IEC 61158-4-24

Edition 3.0 2023-03

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications –
Part 4-24: Data-link layer protocol specification – Type 24 elements

IEC 61158-4-24:2023-03(en)

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

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

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

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

IEC publications search - webstore.iec.ch/advsearchform
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee, …). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: sales@iec.ch.

IEC Products & Services Portal - products.iec.ch
Discover our powerful search engine and read freely all the publications previews. With a subscription you will always have access to up to date content tailored to your needs.

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

IEC 61158-4-24

Edition 3.0 2023-03

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications –
Part 4-24: Data-link layer protocol specification – Type 24 elements

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.20; 35.110

ISBN 978-2-8322-6556-7

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 4-24: Data-link layer protocol specification – Type 24 elements

FOREWORD

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

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

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

IEC 61158-4-24 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation. It is an International Standard.

This third edition cancels and replaces the second edition published in 2019. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

- addition of a new cyclic transmission mode which called "no time slot type" in Subclause 4.3.2.4;
- addition of a new frame format for no time slot type in Subclause 5.4;
- addition of a new DLE element procedure for no time slot type in Subclause 6.2.3.2.4, 6.3.3.2.2.4, 6.3.3.3.2.4;
- addition of a new DLM protcol machine for no time slot type in Subclause 7.5, 7.6; and
- spelling and grammar.

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

Draft	Report on voting
65C/1202/FDIS	65C/1243/RVD

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

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

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

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

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

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

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 document 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:

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

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

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

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 4-24: Data-link layer protocol specification – Type 24 elements

1 Scope

1.1 General

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

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

- in a synchronously-starting cyclic manner, according to a pre-established schedule, or
- in an acyclic manner, as requested by each of those data-link entities.

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

1.2 Specifications

This part of IEC 61158 provides specifies

- 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 datalink service provider;
- procedures for giving communications opportunities to all participating DL-entities (DLEs), sequentially and in a cyclic manner for deterministic and synchronized transfer at cyclic intervals up to 64 ms;
- procedures for giving communication opportunities available for time-critical data transmission together with non-time-critical data transmission without prejudice to the time-critical data transmission;
- procedures for giving cyclic and acyclic communication opportunities for time-critical data transmission with prioritized access;
- procedures for giving communication opportunities based on ISO/IEC/IEEE 8802-3 medium access control, with provisions for nodes to be added or removed during normal operation;
- the structure of the fieldbus DLPDUs used for the transfer of data and control information by the protocol of this document, and their representation as physical interface data units.

1.3 Procedures

The procedures are defined in terms of

- the interactions between peer DL-entities through the exchange of fieldbus DLPDUs;
- the interactions between a DL-service (DLS) provider and a DLS-user in the same system through the exchange of DLS primitives;
- 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 document also specifies conformance requirements for systems implementing these procedures. This document does not contain tests to demonstrate compliance with such requirements.

2 Normative references

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

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

IEC 61158-2:2023, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-3-24:2023, *Industrial communication networks – Fieldbus specifications – Part 3-24: Data-link layer service definition – Type 24 elements*

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

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

ISO/IEC/IEEE 8802-3:2021, *Information technology – Telecommunications and exchange between information technology systems — Requirements for local and metropolitan area networks –Part 3: Standard for Ethernet*

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

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

ISO/IEC 13239:2002, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*

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

3 Terms, definitions, symbols, abbreviated terms and conventions

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

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

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

3.1 Reference model terms and definitions

This document 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	**acknowledgement**	[ISO/IEC 7498-1]
3.1.2	**correspondent (N)-entities**	[ISO/IEC 7498-1]
	correspondent DL-entities (N=2)	
	correspondent Ph-entities (N=1)	
3.1.3	**DL-address**	[ISO/IEC 7498-3]
3.1.4	**DL-protocol**	[ISO/IEC 7498-1]
3.1.5	**DL-protocol-data-unit**	[ISO/IEC 7498-1]
3.1.6	**DL-service-data-unit**	[ISO/IEC 7498-1]
3.1.7	**DLS-user**	[ISO/IEC 7498-1]
3.1.8	**DLS-user-data**	[ISO/IEC 7498-1]
3.1.9	**Event**	[ISO/IEC 19501]
3.1.10	**layer-management**	[ISO/IEC 7498-1]
3.1.11	**primitive name**	[ISO/IEC 7498-1]
3.1.12	**Reset**	[ISO/IEC 7498-1]
3.1.13	**Segmenting**	[ISO/IEC 7498-1]
3.1.14	**State**	[ISO/IEC 19501]
3.1.15	**state machine**	[ISO/IEC 19501]
3.1.16	**systems-management**	[ISO/IEC 7498-1]
3.1.17	**Transition**	[ISO/IEC 19501]
3.1.18	**(N)-entity**	[ISO/IEC 7498-1]
	DL-entity (N=2)	
	Ph-entity (N=1)	
3.1.19	**(N)-layer**	[ISO/IEC 7498-1]
	DL-layer (N=2)	

Ph-layer (N=1)

(N)-service [ISO/IEC 7498-1]

DL-service (N=2)

Ph-service (N=1)

(N)-service-access-point [ISO/IEC 7498-1]

DL-service-access-point (N=2)

Ph-service-access-point (N=1)

3.2 Service convention terms and definitions

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

3.2.1 confirm (primitive)

3.2.2 DL-service-primitive

3.2.3 DL-service-provider

3.2.4 DL-service-user

3.2.5 indication (primitive)

3.2.6 request (primitive)

3.2.7 requestor

3.2.8 response (primitive)

3.3 Common terms and definitions

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

3.3.1
acyclic transmission
non-periodic exchange of telegrams

3.3.2
C1 master
one of the network device types that initiates and controls cyclic transmission

3.3.3
C1 message
message communication that C1 master operates as initiator to exchange messages with slave or C2 master

3.3.4
C2 master
one of the network device types that has the function of monitoring all process data transmitted through the network and can initiate message communication

3.3.5
C2 message
message communication that C2 master operates as initiator to exchange messages with the slave or the C1 master

3.3.6
cyclic transmission
periodic exchange of telegrams

3.3.7
data
generic term used to refer to any information carried over a fieldbus

3.3.8
device
physical entity connected to the fieldbus composed of at least one communication element (the network element) and which has a control element and/or a final element (transducer, actuator, etc.)

3.3.9
event driven mode
transmission mode for the application layer protocol of the communication Type 24 in which a transaction of command-response-exchanging arises as user's demands

3.3.10
frame
synonym for DLPDU

3.3.11
initiator
network device that initiates the exchange of process data or message

3.3.12
interface
shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics as appropriate

3.3.13
input data
process data sent by the slave and received by the C1 master

3.3.14
message
ordered series of octets intended to convey information

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

3.3.15
monitor slave
slave that has the function of monitoring all process data transmitted through the network

3.3.16
network
set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.3.17
node
<protocol> single DL-entity as it appears on one local link

3.3.18
node
<networking> end-point of a link in a network or a point at which two or more links meet

3.3.19
output data
process data sent by the C1 master and received by the slaves

3.3.20
protocol
convention about the data formats, time sequences, and error correction in the data exchange of communication systems

3.3.21
real-time communication
transfer of data in real-time

3.3.22
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.

3.3.23
responder
network device that responds process data or message after it has been initiated by initiator

3.3.24
send data with acknowledge
data transfer service with acknowledge of reception from corresponding DLE

3.3.25
send data without acknowledge
data transfer service without acknowledge of reception from corresponding DLE

3.3.26
slave
one of the network device type that accesses the medium only after it has been initiated by C1 maser or C2 master

3.3.27
sending DLS-user
DL-service user that acts as a source of DL-user-data

3.3.28
station
node

3.3.29
topology
physical network architecture with respect to the connection between the network devices of the communication system

3.3.30
transmission cycle
fixed time period of cyclic transmission

3.3.31
time slot
time period reserved so that initiator and responder can exchange one frame respectively

3.4 Symbols and abbreviations

DA	Destination address
DL-	Data-link layer (as a prefix)
DLE	DL-entity (the local active instance of the data-link layer)
DLL	DL-layer
DLM	DL-management
DLME	DL-management entity (the local active instance of DL-management)
DLMS	DL-management service
DLPDU	DL-protocol-data-unit
DLS	DL-service
DLSAP	DL-service-access-point
DLSDU	DL-service-data-unit
FIFO	First-in first-out (queuing method)
ID	Identifier
OSI	Open systems interconnection
PDU	Protocol data unit
Ph-	Physical layer (as a prefix)
PhE	Ph-entity (the local active instance of the physical layer)
PhL	Ph-layer
PHY	Physical layer device (specified in ISO/IEC/IEEE 8802-3)
QoS	Quality of service
RT	Real-time
SAP	Service access point
SDU	Service data unit

3.5 Additional Type 24 symbols and abbreviations

ACK	Acknowledge
C1MSG	C1 message
C2MSG	C2 message
I/O	Input and/or output
MSG	Message
Rx	Receive
SDA	Send data with acknowledge
SDN	Send data without acknowledge
SM	State machine
Tcycle	Transmission cycle
Tslot	Time slot
Tx	Transmit

3.6 Common conventions

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

The service model, service primitives, and time-sequence diagrams used are entirely abstract descriptions; they do not represent a specification for implementation.

Service primitives, used to represent service user/service provider interactions (see ISO/IEC 10731), convey parameters that indicate information available in the user/provider interactions.

This document uses a tabular format to describe the component parameters of the DLS primitives. The parameters that apply to each group of DLS primitives are set out in tables throughout the remainder of this document. Each table consists of up to six columns, containing the name of the service parameter, and a column each for those primitives and parameter-transfer directions used by the DLS:

- the request primitive’s input parameters;
- the indication primitive’s output parameters;
- the response primitive’s input parameters;
- the confirm primitive’s output parameters.

NOTE The request, indication, response and confirm primitives are also known as requestor.submit, acceptor.deliver, acceptor.submit, and requestor.deliver primitives, respectively (see ISO/IEC 10731).

One parameter (or part of it) is listed in each row of each table. Under the appropriate service primitive columns, a code is used to specify the type of usage of the parameter on the primitive and parameter direction specified in the column:

M	parameter is mandatory for the primitive.
U	parameter is a User option, and can be provided or not depending on the dynamic usage of the DLS-user. When not provided, a default value for the parameter is assumed.
C	parameter is conditional upon other parameters or upon the environment of the DLS-user.
(blank)	parameter is never present.

Some entries are further qualified by items in brackets. These may be a parameter-specific constraint:

(=)	parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table.

In any particular interface, not all parameters need to be explicitly stated. Some may be implicitly associated with the primitive.

In the diagrams which illustrate these interfaces, dashed lines indicate cause-and-effect or time-sequence relationships, and wavy lines indicate that events are roughly contemporaneous.

3.7 Additional Type 24 conventions

3.7.1 Primitive conventions

The following notation, a shortened form of the primitive classes defined in 3.2, is used in the figures.

req	request primitive
ind	indication primitive
cnf	confirm primitive (confirmation)

3.7.2 State machine conventions

The protocol sequences are described by means of state machines.

In state diagrams, states are represented as boxes and state transitions are shown as arrows.

Names of states and transitions of the state diagram correspond to the names in the state table. The textual listing of the state transitions is structured as shown in Table 1.

