BS EN 61158-4-3:2014

BSI Standards Publication

Industrial communication networks — Fieldbus specifications

Part 4-3: Data-link layer protocol specification — Type 3 elements

bsi.

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 61158-4-3:2014. It is identical to IEC 61158-4-3:2014. It supersedes BS EN 61158-4-3:2012 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications: process measurement and control, including fieldbus.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Published by BSI Standards Limited 2014

ISBN 978 0 580 79441 4

ICS 25.040.40; 35.100.20; 35.110

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 November 2014.

Amendments/corrigenda issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61158-4-3

October 2014

ICS 25.040.40; 35.100.20; 35.110

Supersedes EN 61158-4-3:2012

English Version

Industrial communication networks - Fieldbus specifications - Part 4-3: Data-link layer protocol specification - Type 3 elements (IEC 61158-4-3:2014)

Réseaux de communication industriels - Spécifications des bus de terrain - Partie 4-3: Spécification du protocole de la couche liaison de données - Éléments de type 3 (CEI 61158-4-3:2014)

Industrielle Kommunikationsnetze - Feldbusse - Teil 4-3: Protokollspezifikation des Data Link Layer (Sicherungsschicht) - Typ 3-Elemente (IEC 61158-4-3:2014)

This European Standard was approved by CENELEC on 2014-09-19. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Ref. No. EN 61158-4-3:2014 E

Foreword

The text of document 65C/762/FDIS, future edition 3 of IEC 61158-4-3, prepared by SC 65C “Industrial networks” of IEC/TC 65 “Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 61158-4-3:2014.

The following dates are fixed:

• latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2015-06-19
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2017-09-19

This document supersedes EN 61158-4-3:2012

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

This document has been prepared under a mandate given to CENELEC by the European Commission and the European Free Trade Association.

Endorsement notice

The text of the International Standard IEC 61158-4-3:2014 was approved by CENELEC as a European Standard without any modification.

In the official version, for bibliography, the following notes have to be added for the standards indicated:

IEC 60870-5-1	NOTE	Harmonised as EN 60870-5-1
IEC 61158-1	NOTE	Harmonised as EN 61158-1
IEC 61158-5-3	NOTE	Harmonised as EN 61158-5-3
IEC 61158-6-3	NOTE	Harmonised as EN 61158-6-3
IEC 61784-1	NOTE	Harmonised as EN 61784-1
IEC 61784-2	NOTE	Harmonised as EN 61784-2

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE 1 When an International Publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

NOTE 2 Up-to-date information on the latest versions of the European Standards listed in this annex is available here: www.cenelec.eu.

Publication	Year	Title	EN/HD	Year
IEC 61131-3	-	Programmable controllers Part 3: Programming languages	EN 61131-3	-
IEC 61158-2	2014	Industrial communication networks - Fieldbus specifications Part 2: Physical layer specification and service definition	EN 61158-2	2014
IEC 61158-3-3	-	Industrial communication networks - Fieldbus specifications Part 3-3: Data-link layer service definition - Type 3 elements	EN 61158-3-3	-
ISO/IEC 2022	-	Information technology - Character code structure and extension techniques	-	-
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	-	-
ISO 1177	-	Information processing - Character structure for start/stop and synchronous character-oriented transmission	-	-

CONTENTS

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 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.

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 its profile parts. Use of the various protocol types in other combinations may require permission from their respective intellectual-property-right holders.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent concerning Type 3 elements and possibly other types given in the normative elements of this standard.

The following patent rights for Type 3 have been announced by [SI]:

Publication	Title
EP 1253494	Control device with fieldbus

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

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

[SI]: Siemens AG
CT IP M&A
Otto-Hahn-Ring 6
D-81739 Munich
Germany

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 4-3: Data-link layer protocol specification – Type 3 elements

1 Scope

1.1 General

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

This protocol provides communication opportunities to a pre-selected “master” subset of data-link entities in a cyclic asynchronous manner, sequentially to each of those data-link entities. Other data-link entities communicate only as permitted and delegated by those master data-link entities.

For a given master, its communications with other data-link entities can be cyclic, or acyclic with prioritized access, or a combination of the two.

This protocol provides a means of sharing the available communication resources in a fair manner. There are provisions for time synchronization and for isochronous operation.

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 documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

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

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-3-3, *Industrial communication networks – Fieldbus specifications – Part 3-3: Data link service definition – Type 3 elements*

ISO/IEC 2022, *Information technology – Character code structure and extension techniques*

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*

ISO 1177, *Information processing – Character structure for start/stop and synchronous character oriented transmission*

3 Terms, definitions, symbols and abbreviations

For the purposes of this document, 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 called-DL-address [ISO/IEC 7498-3]

3.1.2 calling-DL-address [ISO/IEC 7498-3]

3.1.3 centralized multi-end-point-connection [ISO/IEC 7498-3]

3.1.4 correspondent (N)-entities
correspondent DL-entities (N=2)
correspondent Ph-entities (N=1) [ISO/IEC 7498-3]

3.1.5 demultiplexing [ISO/IEC 7498-3]

3.1.6 DL-address [ISO/IEC 7498-3]

3.1.7 DL-address-mapping [ISO/IEC 7498-3]

3.1.8 DL-connection [ISO/IEC 7498-3]

3.1.9 DL-connection-end-point [ISO/IEC 7498-3]

3.1.10 DL-connection-end-point-identifier [ISO/IEC 7498-3]

3.1.11 DL-connection-mode transmission [ISO/IEC 7498-3]

3.1.12 DL-connectionless-mode transmission [ISO/IEC 7498-3]

3.1.13 DL-data-sink [ISO/IEC 7498-3]

3.1.14 DL-data-source [ISO/IEC 7498-3]

3.1.15 DL-duplex-transmission [ISO/IEC 7498-3]

3.1.16 DL-facility [ISO/IEC 7498-3]

3.1.17 DL-local-view [ISO/IEC 7498-3]

3.1.18 DL-name [ISO/IEC 7498-3]

3.1.19 DL-protocol [ISO/IEC 7498-3]

3.1.20 DL-protocol-connection-identifier [ISO/IEC 7498-3]

3.1.21 DL-protocol-control-information [ISO/IEC 7498-3]

3.1.22 DL-protocol-data-unit [ISO/IEC 7498-3]

3.1.23 DL-protocol-version-identifier [ISO/IEC 7498-3]

3.1.24 DL-relay [ISO/IEC 7498-3]

3.1.25 DL-service-connection-identifier [ISO/IEC 7498-3]

3.1.26 DL-service-data-unit [ISO/IEC 7498-3]

3.1.27 DL-simplex-transmission [ISO/IEC 7498-3]

3.1.28 DL-subsystem [ISO/IEC 7498-3]

3.1.29 DL-user-data [ISO/IEC 7498-3]

3.1.30 flow control [ISO/IEC 7498-3]

3.1.31 layer-management [ISO/IEC 7498-3]

3.1.32 multiplexing [ISO/IEC 7498-3]

3.1.33 naming-(addressing)-authority [ISO/IEC 7498-3]

3.1.34 naming-(addressing)-domain [ISO/IEC 7498-3]

3.1.35 naming-(addressing)-subdomain [ISO/IEC 7498-3]

3.1.36 (N)-entity [ISO/IEC 7498-3]
DL-entity
Ph-entity

3.1.37 (N)-interface-data-unit [ISO/IEC 7498-3]
DL-service-data-unit (N=2)
Ph-interface-data-unit (N=1)

3.1.38 (N)-layer [ISO/IEC 7498-3]
DL-layer (N=2)
Ph-layer (N=1)

3.1.39 (N)-service [ISO/IEC 7498-3]
DL-service (N=2)
Ph-service (N=1)

3.1.40 (N)-service-access-point [ISO/IEC 7498-3]
DL-service-access-point (N=2)
Ph-service-access-point (N=1)

3.1.41 (N)-service-access-point-address [ISO/IEC 7498-3]
DL-service-access-point-address (N=2)
Ph-service-access-point-address (N=1)

3.1.42 peer-entities [ISO/IEC 7498-3]

3.1.43 Ph-interface-control-information [ISO/IEC 7498-3]

3.1.44 Ph-interface-data [ISO/IEC 7498-3]

3.1.45 primitive name [ISO/IEC 7498-3]

3.1.46 reassembling [ISO/IEC 7498-3]

3.1.47 recombining [ISO/IEC 7498-3]

3.1.48 reset [ISO/IEC 7498-3]

3.1.49 responding-DL-address [ISO/IEC 7498-3]

3.1.50 routing [ISO/IEC 7498-3]

3.1.51 segmenting [ISO/IEC 7498-3]

3.1.52 sequencing [ISO/IEC 7498-3]

3.1.53 splitting [ISO/IEC 7498-3]

3.1.54 synonymous name [ISO/IEC 7498-3]

3.1.55 systems-management [ISO/IEC 7498-3]

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 acceptor

3.2.2 asymmetrical service

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

3.2.4 deliver (primitive)

3.2.5 DL-confirmed-facility

3.2.6 DL-facility

3.2.7 DL-local-view

3.2.8 DL-mandatory-facility

3.2.9 DL-non-confirmed-facility

3.2.10 DL-provider-initiated-facility

3.2.11 DL-provider-optional-facility

3.2.12 DL-service-primitive;
primitive

3.2.13 DL-service-provider

3.2.14 DL-service-user

3.2.15 DL-user-optional-facility

3.2.16 indication (primitive)
acceptor.deliver (primitive)

3.2.17 multi-peer

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

3.2.19 requestor

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

3.2.21 submit (primitive)

3.2.22 symmetrical service

3.3 Common terms and definitions

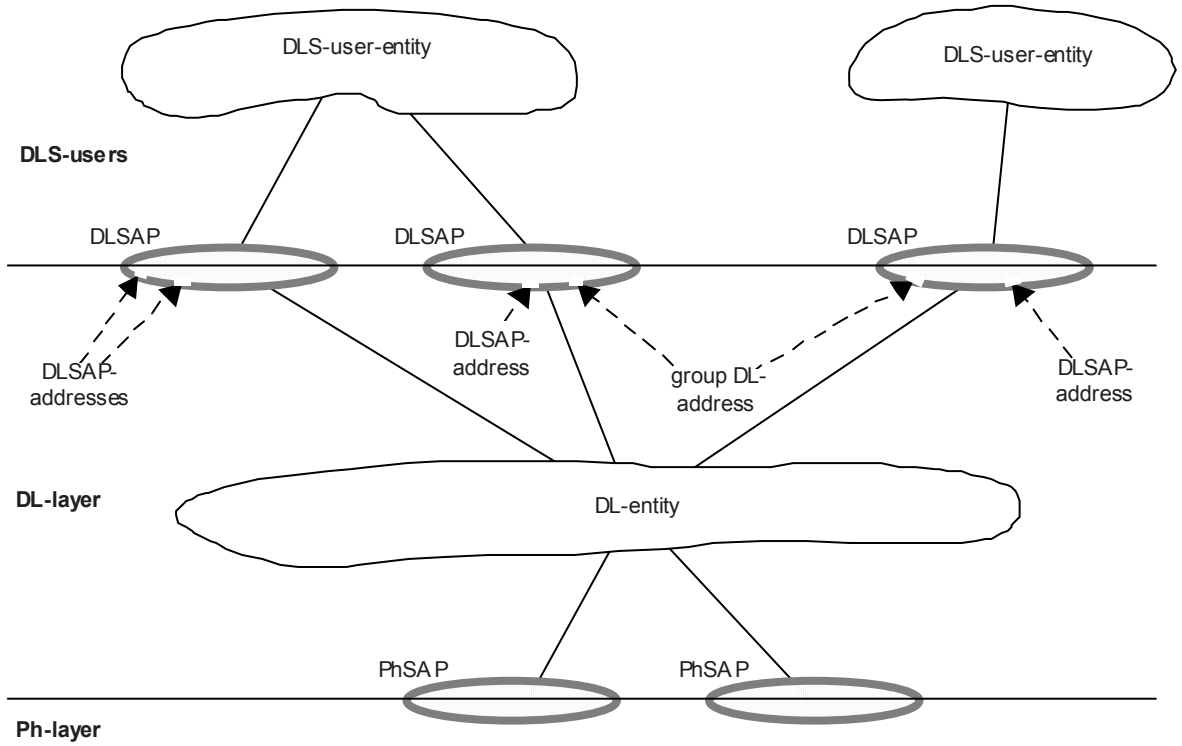
For the purposes of this document, the following terms and definitions apply.

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

3.3.1
DL-segment,
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 1 to entry: 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 1 to entry: 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
(individual) DLSAP-address
DL-address that designates only one DLSAP within the extended link

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

3.3.5
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 1 to entry: An extended link may be composed of just a single link.

3.3.6
frame
denigrated synonym for DLPDU

3.3.7
group DL-address
DL-address that potentially designates more than one DLSAP within the extended link

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

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

3.3.8
node
single DL-entity as it appears on one local link

3.3.9
receiving DLS-user
DL-service user that acts as a recipient of DL-user-data

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

3.3.10
sending DLS-user
DL-service user that acts as a source of DL-user-data

3.4 Additional Type 3 definitions

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

3.4.1
acknowledge DLPDU
reply DLPDU that contains no DLSDU

3.4.2
address extension
DLSAP address or region/segment address

3.4.3
bit time
time to transmit one bit

3.4.4
confirmed message exchange
complete data transfer with request and acknowledgement or response DLPDU

3.4.5
controller_type
hardware class of the communications entity

3.4.6
current master
token holder

3.4.7
data DLPDU
DLPDU that carries a DLSDU from a local DLS-user to a remote DLS-user

3.4.8
DL_status
status that specifies the result of the execution of the associated request

3.4.9
GAP
range of station (DLE) DL-addresses from this station (TS) to its successor (NS) in the logical token ring, excluding stations above HSA

3.4.10
GAP maintenance
registration of new Master and slave stations

3.4.11
isochronous mode
special operational mode that implies both a constant (isochronous) cycle with a fixed schedule of high and low priority messages, and the synchronization of the DLS-users with this constant (isochronous) cycle

3.4.12
local DLS-user
DLS-user that initiates the current service

3.4.13
message exchange
complete confirmed or unconfirmed data transfer

3.4.14
region/segment address
address extension that identifies a particular fieldbus subnetwork

Note 1 to entry: This supports DL-routing between fieldbuses.

3.4.15
request data
DLSDU provided by the remote DLS-user to the local DLS-user

3.4.16
remote DLE
addressed DLE of a service request (that is, the intended receiving DLE of any resulting send/request DLPDU)

3.4.17
remote DLS-user
addressed DLS-user of a service request (that is, the intended receiver of any resulting indication primitive)

3.4.18
reply DLPDU
DLPDU transmitted from a remote DLE to the initiating (local) DLE, and possibly other DLEs

Note 1 to entry: When the remote DLE is a Publisher, the reply DLPDU also can be sent to several remote DLEs.

3.4.19
response DLPDU
reply DLPDU that carries a DLSDU from a remote DLS-user to local DLS-user

3.4.20
send data
DLSDU provided by a local DLS-user to a remote DLS-user

3.4.21
send/request DLPDU
DLPDU that carries either a request for data or a DLSDU or both from a local DLS-user to a remote DLS-user

3.4.22
time master
device which is able to send clock synchronization DLPDUs

Note 1 to entry: Link devices have time master functionality.

3.4.23
time receiver
device which is able to be time synchronized by a time Master

3.4.24
token holder
Master station that controls bus access

3.4.25
token passing
medium access method, in which the right to transmit is passed from master station to master station in a logical ring

3.5 Common symbols and abbreviations

3.5.1 Data units

3.5.1.1	**DLPDU**	DL-protocol data unit
3.5.1.2	**DLSDU**	DL-service data unit
3.5.1.3	**PhIDU**	Ph-interface data unit
3.5.1.4	**PhPDU**	Ph-protocol data unit

3.5.2 Miscellaneous

3.5.2.1	**DL-**	data link layer (as a prefix)
3.5.2.2	**DLCEP**	DL-connection endpoint
3.5.2.3	**DLE**	DL-entity (the local active instance of the Data Link layer)
3.5.2.4	**DLL**	DL-layer

3.5.2.5	**DLM-**	DL-management (as a prefix)
3.5.2.6	**DLMS**	DL-management-service
3.5.2.7	**DLS**	DL-service
3.5.2.8	**DLSAP**	DL-service access point
3.5.2.9	**FIFO**	first-in first-out (queuing method)
3.5.2.10	**LLC**	logical link control
3.5.2.11	**MAC**	medium access control
3.5.2.12	**OSI**	open systems interconnection
3.5.2.13	**Ph-**	physical layer (as a prefix)
3.5.2.14	**PhE**	Ph-entity (the local active instance of the Physical layer)
3.5.2.15	**PhL**	Ph-layer
3.5.2.16	**PhS**	Ph-service
3.5.2.17	**PhSAP**	Ph-service access point
3.5.2.18	**QoS**	quality of service

3.6 Type 3 symbols and abbreviations

3.6.1	**ACK**	acknowledge(ment) DLPDU
3.6.2	**ASM**	active spare time message
3.6.3	**ASP**	active spare time period
3.6.4	**Bus ID**	bus identification, an address extension (region/DL-segment address) that identifies a particular bus as supporting routing between DL-segments
3.6.5	**CRX**	character receive execution
3.6.6	**CS**	clock synchronization
3.6.7	**CTX**	character transmit execution
3.6.8	**DA**	destination address of a DLPDU
3.6.9	**DAE**	destination address extension(s) of a DLPDU, which convey D_SAP_index and/or destination bus ID
3.6.10	**D_SAP**	destination service access point, the DLSAP associated with the remote DLS-user
3.6.11	**D_SAP_index**	destination service access point index – that component of a DLSAP address which designates a DLSAP and remote DLS-user within the remote DLE
3.6.12	**DXM**	data exchange multicast
3.6.13	**ED**	end delimiter of a DLPDU
3.6.14	**EOA**	END-OF-ACTIVITY

3.6.15	**EOD**	END-OF-DATA
3.6.16	**EODA**	END-OF-DATA-AND-ACTIVITY
3.6.17	**EXT**	address extension bit of a DLPDU
3.6.18	**FC**	frame control (frame type) field of a DLPDU
3.6.19	**FCB**	frame count bit of a DLPDU (FC field) used to eliminate lost or duplicated DLPDUs
3.6.20	**FCV**	frame count bit valid bit of a DLPDU, indicates whether the FCB is to be evaluated
3.6.21	**FCS**	frame check sequence (synchronous) or frame checksum (asynchronous)
3.6.22	**FLC**	fieldbus link control
3.6.23	**G**	GAP update factor, the number of token rotations between GAP maintenance (update) cycles
3.6.24	**GAPL**	GAP list containing the status of all stations in this station's GAP
3.6.25	**IsoM**	isochronous mode
3.6.26	**Hd**	Hamming distance, a measure of DLPDU integrity, the minimum number of bit errors that can cause acceptance of a spurious DLPDU
3.6.27	**HSA**	highest station address installed (configured) on this fieldbus
3.6.28	**L**	length of the information field, the part of a DLPDU that is checked by the FCS
3.6.29	**LE**	field giving the length of a DLPDU beyond the fixed part
3.6.30	**LEr**	field that repeats the length to increase DLPDU integrity
3.6.31	**LMS**	list of master stations
3.6.32	**LR**	local resource not available or not sufficient (DL/DLM_status of the service primitive)
3.6.33	**LS**	local service not activated at DL-service access point or local DLSAP not activated (DL/DLM_status of the service primitive)
3.6.34	**lsb**	least significant bit of a field or octet
3.6.35	**max**	the arithmetic maximum function
3.6.36	**MCT**	multicast
3.6.37	**MP**	message transfer message retry transfer periods
3.6.38	**mr**	number of retries
3.6.39	**msb**	most significant bit of a field or octet
3.6.40	**MSRD**	DLS: Send and Request Data with Multicast reply
3.6.41	**mt**	number of retries per token rotation
3.6.42	**n**	number of stations

3.6.43	**NA**	no acknowledgement/response (DL/DLM_status of the service primitive)	
3.6.44	**na**	number of active stations	
3.6.45	**NIL**	locally determined value	
3.6.46	**NO**	not ok (DL/DLM_status of the service primitive)	
3.6.47	**np**	number of passive stations	
3.6.48	**NR**	no response DL-data acknowledgement negative and send data ok (DL_status of the service primitive)	
3.6.49	**NRZ**	non-return-to-zero (PhL), an encoding technique where transitions occur only when successive data bits have different values	
3.6.50	**NS**	next station, the station to which this master will pass the token	
3.6.51	**OK**	service finished according to the rules (DL/DLM_status of the service primitive)	
3.6.52	**PhICI**	PhL Interface Control Information	[ISO/IEC 7498-1]
3.6.53	**PhPCI**	PhL Protocol Control Information	[ISO/IEC 7498-1]
3.6.54	**PON**	power on transition occurs at a station	
3.6.55	**PS**	previous station, the station which passes the token to this master station	
3.6.56	**PSP**	passive spare time period	
3.6.57	**RDH**	response DL-data high and no resource for send data (DL/DLM_status of the service primitive)	
3.6.58	**RDL**	response DL/DLM-data low and no resource for send data (DL/DLM_status of the service primitive)	
3.6.59	**Res**	reserved	
3.6.60	**RET**	retry	
3.6.61	**RR**	no resource for send data and no response DL-data available (negative acknowledgement) (DL/DLM_status of the service primitive)	
3.6.62	**RS**	no service, or no service activated at remote DLSAP (negative acknowledgement) (DL/DLM_status of the service primitive)	
3.6.63	**RSYS**	system message rate (in message transfer periods per second) at which confirmed DL-message exchanges are performed	
3.6.64	**SA**	source address of a DLPDU	
3.6.65	**SAE**	source address extension(s) of a DLPDU, which convey S_SAP_index and/or source bus ID	
3.6.66	**SC**	single character acknowledge DLPDU	
3.6.67	**SD1 to SD4**	start delimiters of asynchronous DLPDU transmission	
3.6.68	**SDA**	Send Data with Acknowledge (DL-service)	

3.6.69	**SDA_H/L**	Send Data with Acknowledge high/low (DLPDU Function)
3.6.70	**SDX**	start delimiter of asynchronous or synchronous DLPDU transmission
3.6.71	**SDL1 to SDL5**	start delimiters of synchronous DLPDU transmission
3.6.72	**SDN**	Send Data with No Acknowledge (DL-service)
3.6.73	**SDN_H/L**	Send Data with No Acknowledge high/low (DLPDU Function)
3.6.74	**SM**	state machine
3.6.75	**SOA**	START-OF-ACTIVITY
3.6.76	**SRC**	send receive control
3.6.77	**SRD**	Send and Request Data with Reply (DL-service)
3.6.78	**SRD_H/L**	Send and Request Data with Reply high/low (DLPDU Function)
3.6.79	**SRU**	send receive unit
3.6.80	**S_SAP**	source service access point, the DLSAP associated with the initiating local DLS-user
3.6.81	**S_SAP_index**	source service access point index – a component of a DLSAP-address which designates that DLSAP within the DLE at which the transaction is being initiated
3.6.82	**Stn**	station, a device implementing a DLE with a fieldbus DL-address
3.6.83	**SYN**	synchronizing bits of a DLPDU (period of idle), which guarantees the specified DLPDU integrity and allows for receiver synchronization
3.6.84	**TA/R**	time to transmit an acknowledgement/response DLPDU
3.6.85	**TASM**	ASM message time
3.6.86	**t_{BIT}**	bit time
3.6.87	**TCSI**	clock synchronization interval time
3.6.88	**T_{CT}**	isochronous cycle time
3.6.89	**TGUD**	GAP update time
3.6.90	**T_{ID}**	idle time
3.6.91	**TIM**	timer state machine
3.6.92	**T_{MP}**	message transfer period
3.6.93	**TP**	token transfer period
3.6.94	**TPSP**	passive spare time
3.6.95	**TPTG**	post transmission gap time
3.6.96	**TQUI**	quiet time
3.6.97	**TRCT**	real isochronous cycle time

3.6.98 T_{RD} receive delay time

3.6.99 TRDY ready time

3.6.100 TRES spare time

3.6.101 T_{RR} real rotation time

3.6.102 TS this station

3.6.103 TS/R send/request DLPDU time

3.6.104 T_{SD} send delay time

3.6.105 TSDI station delay of initiator

3.6.106 TSDR station delay of responder

3.6.107 TSET setup time

3.6.108 T_{SH} time shift

3.6.109 T_{SL} slot time

3.6.110 T_{SM} safety margin time

3.6.111 T_{SR} system reaction time

3.6.112 TSYN synchronization time

3.6.113 TSYNI synchronization interval time

3.6.114 T_{TC} token cycle time

3.6.115 T_{TD} transmission delay time

3.6.116 T_{TF} token DLPDU time

3.6.117 T_{TH} token holding time

3.6.118 T_{TO} timeout time

3.6.119 T_{TP} token transfer period

3.6.120 T_{TR} target rotation time

3.6.121 UART universal asynchronous receiver/transmitter

3.6.122 UC UART character

3.6.123 UE negative acknowledgement, remote user interface error (DL/DLM_status of the service primitive)

4 Common DL-protocol elements

4.1 Frame check sequence

4.1.1 General

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

As in other International Standards (see Note 2), DLPDU-level error detection is provided by calculating and appending a multi-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 $n - k$ redundant bits, and by calculating during reception that the message and concatenated FCS form a legal (*n*,*k*) code word. The mechanism for this checking is as follows:

NOTE 2 For example, ISO/IEC 3309 and ISO/IEC 8802.

The generic form of the generator polynomial for this FCS construction is specified in equation (4) and the polynomial for the receiver's expected residue is specified in equation (9). The specific polynomials for DL-protocol type 3 are specified in Table 1. An exemplary implementation is shown in Annex B.

Table 1 – FCS length, polynomials and constants by Type 3 synchronous

Item	Value	
n-k	16	
G(x)	$X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^3 + X^2 + X + 1$	(see Notes 1, 2, 3
R(x)	$X^{15} + X^{14} + X^{13} + X^9 + X^8 + X^7 + X^4 + X^2$	(see Note 4

NOTE 1 Code words D(X) constructed from this G(X) polynomial have Hamming distance 4 for lengths ≤ 344 octets and Hamming distance 5 for lengths ≤ 15 octets.

NOTE 2 This G(X) polynomial is relatively prime to all, and is thus not compromised by any, of the polynomials commonly used in DCEs (modems): the differential encoding polynomial $1 + X^{-1}$ and all primitive scrambling polynomials of the form $1 + X^{-j} + X^{-k}$.

NOTE 3 This G(X) polynomial is the optimal 16-bit polynomial for burst error detection over DLPDUs of 300 octets or less when the statistics of the error burst have a Poisson distribution (as is the usual case).

NOTE 4 The remainder R(x) is 1110 0011 1001 0100 (X^{15} to X^0, respectively) in the absence of errors.

4.1.2 At the sending DLE

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), F(X), and D(X), of dimension *k*, $n - k$, and *n*, respectively, in an extension field over Base Galois Field(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 1 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

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

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

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

$$\begin{aligned} D(X) &= M(X)\, X^{n-k} + F(X) \\ &= m_1 X^{n-1} + m_2 X^{n-2} + \ldots + m_k X^{n-k} + f_{n-k-1} X^{n-k-1} + \ldots + f_0 \\ &= m_1 X^{n-1} + m_2 X^{n-2} + \ldots + m_k X^{16} + f_{15} X^{15} + \ldots + f_0 \end{aligned} \quad (3)$$

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

$$\begin{aligned} G(X) &= X^{n-k} + g_{n-k-1} X^{n-k-1} + \ldots + 1 \\ &= X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^6 + X^3 + X^2 + X + 1 \end{aligned} \quad (4)$$

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

$$\begin{aligned} L(X) &= \frac{X^{n-k} + 1}{X + 1} = X^{n-k-1} + X^{n-k-2} + \ldots + X + 1 \\ &= X^{15} + X^{14} + X^{13} + X^{12} + \ldots + X^2 + X + 1 \end{aligned} \quad (5)$$

That is,

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

NOTE 2 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 3 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.

4.1.3 At the receiving DLE

The octet sequence indicated by the PhE shall be concatenated into the received DLPDU and FCS, and regarded as a vector V(X) of dimension ***u***

$$V(X) = v_1 X^{u-1} + v_2 X^{u-2} + \ldots + v_{u-1} X + v_u \quad (7)$$

NOTE 1 Because of errors **u** can be different than **n**, the dimension of the transmitted code vector.

A remainder R(X) shall be computed for V(X), the received DLPDU and FCS, by a method similar to that used by the sending DLE (see 4.1.2) in computing F(X)

$$\begin{aligned} R(X) &= L(X)\, X^u + V(X)\, X^{n-k} \ (\text{modulo } G(X)) \\ &= r_{n-k-1} X^{n-k-1} + \ldots + r_0 \end{aligned} \quad (8)$$

Define E(X) to be the error code vector of the additive (modulo-2) differences between the transmitted code vector D(X) and the received vector V(X) resulting from errors encountered (in the PhS provider and in bridges) between sending and receiving DLEs.

$$E(X) = D(X) + V(X) \quad (9)$$

If no error has occurred, so that E(X) = 0, then R(X) will equal a non-zero constant remainder polynomial

$$R_{ok}(X) = L(X)\, X^{n-k} \ (\text{modulo } G(X)) \quad (10)$$

whose value is independent of D(X). Unfortunately R(X) will also equal $R_{ok}(X)$ in those cases where E(X) is an exact non-zero multiple of G(X), in which case there are “undetectable” errors. In all other cases, R(X) will not equal $R_{ok}(X)$; such DLPDUs are erroneous and shall be discarded without further analysis.

NOTE 2 As a typical implementation, the initial remainder of the division is preset to all ones. The received bit stream is multiplied by X^{n-k} and divided (modulo 2) by the generator polynomial G(X), specified in equation (4).

5 Overview of the DL-protocol

NOTE Annex A specifies a number of finite state machines used by the DLE to provide its low-level and high-level protocol functions. The specification of Annex A is complementary to the textual specification in this and related clauses in the body of this standard. In case of conflict the requirements of Annex A take precedence.

5.1 General

From the requirements of the various application fields, for example, process control, factory automation, power distribution, building automation, primary process industry etc., the following characteristic features of the fieldbus data-link protocol are resulting (see Table 2).

Table 2 – Characteristic features of the fieldbus data-link protocol

Feature	Description
Station types	Masters (active stations, with bus access control); Slaves (passive stations, without bus access control); preferably at most 32 masters, optionally up to 127, if the applications are not time critical
Station addressing	0 to 127 (127 = global addresses for broad-cast and multicast messages), address extension for region/DL-segment address and service access address (DLSAP), 6 bit each
Bus access	Hybrid: decentralized and central; "token-passing" between master stations and "Master-Slave" between Master and slave stations
Data transfer services	Send Data with/without Acknowledge Send and Request Data with Reply Send and Request Data with Multicast Reply Clock Synchronization
DLPDU length	max. 255 octets per DLPDU, 0 to 246 octets for each data unit without address extension
Transmission speed	Depending on network topology and cable lengths, for example, step-wise from 9,6 to 12000 kBit/s
Transmission characteristic	Half duplex, asynchronous and synchronous transmission
Data integrity asynchronous transmission	Messages with Hamming distance (Hd) = 4, sync slip detection, special sequence to avoid loss or duplication of data
Data integrity synchronous transmission	Hamming distance (Hd) = 4 for DLPDU lengths shorter than 255 octets and Hd = 5 for lengths shorter than 15 octets, special sequence to avoid loss or duplication of data
NOTE 1 The DLPDU format for asynchronous transmission is the format FT 1.2 (asynchronous transmission with start-stop synchronization) for Telecontrol Equipment and Systems, which is specified in IEC 60870-5-1. NOTE 2 The DLPDU format for synchronous transmission is based on octet characters.	

5.2 Overview of the medium access control and transmission protocol

The fieldbus data-link layer uses controlled medium access accomplished by a hybrid medium access method: a decentralized method according to the principle of token-passing is underlain by a central method according to the master-slave principle. Medium access control may be exercised by each master station (active station). The token-passing is characterized by the following features:

a) Token-passing allows fair media access for all token holders.

EXAMPLE When four token holders produce the same amount of similar priority data they will share the media so that on average each of them can use 25 % of the available message transfer time. With the token-passing procedure, rules exist to share the message transfer time between the token holders without discrimination of any of them. In case of usage of the whole of the available token holding time by one token holder in one token rotation

this token holder is limited to one high priority message in the next token rotation after which it has to transfer the token immediately.

b) Token-passing guarantees short reaction time.

EXAMPLE For urgent messages, a maximum delay is specified for delivery of the information. Thus, a configurable time (T_{TR}) specifies the target rotation time. Independent of the number of token holders and the amount of messages that have to be sent, at least one urgent message can be sent by each token holder. Thus, a short reaction time is guaranteed for one urgent message from each token holding station in each token holding period.

c) Token-passing allows flexible (re-)configuration.

EXAMPLE When a token holder is switched on or off, it will be automatically included or excluded in the logical ring. In the next token rotation after detection and inclusion in the logical token ring, the new token holder has the same rights to send messages as the other token holders. The sharing of the bus message transfer time is organized automatically without changing of parameters of the other token holders.

Medium access control may be exercised by each master station (active station) if the station has the token. The token is passed from master station to master station in a logical ring and thus determines the instant, when a master station may access the medium. Controlled token passing is managed by each station knowing its predecessor (previous station, PS), the station from which it receives the token. Furthermore, each station knows its successor (next station, NS), that is, the station to which the token is transmitted, and its own address (This station, TS). Each master station determines the PS and NS addresses after the initialization of the operating parameters for the first time and then later dynamically according to the algorithm described in 5.3.2.4.

The Master-Slave principle is characterized by the following features:

Communication is always initiated by a master station which has the permission for medium access, the token. slave stations (passive stations) act neutrally in respect to medium access, that is, they do not transmit independently but only on request. If the logical ring consists of only one Master and several slave stations then it is a pure Master-Slave system.

The following error conditions, exceptions and operational states in the system are dealt with:

– multiple tokens,
– lost token,
– error in token passing,
– duplicate station addresses,
– stations with faulty transmitter or receiver,
– adding and removing stations during operation,
– any combinations of master and slave stations.

5.3 Transmission modes and DL-entity

5.3.1 Overview

The exchange of messages takes place confirmed or unconfirmed. A confirmed message exchange consists of a master station's send/request DLPDU and the associated acknowledgement or response DLPDU of a Master or a slave station. User data may be transmitted in the send/request DLPDU as well as in the response DLPDU. The acknowledgement DLPDU does not contain any user data (for DLPDU formats see Clause 7).

An unconfirmed message exchange takes place only in case of token transmission and in case of the transmission of data without acknowledgement (for example, necessary for broadcast messages). In both modes of operation, there is no acknowledgement. In broadcast messages a master station (initiator) addresses all other stations at the same time by means of a global address (highest station address, all address bits are binary "1").

All stations except the respective token holder (initiator) shall in general monitor all requests. The stations acknowledge or respond only when they are addressed. The acknowledgement or response shall arrive within a predefined time, the slot time, otherwise the initiator repeats, depending on the predefined retry limit, the request if it is not a "first request" (see 6.4, FCB). A new request or a token shall not be issued by the initiator before the expiration of a waiting period after receiving an acknowledge or response, the idle time (see 5.5).

If the responder does not acknowledge or respond after a predefined number of retries (see Table 5), it is marked as "non-operational". If a responder is "non-operational", a later unsuccessful request will not be repeated.

The modes of transmission operation define the sequence of the message transfer periods. Three modes are distinguished:

1) Token handling.
2) Send operation.
3) Send and request operation.

5.3.2 Token procedures

5.3.2.1 Token circulation

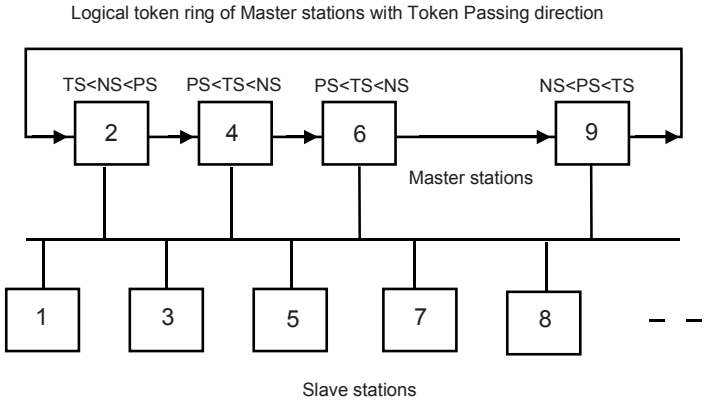
The token is passed from master station to master station in ascending numerical order of station addresses by means of the token DLPDU (see 7.4). To close the logical token ring, the station with the highest address passes the token to the station with the lowest address, see Figure 2.

5.3.2.2 Token reception

If a master station (TS) receives a token DLPDU addressed to itself from a station which is registered as Previous station (PS) in its list of master stations (LMS), it owns the token and may execute message transfer periods. The LMS is generated in a master station in the "Listen_Token" state (see 8.2.4) after power on and is updated and corrected, if necessary, later on upon each receipt of a token DLPDU.

If the token transmitter is not the registered PS, the addressee assumes an error and shall not accept the token. Only a subsequent retry of the same PS is accepted and results in the token receipt, because the token receiver shall assume now that the logical ring has changed. It replaces the originally recorded PS in its LMS by the new one.

Figure 2 shows the logical token-passing ring.



where

TS is this station (address);

PS is previous station (address);

NS is next station (address);

master stations are 2, 4, 6 and 9 (addresses as an example);

slave stations are 1, 3, 5, 7 and 8 (addresses as an example).

Figure 2 – Logical token-passing ring

5.3.2.3 Token transmission

After the master station has finished its message transfer periods – contingent maintenance of the GAP station list (GAPL, see 5.3.2.4) included – it passes the token to its successor (NS) by transmitting the token DLPDU. The functionality of its transceiver is checked by simultaneous monitoring (see 8.2.11, "Pass_Token" state).

If, after transmitting the token DLPDU and after expiration of the synchronization time within the slot time (see 5.5), the token transmitter receives a valid DLPDU, that is, a DLPDU header without any errors, it assumes that its NS owns the token and executes message transfer periods. If the token transmitter receives an invalid DLPDU, it assumes that another master station is transmitting. In both cases it ceases monitoring the token passing and retires, that is, it enters the "Active_Idle" state (see 8.2.5).

If the token transmitter does not recognize any bus activity within the slot time, it repeats the token DLPDU and waits another slot time. It retires thereafter, if it recognizes bus activity within the second slot time. Otherwise, it repeats the token DLPDU to its NS for a last time. If, after this second retry, it recognizes bus activity within the slot time, it retires.

If, after the second retry, there is no bus activity, the token transmitter tries to pass the token to the next but one master station (NS of NS). It continues repeating this procedure until it has found a successor from its LMS. If it does not succeed, the token transmitter assumes that it is the only one left in the logical token ring and transmits the token to itself. If it finds a NS again in a later station registration, it tries again to pass the token.

5.3.2.4 Addition and removal of stations

Master and slave stations may be connected to or disconnected from the transmission medium at any moment. Each master station in the logical token ring is responsible for the addition of new stations in the range from the own station address (TS) to the next station (NS), excluding TS and NS. This address range is called GAP and is represented in the GAP List (GAPL), except the address range between Highest Station Address (HSA: 2 to 126) and 127, which does not belong to the GAP.

Each master station in the logical token ring examines its address range (all GAP addresses) after expiration of the GAP update timer (see 5.5.3.11) for changes concerning Master and slave stations. This is accomplished by examining one address per token receipt, using the "Request DL Status with Reply" request DLPDU (see Table 3, b7=1, Code-No 9: Format 7.1.1 a) and 7.1.2 a)).

Upon receiving the token, GAP maintenance starts immediately after all queued message transfer periods have been executed, if there is still transmission time available (see 5.3.2.6). Otherwise, GAP maintenance starts upon the next or the consecutive token receipts after the high priority message transfer periods and other low priority services invoked before the previous GAP maintenance have been performed (see 5.3.2.7). In realizations, care is necessary to ensure that GAP maintenance and low priority message transfer periods do not block each other.

GAP addresses are examined in ascending order, except the GAP which surpasses the HSA. For example, if the HSA and address 0 are not used by a master station, the master station with the highest address examines address 0 after checking the HSA.

If a station acknowledges positively with the DL status "Master station not ready to enter logical token ring" or "Slave station" (see Table 3, b7=0, Code-No 0, no SC, and Figure 20), it is accordingly marked in the GAPL and the next address is checked. If a station answers with the state "Master station ready to enter logical token ring", the token holder changes its GAPL and LMS and passes the token to the new NS. This station, which has newly been admitted to the logical token ring, has already built up its LMS, when it was in the "Listen_Token" state, so that it is able to determine its GAP range and its NS.

If a station answers with the DL status "Master station in logical token ring", then the token holder does not change its GAP and passes the token to the NS given in the LMS. Thus the skipped master station may retire from the bus in case of permanent not receiving the token. In this case the master station shall enter the "Listen_Token" state. In this state it generates a new LMS and remains in this state until it is addressed once more by a "Request DL Status with Reply" transmitted by its predecessor (PS).

Stations which were registered in the GAPL and which do not respond to a "Request DL Status with Reply" are removed from the GAPL and are recorded as unused station addresses.

Due to performance requirements repeated requests of "Request DL Status with Reply" are not desired.

5.3.2.5 (Re)initialization of the logical token ring

Initialization is primarily a special case of updating the LMS and GAPL. If after power on (PON) of a master station a time-out is encountered in the "Listen_Token" state (no bus activity within Time-out Time T_{TO} (see 5.5)), then the master station shall claim the token ("Claim_Token" state) and start initialization.

When the entire fieldbus system is started, the master station with the lowest station address starts initialization. By transmitting two token DLPDUs addressed to itself (DA = SA = TS) it informs any other master stations (entering a NS into their LMS) that it is now the only station in the logical token ring. Then it transmits a "Request DL Status with Reply" DLPDU to each station in an incrementing address sequence, in order to register other stations. If a station responds with "Master station not ready to enter logical token ring" or with "Slave station" it is entered in the GAPL. The first master station, which answers with "Master station ready to enter logical token ring", is registered as NS in the LMS and thus closes the GAP range of the token holder. Then the token holder passes the token to its NS.

Reinitialization becomes necessary after loss of the token, which causes a time-out (no bus activity). In this case, an entire bus initialization sequence is not required, because LMS and GAPL already exist in the master stations. The time-out expires first in the master station with the lowest address. It takes the token and starts executing regular message transfer periods or passes the token to its NS.

5.3.2.6 Token rotation time

After receiving a token, the DL-entity may carry out high priority and low priority message transfer periods according to the token rotation time. The token rotation time is measured by using the token-rotation-timer. At the beginning the token-rotation-timer is loaded with the target rotation time T_{TR} and decremented with each bit time. The token-rotation-timer stops if the value zero is reached.

The token rotation time shall be measured according to the following rules:

– immediately after token reception the master station shall read the current value of the token-rotation-timer (token holding time T_{TH}) to calculate the real rotation time T_{RR} (difference between the target rotation time and the current value of token-rotation-timer) and shall start the token-rotation-timer with the value target rotation time (T_{TR});

– the T_{RR} represents always the token rotation time of the last token rotation (from the viewpoint of this station).

A system's minimum target rotation time depends on the number of master stations and thus on the token transfer period (T_{TP}) and the duration of high priority message transfer periods (high T_{MP}). The predefined target rotation time T_{TR} shall also contain sufficient time for low priority message transfer periods and a safety margin for potential retries.

In order to keep within the system reaction time required by the field of application, the target rotation time T_{TR} of the token in the logical ring shall be specified.

The system reaction time is defined as the maximum time interval (worst case) between two consecutive high priority message transfer periods of a master station, measured at the DLS-user interface at maximum bus load.

In order to achieve a target rotation time as short as possible, it is recommended for the DLS-user (see IEC 61158-5-3), to declare only important data as high priority message transfer periods and to restrict their length (for example, ≤ 32 octets for the DLSDU, see 6.6).

If the transfer periods defined in 5.6 (equations (52) and (59) as well as (53) and (60)) are included and possible retries are taken into consideration, the operating parameter "target rotation time T_{TR}" (see 5.6 and Table 5), which is necessary for initialization (min T_{TR}), is calculated for the DLS-user as follows:

$$\min T_{TR} = na \times T_{TP} + (na + 1) \times \text{high } T_{MP} + k \times \text{low } T_{MP} + mt \times T_{RMP} \quad (11)$$

where

na is the number of master stations;

k is the estimated number of low priority message transfer periods per token rotation;

T_{TP} is the token transfer period (see 5.6.1.1 and 5.6.2.1);

T_{MP} is the message transfer period, depending on DLPDU length (see 5.6.1.2 and 5.6.2.2);

mt is the numbers of message retry transfer periods per token rotation;

T_{RMP} is the message retry transfer period.

The first term contains one token transfer period per master station. The second term contains one high priority message transfer period for *N+1* master stations. The third term contains the estimated number of low priority message transfer periods per token rotation. The fourth term serves as a safety margin for potential retries. The maximum reaction time for high priority message transfer periods is guaranteed for all bus loads.

5.3.2.7 Message priorities

In the parameter service_class of the DL services the DLS-user (application layer) can choose between two priorities: low and high. The priority is passed to the DLE with the service request.

After token reception, each master station may always execute **one** high priority message transfer period including retries in the case of an error independently of the token holding time T_{TH}.

Further high or low priority message transfer periods may be executed according to the following rules if T_{TH} is still available (see 5.5.5).

– High or low priority messages may be carried out if the calculated real rotation time (T_{RR}) is less than the current value of the token-rotation-timer before the instant of execution of message sending.
– Once a high or low priority message transfer period is started, it is always completed, including any required retry (retries), even if the token-rotation-timer is less than or equal to the value of T_{RR} during the execution.
– If there is no T_{TH} available (the T_{RR} is greater than the current value of the token-rotation-timer before the instant of execution of message sending) then the token shall be passed to NS immediately.

NOTE 1 The prolongation of the token holding time T_{TH} automatically results in a shortening of transmission time for message transfer periods at the next token receipt.

NOTE 2 For further explanation, refer to Annex A and Annex C.

5.3.3 Send or send/request mode

In the send or send/request mode, single message transfer periods are conducted sporadically. The master station's DL-entity initiates this mode due to a local user's request upon receipt of the token. If there are several requests, this mode of operation may be continued until the maximum allowed token holding time expires.

5.4 Service assumed from the PhL

5.4.1 Asynchronous transmission

5.4.1.1 PhS transmission and reception services

The PhL data service includes two service primitives. A request primitive is used to request a service by the DL-entity; an indication primitive is used to indicate a reception to the DL-entity. The names of the respective primitives are as follows:

Ph-ASYN-DATA request

Ph-ASYN-DATA indication

The temporal relationship of the primitives is shown in Figure 3.

