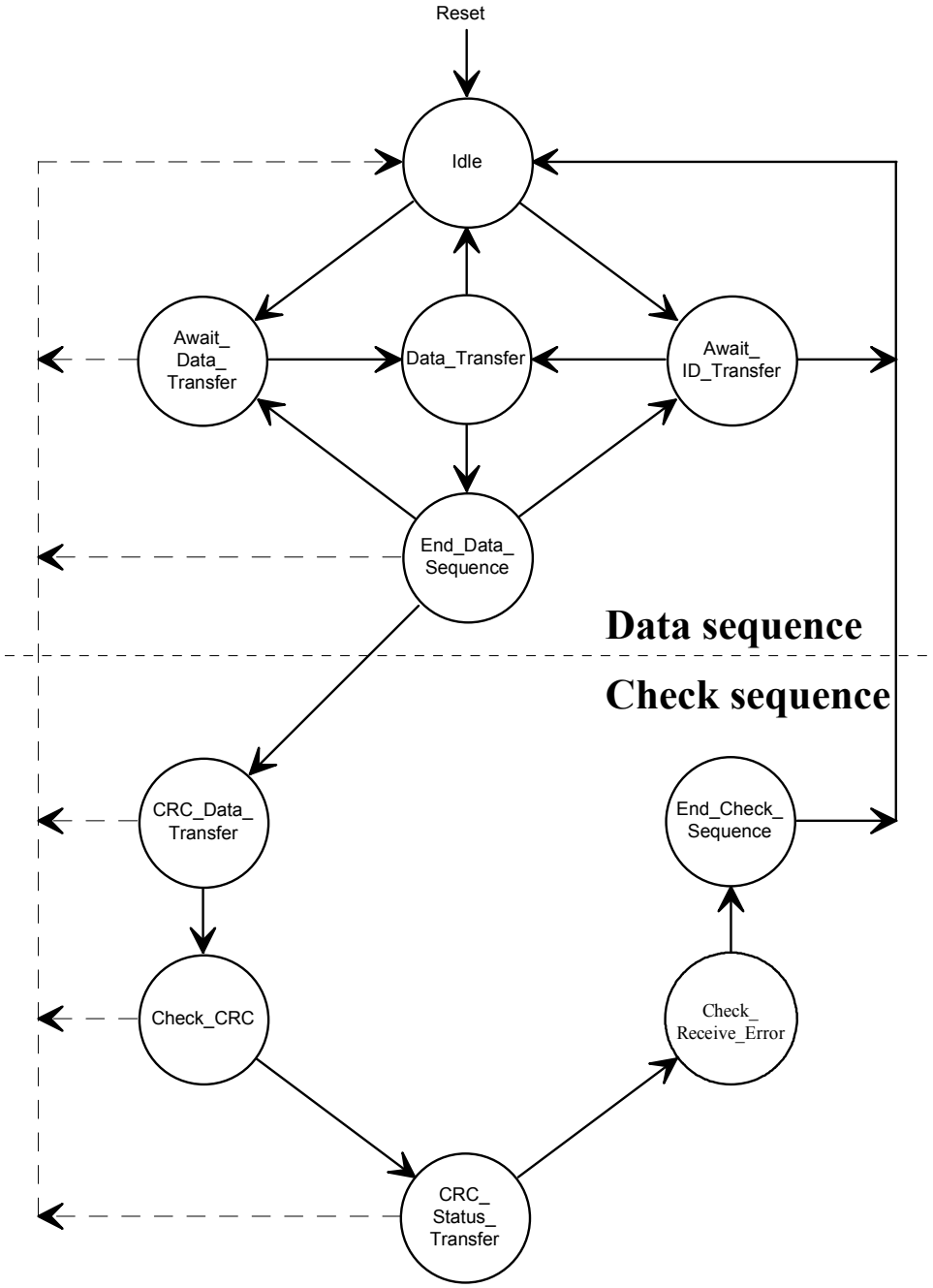


Figure 61 – MAC protocol machine of a master: receipt of a message

4.5.3.5.2.1.1 Data sequence

4.5.3.5.2.1.1.1 Idle

In this state the MAC sublayer shall wait until the MAC-user starts an identification or data cycle. If the MAC-user requests an identification cycle by means of a MAC_Data.request primitive (cycle=ID_cycle), the MAC sublayer assumes the Await_ID_Transmit state. If the

MAC-user requests a data cycle by means of a MAC_Data.request primitive (cycle=data_cycle), it assumes the Await_Data_Cycle state.

If the MAC sublayer receives characters of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=user_data) via the DL-Ph interface before an identification or data cycle is started, there is a transmission error which is reported to the MAC-user with MAC_Event.indication primitive (event=data_noise). The received characters are rejected.

In this case the MAC-user shall request an identification cycle to restore the data consistency.

If the MAC sublayer receives characters of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface before an identification or data cycle is started, there is a transmission error which is reported to the MAC-user with MAC_Event.indication primitive (event=CRC_noise). The received characters are rejected.

In this case the MAC-user shall request an identification cycle to restore the data consistency.

4.5.3.5.2.1.1.2 Await_ID_Transfer

In this state the MAC sublayer waits for the beginning of the data sequence DLPDU. If it receives the first character of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=user_data) via the DL-Ph interface, it shall assume the Data_Transfer state.

If the receive time monitoring responds, there is a transmission error and the MAC sublayer shall stop the receipt of the data sequence DLPDU and assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

If the MAC sublayer receives characters of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface, there is a transmission error which is reported to the MAC-user with MAC_Event.indication primitive (event=CRC_noise). The received characters are rejected. In this case the MAC-user shall request an identification cycle to restore the data consistency.

4.5.3.5.2.1.1.3 Await_Data_Transfer

In this state the MAC sublayer waits for the beginning of the data sequence DLPDU. If it receives the first character of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=user_data) via the DL-Ph interface, it shall assume the Data_Transfer state.

If the receive time monitoring responds, there is a transmission error and the MAC sublayer shall stop the receipt of the data sequence DLPDU and assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

If the MAC sublayer receives characters of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface, there is a transmission error which is reported to the MAC-user with MAC_Event.indication primitive (event=CRC_noise). The received characters are rejected. In this case the MAC-user shall request an identification cycle to restore the data consistency.

4.5.3.5.2.1.1.4 Data_Transfer

In this state the MAC sublayer shall receive the data sequence DLPDU which is transmitted character by character with the Ph-DATA.indication primitive (PhICI=user_data) via the DL-Ph

interface to the MAC sublayer. After the data sequence DLPDU has been completely received it shall assume the End_Data_Sequence state.

NOTE The quantity of characters which the MAC sublayer of a master received from the PhL in a data sequence DLPDU is always equal to the quantity of characters which is transmitted from this MAC sublayer in a data sequence DLPDU to the PhL.

There is a transmission error if the receive time monitoring circuit responds before the data sequence DLPDU has been completely received. The MAC sublayer shall stop the receipt of the data sequence DLPDU and assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the data sequence DLPDU has been completed.

If the MAC sublayer receives characters of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface, there is a transmission error which is reported to the MAC-user with a MAC_Event.indication primitive (event=CRC_noise). The received characters are rejected. In this case the MAC-user shall request an identification cycle to restore the data consistency.

4.5.3.5.2.1.1.5 End_Data_Sequence

In this state the MAC sublayer shall terminate the receipt of the data sequence DLPDU, generate the received MACSDU, and wait for the beginning of a check sequence DLPDU or the start of a new data sequence by the MAC-user.

If the MAC sublayer receives the first character of a check sequence DLPDU via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=CRC_data), it shall assume the CRC_Data_Transfer state and start to receive the check sequence DLPDU.

If the receive time monitoring circuit responds, there is a transmission error and the MAC sublayer shall assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the check sequence DLPDU has been completed.

There is a transmission error if the MAC sublayer receives a character of a data sequence DLPDU via the DL-Ph interface with a Ph-DATA.indication primitive (PhICI=user_data). This transmission error is communicated to the MAC-user with a MAC_Event.indication primitive (event=data_noise) and, if necessary, in the MAC_Data.confirm primitive after the check sequence DLPDU has been completely transmitted. The MAC sublayer shall reject this character.

If the received and sent loopback word are not identical the MAC sublayer assumes the Idle state.

4.5.3.5.2.1.2 Check sequence

4.5.3.5.2.1.2.1 CRC_Data_Transfer

In this state the MAC sublayer shall receive the checksum transmitted via the DL-Ph interface to the MAC sublayer. The checksum is received with Ph-DATA.indication primitives (PhICI=CRC_data). After the checksum has been completely received, the MAC sublayer shall assume the Check_CRC state.

If the receive time monitoring circuit responds, there is a transmission error and the MAC sublayer shall assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm after the transmission of the check sequence DLPDU has been completed.

There is a transmission error if the MAC sublayer receives a character of a data sequence DLPDU via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=user_data). This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the check sequence DLPDU has been completed. The MAC sublayer shall treat this character like a character of a check sequence DLPDU and continue to receive the checksum.

4.5.3.5.2.1.2.2 Check_CRC

In this state the MAC sublayer shall compare the received checksum with the one it previously generated of the last data sequence DLPDU. If the two checksums are identical, the data sequence DLPDU received last was received without errors. Otherwise, there is a transmission error which is communicated to the MAC-user after the end of the check sequence in the MAC_Data.confirm primitive.

If the MAC sublayer receives the first character of a checksum status (PhICI=CRC_status) via the DL-Ph interface by means of a Ph-DATA.indication primitive, the MAC sublayer shall assume the CRC_Status_Transfer state and start to receive the checksum status.

If the receive time monitoring responds, there is a transmission error and die MAC sublayer shall assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the check sequence DLPDU has been completed.

4.5.3.5.2.1.2.3 CRC_Status_Transfer

In this state the MAC sublayer shall receive the checksum status transmitted via the DL-Ph interface to the MAC sublayer. The checksum status is received character by character with the Ph-DATA.indication primitives (PhICI=CRC_status). After the first four characters of the checksum status (r0…r3) have been completely received, the MAC sublayer shall assume the Check_Receive_Error state.

If the receive time monitoring responds, there is a transmission error and the MAC sublayer shall assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the check sequence DLPDU has been completed.

4.5.3.5.2.1.2.4 Check_Receive_Error

In this state, the MAC sublayer shall evaluate the logic state of the first four characters r0…r3 of the received checksum status. If the bit-by-bit logical AND combination $r3 \wedge r1 \wedge r0$ assumes the binary value of logical 1, the data sequence DLPDU received last was received without errors. Otherwise, there is a transmission error which is communicated to the MAC-user in the MAC_Data.confirm primitive after the check sequence has been completed.

If the MAC sublayer received a character of a check sequence via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=CRC_status), it shall assume the End_Check_Sequence state and continue to receive the checksum status.

4.5.3.5.2.1.2.5 End_Check_Sequence

In this state the MAC sublayer shall continue to receive the checksum status. After the checksum status has been completely received, the MAC sublayer shall terminate the receipt of the check sequence DLPDU and assume the Idle state.

If the receive time monitoring responds, there is a transmission error and the MAC sublayer shall assume the Idle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the check sequence DLPDU has been completely transferred.

4.5.3.5.2.2 Data sequence identification

Figure 62 shows the state transitions in the MAC sublayer of a master for the identification of a data sequence as an identification cycle or data cycle.