Table 1 – State transition descriptions

No.	Current state	Event /condition =>action	Next state

The description of state machine elements are shown in Table 2.

Table 2 – Description of state machine elements

Description element	Meaning
No	Number of the transition.
Current state, Next state	Names of the originating state and the target state of transition.
Event	Name or description of the trigger event that fires the transition.
/ conditions	Boolean expression, which shall be true for the transition to be fired.
=>action	List of assignments and service or function invocations. The action should be atomic. The preceding "=>" is not part of the action.
NOTE "/ conditions" can be omitted.	

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

Table 3 – Conventions used in state machines

Convention	Meaning
+ - * /	Arithmetic operators
:=	Value of an item on the left is replaced by value of an item on the right. If an item on the right is a parameter, it comes from the primitive shown as an input event.
=	A logical condition to indicate an item on the left is equal to the item on the right.
<	A logical condition to indicate an item on the left is less than the item on the right.
>	A logical condition to indicate an item on the left is greater than the item on the right.
<=	A logical condition to indicate an item on the left is less than or equal to the item on the right.
>=	A logical condition to indicate an item on the left is greater than or equal to the item on the right.
<>	A logical condition to indicate an item on the left is not equal to the item on the right.
&&	Logical "AND"
||	Logical "OR"

4 Overview of DL-protocol

4.1 Characteristic feature of the DL-protocol

Table 4 shows the characteristic features of the DL protocol of Type 24.

Table 4 – Characteristic features of the fieldbus data-link protocol

Profiles	Description
Station type and max. stations	-C1 master (active station with bus access control, 1 station (mandatory)) -C2 master (active station with restricted bus access control, (optional)) -Slave (passive stations without bus access control, max.127)
Station addressing	1 to 255 (255 = global address for broad-cast messages), 8 bit-width address extension for integrated device
Transmission cycle	15,625 μs to 64 ms
DLSDU size	0 octets to 64 octets
Transmission characteristic	-Cyclic data exchange and cyclic event, synchronized with accurate cycle time (jitter below 1 μs) -Max 62 times (n times/1 station) retry within cycle time -Acyclic message transmission

There are three types of stations, the C1 master, the C2 master and the slave. Data exchange is executed between one master station (C1 master or C2 master) and N slave stations. This protocol supports 2 communication modes, cyclic transmission and acyclic transmission.

In the cyclic transmission mode, transmission is executed cyclically with an accurate period. The transmission cycle is set by the C1 master to a value within a range of 15,625 [μs] to 64 [ms]. Since the set value of the transmission cycle is specific to the transmission line, all of the connected slaves shall support that value. It is not permitted to set different transmission cycle values for the slaves connected in the same network.

The transmission cycle has the I/O data exchange band to transmit process data and the message communication band to transmit message. The protocol machine in the C1 master controls transmission sequence in the cyclic transmission mode. The time period for a master station to exchange with one slave station is called time slot. There are two types of communication sequence using time slot, one is "fixed-width time slot type" whose time slot width is the same for all stations, and the other is "configurable time slot type" whose time slot can be defined for each station. As another communication sequence, "no time slot type", which does not use time slot, is provided. All stations shall use the same-data-length frame when fixed-width time slot type. The width of the time slot is static in both types, and the value is set by the DL-management during initialization. Once the cyclic communication starts, it shall not be changed.

The Acyclic transmission mode is used by the DLS-user that operates in the event driven mode. In the acyclic transmission mode, transmissions are executed sporadically. The same transmission sequence and message communication can be executed in the acyclic transmission, as in the cyclic transmission mode without fixing the transmission cycle.

4.2 DL layer component

The DL layer is composed of three sublayers, the CTC (Cyclic transmission control), the SRC (Send Receive Control) and the DLM (Data-link management). The SRC is positioned at lower layer of the CTC and the DLM covers both the CTC sublayer and the SRC sublayer. The data-link layer component is show in Figure 1.

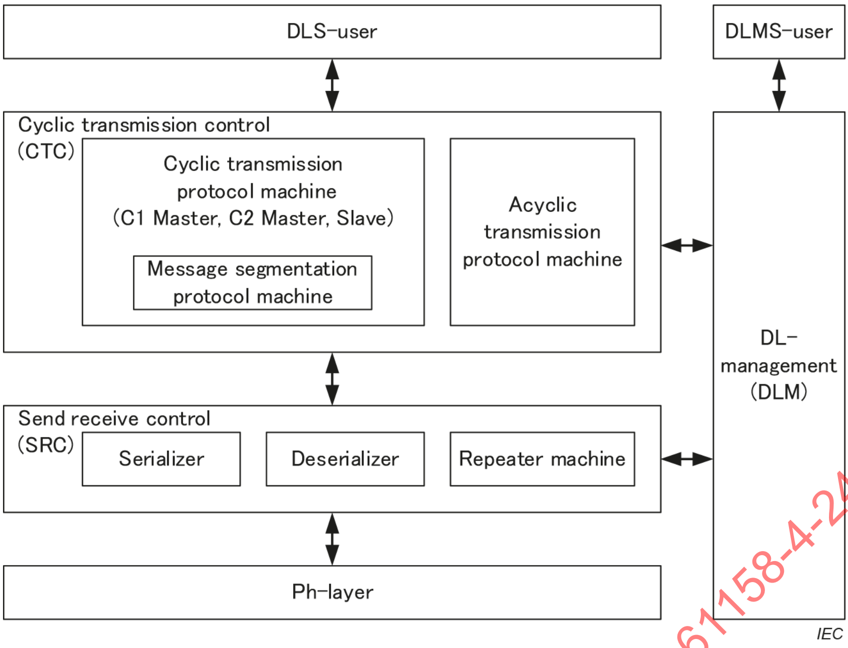


Figure 1 – Data-link layer component

4.2.1 Cyclic transmission control (CTC)

This is a sublayer that builds the DLPDU and executes a protocol machine. It has 2 communication modes, i.e. cyclic transmission mode and acyclic transmission mode. The CTC executes either of them according to a request from the DLMS user.

4.2.2 Send receive control (SRC)

The SRC sends or receives frames by request of the CTC sublayer. It is serialized or de-serialized according to corresponding PHY. When the SRC implements two or more PHY port, the SRC provides frame repeat function between the implemented PHY ports.

4.2.3 DL-management

This is a sublayer that configures the DLE operation by setting the internal variables and manages errors detected by each sublayer.

4.3 Timing sequence

4.3.1 Overview

There are two types of transmission mode, the cyclic transmission mode and the acyclic transmission mode. The communication sequence has three types. "Fixed-width time slot type" is described in 4.3.2.1, "configurable time slot type" is described in 4.3.2.2, and "no time slot type" is described in 4.3.2.4.

The width of the time slot is static for both types, and the value is set by the DL-management during initialization. Once cyclic communication starts, it shall not be changed.

4.3.2 Cyclic transmission mode

4.3.2.1 Fixed-width time slot type

4.3.2.1.1 Overview

Figure 2 shows the transmission sequence of fixed-width time slot type. In this type, all time slots are same period.

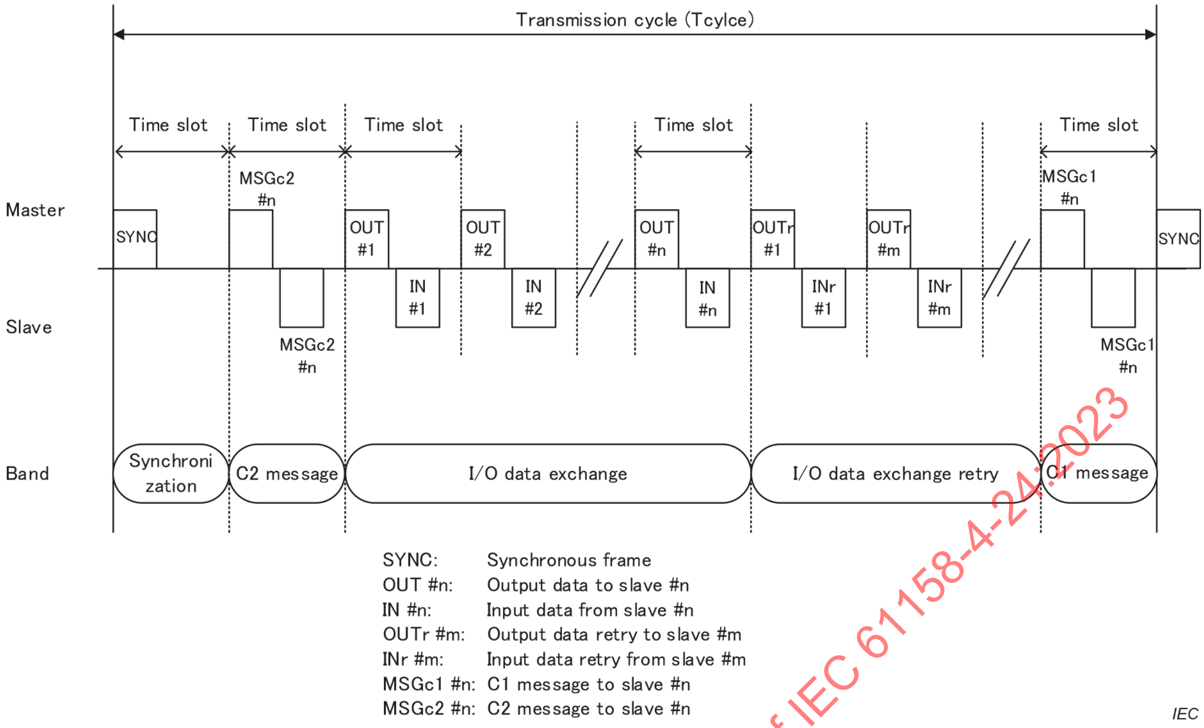


Figure 2 – Timing chart of fixed-width time slot type cyclic communication

4.3.2.1.2 Detailed description of communication band

4.3.2.1.2.1 Synchronization

This is the band through which the C1 master broadcasts a synchronous frame to the slave and the C2 master. One time slot shall be allocated to this band. Within this band, only the transition of the synchronous frame from the C1 master is allowed; the slave and the C2 master are prohibited from transmitting any frame.

4.3.2.1.2.2 C2 message

This is an optional band for a message transmission (C2 message transmission) where the C2 master is the client (primary station) and the C1 master or slave is the server (secondary station). One time slot is allocated to this band, and request and response is transmitted once, respectively.

4.3.2.1.2.3 I/O data exchange

This is the band in which the C1 master exchanges the I/O data with all the slaves that are connected to the network. The time slots of the number of slaves shall be allocated to this band. The C1 master and one slave shall execute the I/O data exchange once within one time slot.

The I/O data exchange retry band is optional. When the I/O data exchange retry band is configured in the transmission cycle, the C1 master shall create a retry list of the slave that failed the I/O data exchange.

4.3.2.1.2.4 I/O data exchange retry

The I/O data exchange retry band is an optional band that the C1 master retries the I/O data exchanges according to the retry list. The C1 master can re-execute the I/O exchange with the retry target slave that has been registered into the retry list in the I/O data exchange band. In this band, some time slots are allocated before cyclic transmission started.

The C1 master can retry according to the registered order of the retry list for up to the number of the allocated time slots. The DLE shall quit retry when it uses all allocated time slots even if a slave that is waiting for retry is registered in the retry list.