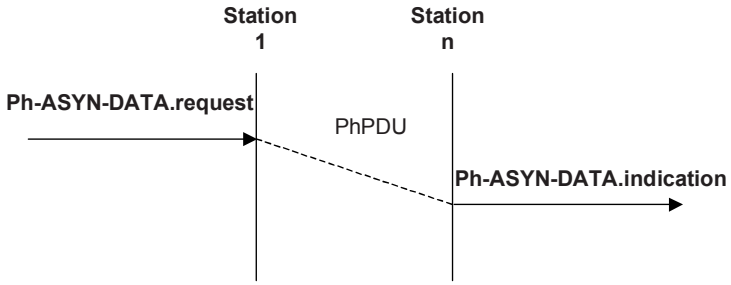


Figure 3 – PhL data service for asynchronous transmission

5.4.1.2 Detailed specification of the service and message exchange

This subclause describes in detail the service primitives and the related parameters in an abstract way. The parameters contain information needed by the PhL-entity.

Parameters of the primitives:

Ph-ASYN-DATA request (DL_symbol)

The parameter DL_symbol shall have one of the following values:

a) ZERO corresponds to a binary "0",

b) ONE corresponds to a binary "1",

c) SILENCE disables the transmitter when no valid DL symbol is to be transmitted.

The Ph-ASYN-DATA request primitive is passed from the DL-entity to the PhL-entity to request that the given symbol shall be sent to the fieldbus medium.

The reception of this primitive shall cause the PhL-entity to attempt encoding and transmission of the DL-symbol.

The Ph-ASYN-DATA request is a primitive, which shall only be generated once per DL-symbol period (t_{BIT}). The PhL-entity may confirm this primitive with a locally defined confirmation primitive.

Ph-ASYN-DATA indication (DL_symbol)

The parameter DL_symbol shall have one of the following values:

– ZERO corresponds to a binary "0",

– ONE corresponds to a binary "1".

The Ph-ASYN-DATA indication primitive is passed from the PhL-entity to the DL-entity to indicate that a DL-symbol was received from the fieldbus medium.

The Ph-ASYN-DATA indication is a primitive, which shall only be generated once per received DL-symbol period (t_{BIT}).

5.4.2 Synchronous transmission

5.4.2.1 General

This subclause defines the assumed physical service (PhS) primitives and their constraints on use by the DLE.

NOTE Proper layering requires that an (N+1)-layer entity not be concerned with, and that an (N)-service interface not overly constrain, the means by which an (N)-layer provides its (N)-services. Thus the Ph-service interface does not require DLEs to be aware of internal details of the PhE (for example, preamble, postamble and DLPDU delimiter signal patterns, number of bits per baud), and do not prevent the PhE from using appropriate evolving technologies.

5.4.2.2 Assumed primitives of the PhS

The granularity of transmission in the fieldbus protocol is one octet. This is the granularity of PhS-user data and timing information exchanged at the PhL-DLL interface.

5.4.2.2.1 PhS characteristics reporting service

The PhS is assumed to provide the following service primitive to report essential PhS characteristics used in DLL transmission, reception, and scheduling activities:

Ph-CHARACTERISTICS indication (minimum-data-rate, framing-overhead)

where

minimum-data-rate — specifies the effective minimum rate of data conveyance in bits per second, including any timing tolerances, of any PhE on the local link.

NOTE 1 A PhE with a nominal data rate of 1 Mbit/s ± 0,01 % would specify a minimum data rate of 0,9999 Mbit/s.

framing-overhead — specifies the maximum number of bit periods, where
period = 1/data rate,
used in any transmission for PhPDUs which do not directly convey data (for example, PhPDUs conveying preamble, DLPDU delimiters, postamble, inter-DLPDU “silence”, and so on).

NOTE 2 If the framing overhead is F and two DL message lengths are L_1 and L_2, then the time to send two immediately consecutive messages of lengths L_1 and L_2 will be at least as great as the time required to send one message of length $L_1 + F + L_2$.

If this framing-overhead is more than the DLEs configured per-DLPDU-PhL-overhead, V(PhLO), then the DLE shall report this discrepancy to DL-management and shall not issue Ph-DATA requests while the discrepancy exists.

NOTE 3 This restriction prohibits DLE transmission while this discrepancy exists. The DLEs local station management may remedy this discrepancy by reconfiguring V(PhLO) to a greater value.

5.4.2.2.2 PhS transmission and reception services

The PhS is assumed to provide the following service primitives for transmission and reception:

Ph-DATA request (class, data);

Ph-DATA indication (class, data);

Ph-DATA confirm (status)

where

class — specifies the Ph-interface-control-information (PhICI) component of the Ph-interface-data-unit (PhIDU). For a Ph-DATA request, its possible values are

START-OF-ACTIVITY — transmission of the PhPDUs which precede Ph-user data should commence;

DATA — the single-octet value of the associated data parameter should be transmitted as part of a continuous correctly-formed transmission;

END-OF-DATA-AND-ACTIVITY — the PhPDUs which terminate Ph-user data should be transmitted after the last preceding octet of Ph-user data, culminating in the cessation of active transmission;

For a Ph-DATA indication, its possible values are:

START-OF-ACTIVITY — reception of an apparent transmission from one or more PhEs has commenced;

DATA — the associated data parameter was received as part of a continuous correctly-formed reception;

END-OF-DATA — the ongoing continuous correctly formed reception of Ph-user data has concluded with correct reception of PhPDUs implying END-OF-DATA;

END-OF-ACTIVITY — the ongoing reception (of an apparent transmission from one or more PhEs) has concluded, with no further evidence of PhE transmission;

END-OF-DATA-AND-ACTIVITY — simultaneous occurrence of END-OF-DATA and END-OF-ACTIVITY;

data — specifies the Ph-interface-data (PhID) component of the PhIDU. It consists of one octet of Ph-user data to be transmitted (Ph-DATA request) or which was received successfully (Ph-DATA indication).

status — specifies either success or the locally detected reason for inferring failure.

The Ph-DATA confirm primitive provides the critical physical timing feedback necessary to inhibit the DLE from starting a second transmission before the first is complete. The final Ph-DATA confirm of a transmission shall not be issued until the PhE has completed the current transmission.

5.4.2.3 Notification of PhS characteristics

The PhE has the responsibility for notifying the DLE of those characteristics of the PhS which are relevant to DLE operation. This notification is accomplished by the PhE by issuing a single Ph-CHARACTERISTICS indication primitive at each of the PhEs PhSAPs at PhE startup.

5.4.2.4 Transmission of Ph-user data

The PhE determines the timing of all transmissions. When a DLE has a DLPDU to transmit, and the DL-protocol gives that DLE the right to transmit, then the DLE shall send the DLPDU, including a concatenated FCS, by making a well-formed sequence of Ph-DATA requests, consisting of a single request specifying START-OF-ACTIVITY; followed by three to 300 consecutive requests, inclusive, specifying DATA; and concluded by a single request specifying END-OF-DATA-AND-ACTIVITY.

The PhE signals its completion of each Ph-DATA request, and its readiness to accept a new Ph-DATA request, with a Ph-DATA confirm primitive; the status parameter of the Ph-DATA confirm primitive conveys the success or failure of the associated Ph-DATA request. A second Ph-DATA request should not be issued until after the Ph-DATA confirm corresponding to the first request has been received from the PhE.

5.4.2.5 Reception of Ph-user data

The PhE reports a received transmission with a well-formed sequence of Ph-DATA indications, which shall consist of either

a) a single indication specifying START-OF-ACTIVITY; followed by consecutive indications specifying DATA; followed by a single indication specifying END-OF-DATA; and concluded by a single indication specifying END-OF-ACTIVITY, or

b) a single indication specifying START-OF-ACTIVITY; followed by consecutive indications specifying DATA; followed by a single indication specifying END-OF-DATA-AND-ACTIVITY, or

c) a single indication specifying START-OF-ACTIVITY; optionally followed by one or more consecutive indications specifying DATA; and concluded by a single indication specifying END-OF-ACTIVITY.

NOTE This last sequence is indicative of an incomplete or incorrect reception. Detection of an error in the sequence of received PhPDUs, or in the PhEs reception process, disables further Ph-DATA indications with a class parameter specifying DATA, END-OF-DATA, or END-OF-DATA-AND-ACTIVITY until after both the end of the current period of activity and the start of a subsequent period of activity have been reported by Ph-DATA indications specifying END-OF-ACTIVITY and START-OF-ACTIVITY, respectively.

In the first two cases, the DLE concatenates the received data and then attempts to parse it into a DLPDU followed by a concatenated FCS. In the last case the DLE discards all reported data and reports the event to DL-management.

5.5 Operational elements

5.5.1 Overview

The following times T are measured in bits. A time t in seconds (s) shall therefore be divided by the bit time t_{BIT}.

5.5.2 Bit time t_{BIT}

The bit time t_{BIT} is the time, which elapses during the transmission of one bit. It is equivalent to the reciprocal value of the data rate:

$$t_{BIT} = \frac{1}{\text{data} \cdot \text{rate}} \ [\text{s bit}^{-1}] \tag{12}$$

5.5.3 Asynchronous transmission

5.5.3.1 Synchronization time (T_{SYN})

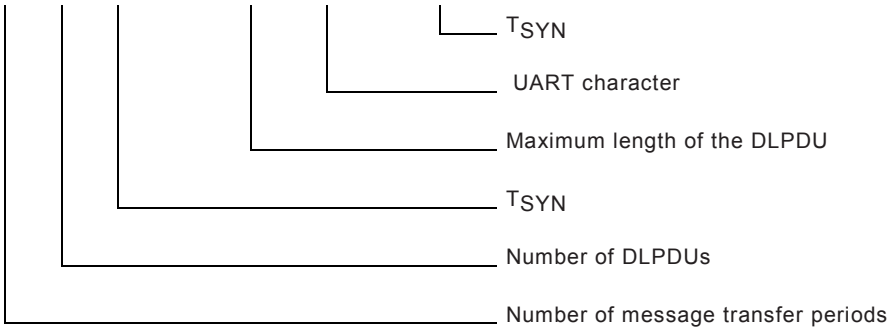
The synchronization time T_{SYN} is the minimum time interval during which each station shall receive idle state (idle = binary "1") from the transmission medium before it may accept the beginning of a send/request DLPDU or token DLPDU. The value of the synchronization time shall be set with:

$$T_{SYN} = 33 \text{ bit} \tag{13}$$

5.5.3.2 Synchronization interval time (T_{SYNI})

The synchronization interval time T_{SYNI} is the maximum allowed time interval between two consecutive synchronization times (T_{SYN}), in order to detect “permanent transmitters”. The value of the synchronization interval time shall be set with:

$$T_{SYNI} = 2 \times (2 \times (33 \text{ bit} + 255 \times 11 \text{ bit})) + 33 \text{ bit} = 11\,385 \text{ bit} \quad (14)$$



This value regards two complete message transfer periods, each of which consists of two DLPDUs of maximum length and the related synchronization times. A transmission disturbance is permitted in one of those synchronization times.

5.5.3.3 Station delay time (T_{SDx})

The station delay time T_{SDx} is the period of time which may elapse between the transmission or receipt of a DLPDUs last bit until the transmission or receipt of a following DLPDUs first bit (with respect to the transmission medium, that is, including line receiver and transmitter). The following three station delays are defined:

a) Station Delay of Initiator (station transmitting request or token DLPDU):

   TSDI

b) Minimum Station Delay of Responders (stations which acknowledge or respond):

   min T_{SDR}

c) Maximum Station Delay of Responders:

   max T_{SDR}

5.5.3.4 Quiet time (T_{QUI})

When transposing the NRZ signals into a different signal coding, the transmitter fall time after switching off the transmitter (at the initiator) shall be taken into account if it is greater than T_{SDR}.

During this quiet time T_{QUI}, transmission and receipt of DLPDUs shall be disabled. This shall also be taken into account when using self-controlled repeaters, whose switching time shall be taken into consideration. The implementation shall ensure, that the following condition is fulfilled:

$$T_{QUI} < \min T_{SDR} \quad (15)$$

In order to fulfil this condition, it may be necessary to prolong min T_{SDR}.

5.5.3.5 Ready time (T_{RDY})

The ready time T_{RDY} is the time within which a master station shall be ready to receive an acknowledgement or response after transmitting a request. The implementation shall ensure, that the following condition is fulfilled:

$$T_{RDY} < \min T_{SDR} \quad (16)$$

In order to fulfil this condition it may be necessary to prolong min T_{SDR}.

When transposing NRZ signals into a different signal coding, the quiet time shall also be taken into consideration when switching off the transmitter. The receiver shall not be enabled before this time:

$$T_{QUI} < T_{RDY} \quad (17)$$

In order to fulfil this condition, it may be necessary to prolong T_{RDY} and thus min T_{SDR} accordingly.

5.5.3.6 Safety margin (T_{SM})

The following time interval is specified as safety margin T_{SM}:

$$T_{SM} = 2 \text{ bit} + 2 \times T_{SET} + T_{QUI} \quad (18)$$

T_{SET} is the set-up time, which expires from the occurrence of an event (for example, an interrupt on the last bit of a sent DLPDU or when synchronization time expires) until the necessary reaction is performed (for example, to start synchronization time or to enable the receiver).

5.5.3.7 Idle time (T_{IDx})

The idle time T_{IDx} is the time which expires at the initiator either after receipt of a DLPDUs last bit (measured at the line receiver) as idle = binary "1" on the transmission medium, until a new DLPDUs first bit is transmitted on the medium (including line transmitter) or between transmitting the last bit of a DLPDU which is not to be acknowledged and transmitting the first bit of the next DLPDU. The idle time shall be at least the synchronization time plus the safety margin T_{SM} (see Figure 4, Figure 5 and Figure 6 case a)).

$$T_{IDx} \geq T_{SYN} + T_{SM} \quad (19)$$

At high data rates (see Figure 4, case b) and c) and Figure 5 case b)) the synchronization time is very short, hence the station delays become significant and shall be taken into consideration.

Two idle times are distinguished. After an acknowledgement, response or token DLPDU the idle time shall be calculated as follows:

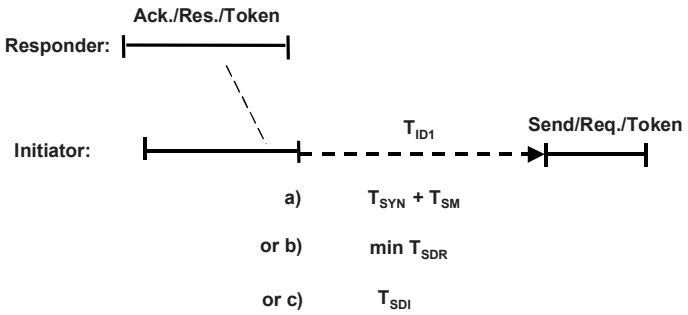


Figure 4 – Idle time T_{ID1}

$$T_{ID1} = \max((T_{SYN} + T_{SM}), (\min T_{SDR}), T_{SDI}) \quad (20)$$

Care shall be taken that the time to update the LMS is not greater than T_{ID1}. This can be accomplished by prolonging T_{SET} or min T_{SDR}. If the necessary prolongation cannot be reached with the range of value of T_{SET}, than the min T_{SDR} shall be made longer.

After a send DLPDU which is **not** to be acknowledged (SDN or CS), the idle time shall be calculated as shown in Figure 5 and equation (34).

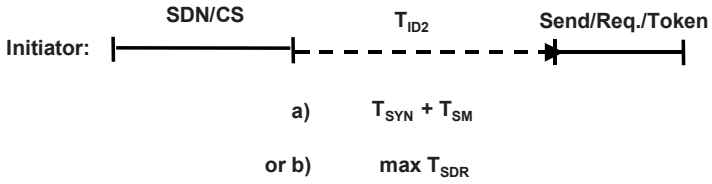


Figure 5 – Idle time T_{ID2} (SDN, CS)

$$T_{ID2} = \max ((T_{SYN} + T_{SM}) , (\max T_{SDR})) \quad (21)$$

Also, after a response DLPDU (MSRD) the idle time shall be calculated as shown in Figure 6 and equation (34).

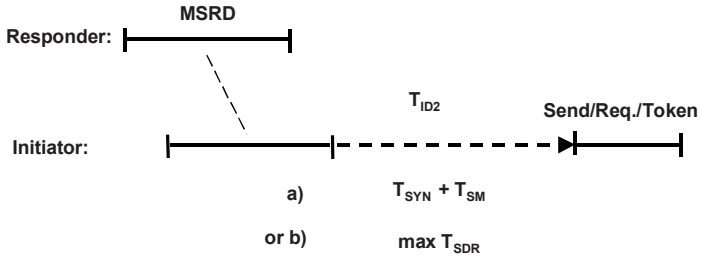


Figure 6 – Idle time T_{ID2} (MSRD)

5.5.3.8 Transmission delay time (T_{TD})

The transmission delay time T_{TD} is the maximum time, which elapses on the transmission medium between transmitter and receiver when a DLPDU is transmitted. When computing it, delay times of repeaters shall be considered, if necessary.

EXAMPLE Computing the transmission delay time: given a line length of 200 m without repeaters, t_{TD} is approximately 1 µs and thus at 500 kbit/s:

T_{TD} = (1µs × 500 kbit/s) = 0,5 bit.

5.5.3.9 Slot time (T_{SL})

The slot time T_{SL} is the maximum time the initiator shall wait for the complete receipt of the first character (see 6.1.1, UART character, 1 UC = 11 bits) of the immediate acknowledgement or response DLPDU, after transmitting the last bit of a send/request DLPDU (including the line transmitter) (see Figure 7). Furthermore, T_{SL} is the maximum time the initiator waits for the token receiver's first UART character (1.UC) after transmitting a token DLPDU (see Figure 8). Theoretically, two slot times are distinguished. After a send/request DLPDU the slot time shall be calculated as follows:

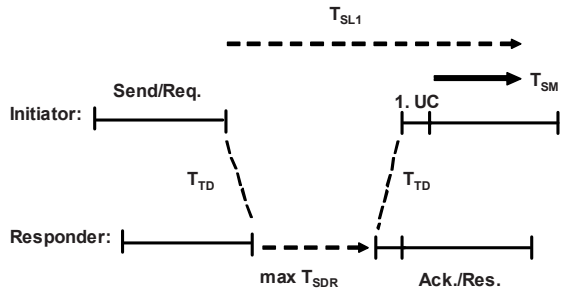


Figure 7 – Slot time T_{SL1}

$$T_{SL1} = 2 \times T_{TD} + \max T_{SDR} + 11 \text{ bit} + T_{SM} \quad (22)$$

After a token DLPDU the slot time shall be calculated as follows:

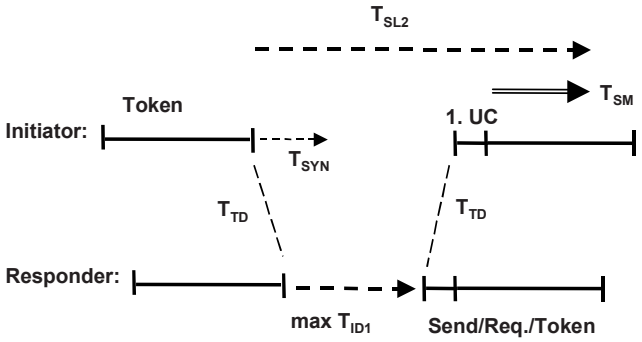


Figure 8 – Slot time T_{SL2}

$$T_{SL2} = 2 \times T_{TD} + \max T_{ID1} + 11 \text{ bit} + T_{SM} \quad (23)$$

In order to simplify the realization, only one slot time, the longer one, may be used in the system. This does not influence the system reaction time negatively, as the slot time is a pure monitoring time.

$$T_{SL} = \max (T_{SL1} , T_{SL2}) \quad (24)$$

5.5.3.10 Time-out time (T_{TO})

The time-out time T_{TO} serves to monitor the Master and slave stations' bus activity and idle time. Monitoring shall start either immediately after PON, in the "Listen_Token" or "Passive_Idle" state, or later after receiving the last bit of a DLPDU. It shall end after receiving the first bit of the following DLPDU. If the phase of no bus activity times out, the bus shall be regarded as inactive. (This is an error case, for example, due to a lost token DLPDU.) The time-out interval shall be set to:

$$T_{TO} = 6 \times T_{SL} + 2 \times n \times T_{SL} \quad (25)$$

For master stations: n = station address (0 to 126)

For slave stations: n = 130, independent of the station's address

The first term ensures that there is sufficient difference to the maximum permissible idle time between two DLPDUs. The second term ensures that not all master stations claim the token at the same moment after an error has occurred.

5.5.3.11 GAP update time (T_{GUD})

The GAP update time T_{GUD} serves for initializing GAP maintenance by the master station. After the first generation of the GAPL, update of the GAP image is cyclically initialized after every interval T_{GUD}. This initialization takes place at the next possible token receipt, if there is still token holding time available after the regular DLPDU transfer periods, or during later token holding phases. The GAP update time is a multiple of the target rotation time T_{TR} and shall be set to:

$$T_{GUD} = G \times T_{TR} \quad \text{(where } 1 \leq G \leq 100\text{)} \tag{26}$$

T_{TR} is the target rotation time (see 5.3.2.6).

5.5.3.12 Isochronous mode

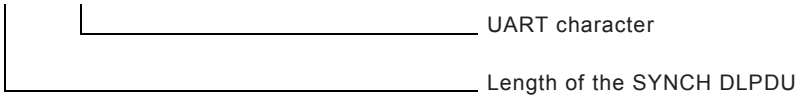
5.5.3.12.1 Isochronous cycle time (T_{CT})

The isochronous cycle time T_{CT} serves to supervise the isochronous cycle. The first cycle starts by sending the SYNCH DLPDU after reception of the token DLPDU.

5.5.3.12.2 IsoM synchronization DLPDU time (T_{SYNCH})

The isochronous mode synchronization DLPDU time describes the time, which is needed to send the SYNCH DLPDU at the start of a new IsoM cycle. The value of the IsoM synchronization message time shall be set with:

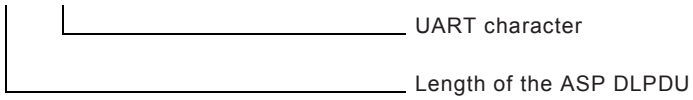
$$T_{SYNCH} = 13 \times 11 \text{ bit} = 143 \text{ bit} \tag{27}$$



5.5.3.12.3 Active spare time DLPDU time (T_{ASM})

The active spare time DLPDU time describes the duration that is needed to send an active spare time (ASP) DLPDU.

$$T_{ASM} = T_{ID1} + (6 \times 11 \text{ bit}) \tag{28}$$



5.5.3.12.4 Real isochronous cycle time (T_{RCT})

The real isochronous cycle time is the read value of the isochronous-cycle-timer.

5.5.3.12.5 Spare time (T_{RES})

Within the isochronous cycle the token is passed by the IsoM Master after the transmission of all high priority messages and the number of configured low priority messages. After receiving the next token addressed to the IsoM Master, the station read the isochronous cycle timer (T_{RCT}) in order to calculate the remaining time before the next SYNCH DLPDU is to be sent.

$$T_{RES} = T_{CT} - T_{RCT} \quad (29)$$

Within the spare time, at least one active spare time DLPDU shall be sent. For further active spare time, DLPDUs shall be valid.

$$T_{RES} > T_{ASM} + T_{ID1} \quad (30)$$

5.5.3.12.6 Passive spare time (T_{PSP})

The passive spare time denotes the part of isochronous cycle where the IsoM Master shows no bus activities. It shall be less than the minimum time-out time T_{TO} under consideration of possible delays in the stations and during transmission. The passive spare time is defined as follow:

$$T_{PSP} < 6 \times T_{SL} - T_{SM} - 2 \times T_{TD} \quad (31)$$

$$T_{PSP} > T_{ID1} + T_{ASM} + \max T_{SDR} \quad (32)$$

5.5.3.12.7 Time shift (T_{SH})

The time shift is the difference between the measured cycle time and the calculated cycle time when the passive-spare-timer expires.

$$T_{SH} = T_{RCT} - T_{CT} \quad (33)$$

5.5.3.13 Send delay time (T_{SD})

The send delay time is the time that elapses between the receiving of a DL-CS-TIME-EVENT.request primitive at the DLE of the time master and the DLPDUs last bit of a resulting TE DLPDU (see 7.7) transmitted by the time master.

5.5.3.14 Receive delay time (T_{RD})

The receive delay time is the time that elapses at a time receiver between the receiving of the last bit of a TE DLPDU (see 7.7) and the receiving of a DLPDUs last bit of a CV DLPDU (see 7.8).

5.5.3.15 Clock synchronization interval time (T_{CSI})

The clock synchronization interval time is used to monitor the synchronization sequences within the DLE.

5.5.4 Synchronous transmission

5.5.4.1 Synchronization time (T_{SYN})

The synchronization time is the minimum time interval during which each station shall receive no activity from the transmission medium before it may accept the beginning of a send/request DLPDU or token DLPDU.

The synchronization time shall correspond to the post-transmission gap time (T_{PTG}) that is defined in 9.2.8 of IEC 61158-2. Its value shall be set to at least 4 bit and may be increased by the DLMS-user up to 32 bit.

$$T_{SYN} = T_{PTG} = T_{QUI} \quad (34)$$

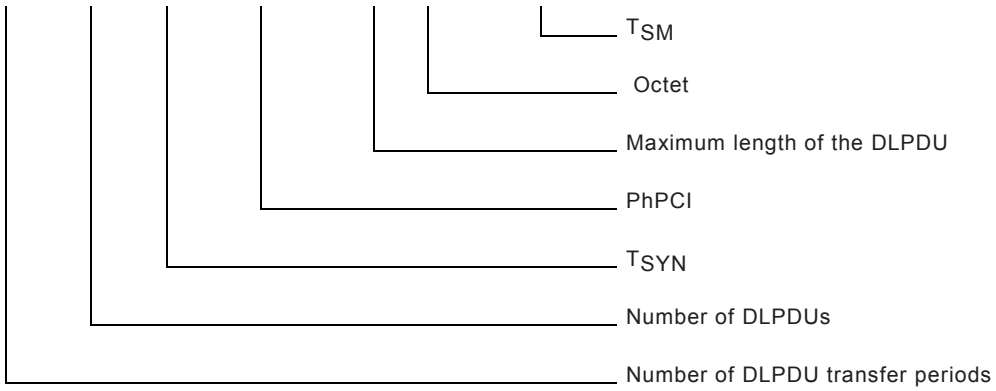
$$T_{SYN} = 4 \text{ to } 32 \text{ bit} \quad (35)$$

5.5.4.2 Synchronization interval time (T_{SYNI})

The synchronization interval time T_{SYNI} is the maximum allowed time interval between two consecutive synchronization times (T_{SYN}), in order to detect "permanent transmitters".

The value of T_{SYNI} shall be set as follows:

$$T_{SYNI} = 2 \times (2 \times (32 \text{ bit} + 80 \text{ bit} + 255 \times 8 \text{ bit})) + 64 \text{ bit} = 8\,672 \text{ bit} \quad (36)$$



This value regards two complete message sequences, each of which consists of two DLPDUs of maximum length and the associated maximum PhL Protocol Control Information (PhPCI: Preamble, Start Delimiter, End Delimiter) and the maximum synchronization time (post-transmission gap). A transmission disturbance is permitted in one of those synchronization times.

5.5.4.3 Station delay time (T_{SDx})

The station delay time T_{SDx} is the period of time, which may elapse between the transmission or receipt of a DLPDUs last octet until the transmission or receipt of a following DLPDUs first octet (with respect to the transmission medium, that is, including line receiver and transmitter). The following three station delays are defined:

a) Station Delay of Initiator (station transmitting request or token DLPDU)

T_{SDI}

b) Minimum Station Delay of Responders (station that acknowledges or responds)

min T_{SDR}

c) Maximum Station Delay of Responders

max T_{SDR}

5.5.4.4 Quiet time (T_{QUI})

The transmitter fall time or repeater switch time corresponds to the post-transmission gap time (T_{SYN}). The following shall apply:

$$T_{QUI} = T_{PTG} = T_{SYN} \quad (37)$$

5.5.4.5 Ready time (T_{RDY})

The ready time T_{RDY} is the time within which a Master station shall be ready to receive an acknowledgement or response after transmitting a request. The implementation shall ensure, that the following condition is fulfilled:

$$T_{RDY} < \min T_{SDR} \quad (38)$$

In order to fulfil this condition it may be necessary to prolong min T_{SDR}.

During the quiet time T_{QUI}, transmission and receipt of DLPDUs shall be disabled. The implementation shall ensure, that the following condition is fulfilled:

$$T_{QUI} = T_{PTG} = T_{SYN} < T_{RDY} \quad (39)$$

In order to fulfil this condition the min T_{SDR} shall be increased according to equation (38) if necessary.

5.5.4.6 Safety margin (T_{SM})

The safety margin T_{SM} is defined as the time interval:

$$T_{SM} = 2 \text{ bit} + 2 \times T_{SET} \quad (40)$$

T_{SET} is the set-up time, which expires from the occurrence of an event (for example, an interrupt on the last octet of a sent DLPDU or on synchronization time expiration) until the execution of the necessary reaction.

5.5.4.7 Idle time (T_{IDx})

The idle time T_{IDx} is the time which expires at the initiator after a Ph-DATA indication primitive until sending of a new DLPDU with Ph-DATA request primitive or after passing a Ph-DATA request primitive with a Ph-DATA confirm primitive to transmit a DLPDU which is not to be acknowledged until passing a new Ph-DATA request primitive for transmitting the next DLPDU. The idle time shall be at least the synchronization time plus the safety margin T_{SM}.

$$T_{IDx} \geq T_{SYN} + T_{SM} \quad (41)$$

Two idle times are distinguished (see description of idle time in 5.5.3.7, Figure 4, Figure 5, and Figure 6). After an acknowledgement, response or token DLPDU the idle time shall be calculated as follows:

$$T_{ID1} = \max ((T_{SYN} + T_{SM}), (\min T_{SDR}), T_{SDI}) \quad (42)$$

Care shall be taken that the time to update the LMS is not greater than T_{ID1}. This can be accomplished by prolonging T_{SET} or min T_{SDR}. If the necessary prolongation cannot be reached with the range of value of T_{SET}, then the min T_{SDR} shall be made longer.

After a send DLPDU, which is not to be acknowledged (SDN, CS), the idle time shall be calculated as follows:

$$T_{ID2} = \max ((T_{SYN} + T_{SM}), (\max T_{SDR})) \quad (43)$$

5.5.4.8 Transmission delay time (T_{TD})

The transmission delay time T_{TD} is the maximum time, which elapses on the transmission medium between transmitter and receiver when a DLPDU is transmitted. When computing it, delay times of repeaters shall be considered if necessary. It shall also conform to any restrictions placed on it in IEC 61158-2, where it is called “propagation delay”.

5.5.4.9 Slot time (T_{SL})

The slot time T_{SL} is the maximum time the initiator shall wait after passing a Ph-DATA request primitive for transmitting a request DLPDU from the Ph-DATA confirm primitive until receiving the first Ph-DATA indication primitive as an indication of receiving the immediate acknowledgement or response (see Figure 9). Furthermore, T_{SL} is the maximum time the initiator shall wait for a Ph-DATA indication primitive after the token DLPDU as reaction to receiving the first DLPDU octet from the token receiver. Theoretically, two slot times are distinguished (see Figure 10). After a send/request DLPDU the following slot time shall be calculated:

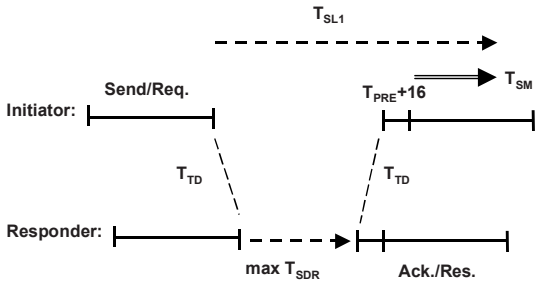


Figure 9 – Slot time T_{SL1}

$$T_{SL1} = 2 \times T_{TD} + \max T_{SDR} + T_{PRE} + 16 \text{ bit} + T_{SM} \quad (44)$$

After a token DLPDU the following slot time shall be calculated:

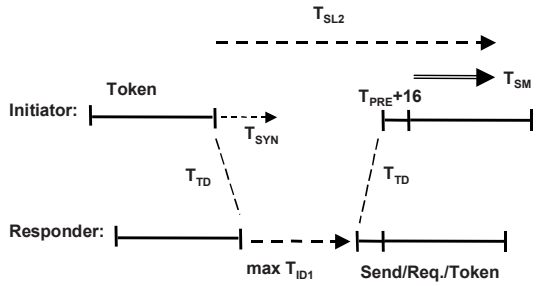


Figure 10 – Slot time T_{SL2}

$$T_{SL2} = 2 \times T_{TD} + \max T_{ID1} + T_{PRE} + 16 \text{ bit} + T_{SM} \quad (45)$$

where T_{PRE} is Preamble period (see IEC 61158-2)

In order to simplify the realization, only the longer slot time may be used in the system. This does not influence the system reaction time negatively, as the slot time is merely a monitoring time.

$$T_{SL} = \max (T_{SL1}, T_{SL2}) \quad (46)$$

5.5.4.10 Time-out time (T_{TO})

The time-out time T_{TO} serves to monitor the Master and Slave stations' bus activity and idle time. Monitoring shall start either immediately after PON, in the "Listen_Token" or "Passive_Idle" state, or later after the reception of a Ph-DATA indication primitive. It shall end upon receipt of a Ph-DATA indication primitive for reception of the first octet of a following DLPDU. If the phase of no bus activity times out, the bus shall be regarded as inactive. (This

is an error case, for example, due to a lost token DLPDU.). The time-out time shall be set to the following value:

$$T_{TO} = 6 \times T_{SL} + 2 \times n \times T_{SL} \quad (47)$$

For Master stations: n = station address (0 to 126)

For Slave stations: n = 130, independent of their station address

5.5.4.11 GAP update time (T_{GUD})

Suclause 5.5.3.11 shall apply.

5.5.4.12 Isochronous Mode

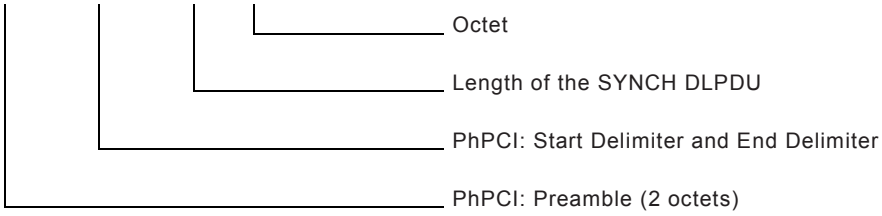
5.5.4.12.1 Isochronous cycle time (T_{CT})

Suclause 5.5.3.12.1 shall apply.

5.5.4.12.2 IsoM synchronization DLPDU time (T_{SYNCH})

The isochronous mode synchronization DLPDU time describes the time, which is needed in order to send the SYNCH DLPDU at the start of a new IsoM cycle. The value of the IsoM synchronization message time shall be set with:

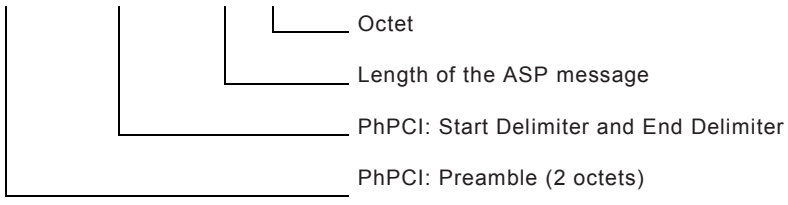
$$T_{SYNCH} = 16 \text{ bit} + 16 \text{ bit} + 13 \times 8 \text{ bit} = 136 \text{ bit} \quad (48)$$



5.5.4.12.3 Active spare time DLPDU time (T_{ASM})

The active spare time DLPDU time describes the duration that is needed to send an active spare time (ASP) DLPDU.

$$T_{ASM} = T_{PTG} + 16 \text{ bit} + 16 \text{ bit} + (6 \times 8 \text{ bit}) \quad (49)$$



5.5.4.12.4 Real isochronous cycle time (T_{RCT})

Suclause 5.5.3.12.4 shall apply.

5.5.4.12.5 Spare time (T_{RES})

Suclause 5.5.3.12.5 shall apply.

5.5.4.12.6 Passive spare time (T_{PSP})

The passive spare time denotes the part of isochronous cycle where the IsoM Master shows no bus activities. It shall be less than the minimum time-out time T_{TO} under consideration of possible delays in the stations and during transmission. The passive spare time is defined as follow:

$$T_{PSP} < 6 \times T_{SL} - T_{SM} - 2 \times T_{TD} \quad (50)$$

$$T_{PSP} > T_{ID1} + T_{ASM} + \max T_{SDR} \quad (51)$$

5.5.4.12.7 Time shift (T_{SH})

Subclause 5.5.3.12.7 shall apply.

5.5.4.13 Send delay time (T_{SD})

The send delay time is the time that elapses between a passed DL-CS-TIME-EVENT request primitive to the DLE of the time master and the DLPDUs last octet of a resulting TE DLPDU (see 7.7) transmitted by the time master.

5.5.4.14 Receive delay time (T_{RD})

The receive delay time is the time that elapses by a time receiver between the receiving of a DLPDUs last octet of a TE DLPDU (see 7.7) and the last octet of a CV DLPDU (see 7.8).

5.5.4.15 Clock synchronization interval time (T_{CSI})

Subclause 5.5.3.15 shall apply.

5.5.5 Timers and counters

5.5.5.1 Asynchronous transmission

5.5.5.1.1 Timers

In order to measure the token rotation time and to realize the supervisory times the following timers shall be implemented:

token-rotation-timer, idle-timer, slot-timer, time-out-timer, syn-interval-timer, GAP-update-timer, isochronous-cycle-timer, passive-spare-timer, send-delay-timer and receive-delay-timer.

token-rotation-timer: When a Master station receives the token, this timer is loaded with the target rotation time T_{TR} and decremented each bit time. When the station again receives the token, the timer value, the remaining time or token holding time T_{TH}, is read and the timer reloaded with T_{TR}. The real rotation time T_{RR} results from the difference $T_{TR} - T_{TH}$.

The token-rotation-timer can be read out at every moment, representing always the value of the actual token rotation. Low priority message transfer periods may be processed if at the instant of processing the real token rotation time is less than the value of the actual token rotation.

idle-timer: This timer monitors the idle state (binary "1"), the synchronization time, immediately on the bus line. The synchronization time preceding each request is necessary for unambiguous receiver synchronization. The idle-timer of Slave stations and Master stations "without token" is loaded with T_{SYN} after the transmission or receipt of a DLPDUs last bit and then decremented each bit time. The receiver shall be enabled immediately after the timer has expired. The timer of a Master station "with token" is loaded according to the

data transmission service with T_{ID1} or T_{ID2} (see 5.5.3). A new request or token DLPDU may only be transmitted after expiration of the timer. When the signalling level is binary "0", the timer is always reloaded.

slot-timer: This timer in a Master station monitors after a request or token pass whether the receiving station responds or becomes active within the predefined time T_{SL}, the slot time. After transmission of a DLPDUs last bit this timer is loaded with T_{SL} and decremented each bit time as soon as the receiver is enabled. If the timer expires before a DLPDUs first bit is received, an error has occurred. Then a retry or a new message transfer period shall be initiated.

time-out-timer: This timer monitors bus activity in Master and Slave stations. After the transmission or receipt of a DLPDUs last bit the timer is loaded with a multiple of the slot time (see 5.5.3) and decremented each bit time as long as no new DLPDU is received. If the timer expires, a fatal error has occurred, which for the Master station causes a (re)initialization. The DLMS-user of the Slave and Master station receives a time-out notification.

syn-interval-timer: Master and Slave stations use this timer to monitor the transmission medium whether a receiver synchronizing (T_{SYN}, idle state, idle = binary "1") occurs within T_{SYNI}. Each time the receiver is synchronized, the timer is loaded with T_{SYNI} (see 5.5.3). From the beginning of a DLPDU (first start bit) the timer is decremented each bit time as long as no new T_{SYN} is detected. If the timer expires, an error has occurred on the transmission medium, for example, stuck at "0" or permanent "0" / "1" edges. The DLMS-user is notified accordingly.

GAP-update-timer: Only Master stations need this timer. Its expiration indicates the moment for GAP maintenance. After a complete GAP check, which may last several token rotations (segmented; see 5.3.2.4), the timer is loaded with a multiple (G) of the target rotation time T_{TR} (see 5.3.2.6).

NOTE For ease of implementation, this timer can be implemented as a counter, decremented at each token receipt.

isochronous-cycle-timer: The timer supervises the isochronous cycle. It is loaded with value equal to 0 at the starting point of the isochronous mode (after receiving the first token) and increments every bit time. The timer is readable at any time. The timer is started either by loading with the value equal to 0 at the first beginning of the isochronous cycle or after expiration of the passive-spare-timer. In cases the timer is started too late, it is loaded with the value of the time difference between the real isochronous cycle time of the last cycle and the isochronous cycle time.

passive-spare- timer: This timer monitors the idle time before sending a SYNCH message. It shall be set according to the time difference between T_{CT} and the current value of the isochronous-cycle-timer T_{RCT} at the end of the last ASP message if the difference is greater than 0. The transmission of the last bit of the last ASP message in an isochronous cycle starts this timer. In the isochronous mode after expiration of the passive-spare-timer, a SYNCH message shall be sent and a synch notification event shall be sent to the DLMS-user.

send-delay-timer: When the time master receives a DL-CS-TIME-EVENT request primitive from the local DLS-user, this timer is started with the value = $2 \times T_{CSI}$ and decremented each bit time. The timer is stopped after confirmation of the transmission of the last portion of a TE DLPDU (see 7.7). The value of send delay time is calculated as the difference between the start value ($2 \times T_{CSI}$) and the read send-delay-timer value. The calculated send delay time is passed to the local DLS-user. In case the timer has expired, the DLS-user is notified of a clock synchronization sequence violation.

receive-delay-timer: When the time receiver receives the DLPDUs last bit of a TE DLPDU (see 7.7), this timer is started with the value = $2 \times T_{CSI}$ and decremented each bit time. The timer is stopped after the receiving the last bit of a CV DLPDU (see 7.8). The value of the

receive delay time is calculated as difference between the start value ($2 \times T_{CSI}$) and the read receive-delay-timer value. The calculated receive delay time is passed to the local DLS-user. In case the timer expired, a clock synchronization sequence violation is notified to the DLS-user.

When a Master station enters the "Listen_Token" state, the idle-timer is loaded with T_{SYN}, the time-out-timer with T_{TO}, the syn-interval-timer with T_{SYNI} and the other timers are cleared. When a Slave station enters the "Passive_Idle" state, the time-out-timer is loaded with T_{TO} and the syn-interval-timer with T_{SYNI}.

5.5.5.1.2 Counters

For installation and maintenance the following pairs of counters (DL-variables) may be used:

For Master stations:

– counter for transmitted DLPDUs (DLPDU_sent_count), except for the SDN and Request DL Status with Reply services,

– counter for transmitted DLPDU retries (Retry_count).

Additional counters may be provided for each individual remote station:

– counter for transmitted DLPDUs (DLPDU_sent_count_sr), except for the SDN and Request DL Status with Reply services and for token DLPDUs,

– counter for transmitted DLPDUs with no response or erroneous response (Error_count), except for the SDN and Request DL Status with Reply services and for token DLPDUs.

For Slave and Master stations:

– counter for received valid start delimiters (SD_count),

– counter for received invalid start delimiters (SD_error_count).

When a station enters the "Listen_Token" or "Passive_Idle" state, the counters are cleared and enabled. If a counter reaches its maximum, counting of this counter as well as of the related comparative counter is stopped. When clearing a counter the related comparative counter is also cleared and they are enabled again. The DLMS-user may access these counters using the Set/Read Value services.

5.5.5.2 Synchronous transmission

5.5.5.2.1 Timers

As explained in 5.5.5.1.1, the following timers shall be implemented to measure the token rotation time and to realize the monitor timers:

Token-rotation-timer, idle-timer, slot-timer, time-out-timer, syn-interval-timer, GAP-update-timer, isochronous-cycle-timer, passive-spare-timer, send-delay-timer and receive-delay-timer.

token-rotation-timer: The functionality of this timer is as defined in 5.5.5.1.1.

idle-timer: This timer monitors the idle state $T_{PTG} = T_{SYN} = T_{QUI}$ on the bus line. The idle-timer in the Master station with the token is loaded with T_{ID1} or T_{ID2} depending on the data transmission service (see 5.5.4). The timer is decremented every bit time, when after a Ph-DATA request primitive (PhICI specifying END-OF-DATA-AND-ACTIVITY) either the Ph-DATA confirm primitive at the transmitter or the Ph-DATA indication primitive (PhICI specifying either END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY) at the receiver is transferred. A new request or a new token DLPDU may be transmitted only after expiration of the timer.

slot-timer: After a request from or a token transfer by a Master station, this timer of the Master station monitors whether the receiving station responds or becomes active within the defined slot time T_{SL}. The timer is initialized with T_{SL} and is decremented every bit time after each transmission of a DLPDU. This DLPDU transmission is indicated by a Ph-DATA confirm primitive after transfer of a Ph-DATA request primitive (PhICI specifying END-OF-DATA-AND-ACTIVITY). If the timer expires before a DLPDU has been received, as indicated by a Ph-DATA indication primitive (PhICI specifying START-OF-ACTIVITY), an error has occurred. As a result of that, a retry or a new message transfer period is initiated.

time-out-timer: This timer monitors bus activity in Master and Slave stations. After transfer of the last Ph-DATA request primitive with PhICI specifying END-OF-DATA-AND-ACTIVITY and return of the corresponding Ph-DATA confirm primitive, or after receiving a Ph-DATA indication primitive with PhICI specifying either END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY, the timer is loaded with a multiple of the slot time (see 5.5.4) and is decremented every bit time as long as no Ph-DATA indication primitive with PhICI specifying START-OF-ACTIVITY has been received. If the timer expires, a fatal error has occurred, which for the Master station causes a (re)initialization. The DLMS-user of the Slave or Master station respectively receives a time-out notification.

syn-interval-timer: Master and Slave stations use this timer to monitor the transmission medium for "permanent transmitters". After every Ph-DATA indication primitive with PhICI specifying START-OF-ACTIVITY, the timer is loaded with the value T_{SYNI} (see 5.5.4) and decremented every bit time, as long as no Ph-DATA indication primitive with PhICI specifying either END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY has been received. If the timer expires, an error of the transmission medium has occurred. The DLMS-user receives a corresponding notification.