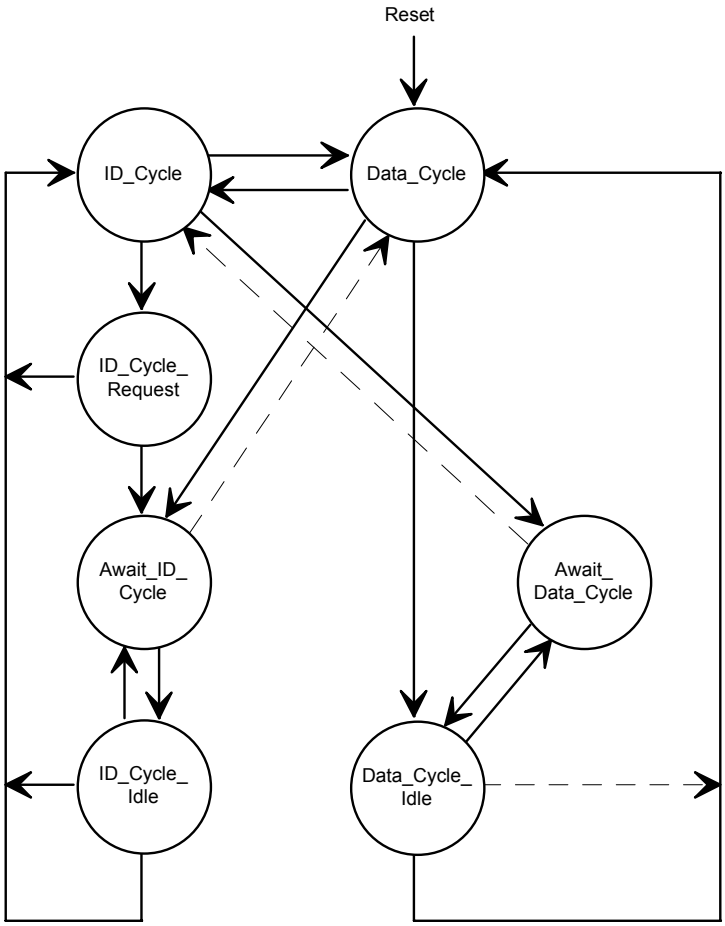


Figure 62 – MAC sublayer of a master: data sequence identification

4.5.3.5.2.2.1 Identification cycle

4.5.3.5.2.2.1.1 ID_Cycle

In this state, the MAC sublayer shall wait for the MAC-user to start an identification or data cycle. If the MAC-user requests an identification cycle by means of a MAC_Data.request primitive (cycle=ID_cycle), the MAC sublayer assumes the ID_Cycle_Request state. If the MAC-user requests a data cycle by means of a MAC_Data.request primitive (cycle=data_cycle), it assumes the Await_Data_Cycle state.

If the MAC receives the beginning of a data cycle via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=data_transfer), there is an error and the MAC sublayer assumes the Data_Cycle state.

4.5.3.5.2.2.1.2 ID_Cycle_Request

If the MAC sublayer receives in this state the request for a data cycle (PhICI=data_transfer) via the DL-Ph interface by means of a Ph-DATA.indication primitive, it assumes the Await_ID_Cycle state.

If the receive time monitoring circuit responds, there is a transmission error and the MAC sublayer shall assume the ID_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

If the MAC sublayer receives the first character of a data sequence DLPDU (PhICI=user_data), there is a transmission error and the MAC sublayer shall assume the ID_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

If the MAC sublayer receives the first character of a check sequence DLPDU (PhICI=CRC_data), there is also a transmission error and the MAC sublayer shall assume the ID_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the data sequence DLPDU has been completed.

4.5.3.5.2.2.1.3 Await_ID_Cycle

In this state the MAC sublayer waits for the beginning of an identification cycle. If the MAC sublayer receives the beginning of an identification cycle via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=ID_transfer), it assumes the ID_Cycle_Idle state.

If the receive time monitoring responds, there is a transmission error and the MAC sublayer shall assume the Data_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

If the MAC sublayer receives the first character of a data sequence DLPDU (PhICI=user_data), there is a transmission error and the MAC sublayer shall assume the Data_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the data sequence DLPDU has been completely transmitted.

If the MAC sublayer receives the first character of a check sequence DLPDU (PhICI=CRC_data), there is also a transmission error and the MAC sublayer shall assume the Data_Cycle state.

4.5.3.5.2.2.1.4 ID_Cycle_Idle

In this state the MAC sublayer waits for the beginning of the data transmission. If it receives the first character of a data sequence DLPDU (PhICI=user_data) via the DL-Ph interface by means of a Ph-DATA.indication primitive, it assumes the ID_Cycle state. There is a transmission error if the layer receives the first character of a check sequence DLPDU (PhICI=CRC_data). The MAC sublayer shall also assume the ID_Cycle state.

If the MAC sublayer receives the request for an identification cycle (PhICI=data_transfer) via the DL-Ph interface by means of a Ph-DATA.indication primitive there is an error, and the MAC sublayer assumes the Await_ID_Cycle state.

If the receive time monitoring circuit responds, there is a transmission error and the MAC sublayer shall assume the ID_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

4.5.3.5.2.2.2 Data cycle

4.5.3.5.2.2.2.1 Data_Cycle

In this state, the MAC sublayer shall wait for the MAC-user to start an identification or data cycle. If the MAC-user requests an identification cycle by means of a MAC_Data.request primitive (cycle=ID_cycle), the MAC sublayer assumes the Await_ID_Request state. If the MAC-user requests a data cycle by means of a MAC_Data.request primitive (cycle=data_cycle), it assumes the Data_Cycle_Idle state.

If the MAC receives the beginning of a data cycle via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=data_transfer), there is an error and the MAC sublayer assumes the ID_Cycle state.

4.5.3.5.2.2.2.2 Await_Data_Cycle

In this state, the MAC sublayer waits for the beginning of a data cycle. If the MAC sublayer receives the beginning of a data cycle via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=data_transfer), it assumes the Data_Cycle_Idle state.

If the receive time monitoring responds, there is a transmission error and the MAC sublayer shall assume the ID_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

If the MAC sublayer receives the first character of a check sequence DLPDU (PhICI=CRC_data), there is also a transmission error and the MAC sublayer shall assume the ID_Cycle state.

If the MAC sublayer receives the first character of a data sequence DLPDU (PhICI=user_data), there is a transmission error and the MAC sublayer shall assume the ID_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the data sequence DLPDU has been completely transmitted.

4.5.3.5.2.2.2.3 Data_Cycle_Idle

In this state the MAC sublayer waits for the beginning of the data transmission. If it receives the first character of a data sequence DLPDU (PhICI=user_data) via the DL-Ph interface by means of a Ph-DATA.indication primitive, it assumes the Data_Cycle state. There is a transmission error if the layer receives the first character of a check sequence DLPDU (PhICI=CRC_data). The MAC sublayer shall also assume the Data_Cycle state.

If the MAC sublayer receives the beginning of an identification cycle with a Ph-DATA.indication primitive (PhICI=ID_transfer) via the DL-Ph interface, there is an error and the MAC sublayer assumes the Await_Data_Cycle state.

If the receive time monitoring circuit responds, there is a transmission error and the MAC sublayer shall assume the Data_Cycle state. This transmission error is communicated to the MAC-user in the MAC_Data.confirm primitive after the transmission of the data sequence DLPDU has been completed.

4.5.4 Slave

4.5.4.1 DLPDU structure

The following DLPDU formats are distinguished: **data sequence DLPDU** and **check sequence DLPDU**.

4.5.4.1.1 Data sequence DLPDU

The MAC sublayer of a slave with the address k shall remove the MACSDU (shown at the right of Figure 63) from a data sequence DLPDU received via the DL-Ph interface as shown in Figure 63 and, if the transmission was error-free, transmit this MACSDU via the MAC-user interface to the MAC-user. The data sequence DLPDU is received from left to right.

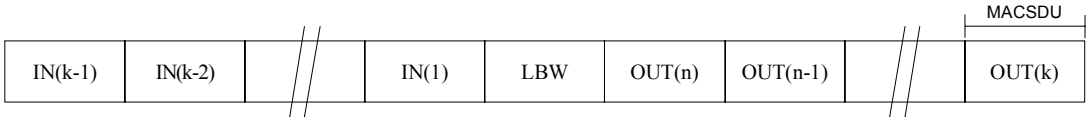


Figure 63 – Data sequence DLPDU received by a slave

Conversely, the MAC sublayer of a slave with the address k shall generate the data sequence DLPDU as shown in Figure 64, by removing the MACSDU destined for its MAC-user from the received data sequence DLPDU and transmitting the rest of the received data sequence DLPDU together with the MACSDU (shown at the left of Figure 64) via the DL-Ph interface to the PhL. This MACSDU was transmitted via the MAC-user interface to the MAC sublayer before. The data sequence DLPDU is transmitted from left to right in the form of PhIDUs, so that first the MACSDU sent from the MAC-user is transmitted to the MAC sublayer and then the received DLPDU without the MACSDU destined for the MAC-user.

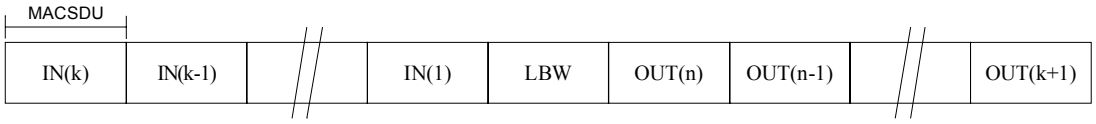


Figure 64 – Data sequence DLPDU transmitted by a slave

4.5.4.1.2 Check sequence DLPDU

For a secured transmission of the data sequence DLPDU the MAC sublayer of a slave transmits a check sequence DLPDU. For this purpose, the MAC sublayer for the transmitted data sequence DLPDU generates a checksum as specified in 4.5.3.3.1, but transmits the 16 redundant check bits together with the checksum status of a check sequence as a check sequence DLPDU via the DL-Ph interface to the PhL.

Conversely, the MAC sublayer of a slave shall compare the 16-bit checksum of a check sequence DLPDU with the checksum generated for the data sequence DLPDU received immediately prior and evaluate the checksum status. The result is communicated to the MAC-user.

4.5.4.2 Checksum

See 4.5.3.3.

4.5.4.3 Checksum status

The MAC sublayer of a slave shall evaluate a checksum status according to Figure 65 that was received in a check sequence DLPDU.

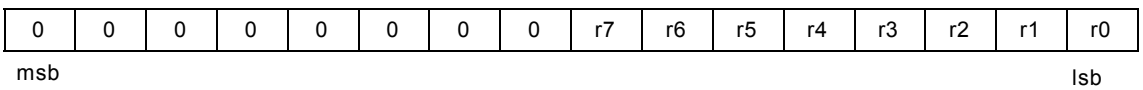


Figure 65 – Checksum status received by the slave

For the logical binary values r7…r0 is: r7=r6=r5=r4=r3∧r2∧r1∧r0, where "∧" represents the bit-wise AND operator.

NOTE 1 If the expression r3∧r2∧r1∧r0 assumes the value logical 1, both the data sequence DLPDUs and the checksum between the master and the evaluating device were transmitted without errors.

NOTE 2 If the expression r3∧r2∧r1∧r0 assumes the value 0, then a transmission error occurred either when the data sequence DLPDUs were transmitted or the checksums between the master and the evaluating device.

The checksum status is received from right to left in a check sequence DLPDU immediately after the checksum has been received. The lsb is transmitted first and the msb last.

Conversely, the MAC sublayer of a slave shall generate a checksum status according to Figure 66.

0	0	0	0	0	0	0	0	t7	t6	t5	t4	t3	t2	t1	t0
msb															lsb

Figure 66 – Checksum status generated by the slave

The logical binary values t7…t0 mean the following:
t0 = CRC_receive_error∧r0∧RxSL_Error
t1 = t0∧r1
t2 = t1∧r2
t3 = t2∧r3
t7 = t6=t5=t4=t3

Here r3…r0 represent the corresponding binary values of the received checksum status and "∧" the bit-by-bit AND operator.

NOTE 3 If the data sequence DLPDU received last did not contain any errors, that is, the checksum received next in a check sequence DLPDU is identical with the checksum generated from the data sequence DLPDU received last, the CRC_receive_error assumes the value logical 1 and otherwise the value logical 0.

The checksum status is transmitted in a check sequence DLPDU immediately after the checksum has been transmitted, starting with the lsb and ending with the msb.

4.5.4.4 Bus access control

Figure 67 and Figure 68 together show the protocol machine of the MAC sublayer of a slave for the bus access control.