4.3.2.1.2.5 C1 message

This is an optional band for a message transmission (C1 message transmission) where the C1 master is the client (initiator) and the C2 master or the slave is the server (responder). One time slot is configured in this band. This band is configured to be shared with the I/O data exchange retry band. When this band is shared and used for the I/O data exchange retry, the C1 message transmission shall be postponed to the C1 message band of the next transmission cycle.

4.3.2.1.3 Estimation of cycle time

The transmission cycle of the fixed-width slot type T_{cycle} is calculated as the sum of the bandwidths described in the following formula.

$$T_{cycle}=T_{sync}+T_{C2msg}+T_{io}+T_{retry}+T_{C1msg} \tag{1}$$

where

T_{sync} is the Sync band;

T_{C2msg} is the C2 message band;

T_{io} is the I/O data exchange band;

T_{retry} is the I/O data exchange retry band;

T_{C1msg} is the C1 message band.

The width of each band shall be allocated with an integral multiple of time slots. The formula shown above can be transformed by indicating the time slot with T_{slot} the number of slave stations connected to the network with N, and the number of retry with N_r.

$$T_{cycle}=T_{slot}+T_{slot}+N\times T_{slot}+N_r\times T_{slot}+T_{slot}=(N+N_r+3)\times T_{slot} \tag{2}$$

The I/O data exchange retry band and the C1 message band are configured to be shared. When they are shared, the formula becomes:

$$T_{cycle}=(N+N_r+2)\times T_{slot} \tag{3}$$

And, the I/O data exchange retry band, the C1 message band and the C2 message band are optional. When the configuration doesn’t assign all of them, the transmission cycle is minimum and is calculated as shown in the following formula:

$$T_{cycle}=(N+1)\times T_{slot} \tag{4}$$

Time slot T_{slot} can be calculated as shown in the following formula:

$$T_{slot}=\max\left(T_{tr\_c}(n)+T_{dly}(n)+T_{gap}+T_{tr\_r}(n)+T_{dly}(n)+T_{gap}\right)=2\times\max\left(T_{tr\_c}(n)+T_{dly}(n)+T_{gap}\right) \quad (5)$$

where

$T_{tr\_c}(n)$ is the output (command) transmission time from C1 master to the slave #n;

$T_{tr\_r}(n)$ is the input (response) transmission time from slave #n to C1 master, it is equal to $T_{tr\_c}(n)$ in case of the fixed-width time slot type;

T_{gap} is the gap between the frames;

$T_{dly}(n)$ is the frame transmission delay time between C1 master and slave #n.

4.3.2.2 Configurable time slot type

4.3.2.2.1 Overview

Figure 3 shows the transmission sequence of configurable time slot type. In this type, the width of time slots that are allocated to exchange the data one by one between the master and the slave is different for each slave. The DLE manages the residual time of the transmission cycle by using the width of the time slots configured for each slave by the DLMS user. Details of each transmission band are described in the following subclauses.

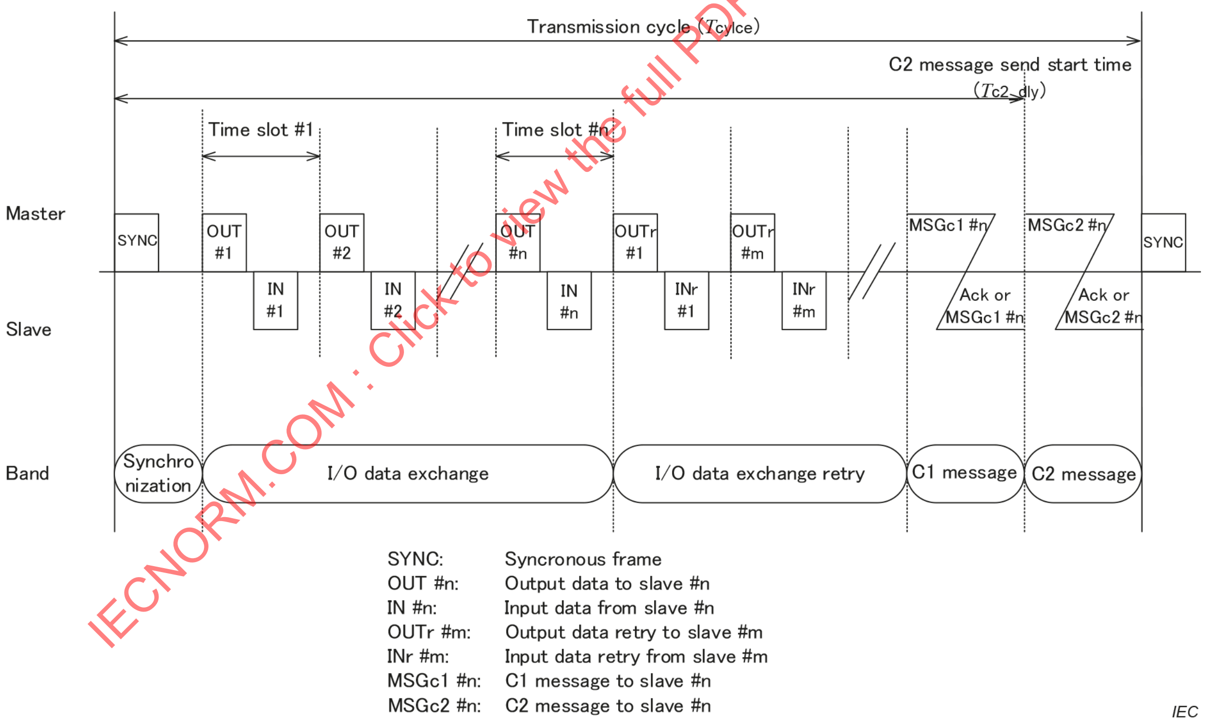


Figure 3 – Timing chart of configurable time slot type cyclic communication

4.3.2.2.2 Detailed description of communication phase

4.3.2.2.2.1 Synchronization

See 4.3.2.1.2.1.

4.3.2.2.2.2 IO data exchange

This is the band where the C1 master executes the I/O data exchange with all slaves connected to the network. The width of this band shall be set in the C1 master before starting cyclic transmission. The C1 master and one slave execute a I/O data exchange which consists of a pair of input and output for each other in each time slot.

When the I/O data exchange retry band which is optional is configured in the transmission cycle, the C1 master shall register the slaves that have failed the I/O data exchange within this band into the retry list as re-transmission targets within the succeeding the I/O data exchange retry band.

4.3.2.2.2.3 IO data exchange retry

This band is basically the same as the case of fixed-width time slot (see 4.3.2.1.2.4). Subclause 4.3.2.2.2.3 describes the differences.

The width of this band shall be set in the C1 master before starting the cyclic transmission. The C1 master can execute the retry for the slaves registered in the retry list in the order of registration, and when the retry succeeds, the C1 master clears the registration. Before executing retry, the C1 master shall compare the time required to complete the I/O data exchange with the slave and the residual time until the end of this band (until C2 message starts). If the residual time is longer than the required time, the C1 master shall execute the retry. If the residual time is equal to or shorter than the required time, the C1 master shall end this band.

When the retry that is executed for a registered the slave does not complete successfully, the C1 master shall clear the registration and then register the slave again at the end of the retry list. The C1 master can repeat retry for the same slaves within the residual time of the band. When the retries for all of the retry targets in the retry list are executed once, the C1 master can execute the retry according to the registered order of the retry list again. The C1 master shall end the band when all of the slaves are cleared from the retry list.

4.3.2.2.2.4 C1 message

This is an optional band for a message transmission (C1 message transmission) where the C1 master is the client (primary station) and the C2 master or the slave is the server (secondary station). The residual time from the end of the I/O data exchange retry band to the start of the C2 message transmission is assigned to this band.

The C1 master can execute the C1 message transmission within the residual time. The C1 master can repeat the C1 message transmission that consists of one pair of request and response within this band. However, the C1 master shall not execute the C1 message transmission if there is no residual time enough for one pair of transmission.

This band is configured to be shared with the I/O data exchange retry band. When this band is shared and used for the I/O data exchange retry, the C1 message transmission shall be postponed to the C1 message band of the next transmission cycle.

4.3.2.2.2.5 C2 message

This is an optional band for a message transmission (C2 message transmission) where the C2 master is the client (primary station) and the C1 master or the slave is the server (secondary station). The start time of this band shall be set both in the C1 master and the C2 master before starting cyclic transmission.

The C2 master can execute the C2 message transmission within the allocated bandwidth. The C2 master can repeat the C2 message transmission that consists of one pair of request and response within this band. However, the C2 master shall not execute the C2 message transmission if there is no residual time enough for one pair of transmission.

This band is configured to be shared with the I/O data exchange retry band and the C1 message band. When this band is shared and used for the I/O data exchange retry or the C1 message, the C2 message shall be postponed to the C2 message band of the next transmission cycle. When this band is shared and the C1 master has finished the I/O data exchange retry and the C1 message before the start time of this band, the C1 master shall send a message token to the C2 master. After the C2 master receives the message token, the C2 master can execute the C2 message transmission immediately.

4.3.2.2.3 Estimation of cycle time

In order for all multiple slaves to synchronize with the C1 master, the C1 master shall measure the transmission delay time of each slave during initialization. The measured delay time shall be locally retained by the C1 master and each slave.

Based on the measured transmission delay time, the C1 master can calculate the response monitoring time and the delay time of cyclic event for each slave to match cyclic event timing in the system.

The cyclic event delay time (T_{idly} in Figure 4) is delivered with the synchronous frame (SYN frame in the Figure 4). The C1 master and each slave generate a cyclic event according to the cyclic event delay time and the transmission delay time retained locally. As a result, cyclic event occurs at the same time throughout the system.

By processing the output data and the input data simultaneously in all slaves at the cycle event timing, the system can operate synchronously.