GAP-update-timer: This timer operates in the same way as described in 5.5.5.1.1.

isochronous-cycle-timer: The functionality of this timer is as defined in 5.5.5.1.1.

passive-spare- timer: This timer monitors the idle time before sending a SYNCH message. It shall be set according to the time difference between T_{CT} and the current value of the isochronous-cycle-timer T_{RCT} at the end of the last ASP message if the difference is greater than 0. The transmission of the last bit of the last ASP message in an isochronous cycle starts this timer. In the isochronous mode after expiration of the passive-spare-timer, a SYNCH message shall be sent and a synch notification event shall be sent to the DLMS-user.

send-delay-timer: When the time master receives a DL-CS-TIME-EVENT request primitive from the local DLS-user, this timer is started with the value = $2 \times T_{CSI}$ and decremented each bit time. The timer is stopped after confirmation of the transmission of the last portion of a TE DLPDU (see 7.7). The value of send delay time is calculated as the difference between the start value ($2 \times T_{CSI}$) and the read send-delay-timer value. The calculated send delay time is passed to the local DLS-user. In case the timer has expired, the DLS-user is notified of a clock synchronization sequence violation.

receive-delay-timer: When the time receiver receives the DLPDUs last bit of a TE DLPDU (see 7.7) indicated by a Ph-DATA indication primitive (PhICI specifying either END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY) has been received, this timer is started with the value = $2 \times T_{CSI}$ and decremented each bit time. The timer is stopped after the receiving of the DLPDUs last bit of a CV DLPDU (see 7.8) indicated by a Ph-DATA indication primitive (PhICI specifying either END-OF-ACTIVITY or END-OF-DATA-AND-ACTIVITY) has been received. The value of the receive delay time is calculated as difference between the start value ($2 \times T_{CSI}$) and the read receive-delay-timer value. The calculated receive delay time is passed to the local DLS-user. In case the timer expired, a clock synchronization sequence violation is notified to the DLS-user.

5.5.5.2.2 Counters

The specifications given in 5.5.5.1 shall apply for the optional counters.

5.6 Cycle and system reaction times

5.6.1 Asynchronous transmission

5.6.1.1 Token transfer period

The base load in a system with several Master stations, that is, the busload caused by medium access control (token DLPDUs) and not by regular message transfer periods, is determined by the token period TP. The total base load per token rotation results from *na* (number of Master stations) token cycles. The transfer period T_{TP} is composed of the token DLPDU time T_{TF}, the transmission delay time T_{TD} and the idle time T_{ID1}. T_{ID1} results from the station delay time T_{SDI} or the synchronization time T_{SYN} respectively. T_{TP} is measured in bits (see Figure 11).

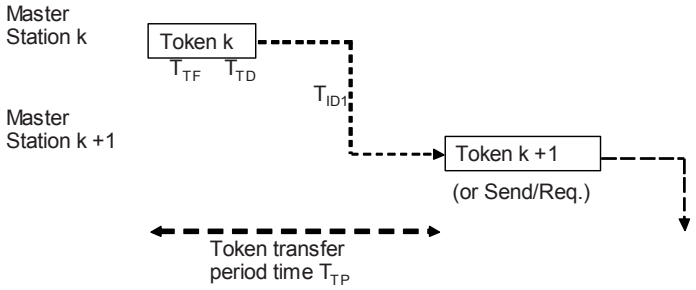


Figure 11 – Token transfer period

$$T_{TP} = T_{TF} + T_{TD} + T_{ID1} \quad (52)$$

The token DLPDU time T_{TF} is determined by the number of UART characters, UC, in the token DLPDU. A UART character always consists of 11 bits (see 6.1.1) and hence the token DLPDU comprises altogether 33 bits. The transmission delay time T_{TD} depends on the line length (about 5 ns/m without repeater) and is mostly substantially less than the other times. The idle time T_{ID1}, which elapses between the token DLPDUs, contains the station delay time T_{SDI} of the token receiver on the one hand, on the other hand the synchronization time T_{SYN} + T_{SM} shall be used, if this sum is larger than T_{SDI} (see 5.5.3). This mainly occurs at low data rates (< 100 kbit/s).

5.6.1.2 Message transfer period

A message transfer period MP consists of the send/request DLPDU and the acknowledgement or response DLPDU. The transfer period is composed of the DLPDU transmission times, the transmission delay times and the station delay times.

The station delay time T_{SDR} elapses between request and acknowledgement or response. This time is needed for decoding the request and assembling the acknowledgement or response DLPDU. It depends on the protocol implementation in the station and is mostly substantially greater than the transmission delay time T_{TD}. The idle time T_{ID}, which elapses between acknowledgement or response and new request, contains also the station delay time (see 5.5.3). However, T_{SYN} + T_{SM} shall be used, if this sum is greater than T_{SDI}.

Figure 12 shows the Message transfer periods.

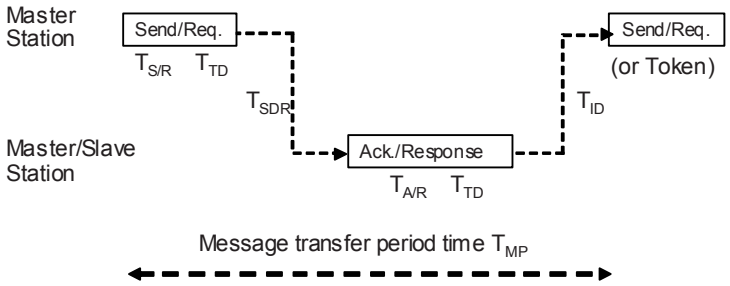


Figure 12 – Message transfer period

$$T_{MP} = T_{S/R} + T_{SDR} + T_{A/R} + T_{ID} + 2 \times T_{TD} \quad (53)$$

If the initiator does not receive a valid response DLPDU, caused by disturbances at a station or at the medium, the initiator will repeat the send/request DLPDU depending on its retry limit and the state of the addressed station. For this retry message transfer period (RMP) another transfer period T_{RMP} applies. T_{RMP} is composed of the send/request DLPDU transmission time and the slot time.

$$T_{RMP} = T_{S/R} + T_{SL} \quad (54)$$

At data rates < 100 kbit/s decoding and evaluation of DLPDUs may partially keep pace with their reception. Station delay times become considerably shorter then.

The DLPDU transmission times ($T_{S/R}$, $T_{A/R}$) are determined by the number of UART characters (UC). Thus, they are calculated as follows:

$T_{S/R}$ = a × 11 bit, where “a” is the number of UC in a send/request DLPDU

$T_{A/R}$ = b × 11 bit, where “b” is the number of UC in an ack/response DLPDU

EXAMPLE a = 6, for the request DLPDU: $T_{S/R}$ = 66 bit; b = 59, for the response DLPDU:$T_{A/R}$ = 649 bit (50 octet DATA_UNIT)

5.6.1.3 System reaction times

The message rate R_{SYS} in the system equals the possible number of message transfer periods per second:

$$R_{SYS} = 1 / t_{MP};\ t_{MP} = T_{MP} \times t_{BIT} \quad (55)$$

The maximum system reaction time T_{SR} in a system with **one** Master station and n Slave stations (Master-Slave system) is calculated from the message transfer period and the number of Slave stations. If message retries are allowed, T_{SR} is calculated as follows:

$$T_{SR} = np \times T_{MP} + mp \times T_{RMP} \quad (56)$$

where

np is the number of Slave stations

mp is the number of message retry transfer periods.

T_{RMP} is the message retry transfer period

The maximum system reaction time in a system with **several** Master stations and Slave stations equals the target rotation time:

$T_{SR} = T_{TR}$ (see 5.3.2.6) (57)

5.6.1.4 Isochronous cycle time

The isochronous cycle time starts with the transmission of the SYNCH DLPDU by the IsoM Master. The IsoM Master transmits all high priority messages and the configured number of low priority messages. To permit a multi-master environment the token is passed to the next Master station thus leaving other Master stations the necessary time to perform their actions.

$$T_{CT} = T_{SYNCH} + N \times T_{TP} + \sum_{i=1}^{N} P_i + T_{RES} \quad (58)$$

where

N is the number of master stations within token ring

P_i = DLPDU time by master station I (includes isochronous master's cyclic DLPDUs).

The positive difference between the real cycle time (T_{RCT}) and the calculated cycle time (T_{CT}) represents the spare time T_{RES}. Depending on the amount of time available, a number of active spare time messages (ASM) are sent. At least one ASM message shall be sent. After each sending of the ASM message, the spare time shall be calculated. The transmission of the last ASM is succeeded by the start of the passive-spare-timer if the difference between the real cycle time (T_{RCT}) and the calculated cycle time (T_{CT}) is greater than zero. The sending of a new SYNCH DLPDU starts after expiration of the passive-spare-timer. If the difference between the real cycle time (T_{RCT}) and the calculated cycle time (T_{CT}) is less than or equal to zero then the sending of a new SYNCH DLPDU starts immediately. The transmission of a new SYNCH DLPDU marks the beginning of the next cycle. The start of the next cycle is notified to the DLMS-user with a synch notification.

In parallel, the value of isochronous-cycle-timer is read and compared with the calculated cycle time. If the result is zero or within the allowed time shift ($maxT_{SH}$) no message is sent to the DLMS-user. If the result is not in the range of the allowed time shift a Synch_Delay event with the value of the difference between real and calculated cycle time is sent to the DLMS-user.

5.6.2 Synchronous transmission

5.6.2.1 Token transfer period

Similar to asynchronous transmission, the following shall apply for the token transfer period T_{TP}:

$$T_{TP} = T_{TF} + T_{TD} \quad (59)$$

Due to the DLPDU character (see 6.1.2) and the changed DLPDU format (see 7.4.2), the token DLPDU time T_{TF} is 80 bit with a preamble of 16 bit and a post-transmission gap time of 8 bit.

NOTE Only the transmission speed of 31,25 kbit/s may be selected.

5.6.2.2 Message transfer period

The following shall apply for the message transfer period T_{MP} :

$$T_{MP} = T_{S/R} + T_{SDR} + T_{A/R} + T_{PTG} + 2 \times T_{TD} \quad (60)$$

The specifications for the message transfer period stipulated in 5.6.1.2 shall apply except for the following cases:

The PDU transmission times ($T_{S/R}$, $T_{A/R}$) are determined by the number of PhL octets. Thus, they are calculated as follows:

$T_{S/R}$ = a, where “a” is the number of octets in a request DLPDU (≤ 255)

$T_{A/R}$ = b, where “b” is the number of octets in an ack/response DLPDU (≤ 255).

EXAMPLE a = 10, for the request DLPDU: $T_{S/R}$ = 80 bit (additionally to 5.6.1.2: 2 octet Preamble and 2 octet PhL start/end delimiter); b = 63, for the response DLPDU: $T_{A/R}$ = 504 bit (additionally to 5.6.1.2: 2 octet Preamble and 2 octet PhL start/end delimiter)

5.6.2.3 System reaction times

See 5.6.1.3.

5.6.2.4 Isochronous cycle time

See 5.6.1.4.

6 General structure and encoding of DLPDUs, and related elements of procedure

6.1 DLPDU granularity

6.1.1 Asynchronous transmission – UART character

6.1.1.1 General

Each DLPDU shall consist of a number of UART characters (UC). Each UART character is a start-stop character for asynchronous transmission, structured as shown in Figure 13.

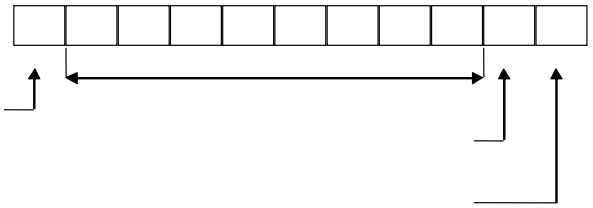


Figure 13 – UART character

The presentation of UART characters is based on ISO/IEC 1177 and ISO/IEC 2022.

6.1.1.2 Transmission rule

Each UART character shall consist of 11 bits: a start bit (ST) which shall be always binary "0", 8 information bits (I) which may be binary "0" or binary "1", an even parity bit (P) which may be binary "0" or binary "1" and a stop bit (SP) which shall be always binary "1".

6.1.1.3 Bit synchronizing

The receiver's bit synchronizing shall always start with the falling edge of the start bit, that is, at the transition from binary "1" to binary "0". The start bit and all consecutive bits shall be scanned in the middle of the bit time. The start bit shall be binary "0" in the middle of the bit,

otherwise the synchronizing shall be regarded as failed and shall be stopped. The synchronizing of the UART character ends with the stop bit being binary "1". If a binary "0" bit is encountered instead of the stop bit, a synchronizing error or UART character error shall be assumed and reported and the next leading edge of a start bit shall be waited for.

A maximum deviation of ± 0,3 % of the nominal data rate (bit-period) for transmission and receipt shall not be exceeded for data rates of less than 1 500 kbit/s. For data rates of 1 500 kbit/s and higher a maximum deviation of ± 0,03 % of the nominal data rate (bit-period) shall not be exceeded.

6.1.2 Synchronous transmission

Each octet of the DLPDU shall be structured as shown in Figure 14 for synchronous transmission.

Figure 14 – Octet structure

6.2 Length octet (LE, LEr)

The two length octets of identical value in the DLPDU header of the format for variable data length shall contain the number of information octets in the DLPDU body. These comprise: DA, SA, FC and the DATA_UNIT. The value shall cover the range from 4 to 249, so that a maximum of 246 octets may be transmitted in a DLPDUs DATA_UNIT (see 6.6). A value < 4 is not permitted, as a DLPDU contains at least DA, SA, FC and **one** DATA octet. The longest DLPDU may contain a total of 255 octets. Figure 15 illustrates the Length octet coding.

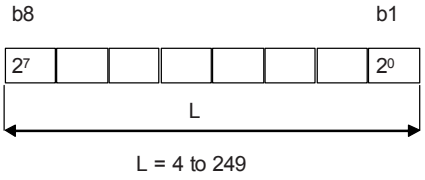


Figure 15 – Length octet coding

6.3 Address octet

6.3.1 Destination and source station address (DA and SA)

The two address octets in the DLPDU header (request, acknowledgement and response DLPDUs) shall contain the destination (DA) and source (SA) station address. The short acknowledgement DLPDU does not contain these two address octets. Figure 16 illustrates the Address octet coding.

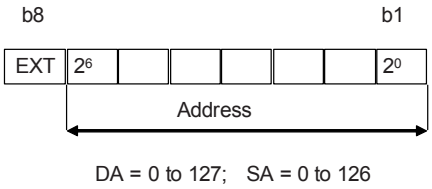


Figure 16 – Address octet coding

Address 127 (b1 to b7 = 1) is reserved as global address for broadcast and multicast messages (DLPDU to all stations or a group of stations selected by means of a DL-service-access-point; it may only occur in SDN and CS, and the response DLPDU of MSRD).

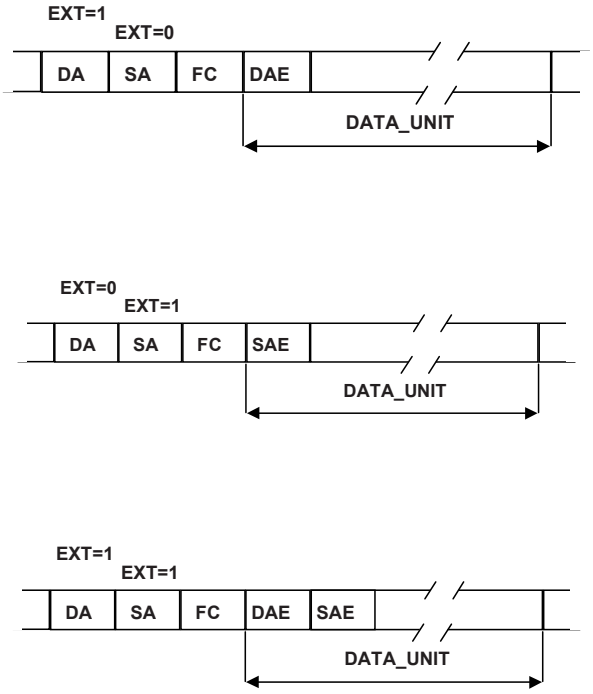
Thus, 127 station addresses (0 to 126) are available for Slave and Master stations, of which preferably no more than 32 may be occupied by Master stations. For non time-critical applications, there may be up to 127 Master stations. As at least one Master station is required, a maximum of 126 addresses for Slave stations is allowed.

Except for MSRD, the send/request DLPDUs address octets shall be sent back mirrored in the acknowledgement or response DLPDU. That is, SA of the acknowledgement or response DLPDU shall contain the destination station address and DA shall contain the source station address of the send/request DLPDU. For MSRD the DA of the response DLPDU shall contain the address 127 and SA of the response DLPDU shall contain the destination station address of the send/request DLPDU.

6.3.2 Address extension (EXT)

Address extensions apply only to DLPDUs with DATA_UNIT. The EXT bit (extension) shall indicate a destination and/or source address extension (DAE, SAE) (see Figure 17), which shall immediately follows the FC octet in the DATA_UNIT. It may be distinguished between access address (DL-service-access-point, DLSAP, see 6.3.4) and region/DL-segment address. Both address types may also occur simultaneously, as each address extension contains an EXT bit again (see bit b8 in Figure 18).

The address extensions of the send/request DLPDU shall be sent back mirrored in the response DLPDU.

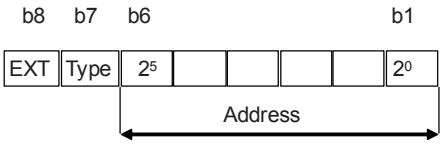


where

EXT = 0: No address extension in the DATA_UNIT

EXT = 1: Address extension follows in the DATA_UNIT.

Figure 17 – DAE/SAE octet in the DLPDU



where

b8 (EXT) denotes an additional address extension:

0: No additional address extension octet

1: One additional address extension octet follows immediately. The following order shall be followed:

First octet: region/DL-segment address with b7=1, b8=1

Second octet: DLSAP with b7=0, b8=0.

b7 denotes the type:

0: 6 bit DL-service-access-point (DLSAP): DAE = 0 to 63; SAE = 0 to 62

1: 6 bit region/DL-segment address to realize hierarchical systems, values are not specified in this standard.

Figure 18 – Address extension octet

6.3.3 Address check

A receiver shall check the destination address information in a DLPDU addressed to itself against TS according to the following rules.

a) If there is no region/DL-segment address in the DLPDU existent (DA[b8=0] or DAE[b7=0]), it shall only check the DA on equality with TS.

b) If there is a region/DL-segment address in the DLPDU existent (DAE[b7=1]), it shall check the DAE and the DA on equality with region/DL-segment address and TS of the addressed station.

c) If the receiver of the station, which is addressed with a region/DL-segment address, does not contain a region/DL-segment address then the DLPDU is not addressed to this station.

6.3.4 DL-service-access-point (DLSAP)

At the DLS interface, (see IEC 61158-3-3) a data transmission service (or several thereof) is processed via a DL-service-access-point (DLSAP). Several DLSAPs may exist at the same time in Master and Slave stations. The related DLSAP-address shall be transmitted together with the message, except for the DLSAP-addresses NIL and CS.

The address extensions DAE and SAE shall be used for the transmission of DLSAPs. The source service access point index (S_SAP_index), which represents the access address of the local user to the DL, shall be transmitted in the SAE octet. The destination service access point index (D_SAP_index), which represents one or all access addresses of the remote user to the DL, shall be transmitted in the DAE octet. S_SAP_index values from 0 to 62 and D_SAP_index values from 0 to 63 may be chosen. The D_SAP_index value 63 denotes the global access address. This D_SAP_index may only be used for the Send Data with No Acknowledge service.

NOTE For DLPDUs with a Function = SDA_H/L or SRD_H/L (see Table 3) a D_SAP_index of 63 in the send/request DLPDU results in a negative acknowledge (RS).

If the transmission of DLSAP addresses is omitted for reasons of DLPDU efficiency, the data transmission services shall be processed via the **default DLSAP** at the initiator or the responder or both. In these cases, all DLPDUs shall be transmitted without the related address extension (SAE or DAE or both). At the DLS interface the default DLSAP is mandatory and is addressed with the value NIL.

The DLSAP-address CS used for the time synchronization services Clock Value and Time Event is omitted in the related DLPDUs for the same reason.

6.4 Control octet (FC)

6.4.1 General

The control octet in the DLPDU header shall indicate the DLPDU type, such as send/request DLPDU and acknowledgement or response DLPDU. In addition, the control octet shall contain the function Code No and the control information, which prevents loss and multiplication of messages, or the station type with the DL status. Figure 19 and Figure 20 illustrate the FC octet coding.

b8	b7	b6	b5	b4	b3	b2	b1
0/1	1	FCB	FCV	2^3	Function code No		2^0

where

b8 = 0/1, b7 = 1: send/request DLPDU

FCB: is Frame count bit: 0 or 1, alternating

FCV: is Frame count bit valid:

0: alternating function of FCB is invalid

1: alternating function of FCB is valid

Function code No: shall be as described in Table 3.

Figure 19 – FC octet coding for send/request DLPDUs

b8	b7	b6	b5	b4	b3	b2	b1
Res	0	Station type and DL status		2^3	Function code No		2^0

where

DLPDU type b7 = 0: is acknowledgement or response DLPDU

b6 and b5 are station type and DL status combined

b6 b5

0 0 Slave station

0 1 Master station not ready to enter logical token ring

1 0 Master station ready to enter logical token ring

1 1 Master station in logical token ring

Res: means reserved (the sender shall set to binary "0", the receiver does not have to interpret this bit)

Function code No: shall be as described in Table 3.

Figure 20 – FC octet coding for acknowledgement or response DLPDUs

Table 3 shows the function code No of the control octet for both send/request DLPDUs and acknowledgement or response DLPDUs.

Table 3 – Transmission function code

Code	Function	Format in subclause described	Possible for the following stations	
	DLPDU type b8 = 1 & b7 = 1	—	Initiator	Receiver/ Responder
0	Clock Value	7.3.1 , 7.3.2	M	M and S
1…15	Reserved	—	—	—
	DLPDU type b8 = 0 & b7 = 1	—	—	—
0	Time Event	7.1.1 , 7.1.2	M	M and S
1,2	Reserved	—	—	—
3	Send data with acknowledge low	7.2.1 , 7.3.1 7.2.2 , 7.3.2	M	M and S
4	Send data with no acknowledge low		M	M and S
5	Send data with acknowledge high		M	M and S
6	Send data with no acknowledge high		M	M and S
7	Send and request data multicast	7.2.1 , 7.3.1 , 7.2.2 , 7.3.2	M	M and S
8	Reserved	—	—	—
9	Request DL-status with reply	7.1.1 , 7.1.2	M	M and S
10, 11	Reserved	—	—	—
12	Send and request data low	7.1.1 , 7.2.1 , 7.3.1	M	M and S
13	Send and request data high	7.1.2 , 7.2.2 , 7.3.2	M	M and S
14	Request ident with reply	7.1.1 , 7.1.2	M	M and S
15	Reserved	—	—	—
	DLPDU type b7 = 0	—	—	—
0	Acknowledgement positive (OK)	7.1.1 , 7.1.2 [a]	M and S	M
1	ACK negative DL/DLM User Error (UE)	7.1.1 , 7.1.2	M and S	M
2	ACK negative no resource for send data (& no response DL data) (RR)		M and S	M
3	ACK negative no service activated (RS)		M and S	M
4 to 7	Reserved	—	—	—
8	Response DL/DLM data low (& send data ok) (DL)	7.2.1 , 7.3.1 7.2.2 , 7.3.2	M and S	M
9	ACK negative no response DL/DLM data, (& send data ok) (NR)	7.1.1 , 7.1.2 [a]	M and S	M
10	Response DL data high, (& send data ok) (DH)	7.2.1 , 7.3.1 7.2.2 , 7.3.2	M and S	M
11	Reserved	—	—	—
12	Response DL data low, no resource for send data (RDL)	7.2.1 , 7.3.1	M and S	M
13	Response DL data high, no resource for send data (RDH)	7.2.2 , 7.3.2	M and S	M
14, 15	Reserved	—	—	—

where M: Master, S: Slave

	b4	b1
Function code No 0:	0	0 0 0
Function code No 15:	1	1 1 1

( ) Value of the L/M_status parameter of the service primitives (see IEC 61158-3-3).

[a] Also possible Short Acknowledgement SC = E5H; exception: if request DL-status with reply, no SC is permitted.

6.4.2 Frame count bit

The frame count bit FCB (b6) prevents the duplication of messages at the responder and the loss at the initiator. However, "Send Data with No Acknowledge" (SDN), "Request DL Status with Reply", "Request Ident with Reply", “Time Event” and “Clock Value” are excluded from this.

In order to manage the security sequence, the initiator shall carry a FCB for each responder. When an send/request DLPDU is transmitted to a responder for the first time or to a responder currently marked as "non operational", the associated FCB shall be set unambiguously. The initiator shall achieve this by an send/request DLPDU with FCV=0 and FCB=1. The responder shall classify such a DLPDU as first message transfer period and store FCB=1 together with the initiator's address (SA and optional SAE[b7=1]) (see Table 4). This message transfer period is not repeated by the initiator.

If a responder supports the region/DL-segment addressing and the send/request DLPDU contains a source region/DL-segment address (SAE[b7=1]), which is unequal to the own region/DL-segment address (DAE[b7=1]), then the responder shall store the SAE[b7=1] together with the SA.

In the following send/request DLPDUs to the same responder the initiator shall set FCV=1 and toggle FCB with each new send/request DLPDU. The responder shall evaluate FCB when receiving an send/request DLPDU addressed to itself with FCV=1. A FCB changed in comparison with the same initiator's (same SA and optional same SAE[b7=1]) preceding send/request DLPDU shall be considered as confirmation of the preceding message transfer period's correct completion. If the send/request DLPDU originates from a different initiator (different SA or optional different SAE[b7=1]), there shall be no evaluation of the FCB. In both cases, the responder shall store the FCB with the source address (SA and optional SAE[b7=1]) until it receives a new DLPDU addressed to itself.

If an acknowledgement or response DLPDU is missing or corrupted, the FCB shall **not** be changed by the initiator in the retry; this indicates the faulty preceding message transfer period. If a responder receives an send/request DLPDU with FCV=1 and the same FCB as in the same initiator's (same SA and optional same SAE[b7=1]) immediately preceding send/request DLPDU, a retry shall be carried out. As a result, the responder shall again transmit the acknowledgement or response DLPDU kept in readiness.

The responder shall keep ready the preceding acknowledgement or response DLPDU for a potential retry, until it detects a confirmation of the preceding message transfer period's correct completion as described above, or until it receives a token DLPDU or a DLPDU with a changed address (SA or DA or optional SAE[b7=1] or DAE[b7=1]).

For "Send Data with No Acknowledge", "Request DL Status with Reply", "Request Ident with Reply", “Time Event” and “Clock Value”, FCV and FCB are both zero for the following DLPDU types; the responder does not need to analyze FCB:

Table 4 – FCB, FCV in responder

b6	b5	←- bit position		
FCB	FCV	Condition	Meaning	Action
0	0	DA = TS/127	Request with no ack Request DL-status with reply Request ident with reply Time event Clock event	Last ack or reply may be deleted
0/1	0/1	DA ≠ TS	Request to other responder	Last ack or reply may be deleted
1	0	DA = TS	First request	FCBM := 1 SAM := SA last Ack or Reply may be deleted
0/1	1	DA = TS SA = SAM FCB ≠ FCBM	New request	last Ack or delete Reply FCBM := FCB have Ack or Reply ready for Retry
0/1	1	DA = TS SA = SAM	Request retry	FCBM := FCB repeat Ack or

b6	b5	←- bit position		
FCB	FCV	Condition	Meaning	Action
		FCB = FCBM		Reply and keep it ready
0/1	1	DA = TS SA ≠ SAM	New initiator	FCBM := FCB SAM := SA have Ack or Reply ready for Retry
—	—	Token-DLPDU	—	last Ack or Reply may be deleted
NOTE FCBM is the stored FCB and SAM is the stored SA.				

6.5 DLPDU content error detection

6.5.1 Asynchronous transmission – frame checksum (FCS)

The 8-bit (1 octet) frame checksum that is required for Hamming distance 4 in a DLPDU shall always immediately precede the end delimiter. The checksum shall be structured as shown in Figure 21.

Figure 21 – FCS octet coding

In DLPDUs of fixed length with no data field (see 7.1.1) the checksum shall be calculated from the two's-complement arithmetic sum of DA, SA and FC (without start and end delimiters).

In DLPDUs of fixed length with data field (see 7.2.1) and in DLPDUs with variable data field length (see 7.3.1) the checksum shall additionally include the DATA_UNIT.

6.5.2 Synchronous transmission – frame check sequence (FCS)

A 16-bit (2 octet) frame check sequence is required. It shall be calculated and appended to the DLPDU as specified in Clause 5, where its Hamming distance properties are also described.

6.6 DATA_UNIT

6.6.1 General

The DATA_UNIT shall consist of the User Data (DLSDU) of the DLS/DLMS-user and optionally of one or more address extension octets (see Figure 22). Up to 4 address extension octets are permissible (see 6.3.1). The User Data comprises a maximum of 242 up to 246 octets depending on the number of used address extension octets.

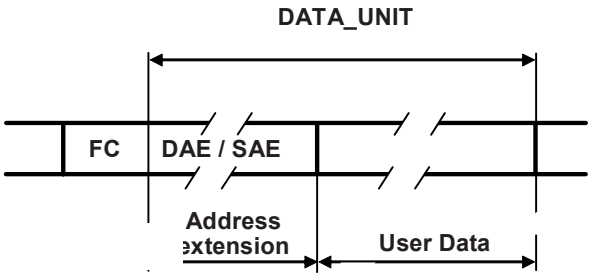
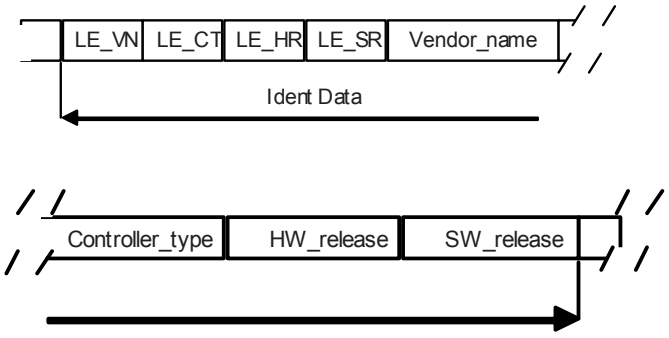


Figure 22 – Data field

The following user data are defined for the Ident DLPDU:

6.6.2 Ident user data

As shown in Figure 23 the Ident user data shall contain the station's Ident_List with vendor name (Vendor_name), fieldbusfieldbus controller type (controller_type) and hardware and software release (HW/SW_release). It comprises a maximum of 200 octets:



where

LE_VN, LE_CT, LE_HR, LE_SR in each case the length of the corresponding data field in Octets (1 octet each, dual coding, significance as in Figure 21).

Vendor_name (VN) is the name of manufacturer as an ASCII string (ISO 7 Bit Code, b8=0).

Controller_type (CT) is the hardware controller type as an ASCII string (ISO 7 Bit Code, b8=0).

HW_release (HR) is the hardware release of controller as an ASCII string (ISO 7 Bit Code, b8=0).

SW_release (SR) is the software release of controller as an ASCII string (ISO 7 Bit Code, b8=0).

Figure 23 – Ident user data

6.7 Error control procedures

6.7.1 Asynchronous transmission

Line protocol errors, for example, character framing errors, overrun errors and parity errors, and transmission protocol errors, for example faulty start delimiters, frame check octets and end delimiters, invalid DLPDU length, response times, etc. shall result in the following station reactions:

A send/request DLPDU or token DLPDU that has been received incorrectly by a station shall not be processed, acknowledged or answered. The initiator shall retry the request after expiration of the slot time. The request shall also be retried if the acknowledgement or response was corrupted. The initiator shall complete a request only after having received a valid response or if the retry (retries) was not (were not) successful (see Table 5). This means that a "Send/Request" shall be kept until the responder confirms its correct receipt by an acknowledgement or response or the retry (retries) was not (were not) successful. In the

same way, a responder shall terminate a "Request" or "Send/Request" only if a new request with altered frame count bit is received or another station is addressed (see 6.4, FCB).

If a station does not acknowledge or respond after retry (retries), it shall be marked as "non operational". When processing the following requests, the initiator shall transmit the request to this station without retry, until the station acknowledges or responds correctly again. After positive acknowledgement, the initiator shall mark again the addressed station as "operational". When processing the next request, the initiator shall continue the original mode of operation with this station.

6.7.2 Synchronous transmission

Errors in the line protocol (see Clause 9 of IEC 61158-2) and in the medium access protocol, such as erroneous start octets and FCS octets, DLPDU length, response times etc. shall result in the station reactions specified in 6.7.1.

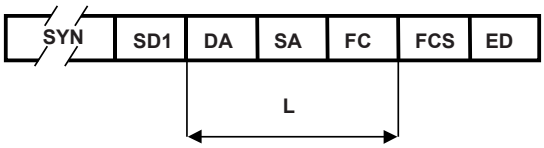
7 DLPDU-specific structure, encoding and elements of procedure

7.1 DLPDUs of fixed length with no data field

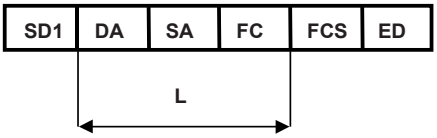
7.1.1 Asynchronous transmission

The formats of DLPDUs of fixed length with no data field shall be as shown in Figure 24.

a) Format of the request DLPDU:



b) Format of the acknowledgement DLPDU:



c) Format of the short acknowledgement DLPDU:

where

SYN is the synchronization period, a minimum of 33 line idle bits

SD1 is the start delimiter, value: 10H

DA is the destination address

SA is the source address

FC is the frame control

FCS is the frame checksum

ED is the end delimiter, value: 16H

L is the information field length, fixed number of octets: L = 3

SC is the single character, value: E5H.

Figure 24 – DLPDUs of fixed length with no data field

Transmission Rules

1) Line idle state shall correspond to signalling level binary "1".
2) Each request DLPDU shall be preceded by at least 33 line idle bits (synchronization time).
3) No idle states are allowed between a DLPDUs UART characters.
4) The receiver shall check:
   – for each UART character: start bit, stop bit and parity bit (even),
   – for each DLPDU: start delimiter, DA, SA, FCS and end delimiter,
   – and the synchronization time in case of a request DLPDU.

   If the check fails, the entire DLPDU shall be discarded.

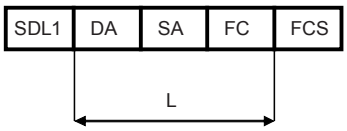
SC and SD1 (as well as SD2 and SD3, see 7.2.1 and 7.3.1) have Hamming distance Hd=4 and are safe against being shifted (see for example IEC 60870-5-1), that is, the single character SC appears to be a DLPDU with Hd=4.

For requests to be acknowledged (Send Data with Acknowledge), only SC is a permissible positive acknowledgement. For requests to be answered (Send and Request Data with Reply), SC is permissible if no data is available (see Table 3, b7=0, Code-No 9).

7.1.2 Synchronous transmission

The formats of DLPDUs of fixed length with no data field shall be as shown in Figure 25.

a) Format of the request DLPDU and format of the acknowledge DLPDU:



b) Format of the short acknowledgement DLPDU:

where

SDL1 is the start octet 1 (start delimiter 1 data link), code: 10H

SDL5 is the start octet 5 (start delimiter 5 data link), code: E5H

DA is the destination address

SA is the source address

FC is the frame control

FCS is the frame check sequence, 2 octets

L is the information field length, fixed number of octets: L = 3.

Figure 25 – DLPDUs of fixed length with no data field

Transmission Rules

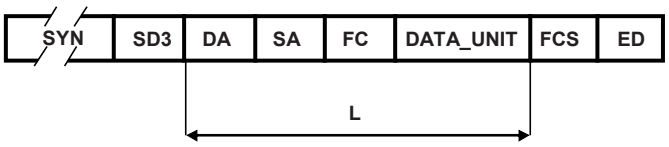
In addition to the transmission rules of the Ph-layer in Clause 9 of IEC 61158-2, the receiver shall check the SDL, DA/SA and FCS octets for each DLPDU. If the check fails, the whole DLPDU shall be discarded.

7.2 DLPDUs of fixed length with data field

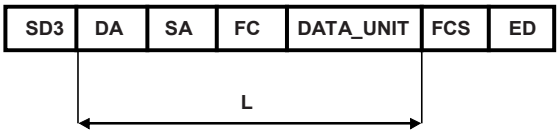
7.2.1 Asynchronous transmission

The format of DLPDUs of fixed length with data field shall be as shown in Figure 26.

a) Format of the send/request DLPDU:



b) Format of the response DLPDU:



where

SYN	is the synchronization period, a minimum of 33 line idle bits
SD3	is the start delimiter, value: A2H
DA	is the destination address
SA	is the source address
FC	is the frame control
DATA_UNIT	is the data field, fixed length (L-3) = 8 octets
FCS	is the frame checksum
ED	is the end delimiter, value: 16H
L	is the information field length, fixed number of octets: L = 11.

Figure 26 – DLPDUs of fixed length with data field

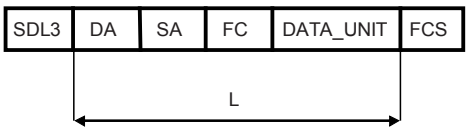
Transmission Rules

The same transmission rules shall apply as for DLPDUs of fixed length with no data field (see 7.1.1).

7.2.2 Synchronous transmission

The formats of DLPDUs of fixed length with data field shall be as shown in Figure 27.

a) Format of the send/request and response DLPDUs:



where

SDL3	is the start delimiter 3 data link, code: A2H
DA	is the destination address
SA	is the source address
FC	is the frame control
DATA_UNIT	is the data field, fixed length (L-3) = 8 octets
FCS	is the frame check sequence, 2 octets
L	is the information field length, fixed number of octets: L = 11.

Figure 27 – DLPDUs of fixed length with data field

Transmission Rules

The same transmission rules shall apply as for DLPDUs of fixed length with no data field (see 7.1.2).

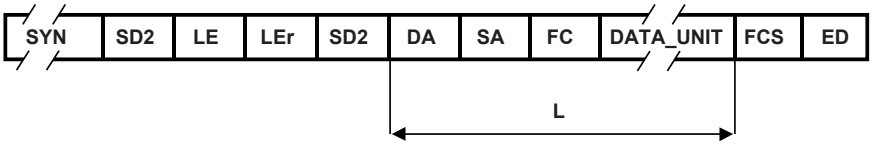
7.3 DLPDUs with variable data field length

7.3.1 Asynchronous transmission

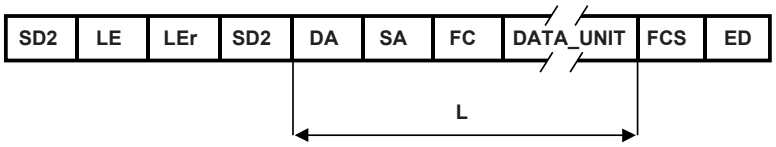
For a variable number of data octets the length information shall also be transmitted in the DLPDU. This length information shall be contained twice in a fixed DLPDU header at the beginning of the DLPDU. Thus it is protected with Hd = 4 and safe against slip.

The format of DLPDUs with variable data field length shall be as shown in Figure 28.

a) Format of the send/request DLPDU:



b) Format of the response DLPDU:



where

SYN	is the synchronization period, a minimum of 33 line idle bits
SD2	is the start delimiter, value: 68H
LE	is the octet length, allowed values: 4 to 249
LEr	is the octet length repeated
DA	is the destination address
SA	is the source address
FC	is the frame control
DATA_UNIT	is the data field, variable length (L-3), max. 246 octets
FCS	is the frame checksum
ED	is the end delimiter, value: 16H
L	is the information field length, variable number of octets: L = 4 to 249.

Figure 28 – DLPDUs with variable data field length

Transmission Rules

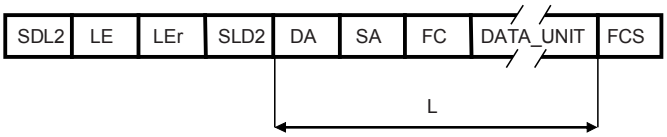
The same transmission rules as for DLPDUs of fixed length with no data field shall be applied (see 7.1.1).

In addition to transmission rule 4, LE shall be identical to LEr, and the information octets shall be counted from the destination address (DA) up to the frame checksum (FCS) and the result shall be compared with LE.

7.3.2 Synchronous transmission

The formats of DLPDUs with variable data field length shall be as shown in Figure 29.

a) Format of the send/request and response DLPDUs:



where

SDL2	is the start delimiter 2 data link, code: 68H
LE	is the length, value: 4 to 249
LEr	is the length (repeated)
DA	is the destination address
SA	is the source address
FC	is the frame control
DATA_UNIT	is the data field, variable length (L-3), max. 246 octets
FCS	is the frame check sequence, 2 octets
L	is the information field length, variable number of octets: L = 4 to 249.

Figure 29 – DLPDUs with variable data field length

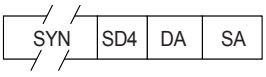
Transmission Rules

The same transmission rules as for DLPDUs of fixed length with no data field shall apply (see 7.1.2). In addition, the receiver shall check if LE and LEr coincide. The information octets shall be counted from the destination address (DA) up to the frame check sequence (FCS) and shall be compared with LE.

7.4 Token DLPDU

7.4.1 Asynchronous transmission

The format of the token DLPDU shall be as shown in Figure 30.



where

SYN	is the synchronization period, a minimum of 33 line idle bits
SD4	is the start delimiter, value: DCH
DA	is the destination address
SA	is the source address.

Figure 30 – Token DLPDU

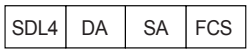
Transmission Rules

1) Line idle state shall correspond to signalling level binary "1".
2) Each token DLPDU shall be preceded by at least 33 line idle bits (synchronization time).
3) No idle states are permitted between a DLPDUs UART characters.
4) The receiver shall check:
   – per UART character: start bit, stop bit and parity bit (even),
   – per DLPDU: synchronization time, start delimiter and DA/SA.

   If the result of the check is negative, the whole DLPDU shall be discarded.

7.4.2 Synchronous transmission

The format of the token DLPDU shall be as shown in Figure 31.



where

SDL4	is the start delimiter 4 data link, code: DCH
DA	is the destination address
SA	is the source address
FCS	is the frame check sequence, 2 octets.

Figure 31 – Token DLPDU

Transmission Rules

The same transmission rules as for DLPDUs of fixed length with no data shall apply (see 7.1.2).

7.5 ASP DLPDU

The ASP DLPDU is generated by the DLE in the isochronous mode to produce bus activity in the spare time of an isochronous cycle to avoid a time-out (T_{TO}). The format of the ASP DLPDU is the same as described in 7.1, with the following restrictions:

– Control Octet (FC) is equal to Request DL Status with Reply (Code 49H)

– DA is equal to SA

– there is no response DLPDU by any DLE for this ASP DLPDU.

7.6 SYNCH DLPDU

The SYNCH DLPDU is generated by the DLE in the isochronous mode to mark the beginning of a new isochronous cycle. The format of the SYNCH DLPDU is the same as described in 7.3, with the following restrictions:

– FC, the Control Octet, is equal to Send Data with No Acknowledge high (Code 46H)

– DA is equal to 127

– SAE is equal to 62

– DAE is equal to 58

– the DATA_UNIT shall contain the value of the operating parameter SYNCHT

– there is no response DLPDU by any DLE for this SYNCH DLPDU.

7.7 Time Event (TE) DLPDU

The TE DLPDU is used to synchronize the clock at the time receiver. The format of the TE DLPDU is the same as described in 7.1, with the following restrictions:

– FC, the Control Octet, is equal to Time Event (Code 40H)

– DA is equal to 127

– there is no response DLPDU by any DLE for this TE DLPDU.

7.8 Clock Value (CV) DLPDU

The CV DLPDU is used to transmit the clock value to the time receivers. The format of the CV DLPDU is the same as described in 7.3, with the following restrictions:

– FC, the Control Octet, is equal to Clock Value (Code C0H)

– DA is equal to 127

– SAE and DAE are not used

– there is no response DLPDU by any DLE for this CV DLPDU.

7.9 Transmission procedures

7.9.1 Asynchronous transmission

Permissible DLPDU sequences (message transfer periods) for asynchronous transmission are described in Figure 32 through Figure 34. Error sequences and broadcast/multicast messages are excluded.