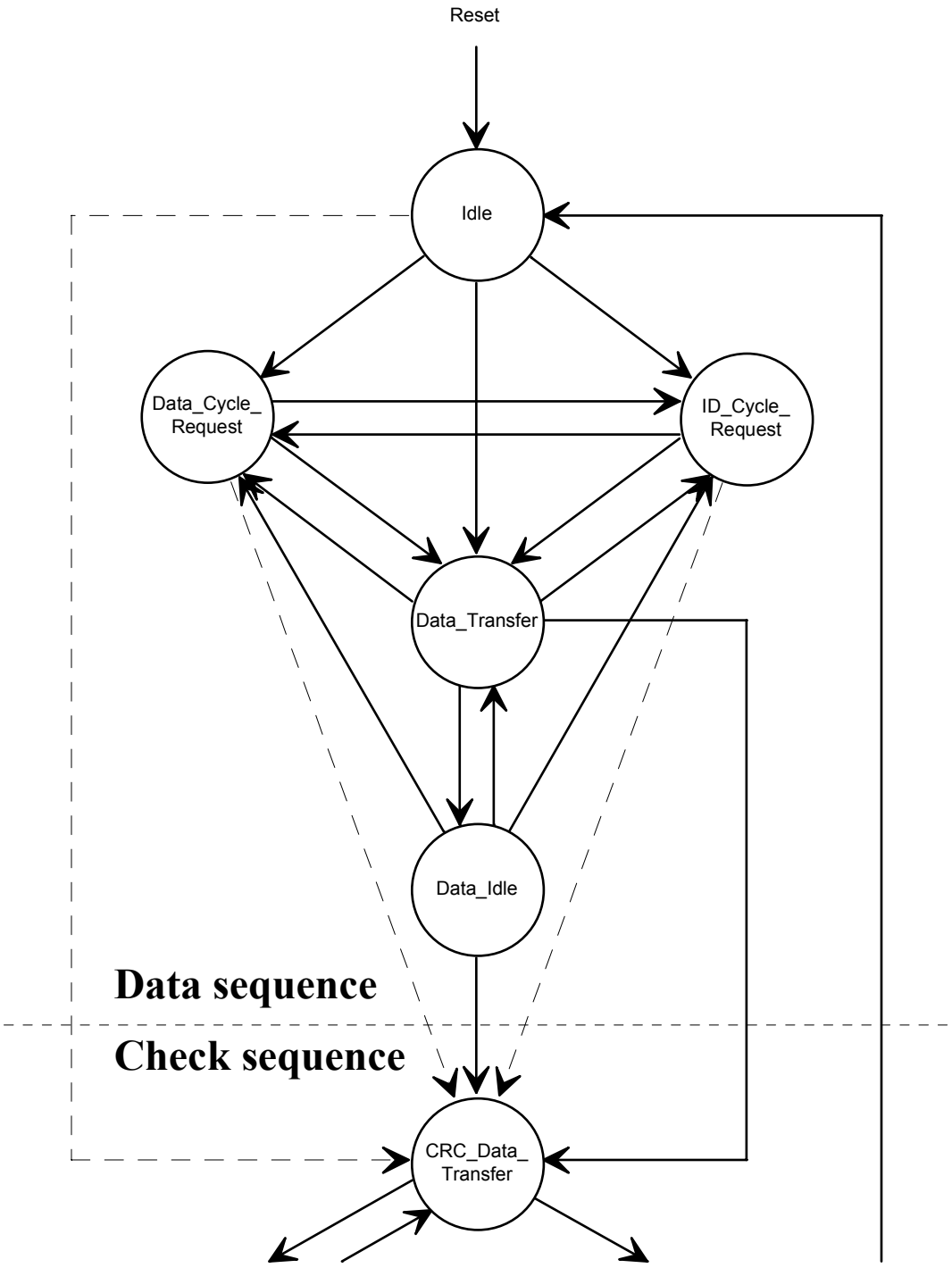


Figure 67 – State transitions of the MAC sublayer of a slave: data sequence

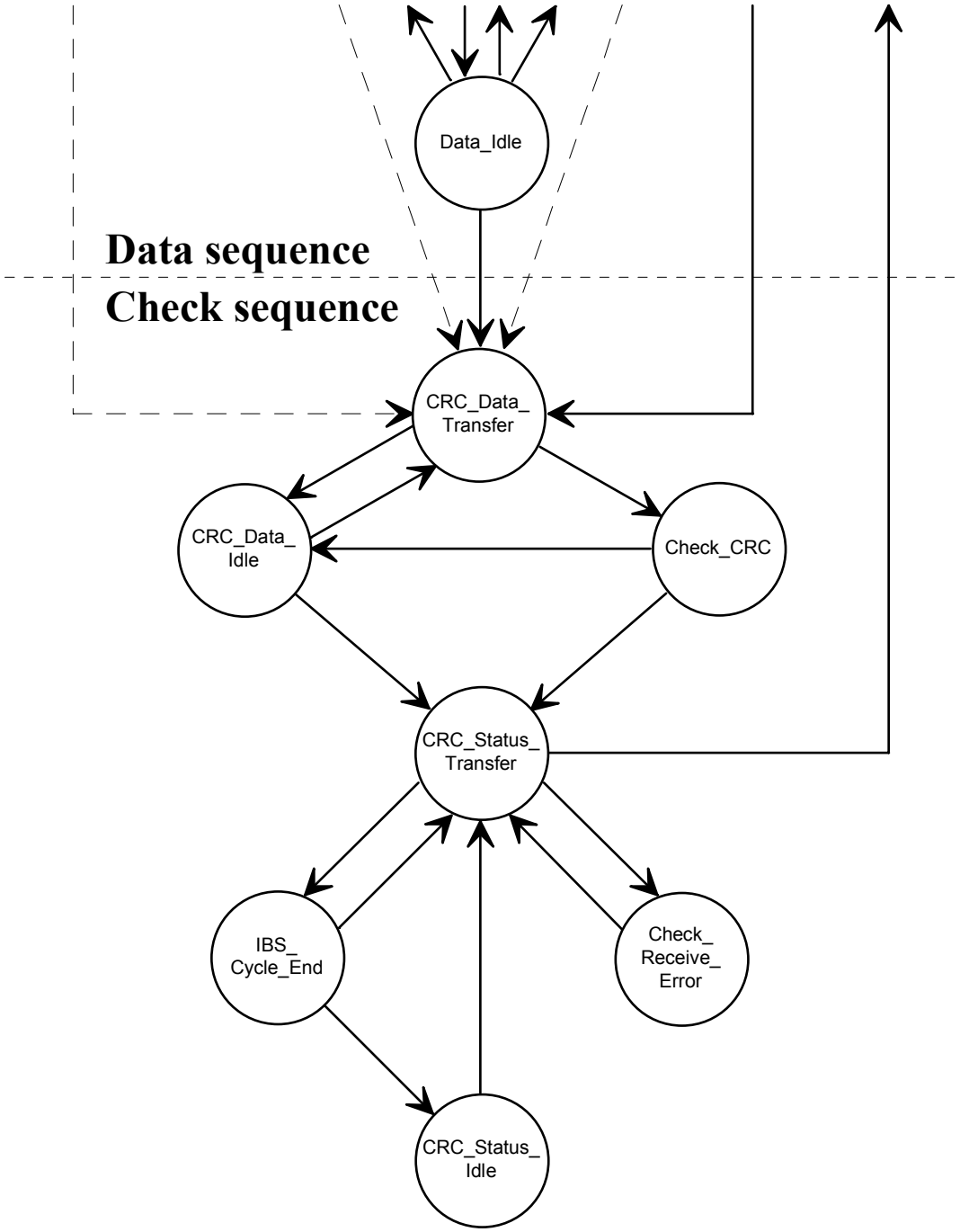


Figure 68 – State transitions of the MAC sublayer of a slave: check sequence

4.5.4.4.1 Basic state

Idle

In this state the MAC sublayer shall first set the checksums for the data sequence DLPDU to be received via the DL-Ph interface and the data sequence DLPDU to be transmitted to its initial value and then wait for the receipt of a Ph-DATA.indication primitive.

If the MAC sublayer receives the request for an identification cycle (PhICI=ID_transfer) via the DL-Ph interface by means of a Ph-DATA.indication primitive, it shall assume the ID_Cycle_Request state. If it receives the request for a data cycle (PhICI=data_transfer), is shall assume the Data_Cycle_Request state.

If the MAC sublayer receives the first character of a data sequence DLPDU via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=user_data), it shall take the MACSDU to be transmitted from the Data_Transmit buffer, generate the data sequence DLPDU to be transmitted and assume the Data_Transfer state. While the MACSDU is accepted, any other access to the Data_Transmit buffer is locked.

An error occurred if the MAC sublayer receives a character of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface. The MAC sublayer shall assume the CRC_Data_Transfer state in order to start to receive and transmit the check sequence DLPDU.

NOTE The MAC sublayer also assumes this state after power on or a reset.

4.5.4.4.2 Data sequence

ID_Cycle_Request

In this state the MAC sublayer shall request an identification cycle by means of a Ph-DATA.request primitive (PhICI=ID_transfer). The PhL confirms the request with a Ph-DATA.confirm primitive. Before that, the management is informed by means of a MAC_Event.indication primitive (ID_Cycle_Request).

If the MAC sublayer receives the first character of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=user_data) via the DL-Ph interface, it shall assume the Data_Transfer state. While the MACSDU is accepted.

If the MAC sublayer receives the request for a data cycle (PhICI=data_transfer) via the DL-Ph interface by means of a Ph-DATA.indication primitive, it shall assume the Data_Cycle_Request state.

An error occurred if the MAC sublayer received a character of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface. The MAC sublayer shall assume the CRC_Data_Transfer state in order to begin to receive and transmit the check sequence DLPDU.

Data_Cycle_Request

In this state the MAC sublayer shall request a data cycle by means of a Ph-DATA.request primitive (PhICI=data_transfer). The PhL confirms the request with a Ph-DATA.confirm primitive. Before that, the management is informed by means of a MAC_Event.indication primitive (Data_Cycle_Request).

If the MAC sublayer receives the first character of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=user_data) via the DL-Ph interface, it shall assume the Data_Transfer state.

If the MAC sublayer receives the request for an identification cycle (PhICI=ID_transfer) via the DL-Ph interface by means of a Ph-DATA.indication primitive, it shall assume the ID_Cycle_Request state and accept the data from the ID_Transmit_Buffer.

An error occurred if the MAC sublayer received a character of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface. The MAC sublayer shall assume the CRC_Data_Transfer state in order to begin to receive and transmit the check sequence DLPDU.

Data_Transfer

In this state, the MAC sublayer shall receive the data sequence DLPDU to be transmitted bit-by-bit via DL-Ph interface to the MAC sublayer by means of Ph-DATA.indication primitives (PhICI=user_data) and on its part transmit the data sequence DLPDU to be sent bit-by-bit by means of Ph-DATA.request primitives (PhICI=user_data) via the DLL–PhL interface to the PhL. The transmission of a character is confirmed to the MAC sublayer with a Ph-DATA.confirm primitive.

The data sequence DLPDU to be sent is transmitted synchronously with the receipt of a data sequence DLPDU, that is, each Ph-DATA.indication primitive with PhICI=user_data causes the sending of a Ph-DATA.request primitive with PhICI=user_data.

If the MAC sublayer receives the beginning of an identification cycle (PhICI=ID_transfer) via the DL-Ph interface by means of a Ph-DATA.indication primitive, the MAC sublayer shall first terminate the receipt and the transmission of the data sequence DLPDUs and then assume the ID_Cycle_Request state.

If the MAC sublayer receives the beginning of a data cycle (PhICI=data_transfer), the MAC sublayer shall first complete the receipt and the transmission of the data sequence DLPDUs and then assume the Data_Cycle_Request state.

If the MAC sublayer is notified with a Ph-DATA.indication primitive (PhICI=user_data_idle) via the DL-Ph interface that the Data_Idle state was detected on the bus, it completes the receipt and transmission of the data sequence DLPDUs and assumes the Data_Idle state.

If the MAC sublayer receives the first character of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface, it shall terminate the receipt and the transmission of the DLPDUs, generate the received MACSDU and assume the CRC_Data_Transfer state.

Data_Idle

If the MAC sublayer receives the character of a data sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=user_data) via the DL-Ph interface, it shall assume the Data_Transfer state and continue to receive and transmit the data sequence of DLPDUs.

If the MAC sublayer receives the beginning of an identification cycle (PhICI=ID_transfer) via the DL-Ph interface by means of a Ph-DATA.indication primitive, the MAC sublayer shall terminate the reception and the transmission of the DLPDUs by means of data sequence. Additional the MAC turns into the ID_Cycle_Request state.

If the MAC sublayer receives the beginning of a data cycle (PhICI=data_transfer), the MAC sublayer shall first terminate the receipt and the transmission of the data sequence DLPDUs and then assume the Data_Cycle_Request state. In addition, the error flag RxSL_Error is set.

If the MAC sublayer receives the first character of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface, it shall terminate the

receipt and transmission of the DLPDUs, generate the received MACSDU and assume the CRC_Data_Transfer state. In addition, the error flag RxSL_Error is set.

4.5.4.4.3 Check sequence

CRC_Data_Transfer

In this state, the MAC sublayer shall receive the checksum transmitted by means of the Ph-DATA.indication primitive (PhICI=CRC_data) via the DL-Ph interface to the MAC sublayer and on its part transmit the checksum to be sent by means of Ph-DATA.request primitive (PhICI=CRC_data) via the DL-Ph interface to the PhL. The checksums are received and transmitted character by character in a check sequence DLPDU. The transmission of a character is confirmed to the MAC sublayer with a Ph-DATA.confirm primitive.