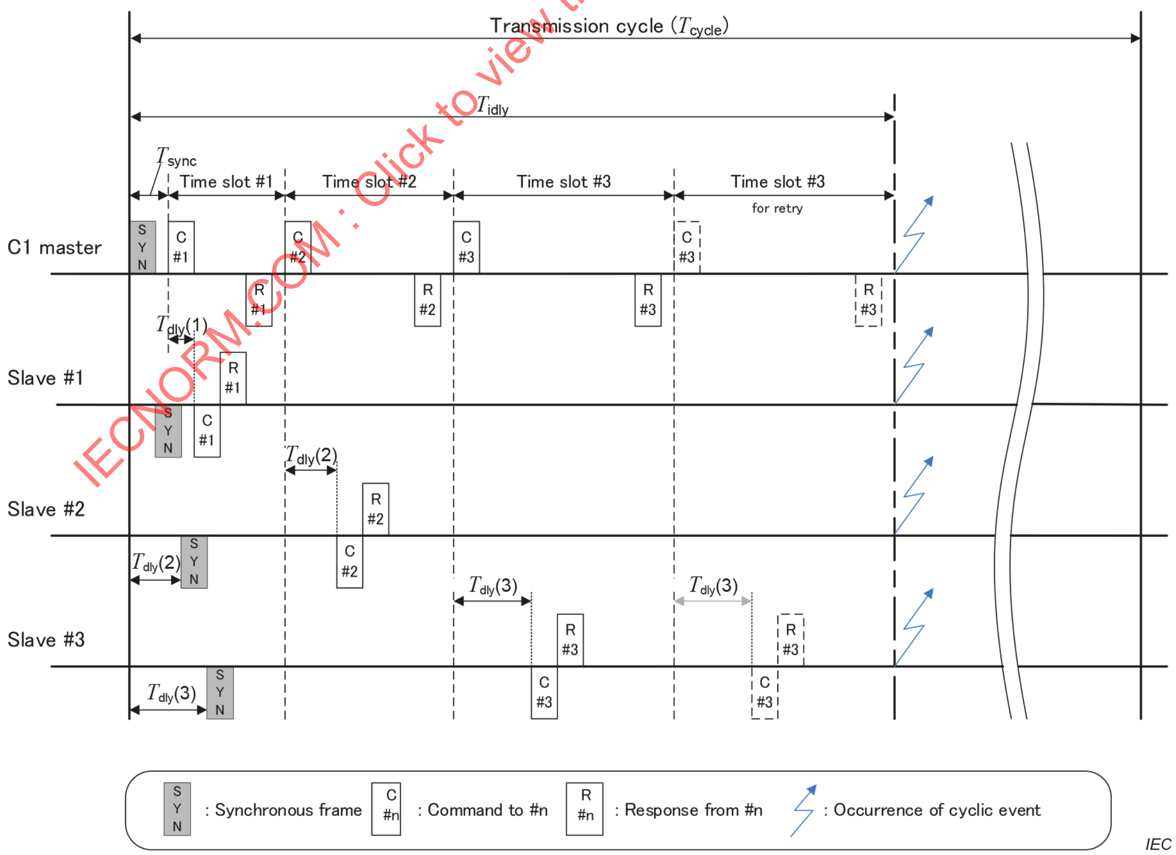


Figure 4 – Schematic diagram of cyclic event occurrence

The transmission cycle of the variable time slot type T_{cycle} is calculated as an aggregation of the band described in 4.3.2.2.1, as shown in the following formula:

$$T_{cycle}=T_{sync}+T_{io}+T_{retry}+T_{C1msg}+T_{C2msg} \quad (6)$$

where

T_{sync} is the Synchronization band;

T_{io} is the I/O data exchange band;

T_{retry} is the I/O data exchange retry band;

T_{c1msg} is the C1 message band;

T_{c2msg} is the C2 message band.

The I/O data exchange retry band and the C1 message band are configured to be shared. When they are shared, the formula becomes:

$$T_{cycle}=T_{sync}+T_{io}+\max\left(T_{retry},T_{C1msg}\right)+T_{C2msg} \quad (7)$$

And, the I/O data exchange retry band, the C1 message band and the C2 message band are optional. When the configuration doesn't assign all of them, the transmission cycle is minimum and is calculated as shown in the following formula:

$$T_{cycle}=T_{sync}+T_{io} \quad (8)$$

The calculation method for the bands mentioned above is shown in the following.

- Synchronization band

  The Sync band T_{sync} is calculated as follows: where T_{tr_s} is the transmission time of the synchronous frame, and T_{gap} is the gap between the frames:

$$T_{sync}=T_{tr\_s}+T_{gap} \quad (9)$$

- I/O data exchange band

  The I/O data exchange band T_{io} is calculated as follows: where N is the number of slaves, $T_{tr_c}(n)$is the output (command) transmission time from the C1 master to the slave #n, $T_{tr\_r}(n)$ is the input (response) transmission time from the slave #n, $T_{dly}(n)$is the frame transmission delay time between the C1 master and the slave #n, and T_{gap} is the inter-packet gap:

$$T_{io}=\sum_{n=1}^{N}\left(T_{tr\_c}(n)+T_{dly}(n)+T_{gap}+T_{tr\_r}(n)+T_{dly}(n)+T_{gap}\right)= \\ \sum_{n=1}^{N}\left(T_{tr\_c}(n)+T_{tr\_r}(n)+2\times T_{dly}(n)\right)+2\times N\times T_{gap} \quad (10)$$

The I/O data exchange retry band T_{retry} is calculated as follows: where N_r is the maximum number of the retry:

$$T_{retry}=\sum_{r=1}^{N_r}\left(T_{tr\_c}(r)+T_{dly}(r)+T_{gap}+T_{tr\_r}(r)+T_{dly}(r)+T_{gap}\right)= \sum_{r=1}^{N_r}\left(T_{tr\_c}(r)+T_{tr\_r}(r)+2\times T_{dly}(r)\right)+2\times N_r\times T_{gap} \quad (11)$$

- C1 message band

  The C1 message band T_{c1msg} is calculated as follows: where $T_{tr\_clc}(m_1)$ is the request transmission time from the primary station (C1 master) to the secondary station m_1 (m_1 is the station number of slaves or the C2 master that becomes the secondary station), $T_{tr\_clr}(m_1)$ is the response transmission time from the secondary station to the primary station, $T_{dly}(m_1)$ is the transmission delay between the primary station and the secondary station, T_{c1msg} is the number of the C1 messages:

$$T_{c1msg}=N_{c1msg}\times\left(T_{tr\_c1c}(m_1)+T_{dly}(m_1)+T_{gap}+T_{tr\_c1r}(m_1)+T_{dly}(m_1)+T_{gap}\right)= N_{c1msg}\times\left(T_{tr\_c1c}(m_1)+T_{tr\_c1r}(m_1)+2\times T_{dly}(m_1)+2\times T_{gap}\right) \quad (12)$$

- C2 message band

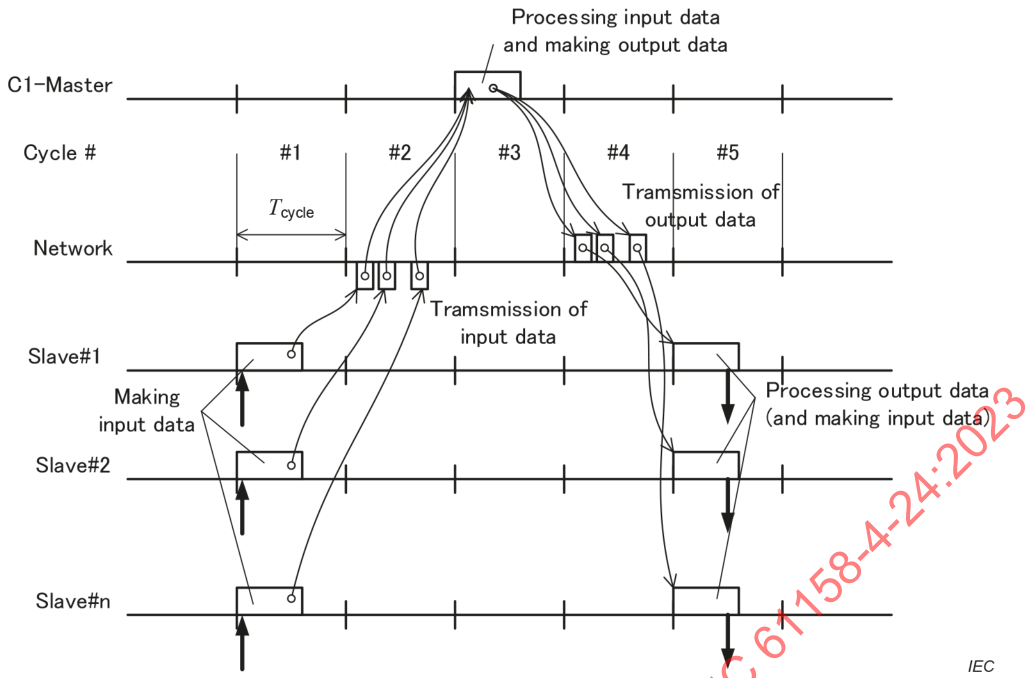
  The C2 message band T_{c2msg} is calculated as follows: where $T_{tr\_c2c}(m_2)$ is the request transmission time from the primary station (C2 master) to the secondary station m_2(m_2 is the station number of the slave or the C1 master that becomes the secondary station), $T_{tr\_c2r}(m_2)$ is the response transmission time from the secondary station to the primary station, $T_{dly}(m_2)$ is the transmission delay between the primary station and the secondary station, N_{c2msg} is the number of the C2 messages:

$$T_{c2msg}=N_{c2msg}\times\left(T_{tr\_c2c}(m_2)+T_{dly}(m_2)+T_{gap}+T_{tr\_c2r}(m_2)+T_{dly}(m_2)+T_{gap}\right)= N_{c2msg}\times\left(T_{tr\_c2c}(m_2)+T_{tr\_c2r}(m_2)+2\times T_{dly}(m_2)+2\times T_{gap}\right) \quad (13)$$

4.3.2.3 Timing relationship between cyclic transmission and data processing

Subclause 4.3.2.3 explains the timing relationship between the cyclic transmission and the data processing by using Figure 5. In cycle #1, the slaves latch input and make the input data to be sent. The input data that each slave made is transmitted to the C1 master in cycle #2. Though it is received by the C1 master, it is not processed by the C1 master at this time. The C1 master starts to process it at the top of the cycle #3. Therefore, the delay from the timing of the slave's latched input to the timing of the master processing it is two transmission cycles.

Similarly, the output data that the C1 master made at cycle #3 is transmitted to all the slaves, slave by slave, in cycle #4. Though it is received by each slave in cycle #4, it is not processed by the slave at this time. All the slaves start to process it all together at the top of cycle #5. Therefore, the delay from the timing of the master's making the output data to the timing of the slave processing it is two transmission cycles, that is same as the input data.



NOTE Output data and input data are transmitted in every cycle, but these drawings are omitted in this figure to explain easily. Data processing by C1 master and slaves in every cycle are also omitted.

Figure 5 – Timing relationship between cyclic transmission and data processing

4.3.2.4 No time slot type

4.3.2.4.1 Overview

Figure 6 shows the transmission sequence of no time slot type. The C1 master of this type sends one frame of output data. After certain time from or at the same time with the timing of receiving of this output data, the C2 master and the slave respond with the input data to the C1 master. As the input data size differs according to slave, the time width for sending the input data also differs according to slave. Details of the timing of receiving of output data and the timing of responding of input data are as described in 4.3.2.4.3.

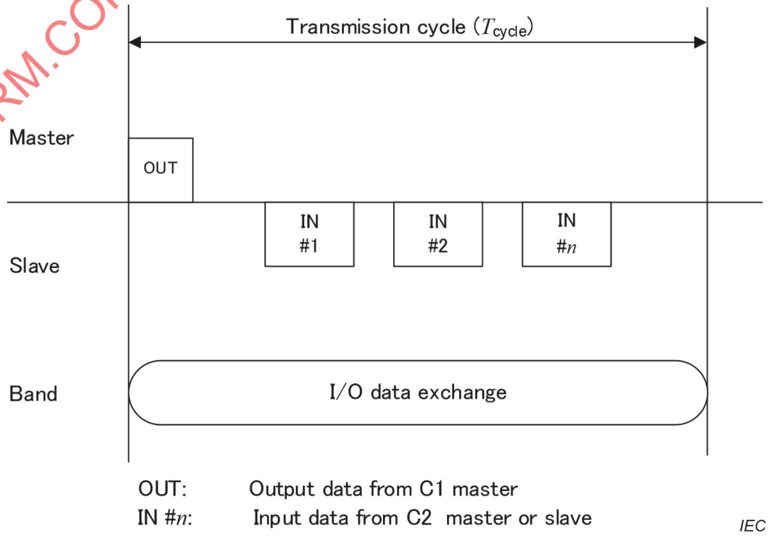


Figure 6 – Timing chart of no time slot type cyclic communication (Master send common address)

Also, the basic transmission cycle of no time slot type can be set as the integer multiple of basic transmission cycle for each slave (T_{cyc}). This is the communication with multiple transmission cycles. For the communication with multiple transmission cycles, "individual transmission cycle multiple (T_{cc_mul})" and "individual transmission position number (T_{cc_pos})" can be set for each slave (see IEC 61158-3-24, 5.3.2.2.2, Table 14).