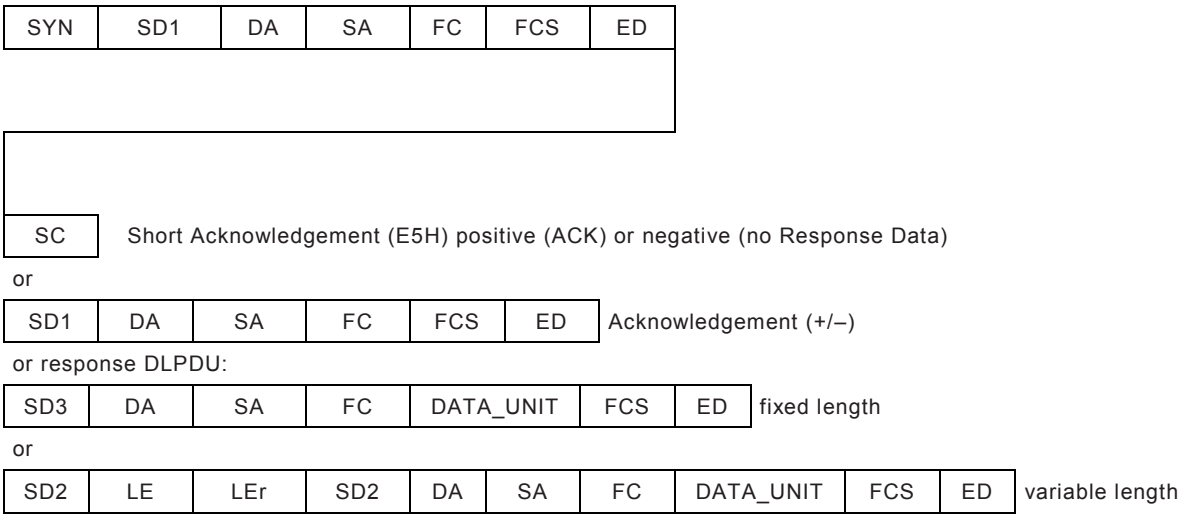


Figure 32 – Send/request DLPDU of fixed length with no data

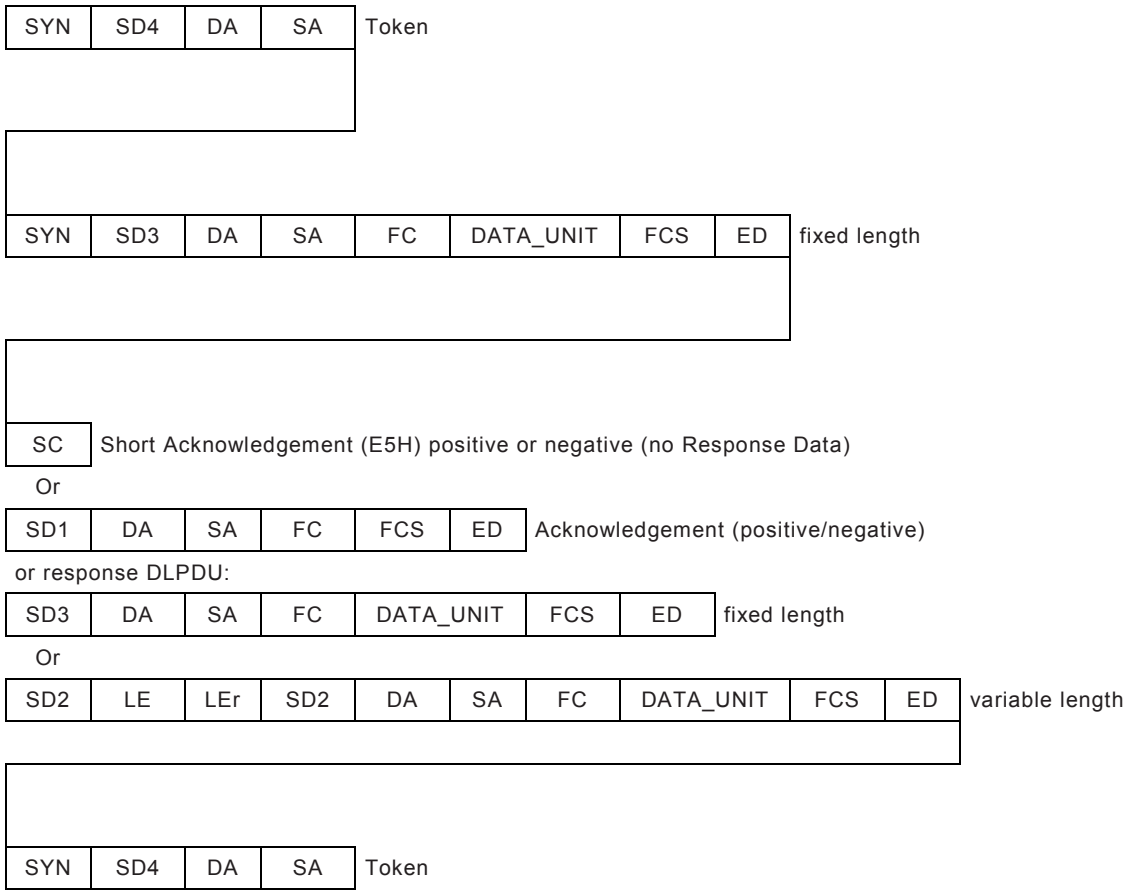


Figure 33 – Token DLPDU and send/request DLPDU of fixed length with data

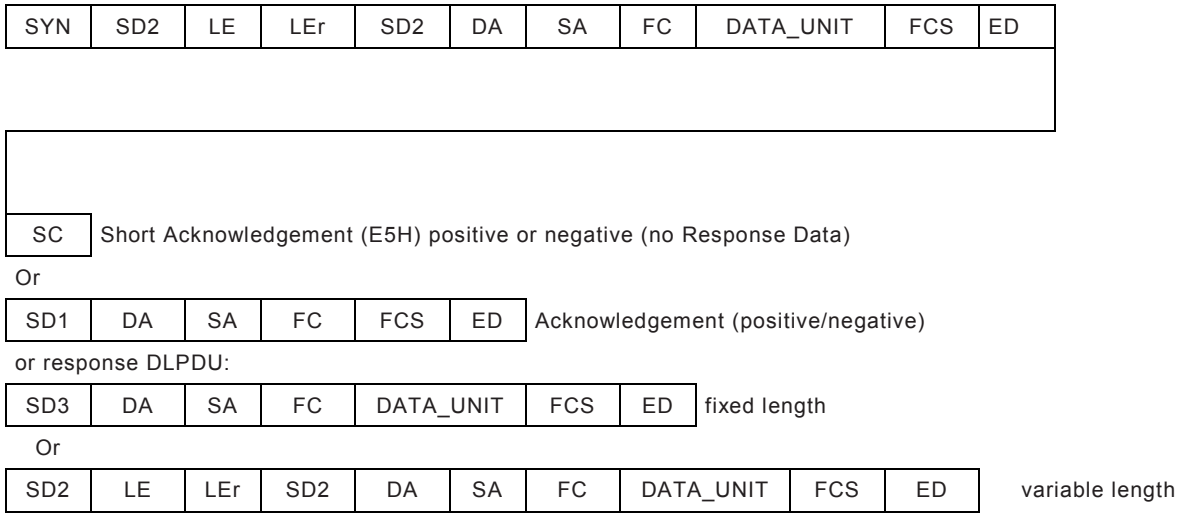


Figure 34 – Send/request DLPDU with variable data field length

7.9.2 Synchronous transmission

Permissible DLPDU sequences (message transfer periods) for synchronous transmission are described in Figure 35 through Figure 37. Error sequences and broadcast/multicast messages are excluded.

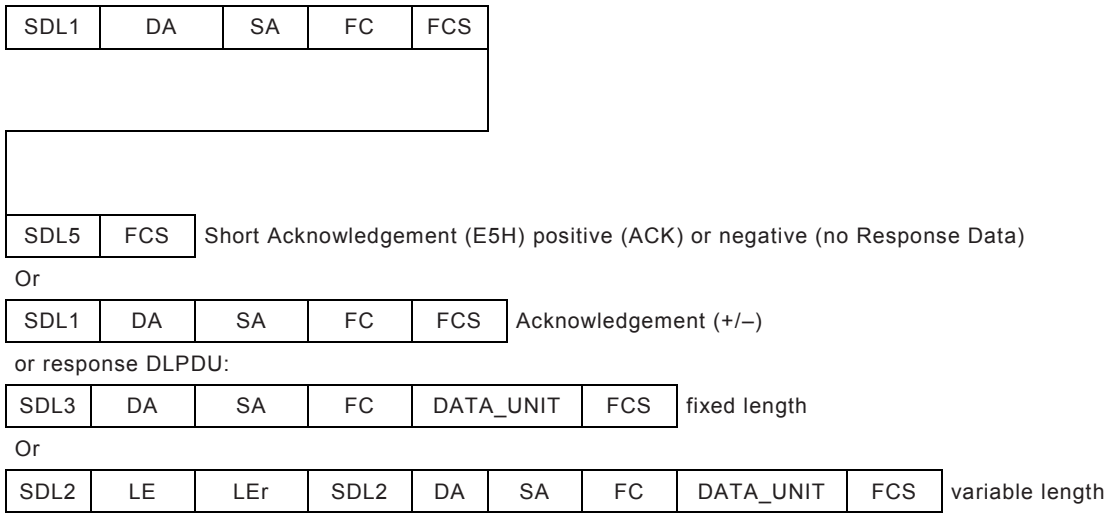


Figure 35 – Send/request DLPDU of fixed length with no data

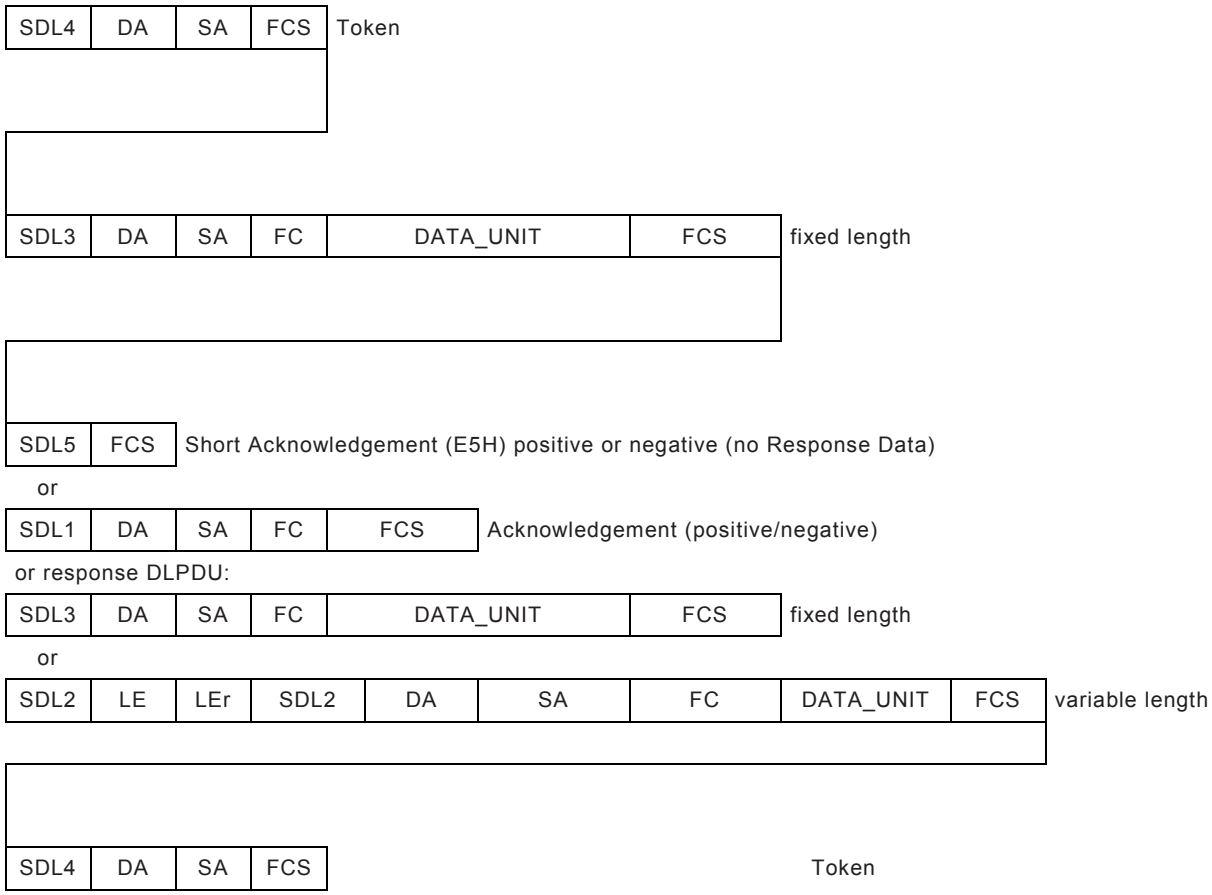


Figure 36 – Token DLPDU and send/request DLPDU of fixed length with data

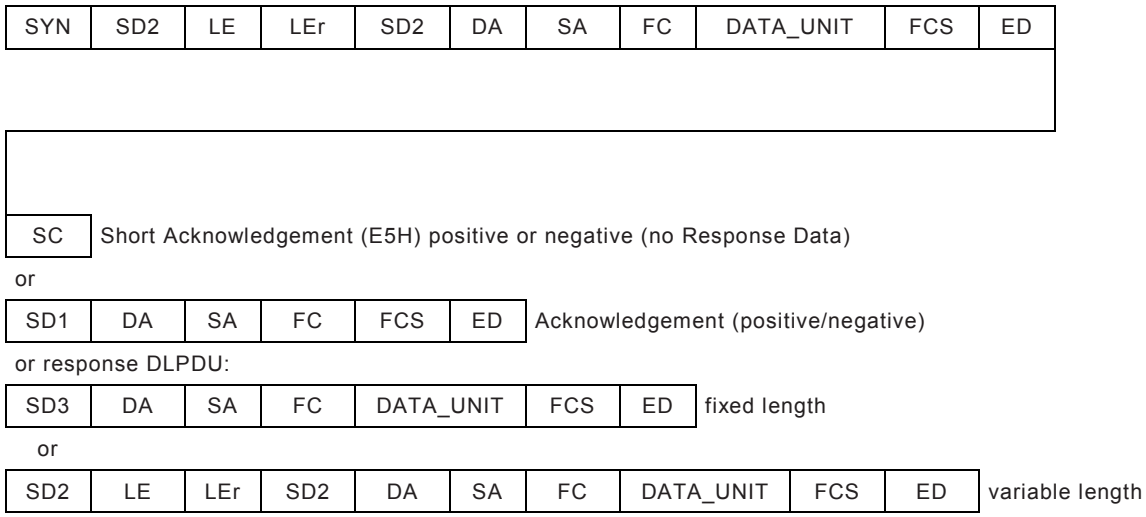


Figure 37 – Send/request DLPDU with variable data field length

8 Other DLE elements of procedure

NOTE Annex A specifies a number of finite state machines used by the DLE to provide its low-level and high-level protocol functions. The specification of Annex A is complementary to the textual specification in this and related clauses in the body of this standard. In case of conflict the requirements of Annex A take precedence.

8.1 DL-entity initialization

After power on (PON) the DL-entity of Master and Slave stations shall enter the "Offline" state. Within this state, the DL-entity does not receive or transmit any signals (DLPDUs) from or to the bus.

The DL-entity may enter the "Passive_Idle" or "Listen_Token" state from the "Offline" state only, if the operating parameters (DL-variables) have been set for correct protocol handling (see IEC 61158-3-3). The operating parameters, shown in Table 5, shall be provided by the DLMS-user.

Table 5 – Operating parameters

Parameter number	Name
1	Station DL-address TS
2	Data rate (kbit/s)
3	Single/redundant media available
4	Hardware release
5	Software release
6	Slot time T_{SL}
7	Station delay time min T_{SDR}
8 (see Note 1)	Station delay time max T_{SDR}
9 (see Note 1)	Transmitter fall/Repeater switch time T_{QUI}
10 (see Note 1)	Setup time T_{SET}
11 (see Note 1)	Target rotation time T_{TR}
12 (see Note 1)	GAP update factor G
13 (see Note 1)	Master station enter/leave the logical ring (in_ring_desired)
14 (see Note 1)	Highest station address (HSA)
15 (see Note 1)	Maximum number of retries (max_retry_limit)

Parameter number	Name
16 (see Note 2)	Clock synchronization interval T_{CSI}
17 (see Note 3)	Contents of the SYNCH User Data (SYNCHT)
18 (see Note 3)	Isochronous cycle time T_{CT}
19 (see Note 3)	Allowed maximal time shift T_{TS}

NOTE 1 This applies only to Master stations.

NOTE 2 This applies only to stations able to support clock synchronization.

NOTE 3 This applies only to stations able to work in isochronous mode.

8.2 States of the media access control of the DL-entity

8.2.1 General

A Master station's media access control state machine (MAC) is described by means of 11 states and the transitions between them. A Slave station's MAC has 2 states.

Figure 38 shows as an overview the combined state diagram of the Master (state 0 to 9 and 11) and the Slave station (state 0 and 10).

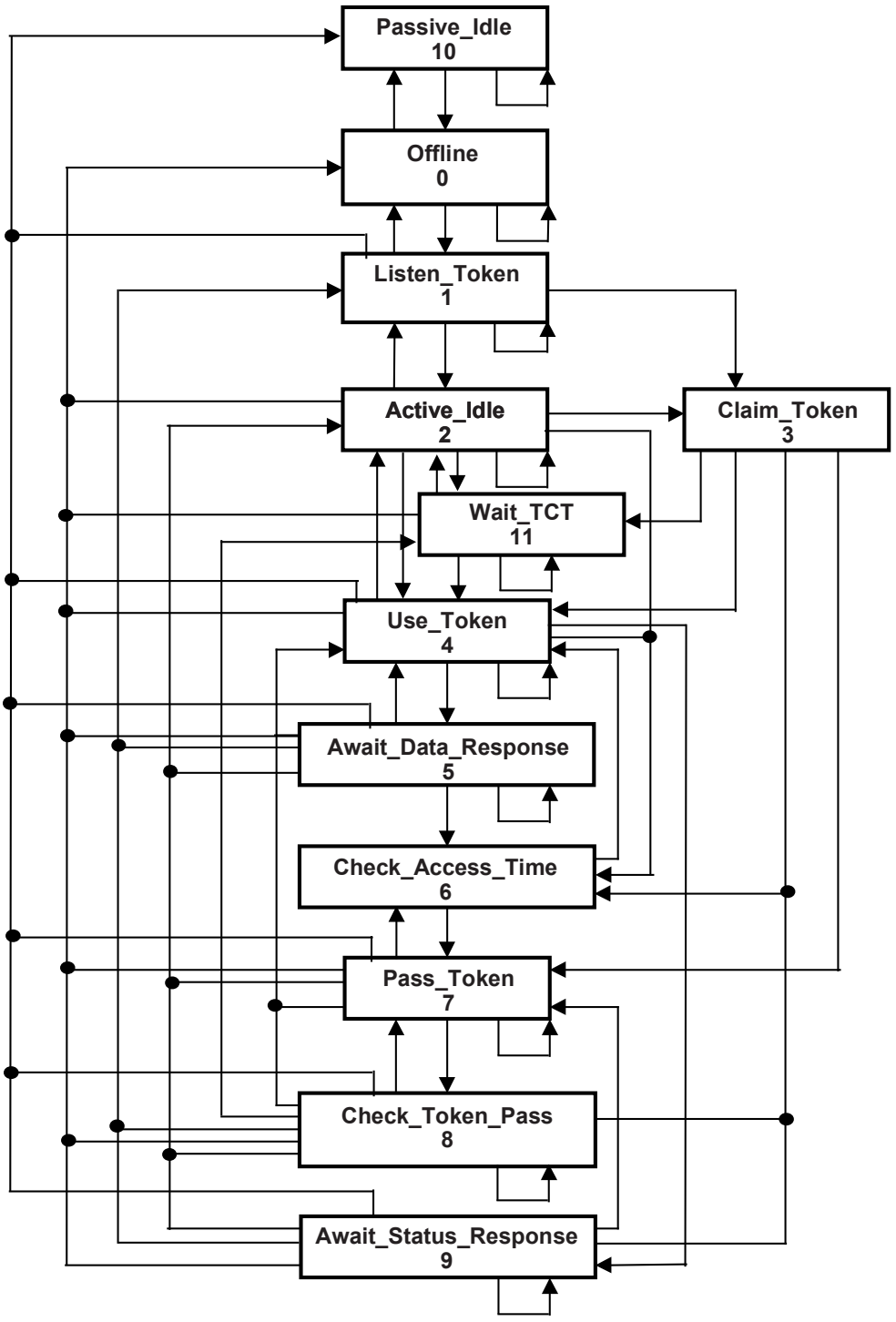


Figure 38 – DL-state-diagram

8.2.2 Offline

The "Offline" state shall be entered immediately after power on, after the DLM-RESET service, or after certain error conditions have been detected. After power on each station should perform a self-test. This internal self-test depends on the implementation and does not influence the other stations. For this reason, the self-test procedure is not specified in this specification.

After terminating the power on sequence, the MAC remains in the "Offline" state until all required operating parameters (see 8.1) have been initialized. The DL-entity may only afterwards connect to the transmission medium, but without becoming active.

8.2.3 Passive_Idle

After its parameters are initialized, the Slave station's MAC or the Master station's MAC depending on the operating parameter "in_ring_desired" (in_ring_desired equal false) (see Table 5) shall enter the "Passive_Idle" state and listen to the line. If a plausible send/request DLPDU addressed to that station is received, the DL-entity shall acknowledge or respond as required, except for DLPDUs with global address (broadcast message, see 6.3.1) and token DLPDUs addressed to itself. The token DLPDU is discarded.

At the occurrence of the DLM-RESET service the MAC re-enters the "Offline" state.

8.2.4 Listen_Token

After its operating parameters have been initialized and the value of the parameter "in_ring_desired" is true, the Master station's MAC shall enter the "Listen_Token" state. In this state the Master station's DL-entity shall monitor the line in order to identify those Master stations which are already in the logical token ring. For that purpose, token DLPDUs are analyzed and the station addresses contained in them are used to generate the list of Master stations (LMS). After listening to two complete token rotations, the MAC shall enter the state "Active_Idle".

During LMS generation a "Request DL Status with Reply" responds with "Master station not ready" after one token rotation is completed. All other DLPDUs are not processed in the "Listen_Token" state, that is, they are neither acknowledged/responded nor indicated to the local DLS-user.

If the MAC detects its own address as source address (SA) in two token DLPDUs when registering the Master stations, it shall assume that another Master station with the same address exists already in the ring. The MAC shall then re-enter the "Offline" state and report this event to the local DLMS-user.

If the MAC observes no bus activity for the time-out period, it shall assume that an initialization or a restoration of the logical token ring is necessary. The MAC shall enter the "Claim_Token" state.

8.2.5 Active_Idle

On leaving the "Listen_Token" state, the Master station's MAC shall enter the "Active_Idle" state and shall wait for received messages. If it receives an send/request DLPDU addressed to itself or as broadcast, it shall proceed, acknowledge or respond as required and shall unfreeze the token-rotation-timer if the token-rotation-timer is frozen.

The "Listen_Token" state shall be entered in the case of an error, if two token DLPDUs with SA = TS are received in immediate succession. This event shall be reported to the local DLMS-user.

If the MAC find out that it was taken from the logical token ring not on its own initiative, it also shall enter in the "Listen_Token" state and report this (Out_of_ring) to the local DLMS-user. The "Listen_Token" state shall also be entered in case of the MACs LMS is not ready, that is, the SA and DA in the token DLPDU are not compliant with the LMS entry for a number of Tokens receipts. Token DLPDUs, which indicate the addition and removal of a Master, shall not be counted as not compliant.

If the MAC observes no bus activity for the time-out period, it shall assume that a restoration of the logical token ring is necessary. The MAC shall try to claim the token and to (re)initialize the logical ring ("Claim_Token" state).

If the MAC is addressed by a "Request DL Status with Reply" transmitted by its predecessor (PS) then:

a) if the DL-address (TS) is contained in the LMS of the MAC, the MAC shall respond with "Master station in logical ring", or

b) if the DL-address (TS) is not contained in the LMS of the MAC, the MAC shall respond with "Master station ready to enter logical token ring".

After receiving a token DLPDU addressed to the MAC itself and the isochronous mode is not set, it shall

1) unfreeze the token-rotation-timer if this timer is frozen,

2) drop the token, if the token is not sent by the predecessor or if the DL-address (TS) is not contained in the LMS of the MAC. In these both cases, the MAC shall update the LMS,

3) otherwise, by processing the transition the T_{RR} (real rotation time) shall be calculated as the difference between target rotation time and the read value of the token-rotation-timer and the token-rotation-timer shall be restarted with target rotation time. The MAC shall enter:

   i) the "Use_Token" state, if a high priority message is available, or

   ii) the "Check_Access_Time" state if no high priority message is available.

After receiving a token DLPDU addressed to the MAC itself and the isochronous mode is set, it shall

1) send an ASP message, and

2) change into the "Wait_TCT" state.

If the MAC receives an ASP message and the isochronous mode is set, it shall freeze the token-rotation-timer.

8.2.6 Claim_Token

The MAC shall enter the "Claim_Token" state after the "Active_Idle" state or the "Listen_Token" state, when its time-out has expired. In this state it shall try to initialize the logical ring or to start an initialization.

When re-initializing and the MAC is not in the isochronous mode, the DL-entity address is contained in the LMS of the MAC, and thus the "Use_Token" state is entered immediately, if high priority message are pending. Otherwise the MAC shall enter the "Check_Access_Time" state.

When re-initializing and the MAC is in the isochronous mode, the DL-entity address is contained in the LMS of the MAC, and thus the MAC shall send an ASP message and shall enter the "Wait_TCT" state.

When initializing (the DL-address (TS) is not contained in the LMS of the MAC), at first the token shall be addressed twice to the own DL-entity, that is, NS = TS, namely in the "Pass_Token" state. This is necessary in order to cause an entry in the other Master stations' LMS.

8.2.7 Wait_TCT

This state shall be entered after the "Claim_Token" state, "Check_Token_Pass" state or "Active_Idle" state when a ASP message was passed if the Master is in the isochronous mode. In this state the MAC shall pass as much ASP messages as spare time is available ($T_{RCT} < (T_{CT}-T_{PSP})$). If no time available to pass an ASP messages then the MAC shall

a) load the passive-spare-time-timer with $T_{CT}–T_{RCT}$,

b) start the passive-spare-time-timer,

c) wait until the passive-spare-time-timer expires, and

d) pass a SYNCH message for synchronization purposes.

The MAC shall indicate the sending of the SYNCH message to the local DLMS-user by a Synch event. If the SYNCH message could not be sent within a defined maximum time shift ($maxT_{SH}$) then the MAC shall pass a Synch_Delay event to the local DLMS-user in addition.

8.2.8 Use_Token

The MAC enters the "Use_Token" state in order to process high priority or low priority message transfer periods.

The MAC shall enter the

a) "Await_Data_Response" state after each transmitted send/request DLPDU with acknowledgement or response and start the slot-timer (see 5.5.5), or

b) "Check_Access_Time" state in the case of a SDN or CS service is transmitted, or

c) "Await_Status_Response" state in the case of a "Request DL Status with Reply" is transmitted.

8.2.9 Await_Data_Response

This state shall be entered in the case of the transmission of requests with acknowledgement or response. The MAC waits **one** slot time for the receipt of the acknowledgement or response DLPDU.

The MAC shall enter the

a) "Check_Access_Time" state in order to check the available token holding time for processing potential further requests, if:

   1) a valid acknowledgement DLPDU or valid response DLPDU addressed to the request's initiator or

   2) after the retry (retries) no valid acknowledgement or response is received and no more retry is possible. In this case the MAC shall notify the DLS-user accordingly. Further requests to this station are not repeated in case of errors, until a correct message transfer period (send/request data) has been performed;

b) "Use_Token state, if:

   1) an invalid DLPDU is received (start-, end-, length-, FCS error; start-, stop-, parity-bit error or slip error) and a retry is possible, or

   2) slot time expires (see 5.5) and a retry is possible;

   and shall retry the transmission of the send/request DLPDU.

Receipt of another valid DLPDU indicates that an error has occurred. The MAC shall enter the "Active_Idle" state and discard the received DLPDU.

8.2.10 Check_Access_Time

In this state the available token holding time shall be computed (see 5.5.5). Only if there is still token holding time available, the MAC may re-enter the "Use_Token" state. Otherwise the MAC shall enter the "Pass_Token" state.

If token holding time is available then the MAC shall transmit:

a) if high priority message are pending a high prior send/request DLPDU and shall enter the "Use_Token" state, or

b) if low priority message are pending a low prior send/request DLPDU and shall enter the "Use_Token" state, or

c) when the GAP update time has expired, but there is still token holding time T_{TH} available, then the MAC shall try to record one possible new station in the GAP before passing the token, in order to include it in the logical ring, if necessary. For that purpose, the MAC transmits a "Request DL Status with Reply" and enters the "Use_Token" state.

8.2.11 Pass_Token

In the "Pass_Token" state the MAC shall try to pass the token to the next station (NS) in the logical ring. When transmitting the token DLPDU, the MAC shall check by simultaneous monitoring if the transceiver is working correctly. If it does not receive its own token DLPDU, there is a fatal error in the transmit or receive channel. The MAC shall stop its activity in the logical ring, enter the "Offline" state and notify the DLMS-user.

If the MAC receives its own token DLPDU corrupted, this may be caused by a temporarily defective transmitter, receiver, or by the bus line. This error condition does not result in stopping activities in the first instance, instead the MAC shall enter the "Check_Token_Pass" state as after receiving the token DLPDU correctly.

Only after the token DLPDU has been retransmitted (due to no reaction from NS) and monitored as incorrect, the MAC shall stop its activity in the logical ring, enter the "Offline" state and notify the DLMS-user.

If the MAC has sent the token successfully then it shall enter the "Check_Token_Pass" state.

Only if no successor is known, that is, this DL-entity is at the moment the only active station on the bus, it shall pass the token to itself and then re-enter:

a) the "Use_Token" state if a high priority message available and has been sent or

b) the "Check_Access_Time" state if no high priority message available.

8.2.12 Check_Token_Pass

"Check_Token_Pass" is the state in which the MAC waits **one** slot time for a reaction of the station to which it has passed the token. This waiting time allows for the delay between the receipt of the token DLPDU and the ensuing transmission reaction of the addressed station.

If the MAC detects a valid DLPDU header within a slot time, it shall assume that the token passing was successful. The DLPDU shall be processed as if it were received in the "Active_Idle" state.

If an invalid DLPDU is detected within the slot time, the MAC shall assume that another station is active and therefore also enter the "Active_Idle" state.

If the MAC does not receive any DLPDU within one slot time, it shall pass the token to the successor again and re-enter the "Pass_Token" state and react as described in 5.3.2.1.

If the MAC does not receive any DLPDU within one slot time and it was the second pass token to the same successor then the MAC shall remove its NS from the LMS, pass the token to the new successor (NS of NS) and re-enter the "Pass_Token" state and react as described in 5.3.2.1.

If the MAC receives a Token with DA equal TS and the isochronous mode is active, then the MAC shall send an ASP message and shall enter the "Wait_TCT" state.

8.2.13 Await_Status_Response

This state is entered from the "Use_Token" state during GAP maintenance. In this state the MAC shall wait **one** slot time for an acknowledgement DLPDU.

The MAC shall enter the "Check_Access_Time" state:

a) if nothing is received during the slot time, or

b) a corrupted DLPDU is received.

If an existing station does not answer it shall be deleted from the GAPL, that is, be marked as an unused address. Then the MAC shall enter the "Check_Access_Time" state.

If a status request is answered by a new Slave station or a Master station that does not want to be included in the ring, that station shall be entered in the GAPL.

If the MAC receives a valid acknowledgement with "Master station ready to enter logical token ring" then a new Master station acknowledges that it wants to be included in the logical ring. The MAC shall shorten the own GAP to the new NS and shall pass the token to this station and enter the "Pass_Token" state.

If the MAC receives any other DLPDU instead of an acknowledgement DLPDU (indicates that multiple tokens may exist), it shall enter the "Active_Idle" state.

8.3 Clock synchronization protocol

8.3.1 Overview

Synchronization of clocks between devices on a fieldbus segment and a time master application is provided in parallel with other communication functions. A time master is a fieldbus master device. The scheme used is a "backwards time based correction". This results in very few real-time constraints being imposed on a field device. There is no requirement for generating messages at precise instants. Instead, knowledge of when a special timer event message has been broadcasted is subsequently distributed and used to calculate appropriate clock adjustments.

There is no confirmation of correct reception at the remote DLE, as neither acknowledgements are given nor local retries take place. Once the data is sent it reaches all remote DLEs at the same time (not taking into account signal propagation time, Physical Layer delays and Data Link Layer delays). For the correct and secure time calculation the time transfer will take place in a sequence of two messages: the first transfer (Time Event) and the second transfer (Clock Value). By receiving the Timer Event, every remote DLE starts its receive-delay-timer. By receiving the clock value, the value of the receive-delay-timer is send within the indication. Each addressed remote DLE that has received the clock value error-free passes it to the DLS-user by means of a DL-CS-CLOCK-VALUE indication primitive (see IEC 61158-3-3 fieldbus connectionless-mode Data Link Service, Clock Synchronization service). If errors occur by a violation of the sequence (e. g. a Timer Event is received twice or expiration of the Clock Synchronization Interval Timer), this error is notified to the DLS-user.

8.3.2 State machine time master

The clock synchronization sequence consists of two messages broadcasted by the time master. When the first message, called Time Event, is broadcast, all of the DLEs receiving it start a receive-delay-timer. The time master then sends a second message, the Clock Value, which contains the actual time when the Time Event was sent. Upon reception of this message, the remote DLEs can calculate their receive delay time value. In conjunction with the received clock value each DLS-users is able to back calculate the Time Event message reception time according to its own clock, and compare it with the value distributed in the clock message. Their clocks may then be adjusted to agree with the time master's. In order to support scheduling and to check that message pairs Time Event / Clock Value are matched, Clock Value also contain the time at which the last clock synchronization occurred, which is the time sent at the last Time Event.

Special DL services exist to support clock synchronization. A station's clock is adjusted by the DLS-user, but precise measurement of the time when sending or receiving the Time Event must be done in the DLE. Therefor the DLE uses special timers to measure the send delay respective receive delay.

On the sending side, the DLE in the time master starts its DL timer – the send-delay-timer – by receiving a DL-CS-TIME-EVENT request from the DLS-user. The DLE will send a TE DLPDU and stop the send-delay-timer when the last bit is sent. It will then deliver a confirmation to the DLS-user that the message was sent, including a service parameter which reports the calculated value of the send delay time. The DLS-user then adds this value to the time to determine when the Time Event was actually sent. A Clock Value DLPDU which includes this time is then broadcast to remote DLEs.

On the receiving side, the DLE starts its timer – the receive-delay-timer – when it receives the TE DLPDU. When the clock value is received (CV DLPDU), the receive-delay-timer is stopped and an indication is delivered to the DLS-user together with the calculated receive delay time to determine when the time message was received according to its clock.

The clock synchronization sequence supports optionally redundant time masters. Although only one is normally active at a given time, when switchover occurs more than one time master may be broadcasting the Time Event and Clock Value messages. Therefore, when the Time Event is received, the source station address is saved by the receiving DLEs. CV DLPDUs from other masters will be ignored.

Figure 39 illustrates an overview of the Clock Synchronization.

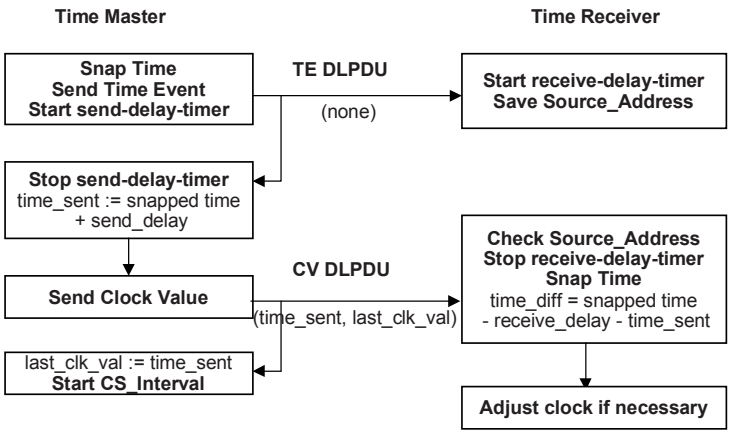


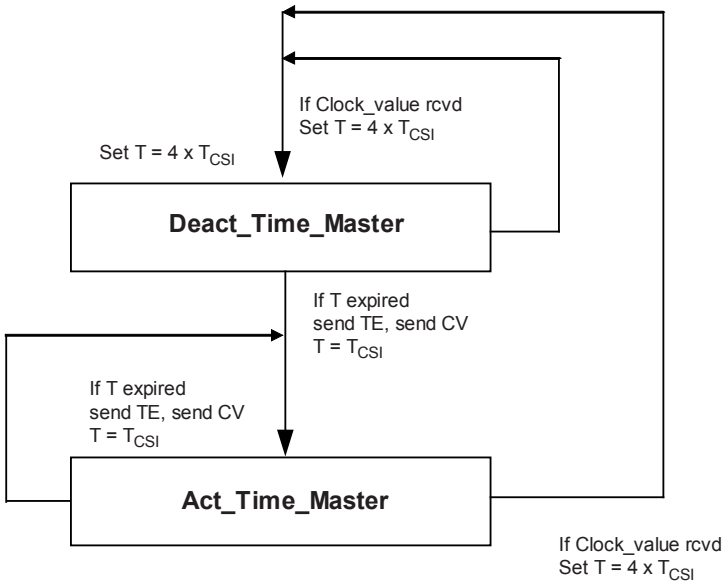
Figure 39 – Overview of clock synchronization

The clock synchronization protocol provides optionally redundant time masters, and a simple election procedure for determining the active time master. One time master on each link is presumed to be the primary time master. Other time masters are standby time masters that can act as sources of clock synchronization when the primary time master is not active. Standby time masters are not ranked in order of importance. The clock synchronization protocol will always select the primary time master as the clock synchronization source if it is present. The figure below is the state machine definition for the operation of a primary time master (TM) or a standby time master.

Standby time masters remain inactive as long as they receive clock synchronization messages from the active time master within a monitoring window. The monitoring window is restarted at each received clock synchronization message. The length of the monitoring window is set to $4 \times T_{CSI}$, a multiple of the Clock Synchronization Interval Time.

If no clock synchronization message is received during the monitoring window it is presumed that the active time master has failed. Standby time master then begin sending clock synchronization messages.

If a standby time master receives a clock synchronization message from another standby time master, then it will fall back to monitoring clock synchronization sequences. Messages received from the primary time master always result in the standby time master falling back to monitoring (see Figure 40).



where

TE, CV: Timer Event, Clock Value

T: Actual elapsed Timer Value

T_{CSI}: Clock Sync Interval

Figure 40 – Time master state machine

8.3.3 State machine time receiver

The receiver state machine is described in Figure 41. The receiver monitors clock messages within a time window of a multiple of the Clock Synchronization Interval Time $4 \times T_{CSI}$.

It adjusts, if necessary, its clock after reception of a correct clock synchronization sequence. No clock adjustment is made and errors are marked in following cases.

a) Different source addresses in Time Event and Clock Value

b) No clock synchronization message received within time window

c) Clock values not matched (different last_clk_val).

The handling of the Time Event, checking the source address of TE/CV DLPDU and the receive-delay-timer is part of the DLE.

With the reception of a valid Clock Value (SA is the same as the SA in the previously received Timer Event), a DL service gives the received synchronization time and the receiver part of the communication delay to the DLS-user. The receiver part of the communication delay includes the time past since the receive-delay-timer was started with the reception of a TE DLPDU till read from the DLE after the reception of a CV DLPDU.

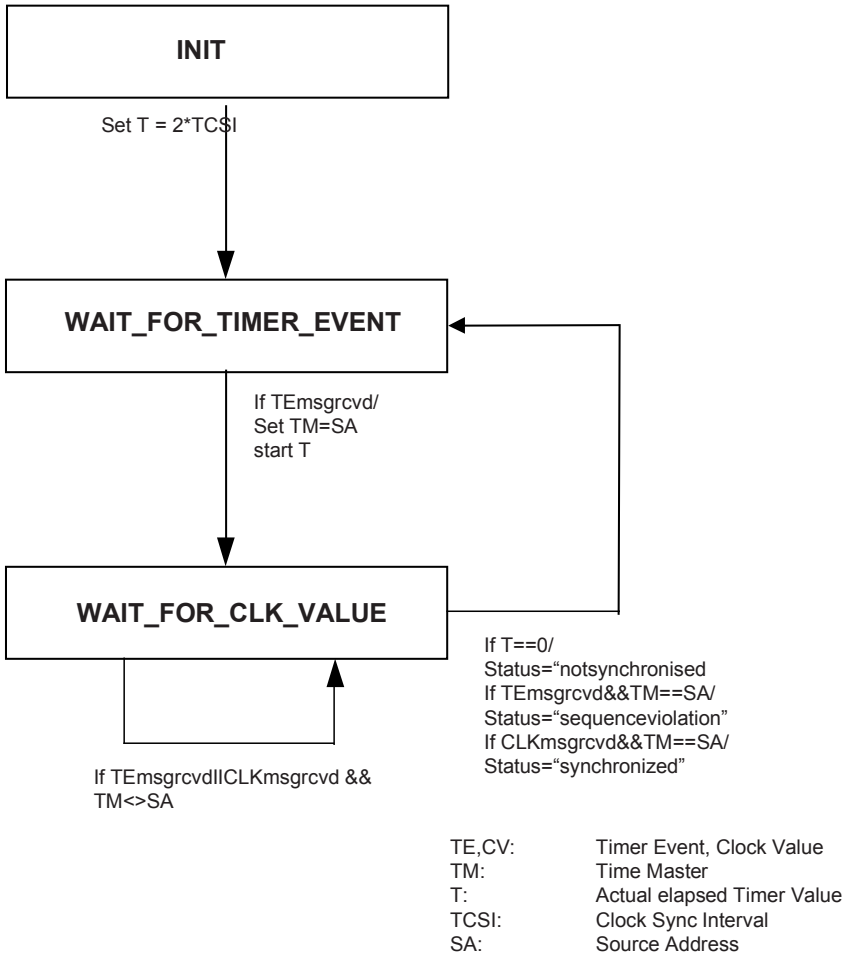


Figure 41 – Time receiver state machine

Figure 42 illustrates the Clock Synchronization sequences.

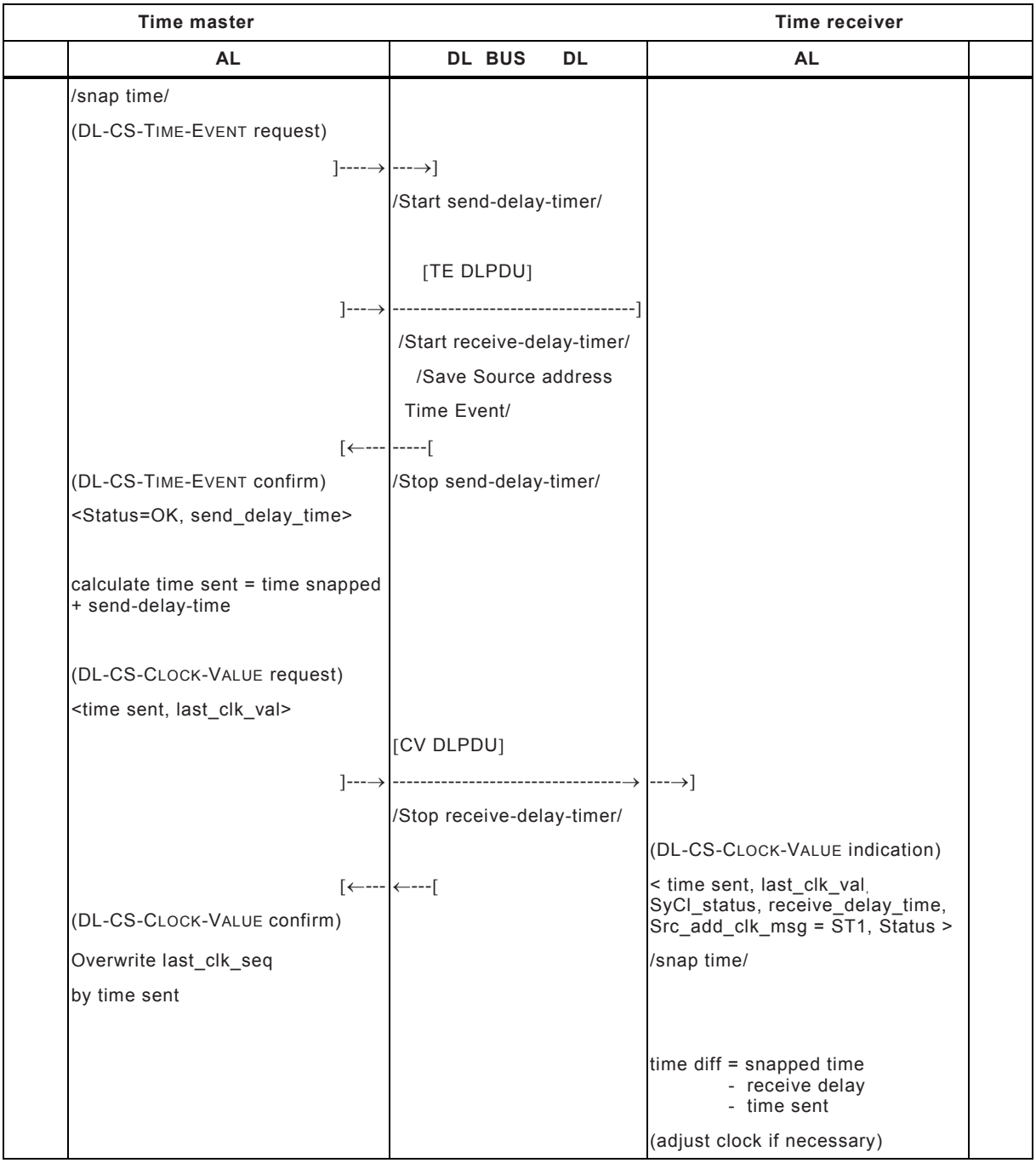


Figure 42 – Clock synchronization

Annex A
(normative)

DL-Protocol state machines

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

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

A.1 Overall structure

The DL protocol of Type 3 consist of the following three parts:

- handling of the services invoked by Application Layer (DL- and DLM-services);
- media access control (token handling and service interactions);
- interface to physical layer with PDU assembling/disassembling.

The Interface between the service handler (FLC/DLM) and Media Access Control (MAC) consists of a set of Queues, a SAP-list and a DL-Data-Resource. (This will be used also by the Send-Receive Unit –SRU.)

The Interface between MAC and SRU consist of a set of services (see A.6.5).

The protocol sequences are described by means of State Machines.

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

The textual listing of the state transitions is structured as follows:

The first row contains the name of the transition.

The second row in define the current state.

The third row contains an optional event followed by Conditions starting with a “/” as first line character and finally followed by the Actions starting with a “=>” as first line character.

The last row contains the next state.

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

Additional conventions for state machines are given in IEC 61158-6-3.

Figure A.1 illustrates the structuring of the Protocol Machines.

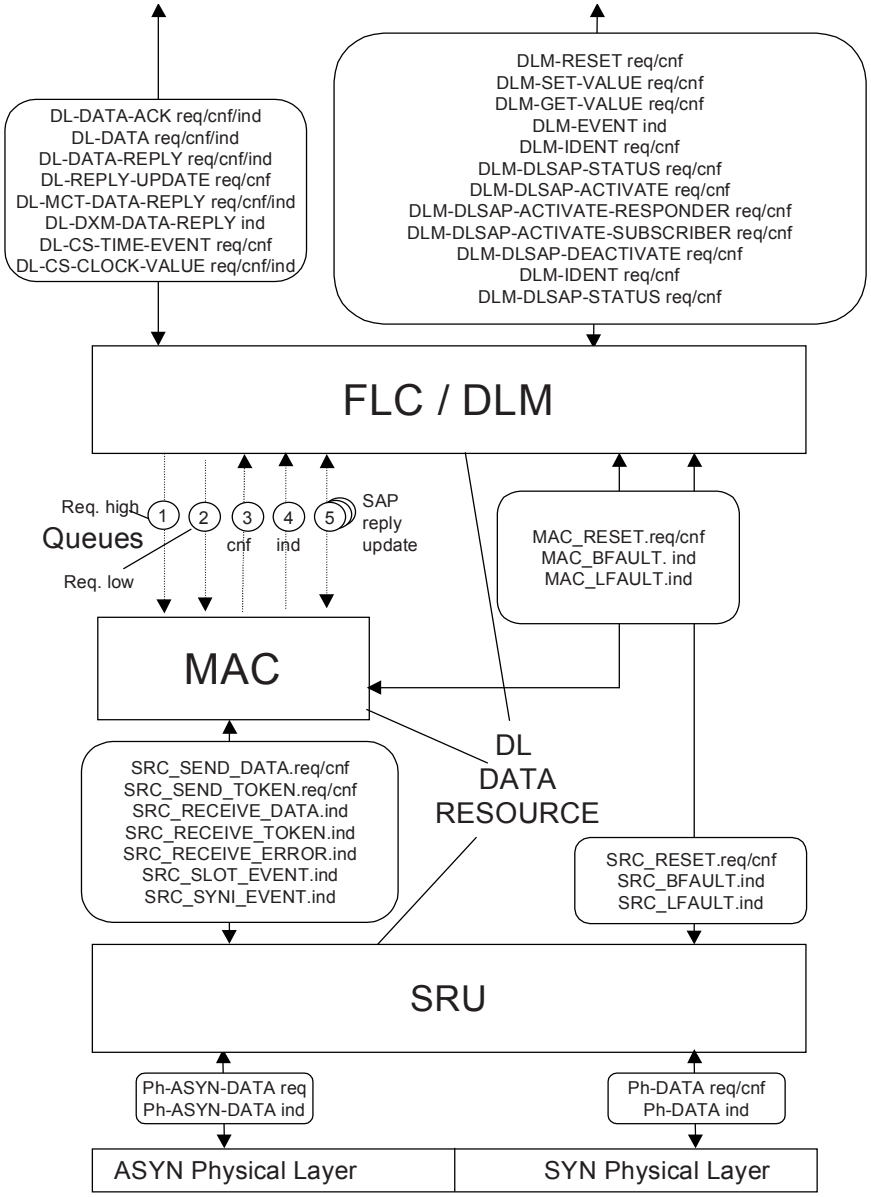


Figure A.1 – Structuring of the protocol machines

A.2 Variation of state machines in different devices

Table A.1 shows the assignment of State Machines parts to devices. The layout of the state machine tables is defined in IEC 61158-6-3, state machine conventions.