An error occurred if the MAC sublayer receives a character of a data sequence DLPDU instead of a checksum character with a Ph-DATA.indication primitive (PhICI=User_Data). In this case, the MAC sublayer shall treat the received character like the character of a checksum, transmit on its part the next character to be sent of its checksum by means of a Ph-DATA.request primitive (PhICI=CRC_data) via the DL-Ph interface to the PhL and continue to receive and transmit the checksums.

The transmission of the check sequence DLPDU to be sent is done synchronously with the receipt of the check sequence DLPDU, that is, each Ph-DATA.indication primitive with PhICI=CRC_data causes the sending of a Ph-DATA.request primitive with PhICI=CRC_data.

After the checksums have been completely received and transmitted, the MAC sublayer shall assume the Check_CRC state to compare the received checksum with the one generated by the layer itself.

If the MAC sublayer is notified via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=CRC_data_idle) that the CRC_Data_Idle state was detected on the bus, it interrupts the receipt and the transmission of the check sequence DLPDUs and assumes the CRC_Data_Idle state.

CRC_Data_Idle

In this state, the MAC sublayer shall request the CRC_Data_Idle state on the bus by means of a Ph-DATA.request primitive (PhICI=CRC_data_idle) via the DL-Ph interface. The PhL confirms the request with a Ph-DATA.confirm primitive.

If the MAC sublayer receives a character of a check sequence DLPDU via the DL-Ph interface by means of a Ph-DATA.indication primitive, it shall assume the CRC_Data_Transfer state, if it was a checksum character (PhICI=CRC_data), and continue to transmit the checksums. If it was the first character of a checksum status (PhICI=CRC_status) the MAC sublayer shall assume the CRC_Status_Transfer state and begin to receive and transmit the checksum status.

An error occurred if the MAC sublayer receives a character of a data sequence DLPDU. In this case the MAC sublayer shall treat the received character like the character of a checksum and assume the CRC_Data_Transfer state.

Check_CRC

In this state the MAC sublayer shall compare the received checksum with the one previously generated of the last data sequence DLPDU. If both checksums are identical, the data sequence DLPDU received last was received without errors, and the CRC_receive_error flag assumes the value logical 1. Otherwise, there is a transmission error and the CRC_receive_error flag assumes the value logical 0.

If the MAC sublayer receives the first character of the checksum status (PhICI=CRC_status) via the DL-Ph interface by means of a Ph-DATA.indication primitive, the MAC sublayer shall assume the CRC_Status_Transfer state and begin to receive and transmit the checksum status.

If the MAC sublayer is communicated via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=CRC_data_idle) that the CRC_Data_Idle state was detected on the bus, then it interrupts the receipt and the transmission of the check sequence DLPDUs and assumes the CRC_Data_Idle state.

CRC_Status_Transfer

In this state the MAC sublayer shall receive the checksum status transmitted by means of Ph-DATA.indication primitive (PhICI=CRC_status) via the DL-Ph interface to the MAC sublayer and transmit on its part the checksum status to be sent by means of Ph-DATA.request primitive (PhICI=CRC_status) via the DL-Ph interface to the PhL. The checksum statuses are received and transmitted character by character in a check sequence DLPDU. The transmission of a character is confirmed in the MAC sublayer with a Ph-DATA.confirm primitive.

The check sequence DLPDU to be sent is transmitted synchronously with the receipt of the check sequence DLPDU, that is, each Ph-DATA.indication primitive with PhICI=CRC_status causes the sending of a Ph-DATA.request primitive with PhICI=CRC_status.

After the first four characters of the checksum statuses (r0…r3 and t0…t3) have been completely received, the MAC sublayer shall assume the Check_Receive_Error state in order to transfer the received MACSDU to the MAC-user when the transmission of the data sequence DLPDU received last contained an error.

After the first eight bits of the checksum status (r0…r7 and t0…t7) have been completely received and transmitted, the MAC sublayer shall assume the IBS_Cycle_End state to report the end of a DLPDU cycle to the MAC-user.

If the MAC sublayer is communicated via the DL-Ph interface by means of a Ph-DATA.indication primitive (PhICI=CRC_status_idle) that the CRC_Data_Idle state was detected on the bus, it interrupts the receipt and the transmission of the check sequence DLPDUs and assumes the CRC_Status_Idle state.

Check_Receive_Error

In this state, the MAC sublayer shall evaluate the logic state of the fourth character t3 that was transmitted last of the checksum status to be transmitted. If the binary value logical 1 was transmitted with the character t3, the data sequence DLPDU received last was transmitted without errors. In this case, the MAC sublayer shall generate the MACSDU from the received data sequence DLPDU, make it available to the MAC-user and communicate this event to the user by means of a MAC_Data.indication primitive. If the data sequence DLPDU originates in an identification cycle, the MACSDU is transmitted in the ID_Receive_Buffer, otherwise in the Data_Receive_Buffer.

A transmission error was recognized when the binary value of logical 0 was transmitted with the character t3. The MAC sublayer shall first destroy the data sequence DLPDU received last and, in this case, shall send no message to the MAC-user.

If the MAC sublayer receives a character of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_status) via the DL-Ph interface it shall assume the CRC_Status_Transfer state and continue to receive the transmission of the check sequence DLPDUs with the checksum status.

IBS_Cycle_End

In this state the MAC sublayer shall report to its MAC-user the end of a DLPDU cycle on the DL-subnetwork by means of a MAC_Event.indication primitive (IBS_Cycle_End) and then set the checksums for the data sequence DLPDU to be received via the DL-Ph interface and for the data sequence DLPDU to be transmitted to their initial values. In addition the flags CRC_Receive_Error and RxSL_Error are to be set to false.

If the MAC sublayer receives the Ph-DATA.indication primitive (PhICI=ID_Transfer) upon the complete receipt of the check sequence DLPDU, it assumes the ID_Cycle_Request state.

If the MAC sublayer receives the Ph-DATA.indication primitive (PhICI=Data_Transfer) upon the complete receipt of the check sequence DLPDU, it assumes the Data_Cycle_Request state.

If the MAC sublayer receives the Ph-DATA.indication primitive (PhICI=User_Data) upon the complete receipt of the check sequence DLPDU, it assumes the Data_Transfer state.

NOTE This message is used to synchronize the MAC-users of the devices.

CRC_Status_Idle

In this state, the MAC sublayer shall request the CRC_Status_Idle state on the bus by means of a Ph-DATA.request primitive (PhICI=CRC_status_idle) via the DL-Ph interface. The PhL confirms the request with a Ph-DATA.confirm primitive.

If the MAC sublayer receives a character of a check sequence DLPDU by means of a Ph-DATA.indication primitive (PhICI=CRC_status) via the DL-Ph interface, it shall accept the data from the Data_Transmit_Buffer, assume the IBS_Cycle_End state and continue to receive and transmit the check sequence DLPDUs with the checksum status.

4.5.5 MAC-User – MAC interface

4.5.5.1 General

4.5.5 describes the services which the MAC makes available to the MAC-user. Figure 69 shows the interface between the MAC-user and the MAC in the layer model.

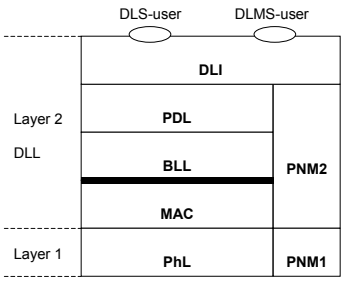


Figure 69 – Interface between MAC-user and MAC in the layer model

4.5.5.2 Overview of the services

The MAC makes the MAC_Cycle service available to the MAC-user. This service is described through the following service primitives:

MAC_Cycle

— MAC_Put_Data.request
— MAC_Put_Data.confirm
— MAC_Get_Data.request
— MAC_Get_Data.confirm
— MAC_Data.request
— MAC_Data.confirm
— MAC_Data.indication

4.5.5.3 Interactions at the MAC-user interface

Figure 70 and Figure 71 show exemplary sequences of interactions at the MAC-user interface for an identification cycle and subsequent data cycle.

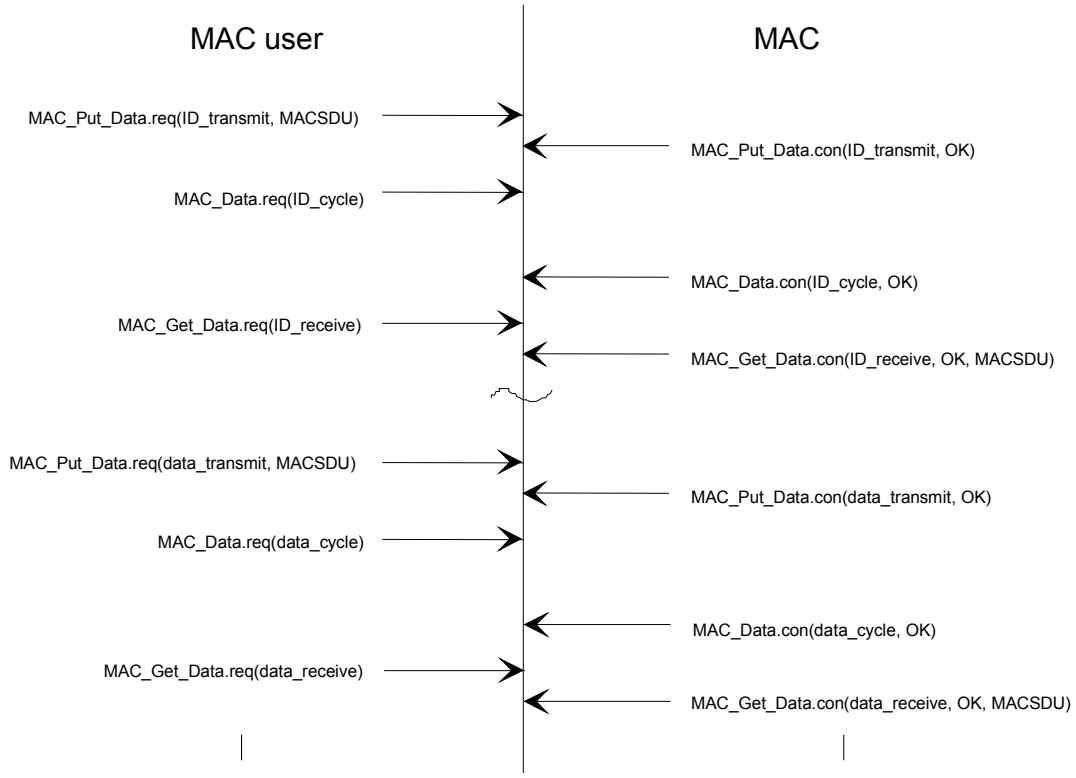


Figure 70 – Interactions at the MAC-user interface (master)

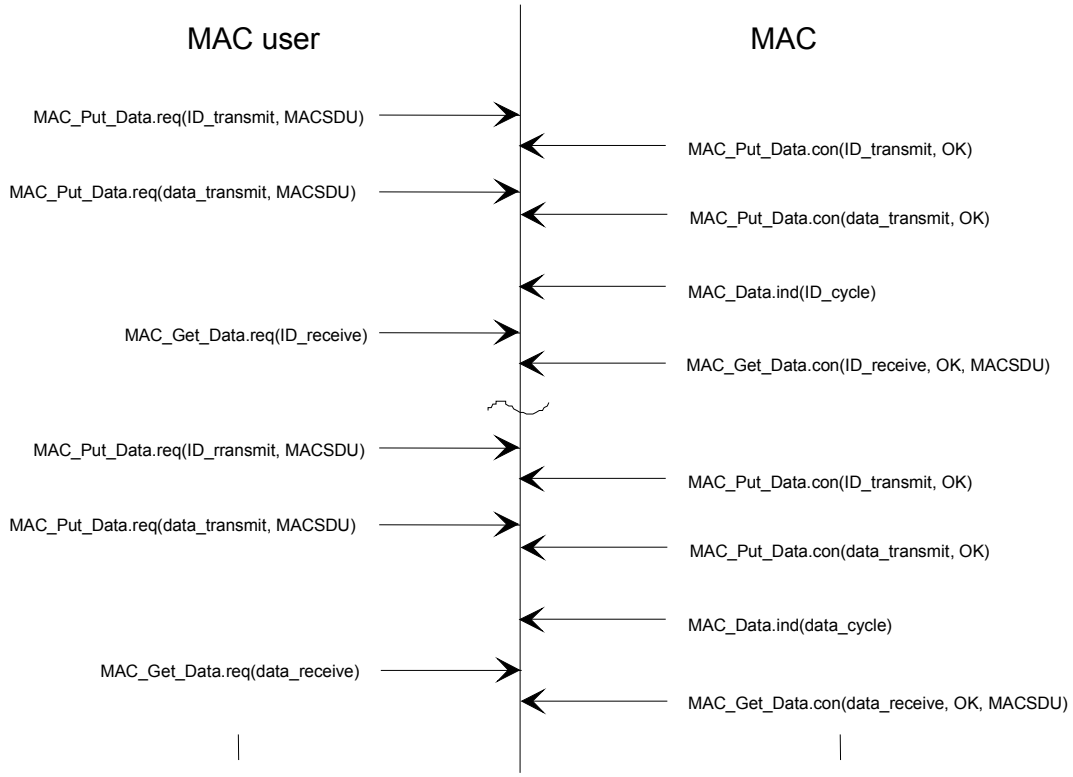


Figure 71 – Interactions at the MAC-user interface (slave)

4.5.5.4 Detailed definitions of the services and interactions

4.5.5.4.1 MAC_Put_Data.request (buffer, MACSDU)

With this service primitive the MAC-user makes the data to be transmitted available to the MAC sublayer. The parameters have the following meanings:

buffer:

This parameter determines the memory area which is to store the data transmitted by the MAC-user. Two memory areas are defined:

buffer = **ID_transmit**
This memory area is available to the management of the master for the transmission of control codes and the slave for the device codes and is transmitted in an identification cycle.

buffer = **data_transmit**
This memory area is available for the transmission of user data and is transmitted in a data cycle.