T_{cc_mul} indicates the transmission cycle in which the slave returns a response. T_{cc_pos} indicates the timing at which transmission cycle the slave returns a response. The C1 master shall send the command frame with the Cycle Counter setting, which is to be incremented for each transmission cycle. The slaves shall divide the Cycle Counter value with T_{cc_mul} value set for the slaves and output division remainder. If the division remainder match with T_{cc_pos}, they shall respond to the C1 master. Figure 7 shows the timing chart when the setting of all slaves is T_{cc_mul} = 1 and T_{cc_pos} = 0. All slaves can respond with the input data to every transmission cycle.

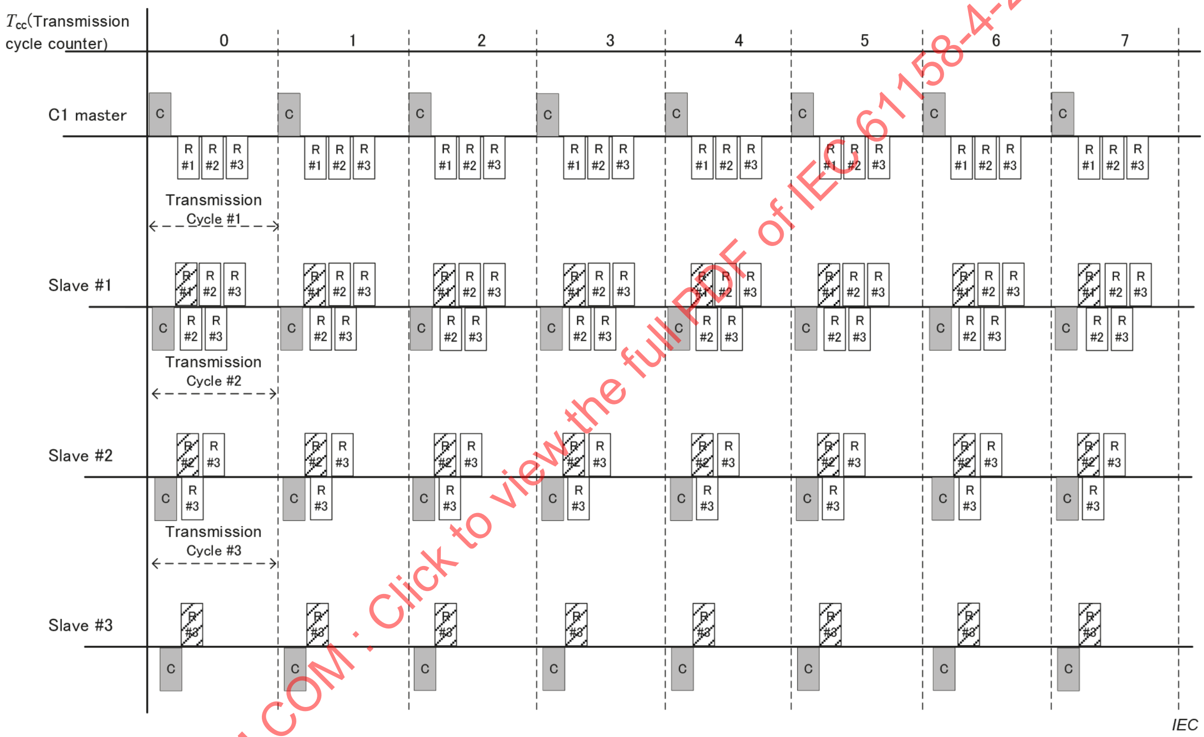


Figure 7 – Timing chart for multiple transmission cycle setting

Figure 8 shows the timing chart when individual transmission cycles are set for each slave. As Slave #1 has the setting of T_{cc_mul} = 1 and T_{cc_pos} = 0, it returns the response (R#1) for all transmission cycles. As Slave #2 has the setting of T_{cc_mul} = 2 and T_{cc_pos} = 1, it returns the response (R#2) once every two transmission cycles and at the second transmission cycle. As Slave #3 has the setting of T_{cc_mul} = 4 and T_{cc_pos} = 2, it returns the response (R#3) once every four transmission cycles and at the third transmission cycle.

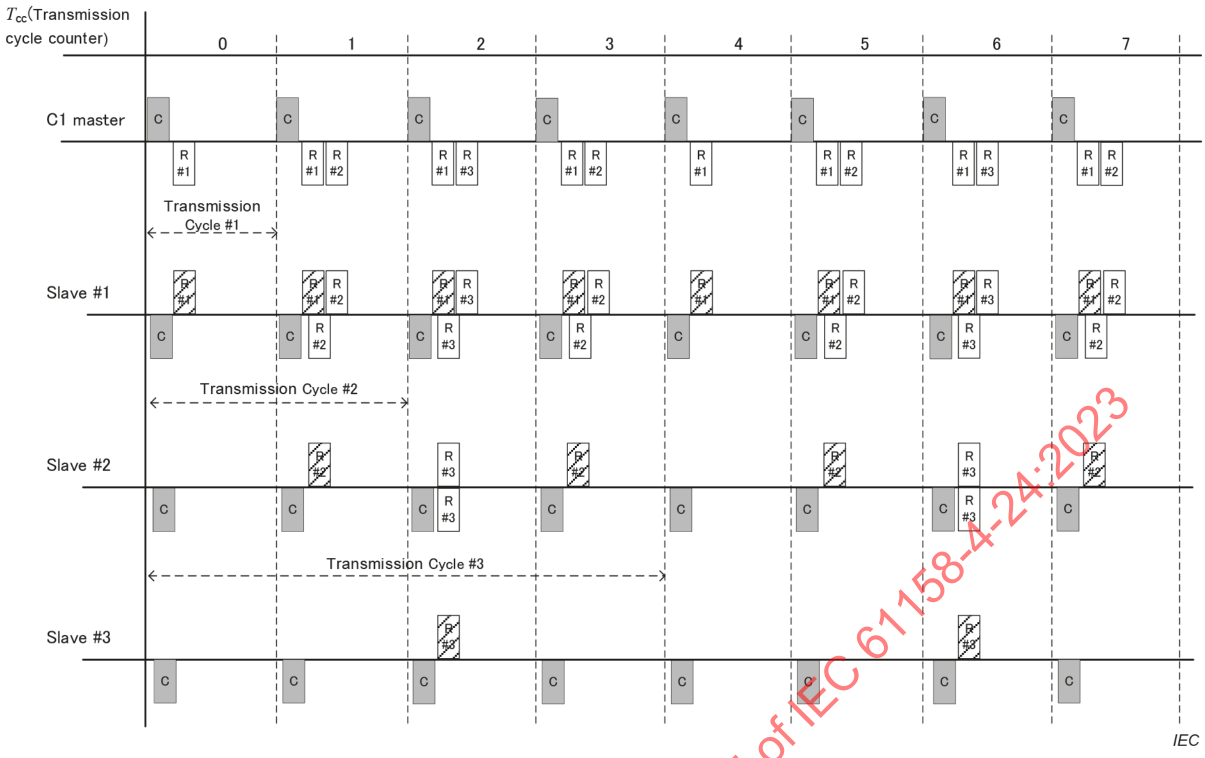


Figure 8 – Timing chart for multiple transmission cycle setting igure title

4.3.2.4.2 Detailed description of I/O data exchange band

The C1 master exchanges the I/O data with all the C2 master and slaves that are connected to the network. The C1 master should transmit the output data and all slaves should respond with the input data.

The C1 master can send the output data by designating the C2 master or the slave in the address field. The designated C2 master or the slave can respond with the input data corresponding to the output data and other slaves whose address does not match can return the input data.

4.3.2.4.3 Estimation of cycle time

The cycle time of no time slot type is defined in the following formula:

$$T_{\text{cycle}}=T_{\text{io}} \quad (14)$$

where

T_{io} is the I/O data exchange band.

Figure 9 shows the schematic diagram for estimating the cycle time for this type. In Figure 9, the nodes for the C1 master, the C2 master, the number of slaves, and each connection node are defined as follows:

where

N is the total number of nodes for connecting the C1 master, the C2 master and slave;

$\#n$ is the node number assigned by the C1 master;

L is the total number of C2 masters;

$\#l$ is the C2 master number;

M is the total number of slaves;

#m is the slave number.

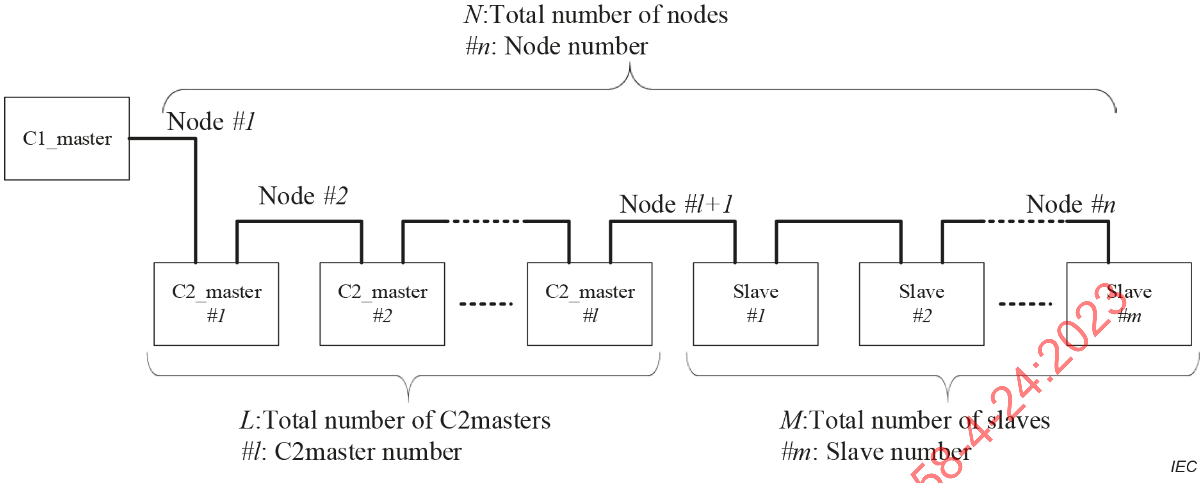


Figure 9 – Schematic diagram for connection

The formula for the estimation of transmission cycles of this type is available for the following two operations. One is when the response with the input data is made after certain time interval from the timing of receiving by multiple C2 masters and multiple slaves of the output data. The other is when the response with the input data is made at the same time as the timing of receiving by multiple C2 masters and multiple slaves.

Figure 10 shows the timing chart when each slave responds with the input data after certain time interval from the timing of receiving of the output data.