Table A.1 – Assignment of state machines

	Machine	Options	Remarks
Slave	FLC	SDN/SDA/SRD/CS/MSRD Indication SRD reply-update Req/Con	
		DLSAP-Activate/Deactivate	
	DLM	Limited set of FDL-variables No DLSAP-Activate-service	
	MAC	Only OFFLINE- and PASSIVE-IDLE-State	support for Data rate-detection
	SRU	No Token No Request (sending) No Response (receiving)	
Master	FLC	Full set of services	
	DLM	Full set of services	
	MAC	Full set of states	
	SRU	Full set of services/states	
Only Master	MAC	Passive idle omitted No active/passive switching	Other option: Passive but no switching

A.3 DL Data Resource

The DL Data Resource models a data interface for all state machines of the Data Link Layer. It contains queues and buffers for exchanging data between the sublayers of the DL. Moreover it contains management information which have to be accessed by various sublayers of the DL. Table A.2 illustrates all Data Resources of the Data Link Layer.

Table A.2 – Data resource

Name	Struct element	Type	Range	Remark
FDL-Variables		S0		
SAP_List		[0..63, CS, NIL] of S1		
H_List		S2		
L_List		S2		
C_List		S3		
I_List		S3		
Ident_List		S12		
S0	**Variables**			
	TS	U8	0 to 126	DL-address of this station
	Data_rate		9,6; 19,2; 31,25; 45,45; 93,75; 187,5; 500; 1 500, 3 000; 6 000; 12 000 kBit/s and others	Data rate of this fieldbus
	Medium_ redundancy		single; redundant	Availability of redundant media
	HW-Release		LE_HR; hardware release identification	Hardware release number
	SW-Release		LE_SR; software release identification	Software release number
	SYNCHT		DSAP SSAP LSDU	Contents of the SYNCH DLPDU
	Tct	U32	1 to 2^{24}-1 (bit times)	Isochronous cycle time
	Isochronous_ mode		0 1 2 3	non Isochronous with cycle correction with cycle drop TTR timer stop
	maxTsh	U8	1 to 256 (bit times)	maximal time shift
	Tcsi	U32	1 to 2^{32}-1 (bit times)	Clock Synchronization Interval Time
	Preamble_length	U8	8 to 64 (bit)	Length of the preamble of physical frames at synchronous transmission
	Tsyn	U8	4 to 32 (bit times)	post transmission gap time of synchronous transmission
	Tsl	U16	52 to 2^{16}-1 (bit times)	Slot time
	minTsdr	U16	2^0 to 2^{16}-1 (bit times)	Smallest station delay time
	maxTsdr	U16	2^0 to 2^{16}-1 (bit times)	Largest station delay time
	Tqui	U8	0 to 2^8-1 (bit times)	Transmitter fall time (line state uncertain time) or repeater switch time
	Tset	U8	2^0 to 2^8-1 (bit times)	Setup time
	Ttr	U32	2^0 to 2^{24}-1 (bit times)	Target rotation time
	G	U8	1 to 100	GAP update factor
	in_ring_desired	Bool	true; false	Request entry into or exit out of the logical token ring
	HSA	U8	0 to 126	Highest station address on this fieldbus

Name	Struct element	Type	Range	Remark
	max_retry_limit	U8	0 to 15 (preferably 0)	Maximum number of retries
	DLPDU_sent_ count	U32	0 to 2^{32}-1	Number of DLPDUs sent
	Retry_count	U16	0 to 2^{16}-1	Number of DLPDU repeats
	DLPDU_sent_ count_sr	[0..126] of U32	max. 126 entries of 0 to 2^{32}-1	List of numbers of DLPDUs sent per station
	Error_count	[0..126] of U16	max. 126 entries of 0 to 2^{16}-1	List of numbers of no or erroneous responses per station
	SD_count	U32	0 to 2^{32}-1	Number of correct start delimiters
	SD_error_count	U16	0 to 2^{16}-1	Number of defective start delimiters
	Trr	U32	2^0 to 2^{24}-1 (bit times)	Real rotation time
	LMS	[0..126] of U8	up to 127 DL-addresses	List of Master stations in the logical ring
	GAPL	[0..126] of DLE status	max. 126 DL-addresses (0 to 126) inclusive DLE status	List all of stations in the own GAP
	Cycle_violation_ count	U32		Number of cycle violations which occurred since the start of isochronous mode
	Tid1	U16	33 bit + 2 bit + 2 × Tset + Tqui	Deduced Variables
	Tid2	U16	max (Tid1, maxTsdr)	
S1	**SAP_List**			
	Access	U8	0..126, NIL	
	Function_List_R	List of		SDN_H/L,SDA_H/L,SRD_H/L, MSRD_H/L, CS(TE/CV)
	Function_List_I	List of		SDN_H/L, SDA_H/L,SRD_H/L, MSRD_H/L, CS(TE/CV)
	LenList	[0..16] of U8	0..246	List of permitted DLSDU lengths depending on FC.Function
	Indication_mode		ALL/DATA/ UNCHANGED	
	Publisher_ enabled		TRUE/FALSE	
	Ibuffer	S7		Indication Buffer
	Ubuffer	S8		Update Buffer
	Sbuffer	S7		Subscriber Buffer
S2	**H_List/L_List**			
	Num_entry	U16		
	First_entry	Ref to S9		
	Last_entry	Ref to S9		
	Insert()			
	Remove()			

Name	Struct element	Type	Range	Remark
S3	**C_List/I_List**			
	Num_entry	U16		
	First_entry	Ref to S10		
	Last_entry	Ref to S10		
	Insert()			
	Remove()			
S4	**REQM**			
	DA	U8	0..126	
	DSAP	U8	0..63, CS, NIL	
	SSAP	U8	0..63, CS, NIL	
	FC	S6		
	DLSDU	S11		
	Serv_class		high/low	
	Conf	Bool		
S5	**RESM**			
	DA	U8	0..126	
	DSAP	U8	0..63, CS, NIL	
	SSAP	U8	0..63, CS, NIL	
	FC	S6		
	DLSDU	S11		
S6	**FC**			
	Function		req rsp	SDN_H/L, SDA_H/L, SRD_H/L, MSRD_H/L, Ident, LSAP_Status, CS(TE/CV)
	Frame_type		req/rsp	
	FCB	U8	0,1	
	FCV	U8	0,1	
	Stn-type		Slave, M_n_rdy, M_rdy, M_in_ring	
S7	**Ibuffer**			
	Num_entry	U16		
	First_entry	Ref to S10		
	Last_entry	Ref to S10		
	Insert()			
	Remove()			
S8	**Ubuffer**			
	High_reference	U32		
	High_buffer	S11		
	High_transmit		SINGLE/MULTIPLE	
	Low_reference	U32		
	Low_buffer	S11		
	Low_transmit		SINGLE/MULTIPLE	

Name	Struct element	Type	Range	Remark
S9	**H/L_List entry**			
	Next entry	Ref to S9		
	DA	U8	0..127	
	DSAP	U8	0..63, CS, NIL	
	SSAP	U8	0..63, CS, NIL	
	FC	S6		
	DLSDU	S11		
	conf	Bool		
S10	**C/I_List entry**			
	Next entry	Ref to S10		
	DA	U8	0..127	
	SA	U8	0..127	
	DSAP	U8	0..63, CS, NIL	
	SSAP	U8	0..63, CS, NIL	
	FC	S6		
	DLSDU	S11		
	conf	Bool		
	R_Status	NO,DL,DH,OK,R, RR,UE,NR,RDH, RDL,NA,DS		
	Reference	U32		
S11	**DLSDU**			
	Len	U8	0..246	
	Data	[0..246] of U8		
S12	**Ident_List**			
	Vendor_Name	[0..32] of U8		
	Model_Name	[0..32] of U8		
	HW_Release	[0..32] of U8		
	SW_Release	[0..32] of U8		

A.4 FLC / DLM

A.4.1 Primitive definitions

A.4.1.1 Primitive exchanged between DL-User and FLC

Table A.3 shows all primitives issued by DL-User to the FLC. See IEC 61158-3-3 for a description of the functions.

Table A.3 – Primitives issued by DL-User to FLC

Primitive name	Associated parameters
DL-DATA-ACK request	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU

Primitive name	Associated parameters
DL-DATA request	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU
DL-DATA-REPLY request	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU
DL-MCT-DATA-REPLY request	Service_class, D_addr D_SAP_index, S_SAP_index, DLSDU,
DL-REPLY-UPDATE request	Service_class, S_SAP_index, DLSDU, Transmit_strategy, Reference
DL-CS-TIME-EVENT request	D_addr, D_SAP_index, S_SAP_index
DL-CS-CLOCK-VALUE request	D_addr, D_SAP_index, S_SAP_index, DLSDU

Table A.4 shows all primitives issued by the FLC to the DL-User. See IEC 61158-3-3 for a description of the functions.

Table A.4 – Primitives issued by FLC to DL-User

Primitive name	Associated parameters
DL-DATA-ACK confirm	Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status
DL-DATA confirm	Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status
DL-DATA-REPLY confirm	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status
DL-REPLY-UPDATE confirm	Service_class, S_SAP_index, DL_status
DL-MCT-DATA-REPLY confirm	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status

Primitive name	Associated parameters
DL-CS-TIME-EVENT confirm	D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status
DL-CS-CLOCK-VALUE confirm	D_addr, D_SAP_index, S_SAP_index, DL_status
DL-DATA-ACK indication	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU
DL-DATA indication	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU
DL-DATA-REPLY indication	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU, Update_status, Reference
DL-MCT-DATA-REPLY indication	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU, Update_status, Reference
DL-DXM-REPLY indication	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU
DL-CS-CLOCK-VALUE indication	D_addr, D_SAP_index, S-addr, S_SAP_index, DLSDU, Receive_delay_time CS_status

A.4.1.2 Primitive exchanged between DL-User and DLM

Table A.5 shows all primitives issued by the DL-User to the DLM. See IEC 61158-3-3 for a description of the functions.

Table A.5 – Primitives issued by DL-User to DLM

Primitive name	Associated parameters
DLM-RESET request	(none)
DLM-SET-VALUE request	Variable_name(1 to n), Index(1 to k), Desired_value (1 to n)
DLM-GET-VALUE request	Variable_name(1 to n), Index(1 to k)
DLM-DLSAP-ACTIVATE request	S_SAP_index, Access, Service_list
DLM-DLSAP-ACTIVATE-RESPONDER request	S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled
DLM-DLSAP-ACTIVATE-SUBSCRIBER request	S_SAP_index, DLSDU_length_list
DLM-DLSAP-DEACTIVATE request	S_SAP_index
DLM-IDENT request	DL_addr
DLM-DLSAP-STATUS request	D_SAP_index, DL_addr

Table A.6 shows all primitives issued by the DLM to the DL-User. See IEC 61158-3-3 for a description of the functions.

Table A.6 – Primitives issued by DLM to DL-User

Primitive name	Associated parameters
DLM-RESET confirm	DLM_Status
DLM-SET-VALUE confirm	Current_value(1 to n), DLM_status(1 to n)
DLM-GET-VALUE confirm	Desired_value(1 to n), DLM_status(1 to n)
DLM-EVENT indicate	Event/Fault TSH
DLM-DLSAP-ACTIVATE confirm	S_SAP_index, DLM_status
DLM-DLSAP-ACTIVATE-RESPONDER confirm	S_SAP_index, DLM_status
DLM-DLSAP-ACTIVATE-SUBSCRIBER confirm	S_SAP_index, DLM_status
DLM-DLSAP-DEACTIVATE confirm	S_SAP_index, DLM_status
DLM-IDENT confirm	DL-addr, Ident_list, DLM_status
DLM-DLSAP-STATUS confirm	D_SAP_Index, DL-addr, Access, Service_type(1 to n), Role_in_service_list(1 to n), DLM_status

A.4.1.3 Parameters of FLC Primitives

Table A.7 shows all parameters used with primitives between the DL-User and the FLC.

Table A.7 – Parameters used with primitives exchanged between DL-User and FLC

Parameter name	Description
S_SAP_index	Identifier of the local Service Access Point
D_SAP_index	Identifier of a remote Service Access Point
D_addr	Station address of the receiving DLE
S_addr	Station address of the sending DLE
Service_class	Indicates priority of the service
DL_status	Status of the service execution
DLSDU	Data Unit from or to the DLS-user
Transmit_strategy	Operation mode of the update buffer (single = buffer is transmitted once, multiple = buffer is transmitted repeated)
Reference	Reference to identify the DLSDU passed to the local DLE
Update_status	Indicates the presence of a DLSDU from a remote DLE in the update buffer
Send_delay_time	Protocol delay time between the invocaton of the Clock Synchronization service request primitive and the transmission of the DLPDU
Receive_delay_time	Protocol delay time between the reception of the previous and the current DLPDU of Clock Synchronization
CS_status	Status of the Clock Synchronization sequence

A.4.1.4 Parameters of DLM primitives

Table A.8 shows all parameters used with primitives between the DL-User and the DLM.

Table A.8 – Parameters used with primitives exchanged between DL-User and DLM

Parameter name	Description
S_SAP_index	Identifier of the local Service Access Point
D_SAP_index	Identifier of a remote Service Access Point
Variable_name(1 to n)	List of DL variable names
Desired_value (1 to n)	List of DL variable values
Current_value (1 to n)	List of DL variable values
Access	Permission to use of a local SAP by one or several remote DLEs
Service_list	Type of services accepted at a local SAP
DLSDU_length_list	Maximum length of incoming or outgoing DLSDU at a local SAP
Indication_mode	Mode for Indication of received DLPDUs without user data
DLM_status	Status of the service execution
Event/Fault	Indicates the cause of an event or a fault
DL-addr	Station address of the local or a remote DLE
Ident_list	Specifies Vendor Name, Controller Type; HW/SW-Release
Service_type(1 to n)	Specifies the DL-services activated
Role_in_service_list(1 to n)	Specifies the role in service (initiator or responder or both)

A.4.2 State machine description

The FLC forms the interface between DL Protocol Machines and DL-User for SDA, SDN, SRD, MSRD and CS services.

The DLM forms the interface between DLM Protocol Machines and DLM-User for management services.

The FLC services are processed in the same way as the DLM services. For this reason the DLM module is described in the same state machine as the FLC module and is contained in the following description.

The services are all handled in the READY-State.

The service request will be validated first. A negative validation will result in a negative confirmation. Otherwise the service will be put into the high or low priority service queue according to the parameter service class. Services executed by MAC will be moved from the request queues to the confirmation queues. All valid incoming services will be put into the indication queue.

The set of SAP (de-)activate services are used to organize the SAP list. The Reply Update services are used to put reply data in the service list.

Local Variables

The FLC / DLM does not use local variables. All variables which have to be accessed by the FLC / DLM are contained in the DL Data Resource.

State Table Nomenclature

The standard suffixes ".req", ".cnf" and ".ind" are used to indicate the request, confirm and indication primitives, respectively. Service primitive names are entirely upper-case.

A.4.2.1 FLC / DLM Table

The FLC and DLM State Table is shown in Table A.9.

Table A.9 – FLC/DLM state table

No.	Current state	Event /condition ⇒action	Next state
1	READY	DLM-RESET.req ⇒ MAC_reset := FALSE SRC_reset := FALSE MAC_RESET.req SRC_RESET.req	WAIT_RESET_CNF
2	WAIT_RESET_CNF	MAC_RESET.cnf /SRC_reset ⇒ RESET_LIST DLM-RESET.cnf	READY
3	WAIT_RESET_CNF	MAC_RESET.cnf /!SRC_reset ⇒ MAC_reset := TRUE	WAIT_RESET_CNF

No.	Current state	Event /condition ⇒action	Next state
4	WAIT_RESET_CNF	SRC_RESET.cnf /MAC_reset ⇒ RESET_LIST DLM-RESET.cnf	READY
5	WAIT_RESET_CNF	SRC_RESET.cnf /!MAC_reset ⇒ SRC_reset := TRUE	WAIT_RESET_CNF
6	READY	DLM-GET-VALUE.req (Variable_name((1 to n), Index(1 to k))) /!CHECK_PAR_READVALUE (Variable_name((1 to n), Index(1 to k))) ⇒ DLM_status (1 to n) := IV Current_value (1 to n) := NIL DLM-GET-VALUE.cnf (Current_value(1 to n),DLM_status(1 to n))	READY
7	READY	DLM-GET-VALUE.req (Variable_name((1 to n), Index(1 to k))) /CHECK_PAR_READVALUE (Variable_name((1 to n), Index(1 to k))) ⇒ Check_variable_names (Variable_name((1 to n), Index(1 to k))) Set_current_values (Variable_name((1 to n), Index(1 to k))) DLM-GET-VALUE.cnf (Current_value(1 to n),DLM_status(1 to n))	READY
8	READY	DLM-SET-VALUE.req (Variable_name((1 to n), Index(1 to k)), Desired_value(1 to n)) /!CHECK_PAR_SETVALUE (Variable_name((1 to n), Index(1 to k)), Desired_value(1 to n)) ⇒ DLM_status (1 to n) := IV DLM-SET-VALUE.cnf (DLM_status (1 to n))	READY
9	READY	DLM-SET-VALUE.req (Variable_name((1 to n), Index(1 to k)), Desired_value(1 to n)) /CHECK_PAR_SETVALUE (Variable_name((1 to n), Index(1 to k)), Desired_value(1 to n)) ⇒ Set_variable_list (Variable_name((1 to n), Index(1 to k)), Desired_value(1 to n)) DLM-SET-VALUE.cnf (DLM_status (1 to n))	READY
10	READY	MAC_BFAULT.ind (Fault_type, Tsh) ⇒ Event/Fault := Fault_type DLM-EVENT.ind (Event/Fault, Tsh)	READY
11	READY	MAC_BFAULT.ind (Fault_type) ⇒ Event/Fault := Fault_type DLM-EVENT.ind (Event/Fault)	READY
12	READY	MAC_LFAULT.ind (Fault_type) ⇒ Event/Fault := Fault_type DLM-EVENT.ind (Event/Fault)	READY
13	READY	SRC_BFAULT.ind (Event_type) ⇒ Event/Fault := Event_type DLM-EVENT.ind (Event/Fault)	READY
14	READY	SRC_LFAULT.ind (Event_type) ⇒ Event/Fault := Event_type DLM-EVENT.ind (Event/Fault)	READY
15	READY	DLM-DLSAP-ACTIVATE.req (S_SAP_index, Access, Service_list) /!CHECK_PAR_DLSAP ⇒ DLM_status := IV DLM-DLSAP-ACTIVATE.cnf (S_SAP_index, DLM_status)	READY

No.	Current state	Event /condition ⇒action	Next state
16	READY	DLM-DLSAP-ACTIVATE.req (S_SAP_index, Access, Service_list) /CHECK_PAR_DLSAP && ACTIVATED (S_SAP_index) ⇒ DLM_status := NO DLM-DLSAP-ACTIVATE.cnf (S_SAP_index, DLM_status)	READY
17	READY	DLM-DLSAP-ACTIVATE.req (S_SAP_index, Access, Service_list) /CHECK_PAR_DLSAP && !ACTIVATED (S_SAP_index) ⇒ SET_SAP_LIST (S_SAP_index, Access, Service_list) DLM_status := OK DLM-DLSAP-ACTIVATE.cnf (S_SAP_index, DLM_status)	READY
18	READY	DLM-DLSAP-ACTIVATE-RESPONDER.req (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) /!CHECK_PAR_DLSAP_RES ⇒ DLM_status := IV DLM-DLSAP-ACTIVATE-RESPONDER.cnf (S_SAP_index, DLM_status)	READY
19	READY	DLM-DLSAP-ACTIVATE-RESPONDER.req (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) /CHECK_PAR_DLSAP_RES && RACTIVATED (S_SAP_index) && Indication_mode≠UNCHANGED ⇒ DLM_status := NO DLM-DLSAP-ACTIVATE-RESPONDER.cnf (S_SAP_index, DLM_status)	READY
20	READY	DLM-DLSAP-ACTIVATE-RESPONDER.req (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) /CHECK_PAR_DLSAP_RES && !RACTIVATED (S_SAP_index) && Indication_mode=UNCHANGED ⇒ DLM_status := NO DLM-DLSAP-ACTIVATE-RESPONDER.cnf (S_SAP_index, DLM_status)	READY
21	READY	DLM-DLSAP-ACTIVATE-RESPONDER.req (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) /CHECK_PAR_DLSAP_RES && !RACTIVATED (S_SAP_index) && Indication_mode≠UNCHANGED ⇒ SET_RSAP_LIST (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) DLM_status := OK DLM-DLSAP-ACTIVATE-RESPONDER.cnf (S_SAP_index, DLM_status)	READY
22	READY	DLM-DLSAP-ACTIVATE-RESPONDER.req (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) /CHECK_PAR_DLSAP_RES && RACTIVATED (S_SAP_index) && Indication_mode=UNCHANGED && CHECK_SAP_LIST (S_SAP_index, DLSDU_length_list) ⇒ SET_RSAP_LIST (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) DLM_status := OK DLM-DLSAP-ACTIVATE-RESPONDER.cnf (S_SAP_index, DLM_status)	READY
23	READY	DLM-DLSAP-ACTIVATE-RESPONDER.req (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled) /CHECK_PAR_DLSAP_RES && RACTIVATED (S_SAP_index) && Indication_mode=UNCHANGED && !CHECK_SAP_LIST (S_SAP_index, DLSDU_length_list) ⇒ DLM_status := NO DLM-DLSAP-ACTIVATE-RESPONDER.cnf (S_SAP_index, DLM_status)	READY
24	READY	DLM-DLSAP-ACTIVATE-SUBSCRIBER.req (S_SAP_index, DLSDU_length_list) /!CHECK_PAR_DLSAP_SUB ⇒ DLM_status := IV DLM-DLSAP-SUBSCRIBER.cnf (S_SAP_index, DLM_status)	READY

No.	Current state	Event /condition ⇒action	Next state
25	READY	DLM-DLSAP-ACTIVATE-SUBSCRIBER.req (S_SAP_index, DLSDU_length_list) /CHECK_PAR_DLSAP_SUB && SACTIVATED (S_SAP_index) ⇒ DLM_status := NO DLM-DLSAP-SUBSCRIBER.cnf (S_SAP_index, DLM_status)	READY
26	READY	DLM-DLSAP-ACTIVATE-SUBSCRIBER.req (S_SAP_index, DLSDU_length_list) /CHECK_PAR_DLSAP_SUB && !SACTIVATED (S_SAP_index) ⇒ SET_SSAP_LIST (S_SAP_index, DLSDU_length_list) DLM_status := OK DLM-DLSAP-SUBSCRIBER.cnf (S_SAP_index, DLM_status)	READY
27	READY	DLM-DLSAP-DEACTIVATE.req (S_SAP_index) /!CHECK_PAR_DLSAP_DEACT ⇒ DLM_status := IV DLM-DLSAP-DEACTIVATE.cnf (S_SAP_index, DLM_status)	READY
28	READY	DLM-DLSAP-DEACTIVATE.req (S_SAP_index) /CHECK_PAR_DLSAP_DEACT && !ACTIVATED (S_SAP_index) ⇒ DLM_status := NO DLM-DLSAP-DEACTIVATE.cnf (S_SAP_index, DLM_status)	READY
29	READY	DLM-DLSAP-DEACTIVATE.req (S_SAP_index) /CHECK_PAR_DLSAP_DEACT && ACTIVATED (S_SAP_index) ⇒ RESET_SAP_LIST (S_SAP_index) DLM_status := OK DLM-DLSAP-DEACTIVATE.cnf (S_SAP_index, DLM_status)	READY
30	READY	DLM-DLSAP-STATUS.req (D_SAP_index, D_addr) /!CHECK_PAR_STATUS ⇒ Access := NIL Service_list (1 to n) := NIL Role_in_service_list (1 to n) := NIL DLM_status := IV DLM-DLSAP-STATUS.cnf (D_SAP_index, D_addr, Access, Service_type (1 to n), Role_in_service_list (1 to n), DLM_status)	READY
31	READY	DLM-DLSAP-STATUS.req (D_SAP_index, D_addr) /CHECK_PAR_STATUS && D_addr = Variables.TS && SAP_List[D_SAP_index] = NIL ⇒ Access := NIL Service_list (1 to n) := NIL Role_in_service_list (1 to n) := NIL DLM_status := LR DLM-DLSAP-STATUS.cnf (D_SAP_index, D_addr, Access, Service_type (1 to n), Role_in_service_list (1 to n), DLM_status)	READY
32	READY	DLM-DLSAP-STATUS.req (D_SAP_index, D_addr) /CHECK_PAR_STATUS && D_addr = Variables.TS && SAP_List ≠ NIL[D_SAP_index] ⇒ Write_SAPstatus_list() DLM_status := OK DLM-DLSAP-STATUS.cnf (D_SAP_index, D_addr, Access, Service_type (1 to n), Role_in_service_list (1 to n), DLM_status)	READY
33	READY	DLM-IDENT.req (D_addr) /!CHECK_PAR_IDENT ⇒ Ident_list := NIL DLM_status := IV DLM-IDENT.cnf (D_addr, Ident_list, DLM_status)	READY

No.	Current state	Event /condition ⇒action	Next state
34	READY	DLM-IDENT.req (D_addr) /CHECK_PAR_IDENT && D_addr = Variables.TS && Ident_List = NIL ⇒ Ident_list := NIL DLM_status := LR DLM-IDENT.cnf (D_addr, Ident_list, DLM_status)	READY
35	READY	DLM-IDENT.req (D_addr) /CHECK_PAR_IDENT && D_addr = Variables.TS && Ident_List ≠ NIL ⇒ Write_ident_list() DLM_status := OK DLM-IDENT.cnf (D_addr, Ident_list, DLM_status)	READY
36	READY	DLM-IDENT.req (D_addr) /CHECK_PAR_IDENT && Ident_Pending = TRUE && D_addr ≠ Variables.TS ⇒ Ident_list := NIL DLM_status := LR DLM-IDENT.cnf (D_addr, Ident_list, DLM_status)	READY
37	READY	DLM-IDENT.req (D_addr) /CHECK_PAR_IDENT && Ident_Pending = FALSE && D_addr ≠ Variables.TS ⇒ Ident_Pending := TRUE PUT_LREQ (IDENT)	READY
38	READY	DL-DATA.req (Service_class, D_Addr, D_SAP_index, S_SAP_index, DLSDU) /!CHECK_PAR_SDN ⇒ DL_status := IV DL-DATA.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY
39	READY	DL-DATA.req (Service_class, D_Addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDN && !CHECK_SAP(Service_class, S_SAP_index, SDN) ⇒ DL_status := LS DL-DATA.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY
40	READY	DL-DATA.req (Service_class, D_Addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDN && CHECK_SAP(Service_class, S_SAP_index, SDN) && !RESOURCE(S_SAP_index, DLSDU.Len) ⇒ DL_status := LR DL-DATA.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY
41	READY	DL-DATA.req (Service_class, D_Addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDN && CHECK_SAP(Service_class, S_SAP_index, SDN) && RESOURCE(S_SAP_index, DLSDU.Len) && Service_class=high ⇒ PUT_HREQ (SDN_H)	READY
42	READY	DL-DATA.req (Service_class, D_Addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDN && CHECK_SAP(Service_class, S_SAP_index, SDN) && RESOURCE(S_SAP_index, DLSDU.Len) && Service_class=low ⇒ PUT_LREQ (SDN_L)	READY
43	READY	DL-DATA-ACK.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /!CHECK_PAR_SDA ⇒ DL_status := IV DL-DATA-ACK.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY

No.	Current state	Event /condition ⇒action	Next state
44	READY	DL-DATA-ACK.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDA && !CHECK_SAP(Service_class, S_SAP_index, SDA) ⇒ DL_status := LS DL-DATA-ACK.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY
45	READY	DL-DATA-ACK.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDA && CHECK_SAP(Service_class, S_SAP_index, SDA) && !RESOURCE(S_SAP_index, DLSDU.Len) ⇒ DL_status := LR DL-DATA-ACK.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY
46	READY	DL-DATA-ACK.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDA && CHECK_SAP(Service_class, S_SAP_index, SDA) && RESOURCE(S_SAP_index, DLSDU.Len) && Service_class = high ⇒ PUT_HREQ (SDA_H)	READY
47	READY	DL-DATA-ACK.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_SDA && CHECK_SAP(Service_class, S_SAP_index, SDA) && RESOURCE(S_SAP_index, DLSDU.Len) && Service_class = low ⇒ PUT_LREQ (SDA_L)	READY
48	READY	DL-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /!CHECK_PAR_REPLY ⇒ DL_status := IV DL-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY
49	READY	DL-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && !CHECK_SAP (Service_class, S_SAP_index, SRD) ⇒ DL_status := LS DL-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY
50	READY	DL-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && CHECK_SAP (Service_class, S_SAP_index, SRD) && !RESOURCE (S_SAP_index, DLSDU.Len) ⇒ DL_status := LR DL-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY
51	READY	DL-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && CHECK_SAP (Service_class, S_SAP_index, SRD) && RESOURCE (S_SAP_index, DLSDU.Len) && Service_class=high ⇒ PUT_HREQ (SRD_H)	READY
52	READY	DL-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && CHECK_SAP (Service_class, S_SAP_index, SRD) && RESOURCE (S_SAP_index, DLSDU.Len) && Service_class=low ⇒ PUT_LREQ (SRD_L)	READY
53	READY	DL-MCT-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /!CHECK_PAR_REPLY ⇒ DL_status := IV DL-MCT-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY
54	READY	DL-MCT-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && !CHECK_SAP (Service_class, S_SAP_index, SRD) ⇒ DL_status := LS DL-MCT-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY

No.	Current state	Event /condition ⇒action	Next state
55	READY	DL-MCT-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && CHECK_SAP (Service_class, S_SAP_index, SRD) && !RESOURCE (S_SAP_index, DLSDU.Len) ⇒ DL_status := LR DL-MCT-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY
56	READY	DL-MCT-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && CHECK_SAP (Service_class, S_SAP_index, SRD) && RESOURCE (S_SAP_index, DLSDU.Len) && Service_class=high ⇒ PUT_HREQ (MSRD_H)	READY
57	READY	DL-MCT-DATA-REPLY.req (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU) /CHECK_PAR_REPLY && CHECK_SAP (Service_class, S_SAP_index, SRD) && RESOURCE (S_SAP_index, DLSDU.Len) && Service_class=low ⇒ PUT_HREQ (MSRD_L)	READY
58	READY	DL-REPLY-UPDATE.req (Service_class, S_SAP_index, DLSDU, Transmit_strategy, Reference) /!CHECK_PAR_UPDATE ⇒ DL_status := IV DL-REPLY-UPDATE.cnf (S_SAP_index, DL_status)	READY
59	READY	DL-REPLY-UPDATE.req (Service_class, S_SAP_index, DLSDU, Transmit_strategy, Reference) /CHECK_PAR_UPDATE && !CHECK_SAP (Service_class, S_SAP_index, SRD) ⇒ DL_status := LS DL-REPLY-UPDATE.cnf (S_SAP_index, DL_status)	READY
60	READY	DL-REPLY-UPDATE.req (Service_class, S_SAP_index, DLSDU, Transmit_strategy, Reference) / CHECK_PAR_UPDATE && CHECK_SAP (Service_class, S_SAP_index, SRD) && !RESOURCE (S_SAP_index, DLSDU.Len) ⇒ DL_status := LR DL-REPLY-UPDATE.cnf (S_SAP_index, DL_status)	READY
61	READY	DL-REPLY-UPDATE.req (Service_class, S_SAP_index, DLSDU, Transmit_strategy, Reference) /CHECK_PAR_UPDATE && CHECK_SAP (Service_class, S_SAP_index, SRD) && RESOURCE (S_SAP_index, DLSDU.Len) && Service_class=high ⇒ DL_status = OK PUT_HUBUFFER (S_SAP_index) DL-REPLY-UPDATE.cnf (S_SAP_index, DL_status)	READY
62	READY	DL-REPLY-UPDATE.req (Service_class, S_SAP_index, DLSDU, Transmit_strategy, Reference) /CHECK_PAR_UPDATE && CHECK_SAP (Service_class, S_SAP_index, SRD) && RESOURCE (S_SAP_index, DLSDU.Len) && Service_class=low ⇒ DL_status = OK PUT_LUBUFFER (S_SAP_index) DL-REPLY-UPDATE.cnf (S_SAP_index, DL_status)	READY
63	READY	/C_List.Num_entry≠0 && C_List.First_entry.Service = IDENT ⇒ Ident_pending := FALSE GET_IDENT_CNF() DLM-IDENT.cnf (D_addr, Ident_list, DLM_status)	READY

No.	Current state	Event /condition ⇒action	Next state
64	READY	/C_List.Num_entry≠0 && C_List.First_entry.Service = DLSAP-STATUS ⇒ Status_pending := FALSE GET_DLSAPSTATUS_CNF() DLM-DLSAP-STATUS.cnf (D_SAP_index, D_addr, Access, Service_type (1 to n), Role_in_service_list (1 to n), DLM_status)	READY
65	READY	/C_List.Num_entry≠0 && C_List.First_entry.Function = SDA_H || C_List.First_entry.Function = SDA_L ⇒ GET_SDA/SDN_CNF() DL-DATA-ACK.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY
66	READY	/C_List.Num_entry≠0 && C_List.First_entry.Service = SDN ⇒ GET_SDA/SDN_CNF() DL-DATA.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status)	READY
67	READY	/C_List.Num_entry≠0 && C_List.First_entry.Service = SRD ⇒ GET_SRD_CNF() DL-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY
68	READY	/C_List.Num_entry≠0 && C_List.First_entry.Service = MSRD ⇒ GET_SRD_CNF() DL-MCT-DATA-REPLY.cnf (Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status)	READY
69	READY	/I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function = SDA_H || I_List.First_entry.FC.Function = SDA_L ⇒ GET_SDA/SDN_IND() DL-DATA-ACK.ind (Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU)	READY
70	READY	/I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function= SDN_H || I_List.First_entry.FC.Function = SDN_L ⇒ GET_SDA/SDN_IND() DL-DATA.ind (Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU)	READY
71	READY	/I_List.Num_entry≠0 && I_List.First_entry.DA ≠ 127 && I_List_entry.First_entry.FC.Function= SRD_H || I_List.First_entry.FC.Function = SRD_L ⇒ GET_SRD_IND() DL-DATA-REPLY.ind (Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU, Update_status, Reference)	READY
72	READY	/I_List.Num_entry≠0 && I_List.First_entry.DA = 127&& I_List_entry.First_entry.FC.Function= SRD_H || I_List.First_entry.FC.Function = SRD_L ⇒ GET_DXM_IND() DL-DXM-REPLY.ind (Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU)	READY
73	READY	/I_List.Num_entry≠0 && I_List.First_entry.DA ≠ 127 && I_List_entry.First_entry.FC.Function= MSRD_H || I_List.First_entry.FC.Function = MSRD_L ⇒ GET_SRD_IND() DL-MCT-DATA-REPLY.ind (Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU, Update_status, Reference)	READY

No.	Current state	Event /condition ⇒action	Next state
101	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==STE &!CHECK_PAR_TE ⇒ DL_status:=IV DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
102	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==STE &CHECK_PAR_TE && !CHECK_SAP(High, CS, TE/CV) ⇒ DL_status:=LS DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
103	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==STE &CHECK_PAR_TE && CHECK_SAP(High, CS, TE/CV) && !RESOURCE(CS) ⇒ DL_status:=LR DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
104	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==STE &CHECK_PAR_TE && CHECK_SAP(High, CS, TE/CV) && RESOURCE(CS) ⇒ TSDT.start(2*Tcsi); PUT_HREQ (TE) Tm_State:=W_STE	READY
105	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==STE ⇒ DL_status:=SV DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
106	READY	/Tm_State==STE &Tr_State ≠ W_TE && TM < TS ⇒ TSDT.start(4*Tcsi); Tm_State:=CONFLICT	READY
107	READY	/Tm_State==W_STE &C_List.Num_entry≠0 && C_List.First_entry.Service=TE && C_List.First_entry.R_status=OK ⇒ Send_delay_time := 2*Tcsi-TSDT.cv; TSDT.stop; GET_TE_CNF(); Status:=OK DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status) Tm_State:=SCV	READY
108	READY	/Tm_State==W_STE &C_List.Num_entry≠0 && C_List.First_entry.Service=TE && C_List.First_entry.R_status≠OK ⇒ Send_delay_time := 2*Tcsi-TSDT.cv; TSDT.stop; GET_TE_CNF(); Status:=DS DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status) Tm_State:=STE	READY

No.	Current state	Event /condition ⇒action	Next state
109	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==W_STE ⇒ Send_delay_time := 0 DL_status:=SV DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status) Tm_State:=W_STE	READY
110	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==W_STE ⇒ DL_status:=SV DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=W_STE	READY
111	READY	/Tm_State==W_STE &Tr_State ≠ W_TE && TM < TS ⇒ TSDT.start(4*Tcsi); Tm_State:=W_CC	READY
112	READY	/Tm_State==W_STE &TSDT expired ⇒ TSDT.start(4*Tcsi); Tm_State:=W_DS	READY
113	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==SCV &!CHECK_PAR_CV ⇒ DL_status:=IV DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=SCV	READY
114	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==SCV &CHECK_PAR_CV && !CHECK_SAP(High, CS, TE/CV) ⇒ DL_status:=LS DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=SCV	READY
115	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==SCV &CHECK_PAR_CV && CHECK_SAP(High, CS, TE/CV) && !RESOURCE(CS) ⇒ DL_status:=LR DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=SCV	READY
116	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==SCV &CHECK_PAR_CV && CHECK_SAP(High, CS, TE/CV) && RESOURCE(CS) ⇒ DLSDU:=CS_list; TSDT.start(2*Tcsi); PUT_HREQ (CV) Tm_State:=W_SCV	READY
117	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==SCV ⇒ Send_delay_time := 0; DL_status:=SV DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status) Tm_State:=SCV	READY

No.	Current state	Event /condition ⇒action	Next state
118	READY	/Tm_State==SCV &Tr_State ≠ W_TE && TM < TS ⇒ TSDT.start(4*Tcsi); Tm_State:=CONFLICT	READY
119	READY	/Tm_State==W_SCV &C_List.Num_entry≠0 && C_List.First_entry.Service=CV && C_List.First_entry.R_status=OK ⇒ GET_CV_CNF(); Status:=OK DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
120	READY	/Tm_State==W_SCV &C_List.Num_entry≠0 && C_List.First_entry.Service=CV && C_List.First_entry.R_status≠OK ⇒ GET_CV_CNF(); Status:=DS DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
121	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==W_SCV ⇒ DL_status:=SV DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=W_SCV	READY
122	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==W_SCV ⇒ Send_delay_time := 0; DL_status:=SV DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status) Tm_State:=W_SCV	READY
123	READY	/Tm_State==W_SCV &Tr_State ≠ W_TE && TM < TS ⇒ TSDT.start(4*Tcsi); Tm_State:=W_CC	READY
124	READY	/Tm_State==W_STE &TSDT expired ⇒ TSDT.start(4*Tcsi); Tm_State:=W_DS	READY
125	READY	DL-CS-TIME-EVENT.req(D_addr, D_SAP_index, S_SAP_index) /Tm_State==CONFLICT ⇒ Send_delay_time := 0; DL_status:=SV DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status) Tm_State:=CONFLICT	READY
126	READY	DL-CS-CLOCK-VALUE.req (D_addr, D_SAP_index, S_SAP_index, CS_list) /Tm_State==CONFLICT ⇒ DL_status:=SV DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=CONFLICT	READY
127	READY	/Tm_State==CONFLICT &Tr_State ≠ W_TE && TM < TS ⇒ TSDT.start(4*Tcsi) Tm_State:=CONFLICT	READY

No.	Current state	Event /condition ⇒action	Next state
128	READY	/Tm_State==CONFLICT &TSDT expired ⇒ Tm_State:=STE	READY
129	READY	/Tm_State==W_CC &C_List.Num_entry≠0 && C_List.First_entry.Service=TE ⇒ Send_delay_time := 0; GET_TE_CNF(); Status:=SV DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, Send_delay_time, DL_status) Tm_State:=CONFLICT	READY
130	READY	/Tm_State==W_CC &C_List.Num_entry≠0 && C_List.First_entry.Service=CV ⇒ GET_CV_CNF(); Status:=SV DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=CONFLICT	READY
131	READY	/Tm_State==W_DS &C_List.Num_entry≠0 && C_List.First_entry.Service=TE ⇒ GET_TE_CNF(); Status:=DS DL-CS-TIME-EVENT.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
132	READY	/Tm_State==W_DS &C_List.Num_entry≠0 && C_List.First_entry.Service=CV ⇒ GET_CV_CNF(); Status:=DS DL-CS-CLOCK-VALUE.cnf (D_addr, D_SAP_index, S_SAP_index, DL_status) Tm_State:=STE	READY
201	READY	/Tr_State==W_TE &I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function = TE ⇒ TRDT.start(2*Tcsi); GET_TE/CV_IND(); TM:=S_addr Tr_State:=W_CV	READY
202	READY	/Tr_State==W_TE &I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function = CV ⇒ GET_TE/CV_IND() Tr_State:=W_TE	READY
203	READY	/Tr_State==W_CV &I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function = TE && I_List_entry.First_entry.SA ≠ TM ⇒ GET_TE/CV_IND() Tr_State:=W_CV	READY
204	READY	/Tr_State==W_CV &I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function = TE && I_List_entry.First_entry.SA = TM ⇒ TRDT.stop; Receive_delay_time := 2*Tcsi – TRDT.cv; CS_list := DLSDU; CS_status:=SV; GET_TE/CV_IND() DL-CS-CLOCK-VALUE.ind (D_addr, D_SAP_index, S_addr, S_SAP_index, CS_list, CS_status, Receive_delay_time) Tr_State:=W_TE	READY

No.	Current state	Event /condition ⇒action	Next state
205	READY	/Tr_State==W_CV &I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function = CV && I_List_entry.First_entry.SA ≠ TM ⇒ GET_TE/CV_IND() Tr_State:=W_CV	READY
206	READY	/Tr_State==W_CV &I_List.Num_entry≠0 && I_List_entry.First_entry.FC.Function = CV && I_List_entry.First_entry.SA = TM ⇒ TRDT.stop; Receive_delay_time := 2*Tcsi – TRDT.cv; CS_list := DLSDU; CS_status:=SV; GET_TE/CV_IND() DL-CS-CLOCK-VALUE.ind (D_addr, D_SAP_index, S_addr, S_SAP_index, CS_list, CS_status, Receive_delay_time) Tr_State:=W_TE	READY
207	READY	/Tr_State==W_CV &TCSI expired ⇒ Receive_delay_time := 0; CS_list := NULL; CS_status:=SV DL-CS-CLOCK-VALUE.ind (D_addr, D_SAP_index, S_addr, S_SAP_index, CS_list, CS_status, Receive_delay_time) Tr_State:=W_TE	READY

A.4.2.2 Functions

The FLC and DLM Functions are summarized in Table A.10.