MACSDU:

This parameter contains the data to be transmitted. The data quantity depends on the number and type of devices in the one-total-DLPDU.

4.5.5.4.2 MAC_Put_Data.confirm (buffer, status)

This service primitive is the confirmation of the MAC sublayer for a MAC_Put_Data.request primitive. The parameters have the following meanings:

buffer:

This parameter has the same meaning as for the MAC_Put_Data.request primitive.

status:

This parameter indicates the result of the data transfer and can assume the following values:

status = **OK**
The data passed on with the MAC_Put_Data.request primitive could be accepted by the MAC sublayer.

status = **NO**
The data passed on with the MAC_Put_Data.request primitive could **not** be accepted by the MAC sublayer, since the requested memory area was occupied when the data was transmitted.

The MAC-user is responsible for the response to this service primitive.

4.5.5.4.3 MAC_Get_Data.request (buffer)

With this service primitive the MAC-user requests data from the MAC sublayer which has been transmitted from one device via the medium to the MAC sublayer. The data is transmitted by means of a MAC_Get_Data.confirm primitive from the MAC sublayer to the MAC-user.

The parameters have the following meanings:

buffer:

This parameter determines the memory area which contains the data to be transmitted to the MAC-user. Two memory areas are defined:

buffer = **ID_receive**
This memory area is available to the MAC-user on the slave for the storage of control codes and the master for device codes which were transmitted in an identification cycle.

buffer = **data_receive**
This memory area is available to the MAC-user for the storage of user data which was transmitted in a data cycle.

4.5.5.4.4 MAC_Get_Data.confirm (buffer, status, MACSDU)

This service primitive is the confirmation of the MAC sublayer to a MAC_Get_Data.request primitive. With this primitive the MAC sublayer transmits the received MACSDU from last DLPDU cycle to the MAC-user.

The parameters have the following meanings:

buffer:

This parameter has the same meaning as for the MAC_Get_Data.request primitive (as specified in 4.5.5.4.3).

status:

This parameter indicates the MAC-user the result of the data transmission and can assume the following values:

status = **OK**
The data could be transmitted to the MAC-user.

status = **NO**
The requested data could **not** be transmitted to the MAC-user.

MACSDU:

If status = OK this parameter contains the data read in during the last message transmission service.

The MAC-user is responsible for responding to this service primitive.

4.5.5.4.5 MAC_Data.request (cycle)

With this service primitive the MAC-user of an active device (master) starts a message DLPDU cycle. The data to be transmitted was previously transferred to the MAC sublayer by means of a MAC_Put_Data.request primitive before.

The parameters have the following meanings:

cycle:

cycle = **ID_cycle**
An identification cycle is started to transmit the control codes to the slaves and to request the device codes of the slaves.

cycle = **data_cycle**
A data cycle for the transmission of user data between the master and the slaves.

4.5.5.4.6 MAC_Data.confirm (cycle, status)

This service primitive is the confirmation of the MAC sublayer for a MAC_Data.request primitive. It indicates that the DLPDU cycle, started by means of the MAC_Data.request primitive, was terminated positively or negatively.

The parameters have the following meanings:

cycle:

This parameter has the same meaning as for the MAC_Data.request primitive.

status:

This parameter indicates whether the requested data transmission could be carried out successfully (status = **OK**), or whether an error occurred during the data transmission (status = **NO**).

The MAC-user is responsible for the response to this service primitive.

4.5.5.4.7 MAC_Data.indication (cycle)

With this service primitive the MAC sublayer indicates to the MAC-user that new data is available from a valid identification cycle or a valid data cycle. The MAC-user can use this data by means of a MAC_Get_Data.request primitive.

The parameter has the following meaning:

cycle:

This parameter indicates whether the received data stems from an identification cycle or a data cycle and in which memory area it is available to the MAC-user. Two values are defined:

cycle = **ID_cycle**
Control and/or ID data was received in an identification cycle. This data is available to the MAC-user in the ID_receive buffer.

cycle = **data_cycle**
User data for the process data channel and the Parameter data channel was received in a data cycle. This data is available to the MAC-user in the data_receive buffer.

The MAC-user is responsible for responding to this service primitive.

4.5.6 MAC-PNM2 interface

4.5.6.1 General

The management of the MAC is part of the MAC that provides the management functionality of the MAC requested by the PNM2. The management of the MAC handles the initialization, the monitoring and the error recovery in the MAC.

4.5.6 defines the administrative MAC management services which are available to the PNM2, together with their service primitives and the associated parameters. Figure 72 shows the interface between MAC and PNM2 in the layer model.

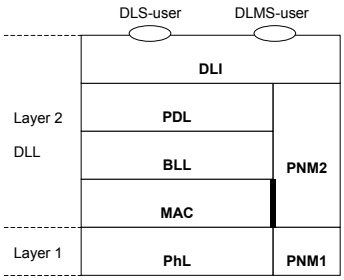


Figure 72 – Interface between MAC and PNM2 in the layer model

The service interface between MAC and PNM2 makes the following functions available.

— Reset of the MAC.

— Request and change of the current operating parameters of the MAC.

— Indication of unexpected events, errors, and status changes which occurred or were detected in the MAC.

4.5.6.2 Overview of the services

The MAC makes the following services available to the PNM2.

— Reset MAC.

— Set Value MAC or Get Value MAC.

— Event MAC.

The MAC services are described by primitives (beginning with MAC_...).

4.5.6.2.1 Reset MAC

The PNM2 uses this required service to reset the MAC. The reset is equivalent to power on. Upon execution, the PNM2 receives a confirmation.

Service primitives:

— MAC_Reset.request

— MAC_Reset.confirm

4.5.6.2.2 Set Value MAC

The PNM2 uses this optional service to set a new value to the MAC variables. Upon completion, the PNM2 receives a confirmation from the MAC whether the defined variables assumed the new value.

Service primitives:

— MAC_Set_Value.request

— MAC_Set_Value.confirm

4.5.6.2.3 Get Value MAC

The PNM2 uses this optional service to read the variables of the MAC. The current value of the defined variable is transmitted in the response of the MAC.

Service primitives:

— MAC_Get_Value.request

— MAC_Get_Value.confirm

4.5.6.2.4 Event MAC

The MAC uses this required service to inform the MAC-user about certain events or errors in the MAC.

Service primitive:

— MAC_Event.indication

4.5.6.3 Overview of the interactions

Figure 73 and Figure 74 show the time relations of the services primitives:

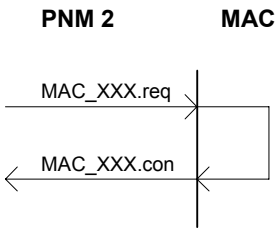


Figure 73 – Reset, Set Value and Get Value MAC services

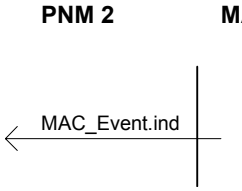


Figure 74 – Event MAC service

4.5.6.4 Detailed definitions of the services and interactions

4.5.6.4.1 MAC_Reset

The MAC_Reset service is mandatory. The PNM2 transfers a MAC_Reset.request primitive to the MAC to reset it (see Table 39).

Table 39 – MAC_Reset

Parameter name	Request	Confirm
Argument	M	
Result(+)		M

4.5.6.4.2 MAC_Set_Value

The MAC_Set_Value service is optional. The PNM2 transfers a MAC_Set_Value.request primitive to the MAC in order to set a defined MAC variable to a desired value. After receipt of this primitive, the MAC tries to select the variable and to set the new value. Upon completion, the MAC transmits a MAC_Set_Value.confirm primitive to the PNM2 (see Table 40).

Table 40 – MAC_Set_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
desired_value	M	
Result(+)		M

variable_name:

This parameter defines the MAC variable which is set to a new value.

desired_value:

This parameter declares the value for the new MAC variable.

Table 41 provides information on which MAC variables may be set to which new values.

Table 41 – MAC variables

Name of MAC variable	Value range	Remarks
Loopback_word (LBW)	Bit 15 = **1**, Bit 14 to Bit 4 are used for setting the LBW, Bit 3 to Bit 0 are used as DLPDU counter (see 4.5.3.2)	master side only
Time_timeout	Value will be defined from the system management and depends from the actual configuration of the DL-subnetwork	master side only

4.5.6.4.3 MAC_Get_Value

The MAC_Get_Value service is optional. The PNM2 transfers a MAC_Get_Value.request primitive to the MAC to read out the current value of a specified MAC variable. After receipt of this primitive, the MAC tries to select the specified variable and to transmit its current value to the PNM2 with a MAC_Get_Value.confirm primitive (see Table 42).

Table 42 – MAC_Get_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
Result(+)		M
current_value		M

variable_name:

This parameter defines MAC variable the value of which is to be read out.

desired_value:

This parameter contains the read-out value of the MAC variable.

The MAC variables to be read are exactly those variables that can be written to with the MAC_Set_Value.

4.5.6.4.4 MAC_Event

The MAC_Event service is mandatory. The MAC transfers a MAC_Event.indication primitive to the PNM2 to inform it about important events or errors in the MAC (see Table 43).

Table 43 – MAC_Event

Parameter name	Indication
Argument	M
event	M

event:

This parameter defines the event which occurred or the error source in the MAC and can assume the following values (see Table 44):

Table 44 – MAC_Event

Name	Meaning	Mandatory/optional
data_cycle_request	The request for a data cycle was detected on the transmission medium.	O
ID_cycle_request	The request for an identification data cycle was detected on the transmission medium.	O
IBS_cycle_end	A DLPDU cycle is ended.	M
data_noise	Before an identification or data cycle was started, characters of a data sequence DLPDU had been received. (bus master only)	M
CRC_noise	Before an identification or data cycle was started, characters of a check sequence DLPDU had been received. (bus master only)	M

4.6 Peripherals network management for layer 2 (PNM2)

4.6.1 Functionality of the PNM2

The management for layer 2 (PNM2) handles the initialization, the monitoring and the error recovery between PNM2-user and the logical functions in the MAC, BLL and PDL (see Figure 75).

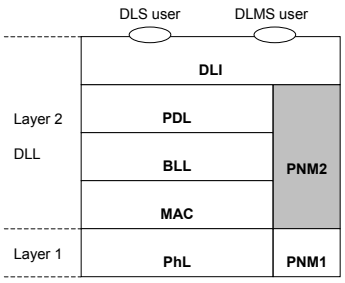


Figure 75 – Location of the PNM2 in the DLL

4.6.2 PNM2-User-PNM2 interface

4.6.2.1 General

4.6.2 defines the administrative PNM2 (Peripherals Network Management for the layer 2) services which are available to the PNM2-user, together with their service primitives and the associated parameters. Figure 76 shows the interface between PNM2-user and PNM2 in the layer model.

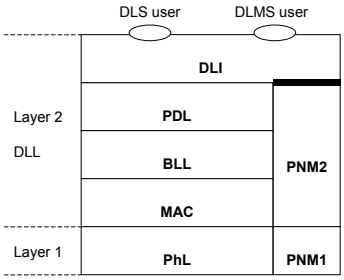


Figure 76 – Interface between PNM2-user and PNM2 in the layer model

The service interface between PNM2-user and PNM2 makes the following functions available.