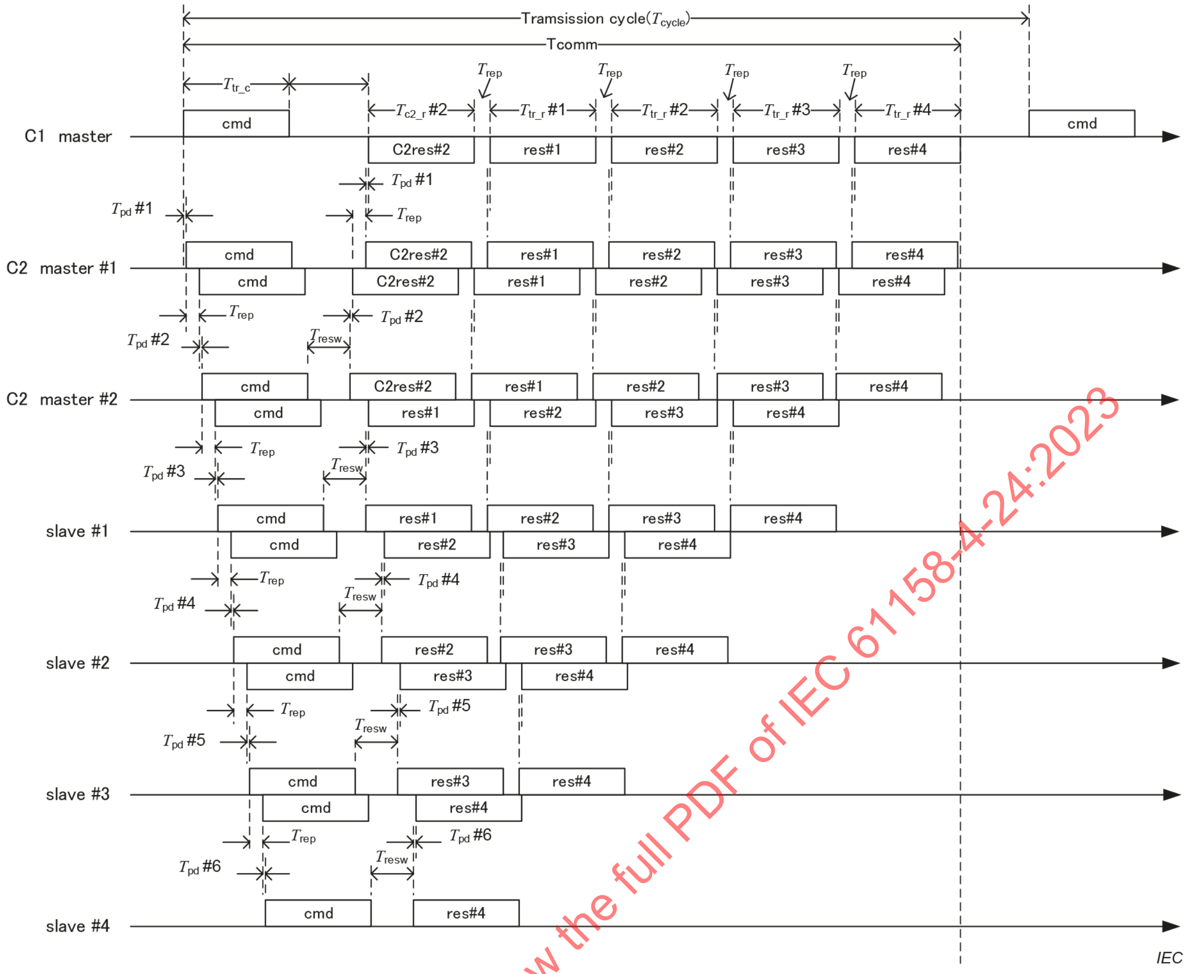


Figure 10 – Schematic diagram of INPUT data response timing at the same interval

With Figure 10 as the operating condition, the Formula (15) is given as follows:

$$T_{\text{cycle}}=2\times\sum_{n=1}^{N}\left(T_{\text{pd}}(\#n)+T_{\text{rep}}\times(\text{n-1})\right)-T_{\text{pd}}(N)+T_{\text{tr\_c}}+T_{\text{resw}}+T_{\text{c2\_r}}(\#l)+\sum_{m=1}^{N-L}\left(T_{\text{tr\_r}}(\#m)\right) \qquad (15)$$

where

$T_{pd}(\#n)$ is the propagation delay time of the n-th node;

T_{rep} is the output (command) or the input (response) repeating time in the slave;

T_{tr_c} is the output (command) transmission time to send from the C1 master to all slaves;

T_{resw} is the response waiting time of the C2 master or the slave to the output (command) from the C1 master;

$T_{c2\_r}(\#l)$ is the input (response) transmission time from the l–th C2 master to the C1 master;

$T_{tr\_r}(\#m)$ is the input (response) transmission time from the m–th slave to the C1 master.

Next, Figure 11 shows the timing chart when each slave responds with the input data at the same time as the timing of receiving of the output data.

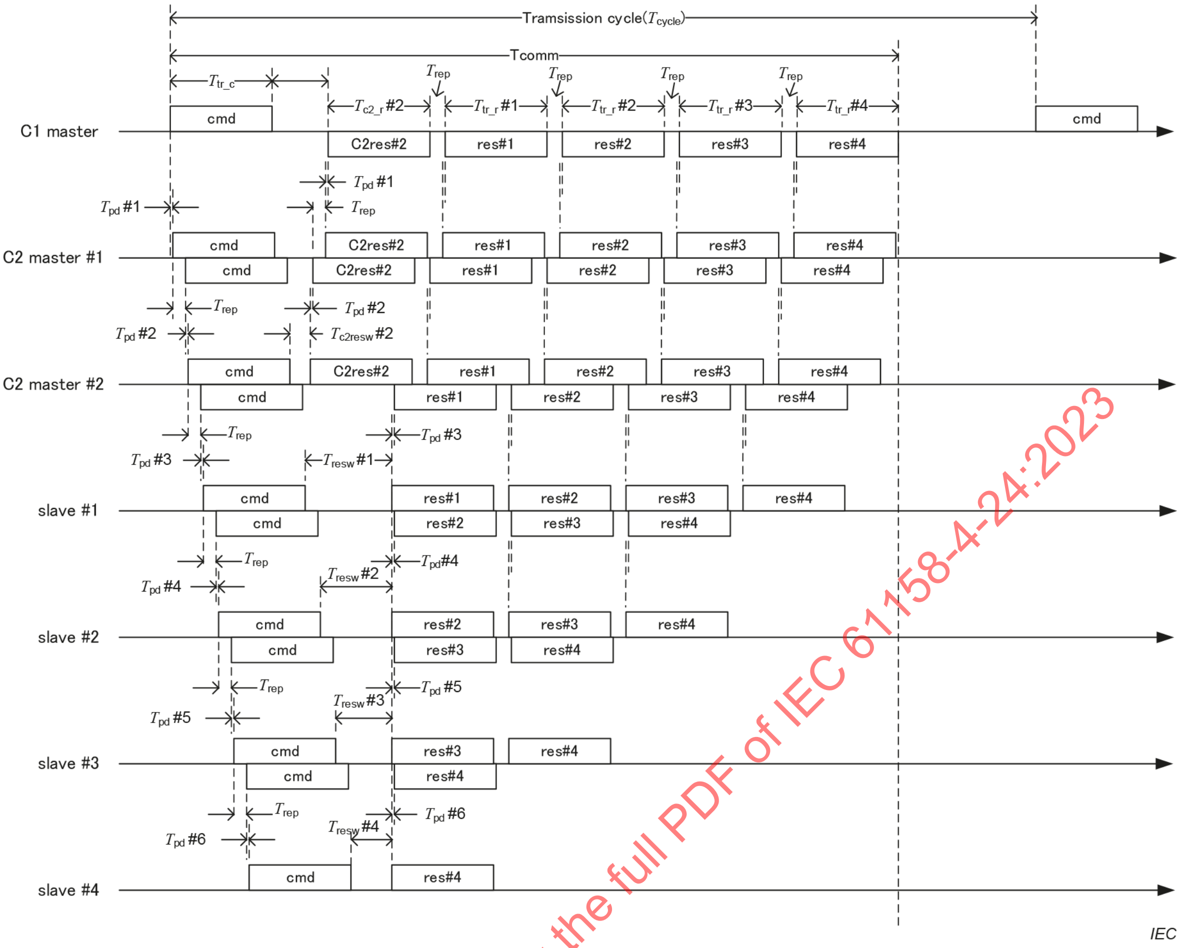


Figure 11 – Schematic diagram of INPUT data response timing at the same time

With Figure 11 as the operating condition, the Formula (16) is given as follows:

$$T_{cycle}=2\times\sum_{n=1}^{N}\left(T_{pd}(\#n)+T_{rep}\times(n-1)\right)-T_{pd}(N)+T_{tr\_c}+T_{C2resw}(\#l)+T_{c2\_r}(\#l)+\sum_{m=1}^{N-L}\left(T_{tr\_r}(\#m)\right) \qquad (16)$$

where

$T_{pd}(\#n)$ is the propagation delay time of the n-th node;

T_{rep} is the output (command) or the input (response) repeating time in the slave;

T_{tr_c} is the output (command) transmission time to send from the C1 master to all slaves;

$T_{c2resw}(\#l)$ is the response waiting time of the l–th C2 master to the output (command) from the C1 master;

$T_{c2\_r}(\#l)$ is the input (response) transmission time from the l–th C2 master to the C1 master;

$T_{tr\_r}(\#m)$ is the input (response) transmission time from the m–th slave to the C1 master.

4.3.2.5 Timing relationship between cyclic transmission and data processing for no time slot type

The timing relationship between cyclic transmission and data processing for no time slot type is as described in 4.3.2.3. However, the C1 master of no time slot type consolidates the output data as one block of data.

4.3.3 Acyclic transmission mode

The acyclic transmission can be used in a system that does not require real-time communication or cyclical data exchange. In the acyclic transmission mode, it is possible to execute the same transmission sequence as a cyclic transmission mode without fixing the transmission cycle. Although the C2 message communication is also possible, the DLS-user shall execute the arbitration of the transmission timing. In the acyclic transmission mode, the data length is fixed at 64 octets.

Since the acyclic transmission does not use the synchronous frame, slaves execute processing of the output data sent by the master and processing of the data to send the input data at its own timing. The slaves do not operate simultaneously.

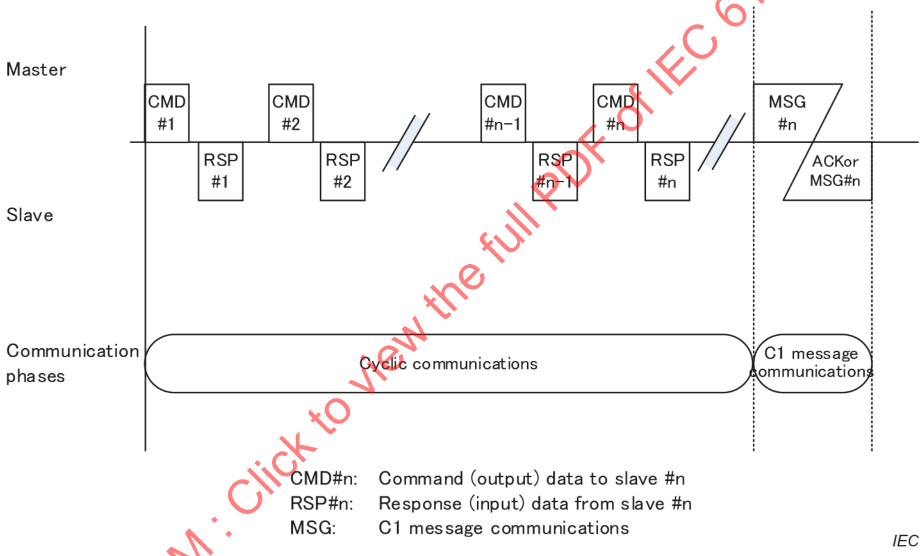


Figure 12 – Timing chart example of acyclic communication

4.4 Service assumed from the PhL

4.4.1 General requirement

There are two types of Ph-service which DLE requires. One is defined in IEC 61158-2, Type 24 and the other is defined in ISO/IEC/IEEE 8802-3:2021, Clause 6.