Table A.10 – FLC / DLM function table

Function name	Operations
ACTIVATED (S_SAP_index)	Check that SAP is activated by setting its entry to valid value. SAP_List(S_SAP_index) ≠ NIL
RACTIVATED (S_SAP_index)	Check that SAP is activated as Responder.
SACTIVATED (S_SAP_index)	Check that SAP is activated as Subscriber.
CHECK_PAR_DLSAP	Check that all parameters of Service DLM-DLSAP-ACTIVATE.req are valid. Number of paramters must be 3. S_SAP_index: 0..63, NIL Access: 1..127 Service_list-Service_list_length: 1..(1 + 4 × 3) Service_list.n-th service_activate: “SDA” or “SDN” or “SRD” Service_list.n-th role_in_service: “INITIATOR” or “RESPONDER” (SDA, SDN only) or “BOTH” (SDA, SDN only) Service_list. nth DLSDU_length_list (1 to 4): 0..246
CHECK_PAR_DLSAP_DEACT	Check that all parameters of Service DLM-DLSAP-ACTIVATE-RESPONDER.req are valid. Number of paramters must be 1. S_SAP_index: 0..63, NIL

Function name	Operations
CHECK_PAR_DLSAP_SUB	Check that all parameters of Service DLM-DLSAP-ACTIVATE-RESPONDER.req are valid. Number of paramters must be 2. S_SAP_index: 0..62, NIL DLSDU_length_list (1 to 2): 0..246
CHECK_PAR_DLSAP_RES	Check that all parameters of Service DLM-DLSAP-ACTIVATE-RESPONDER.req are valid. Number of paramters must be 4. S_SAP_index: 0..62, NIL Access: 1..127 DLSDU_length_list (1 to 4): 0..246 Indication_mode: “ALL or “DATA” or “UNCHANGED”
CHECK_PAR_IDENT	Check that all parameters of Service DLM-IDENT.req are valid. Number of paramters must be 1. D_addr: 0..126
CHECK_PAR_READVALUE (Variable_name((1 to n), Index(1 to k)))	Check that the requested management variable names are valid. Possible values for MAC sublayer are: in_ring_desired Data_rate TS Ttr G HSA max_retry_limit SYNCHT Tct Isochronous_mode maxTsh Tcsi HW_Release SW_Release Trr LMS GAPL Possible values for SRC sublayer are: minTsdr maxTsdr Tsl Tqui Tset Medium_redundancy DLPDU_sent_count Retry_count DLPDU_sent_count_sr(1 to n) Error_count(1 to n) SD_count SD_error_count

Function name	Operations
CHECK_PAR_SETVALUE (Variable_name((1 to n), Index(1 to k)))	Check that the requested management variable names are valid. Possible values for MAC sublayer are: in_ring_desired Data_rate TS Ttr G HSA max_retry_limit SYNCHT Tct Isochronous_mode maxTsh Tcsi HW_Release SW_Release Possible values for SRC sublayer are: minTsdr maxTsdr Tsl Tqui Tset Medium_redundancy DLPDU_sent_count Retry_count DLPDU_sent_count_sr(1 to n) Error_count(1 to n) SD_count SD_error_count
CHECK_PAR_SDA	Check that all parameters of Service DL-DATA-ACK.req are valid. Number of paramters must be 5. Service_class: “high” or “low” D_addr: 0..126 D_SAP_index: 0..62, NIL S_SAP_index: 0..62, NIL DLSDU.Len: 1..246 DLSDU.Data: according to DLSDU.Len
CHECK_PAR_SDN	Check that all parameters of Service DL-DATA.req are valid. Number of paramters must be 5. Service_class: “high” or “low” D_addr: 0..127 D_SAP_index: 0..63, NIL S_SAP_index: 0..62, NIL DLSDU.Len: 1..246 DLSDU.Data: according to DLSDU.Len
CHECK_PAR_STATUS	Check that all parameters of Service DLM-DLSAP-STATUS.req are valid. Number of paramters must be 2. DLSAP: 0..63, NIL, CS D_addr: 0..126
CHECK_PAR_REPLY	Check that all parameters of Service DL-DATA-REPLY.req are valid. Number of paramters must be 5. Service_class: “high” or “low” D_addr: 0..126 D_SAP_index: 0..62, NIL S_SAP_index: 0..62, NIL DLSDU.Len: 0..246 DLSDU.Data: according to DLSDU.Len
CHECK_PAR_UPDATE	Check that all parameters of Service DL-REPLY-UPDATE.req are valid. Number of paramters must be 4. Service_class: “high” or “low” S_SAP_index: 0..62, NIL DLSDU.Len: 0..246 DLSDU.Data: according to DLSDU.Len Transmit_strategy: “SINGLE” or “MULTIPLE”
CHECK_PAR_TE	Check that all parameters of Service DL-CS-TIME-EVENT.req are valid. Number of paramters must be 3. D_addr = 127 D_SAP_index = CS S_SAP_index = CS

Function name	Operations
CHECK_PAR_CV	Check that all parameters of Service DL-CS-CLOCK-VALUE.req are valid. Number of paramters must be 4. D_addr = 127 D_SAP_index = CS S_SAP_index = CS Length of DLSDU: 18
CHECK_SAP (Service_class, S_SAP_index, Function)	Check that the SAP is activated to perform the current service request. SAP_List(S_SAP_index) ≠ NIL SAP_List(S_SAP_index).Function_List_I contains combination (Service_class, Function)
CHECK_SAP_LIST (S_SAP_index, DLSDU_length_list)	Check that the parameters of the service primitive conform to the settings of the SAP. DLSDU_length_list.Max_DLSDU_length_req_low = SAP_List(S_SAP_index).Ubuffer.Low_buffer.DLSDU.Len) DLSDU_length_list.Max_DLSDU_length_req_high = SAP_List(S_SAP_index).Ubuffer.High_buffer.DLSDU.Len) DLSDU_length_list.Max_DLSDU_length_ind_low = SAP_List(S_SAP_index).Ibuffer.Low_len) DLSDU_length_list.Max_DLSDU_length_ind_high = SAP_List(S_SAP_index).Ibuffer.High_len)
Check_variable_names (Variable_name((1 to n), Index(1 to k)))	Check that the management variables are set to valid values. LOOP for i from 1 to n for all variable names: if Variables.Variable_name (i) = NIL DLM_status (i) := NO else DLM_status (i) := OK
GET_DLSAPSTATUS_CNF ( )	Get a management service confirmation from the confirmation queue to MAC. C_List.Num_entry-- D_addr := C_List.First_Entry.DA D_SAP_index := C_List.First_Entry.DSAP retrieve Access from C_List.First_Entry.DLSDU.Data retrieve Service_type (1 to n) from C_List.First_Entry.DLSDU.Data retrieve Role_in_service_list (1 to n) from C_List.First_Entry.DLSDU.Data DLM_status := C_List.First_Entry.R_Status C_List.Remove()
GET_IDENT_CNF ( )	Get a management service confirmation from the confirmation queue to MAC. C_List.Num_entry-- D_addr := C_List.First_Entry.DA Ident_list := C_List.First_Entry.DLSDU.Data DLM_status := C_List.First_Entry.R_Status C_List.Remove()

GET_SDA/SDN_CNF
(
)
	Get a service confirmation from the confirmation queue to MAC.
	C_List.Num_entry--
	if (C_List.First_Entry.FC.Function = "SDA_H" or "SDN_H") Service_class := high
	if (C_List.First_Entry.FC.Function = "SDA_L" or "SDN_L") Service_class := low
	D_addr := C_List.First_Entry.DA
	D_SAP_index := C_List.First_Entry.DSAP
	S_SAP_index := C_List.First_Entry.SSAP
	DL_status := C_List.First_Entry.R_Status
	C_List.Remove()

Function name	Operations

GET_SDA/SDN_IND
(
)
	Get a service indication from the indication queue to MAC.
	I_List.Num_entry--
	if (I_List.First_Entry.FC.Function = "SDA_H" or "SDN_H") Service_class := high
	if (I_List.First_Entry.FC.Function = "SDA_L" or "SDN_L") Service_class := low
	D_addr := I_List.First_Entry.DA
	D_SAP_index := I_List.First_Entry.DSAP
	S_addr := I_List.First_Entry.SA
	S_SAP_index := I_List.First_Entry.SSAP
	DLSDU := I_List.First_Entry.DLSDU
	I_List.Remove()
	SAP_List(S_SAP_index).Ibuffer.Insert()
	SAP_List(S_SAP_index).Ibuffer.Num_entry++

GET_SRD_CNF
(
)
	Get a service confirmation from the confirmation queue to MAC.
	C_List.Num_entry--
	if (C_List.First_Entry.FC.Function = "SRD_H") Service_class := high
	if (C_List.First_Entry.FC.Function = "SRD_L") Service_class := low
	D_addr := C_List.First_Entry.DA
	D_SAP_index := C_List.First_Entry.DSAP
	S_SAP_index := C_List.First_Entry.SSAP
	DLSDU := C_List.First_Entry.DLSDU
	DL_status := C_List.First_Entry.R_Status
	C_List.Remove()

GET_SRD_IND
(
)
	Get a service indication from the indication queue to MAC.
	I_List.Num_Entry--
	if (I_List.First_Entry.FC.Function = "SRD_H") Service_class := high
	if (I_List.First_Entry.FC.Function = "SRD_L") Service_class := low
	D_addr := I_List.First_Entry.DA
	D_SAP_index := I_List.First_Entry.DSAP
	S_addr := I_List.First_Entry.SA
	S_SAP_index := I_List.First_Entry.SSAP
	DLSDU := I_List.First_Entry.DLSDU
	R_Status := I_List.First_Entry.Update_status
	Reference := I_List.First_Entry.Reference
	I_List.Remove()
	SAP_List(S_SAP_index).Ibuffer.Insert()
	SAP_List(S_SAP_index).Ibuffer.Num_entry++

GET_DXM_IND ( )	Get a service indication from the indication queue to MAC. I_List.Num_entry-- if (I_List.First_Entry.FC.Function = "DH") Service_class := high else Service_class := low D_addr := I_List.First_Entry.DA D_SAP_index := I_List.First_Entry.DSAP S_addr := I_List.First_Entry.SA S_SAP_index := I_List.First_Entry.SSAP DLSDU := I_List.First_Entry.DLSDU I_List.Remove() SAP_List(S_SAP_index).Sbuffer.Insert() SAP_List(S_SAP_index).Sbuffer.Num_entry++
GET_TE_CNF	Get a service confirmation from the confirmation queue to MAC. C_List.Num_entry-- D_addr := C_List.First_Entry.DA D_SAP_index := C_List.First_Entry.DSAP S_SAP_index := C_List.First_Entry.SSAP DL_status := C_List.First_Entry.R_Status C_List.Remove()
GET_CV_CNF	Get a service confirmation from the confirmation queue to MAC. C_List.Num_entry-- D_addr := C_List.First_Entry.DA D_SAP_index := C_List.First_Entry.DSAP S_SAP_index := C_List.First_Entry.SSAP DL_status := C_List.First_Entry.R_Status C_List.Remove()

Function name	Operations
GET_TE/CV_IND()	Get a service indication from the indication queue to MAC. I_List.Num_entry-- D_addr := I_List.First_Entry.DA D_SAP_index := CS S_addr := I_List.First_Entry.SA S_SAP_index := CS DLSDU := I_List.First_Entry.DLSDU I_List.Remove() SAP_List(S_SAP_index).Ibuffer.Insert() SAP_List(S_SAP_index).Ibuffer.Num_entry++ Get a service indication from the indication queue to MAC.
PUT_HREQ (Function)	Put a service request to the high prior service queue to MAC. H_List.Insert() H_List.First_Entry.DA := D_addr H_List.First_Entry.DSAP := D_SAP_index H_List.First_Entry.SSAP := S_SAP_index H_List.First_Entry.FC.Frame_type := req H_List.First_Entry.FC.Function := Function H_List.First_Entry.DLSDU := DLSDU H_List.Num_entry++
PUT_HUBUFFER (S_SAP_index)	Put a service request to the high prior update buffer of the LSAP. SAP_List(S_SAP_index).Ubuffer.High_reference := Reference SAP_List(S_SAP_index).Ubuffer.High_transmit := Transmit_strategy SAP_List(S_SAP_index).Ubuffer.High_buffer := DLSDU
PUT_LREQ (Function)	Put a service request to the low prior service queue to MAC. L_List.Insert() L_List.First_Entry.DA := D_addr L_List.First_Entry.DSAP := D_SAP_index L_List.First_Entry.SSAP := S_SAP_index L_List.First_Entry.FC.Frame_type := req L_List.First_Entry.FC.Function := Function L_List.First_Entry.DLSDU := DLSDU L_List.Num_entry++
PUT_LUBUFFER (S_SAP_index)	Put a service request to the low prior update buffer of the LSAP. SAP_List(S_SAP_index).Ubuffer.Low_reference := Reference SAP_List(S_SAP_index).Ubuffer.Low_transmit := Transmit_strategy SAP_List(S_SAP_index).Ubuffer.Low_buffer := DLSDU
RESET_LIST	Reset all activated SAP by setting its entry to invalid value and reset all services queued in the data interface to MAC sublayer. SAP_List (0..63, NIL).I/Ubuffer.Remove() until empty SAP_List (0..63, NIL) := NIL I_List.Num_entry-- I_List.Remove() until empty C_List.Num_entry-- C_List.Remove() until empty
RESET_SAP_LIST (S_SAP_index)	Reset an activated SAP by setting its entry to invalid value. SAP_List(S_SAP_index) := NIL
RESOURCE (S_SAP_index, DLSDU.Len)	Check that local SAP resources are available to handle the requested data SAP_List(S_SAP_index) ≠ NIL SAP_List(S_SAP_index).Function_List_I contains combination (Service_class, Function) DLSDU.Len ≤ DLSDU_length_list-entry of combination (Service_class, Function)
Set_current_values (Variable_name((1 to n), Index(1 to k)))	Set the management variables for confirmation to DLMS-user. LOOP for i from 1 to n for all variable names: Current_value (i) := Variables.Variable_name (i)
SET_RSAP_LIST (S_SAP_index, DLSDU_length_list)	Activate a SAP as SRD responder by setting its entries to the requested values. SAP_List(S_SAP_index).Sbuffer.Low_len) := DLSDU_length_list.Max_DLSDU_DXM_length_ind_low SAP_List(S_SAP_index).Sbuffer.High_len) := DLSDU_length_list.Max_DLSDU_DXM_length_ind_high

Function name	Operations
SET_RSAP_LIST (S_SAP_index, Access, DLSDU_length_list, Indication_mode, Publisher_enabled)	Activate a SAP as SRD responder by setting its entries to the requested values. SAP_List(S_SAP_index).Access := Access SAP_List(S_SAP_index).Indication_mode := Indication_mode SAP_List(S_SAP_index).Publisher_enabled := Publisher_enabled SAP_List(S_SAP_index).Ubuffer.Low_buffer.DLSDU.Len) := DLSDU_length_list.Max_DLSDU_length_req_low SAP_List(S_SAP_index).Ubuffer.High_buffer.DLSDU.Len) := DLSDU_length_list.Max_DLSDU_length_req_high SAP_List(S_SAP_index).Ibuffer.Low_len) := DLSDU_length_list.Max_DLSDU_length_ind_low SAP_List(S_SAP_index).Ibuffer.High_len) := DLSDU_length_list.Max_DLSDU_length_ind_high
SET_SSAP_LIST (S_SAP_index, Access, Service_list)	Activate a SAP by setting its entries to the requested values. SAP_List(S_SAP_index).Access := Access compose SAP_List(S_SAP_index).Function_list_I based on Service_list compose SAP_List(S_SAP_index).Function_list_R based on Service_list

Set_variable_list
(
Variable_name ((1 to n),
Index(1 to k)),
Desired_value (1 to n)
)
	Check that the requested management variable names are inside their ranges and set them to the list of variables and set the corresponding DLM_status:
	LOOP for i from 1 to n for all variable names:
	vn := Variable_name(i)
	if (LOWLIM(vn) ≤ Desired_value(i) ≤ HIGHLIM(vn))
	Variables.vn := Desired_value(i)
	DLM_status(i) := OK
	else
	DLM_status(i) := IV
	
	with following LOWLIM and HIGHLIM for variables of MAC sublayer:
	in_ring_desired = 0, 1
	data_rate = 9,6 … 12000 // kbit/s
	TS = 0 … 126
	Ttr = 1 … 2^24-1
	G = 1 … 100 with Tgud = G * Ttr < 2°24
	HSA = TS … 126
	max_retry_limit = 1 … 8
	SYNCHT = Octetstring with length 0 or 2
	Tct = 1..2^32-1
	maxTsh = 1..256
	Tcsi = 1 … 2^32-1
	HW_Release = Visible String with length 0 … 32
	SW_Release = Visible String with length 0 … 32
	with following LOWLIM and HIGHLIM for variables of SRC sublayer:
	minTsdr = 1 … 2^16-1
	maxTsdr = minTdsr … 2^16-1
	Tsl = maxTsdr … 2^16-1
	Tqui = 0 … 255
	Tset = 1 … 255
	DLPDU_sent_cou nt = 0
	Retry_count = 0
	DLPDU_sent_count_sr(1 to n) = 0
	Error_count(1 to n) = 0
	SD_count = 0
	SD_error_count = 0
	with following LOWLIM and HIGHLIM for variables of PHY sublayer:
	Medium_redundancy = 0, 1
	Interface_mode = "FULL_DUPLEX" or "HALF_DUPLEX"
	Loop_back_mode = "DISABLED" or "in MDS" or "in MAU"
	Preamble_extension = 0..7
	Tptg = 0..7
	Tics = 0..7
	Transmitter_output_channel (1 to 8) = "ENABLED" or "DISABLED"
	Receiver_inpiut_channel (1 to 8) = "ENABLED" or "DISABLED"
	Preferred_receive_channel = "NONE" or 1..8

Write_ident_list ( )	Compose parameter Ident_list for confirmation to DLMS user. Ident_list := Ident_List.Vendor_name + Ident_List.Model_name + Ident_List.HW_release + Ident_List.SW_release

Function name	Operations
Write_SAPstatus_list ( )	Compose parameters Access, Service_type, Role_in_service_list for confirmation to DLMS user. Access := SAP_List(D_SAP_index).Access compose Service_type(1 to n) based on SAP_List(D_SAP_index).Sevice_listI/R compose Role_in_service_list(1 to n) based on SAP_List(D_SAP_index).Sevice_listI/R

A.5 MAC

A.5.1 Primitive definitions

A.5.1.1 Primitives exchanged between DLM and MAC

Table A.11 shows the primitives issued by the DLM to the MAC.

Table A.11 – Primitives issued by DLM to MAC

Primitive name	Associated parameters
MAC_RESET.req	none

Table A.12 shows the primitives issued by the MAC to the DLM.

Table A.12 – Primitives issued by MAC to DLM

Primitive name	Associated parameters
MAC_RESET.cnf	none
MAC_LFAULT.ind	Fault_type
MAC_BFAULT.ind	Fault_type Tj

A.5.1.2 Parameters of MAC primitives

Table A.13 shows the parameters used with primitives exchanged between the DLM and the MAC.

Table A.13 – Parameters used with primitives exchanged between DLM and MAC

Parameter name	Description
Fault_type	This parameter contains the error type. Possible values: State_conflict, Faulty_transceiver Double_token, Duplicate_address, Not_synchronized, Out_of_ring, Time_out, Hsa_error, In_ring
Tj	Jitter time in Isochronous Mode

A.5.2 State machine description

A.5.2.1 General

The Media Access Control is responsible for the message exchange and control of the various components using the media.

For a token based system, a main focus of this state machine is the token handling in all situations (see 5.3.2). This will be done in the states LISTEN-TOKEN, CLAIM-TOKEN, ACTIVE-IDLE (token-receipt) and PASS-TOKEN. To identify new stations ready to enter the ring, a FDL-Status request will be executed periodically. The reply will be checked in the state AWAIT-STATUS-RESPONSE. The state WAIT-TCT is used in isochronous mode to keep the interval between SYNCH requests constant.

The other main task is the correct execution of service sequences (Request, Response) including retries and duplication detection. This will be done by checking the high and low priority request queues when the station is token holder (state USE-TOKEN). In the state AWAIT-DATA-RESPONSE the reply to the processed service should be received. The state CHECK-TOKEN-PASS is used for checking the Token-hold-time.

The incoming service indications are processed in the states ACTIVE-IDLE, PASSIVE-IDLE and CHECK-TOKEN-PASS. All valid services received will be put into the indication queue and the appropriate reply activity will be invoked.

All local variables of the MAC are shown in Table A.14.

Table A.14 – Local MAC variables

Struct element	Type	Range	Remark
FCB	[0..126] of U8	0,1	—
FCV	[0..126] of U8	0,1	—
REQM	S4	—	—
RESM	S5	—	—
Cnf	Bool	—	Single variables
Dup_add_count	U8	0, 1	—
Gap_lp_cnt	U16	—	—
Gap_to_do	Bool	—	—
Gud_timer	U32	—	—
Isochronous_Start	Bool	—	—
LMS_cnt	U16	—	—
Req_h_cnt	U16	—	—
Retry_cnt	U8	0..15	—
Second	U8	0, 1	—
Token_holder	U8	0..126, NIL	—
Tok_cnt	U8	—	—
Tok_err_cnt	U8	—	—
Trr	U24	—	—
T_cnt	U16	0..259	—
TRT	—	—	Token Rotation Timer
cv	U24	—	Current Value of
Start(Ttr)	Trigger	—	Starting count down (from Ttr)

A.5.2.2 MAC state table

The MAC state table is shown in Table A.15.

Table A.15 – MAC state table

No.	Current state	Event /condition ⇒action	Next state
1	OFFL	/H_List.Num_entry ≠ 0 ⇒ SETUP_HCON_DS	OFFL
2	OFFL	/L_List.Num_entry ≠ 0 ⇒ SETUP_LCON_DS	OFFL
3	OFFL	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
4	OFFL	/Data_Rate ≠ NIL && In_ring_desired ⇒ INIT_FCBV_LIST, RESM := empty, INIT_LMS, Isochronous_Start = TRUE,T_cnt:=0	LISTEN
5	OFFL	/Data_Rate ≠ NIL && !In_ring_desired ⇒ RESM := empty	PASSIVE_I
6	OFFL	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
7	OFFL	SRC_RECEIVE_ERROR.ind ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
8	OFFL	SRC_RECEIVE_TOKEN.ind (DA, SA) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
9	OFFL	SRC_SEND_DATA.cnf ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
10	OFFL	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
11	OFFL	SRC_SLOT_EVENT.ind ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
12	OFFL	SRC_SYNI_EVENT.ind ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
13	LISTEN	/!In_ring_desired ⇒	PASSIVE_I
14	LISTEN	/H_List.Num_entry ≠ 0 ⇒ SETUP_HCON_DS	LISTEN
15	LISTEN	/L_List.Num_entry ≠ 0 ⇒ SETUP_LCON_DS	LISTEN
16	LISTEN	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL

No.	Current state	Event /condition ⇒action	Next state
17	LISTEN	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) ⇒ T_cnt := 0	LISTEN
18	LISTEN	SRC_RECEIVE_ERROR.ind ⇒ T_cnt := 0	LISTEN
19	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /SA = TS || DA = TS && Dup_add_count > 0 ⇒ Fault_type := Duplicate_address, Tok_cnt--, T_cnt := 0, Dup_add_count := 0 MAC_BFAULT.ind (Fault_type)	OFFL
20	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /DA = TS || SA = TS && Dup_add_count = 0 ⇒ Dup_add_count++, Tok_cnt--,T_cnt := 0	LISTEN
21	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /(DA ≠ TS && SA ≠ TS) && (DA > HSA || SA > HSA) && Dup_hsa_count > 0 ⇒ Fault_type := Hsa_error, Tok_cnt--, Dup_hsa_count := 0, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	OFFL
22	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /(DA ≠ TS && SA ≠ TS) && (DA > HSA || SA > HSA) && Dup_hsa_count = 0 ⇒ Dup_hsa_count++, Tok_cnt--,T_cnt := 0	LISTEN
23	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /DA ≠ TS && SA ≠ TS && DA =< HSA && SA =< HSA && First = NIL ⇒ BUILD_LMS(DA), Token_holder := DA, Tok_cnt--,T_cnt := 0,First:=SA	LISTEN
24	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /DA ≠ TS && SA ≠ TS && DA =< HSA && SA =< HSA && First ≠ NIL && (SA ≠ Token_holder || Tok_cnt=0) ⇒ INIT_LMS, BUILD_LMS(DA), Token_holder := DA,Tok_cnt--, T_cnt := 0, First:=DA	LISTEN
25	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /DA ≠ TS && SA ≠ TS && DA =< HSA && SA =< HSA && First ≠ NIL && SA = Token_holder && Tok_cnt > 0 && DA ≠ First && LMS_cnt = 0 ⇒ LMS_UPDATE(DA, SA), Token_holder := DA ,Tok_cnt--, T_cnt := 0	LISTEN
26	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /DA ≠ TS && SA ≠ TS && DA =< HSA && SA =< HSA && First ≠ NIL && SA = Token_holder && Tok_cnt > 0 && DA = First && LMS_cnt = 0 ⇒ LMS_UPDATE(DA, SA), Token_holder := DA, LMS_cnt := 1 , Tok_cnt--, T_cnt := 0	LISTEN
27	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /DA ≠ TS && SA ≠ TS && DA =< HSA && SA =< HSA && First ≠ NIL && SA = Token_holder && LMS[SA]=DA && Tok_cnt > 0 && DA ≠ First && LMS_cnt = 1 ⇒ Token_holder := DA, Tok_cnt--, T_cnt := 0	LISTEN
28	LISTEN	SRC_RECEIVE_TOKEN.ind (DA, SA) /DA ≠ TS && SA ≠ TS && DA =< HSA && SA =< HSA && First ≠ NIL && SA = Token_holder && LMS[SA]=DA && Tok_cnt > 0 && DA = First && LMS_cnt = 1 ⇒ Tok_cnt--, T_cnt := 0, LMS_cnt++, RECV_ERR_cnt := 0	ACTIVE_I
29	LISTEN	SRC_SEND_DATA.cnf ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL

No.	Current state	Event /condition ⇒action	Next state
30	LISTEN	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
31	LISTEN	SRC_SLOT_EVENT.ind /!TIME_OUT ⇒ T_cnt ++	LISTEN
32	LISTEN	SRC_SLOT_EVENT.ind /TIME_OUT ⇒ Fault_type := Time_out, T_cnt := 0, HoldToken:=1, RECV_ERR_cnt:=0, TRT.start(0) MAC_BFAULT.ind (Fault_type)	CLAIM_T
33	LISTEN	SRC_SYNI_EVENT.ind ⇒ Fault_type := Not_synchronized, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	LISTEN
34	ACTIVE_I	/!In_ring_desired ⇒	PASSIVE_I
35	ACTIVE_I	/LMS[TS] = NIL && H_List.Num_entry ≠ 0 ⇒ SETUP_HCON_DS	ACTIVE_I
36	ACTIVE_I	/LMS[TS] = NIL && L_List.Num_entry ≠ 0 ⇒ SETUP_LCON_DS	ACTIVE_I
37	ACTIVE_I	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
38	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = SA && FC.Frame = req && FC.Function = FDL_status && Isochronous_mode>0 ⇒ TRT_OFF	ACTIVE_I
39	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = FDL_status && LMS(TS) = NIL ⇒ DLSDU,SSAP,DSAP:=NIL, FC.Frame := rsp, FC.Stn-Type := M_rdy, FC.Function := OK , RESM := empty, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
40	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = FDL_status && LMS(TS) ≠ NIL ⇒ DLSDU,SSAP,DSAP:=NIL, FC.Frame := rsp, FC.Stn-Type := M_in_ring, FC.Function := OK , RESM := empty, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
41	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = Ident ⇒ DA := SA, SA := TS, DSAP <:⇒ SSAP, RESM := empty, IDENT, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
42	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = DLSAP_status ⇒ DA := SA, SA := TS, DSAP <:⇒ SSAP, RESM := empty, LSAP_STATUS, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
43	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =127) && FC.Frame = res && SAP_CHECK(DSAP) && B_BUF(DSAP) ⇒ RESM := empty, SETUP_SIND(0,NO), T_cnt := 0	ACTIVE_I
44	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =127) && FC.Frame = res && (!SAP_CHECK(DSAP) || !B_BUF(DSAP)) ⇒ RESM := empty, T_cnt := 0	ACTIVE_I
45	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function =TE || FC.Function = CV) && SAP_CHECK(CS) && I_BUF(CS) ⇒ RESM := empty, SETUP_IND(0,NO), T_cnt := 0, TRT_ON	ACTIVE_I
46	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = CV || FC.Function = TE) && (!SAP_CHECK(CS) || !I_BUF(CS)) ⇒ RESM := empty, T_cnt := 0, TRT_ON	ACTIVE_I
47	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = SDN_H || FC.Function = SDN_L) && SAP_CHECK(DSAP) && I_BUF(DSAP) ⇒ RESM := empty, SETUP_IND(0,NO), T_cnt := 0, TRT_ON	ACTIVE_I
48	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = SDN_H || FC.Function = SDN_L) && (!SAP_CHECK(DSAP) || !I_BUF(DSAP)) ⇒ RESM := empty, T_cnt := 0, TRT_ON	ACTIVE_I
49	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_L || FC.Function = SDA_H || FC.Function = SRD_L || FC.Function = SRD_H || FC.Function = SRD_BCT) && RETRY ⇒ DA := RESM.DA, SA := TS, FC := RESM.FC, DSAP := RESM.DSAP, SSAP := RESM.SSAP, DLSDU := RESM.DLSDU, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
50	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_H || FC.Function = SDA_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) ⇒ SETUP_IND(0,NO), FC.Function := SC, SETUP_RESM, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
51	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_L || FC.Function = SDA_H) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RR, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
52	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) && U_BUF(DSAP) ⇒ SETUP_REPLY, if(SAP_List[DSAP].Indication_Mode = ALL || Upd_status ≠ NO) SETUP_IND(Ref, Upd_status), DA := R_DA, SA := TS, FC.Function := Upd_status, FC.Frame := rsp, SETUP_STN_TYPE, DSAP ≤> SSAP, DLSDU := R_SDU, SETUP_RESM, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
53	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) && U_BUF(DSAP) ⇒ SETUP_REPLY,DA := R_DA, SA := TS, FC.Function := R_FUNCTION, FC.Frame := rsp, SETUP_STN_TYPE, DSAP <:⇒ SSAP, DLSDU := R_SDU, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
54	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) && !U_BUF(DSAP) ⇒ SETUP_IND(0,NO), FC.Function := SC, SETUP_RESM, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
55	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) && !U_BUF(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RR, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
56	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L || FC.Function = SDA_H || FC.Function = SDA_L) && !RETRY && !SAP_CHECK(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RS, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
57	ACTIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA ≠ TS && (DA≠127 || (FC.Function ≠ SDN_H && FC.Function ≠ SDN_L)) || FC.Frame ≠ req || INVALID_FUNCTION) && !(DA = SA && FC.Frame = req && FC.Function = FDL_status && Isochronous_mode=0) ⇒ RESM := empty, T_cnt := 0, TRT_ON	ACTIVE_I
58	ACTIVE_I	SRC_RECEIVE_ERROR.ind ⇒ T_cnt := 0, TRT_ON	ACTIVE_I
59	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /DUPLICATE_ADDRESS && Dup_add_count > 0 ⇒ INIT_FCBV_LIST, INIT_LMS, RESM := empty, Dup_add_count := 0, Fault_type := Duplicate_address, Second := 0, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	LISTEN
60	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /DUPLICATE_ADDRESS && Dup_add_count = 0 ⇒ Dup_add_count++, TOK_CNT_UPD, Second := 0, RESM := empty, T_cnt := 0, TRT_ON	ACTIVE_I
61	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt > (Limit – 2) ⇒ INIT_FCBV_LIST, INIT_LMS, RESM := empty, Fault_type := Out_of_ring, Second := 0, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	LISTEN
62	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && TOKEN_ERROR ⇒ RESM := empty, TOK_CNT_UPD, Tok_err_cnt++, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
63	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && LMS[SA] ≠ DA && DA ≠ TS ⇒ RESM := empty, LMS_UPDATE(DA, SA), TOK_CNT_UPD, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I
64	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && LMS[SA] = DA && DA ≠ TS ⇒ RESM := empty, TOK_CNT_UPD, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I
65	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[SA] = DA && Tct=0 && H_list.Num_entry ≠ 0 ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, Second := 0, Req_h_cnt := H_list.Num_entry, SETUP_HREQ, T_cnt := 0, TRT_ON, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
66	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second = 0 ⇒ RESM := empty, Second++, T_cnt := 0, TRT_ON	ACTIVE_I
67	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second > 0 && Tct = 0 && H_list.Num_entry ≠ 0 ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, LMS_UPDATE(DA, SA), Second := 0, Req_h_cnt := H_list.Num_entry, SETUP_HREQ, T_cnt := 0, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
68	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[DA] = NIL && ((LMS[SA] > SA && LMS[SA] > TS && SA < TS) || (LMS[SA] ≤ SA && ((SA < TS) || (TS < LMS[SA])))) ⇒ RESM := empty, TTH_INIT, LMS_UPDATE(DA, SA), TOK_CNT_UPD, Second := 0, GAP_INIT(0), Fault_Type := In_ring, T_cnt := 0, TRT_ON MAC_BFAULT.ind (Fault_type)	ACTIVE_I
69	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[SA] = DA && Tct=0 && H_list.Num_entry = 0 ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, Second := 0, Req_h_cnt := H_list.Num_entry, T_cnt := 0, TRT_ON, HoldToken:=1	CHECK_A
70	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second > 0&&Tct=0 && H_list.Num_entry = 0 ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, LMS_UPDATE(DA, SA), Second := 0, Req_h_cnt := H_list.Num_entry, T_cnt := 0, TRT_ON, HoldToken:=1	CHECK_A
71	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[SA] = DA && Tct > 0 ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, Second := 0, Req_h_cnt := H_list.Num_entry, DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_OFF, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	WAIT_TCT

No.	Current state	Event /condition ⇒action	Next state
72	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second > 0&& Tct > 0 ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE(DA, SA), TOK_CNT_UPD, LMS_UPDATE, Second := 0, Req_h_cnt := H_list.Num_entry, DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, DLSDU,SSAP,DSAP:=NIL,T_cnt := 0, TRT_OFF, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	WAIT_TCT
73	ACTIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt ≤ (Limit – 2) && !TOKEN_ERROR && DA = TS && (!LMS[LMS[SA]]=DA && !(LMS[DA] = NIL && ((LMS[SA] > SA && LMS[SA] > TS && SA < TS) || (LMS[SA] ≤ SA && ((SA < TS) || (TS < LMS[SA]))))) ⇒ RESM := empty, LMS_UPDATE(DA, SA), TOK_CNT_UPD, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I
74	ACTIVE_I	SRC_SEND_DATA.cnf ⇒	ACTIVE_I
75	ACTIVE_I	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
76	ACTIVE_I	SRC_SLOT_EVENT.ind /!TIME_OUT ⇒ T_cnt ++, TRT_ON	ACTIVE_I
77	ACTIVE_I	SRC_SLOT_EVENT.ind /TIME_OUT ⇒ RESM := empty, Fault_type := Time_out, T_cnt := 0, TRT_ON, HoldToken:=1 MAC_BFAULT.ind (Fault_type)	CLAIM_T
78	ACTIVE_I	SRC_SYNI_EVENT.ind ⇒ Fault_type := Not_synchronized, T_cnt := 0, TRT_ON MAC_BFAULT.ind (Fault_type)	ACTIVE_I
79	CLAIM_T	/LMS[TS] ≠ NIL && H_list.Num_entry ≠ 0 && Tct=0 ⇒ GAP_UPDATE, TTH_UPDATE, Req_h_cnt := H_list.Num_entry, SETUP_HREQ, Isochronous_Start = TRUE SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
80	CLAIM_T	/LMS[TS] ≠ NIL && H_list.Num_entry = 0 && Tct=0 ⇒ GAP_UPDATE, TTH_UPDATE, Req_h_cnt := H_list.Num_entry, Isochronous_Start = TRUE	CHECK_A
81	CLAIM_T	/LMS[TS] ≠ NIL && Tct > 0 ⇒ GAP_UPDATE, TTH_UPDATE, Req_h_cnt := H_list.Num_entry, DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, DLSDU,SSAP,DSAP:=NIL SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	WAIT_TCT
82	CLAIM_T	/LMS[TS] = NIL ⇒ INIT_LMS, BUILD_LMS(TS), Second := 0, Fault_Type := In_ring, Retry_cnt := 0, DA := TS, SA := TS, TTH_INIT, if (NS := (TS+1) mod (HSA+1)) Gap_to_do := FALSE else Gap_to_do := TRUE, Gap_Ip_cnt := 0 MAC_BFAULT.ind (Fault_type), SRC_SEND_TOKEN.req (DA, SA)	PASS_T
83	WAIT_TCT	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL

No.	Current state	Event /condition ⇒action	Next state
84	WAIT_TCT	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
85	WAIT_TCT	SRC_RECEIVE_ERROR.ind ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
86	WAIT_TCT	SRC_RECEIVE_TOKEN.ind (DA, SA) ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
87	WAIT_TCT	SRC_SEND_DATA.cnf /Isochronous_Start ⇒ Tct:= TCT.cv, TCT.start(0), Fault_type:=SYNCH, DA:=0x7f,SSAP:=SYNCH.SSAP,DSAP:=SYNCH.DSAP FC.Function:=SDN_H,FC.Frame:=req, FC.FCB:=0, FC.FCV:=0, DLSDU:=SYNCH.DLSDU, Synch := TRUE, Cnf := FALSE, TRT_ON Isochronous_Start = FALSE SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU) MAC_BFAULT.ind (Fault_type)	USE_T
88	WAIT_TCT	SRC_SEND_DATA.cnf /!Isochronous_Start && Isochronous_mode=1 && TCT.cv ≥ Tct ⇒ Tsh :=TCT.cv-Tct, TCT.start(2*Tct-TCT.cv), Fault_type:=Synch_Delay, DA:=0x7f,SSAP:=SYNCH.SSAP,DSAP:=SYNCH.DSAP FC.Function:=SDN_H,FC.Frame:=req, FC.FCB:=0, FC.FCV:=0, DLSDU:=SYNCH.DLSDU, Synch := TRUE, Cnf := FALSE, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU) MAC_BFAULT.ind (Fault_type,Tsh)	USE_T
89	WAIT_TCT	SRC_SEND_DATA.cnf /!Isochronous_Start && Isochronous_mode=2 && TCT.cv ≥ Tct ⇒ Tsh :=Tct, TCT.start(2*Tct-TCT.cv), Fault_type:=SynchDelay, DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, FC.FCB:=0, FC.FCV:=0, DLSDU,SSAP,DSAP:=NIL SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU) MAC_BFAULT.ind(Fault_Type,Tsh)	WAIT_TCT
90	WAIT_TCT	SRC_SEND_DATA.cnf /!Isochronous_Start && Isochronous_mode=0 && TCT.cv ≥ Tct && H_list.Num_entry ≠ 0 ⇒ RESM := empty, Req_h_cnt := H_list.Num_entry, SETUP_HREQ SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
91	WAIT_TCT	SRC_SEND_DATA.cnf /!Isochronous_Start && Isochronous_mode=0 && TCT.cv ≥ Tct && H_list.Num_entry = 0 ⇒ RESM := empty, Req_h_cnt := H_list.Num_entry	CHECK_A
92	WAIT_TCT	SRC_SEND_DATA.cnf /!Isochronous_Start && Tct-Tpsp ≤ TCT.cv < Tct ⇒ TPSP.start(Tct-TCT.cv)	WAIT_TCT

No.	Current state	Event /condition ⇒action	Next state
93	WAIT_TCT	SRC_SEND_DATA.cnf /!Isochronous_Start && TCT.cv < Tct-Tpsp ⇒ DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, FC.FCB:=0, FC.FCV:=0, DLSDU,SSAP,DSAP:=NIL SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	WAIT_TCT
94	WAIT_TCT	TPSP expired /Isochronous_mode>0 ⇒ TCT.start(2*Tct-TCT.cv), Fault_type := SYNCH, DA:=0x7f,SSAP:=SYNCH.SSAP,DSAP:=SYNCH.DSAP FC.Function:=SDN_H,FC.Frame:=req, FC.FCB:=0, FC.FCV:=0, DLSDU:=SYNCH.DLSDU, Synch := TRUE, Cnf := FALSE, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU) MAC_BFAULT.ind(Fault_Type)	USE_T
95	WAIT_TCT	TPSP expired / (Isochronous_Mode=0 || Isochronous_Mode=3) && H_list.Num_entry ≠ 0 ⇒ RESM := empty, Req_h_cnt := H_list.Num_entry, SETUP_HREQ SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
96	WAIT_TCT	TPSP expired / (Isochronous_Mode=0 || Isochronous_Mode=3) && H_list.Num_entry = 0 ⇒ RESM := empty, Req_h_cnt := H_list.Num_entry, T_cnt := 0	CHECK_A
97	WAIT_TCT	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
98	WAIT_TCT	SRC_SLOT_EVENT.ind ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
99	WAIT_TCT	SRC_SYNI_EVENT.ind ⇒ Fault_type := Not_synchronized, T_cnt := 0, TRT_ON, HoldToken:=0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I
100	USE_T	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
101	USE_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
102	USE_T	SRC_RECEIVE_ERROR.ind ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
103	USE_T	SRC_RECEIVE_TOKEN.ind (DA, SA) ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
104	USE_T	SRC_SEND_DATA.cnf /Cnf && !Gap_in_action ⇒ DLPDU_sent_count++	AW_DATA
105	USE_T	SRC_SEND_DATA.cnf /!Cnf && !Synch ⇒ SETUP_CON(OK, NIL)	CHECK_A

No.	Current state	Event /condition ⇒action	Next state
106	USE_T	SRC_SEND_DATA.cnf /!Cnf && Synch && TCT.cv >143+ maxTsh ⇒ Synch := False,Fault_type:=SynchDelay Tsh := TCT.cv-143 MAC_BFAULT.ind (Fault_type,Tsh)	CHECK_A
107	USE_T	SRC_SEND_DATA.cnf /!Cnf && Synch&& TCT.cv =<143+ maxTsh ⇒ Synch = False	CHECK_A
108	USE_T	SRC_SEND_DATA.cnf /Cnf && Gap_in_action ⇒ Gap_in_action := FALSE	AW_STATUS
109	USE_T	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
110	USE_T	SRC_SLOT_EVENT.ind ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
111	USE_T	SRC_SYNI_EVENT.ind /Gap_in_action ⇒ Gap_in_action := FALSE, Fault_type := Not_synchronized, T_cnt := 0, HoldToken:=0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I
112	USE_T	SRC_SYNI_EVENT.ind /!Gap_in_action ⇒ FCV_CLEAR, SETUP_CONM(DS), Fault_type := Not_synchronized, HoldToken:=0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I
113	AW_DATA	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
114	AW_DATA	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(FC.Frame = req || (FC.Function ≠ SC && DA ≠ TS)) ⇒ FCV_CLEAR, SETUP_CONM(DS), Fault_type := Double_token, T_cnt := 0, HoldToken:=0, if (RECV_ERR_cnt>0) RECV_ERR_cnt-- MAC_BFAULT.ind (Fault_type)	ACTIVE_I
115	AW_DATA	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /FC.Frame = rsp && DA = TS && FC.Function ≠ SC && (REQM.DA≠SA || (DSAP,SSAP ≠ Nil && (REQM.DSAP ≠ SSAP || REQM.SSAP ≠ DSAP))) ⇒ FCV_CLEAR, SETUP_CONM(DS), T_cnt := 0, HoldToken:=0, if (RECV_ERR_cnt>0) RECV_ERR_cnt--	ACTIVE_I
116	AW_DATA	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) / FC.Frame = rsp && DA = TS && FC.Function ≠ SC && REQM.DA=SA && (DSAP,SSAP =Nil || (REQM.DSAP = SSAP && REQM.SSAP = DSAP)) ⇒ SETUP_CON(FC, DLSDU), T_cnt := 0, FCB_UPDATE , if (RECV_ERR_cnt>0) RECV_ERR_cnt--	CHECK_A
117	AW_DATA	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) / FC.Frame = rsp && FC.Function =SC ⇒ SETUP_CON(FC, DLSDU), T_cnt := 0, FCB_UPDATE, HoldToken:=0 , if (RECV_ERR_cnt>0) RECV_ERR_cnt--	CHECK_A

No.	Current state	Event /condition ⇒action	Next state
118	AW_DATA	SRC_RECEIVE_ERROR.ind /RECV_ERR_cnt ≥ RECV_ERR_limit ⇒ FCV_CLEAR, SETUP_CONM(DS), T_cnt := 0, RECV_ERR_cnt+=2	LISTEN
119	AW_DATA	SRC_RECEIVE_ERROR.ind /!(Isochronous_mode=2 && REQM.serv_class high) && Retry_cnt > 0 && FCV = 1 && RECV_ERR_cnt < RECV_ERR_limit ⇒ Error_count[SA]--, DA := REQM.DA, SA := TS, FC := REQM.FC, DSAP := REQM.DSAP, SSAP := REQM.SSAP, DLSDU := REQM.DLSDU, Retry_cnt--, T_cnt := 0, RECV_ERR_cnt+=2 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
120	AW_DATA	SRC_RECEIVE_ERROR.ind /!(Isochronous_mode=2 && REQM.serv_class = high) && (Retry_cnt = 0 || FCV = 0) && RECV_ERR_cnt < RECV_ERR_limit ⇒ FCV_CLEAR, SETUP_CONM(NA), T_cnt := 0, RECV_ERR_cnt+=2	CHECK_A
121	AW_DATA	SRC_RECEIVE_ERROR.ind /Isochronous_mode=2 && REQM.serv_class = high && RECV_ERR_cnt < RECV_ERR_limit ⇒ SETUP_CONM(NA), T_cnt := 0, RECV_ERR_cnt+=2	CHECK_A
122	AW_DATA	SRC_RECEIVE_TOKEN.ind (DA, SA) /In_ring_desired ⇒ FCV_CLEAR, SETUP_CONM(DS), Fault_type := Double_token, T_cnt := 0, HoldToken:=0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I
123	AW_DATA	SRC_SEND_DATA.cnf ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
124	AW_DATA	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
125	AW_DATA	SRC_SLOT_EVENT.ind /!(Isochronous_mode=2 && REQM.serv_class = high) && Retry_cnt > 0 && FCV = 1 ⇒ Error_count[SA]--, DA := REQM.DA, SA := TS, FC := REQM.FC, DSAP := REQM.DSAP, SSAP := REQM.SSAP, DLSDU := REQM.DLSDU, Retry_cnt-- SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
126	AW_DATA	SRC_SLOT_EVENT.ind /!(Isochronous_mode=2 && REQM.serv_class = high) && Retry_cnt = 0 || FCV = 0 ⇒ FCV_CLEAR, SETUP_CONM(NA)	CHECK_A
127	AW_DATA	SRC_SLOT_EVENT.ind /Isochronous_mode=2 && REQM.serv_class = high ⇒ SETUP_CONM(NA)	CHECK_A
128	AW_DATA	SRC_SYNI_EVENT.ind /In_ring_desired ⇒ FCV_CLEAR, SETUP_CONM(DS), Fault_type := Not_synchronized , HoldToken:=0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
129	CHECK_A	/ (TTH_AVAILABLE || Isochronous_Mode=2 || Isochronous_Mode=3) && Req_h_cnt ≠ 0 ⇒ SETUP_HREQ, Req_h_cnt--, SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
130	CHECK_A	/TTH_AVAILABLE && Req_h_cnt = 0 && Gap_lp_cnt = 0 ⇒ FC.Frame := req, FC.FCB := 0, FC.FCV :=0, FC.Function := FDL_status, DA := Gap_address, SA := TS, DLSDU,SSAP,DSAP:=NIL, Cnf := TRUE, Gap_in_action := TRUE SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
131	CHECK_A	/TTH_AVAILABLE && Req_h_cnt = 0 && Gap_lp_cnt ≠ 0 && L_list.Num_entry ≠ 0 ⇒ SETUP_LREQ, SA := TS SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
132	CHECK_A	/!((TTH_AVAILABLE || Isochronous_Mode=2 || Isochronous_Mode=3) && Req_h_cnt <> 0) || !TTH_AVAILABLE || (Req_h_cnt = 0 && Gap_lp_cnt <> 0 && L_list.Num_entry = 0) ⇒ Second := 0, DA := NS(TS), SA := TS, Retry_cnt:=0 SRC_SEND_TOKEN.req (DA, SA)	PASS_T
133	PASS_T	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
134	PASS_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
135	PASS_T	SRC_RECEIVE_ERROR.ind ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
136	PASS_T	SRC_RECEIVE_TOKEN.ind (DA, SA) ⇒ Fault_type := State_conflict, T_cnt := 0 MAC_LFAULT.ind (Fault_type)	OFFL
137	PASS_T	SRC_SEND_DATA.cnf ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
138	PASS_T	SRC_SEND_TOKEN.cnf(Status) /Status = token_pass_failed && Second = 0 ⇒ Second++, DS := NS(TS), SA := TS SRC_SEND_TOKEN.req (DA, SA)	PASS_T
139	PASS_T	SRC_SEND_TOKEN.cnf(Status) /Status = token_pass_failed && Second > 0 ⇒ Second := 0, Fault_type := Faulty_transceiver MAC_BFAULT.ind (Fault_type)	OFFL
140	PASS_T	SRC_SEND_TOKEN.cnf(Status) /Status = no_token_pass ⇒ Second := 0, Fault_type := Faulty_transceiver MAC_BFAULT.ind (Fault_type)	OFFL