— Reset of the layer 2 (local)
— Request and change of the current operating parameters of PDL, BLL, MAC (local)
— Indication of unexpected events, errors, and status changes (local and remote)
— Read-out of the active DL-subnetwork configuration
— Read-out of the current DL-subnetwork configuration
— Setting of a certain DL-subnetwork configuration.

4.6.2.2 Overview of the services

4.6.2.2.1 Available services

The PNM2 makes the PNM2-user the following services available.

— Reset PNM2
— Set Value PNM2 or Get Value PNM2
— Event PNM2
— Get Current Configuration PNM2 (master only)
— Get Active Configuration PNM2 (master only)
— Set Active Configuration PNM2 (master only).

4.6.2.2.2 Reset PNM2

The PNM2-user uses this required service to cause the PNM2 to reset Layer 2 (DLL) and itself. The reset is equivalent to power on. The PNM2-user receives a confirmation for this service.

Service primitives:

— PNM2_Reset.request
— PNM2_Reset.confirm

4.6.2.2.3 Set value PNM2

The PNM2-user uses this optional service to set a new value to the variables of the layers 1 or 2. It receives a confirmation on whether the defined variables assumed the new value.

Service primitives:

— PNM2_Set_Value.request
— PNM2_Set_Value.confirm

4.6.2.2.4 Get value PNM2

The PNM2-user uses this optional service to read out variables of the layer 2. The current value of the defined variable is transferred in the response of the PNM2.

Service primitives:

— PNM2_Get_Value.request

— PNM2_Get_Value.confirm

4.6.2.2.5 Event PNM2

The PNM2 uses this required service to inform the PNM2-user on certain events or errors in the layer 2.

Service primitive:

— PNM2_Event.indication

4.6.2.2.6 Get current configuration PNM2 (master only)

The PNM2-user of the master uses this required service to read out the current DL-subnetwork configuration.

Service primitives:

— PNM2_Get_Current_Configuration.request

— PNM2_Get_Current_Configuration.confirm

4.6.2.2.7 Get active configuration PNM2 (master only)

The PNM2-user of the master uses this required service to read out the active DL-subnetwork configuration.

Service primitives:

— PNM2_Get_Active_Configuration.request

— PNM2_Get_Active_Configuration.confirm

4.6.2.2.8 Set active configuration PNM2 (master only)

The PNM2-user of the master uses this required service to set a certain DL-subnetwork configuration.

Service primitives:

— PNM2_Set_Active_Configuration.request

— PNM2_Set_Active_Configuration.confirm

4.6.2.3 Overview of the interactions

The PNM2 services are described by the following primitives (beginning with PNM2_…):

Figure 76, Figure 77, Figure 78 and Figure 79 show the time relations of the services primitives:

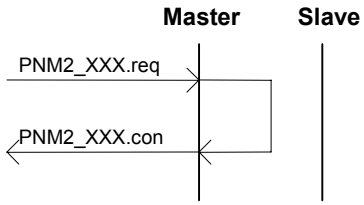


Figure 77 – Reset, Set Value, Get Value and Get Active Configuration services

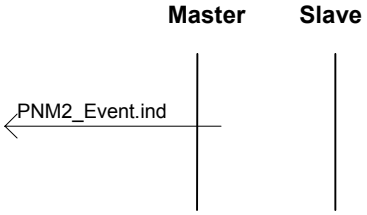


Figure 78 – Event PNM2 service

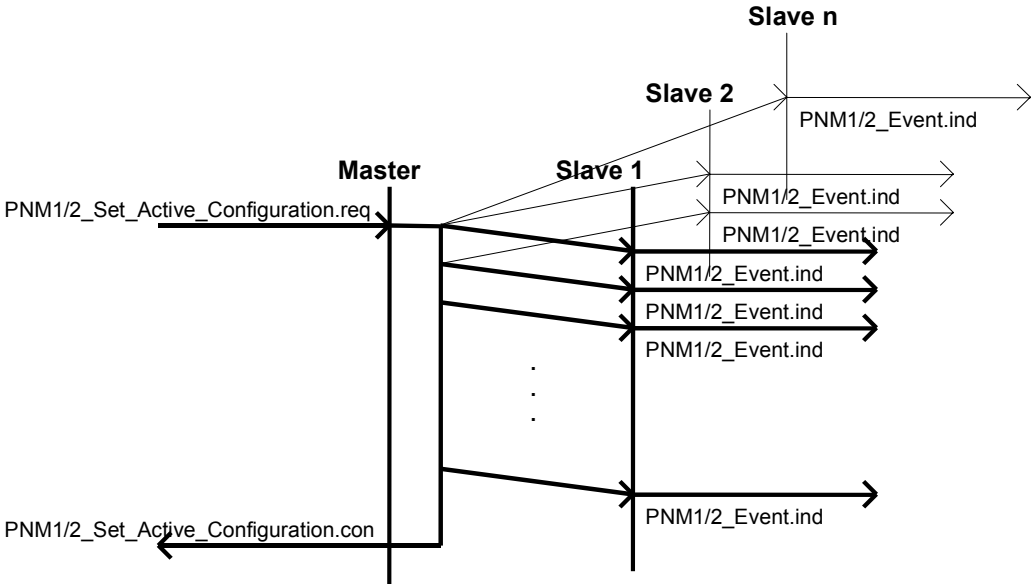


Figure 79 – Set Active Configuration, Get Current Configuration service

4.6.2.4 Detailed definition of the services and interactions

4.6.2.4.1 PNM2_Reset

The PNM2_Reset service is mandatory. The PNM2-user passes a PNM2_Reset.request primitive to the PNM2 to cause it to reset the layer 2.

After the confirmations of the PDL, BLL and MAC by means of corresponding confirmation primitives the PNM2 resets itself and communicates a PNM2_Reset.confirm primitive to the PNM2-user (see Table 45).

Table 45 – PNM2_Reset

Parameter name	Request	Confirm
Argument	M	
Result(+)		S
M_status (=OK)		M
Result(-)		S
M_status (NOK)		M

M_status:

This parameter contains a confirmation on the execution of the service. The following possible values are defined (see Table 46):

Table 46 – M_status values of the PNM2_Reset

Value	Meaning
OK	Positive confirmation; the reset function was carried out successfully.
NOK	Failure

4.6.2.4.2 PNM2_Set_Value

The PNM2_Set_Value-Service is optional. The PNM2-user transfers a PNM2_Set_Value.request primitive to the PNM2 to set a defined variable of the layer 2 to a requested value. The management transmits the individual PDL, BLL, MAC and/or Ph-SET-VALUE.request primitives to the corresponding layers and sends a PNM2_Set_Value.confirm primitive to the PNM2-user after it has received all associated confirmation primitives (see Table 47).

Table 47 – PNM2_Set_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
desired_value	M	
Result(+)		S
M_status (=OK)		M
Result(-)		S
M_status (≠OK)		M

variable_name:

This parameter contains a variable of the PDL, BLL or MAC. The selectable variables are defined in the corresponding subclauses of the individual layers.

desired_value:

This parameter contains a value for the selected variable. The permitted values or value ranges are determined in the corresponding subclauses of the individual layers.

M_status

This parameter contains a confirmation on the execution of the service. The following possible values are determined (see Table 48):

Table 48 – M_status values of the PNM2_Set_Value

Value	Meaning
OK	Positive confirmation; the variable has the new value.
NO	The variable does not exist or could not assume the new value.
IV	Invalid parameters in the request

4.6.2.4.3 PNM2_Get_Value

The PNM2_Get_Value service is optional. The PNM2-user transfers a PNM2_Get_Value.request primitive to the PNM2 in order to read out the current value of a specified variable of the layer 2. The management transfers the individual PDL, BLL, MAC and/or Ph_Get_Value.request primitives to the corresponding layers and sends a PNM2_Get_Value.confirm primitive with requested values to the PNM2-user after all associated confirmation primitives have been received (see Table 49).

Table 49 – PNM2_Get_Value

Parameter name	Request	Confirm
Argument	M	
variable_name	M	
Result(+)		S
current_value		M
M_status (=OK)		M
Result(-)		S
M_status (≠OK)		M

variable_name:

This parameter contains a variable of the PDL, BLL, MAC or PhL. The selectable variables are defined in the corresponding subclauses of the individual layers.

current_value:

This parameter receives the current value for the selected variable.

M_status:

This parameter contains a confirmation about the execution of the service. The following possible values are defined (see Table 50).

Table 50 – M_status values of the PNM2_Get_Value

Value	Meaning
OK	Positive confirmation; the variable could be read.
NO	The variable does not exist or could not be read.
IV	Invalid parameters in the request

4.6.2.4.4 PNM2_Event

The PNM2_Event-Service is mandatory. After receipt from PDL, BLL and MAC a indication the PNM2 transfers a PNM2_Event.indication primitive to the PNM2-user to inform it about important events or errors in the layers. After DL-subnetwork errors have been reported, the PNM2 carries out a configuration check. If the configuration differs from the configuration prior to the DL-subnetwork error, the PNM2 automatically generates an event with information on the configuration change (see Table 51)

Table 51 – PNM2_Event

Parameter name	Indication
Argument	M
event	M
add_info	C

event:

This parameter defines the event which occurred or the error cause. The possible values are defined in the corresponding subclauses of the respective layers. The possible values of the errors which occurred in the PNM2 are defined in Table 52.

Table 52 – MAC Events

Name	Meaning
Configuration_change	The configuration of the DL-subnetwork changed during operation

add_info:

This parameter contains additional information about the events or errors which occurred.

4.6.2.4.5 PNM2_Get_Current_Configuration

The PNM2-user of the master uses this service to read out the current configuration of the DL-subnetwork. To do so, the PNM2 carries out ID cycles to detect the currently connected slaves and transfers the detected configuration to the PNM2-user in the current_configuration parameter. The configuration of the DL-subnetwork after the service has been executed can be determined with the network_configuration parameter(see Table 53).

Table 53 – PNM2_Get_Current_Configuration

Parameter name	Request	Confirm
Argument	M	
network_configuration	M	
Result(+)		S
current_configuration		M
Result(-)		S
error_type		M

current_configuration:

This parameter contains the current configuration of the DL-subnetwork. The parameter is equivalent to the active_configuration parameter of the Get_Active_Configuration service.

network_configuration:

This parameter defines the configuration of the DL-subnetwork after the service has been carried out.

Closed: The outgoing interfaces of all slaves are closed.

Open: The outgoing interfaces of all slaves are open.

error_type:

This parameter specifies why the service could not be executed successfully. Possible error causes:

— An error was detected when a ring segment was connected.

— No ID cycles could be run (DL-subnetwork error).

add_info:

This parameter provides additional information about the error cause (for example, ring segment number when the outgoing interface could not be opened)

4.6.2.4.6 PNM2_Get_Active_Configuration

The PNM2-user of the master uses this service to read out the active configuration of the DL-subnetwork. The PNM2 transfers the currently active configuration in the active_configuration parameter to the PNM2-user. To provide the service, the PNM2 does not need to run ID cycles. The service is locally responded to. The PNM2 logs all changes of the configuration, so that the active_configuration parameter which is kept locally is always up to date (see Table 54).

Table 54 – PNM2_Get_Active_Configuration

Parameter name	Request	Confirm
Argument	M	
Result(+)		S
active_configuration		M
Result(-)		S
error_type		M

active_configuration:

This parameter contains the active configuration of the DL-subnetwork. The entries in the list are ordered according to the physical order of the slaves in the ring. The parameter has the following structure according to Figure 80:

ID code of the 1st slave	Ring segment level of the 1st slave
ID code of the 2nd slave	Ring segment level of the 2nd slave
…	…
ID code of the nth slave	Ring segment level of the nth slave

Figure 80 – The active_configuration parameter

error_type:

This parameter indicates why the service could not be executed successfully.

4.6.2.4.7 PNM2_Set_Active_Configuration

The PNM2-user of the master uses this service to generate a certain active configuration of the DL-subnetwork. The PNM2 converts the target configuration in control commands for the switching on or off of certain ring segments. If the new configuration cannot be accepted, the exact error cause is communicated to the PNM2-user and the old configuration is retained (see Table 55).

Table 55 – PNM2_Set_Active_Configuration

Parameter name	Request	Confirm
Argument	M	
active_configuration	M	
Result(+)		S
Result(-)		S
error_type		M
add_info		C

active_configuration:

This parameter contains the new active configuration of the DL-subnetwork to be generated. The structure corresponds to the structure of the Active_Configuration parameter of the Get_Active_Configuration service.

error_type:

This parameter indicates why the service could not be executed successfully. Possible error causes:

— An error was detected when a ring segment was connected to the ring. The new configuration could not be generated.