Both types require the following interface elements:

- Transmit Machine,
- Receive Machine,
- Serializer,
- Deserializer.

4.4.2 DL_Symbols

The PhL Interface Data Units present at the DLL-PhL interface shall be the DL_symbols. The DL_symbol shall have one of the following values:

- ZERO corresponds to a binary "0";
- ONE corresponds to a binary "1".

4.4.3 Assumed primitives of the PhS

The PhS is assumed to provide the following two categories of primitives to the Type 24 DL-protocol.

- Service primitives for transmitting and receiving frames to / from other peer DLEs.
- Service primitives that provide information needed by the local DLE to perform the media access functions.

The assumed primitives of the PhS are grouped into these two categories:

- Transfer of Data to the corresponding DLE
  - PLS_DATA request
  - PLS_DATA indication
- Indication from the local PhL
  - PLS_CARRIER indication
  - PLS_SIGNAL indication
  - PLS_DATA_VALID indication

4.5 Local parameters, variables, counters, timers

4.5.1 Overview

This specification uses the DLS-user request parameters P(…) and local variables V(…) as a means of clarifying the effect of certain actions and the conditions under which those actions are valid, local timers T(…) as a means of monitoring actions of the distributed DLS-provider and of ensuring a local DLE response to the absence of those actions. It also uses local queues Q(…) as a means of ordering certain activities, of clarifying the effects of certain actions, and of clarifying the conditions under which those activities are valid.

Unless otherwise specified, at the moment of their creation or of DLE activation:

- all variables shall be initialized to their default value, or to their minimum permitted value if no default is specified;
- all timers shall be initialized to inactive;
- all queues shall be initialized to empty.

DL-management can change the values of configuration variables.

4.5.2 Variables, parameters, counters and timers to support DLE function

4.5.2.1 V(MA), P(MA)

This variable is used by the CTC to record station address of this station. This variable shall be implemented by all stations. Its range is 1 to $2^{16} - 1$. When the station adopts short format of the DLPDU, the lower 8 bits of this variable is effective. When the station adopts short format II of the DLPDU, the lower 5 bits of this variable are effective.

4.5.2.2 V(Tcycle), P(Tcycle)

This variable is used by the CTC to record transmission period of cyclic communication. This variable shall be implemented by all stations. The value of time depends on V(T_{unit}). Its range is 1 to 64 000.

4.5.2.3 V(Nmax_slaves), P(Nmax_slaves)

This variable is used by the CTC to record the number of connectable slave stations. Its range is 1 to 62.

4.5.2.4 V(Cyc_sel), P(Cyc_sel)

This variable is used by the DLM to record the selection of transmission mode, which is cyclic or acyclic. The value is listed at Table 5.

Table 5 – List of the values of the variable Cyc_sel

Value	Symbol	Description
0	CMode_Cyclic	Cyclic transmission mode
1	CMode_Acyclic	Acyclic transmission mode

4.5.2.5 V(Nmax_dly_cnt), P(Nmax_dly_cnt)

This variable is used by the DLM to record maximum-measurement-count, which limits the number of delay measurement execution. This variable is implemented only by the C1 master adopts configurable time slot. Its default value is 2. Its range is 2 to 31.

4.5.2.6 V(IO_sz), P(IO_sz)

This variable is used by the CTC to hold and designate the data size of send and receive frame, which is transferred in the I/O data exchange band. Its range is 0 to 64. When the CTC adopts fixed-width time slot, its value shall be same for all stations.

4.5.2.7 V(Pkt_sz), P(Pkt_sz)

This variable is used by message segmentation machine to record the message packet size of the cyclic transmission. Its range is 4 to 500. When CTC adopts fixed-width time slot, its value shall be equal to V(IO_sz).

4.5.2.8 V(Nmax_retry), P(Nmax_retry)

This variable is used by the CTC to record maximum-retry-count, which limits the number of retries in the I/O data exchange retry band. This variable is implemented only by the C1 master. Its default value is 0, meaning retry are not permitted. Its range is 0 to 62.

4.5.2.9 V(Tslot), P(Tslot)

This variable is used by the CTC to hold and designate the value of the timeout time for the I/O data exchange to each station. This variable shall be implemented by the C1 master and the C2 master. Its range is 0 to less than V(T_{cycle}). The time that a set value indicates depends on V(T_{unit}). When the CTC adopts fixed-width time slot, its value shall be same for all stations.

4.5.2.10 V(Tunit), P(Tunit)

This variable is used by the CTC to record the unit of a set value of the variable concerning time, when the CTC adopts configurable time slot. Table 6 shows its range. This variable shall be implemented only by the CTC which adopts configurable time slot. The CTC which adopts fixed-width time slot shall not this variable, and shall use 250 ns as time unit for all variables concerning time. The CTC that adopts no time slot shall use 1 ns as time unit for all variables concerning time.

Table 6 – List of the values of the variable Tunit

Value	Definition	Tcycle
0	10 ns	31,25 µs to 500 µs
1	100 ns	More than 500 µs to 4 ms
2	1 µs	More than 4 ms to 64 ms
NOTE Value 0 is default.		

4.5.2.11 V(Tidly), P(Tidly)

This variable is used by the CTC to record the timing of event, which is issued periodically to synchronize the DLS-user. This variable shall be implemented by all stations, which adopt configurable time slot. The value is delay time from the start of cyclic transmission. Its range is 0 to less than V(T_{cycle}). The time that a set value indicates depends on V(T_{unit}).

4.5.2.12 V(Tc2_dly), P(Tc2_dly)

This variable is used by the CTC to record the timing to start the C2 message communication. This variable shall be implemented by the C1 master and the C2 master, which adopt configurable time slot. The value is delay time from the end of cyclic transmission. Its range is 0 to less than V(T_{cycle}). The time that a set value indicates depends on V(T_{unit}).

4.5.2.13 V(Tmsg), P(Tmsg)

This variable is used by the CTC to record the time period for message communication. This variable shall be implemented by the C1 master and the C2 master. Its range is 0 to less than V(T_{cycle}). The time that a set value indicates depends on V(T_{unit}). When the CTC adopts fixed-width time slot, its value shall be same for all stations.

4.5.2.14 V(Twrpt)

This variable is used by the SRC to record the time period for transmission delay measurement. This variable shall be implemented by the C2 master and the slave. Its range is V(T_{slot}) × 3 to less than 64 ms. The default value is 500 µs. The time that a set value indicates depends on V(T_{unit}).

4.5.2.15 V(IO_MAP), P(IO_MAP)

This variable is used by the DLM and the CTC to record the information of the slaves and the C2 master which to be connected in the network. See IEC 61158-3-24, 5.3.2.2.13 for the details.

4.5.2.16 V(Sts_STI), P(Sts_STI)

This variable is used by the DLM to hold the connection status of the stations which to be connected in the network. See IEC 61158-3-24, 5.3.3.2.2.1 for the details.

4.5.2.17 V(Sts_Err), P(Sts_Err)

This variable is used by the DLM to hold the error factor which occurred in the DLE.

4.5.2.18 V(Fc2msg)

This variable is used by the CTC to hold the presence of the C2 message communication band in cyclic transmission mode. This variable shall be implemented by the C1 master and the C2 master. Its value is 0 or 1. The value shall be set to 1 when the C2 message communication band is assigned, and set to 0 when it is not assigned.

4.5.2.19 V(Nslave)

This variable is used by the CTC to hold and designate slave number, which is being processed in the current time slot. This variable shall be implemented by the C1 master and the C2 master. Its range is 0 to V(Nmax_slaves). The value is reset to zero at the start of each cycle of the cyclic transmission. And it is incremented when the I/O data exchange to a slave is executed.

4.5.2.20 V(Nretry)

This variable is used by the CTC to hold and designate the element of retry list, which is being processed in the current time slot. The value is reset to zero at the beginning of each transmission cycle. And it is incremented when the I/O data exchange to a slave is fault, and decremented when the I/O data exchange retry is executed.

4.5.2.21 V(Nrest_slot)

This variable is used by the CTC to hold and designate the number of rest time slot for retry within the end of this cycle. Its range is 0 to V(Nmax_retry). This variable shall be implemented by the C1 master. The value is decremented in the range of V(Nmax_retry) to 0.

4.5.2.22 V(Ndly_cnt)

This variable is used by the DLM to hold and designate the count of executed delay measurement. This variable shall be implemented by the C1 master, which adopt configurable time slot. The value is reset to zero at the start of each cycle of cyclic transmission, and incremented when delay measurement is executed. Its range is 0 to V(Nmax_dly_cnt).

4.5.2.23 V(PDUType)

This variable is used by the CTC and the DLM to hold and designate the selection of the DLPDU type, from Basic format DLPDU. Short format DLPDU, or Short format II. When the DLE adopts each of them, it is not required to implement this variable. The value is listed at Table 7.

Table 7 – List of the values of the variable PDUType

Value	Symbol	Description
0	PDUBasic	Basic format DLPDU
1	PDUShort	Short format DLPDU
2	PDUShort II	Short format II DLPDU

4.5.2.24 V(SlotType)

This variable is used by the CTC and the DLM to hold and designate the selection of time slot type, which is fixed-width or configurable. When the DLE adopts each of them, it is not required to implement this variable. The value is listed at Table 8.

Table 8 – List of the values of the variable SlotType

Value	Symbol	Description
0	TSFixed	Fixed-width time slot
1	TSConfig	Configurable time slot

4.5.2.25 V(Nms_1), V(Nms_2)

This variable is used by the CTC to hold the number of segmentations to be sent in the message communication using basic format of the DLPDU. The suffix "_1" and "_2" show the band of the message communication, and indicate the C1 message communication and the C2 message communication respectively. This variable shall be implemented by all stations which execute message communication. The value is reset to zero at the first segment of each message and it is incremented by one every time CTC received acknowledge of one segment normally from peer station. Its range is 0 to 127.

4.5.2.26 V(Nmr_1), V(Nmr_2)

This variable is used by the CTC to hold the number of segments to be received in message communication using basic format of the DLPDU. The suffix "_1" and "_2" show the band of the message communication, and indicate the C1 message communication and the C2 message communication respectively. This variable shall be implemented by all stations which execute message communication. The value is reset to zero at the first segment of each message. And it is incremented when one segment is received from peer station. Its range is 0 to 127.