No.	Current state	Event /condition ⇒action	Next state
141	PASS_T	SRC_SEND_TOKEN.cnf(Status) /NS = TS && Status = OK && !In_ring_desired ⇒ GAP_UPDATE, TTH_UPDATE, Req_h_cnt := H_list.Num_entry SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
142	PASS_T	SRC_SEND_TOKEN.cnf(Status) /NS = TS && Status = OK && In_ring_desired && Tct > 0 ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, Second := 0, Req_h_cnt := H_list.Num_entry, DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_OFF SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	WAIT_TCT
143	PASS_T	SRC_SEND_TOKEN.cnf(Status) /NS = TS && Status = OK && H_list.Num_entry ≠ 0 && In_ring_desired && Tct=0 ⇒ GAP_UPDATE, TTH_UPDATE, Req_h_cnt := H_list.Num_entry SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
144	PASS_T	SRC_SEND_TOKEN.cnf(Status) /NS = TS && Status = OK && H_list.Num_entry = 0 && In_ring_desired && Tct=0 ⇒ GAP_UPDATE, TTH_UPDATE, Req_h_cnt := H_list.Num_entry	CHECK_A
145	PASS_T	SRC_SEND_TOKEN.cnf(Status) /NS ≠ TS && Status = OK ⇒ HoldToken:=0	CHECK_T
146	PASS_T	SRC_SLOT_EVENT.ind ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
147	PASS_T	SRC_SYNI_EVENT.ind /In_ring_desired ⇒ Fault_type := Not_synchronized, HoldToken:=0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I
148	CHECK_T	/LMS[TS] = NIL && H_List.Num_entry ≠ 0 ⇒ SETUP_HCON_DS	CHECK_T
149	CHECK_T	/LMS[TS] = NIL && L_List.Num_entry ≠ 0 ⇒ SETUP_LCON_DS	CHECK_T
150	CHECK_T	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
151	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = SA && FC.Frame = req && FC.Function = FDL_status && Tct > 0 ⇒ TRT_OFF	ACTIVE_I
152	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = FDL_status && LMS(TS) = NIL ⇒ DLSDU,SSAP,DSAP:=NIL, FC.Frame := rsp, FC.Stn-Type := M_rdy, FC.Function := OK , RESM := empty, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
153	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = FDL_status && LMS(TS) ≠ NIL ⇒ DLSDU,SSAP,DSAP:=NIL, FC.Frame := rsp, FC.Stn-Type := M_in_ring, FC.Function := OK , RESM := empty, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
154	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = Ident ⇒ DA := SA, SA := TS, DSAP <:⇒ SSAP, RESM := empty, IDENT, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
155	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = LSAP_status ⇒ DA := SA, SA := TS, DSAP <:⇒ SSAP, RESM := empty, LSAP_STATUS, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
156	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =127) && FC.Frame = res && SAP_CHECK(DSAP) && B_BUF(DSAP) ⇒ RESM := empty, SETUP_SIND(0,NO), T_cnt := 0	ACTIVE_I
157	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =127) && FC.Frame = res && (!SAP_CHECK(DSAP) || !B_BUF(DSAP)) ⇒ RESM := empty, T_cnt := 0	ACTIVE_I
158	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function =TE || FC.Function = CV) && SAP_CHECK(CS) && I_BUF(CS) ⇒ RESM := empty, SETUP_IND(0,NO), T_cnt := 0, TRT_ON	ACTIVE_I
159	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = CV || FC.Function = TE) && (!SAP_CHECK(CS) || !I_BUF(CS)) ⇒ RESM := empty, T_cnt := 0, TRT_ON	ACTIVE_I
160	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = SDN_H || FC.Function = SDN_L) && SAP_CHECK(DSAP) && I_BUF(DSAP) ⇒ RESM := empty, SETUP_IND(0,NO), T_cnt := 0, TRT_ON	ACTIVE_I
161	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = SDN_H || FC.Function = SDN_L) && (!SAP_CHECK(DSAP) || !I_BUF(DSAP)) ⇒ RESM := empty, T_cnt := 0, TRT_ON	ACTIVE_I
162	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_L || FC.Function = SDA_H || FC.Function = SRD_L || FC.Function = SRD_H || FC.Function = SRD_BCT) && RETRY ⇒ DA := RESM.DA, SA := TS, FC := RESM.FC, DSAP := RESM.DSAP, SSAP := RESM.SSAP, DLSDU := RESM.DLSDU, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
163	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_H || FC.Function = SDA_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) ⇒ SETUP_IND(0,NO), FC.Function := SC, SETUP_RESM, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
164	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_L || FC.Function = SDA_H) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RR, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
165	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) && U_BUF(DSAP) ⇒ SETUP_REPLY, if(SAP_List[DSAP].Indication_Mode = ALL || Upd_status ≠ NO) SETUP_IND(Ref, Upd_status), DA := R_DA, SA := TS, FC.Function := Upd_status, FC.Frame := rsp, SETUP_STN_TYPE, DSAP ≤> SSAP, DLSDU := R_SDU, SETUP_RESM, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
166	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) && U_BUF(DSAP) ⇒ SETUP_REPLY,DA := R_DA, SA := TS, FC.Function := R_FUNCTION, FC.Frame := rsp, SETUP_STN_TYPE, DSAP <:⇒ SSAP, DLSDU := R_SDU, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
167	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) && !U_BUF(DSAP) ⇒ SETUP_IND(0,NO), FC.Function := SC, SETUP_RESM, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
168	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) && !U_BUF(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RR, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
169	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L || FC.Function = SDA_H || FC.Function = SDA_L) && !RETRY && !SAP_CHECK(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RS, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_ON SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	ACTIVE_I
170	CHECK_T	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA ≠ TS && (DA≠127 || (FC.Function ≠ SDN_H && FC.Function ≠ SDN_L)) || FC.Frame ≠ req || INVALID_FUNCTION) && !(DA = SA && FC.Frame = req && FC.Function = FDL_status && Isochronous_mode) ⇒ RESM := empty, T_cnt := 0	ACTIVE_I
171	CHECK_T	SRC_RECEIVE_ERROR.ind / In_ring_desired ⇒ T_cnt := 0	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
172	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /DUPLICATE_ADDRESS && Dup_add_count > 0 && In_ring_desired ⇒ INIT_FCBV_LIST, INIT_LMS, RESM := empty, Dup_add_count := 0, Fault_type := Duplicate_address, Second := 0, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	LISTEN
173	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /DUPLICATE_ADDRESS && Dup_add_count = 0 && In_ring_desired ⇒ Dup_add_count++, TOK_CNT_UPD, Second := 0, RESM := empty, T_cnt := 0, TRT_ON	ACTIVE_I
174	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt > (Limit – 2) && In_ring_desired ⇒ INIT_FCBV_LIST, INIT_LMS, RESM := empty, Fault_type := Out_of_ring, Second := 0, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	LISTEN
175	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && TOKEN_ERROR && Tok_err_cnt ≤ (Limit – 2) && In_ring_desired ⇒ RESM := empty, TOK_CNT_UPD, Tok_err_cnt++, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I
176	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && LMS[SA] ≠ DA && DA ≠ TS && In_ring_desired ⇒ RESM := empty, LMS_UPDATE(DA, SA), TOK_CNT_UPD, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I
177	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && LMS[SA] = DA && DA ≠ TS && In_ring_desired ⇒ RESM := empty, TOK_CNT_UPD, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I
178	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[SA] = DA && Tct=0 && H_list.Num_entry ≠ 0 && In_ring_desired ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, Second := 0, Req_h_cnt := H_list.Num_entry, SETUP_HREQ, T_cnt := 0, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
179	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second = 0 && In_ring_desired ⇒ RESM := empty, Second++, T_cnt := 0, TRT_ON	ACTIVE_I
180	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second > 0 && Tct=0 && H_list.Num_entry ≠ 0 && In_ring_desired ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, LMS_UPDATE(DA, SA), Second := 0, Req_h_cnt := H_list.Num_entry, SETUP_HREQ, T_cnt := 0, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	USE_T
181	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[DA] = NIL && ((LMS[SA] > SA && LMS[SA] > TS && SA < TS) || (LMS[SA] ≤ SA && ((SA < TS) || (TS < LMS[SA])))) && In_ring_desired ⇒ RESM := empty, TTH_INIT, LMS_UPDATE(DA, SA), TOK_CNT_UPD, Second := 0, GAP_INIT(0), Fault_Type := In_ring, T_cnt := 0, TRT_ON MAC_BFAULT.ind (Fault_type)	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
182	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[SA] = DA && Tct=0 && H_list.Num_entry = 0 && In_ring_desired ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, Second := 0, Req_h_cnt := H_list.Num_entry, T_cnt := 0, TRT_ON, HoldToken:=1	CHECK_A
183	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second > 0 && Tct=0 && H_list.Num_entry = 0 && In_ring_desired ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, LMS_UPDATE(DA, SA), Second := 0, Req_h_cnt := H_list.Num_entry, T_cnt := 0, TRT_ON	CHECK_A
184	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[SA] = DA && Tct > 0 && In_ring_desired ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, Second := 0, Req_h_cnt := H_list.Num_entry, DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0, TRT_OFF, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	WAIT_TCT
185	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && LMS[LMS[SA]] = DA && Second > 0&& Tct > 0 && In_ring_desired ⇒ RESM := empty, GAP_UPDATE, TTH_UPDATE, TOK_CNT_UPD, LMS_UPDATE(DA, SA), Second := 0, Req_h_cnt := H_list.Num_entry, DA,SA:=TS, FC.Function := FDL_status, FC.Frame:=req, DLSDU,SSAP,DSAP:=NIL,T_cnt := 0, TRT_OFF, HoldToken:=1 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	WAIT_TCT
186	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!DUPLICATE_ADDRESS && Tok_err_cnt <= (Limit – 2) && !TOKEN_ERROR && DA = TS && (!LMS[LMS[SA]]=DA && !(LMS[DA] = NIL && ((LMS[SA] > SA && LMS[SA] > TS && SA < TS) || (LMS[SA] ≤ SA && ((SA < TS) || (TS < LMS[SA])))) && In_ring_desired ⇒ RESM := empty, LMS_UPDATE(DA, SA), TOK_CNT_UPD, Second := 0, T_cnt := 0, TRT_ON	ACTIVE_I
187	CHECK_T	SRC_RECEIVE_TOKEN.ind (DA, SA) /!In_ring_desired ⇒ Fault_Type := Double_token, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	PASSIVE_I
188	CHECK_T	SRC_SEND_DATA.cnf ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
189	CHECK_T	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
190	CHECK_T	SRC_SLOT_EVENT.ind /Retry_cnt < 2 ⇒ DS := NS(TS), SA := TS, Retry_cnt++, HoldToken:=1 SRC_SEND_TOKEN.req (DA, SA)	PASS_T

No.	Current state	Event /condition ⇒action	Next state
191	CHECK_T	SRC_SLOT_EVENT.ind /Retry_cnt = 2 ⇒ EX_LMS (NS), Retry_cnt := 0, DS := NS(TS), SA := TS, HoldToken:=1 SRC_SEND_TOKEN.req (DA, SA)	PASS_T
192	CHECK_T	SRC_SYNI_EVENT.ind / In_ring_desired ⇒ Fault_type := Not_synchronized, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I
193	AW_STATUS	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
194	AW_STATUS	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(FC.Frame = req || (FC.Function ≠ SC && DA ≠ TS)) ⇒ Fault_type := Double_token, NEXT_GAP, T_cnt := 0, HoldToken:=0, if (RECV_ERR_cnt>0) RECV_ERR_cnt-- MAC_BFAULT.ind (Fault_type)	ACTIVE_I
195	AW_STATUS	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /FC.Frame = rsp && DA = TS && SA=GAP_address && FC.Function = OK && FC.Stn-Type = M_rdy ⇒ DA := SA, SA := TS, LMS_UPDATE(DA, SA), GAP_INIT(NIL), T_cnt := 0 , Retry_cnt:=0, if (RECV_ERR_cnt>0) RECV_ERR_cnt--, SRC_SEND_TOKEN.req (DA, SA)	PASS_T
196	AW_STATUS	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /FC.Frame = rsp && (DA = TS || FC.Function = SC) && (FC.Function ≠ OK || FC.Stn-Type ≠ M_rdy) ⇒ NEXT_GAP , T_cnt := 0 , if (RECV_ERR_cnt>0) RECV_ERR_cnt--	CHECK_A
197	AW_STATUS	SRC_RECEIVE_ERROR.ind /RECV_ERR_cnt ≥ RECV_ERR_limit ⇒ T_cnt := 0, HoldToken:=0	LISTEN
198	AW_STATUS	SRC_RECEIVE_ERROR.ind /RECV_ERR_cnt < RECV_ERR_limit ⇒ NEXT_GAP , T_cnt := 0	CHECK_A
199	AW_STATUS	SRC_RECEIVE_TOKEN.ind (DA, SA) /In_ring_desired ⇒ Fault_type := Double_token, NEXT_GAP, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I
200	AW_STATUS	SRC_SEND_DATA.cnf ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
201	AW_STATUS	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
202	AW_STATUS	SRC_SLOT_EVENT.ind ⇒ NEXT_GAP	CHECK_A
203	AW_STATUS	SRC_SYNI_EVENT.ind /In_ring_desired ⇒ Fault_type := Not_synchronized, NEXT_GAP, HoldToken:=0 MAC_BFAULT.ind (Fault_type)	ACTIVE_I

No.	Current state	Event /condition ⇒action	Next state
204	PASSIVE_I	/H_List.Num_entry ≠ 0 ⇒ SETUP_HCON_DS	PASSIVE_I
205	PASSIVE_I	/L_List.Num_entry ≠ 0 ⇒ SETUP_LCON_DS	PASSIVE_I
206	PASSIVE_I	MAC_RESET.req ⇒ RESET_HL_LIST MAC_RESET.cnf	OFFL
207	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = FDL_status ⇒ DLSDU,SSAP,DSAP:=NIL, FC.Frame := rsp, FC.Stn-Type := Slave, FC.Function := OK , RESM := empty, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
208	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = Ident ⇒ DA := SA, SA := TS, DSAP <:⇒ SSAP, RESM := empty, IDENT, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
209	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && FC.Function = LSAP_status ⇒ DA := SA, SA := TS, DSAP <:⇒ SSAP, RESM := empty, LSAP_STATUS, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
210	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =127) && FC.Frame = res && SAP_CHECK(DSAP) && B_BUF(DSAP) ⇒ RESM := empty, SETUP_SIND(0,NO), T_cnt := 0	PASSIVE_I
211	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =127) && FC.Frame = res && (!SAP_CHECK(DSAP) || !B_BUF(DSAP)) ⇒ RESM := empty, T_cnt := 0	PASSIVE_I
212	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function =TE || FC.Function = CV) && SAP_CHECK(CS) && I_BUF(CS) ⇒ RESM := empty, SETUP_IND(0,NO), T_cnt := 0, TRT_ON	PASSIVE_I
213	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = CV || FC.Function = TE) && (!SAP_CHECK(CS) || !I_BUF(CS)) ⇒ RESM := empty, T_cnt := 0, TRT_ON	PASSIVE_I
214	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = SDN_H || FC.Function = SDN_L) && SAP_CHECK(DSAP) && I_BUF(DSAP) ⇒ RESM := empty, SETUP_IND(0,NO), T_cnt := 0	PASSIVE_I
215	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA =TS || DA =127) && FC.Frame = req && (FC.Function = SDN_H || FC.Function = SDN_L) && (!SAP_CHECK(DSAP) || !I_BUF(DSAP)) ⇒ RESM := empty, T_cnt := 0	PASSIVE_I

No.	Current state	Event /condition ⇒action	Next state
216	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_L || FC.Function = SDA_H || FC.Function = SRD_L || FC.Function = SRD_H || FC.Function = SRD_BCT) && RETRY ⇒ DA := RESM.DA, SA := TS, FC := RESM.FC, DSAP := RESM.DSAP, SSAP := RESM.SSAP, DLSDU := RESM.DLSDU, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
217	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_H || FC.Function = SDA_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) ⇒ SETUP_IND(0,NO), DA := NIL, SA := NIL, FC.Function := OK, DLSDU,SSAP,DSAP:=NIL, SETUP_RESM, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
218	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SDA_L || FC.Function = SDA_H) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RR, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
219	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) && U_BUF(DSAP) ⇒ SETUP_REPLY, if(SAP_List[DSAP].Indication_Mode = ALL || Upd_status ≠ NO) SETUP_IND(Ref, Upd_status), DA := R_DA, SA := TS, FC.Function := Upd_status, FC.Frame := rsp, SETUP_STN_TYPE, DSAP ≤> SSAP, DLSDU := R_SDU, SETUP_RESM, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
220	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) && U_BUF(DSAP) ⇒ SETUP_REPLY,DA := R_DA, SA := TS, FC.Function := R_FUNCTION, FC.Frame := rsp, SETUP_STN_TYPE, DSAP <:⇒ SSAP, DLSDU := R_SDU, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
221	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && I_BUF(DSAP) && !U_BUF(DSAP) ⇒ SETUP_IND(0,NO), FC.Function := SC, SETUP_RESM, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
222	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L) && !RETRY && SAP_CHECK(DSAP) && !I_BUF(DSAP) && !U_BUF(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RR, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I

No.	Current state	Event /condition ⇒action	Next state
223	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /DA = TS && FC.Frame = req && (FC.Function = SRD_H || FC.Function = SRD_BCT || FC.Function = SRD_L || FC.Function = SDA_H || FC.Function = SDA_L) && !RETRY && !SAP_CHECK(DSAP) ⇒ RESM := empty, DA := SA, SA := TS, FC.Frame := rsp, SETUP_STN_TYPE, FC.Function := RS, DLSDU,SSAP,DSAP:=NIL, T_cnt := 0 SRC_SEND_DATA.req (DA, SA, FC, DSAP, SSAP, DLSDU)	PASSIVE_I
224	PASSIVE_I	SRC_RECEIVE_DATA.ind (DA, SA, FC, DSAP, SSAP, DLSDU) /(DA ≠ TS && (DA≠127 || (FC.Function ≠ SDN_H && FC.Function ≠ SDN_L)) || (FC.Frame ≠ req && (DA ≠ 127) || INVALID_FUNCTION) ⇒ RESM := empty, T_cnt := 0	PASSIVE_I
225	PASSIVE_I	SRC_RECEIVE_ERROR.ind ⇒ T_cnt := 0	PASSIVE_I
226	PASSIVE_I	SRC_RECEIVE_TOKEN.ind (DA, SA) ⇒ RESM := empty, T_cnt := 0	PASSIVE_I
227	PASSIVE_I	SRC_SEND_DATA.cnf ⇒	PASSIVE_I
228	PASSIVE_I	SRC_SEND_TOKEN.cnf(Status) ⇒ Fault_type := State_conflict MAC_LFAULT.ind (Fault_type)	OFFL
229	PASSIVE_I	SRC_SLOT_EVENT.ind /!TIME_OUT ⇒ T_cnt ++	PASSIVE_I
230	PASSIVE_I	SRC_SLOT_EVENT.ind /TIME_OUT ⇒ Fault_type := Time_out, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	PASSIVE_I
231	PASSIVE_I	SRC_SYNI_EVENT.ind ⇒ Fault_type := Not_synchronized, T_cnt := 0 MAC_BFAULT.ind (Fault_type)	PASSIVE_I

A.5.2.3 Functions

All functions of the MAC are summarized in Table A.16.

Table A.16 – MAC function table

Function name	Operations
IDENT	- Prepares Ident-Response send Ident data, if Ident buffer available send SC, if no Ident buffer
LSAP_STATUS	- Prepares Ident-Response send LSAP data, if LSAP-status buffer available send SC, if no LSAP-status buffer
INIT_FCBV_LIST	FCV[0..126]:= 0, FCB[0..126]:= 1

Function name	Operations
INIT_LMS	LMS[0..126]:= NIL Tok_err_cnt:=0, Tok_cnt:=255 First :=NIL, LMS_cnt:=0
BUILD_LMS(Ad)	LMS[0..126]:= NIL, LMS[Ad]:=Ad
LMS_UPDATE	if(DA < LMS[SA] ≤ SA || SA < DA < LMS[SA] || LMS[SA] ≤ SA < DA) LMS[DA] := LMS[SA], LMS[SA] := DA else if (DA = LMS[LMS[SA]]) LMS[LMS[SA]] := NIL, LMS[SA] :=DA else Tok_err_cnt++
EX_LMS (Ad)	LMS[TS]:= LMS[Ad], LMS[Ad]:=NIL
TOK_CNT_UPD	if (Tok_cnt--=0) Tok_cnt:=255, Tok_err_cnt:=0, Dup_add_cnt:=0
DUPLICATE_ADDRESS	SA = TS || SA > HSA || DA > HSA
TOKEN_ERROR	LMS[SA] := NIL
FCB_UPDATE	FCB[DA]:= not FCB[DA], FCV[DA]:=1
FCV_CLEAR	FCV[DA]:=0, FCB[DA]:=1
SETUP_STN_TYPE	if (In_ring_desired = FALSE) FC.Stn-Type:=Slave else if (LMS[TS] = NIL) FC.Stn-Type:=M_rdy else FC.Stn-Type:=M_in_ring
R_FUNCTION	if (Upd_sts=DL) FC.Function:=RDL else FC.Function:=RDH
SETUP_RESM	RESM.DA := DA RESM.DSAP := DSAP RESM.SSAP := SSAP RESM.FC := FC RESM.DLSDU := DLSDU
RETRY	RESM≠NIL && RESM.SA=SA && DA=TS && FCV[DA]=1 && FC.FCV=1 && FCB[DA]=FC.FCB
SETUP_HREQ	Req_h_cnt--, H_List.Num_entry-- REQM.DA,DA := H_List.First_Entry.DA REQM.DSAP,DSAP := H_List.First_Entry.DSAP REQM.SSAP,SSAP := H_List.First_Entry.SSAP FC := H_List.First_Entry.FC REQM.DLSDU,DLSDU := H_List.First_Entry.DLSDU Cnf := H_List.First_Entry.conf REQM.serv_class := high SA:=TS if (FC.Function≠ FDL_Status, Ident, LSAP_Status, SDN_H, SDN_L) (FC.FCB:=FCB[DA], FC.FCV:=FCV[DA]) else (FC.FCB:=0, FC.FCV:=0) FC.Frame:=req REQM.FC:=FC H_List.Remove(), Retry_cnt := Max_Retry_Limit

Function name	Operations
SETUP_LREQ	Gap_lp_cnt--, L_List.Num_entry-- REQM.DA,DA := L_List.First_Entry.DA REQM.DSAP,DSAP := L_List.First_Entry.DSAP REQM.SSAP,SSAP := L_List.First_Entry.SSAP FC := L_List.First_Entry.FC REQM.DLSDU,DLSDU := L_List.First_Entry.DLSDU Cnf := L_List.First_Entry.conf REQM.serv_class := low SA:=TS if (FC.Function≠ FDL_Status, Ident, LSAP_Status, SDN_H, SDN_L) (FC.FCB:=FCB[DA], FC.FCV:=FCV[DA]) else (FC.FCB:=0, FC.FCV:=0) FC.Frame:=req REQM.FC:=FC L_List.Remove(), Retry_cnt := Max_Retry_Limit
SETUP_CONM(Err_type)	C_List.Insert() C_List.Last_Entry.DA := REQM.DA C_List.Last_Entry.DSAP := REQM.DSAP C_List.Last_Entry.SSAP := REQM.SSAP C_List.Last_Entry.serv_class := REQM.serv_class C_List.Last_Entry.FC := Err_type C_List.Last_Entry.DLSDU := NIL C_List.Num_entry++
RESET_HL_LIST	while (H_List.Num_entry ≠ 0) SETUP_HCON_DS while (L_List.Num_entry ≠ 0) SETUP_LCON_DS
SETUP_HCON_DS	H_List.Num_entry-- C_List.Insert() C_List.Last_Entry.DA := H_List.First_Entry.DA C_List.Last_Entry.DSAP := H_List.First_Entry.DSAP C_List.Last_Entry.SSAP := H_List.First_Entry.SSAP C_List.Last_Entry.serv_class := H_List.First_Entry.serv_class C_List.Last_Entry.FC:= H_List.First_Entry.FC C_List.Last_Entry.Status:= DS C_List.Num_entry++ H_List.Remove()
SETUP_LCON_DS	L_List.Num_entry-- C_List.Insert() C_List.Last_Entry.DA := L_List.First_Entry.DA C_List.Last_Entry.DSAP := L_List.First_Entry.DSAP C_List.Last_Entry.SSAP := L_List.First_Entry.SSAP C_List.Last_Entry.serv_class := L_List.First_Entry.serv_class C_List.Last_Entry.FC:= L_List.First_Entry.FC C_List.Last_Entry.Status:= DS C_List.Num_entry++ L_List.Remove()

Function name	Operations
SETUP_CON(FC,DLSDU)	C_List.Insert() C_List.Last_Entry.DA := REQM.DA C_List.Last_Entry.DSAP := REQM.DSAP C_List.Last_Entry.SSAP := REQM.SSAP C_List.Last_Entry.serv_class := REQM.serv_class C_List.Last_Entry.FC:= REQM.FC C_List.Last_Entry.DLSDU := DLSDU C_List.Last_Entry.Status:= FC C_List.Num_entry++
SETUP_IND(Ref,Upd_sts)	SAP_List[DSAP].Ibuffer.Num_entry-- I_List.Insert() I_List.Last_Entry.DA := DA I_List.Last_Entry.SA := SA I_List.Last_Entry.DSAP := DSAP I_List.Last_Entry.SSAP := SSAP I_List.Last_Entry.FC := FC I_List.Last_Entry.DLSDU := DLSDU I_List.Last_Entry. Status := Upd_sts I_List.Last_Entry.Reference := Ref I_List.Num_entry++ SAP_List[DSAP].Ibuffer.Remove()
SETUP_SIND(Ref,Upd_sts)	SAP_List[DSAP].Sbuffer.Num_entry-- I_List.Insert() I_List.Last_Entry.DA := DA I_List.Last_Entry.SA := SA I_List.Last_Entry.DSAP := DSAP I_List.Last_Entry.SSAP := SSAP I_List.Last_Entry.FC := FC I_List.Last_Entry.DLSDU := DLSDU I_List.Last_Entry. Status := OK I_List.Num_entry++ SAP_List[DSAP].Sbuffer.Remove()
SAP_CHECK (DSAP)	(SA= SAP_List[DSAP].Access || SAP_List[DSAP].Access=NIL) && (FC.Function is in SAP_List[DSAP].Function_List_R) && DLSDU.Len ≤ SAP_List[DSAP].LenList[FC.Function]
I_BUF(DSAP)	SAP_List[DSAP]. Ibuffer.First_Entry ≠ NIL
B_BUF(DSAP)	SAP_List[DSAP]. Sbuffer.First_Entry ≠ NIL
U_BUF(DSAP)	SAP_List[DSAP]. Ubuffer.High_buffer ≠ NIL || SAP_List[DSAP]. Ubuffer.Low_buffer ≠ NIL

Function name	Operations

SETUP_REPLY	if (FC.Function = SRD_BCT) R_DA=127
	else R_DA=SA
	If (SAP_List[DSAP]. Ubuffer.High_buffer.Len ≠ 0)
	(Upd_sts := DH, Ref := SAP_List[DSAP]. Ubuffer.High_reference,
	R_SDU := SAP_List[DSAP]. Ubuffer.High_buffer,
	if(SAP_List[DSAP].Ubuffer.High_transmit = SINGLE) SAP_List[DSAP]. Ubuffer.High_buffer=NIL)
	else(Upd_sts := DL, Ref := SAP_List[DSAP]. Ubuffer.Low_reference,
	R_SDU := SAP_List[DSAP]. Ubuffer.Low_buffer,
	if(SAP_List[DSAP].Ubuffer.Low_transmit = SINGLE) SAP_List[DSAP]. Ubuffer.Low_buffer=NIL)

DECODE(func)	if (func = SDN_H) (Service := SDN, Serv_class := High)
	if (func = SDA_H) (Service := SDA, Serv_class := High)
	if (func = SRD_H) (Service := SRD, Serv_class := High)
	if (func = SDN_L) (Service := SDN, Serv_class := Low)
	if (func = SDA_L) (Service := SDA, Serv_class := Low)
	if (func = SRD_L) (Service := SRD, Serv_class := Low)

TIME_OUT	(In_ring_desired=true && T_cnt = 2*TS+6) || (In_ring_desired=false && T_cnt = 266)
INVALID_ FUNCTION	FC.Function = 0 || FC.Function = 1 || FC.Function = 2 || FC.Function = 7 || FC.Function = 8 || FC.Function = 10 || FC.Function = 11
GAP_INIT	Gap_to_do := true GAP_address := (TS + 1) mod (HSA + 1) if (GAP_address = LMS [TS]) GAP_to_do := false, Gud_timer := Tgud
NEXT_GAP	GAP_lp_cnt := NIL GAP_address := (GAP_address + 1) mod (HSA + 1) if (GAP_address = LMS [TS]) GAP_to_do := false, Gud_timer := Tgud

GAP_UPDATE	Gud_timer := Gud_timer – Ttr,
	if (GAP_to_do && GAP_lp_cnt = NIL) GAP_lp_cnt := L_list.Num_entry,
	if (!GAP_to_do && Gud_timer<0)
	(GAP_address := (TS + 1) mod (HSA + 1), Gap_to_do := TRUE
	if (GAP_address = LMS [TS]) GAP_to_do := false, Gud_timer := Tgud

TTH_INIT	Trr := Ttr, TRT.start(Ttr)
TTH_UPDATE	Trr :=Ttr-TRT.cv, TRT.start(Ttr)
TTH_AVAILABLE	Trr < TRT.cv
TRT_ON	if (TRT.stopped) (TRT.start(Trr), TRT.stopped := FALSE)
TRT_OFF	TRT.stop, Trr := TRT.cv, TRT.stopped:= TRUE

A.6 SRU

A.6.1 Overview

SRU contains the following parts:

a) Message oriented Main-SM (SRC)

b) Character Receive SM (CRX)

c) Timer-SM (TIM)

d) Character Send SM (CTX).

Machine internal communication is done without any time delay.

The CTX and CRX machines are used only in asynchrounous mode to transfer a bit oriented stream in a character oriented stream. This task can be done with standard components (called Universal Asynchrounous Receiver / Transmitter) and are not described further. The CRX (together with the physical receive units) shall allow a distortion of the input signal of ±10 %.

The timer module is also a standard component that just implements the required timers. The Wait-service selects the specific idle-timers. The respective timer is started with StartIdle and Start. The other timer services are used to indicate the expiration of timers.

The SRC is responsible for coding/encoding of DLPDUs. This module is reponsible for the transmission procedures as well (see Clause 21 for further description for send receive procedures).

Both the asynchronous and the synchronous interfaces are shown in Figure A.2.

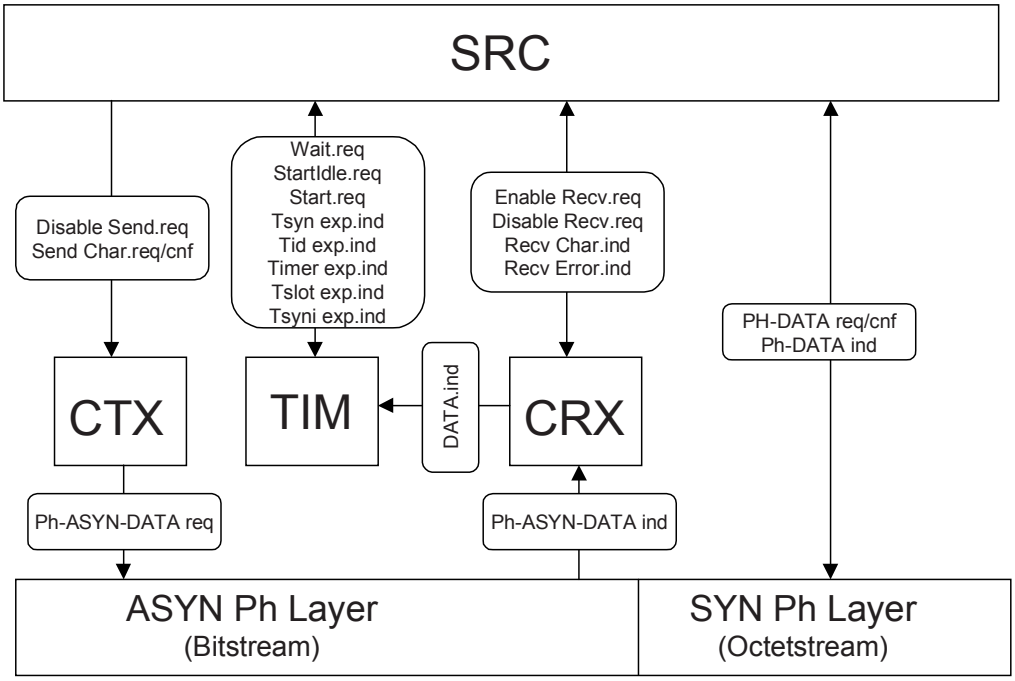


Figure A.2 – Structure of the SRU Machine

A.6.2 Character send SM(CTX)

This state machine implements an UART which has the ability to send without gaps between stop-bit and preceding start-bit. In conjunction with the underlying physical layer it produces a maximum of less than 0,3 % deviation of the nominal data rate. For data rates of 1 500 kbit/s and higher the maximum clock drift of ± 0,03 % should not be exceeded.This model describes a SRC which gets an indication after a complete character has been send (this model implies a zero delay between SRC and CTX). Real devices may implement a double buffering which has influence on the termination of a send sequence (a sender empty indication is required).

Interface to SRC:

– Send_Char.req/.cnf
– Disable_Send.req

A.6.3 Character receive SM (CRX)

This state machine implements an UART which has the ability to receive without gaps between stop-bit and preceding start-bit. It has to detect stop-bit and parity errors. Together with the underlying physical layer it has to detect at least signals with 10 % distortion correctly. If there is a signal change detectible by the underlying physical layer within 10 % and 90 % of the signal, the resulting bit-value is undefined.

When a change to start signal without an valid character is detected with enabled receiver the CRX has to report an error to the SRC. It has be at least as sensitive as the Timer-SM (a trigger for the timer must trigger the receiver as well).

Interface to SRC:

– Receive_Char.ind
– Enable_Recv.req
– Disable_Recv.req
– Receive_Error.ind

A.6.4 Timer-SM (TIM)

The timer module implements three different timers.

One Timer is used for one shot events and will be started by the SRC with a start.req primitive.

Two Timers start when the receiver is disabled or the bus changes from active (logical 0) to inactive (logical 1).

Timer for Tsyn/Tid

This timer is set to 0 when receiver is enabled and the signal is ZERO. If a synchronous Ph Layer is used the timer is loaded with StartIdle.req.

Two thresholds will produce indications:

– the threshold Tsyn is a fixed value;
– the threshold Tid will be loaded according to Tid1/Tid2 with the wait.req primitive.

Timer for Tsyni/Tslot

This timer is reloaded with Tslot -Tsyn when Tsyn-timer expires.

This timer is reloaded with Tslot when this timer expires with Tslot.

This timer is reloaded with Tsyn, when the receiver becomes active.

Interface to SRC:

– Tsyn_exp.ind
– Tid_exp.ind
– Tslot_exp.ind
– Tsyni_exp.ind

– Timer_exp.ind
– Wait.req
– Start.req
– StartIdle.req

CTX, CRX and TIM are not described as state machines.

A.6.5 Primitive definition of SRC

A.6.5.1 Primitives exchanged between DLM and SRC

This interface is not worked out further. The primitives issued by the DLM to the SRC are shown in Table A.17.

Table A.17 – Primitives issued by DLM to SRC

Primitive name	Associated parameters
SRC_RESET.req	(none)

Primitives Exchanged between SRC and DLM.

This interface is not described in detail. The primitives issued by the SRC to the DLM are shown in Table A.18.

Table A.18 – Primitives issued by SRC to DLM

Primitive name	Associated parameters
SRC_RESET.cnf	(none)
MAC_LFAULT.ind	Fault_type
MAC_BFAULT.ind	Fault_type

A.6.5.2 Primitives exchanged between MAC and SRC

The primitives issued by the MAC to the SRC are shown in Table A.19.

Table A.19 – Primitives issued by MAC to SRC

Primitive name	Associated parameters
SRC_SEND_TOKEN.req	DA, SA
SRC_SEND_DATA.req	DA, SA, FC, DSAP, SSAP, DLSDU

The primitives issued by the SRC to the MAC are shown in Table A.20.

Table A.20 – Primitives issued by SRC to MAC

Primitive name	Associated parameters
SRC_SEND_TOKEN.cnf	Status
SRC_SEND_DATA.cnf	(none)
SRC_RECEIVE_DATA.ind	DA, SA, FC, DSAP, SSAP, DLSDU
SRC_RECEIVE_TOKEN.ind	DA, SA
SRC_RECEIVE_ERROR.ind	(none)
SRC_SLOT_EVENT.ind	(none)
SRC_SYNI_EVENT.ind	(none)

A.6.5.3 Parameters of SRC Primitives

All parameters used with primitives exhcanged between the MAC and the SRC are shown in Table A.21.

Table A.21 – Parameters used with primitives exchanged between MAC and SRC

Parameter name	Description
DA	Station address of the receiving unit
SA	Station address of the sending unit
FC	The FC structure is described in Table A.22
DSAP	Identifier of a remote Service Access Point
SSAP	Identifier of the local Service Access Point
DLSDU	Data unit of a DLS-user
Status	Status of the service execution

Table A.22 – FC structure

FC.frame	FC.function	FCB,FCV	FC.stn-type
req	SDN_H,SDN_L, SDA_H, SDA_L, SRD_H, SRD_L, TE, CV, FDL_Status, LSAP_Status, Ident	0,0: all services 0,1: SYN for SRD,SDA 1,0: normal Op. SDA,SRD 1,1: normal Op. SDA,SRD	
rsp	OK, RS, RR, DL, DH, NR, RDL, RDH, SC		Slave, M_n_rdy, M_rdy, M_in_ring

A.6.6 State machine description

A.6.6.1 General

The Send Receive Control forms messages out of a stream of data offered by the Physical Layer and vice versa. The SRC supports both Synchronous and Asynchronous Physical layers.

The local variables of the SRC are shown in Table A.23.

Table A.23 – Local variables of SRC

Name	Type	Range	Remark
SDNM	Bool	—	SDN Marker = SDN (was) executed
SType	Enum	(I/R)	Send type (Initiator/Response)
DAE	U8	0,0x80	Address Extension
SAE	U8	0,0x80	Address Extension
F	U8	—	FC-Field
Fcs	U8	—	Checksum
Char	U8	—	—
I	U8	—	—
j	U8	—	—
TOK	Bool	—	Marker Token was sent
RxSD	Bool	—	Marker SD was received
SDdata	U8	—	Storage of SD
C1,C2	U8	—	FCS components

A.6.6.2 SRC state table

The state table of the SRC is shown in Table A.24.

Table A.24 – SRC state table

No.	Current state	Event /condition ⇒action	Next state
1	any-state	SRC_RESET.req /Data_rate = NIL ⇒ SRC_RESET.cnf	OFFL
2	any-state	SRC_RESET.req /!PHSYN && Data_rate ≠ NIL ⇒ SRC_RESET.cnf	Idle
3	any-state	SRC_RESET.req /PHSYN && Data_rate ≠ NIL ⇒ SRC_RESET.cnf, StartIdle.req	AW_SD
4	Idle	Tsyn_exp.ind ⇒ Enable_Recv.req	AW_SD
5	Idle	Tid_exp.ind ⇒	Idle
6	Idle	Tsyni_exp.ind ⇒ SRC_SYNI_EVENT.ind	Idle
7	Idle	SRC_SEND_TOKEN.req (DA,SA) ⇒ SDNM := FALSE Wait.req (Tid1)	SND_SD4
8	Idle	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /DLSDU.Len = 0 && DSAP = NIL && SSAP = NIL && FC.Frame = req && SDNM = FALSE ⇒ SET_SDNM(FC.Function) Wait.req (Tid1)	ISND_SD1
9	Idle	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /(DLSDU.Len ≠ 0 || DSAP ≠ NIL || SSAP ≠ NIL) && FC.Frame = req && SDNM = FALSE ⇒ SET_DAE, SET_SAE, SET_SDNM(FC.Function), j:=(SAE+DAE)/0x80+3 Wait.req (Tid1)	ISND_SD2
10	Idle	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /DLSDU.Len = 0 && DSAP = NIL && SSAP = NIL && FC.Frame = req && SDNM = TRUE ⇒ SET_SDNM(FC.Function) Wait.req (Tid2)	ISND_SD1
11	Idle	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /(DLSDU.Len ≠ 0 || DSAP ≠ NIL || SSAP ≠ NIL) && FC.Frame = req && SDNM = TRUE ⇒ SET_DAE, SET_SAE, SET_SDNM(FC.Function), j:=(SAE+DAE)/0x80+3 Wait.req (Tid2)	ISND_SD2
12	Idle	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /DLSDU.Len = 0 && DSAP = NIL && SSAP = NIL && FC.Frame = rsp && FC.Function ≠ SC ⇒ SDNM := FALSE	RSND_SD1

No.	Current state	Event /condition ⇒action	Next state
13	Idle	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /(DLSDU.Len ≠ 0 || DSAP ≠ NIL || SSAP ≠ NIL) && FC.Frame = rsp ⇒ SET_DAE, SET_SAE, SDNM := FALSE , j:=(SAE+DAE)/0x80+3	RSND_SD2
14	Idle	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /FC.Frame = rsp && FC.Function = SC ⇒ SDNM := FALSE	SND_SC
15	AW_SD	SRC_SEND_TOKEN.req (DA,SA) ⇒ SDNM := FALSE Wait.req (Tid1)	SND_SD4
16	AW_SD	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /DLSDU.Len = 0 && DSAP = NIL && SSAP = NIL && FC.Frame = req && SDNM = FALSE ⇒ SET_SDNM(FC.Function) Disable_Recv.req Wait.req (Tid1)	ISND_SD1
17	AW_SD	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /(DLSDU.Len ≠ 0 || DSAP ≠ NIL || SSAP ≠ NIL) && FC.Frame = req && SDNM = FALSE ⇒ SET_DAE, SET_SAE, SET_SDNM(FC.Function), j:=(SAE+DAE)/0x80+3 Disable_Recv.req Wait.req (Tid1)	ISND_SD2
18	AW_SD	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /DLSDU.Len = 0 && DSAP = NIL && SSAP = NIL && FC.Frame = req && SDNM = TRUE ⇒ SET_SDNM(FC.Function) Disable_Recv.req Wait.req (Tid2)	ISND_SD1
19	AW_SD	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /(DLSDU.Len ≠ 0 || DSAP ≠ NIL || SSAP ≠ NIL) && FC.Frame = req && SDNM = TRUE ⇒ SET_DAE, SET_SAE, SET_SDNM(FC.Function), j:=(SAE+DAE)/0x80+3 Disable_Recv.req Wait.req (Tid2)	ISND_SD2
20	AW_SD	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /DLSDU.Len = 0 && DSAP = NIL && SSAP = NIL && FC.Frame = rsp && FC.Function ≠ SC ⇒ SDNM := FALSE Disable_Recv.req	RSND_SD1
21	AW_SD	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /(DLSDU.Len ≠ 0 || DSAP ≠ NIL || SSAP ≠ NIL) && FC.Frame = rsp ⇒ SET_DAE, SET_SAE, SDNM := FALSE , j:=(SAE+DAE)/0x80+3 Disable_Recv.req	RSND_SD2
22	AW_SD	SRC_SEND_DATA.req (DA,SA,FC,DSAP,SSAP,DLSDU) /FC.Frame = rsp && FC.Function = SC ⇒ SDNM := FALSE Disable_Recv.req	SND_SC
23	AW_SD	RX_DATA(data) /data = SD1 ⇒ SD_count++, DLSDU.Len := 0, CHECKINI(data), i:=0	AW_DA

No.	Current state	Event /condition ⇒action	Next state
24	AW_SD	RX_DATA(data) /data = SD2 ⇒ SD_count++, CHECKINI(data)	AW_LE
25	AW_SD	RX_DATA(data) /data = SD3 ⇒ SD_count++, DLSDU.Len := 8, CHECKINI(data)	AW_DA
26	AW_SD	RX_DATA(data) /!PHSYN && data = SC ⇒ SD_count++, FC.Function := SC, FC.Frame := rsp, i:=0 SRC_RECEIVE_DATA.ind (DA,SA,FC,DSAP,SSAP,DLSDU), Disable_Recv.req	Idle
27	AW_SD	RX_DATA(data) /PHSYN && data = SC ⇒ SD_count++, FC.Function := SC, FC.Frame := rsp, i:=0, TOK := FALSE	AW_CRC1
28	AW_SD	RX_DATA(data) /data = SD4 ⇒ SD_count++	AW_DAT
29	AW_SD	RX_DATA(data) /data ≠ SC, SD1, SD2, SD3, SD4 ⇒ SD_error_count++ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
30	AW_SD	Recv_Error.ind ⇒ SD_error_count++ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
31	AW_SD	Tslot_exp.ind ⇒ SRC_SLOT_EVENT.ind	AW_SD
32	AW_SD	Tsyn_exp.ind /!PHSYN ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req, Enable_Recv.req	AW_SD
33	AW_SD	Tsyn_exp.ind /PHSYN ⇒	AW_SD
34	AW_SD	Ph-DATA.ind(SOD,data) ⇒	AW_SD
35	AW_LE	RX_DATA(data) /DLSDU.Len > 3 && DLSDU.Len < 250 ⇒ DLSDU.Len := (data – 3), i:=0	AW_LER
36	AW_LE	RX_DATA(data) /DLSDU.Len < 4 || DLSDU.Len > 249 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
37	AW_LER	RX_DATA(data) /data = DLSDU.Len + 3 ⇒	AW_SDR
38	AW_LER	RX_DATA(data) /data ≠ (DLSDU.Len + 3) ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle

No.	Current state	Event /condition ⇒action	Next state
39	AW_SDR	RX_DATA(data) /data = SD2 ⇒	AW_DA
40	AW_SDR	RX_DATA(data) /data ≠ SD2 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
41	AW_DA	RX_DATA(data) ⇒ DA := data AND 0x7f, DAE := data AND 0x80, CHECKS(data), DSAP := NIL	AW_SA
42	AW_SA	RX_DATA(data) ⇒ SA := data AND 0x7f, SAE := data AND 0x80, CHECKS(data), SSAP := NIL	AW_FC
43	AW_FC	RX_DATA(data) /DLSDU.Len > 0 && DAE ≠ 0 ⇒ CHECKS(data), SETUP_FC	AW_DSAP
44	AW_FC	RX_DATA(data) /DLSDU.Len > 0 && DAE = 0 && SAE ≠ 0 ⇒ CHECKS(data), SETUP_FC	AW_SSAP
45	AW_FC	RX_DATA(data) /DLSDU.Len > 0 && DAE = 0 && SAE = 0 ⇒ CHECKS(data), SETUP_FC	AW_DATA
46	AW_FC	RX_DATA(data) /!PHSYN && DLSDU.Len = 0 && DAE = 0 && SAE = 0 ⇒ CHECKS(data), SETUP_FC	AW_FCS
47	AW_FC	RX_DATA(data) /PHSYN && DLSDU.Len = 0 && DAE = 0 && SAE = 0 ⇒ CHECKS(data), SETUP_FC	AW_CRC1
48	AW_FC	RX_DATA(data) /DLSDU.Len = 0 && (DAE ≠ 0 || SAE ≠ 0) ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
49	AW_DSAP	RX_DATA(data) /data && 0xc0 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
50	AW_DSAP	RX_DATA(data) /DLSDU.Len > 1 && SAE ≠ 0 ⇒ DSAP:=data, DLSDU.Len := DLSDU.Len -1, CHECKS(data)	AW_SSAP
51	AW_DSAP	RX_DATA(data) /DLSDU.Len > 1 && SAE = 0 ⇒ DSAP:=data, DLSDU.Len := DLSDU.Len -1, CHECKS(data)	AW_DATA
52	AW_DSAP	RX_DATA(data) /!PHSYN && DLSDU.Len = 1 && SAE = 0 ⇒ DSAP:=data, DLSDU.Len := DLSDU.Len -1, CHECKS(data)	AW_FCS
53	AW_DSAP	RX_DATA(data) /PHSYN && DLSDU.Len = 1 && SAE = 0 ⇒ DSAP:=data, DLSDU.Len := DLSDU.Len -1, CHECKS(data)	AW_CRC1

No.	Current state	Event /condition ⇒action	Next state
54	AW_DSAP	RX_DATA(data) /DLSDU.Len = 1 && SAE ≠ 0 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
55	AW_SSAP	RX_DATA(data) /data && 0xc0 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
56	AW_SSAP	RX_DATA(data) /DLSDU.Len > 1 ⇒ SSAP:=data, DLSDU.Len := DLSDU.Len -1, CHECKS(data)	AW_DATA
57	AW_SSAP	RX_DATA(data) /!PHSYN && DLSDU.Len = 1 ⇒ SSAP:=data, DLSDU.Len := DLSDU.Len -1, CHECKS(data)	AW_FCS
58	AW_SSAP	RX_DATA(data) /PHSYN && DLSDU.Len = 1 ⇒ SSAP:=data, DLSDU.Len := DLSDU.Len -1, CHECKS(data)	AW_CRC1
59	AW_DATA	RX_DATA(data) /DLSDU.Len > i + 1 ⇒ DLSDU.Data[i] := data, i := i + 1, CHECKS(data)	AW_DATA
60	AW_DATA	RX_DATA(data) /!PHSYN && DLSDU.Len = i + 1 ⇒ DLSDU.Data[i] := data, CHECKS(data)	AW_FCS
61	AW_DATA	RX_DATA(data) /PHSYN && DLSDU.Len = i + 1 ⇒ DLSDU.Data[i] := data, CHECKS(data), TOK := FALSE	AW_CRC1
62	AW_FCS	RX_DATA(data) /data = Fcs mod 256 ⇒	AW_ED
63	AW_FCS	RX_DATA(data) /data ≠ (Fcs mod 256) ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
64	AW_ED	RX_DATA(data) /data = ED && DA ≠ 0x7f ⇒ SRC_RECEIVE_DATA.ind (DA,SA,FC,DSAP,SSAP,DLSDU), Start.req (Tsdr), Disable_Recv.req	Idle
65	AW_ED	RX_DATA(data) /data = ED && DA = 0x7f ⇒ SDNM := TRUE SRC_RECEIVE_DATA.ind (DA,SA,FC,DSAP,SSAP,DLSDU), Start.req (Tsdr), Disable_Recv.req	Idle
66	AW_ED	RX_DATA(data) /data ≠ ED ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
67	AW_CRC1	RX_DATA(data) /data = C1 ⇒	AW_CRC2

No.	Current state	Event /condition ⇒action	Next state
68	AW_CRC1	RX_DATA(data) /data ≠ C1 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	AW_EODAF
69	AW_CRC2	RX_DATA(data) /data = C2 ⇒	AW_EODA
70	AW_CRC2	RX_DATA(data) /data ≠ C2 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	AW_EODAF
71	AW_EODA	RX_DATA(data) ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	AW_EODAF
72	AW_EODA	Ph-DATA.ind(EODA,data) /!TOK && DA ≠ 0x7f ⇒ SRC_RECEIVE_DATA.ind (DA,SA,FC,DSAP,SSAP,DLSDU), Start.req (Tsdr), StartIdle.req	AW_SD
73	AW_EODA	Ph-DATA.ind(EODA,data) /TOK && DA ≠ 0x7f ⇒ SRC_RECEIVE_TOKEN.ind (DA, SA), StartIdle.req	AW_SD
74	AW_EODA	Ph-DATA.ind(EODA,data) /DA = 0x7f ⇒ SDNM := TRUE SRC_RECEIVE_DATA.ind (DA,SA,FC,DSAP,SSAP,DLSDU),StartIdle.req	AW_SD
75	AW_EODAF	Ph-DATA.ind(DATA,data) ⇒	AW_EODAF
76	AW_EODAF	Ph-DATA.ind(EODA,data) ⇒ StartIdle.req	AW_SD
77	AW_DAT	RX_DATA(data) /(data && 0x80) = 0 ⇒	AW_SAT
78	AW_DAT	RX_DATA(data) /(data && 0x80) ≠ 0 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
79	AW_SAT	RX_DATA(data) /!PHSYN && (data && 0x80) = 0 ⇒ SRC_RECEIVE_TOKEN.ind (DA, SA), Disable_Recv.req	Idle
80	AW_SAT	RX_DATA(data) /PHSYN && (data && 0x80) = 0 ⇒ TOK := TRUE	AW_CRC1
81	AW_SAT	RX_DATA(data) /(data && 0x80) ≠ 0 ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
82	SND_SD4	Tsyni_exp.ind ⇒ SRC_SYNI_EVENT.ind Disable_Recv.req	Idle
83	SND_SD4	Tsyn_exp.ind ⇒	SND_SD4

No.	Current state	Event /condition ⇒action	Next state
84	SND_SD4	Timer_exp.ind ⇒	SND_SD4
85	SND_SD4	Tid_exp.ind /!PHSYN ⇒ data:=SD4 Enable_Recv.req, TX_DATA(data)	SND_DAT
86	SND_SD4	Tid_exp.ind /PHSYN ⇒ CHECKINI(data) Ph-DATA.req(SOA,data)	SND_SOAT
87	SND_SOAT	Ph-DATA.cnf ⇒ data := SD4 Ph-DATA.req(DATA,data)	SND_DAT
88	SND_DAT	TX_DATA_CNF ⇒ CHECKS(DA), data:=DA TX_DATA(data)	SND_SAT
89	SND_SAT	TX_DATA_CNF /!PHSYN ⇒ CHECKS(DA), data:=SA TX_DATA(data)	SND_TOK
90	SND_SAT	TX_DATA_CNF /PHSYN ⇒ CHECKS(DA), data:=SA TX_DATA(data)	ST_SY_C1
91	SND_TOK	TX_DATA_CNF /!RxSD ⇒ Status := no_token_pass SRC_SEND_TOKEN.cnf(Status) Disable_Recv.req, Disable_Send.req Start.req(Tqui)	AW_QUI
92	SND_TOK	TX_DATA_CNF /RxSD && SDdata≠SD4 ⇒ Status := token_pass_failed SRC_SEND_TOKEN.cnf(Status) Disable_Recv.req, Disable_Send.req Start.req(Tqui)	AW_QUI
93	SND_TOK	TX_DATA_CNF /RxSD && SDdata=SD4 ⇒ Status := token_pass_ok SRC_SEND_TOKEN.cnf(Status) Disable_Recv.req, Disable_Send.req Start.req(Tqui)	AW_QUI
94	ST_SY_C1	Ph-DATA.cnf ⇒ data := Crc1 Ph-DATA.req(DATA,data)	ST_SY_C2
95	ST_SY_C2	Ph-DATA.cnf ⇒ data := Crc2 Ph-DATA.req(DATA,data)	ST_SY_EO
96	ST_SY_EO	Ph-DATA.cnf ⇒ Ph-DATA.req(EODA,data)	ST_SY_END

No.	Current state	Event /condition ⇒action	Next state
97	ST_SY_END	Ph-DATA.cnf /PHSYN && !RxSD ⇒ Status := no_token_pass SRC_SEND_TOKEN.cnf(Status), Start.req(Tqui)	AW_QUI
98	ST_SY_END	Ph-DATA.cnf /PHSYN && RxSD && SDdata≠SD4 ⇒ Status := token_pass_failed SRC_SEND_TOKEN.cnf(Status), Start.req(Tqui)	AW_QUI
99	ST_SY_END	Ph-DATA.cnf /PHSYN && RxSD && SDdata=SD4 ⇒ Status := token_pass_ok SRC_SEND_TOKEN.cnf(Status), Start.req(Tqui)	AW_QUI
100	ISND_SD1	Tsyni_exp.ind ⇒ SRC_SYNI_EVENT.ind Disable_Recv.req	Idle
101	ISND_SD1	Tsyn_exp.ind ⇒	ISND_SD1
102	ISND_SD1	Timer_exp.ind ⇒	ISND_SD1
103	ISND_SD1	Tid_exp.ind /SA ≠ DA || FC.Function ≠ 9 ⇒ SType:=I data:=SD1 CHECKINI(data) TX_DATA(data)	SND_DAD
104	ISND_SD1	Tid_exp.ind /!PHSYN && SA = DA && FC.Function = 9 ⇒ SType:=R data:=SD1 CHECKINI(data) TX_DATA(data)	SND_DAD
105	ISND_SD1	Tid_exp.ind /SA ≠ DA || FC.Function ≠ 9 ⇒ SType:=I CHECKINI(data) Ph-DATA.req(SOA,data)	SND_SOAD
106	ISND_SD1	Tid_exp.ind /SA = DA && FC.Function = 9 ⇒ SType:=R CHECKINI(data) Ph-DATA.req(SOA,data)	SND_SOAD
107	ISND_SD2	Tsyni_exp.ind ⇒ SRC_SYNI_EVENT.ind Disable_Recv.req	Idle
108	ISND_SD2	Tsyn_exp.ind ⇒	ISND_SD2
109	ISND_SD2	Timer_exp.ind ⇒	ISND_SD2

No.	Current state	Event /condition ⇒action	Next state
110	ISND_SD2	Tid_exp.ind /!PHSYN ⇒ SType:=I data:=SD2 CHECKINI(data) TX_DATA(data)	SND_LE
111	ISND_SD2	Tid_exp.ind /PHSYN ⇒ SType:=I CHECKINI(data) Ph-DATA.req(SOA,data)	SND_SOAL
112	RSND_SD1	Tsyn_exp.ind ⇒	RSND_SD1
113	RSND_SD1	Timer_exp.ind /!PHSYN ⇒ SType:=R data:=SD1 CHECKINI(data) TX_DATA(data)	SND_DAD
114	RSND_SD1	Timer_exp.ind /PHSYN ⇒ SType:=R CHECKINI(data) Ph-DATA.req(SOA,data)	SND_SOAD
115	RSND_SD2	Tsyn_exp.ind ⇒	RSND_SD2
116	RSND_SD2	Timer_exp.ind /!PHSYN ⇒ SType:=R data:=SD2 CHECKINI(data) TX_DATA(data)	SND_LE
117	RSND_SD2	Timer_exp.ind /PHSYN ⇒ SType:=R CHECKINI(data) Ph-DATA.req(SOA,data)	SND_SOAL
118	SND_SC	Timer_exp.ind /!PHSYN ⇒ SType:=R, data:=SC, CHECKINI(data) TX_DATA(data)	SND_ENDD
119	SND_SC	Timer_exp.ind /PHSYN ⇒ SType:=R, CHECKINI(data) Ph-DATA.req(SOA,data)	SND_SOAS
120	SND_SOAL	Ph-DATA.cnf ⇒ data := SD2 Ph-DATA.req(DATA,data)	SND_LE

No.	Current state	Event /condition ⇒action	Next state
121	SND_LE	TX_DATA_CNF ⇒ data := DLSDU.Len + j, CHECKSYN(data) TX_DATA(data)	SND_LER
122	SND_LER	TX_DATA_CNF ⇒ CHECKSYN(data) TX_DATA(data)	SND_SD2R
123	SND_SD2R	TX_DATA_CNF ⇒ data := SD2, CHECKSYN(SD2) TX_DATA(data)	SND_DAD
124	SND_SOAD	Ph-DATA.cnf ⇒ data := SD1 Ph-DATA.req(DATA,data)	SND_DAD
125	SND_DAD	TX_DATA_CNF ⇒ data := DA + DAE, CHECKS(data) TX_DATA(data)	SND_SAD
126	SND_SAD	TX_DATA_CNF ⇒ data := SA + SAE, CHECKS(data) TX_DATA(data)	SND_FC
127	SND_FC	TX_DATA_CNF /DAE ≠ 0 ⇒ SET_F, data:= F, CHECKS(data) TX_DATA(data)	SND_DSAP
128	SND_FC	TX_DATA_CNF /DAE = 0 && SAE ≠ 0 ⇒ SET_F, data:= F, CHECKS(data) TX_DATA(data)	SND_SSAP
129	SND_FC	TX_DATA_CNF /DLSDU.Len > 0 && DAE = 0 && SAE = 0 ⇒ SET_F, data:= F, CHECKS(data) TX_DATA(data)	SND_DATA
130	SND_FC	TX_DATA_CNF /DLSDU.Len = 0 && DAE = 0 && SAE = 0 ⇒ SET_F, data:= F, CHECKS(data) TX_DATA(data)	SND_FCS
131	SND_DSAP	TX_DATA_CNF /SAE ≠ 0 ⇒ data := DSAP, CHECKS(data) TX_DATA(data)	SND_SSAP
132	SND_DSAP	TX_DATA_CNF /DLSDU.Len>0 && SAE = 0 ⇒ data := DSAP, CHECKS(data) TX_DATA(data)	SND_DATA
133	SND_DSAP	TX_DATA_CNF /DLSDU.Len = 0 && SAE = 0 ⇒ data := DSAP, CHECKS(data) TX_DATA(data)	SND_FCS

No.	Current state	Event /condition ⇒action	Next state
134	SND_SSAP	TX_DATA_CNF /DLSDU.Len > 0 ⇒ data := SSAP, CHECKS(data) TX_DATA(data)	SND_DATA
135	SND_SSAP	TX_DATA_CNF /DLSDU.Len = 0 ⇒ data := SSAP, CHECKS(data) TX_DATA(data)	SND_FCS
136	SND_DATA	TX_DATA_CNF /DLSDU.Len > i ⇒ data:=DLSDU.Data[i], i := i + 1, CHECKS(data) TX_DATA(data)	SND_DATA
137	SND_DATA	TX_DATA_CNF /!PHSYN && DLSDU.Len = i ⇒ data:=DLSDU.Data[i], CHECKS(data) TX_DATA(data)	SND_FCS
138	SND_DATA	TX_DATA_CNF /PHSYN && DLSDU.Len = i ⇒ data:=DLSDU.Data[i], CHECKS(data) Ph-DATA.req(DATA,data)	SND_SY_C1
139	SND_FCS	TX_DATA_CNF ⇒ data := (Fcs mod 256) TX_DATA(data)	SND_ED
140	SND_ED	TX_DATA_CNF ⇒ data := ED TX_DATA(data)	SND_ENDD
141	SND_ENDD	TX_DATA_CNF ⇒ SRC_SEND_DATA.cnf, Disable_Send.req, Start.req(Tqui)	AW_QUI
142	SND_SOAS	Ph-DATA.cnf ⇒ data := SC Ph-DATA.req(DATA,data)	SD_SY_C1
143	SD_SY_C1	Ph-DATA.cnf ⇒ data := Crc1 Ph-DATA.req(DATA,data)	SD_SY_C2
144	SD_SY_C2	Ph-DATA.cnf ⇒ data := Crc2 Ph-DATA.req(DATA,data)	SD_SY_EO
145	SD_SY_EO	Ph-DATA.cnf ⇒ Ph-DATA.req(EOA,data)	SD_SY_END
146	SD_SY_END	Ph-DATA.cnf ⇒ SRC_SEND_DATA.cnf, Start.req(Tqui)	AW_QUI
147	AW_QUI	Timer_exp.ind /PHSYN ⇒ StartIdle.req	AW_SD

No.	Current state	Event /condition ⇒action	Next state
148	AW_QUI	Timer_exp.ind /!PHSYN && SType=I && !SDNM ⇒ Enable_Recv.req	AW_SD
149	AW_QUI	Timer_exp.ind /!PHSYN && SType=R || SDNM ⇒	idle
150	S-State	RX_DATA /!RxSD ⇒ SDdata := data, RxSD :=TRUE	S-State
151	S-State	RX_DATA /RxSD ⇒	S-State
152	S-State	Ph-DATA.ind(EOA,Data) ⇒	S-State
153	S-State	Ph-DATA.ind(EOAD,Data) ⇒	S-State
154	AW-State	Recv_Error.ind ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req	Idle
155	AW-State	Tsyn_exp.ind ⇒ SRC_RECEIVE_ERROR.ind, Disable_Recv.req, Enable_Recv.req	AW_SD
156	AW-State	Ph-DATA.ind(EOA,Data) ⇒ SRC_RECEIVE_ERROR.ind, StartIdle.req	AW_SD

A.6.6.3 Functions

All function of the SRC are shown in Table A.25.

Table A.25 – SRC functions

Function	Description
PHSYN	Data_rate = 31,25 kbit/s
CHECKINI(data)	if (PHSYN) then (Init CRC(C1,C2)) else (Fcs := 0)
CHECKS(data)	if (PHSYN) then (Update CRC(C1,C2)) else (Fcs := 0)
CHECKSYN(data)	Update CRC(C1,C2)
RX_DATA(data)	if (PHSYN) then (Ph-DATA.ind(DATA,data)) else (Recv_Char.ind(data))
TX_DATA(data)	if (PHSYN) then (Ph-DATA.req(DATA,data)) else (Send_Char.req(data))
TX_DATA_CNF	if (PHSYN) then (Ph-DATA.cnf) else (Send_Char.cnf)
SET_SDNM	(if (FC.Function = SDN_L, SDN_H,TE,CV) SDNM = TRUE else SDNM = FALSE)
SETUP_FC	(FC.Function = Char AND 0x0f, if (Char AND 0x40) (FC.FCB = Char AND 0x20, FC.FCV = Char AND 0x10, FC.Frame_type =req) else (FC.Stn-Type= Char AND 0x30, FC.Frame_type =rsp))
SET_F	(F = FC.Function, if (FC.Frame_type =req) (0x80 + FC.FCB * 0x20 + FC.FCV * 0x10) else (F = F + FC.Stn-Type* 0x30))

Annex B
(informative)

Type 3 (synchronous): exemplary FCS implementations

This annex provides an example implementation of FCS generation and FCS syndrome checking for Type 3 when used with a synchronous PhL.

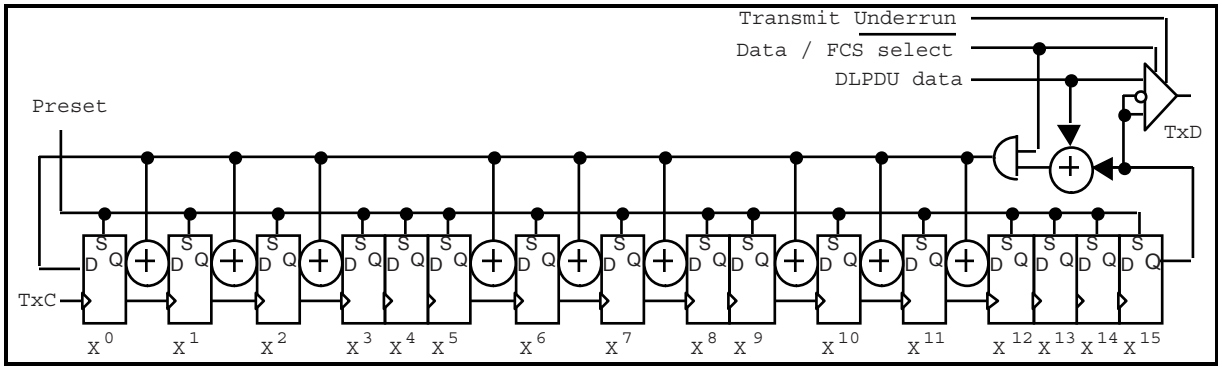


Figure B.1 – Example of FCS generation for Type 3 (synchronous)

In this example, shown in Figure B.1, the FCS is computed in a register consisting of 16 presettable master-slave flip-flops which are interconnected as a linear feedback shift register, with its least significant bit depicted on the left. The initial preset of the register before transmission serves to include the initial L(X) term in the FCS computation. Feedback is disabled during transmission of the FCS itself, and the FCS is transmitted complemented to provide the final L(X) inclusion in the FCS computation. Also shown is optional logic to inhibit the final complementation and transmit a massively incorrect FCS in the case of a transmitter underrun.

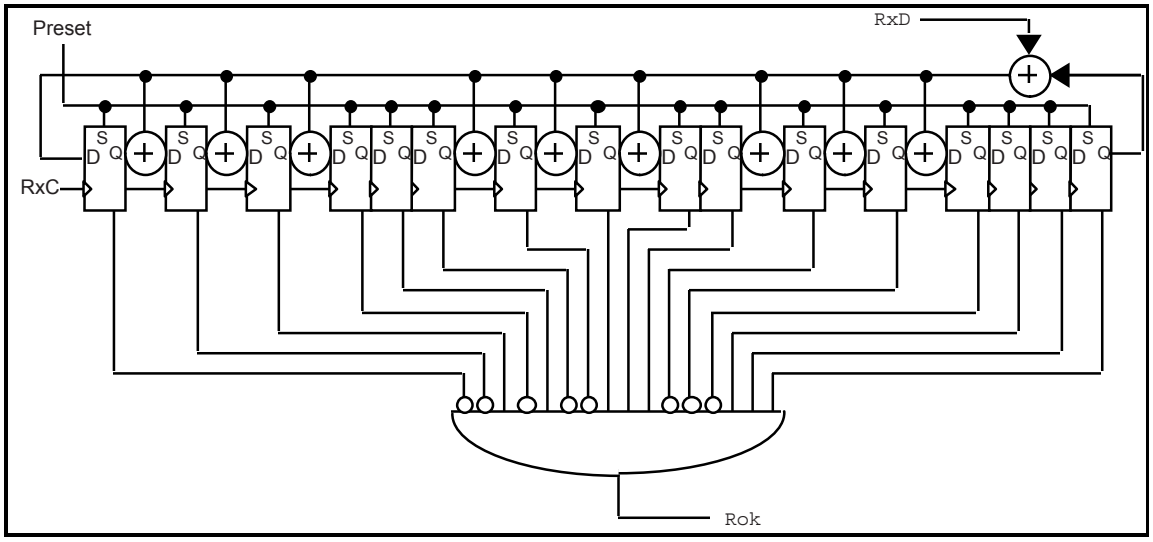


Figure B.2 – Example of FCS syndrome checking on reception for Type 3 (synchronous)

In this example, shown in Figure B.2, the residual FCS is computed in a similar register. The Q outputs of the 16 flip-flops are compared to the expected residual value by the 16-input “and” gate, half of whose inputs are complemented.

In this example, the FCS is computed in a register consisting of 16 presettable master-slave flip-flops which are interconnected as a linear feedback shift register, with its least significant bit depicted on the left. The initial pre-set of the register before transmission serves to include the initial L(X) term in the FCS computation. Feedback is disabled during transmission of the FCS it self, and the FCS is transmitted complemented to provide the final L(X) inclusion in the FCS computation. Also shown is optional logic to inhibit the final complementation and transmit a massively incorrect FCS in the case of a transmitter underrun.

Annex C
(informative)

Type 3: Exemplary token procedure and message transfer periods

C.1 Procedure of token passing

For explanatory purposes, the description of the token rotation time in 5.3.2.6 and message priorities in 5.3.2.7 are amended. The timers are presented as functions of the time t.

The token rotation time is measured by using the token-rotation-timer. At the beginning (t_{i-1} in Figure C.1) the token-rotation-timer is loaded with the target rotation time T_{TR} and decrement each bit time.

After receiving a token, the DL-entity may carry out high priority and low priority message transfer periods, according to the following rules.

– On reception of the token (t_i) the T_{RR} (real rotation time for the last token cycle) is derived from the token-rotation-timer by reading its current value that represents T_{TH} and calculating $T_{RR(i)} = T_{TR} – T_{TH}$. The timer is loaded with the value T_{TR} and restarted.
– One high priority message transfer period is always possible at this moment (just after token reception).
– A further high priority or low priority message transfer period or generally a low priority message transfer period may only be carried out if time to hold the token is still available at the instant of execution, that means that the actual value of T_{RR} of the last, just ending token cycle (from the viewpoint of this station) is less than the current value of the token-rotation-timer.

In order to avoid unnecessary timers in an implementation, the increase of T_{RR} during token usage is measured indirectly by use of the token-rotation-timer, which is measuring the token hold time of the just started actual token cycle. As known from the timer concepts of IEC 61131-3, one timer can be used for measurement of parallel but shifted times and therefore the availability of the token holding time is derived from the token-rotation-timer. Thus, a message transfer period can be executed, if at the instant of execution, the read value of the token-rotation-timer is greater than T_{RR}, as shown in Figure C.1.

Figure C.1 shows the relationship of T_{TR}, T_{RR}, T_{TH}.

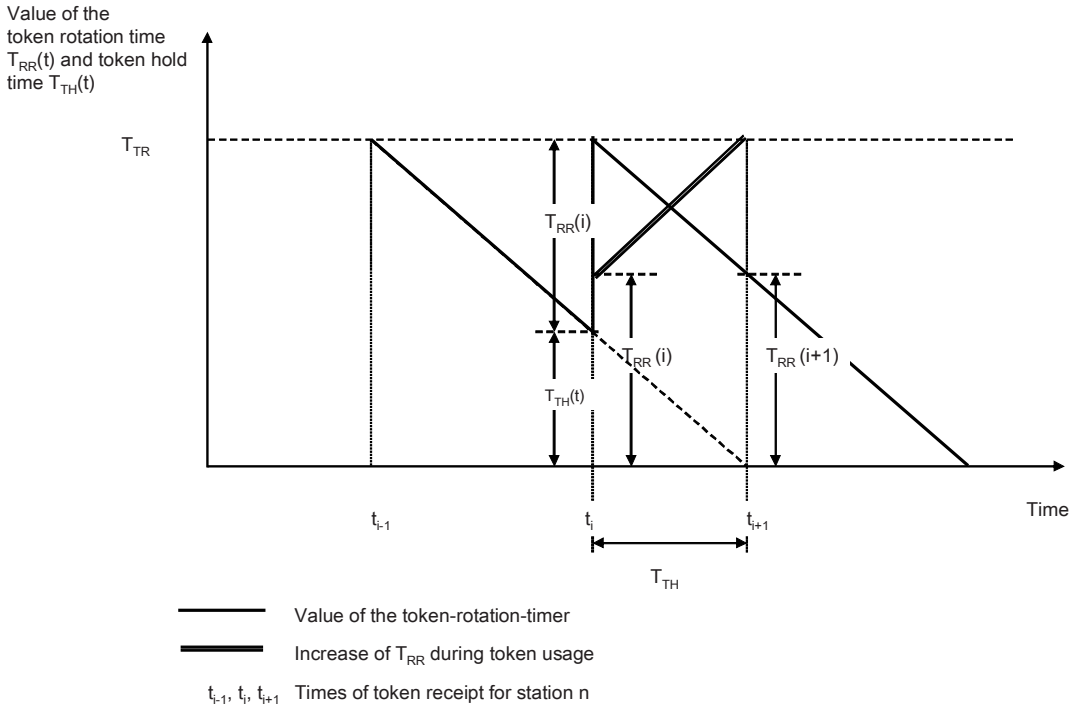


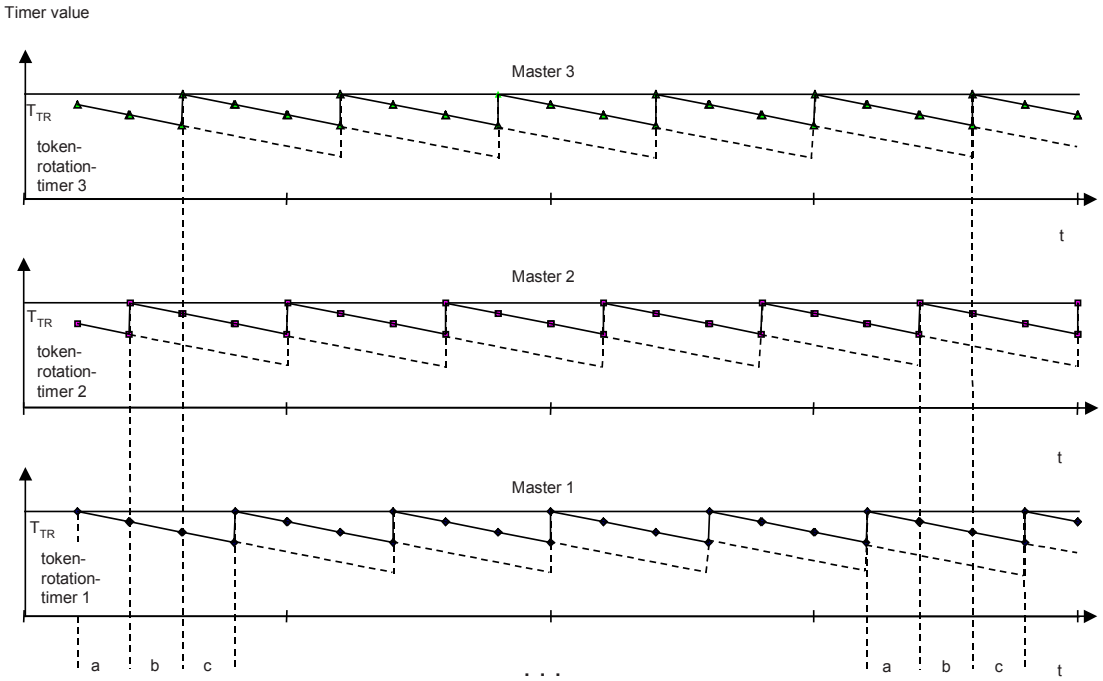
Figure C.1 – Derivation of the token holding time (T_{TH})

C.2 Examples for token passing procedure

The consumption of available token holding time depends of the actual working load. In the following examples the different working load situations are demonstrated together with the usage of the token holding time.

The figures illustrate the behavior of the token passing in cases of varying load. The used time scale only demonstrates the relationship without any relation to real-time. Below are listed some enumerated cases to explain the Token Passing procedure.

Case one: A startup situation is depicted in Figure C.2. The token is passed by the token holder without usage of the available token holding time (T_{TH}). During the startup phase no messages are sent. As depicted in Figure C.2, the timer value of each token holder is always greater than zero in case of token receipt, for example, the read value from token-rotation-timer is greater than zero. Thus T_{TH} is still available.

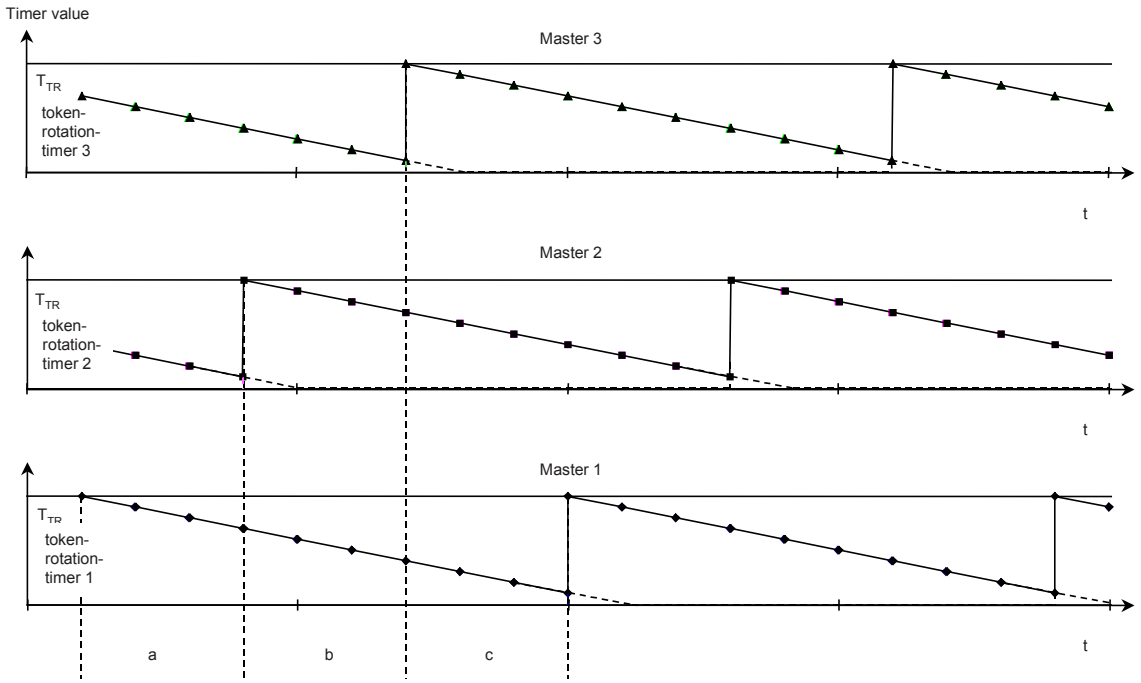


a $T_{TH} > 0$, no message transfer, token passed from Master 1 to Master 2
b $T_{TH} > 0$, no message transfer, token passed from Master 2 to Master 3
c $T_{TH} > 0$, no message transfer, token passed from Master 3 to Master 1

Figure C.2 – No usage of token holding time (T_{TH})

Case two: The bus transfer time is shared fairly between all token holders (see Figure C.3). Each token holder only uses a part of the token holding time (T_{TH}). The actual token holder only sends one message in each token cycle. After that it passes the token to the next token holder. The T_{TH} used by a token holder has a constant (for that token holder) value (see Figure C.3: a, b, c).

As depicted in Figure C.3 some T_{TH} time is still available in each station at the time of token receipt because the read value of the token-rotation-timer is greater than zero at that instant.



a $T_{TH} > 0$, message transfer by Master 1, token passed from Master 1 to Master 2
b $T_{TH} > 0$, message transfer by Master 2, token passed from Master 2 to Master 3
c $T_{TH} > 0$, message transfer by Master 3, token passed from Master 3 to Master 1

Figure C.3 – Usage of token holding time (T_{TH}) for message transfer (equivalence between T_{TH} of each Master station)

Case three: This case is characterized by a load change, from no load to a maximum load, that is a worst-case situation. This means that each token holder has a maximum amount of messages to send after a period of no message requests. That leads to the fact that each token holder tries to use the whole available token holding time for that token rotation. The Token Passing procedure guarantees a fair access to the medium for all token holders even in this situation.

In case of using the whole available bus transfer time (T_{TR}) by one token holder within one token cycle, no local T_{TH} time is available on token arrival in any of the other token holders for that cycle, so they are only allowed to send one high priority message. Within the next token cycle the token holder that had used the whole T_{TR} previously is limited to one high priority message, after which it immediately passes the token. This situation is depicted in Figure C.4. After a startup phase in which the token is passed by the token holders, Master station 1 uses the whole available bus transfer time to send messages (see Figure C.4: d). In this case Master station 2 and Master station 3 have no T_{TH}, indicated by a read value of zero of the token-rotation-timer at token receipt. Because Master station 3 has to transfer a high priority message it sends this message after receiving the token (see Figure C.4: f).

In the next token cycle (g), Master station 1 passes the token immediately after token receipt as its read timer value has reached zero and it has no high priority messages waiting. Now Master station 2 has the possibility to use the whole bus transfer time for message transfer within this token cycle (h). Master station 2 also uses all the available T_{TH} time because of its high working load, after which it is forced to pass the token to Master station 3.

Master station 3 is now limited by its local T_{TH} that has reached zero, and as it has no high priority messages, it passes the token immediately after receipt (see Figure C.4: i). In the following token cycle no T_{TH} is available for Master station 1 or Master station 2. They could send one high priority message, but in this example they pass the token immediately without message transfer (see Figure C.4: j and k). At this point in the cycle the read value of the token-rotation-timer in Master station 3 is greater than zero, so this token holder can use the available T_{TH} time for message transfer (see Figure C.4: l).

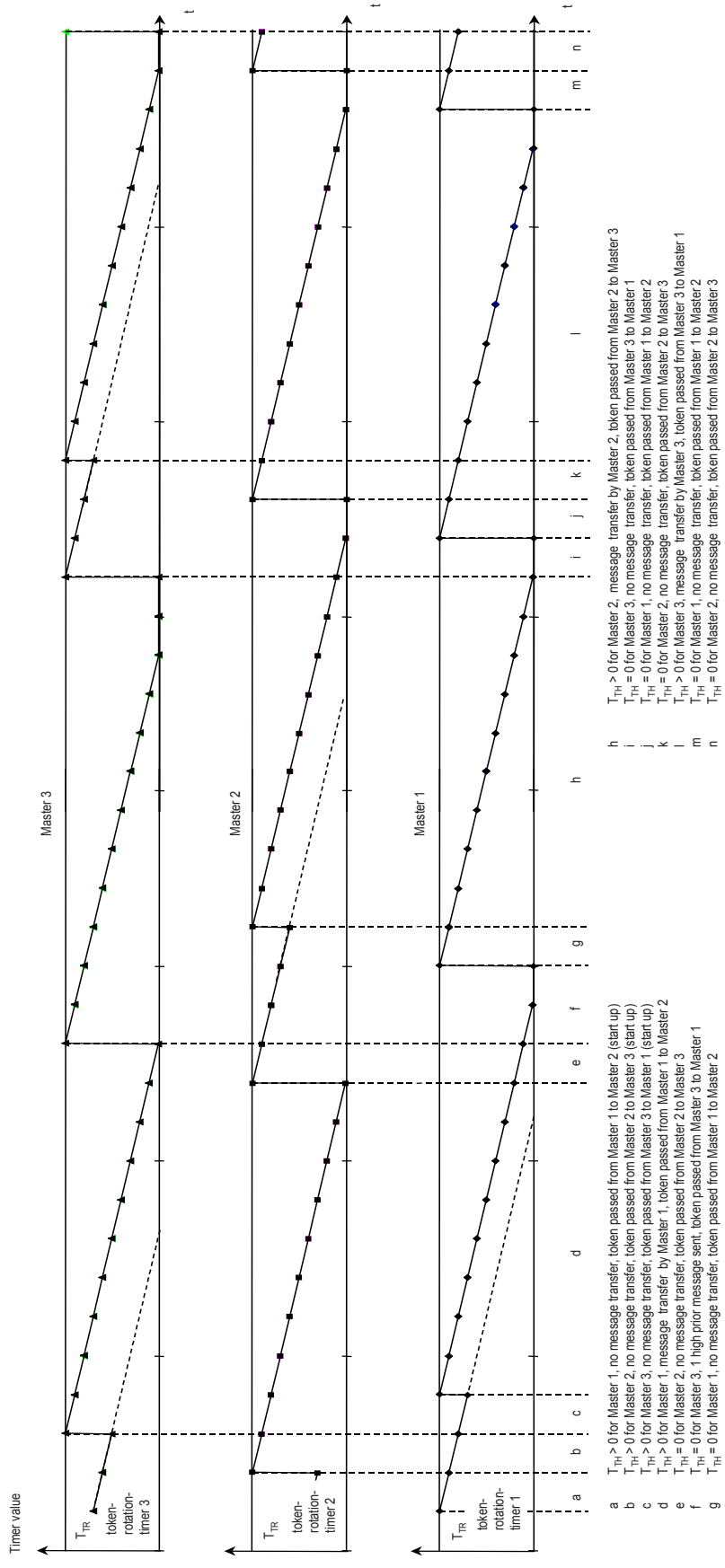


Figure C.4 – Usage of token holding time (T_{TH}) in different working load situations

C.3 Examples for message transfer periods – asynchronous transmission

The following examples give a detailed overview about the possible message transfer periods for asynchronous transmission according to typical messages used by the DLS-user:

Symbols used in the following examples:

T_{SC}	bit time to transmit a single character SC; (11 bits)
T_{SAP}	bit time to transmit D_SAP_index and S_SAP_index; (22 bits)
T_{SD1}	bit time to transmit a DLPDU with start delimiter SD1; (66 bits)
T_{SD2}	bit time to transmit a DLPDU with start delimiter SD2 (without DLSDU and Address-Extensions); (99 bits)
T_{SD3}	bit time to transmit a DLPDU with start delimiter SD3, DLPDU format of fix length with data field; (154 bits)
Req-PDU	number of octets in the DATA_UNIT
Res-PDU	number of octets in the DATA_UNIT

EXAMPLES

a) GAP request with reply (T_{GAP1})

$$T_{GAP1} = T_{ID1} + T_{SDR} + 2 \times T_{SD1} + 2 \times T_{TD} \quad \text{(C.1)}$$

b) GAP request with no reply (T_{GAP2})

$$T_{GAP2} = T_{ID1} + T_{SD1} + T_{SL} \quad \text{(C.2)}$$

c) Cyclic message transfer (T_{CY})

$$T_{CY1} = T_{ID1} + T_{SDR} + 2 \times T_{SD2} + 2 \times T_{TD} + 11 \times (\text{Req-PDU} + \text{Res-PDU}) \quad \text{(C.3)}$$

$$T_{CY2} = T_{ID1} + T_{SDR} + T_{SD2} + T_{SC} + 2 \times T_{TD} + 11 \times \text{Req-PDU} \quad \text{(C.4)}$$

$$T_{CY3} = T_{ID1} + T_{SD2} + T_{SL} + 11 \times \text{Req-PDU} \quad \text{(C.5)}$$

d) Acyclic message transfer with response (T_{ACR})

$$T_{ACR1} = T_{ID1} + T_{SDR} + 2 \times T_{SD2} + 2 \times T_{TD} + 2 \times T_{SAP} + 11 \times \text{Res-PDU} \quad \text{(C.6)}$$

$$T_{ACR2} = T_{ID1} + T_{SDR} + T_{SD2} + T_{SD3} + 2 \times T_{TD} + T_{SAP} \quad \text{(C.7)}$$

e) Acyclic message transfer with acknowledgement (T_{ACY})

$$T_{ACY1} = T_{ID1} + T_{SDR} + T_{SD2} + T_{SAP} + T_{SC} + 2 \times T_{TD} + 11 \times \text{Req-PDU} \quad \text{(C.8)}$$

$$T_{ACY2} = T_{ID1} + T_{SDR} + T_{SD2} + T_{SAP} + T_{SD1} + 2 \times T_{TD} + 11 \times \text{Req-PDU} \quad \text{(C.9)}$$

f) Broadcast message transfer (T_{BC})

$$T_{BC} = T_{ID2} + T_{SD2} + T_{SAP} + T_{TD} + 11 \times \text{Req-PDU} \quad \text{(C.10)}$$

Examples for message transfer periods – synchronous transmission

The following examples give a detailed overview about the possible message transfer periods for synchronous transmission according to typical messages used by the DLS-user:

Symbols used in the following examples:

T_{SC}	bit time to transmit a single character SC including CRC; (24 bits)
T_{SAP}	bit time to transmit D_SAP_index and S_SAP_index; (16 bits)
T_{SD1}	bit time to transmit a DLPDU with start delimiter SD1; (48 bits)
T_{SD2}	bit time to transmit a DLPDU with start delimiter SD2 (without DLSDU and Address-Extensions); (72 bits)

T_{SD3}	bit time to transmit a DLPDU with start delimiter SD3, DLPDU format of fix length with data field; (112 bits)
T_{PhL}	bit time to transmit preamble, physical start delimiter and end delimiters as well as post-transmission gap time; unit in bit (40 bits: 16 bits preamble, 8 bits start delimiter and 8 bits end delimiter, 8 bits post-transmission gap time)
Req-PDU	number of octets in the DATA_UNIT
Res-PDU	number of octets in the DATA_UNIT

EXAMPLES

a) GAP request with reply (T_{GAP1})

$$T_{GAP1} = 2 \times T_{PhL} + T_{SDR} + 2 \times T_{SD1} + 2 \times T_{TD} \quad \text{(C.11)}$$

b) GAP request with no reply (T_{GAP2})

$$T_{GAP2} = T_{PhL} + T_{SD1} + T_{SL} \quad \text{(C.12)}$$

c) Cyclic message transfer (T_{CY})

$$T_{CY1} = 2 \times T_{PhL} + T_{SDR} + 2 \times T_{SD2} + 2 \times T_{TD} + 8 \times (\text{Req-PDU} + \text{Res-PDU}) \quad \text{(C.13)}$$

$$T_{CY2} = 2 \times T_{PhL} + T_{SDR} + T_{SD2} + T_{SC} + 2 \times T_{TD} + 8 \times \text{Req-PDU} \quad \text{(C.14)}$$

$$T_{CY3} = T_{PhL} + T_{SD2} + T_{SL} + 8 \times \text{Req-PDU} \quad \text{(C.15)}$$

d) Acyclic message transfer with response (T_{ACR})

$$T_{ACR1} = 2 \times T_{PhL} + T_{SDR} + 2 \times T_{SD2} + 2 \times T_{TD} + 2 \times T_{SAP} + 8 \times \text{Res-PDU} \quad \text{(C.16)}$$

$$T_{ACR2} = 2 \times T_{PhL} + T_{SDR} + T_{SD2} + T_{SD3} + 2 \times T_{TD} + T_{SAP} \quad \text{(C.17)}$$

e) Acyclic message transfer with acknowledgement (T_{ACY})

$$T_{ACY1} = 2 \times T_{PhL} + T_{SDR} + T_{SD2} + T_{SAP} + T_{SC} + 2 \times T_{TD} + 8 \times \text{Req-PDU} \quad \text{(C.18)}$$

$$T_{ACY2} = 2 \times T_{PhL} + T_{SDR} + T_{SD2} + T_{SAP} + T_{SD1} + 2 \times T_{TD} + 8 \times \text{Req-PDU} \quad \text{(C.19)}$$

f) Broadcast message transfer (T_{BC})

$$T_{BC} = T_{PhL} + T_{SD2} + T_{SAP} + T_{TD} + 8 \times \text{Req-PDU} \quad \text{(C.20)}$$

Bibliography

IEC 60870-5-1, *Telecontrol equipment and systems – Part 5: Transmission protocols – Section One: Transmission frame formats*

IEC 61158-1, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-5-3, *Industrial communication networks – Fieldbus specifications – Part 5-3: Application layer service definition – Type 3 elements*

IEC 61158-6-3, *Industrial communication networks – Fieldbus specifications – Part 6-3: Application layer protocol specification – Type 3 elements*

IEC 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

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

ISO/IEC 3309, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures – Frame structure*[2]

ISO/IEC 8802 (all parts), *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks*

_____

[2] Withdrawn.

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

bsi.

...making excellence a habit.™