— No ID cycles could be run; a fatal bus error.

add_info:

This parameter provides additional information about the error cause (for example, ring segment number when the outgoing interface could not be opened).

4.7 Parameters and monitoring times of the DLL

The DLL parameters and times described below are used to monitor the DL-subnetwork operation in the DLL of the master. The monitoring times are measured either in seconds (s) or in the number of bus cycles.

4.7.1 PDL parameters

4.7.1.1 SPA_acknowledge_timeout $T_{TO\_SPA\_ACK}$

The SPA_acknowledge_timeout is the time which the local PDL waits for the associated PDL_Data_Ack.confirm primitive after the sending of a PDL_Data_Ack.request primitive. If

there is a timeout, the local PDL shall attempt up to *max_spa_retry*-times to send the DLSDU to the remote PDL. If no attempt was successful, the PDL shall return a negative acknowledgement to the user. In addition, this communication relationship shall be locally disconnected and the PDL shall attempt to synchronize itself again with the corresponding PDL of the communication partner.

The SPA_acknowledge_timeout can be calculated as follows:

$t_{TO\_SPA\_ACK}$ = (DIST + add_wait) * t_{UP}

where

DIST is a constant number of 5 bus cycles;

add_wait is an additional redundancy of 1 to 4 bus cycles;

t_{UP} is the update time.

4.7.1.2 max_spa_retry

This DLL parameter specifies the maximum number of repeated attempts to send SPA PDUs. It can be parameterized by means of a PNM2_Set_Value.request primitive.
Value range : 0, 2, 4, 6 …14

4.7.1.3 max_swa_count

This DLL parameter specifies the maximum number of successive data cycles with errors which are allowed before the PDL protocol machine reports a multiple data cycle error and carries out a synchronization with the protocol machine of the remote device.
Value range : 0 … 255

4.7.2 BLL parameters

4.7.2.1 update_time t_{UP}

The update time is the time which passes between two starts of bus cycles. By setting the update time with a PNM2_Set_Value.request primitive the time-equidistance of the DL-subnetwork can be obtained. The update time shall be greater or equal to the bus cycle time (except zero) and is preset by the system to the value zero (default).

The value zero means that the update time is not defined. Thus, the automatic start of a bus cycle merely depends on the end of the previous bus cycle and the complete processing of the PDL protocol machines and not of the timeout of the update timer. That means, the time aquidistance is deactivated by the default setting.

Parameter size: 4 octets

Settable values: $t_{UP} \times 0{,}1$ ms

4.7.2.2 bus_timeout t_{TO_BUS}

The bus timeout is the maximum time which may pass between two valid data cycles. If this time is exceeded, there is a fatal bus error, which could not be repaired independently (for example, environment with strongly interference or broken cable). The bus timeout can be parameterized with a PNM2_Set_Value.request primitive. If the bus timeout is set to the value zero, the bus monitoring is disabled.

Parameter size: 4 octets

Settable values: $t_{TO\_BUS} \times 1$ ms

4.7.3 MAC parameters

4.7.3.1 Device code

Figure 81 shows the structure of the device code:

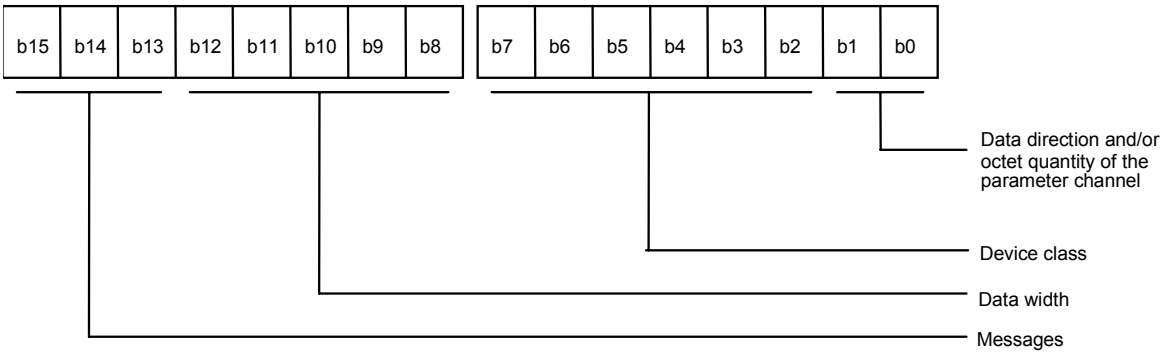


Figure 81 – Device code structure

Bits 0 and 1 have to be interpreted differently for devices with or without parameter channel. For devices without parameter channel the bits indicate the direction of the user data. For devices with parameter channel the bits indicate the number of octets which are used for the parameter channel.

Bits 6 and 7 of the device code distinguish whether the device has a parameter channel or not. For devices with a parameter channel the bits 6 and 7 shall have only the value combination Bit6 =1 and Bit7 =1.

4.7.3.2 Data direction (bit 0 and bit 1 ≠ 1)

If bits 6 and 7 ≠ 1, the bits indicate whether the device occupies input and/or output addresses (see Table 56).

Table 56 – Data direction

Bit 1	Bit 0	Meaning
0	0	No data address (for example, bus coupler)
0	1	Only output addresses occupied
1	0	Only input addresses occupied
1	1	Input and output addresses occupied

4.7.3.3 Number of octets occupied in the parameter channel (bit 7 = 1 and bit 6 = 1)

If bits 7 and 6 are both 1, bits 1 and 0 indicate how many octets of the parameter channel the device occupies (see Table 57).

Table 57 – Number of the occupied octets in the parameter channel

Bit 1	Bit 0	Number of occupied words of the parameter channel
0	0	4 octets
0	1	8 octets
1	0	Reserved
1	1	2 octets (standard)

4.7.3.4 Device class

Certain bit combinations of the bits 2 through 7 indicate the device class (see Table 58). The other combinations are reserved for the identification of device functions. These specifications should be described in device profiles.

Table 58 – Device class

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Device class
0	0	0	0	1	0	0	0	Bus coupler with local bus branch
0	0	0	0	1	1	0	0	Bus coupler with remote bus branch
0	0	0	0	1	0	1	1	Bus coupler with I/O data
0	1	1	1	1	1	x	x	Analog local bus device
1	0	1	1	1	1	x	x	Digital local bus device
1	1	0	1	1	1	x	x	Local bus device with parameter channel
0	0	0	0	0	0	x	x	Digital remote bus device
0	0	1	1	0	0	x	x	Analog remote bus device
1	1	1	1	0	0	x	x	Remote bus device with parameter channel

where
x "don't care".

4.7.3.5 Control data

Bits 13 to 15 return control data from the device to the master (see Table 59).

Table 59 – Control data

Bit 15	Bit 14	Bit 13	Meaning
x	x	1	Reserved
x	1	X	CRC receive error
1	x	X	Reserved

where
x "don't care".

4.7.3.6 Data width

The data width specifies how many bits the device occupies on the bus. If a device has, for example, 16 bit inputs and 32 bit outputs, it occupies 32 bit (4 octets) in the ring (the higher value is decisive) (see Figure 82 and Table 60).

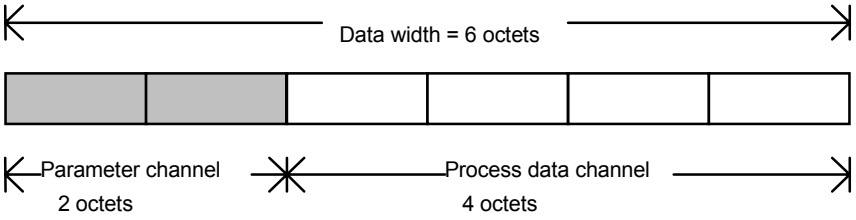


Figure 82 – Relations between data width, process data channel and parameter channel

Table 60 – Data width

Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Data width
0	0	0	0	0	0
0	1	1	0	0	1 bit
0	1	1	0	1	2 bits
0	1	0	0	0	4 bits
0	1	0	0	1	1 octet
0	1	0	1	0	12 bits
0	0	0	0	1	2 octets
0	1	0	1	1	3 octets
0	0	0	1	0	4 octets
0	0	0	1	1	6 octets
0	0	1	0	0	8 octets
0	0	1	0	1	10 octets
0	1	1	1	0	12 octets
0	1	1	1	1	14 octets
0	0	1	1	0	16 octets
0	0	1	1	1	18 octets
1	0	1	0	1	20 octets
1	0	1	1	0	24 octets
1	0	1	1	1	28 octets
1	0	0	1	0	32 octets
1	0	0	1	1	48 octets
1	0	0	0	1	52 octets
1	0	1	0	0	64 octets
1	0	0	0	0	Reserved
1	1	x	x	x	Reserved

where
x = "don't care".

4.7.3.7 Control code

Figure 83 shows the structure of the control code:

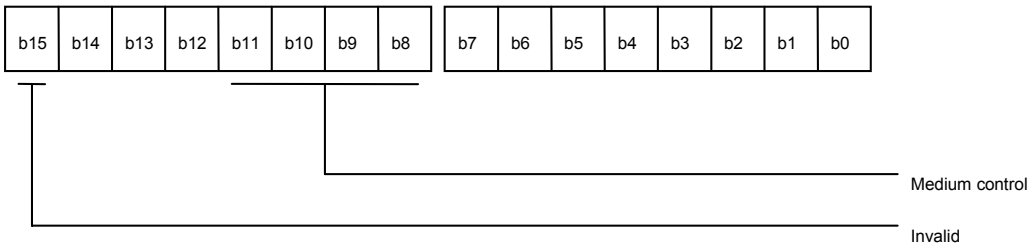


Figure 83 – Structure of the control code

4.7.3.8 Invalid

Bit 15 defines whether the control code is effective. If bit 15 equals 0, the code is effective.

4.7.3.9 Medium control (Bit 8 to Bit 11)

Bits 8 to 11 control the MAU of the outgoing interfaces (see Table 61).

Table 61 – Medium control

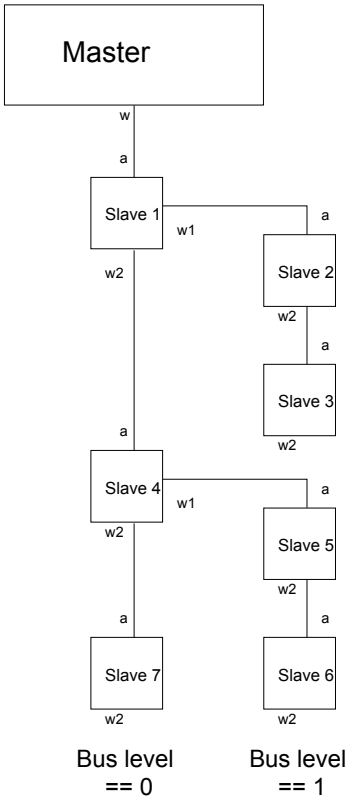
Bit 11	Bit 10	Bit 9	Bit 8	Meaning
X	x	X	1	Reset of the ring segment which is connected to the outgoing interface 1
X	x	1	X	Reset of the ring segment which is connected to the outgoing interface 2
X	1	X	X	Outgoing interface 1 disabled
1	x	X	x	Outgoing interface 2 disabled
where x "don't care".				
NOTE The remaining bits of the control code are reserved.				

Annex A
(informative)
–
Implementation possibilities of definite PNM2 functions

A.1 Acquiring the current configuration

A.1.1 Configuration data memory in the master

The configuration data is stored with slave information code as an image of the ring in the sequence of the position of the slaves in the ring. The ring segment level contains the level of the slave if the bus configuration is depicted as a tree structure. Figure A.1 shows a DL-subnetwork configuration in the form of a tree structure and the ring segment level of the individual branches. The positions of the slaves in the ring correspond to their numbering:



Explanations:

w outgoing interface of the master
a incoming interface of a slave
w1 first outgoing interface of a slave
w2 second outgoing interface of a slave

The ring segment level is increased when the devices are connected to w1.

Figure A.1 – DL-subnetwork configuration in the form of a tree structure

The DL-subnetwork configurations are stored in the master as matrices (see Table A.1):

Table A.1 – DL-subnetwork configuration in the form of a matrix

Device code	Ring segment level	No. of the slave
…	…	1
…	…	2
…	…	3
…	…	4
…	…	…
…	…	n

A.1.2 Acquire_Configuration function

The function is described with the Acquire_Configuration.request and Acquire_ Configuration.confirm primitives. The Acquire_Configuration.request primitive does not contain any parameters, the Acquire_Configuration.confirm primitive contains the configuration to be acquired with a result (+) or the error_code with a result (-) (see Table A.2).

Table A.2 – Acquire_Configuration

Parameter name	Request	Confirm
Argument	M	
Result (+)		S
current_configuration		M
Result (-)		S
error_code		M

result (+):

A current configuration could be acquired.

current_configuration:

This parameter contains the currently acquired configuration in the form of the device codes and the ring segment level as matrix.

result (-):

No configuration could be acquired.

error_code:

The error_code describes the error cause. Possible errors are:

— Too many cycles with errors when the configuration was acquired.

A.1.3 State machine for the acquisition of the configuration

A.1.3.1 General

The current configuration is acquired by connecting the outgoing interfaces of the slaves step by step. In order to get an algorithm that is as fast as possible and has short bus cycle times, the interfaces are closed again when the end of a branch is reached.

As the procedure is repeated for every outgoing interface, the acquisition of the configuration can be described by a recursion. The Acquire_Configuration.request primitive is called again for every outgoing interface. Thus, the state machine shown in Figure A.2 applies to every call of the Acquire_Configuration.request primitive:

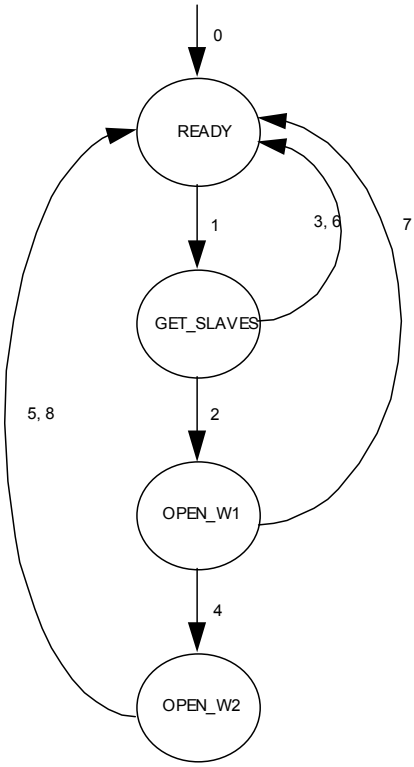


Figure A.2 – State machine for the acquisition of the current configuration

A.1.3.2 States of the state machine

READY

The state machine is ready to respond to an Acquire_Configuration.request primitive.

GET_SLAVES

The state machine runs ID cycles without connecting the outgoing interfaces to identify the slaves which are already in the ring.

OPEN_W1

The state machine opens the outgoing interface W1 of the last identified slave and acquires the configuration at W1 by means of an Acquire_Configuration.request primitive.

OPEN_W2

After the configuration has been acquired the state machine closes the outgoing interface W1, opens the outgoing interface W2 of the last identified slave and acquires the configuration at W2 by means of an Acquire_Configuration.request primitive.

Table A.3 describes the state transitions.

Table A.3 – State transitions of the state machine for the acquisition of the current configuration

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
Power on	0	READY
READY Acquire_Configuration.request ⇒ run ID cycles without opening or closing the outgoing interfaces, to identify slaves which already exist in the ring.	1	GET_SLAVES
GET_SLAVES ID cycles ended \at least one new slave could be detected ⇒ open W1, Acquire_Configuration.request	2	OPEN_W1
GET_SLAVES ID cycles completed \no new slave could be detected ⇒ Acquire_Configuration.confirm (+)	3	READY
OPEN_W1 Acquire_Configuration.confirm (+) ⇒ close W1, open W2, Acquire_Configuration.request	4	OPEN_W2
OPEN_W2 Acquire_Configuration.confirm (+) ⇒ close W2, Acquire_Configuration.confirm (+)	5	READY
GET_SLAVES, OPEN_W1 or OPEN_W2 Acquire_Configuration.confirm (-) or several ID cycles could no be completed without errors ⇒ Acquire_Configuration.confirm (-)	6 - 8	READY

A.2 Comparing the acquired and stored configurations prior to a DL-subnetwork error

A.2.1 General

After the current configuration has been acquired, the data of two configurations is stored in the master: the stored configuration, and the currently acquired configuration. Then these two configurations are compared. The example stops the configuration after the first error has been detected. The comparison may provide the following results.

— No error has been detected, that is, the two configurations are identical.

— No current configuration could be detected, that is, the second configuration list is empty.

— The configuration became longer.

— The configuration became shorter.

— A ring segment became longer.

— A ring segment became shorter.

— At a certain DL-subnetwork position there is a slave with another device code.

A comparison is carried out by the Check_Configuration function.

A.2.2 Check_Configuration function

The Check_Configuration function is described with the Check_Configuration.request and Check_Configuration.confirm primitives.

The Check_Configuration.request primitive has no parameters. Besides the result (+ or -), the confirmation contains in the event of an error an error_code and an add_code with the error position in the ring (see Table A.4).

Table A.4 – Check_Configuration

Parameter name	Request	Confirm
Argument	M	
Result (+)		S
Result (-)		S
error_code		M
slave_position		M

result (+):

No error has been detected, that is, the two configurations are identical.

result (-):

An error has been detected, that is, the two configurations are not identical.

error_code:

This parameter describes the type of error. Possible errors are the respective meanings of the slave_position.

— No configuration available; slave_position does not have a meaning.
— The configuration became shorter; slave_position indicates the first missing slave.
— The configuration became larger; slave_position indicates the first additional slave.
— Additional slave to W1 of slave_position.
— The slave with the slave_position number is missing.
— The slave with the slave_position number is a slave with an incorrect device code.

slave_position:

Slave_position contains the error position in the ring.

A.2.3 Compare_Slave function

The Compare_Slave function compares the device code and the ring segment level of two slaves and returns the result of the comparison in the result and error_code parameters (see Table A.5).

Table A.5 – Compare_Slave

Parameter name	Request	Confirm
Argument	M	
saved_data	M	
current_data	M	
Result (+)		S
Result (-)		S
error_code		M

saved_data:

This parameter contains the ID code and the ring segment level of a slave which is stored in the master's configuration.

current_data:

This parameter contains the ID code and the ring segment level of a slave of the currently acquired configuration.

result (+):

The two slaves have the same ID code and ring segment level.

result (-):

The two slaves have different ID codes and/or ring segment levels.

error_code:

Error_code describes the type of distinction. The following is possible.

— The ring segment level of the slave in the currently acquired configuration is higher.

— The ring segment level is lower.

— The ID codes are different.

A ring segment level which is too high or too low has a higher priority than an invalid ID code. Thus, error_code gives no information on the ID code when the ring segment level is incorrect. However, if a wrong ID code is reported, the ring segment level is definitely okay.

A.2.4 State Machine for Comparing the Configuration Data

A.2.4.1 General

Figure A.3 shows the state machine for comparing two configurations.

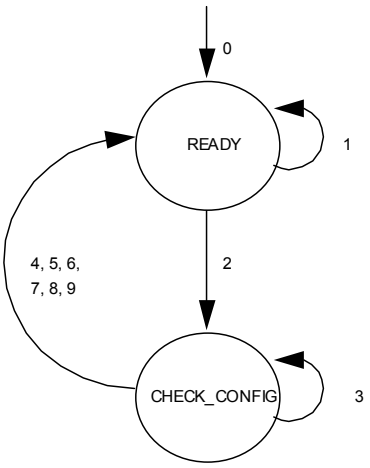


Figure A.3 – State machine for comparing two configurations

A.2.4.2 States of the state machine

READY

The state machine is ready to execute the Check_Config function.

CHECK_CONFIG

The configurations are being compared by means of the Check_Config function.

Table A.6 describes the state transitions.

Table A.6 – State transitions of the state machine for comparing two configurations

Initial state **event** **\condition** **⇒ action**	**Transition**	**Follow-up state**
Power_On	**0**	**READY**
READY Check_Config.request \quantity of current identified slaves == 0 ⇒ Check_Config.confirm (-) with error_code = 'no configuration available'	**1**	**READY**
READY Check_Config.request \quantity of identified slaves != 0 ⇒ slave_no = 1, m = slave quantity in current configuration, n = slave quantity in stored configuration, Compare_Slave.request(saved_config.[slave_no], current_config.[slave_no])	**2**	**CHECK_CONFIG**
CHECK_CONFIG Compare_Slave.confirm (+) \slave_no < n AND slave_no < m ⇒ slave_no++, Compare_Slave.request(saved_config.[slave_no], current_config.[slave_no])	**3**	**CHECK_CONFIG**
CHECK_CONFIG Compare_Slave.confirm (+) \slave_no == n AND slave_no == m ⇒ Check_Config.confirm (+)	**4**	**READY**
CHECK_CONFIG Compare_Slave.confirm (+) \slave_no < n AND slave_no == m ⇒ Check_Config.confirm (-) with error_code = 'the configuration became shorter' and slave_position = slave_no + 1	**5**	**READY**
CHECK_CONFIG Compare_Slave.confirm (+) \slave_no == n AND slave_no < m ⇒ Check_Config.confirm (-) with error_code = 'the configuration became longer' and slave_position = slave_no + 1	**6**	**READY**
CHECK_CONFIG Compare_Slave.confirm (-) \error_code == 'ring segment level higher than expected' ⇒ Check_Config.confirm (-) with error_code = 'additional slave at W1 of slave_-position' and slave_position = slave_no - 1	**7**	**READY**
CHECK_CONFIG Compare_Slave.confirm (-) \error_code == 'ring segment level lower than expected' ⇒ Check_Config.confirm (-) with error_code = 'slave missing' and slave_position = slave_no	**8**	**READY**
CHECK_CONFIG Compare_Slave.confirm (-) \error_code == 'different ID codes' ⇒ Check_Config.confirm (-) with error_code = 'wrong slave' and slave_position = slave_no	**9**	**READY**

A.2.4.3 State Machine for Comparing One Line of Two Configuration Matrices

Figure A.4 shows the state machine for comparing one line of two configuration matrices.

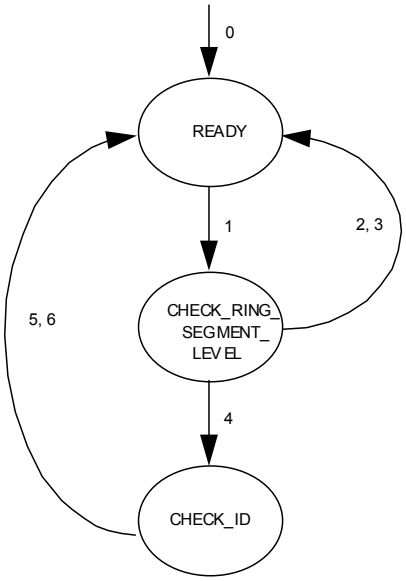


Figure A.4 – State machine for comparing one line of two configuration matrices

A.2.4.4 States of the state machine

READY

The state machine is ready to execute the Compare_Slave function.

CHECK_RING_SEGMENT_LEVEL

The Compare_Slave function was called. The ring segment levels are compared.

CHECK_ID

After the comparison of the ring segment levels the ID codes are compared as well. This comparison only takes place when the ring segment levels are identical.

Table A.7 describes the state transitions.

Table A.7 – State transitions of the state machine for comparing one line of two configuration matrixes

Initial state event \condition ⇒ action	**Transition**	**Follow-up state**
Power on	0	READY
READY Compare_Slave.request ⇒ compare ring segment levels	1	CHECK_RING_ SEGMENT_LEVEL
CHECK_RING_SEGMENT_LEVEL ring segment level higher than expected ⇒ Compare_Slave.confirm (-) with error_code = 'ring segment level higher than expected'	2	READY
CHECK_RING_SEGMENT_LEVEL ring segment level lower than expected ⇒ Compare_Slave.confirm (-) with error_code = 'ring segment level lower than expected'	3	READY
CHECK_RING_SEGMENT_LEVEL ring segment level are not identical ⇒ compare the ID codes	4	CHECK_ID
CHECK_ID ID codes are identical ⇒ Compare_Slave (+)	5	READY
CHECK_ID ID codes are not identical ⇒ Compare_Slave (-) with error_code = 'different ID codes'	6	READY

Bibliography

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 61158-5-8, *Industrial communication networks – Fieldbus specifications – Part 5-8: Application layer service definition – Type 8 elements*

IEC 61158-6-8, *Industrial communication networks – Fieldbus specifications – Part 6-8: Application layer protocol specification – Type 8 elements*

IEC 61784-1(Ed.2.0), *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

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

EN 50254, *High efficiency communication subsystem for small data packages*

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
P.O. Box 131
CH-1211 Geneva 20
Switzerland

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