4.5.2.27 V(Fmp_1), V(Fmp_2)

This variable is used by the CTC to hold and designate the flag of the polling request in message communication using basic format of the DLPDU. The suffix "_1" and "_2" show the band of the message communication, and indicate the C1 message communication and the C2 message communication respectively. This variable shall be implemented by all stations which execute message communication. Its range is 0 or 1. The value 1 means the polling request and 0 (default) means the other.

4.5.2.28 V(Fmf_1), V(Fmf_2)

This variable is used by the CTC to hold and designate the flag of the last segment in message communication using basic format of the DLPDU. The suffix "_1" and "_2" show the band of message communication, and indicate the C1 message communication and the C2 message communication respectively. This variable shall be implemented by all stations, which execute the message communication. Its range is 0 or 1. The value is reset to zero at the first segment of each message, and set one when the last segment of the message is sent or received.

4.5.2.29 V(Ten)

This variable is used by the DLM to hold the timestamp of delay measurement end when the DLE adopts configurable time slot. The DLE shall implement this variable only when the DLE adopts configurable time slot.

4.5.2.30 V(Tdly)

This variable is used by the DLM to hold the transmission delay measured in CompDly state of the DLM when the DLE adopts configurable time slot. The DLE shall implement this variable only when the DLE adopts configurable time slot.

4.5.2.31 V(Tmax_dly)

This variable is used by the DLM to hold the transmission delay measured in CompDly state of the DLM when the DLE adopts configurable time slot. The DLE shall implement this variable only when the DLE adopts configurable time slot.

4.5.2.32 V(Tst)

This variable is used by the DLM to hold the timestamp of delay measurement start time when the DLE adopts configurable time slot. The DLE shall implement this variable only when the DLE adopts configurable time slot.

4.5.2.33 T(Tcycle)

T(T_{cycle}) is used by the CTC to measure the cyclic transmission period. The value is decremented in the range of V(T_{cycle}) to 0.

4.5.2.34 T(Tslot)

T(T_{slot}) is used by the CTC to measure the time elapsed since last sending a frame. The value is decremented in the range of V(Tslot) to 0.

4.5.2.35 T(Tmsg)

T(T_{msg}) is used by the CTC to measure the time period of message communication band. The value is decremented in the range of V(T_{msg}) to 0.

4.5.2.36 T(Twrpt)

T(T_{wrpt}) is used by the SRC to watch repeat function. The value is decremented in the range of V(Twrpt) to 0.

4.5.2.37 Q(MSGc1s), Q(MSGc2s)

The queue buffer is implemented in message segmentation protocol machine to transfer the send message. The CTC enqueues the message to be sent, then MSM dequeues the message to build the DLPDU and send it.

4.5.2.38 Q(MSGc1r), Q(MSGc2r)

The queue buffer is implemented in message segmentation protocol machine to transfer the received message. MSM builds the message from the received DLPDU and enqueues it, then CTC dequeues the message to transfer it to the DLS-user.

4.5.2.39 V(Port0_UP), P(Port0_UP)

V(Port0_UP) and P(Port0_UP) are used for no time slot type. This is the status to show that the two ports, Port0 and Port1, are either upstream or downstream. The port that receives the first is set as the upstream port. When the Port0 is upstream, the setting of Port0_UP is 1.

5 DLPDU structure

5.1 Overview

5.1.1 Transfer syntax for bit sequences

For transmission across Type 24 DL a bit sequence is reordered into a sequence of octets. Hexadecimal notation is used for octets as specified in ISO/IEC 9899. Let b = $b_0 \ldots b_{n-1}$ be a bit sequence. Denote k as a non-negative integer such that $8(k - 1) \leq n < 8k$. Then b is transferred in k octets assembled as shown in Table 9. The bits b_i, i > n of the highest numbered octet shall be ignored.

Table 9 – Transfer syntax for bit sequences

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

When the DLE implemented over the PHY defined in IEC 61158-2, Type 24, octet 1 is transmitted first and octet k is transmitted last. Hence the bit sequence is transferred as follows across the network:

$$b_0, b_1, ..., b_7, b_8, ..., b_{15}, ...$$

5.1.2 Data type encodings

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

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

is assigned the value

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

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

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

Table 10 – Bit order

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

The Unsigned*n* data types are transferred as specified in Table 10. Unsigned data types as Unsigned1 to Unsigned7 and Unsigned 9 to Unsigned15 will be used too. In this case the next element will start at the first free bit position as denoted in 5.1.1.

5.1.3 Frame format

There are three types of frame format for MECHATROLINK family. One is the basic format and the remaining two are the short format based on ISO/IEC 13239:2002 (HDLC) and the short format II.

5.2 Basic format DLPDU structure

5.2.1 General

5.2.1.1 Frame format

Figure 13 shows the structure of the basic frame format.

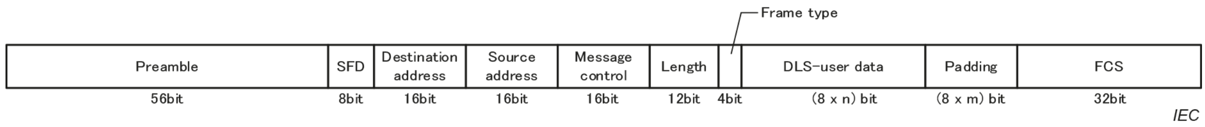


Figure 13 – Basic format DLPDU structure

5.2.1.2 Preamble

The preamble field is identical to ISO/IEC/IEEE 8802-3:2021, Clause 3. The preamble field is a 56-bit field that is used to allow the physical signaling part circuitry to reach its steady state synchronization with the receiving frame timing.

The preamble pattern is:

"10101010 10101010 10101010 10101010 10101010 10101010 10101010."

5.2.1.3 Start frame delimiter (SFD)

The start frame delimiter is identical to ISO/IEC/IEEE 8802-3:2021, Clause 3. The SFD field is the sequence of bit pattern "10101011". It immediately follows the preamble pattern and indicates the start of a frame.

5.2.1.4 Destination address (DA)

The destination address shall contain the node address of the destination DLE.

The higher 8 bits of a 16-bit address is used as an extended address and the lower 8 bits is used as a station address (see Table 11).

For both station addresses and extension address, the usable addresses are specified in Table 12 and Table 13.

Table 11 – Destination and source address format

Octet number	Size	Contents
1	Unsigned8	Station address
2	Unsigned8	Extended address

Table 12 – Station address

Station address	Contents
0x00	Reserved
0x01	C1 master
0x02	C2 master
0x03 to 0xEF	Slave, Gateway
0xF0 to 0xFE	Reserved
0xFF	Broadcast address

Table 13 – Extended address

Extended address	Contents
0x00 to 0xFEh	Device address for the node that has multiple device, or remote address for gateway node
0xFF	Broadcast address[a]
[a] Only the synchronous frame can use broadcast address.	

5.2.1.5 Source address (SA)

The source address shall contain the node address of the source DLE. See 5.2.1.5 for its format and its range.

5.2.1.6 Message control

The message control field is used for sending data with acknowledge service. This field is effective only when the frame type which follows this field is a message frame. This field shall be zero except when the frame type is a message frame followed by a message frame.

This field has two formats. One is Information transfer format and the other is supervisory format (see Table 14 and Table 15). Each field shall have the same function as the field with the same name in ISO/IEC 13239:2002 (HDLC).

NOTE In HDLC and this specification, the order of transmitting the bit is reverse.

Table 14 – Message control field format (Information transfer format)

Octet number	Bit number	Size	Symbol	Contents
1	b_6 to b_0	Unsigned7	N(R)	Transmitting receive sequence number
	b_7	Unsigned1	P/F	Poll bit – primary transmissions Final bit – secondary transmission
2	b_{14} to b_8	Unsigned7	N(S)	Transmitting send sequence number
	b_{15}	Unsigned1	-	Reserved (0)[a]
[a] This bit shall be zero.				

Table 15 – Message control field format (Supervisory format)

Octet number	Bit number	Size	Symbol	Contents
1	b_6.. to b_0	Unsigned7	N(R)	Transmitting receive sequence number
	b_7	Unsigned1	-	Reserved (1)[a]
2	b_{11}.. to b_8	Unsigned4	-	Reserved (0)[b]
	b_{13}.. to b_{12}	Unsigned2	S	Supervisory function bits
	b_{14}	Unsigned1	-	Reserved (0)[b]
	b_{15}	Unsigned1	-	Reserved (1)[a]

[a] This bit shall be one.

[b] This bit shall be zero.

Table 16 – The list of Supervisory function bits

Value	Symbol	Description	Note
0	RR	Receive ready (RR) command or RR response	
1	REJ	Reject (REJ) command or REJ response	
2	RNR	Receive not ready (RNR) response	
3	–	Reserved	

5.2.1.7 Frame type and data length field

The frame type and data length field consist of a higher 4-bit area where the frame type is set and a lower 12-bit area where the data length is set (see Table 17).

The frame type is used for identifying the contents of a frame and also for identifying the type of the protocol for message communication. Table 18 shows the definition of the frame type. The data length indicates the size of the data stored in a frame in terms of the number of octets.

Table 17 – Frame type and data length format

Octet number	Bit number	Size	Contents
1	b_{11}.. to b_0	Unsigned12	Data length (lower 8 bits)
2			Data length (higher 4 bits)
	b_{15}.. to b_{12}	Unsigned4	Frame type

Table 18 – The list of Frame type

Value	Symbol	Description
0	–	Reserved
1	FT_SYNC	Synchronous Frame
2	FT_IO	Output data and Input data frame
3	FT_DLST	Delay measurement start frame
4	FT_DLMS	Delay measurement frame
5	FT_MTKN	Message token frame
6	FT_STS	Status frame
7	FT_CINF	Cyclic information frame
8 to 11	–	Reserved
12	FT_MSG	Message frame
13 to 15	–	Reserved

5.2.1.8 DLS-user data field

The data format varies according to the frame type. When the application data cannot be set in one frame which type is the message frame, it is possible to set and send the data in multiple frames using the message control.

5.2.1.9 Frame check sequence field (FCS)

The frame check sequence (FCS) uses 32-bit CRC-CCITT. The FCS is calculated in the range from the destination address to the end of the data except the FCS itself.

5.2.2 Synchronous frame

The C1 master uses this frame to synchronize the slaves and the C2 master. Only the C1 master can send this frame. The C1 master shall set the station address as the broadcast address (0xFF), and the slaves and the C2 master shall receive this frame.

When the slaves and the C2 master receive this frame, they shall refresh the local clock with the time calculated by adding the transmission delay measured in advance (notified with delay measurement frame) to the current time stored this frame.

Table 19 and Table 20 show detailed the data format of this frame.

Table 19 – Data format of the Synchronous frame

Octet number	Size	Contents
1 to 4	Unsigned32	Timestamp
5 to 6	Unsigned16	Cyclic event delay time
7 to 8	Unsigned16	Reserved

Table 20 – The field list of the Synchronous frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Timestamp	Time stamp of the C1 master at the frame is sent	0 to 2^{32} – 1	–	The unit is defined according to the setting of the variable V(T_{unit}).
Cyclic event delay time	Delay time from the current time stored in this frame to the time when the cyclic event is indicated	0 to 2^{16} – 1	0	The unit is defined according to the setting of the variable V(T_{unit}).

5.2.3 Output data or Input data frame

The C1 master uses this frame for the output data to be sent to slaves, and slaves use this frame for the input data to be sent to the C1 master, within the I/O data exchange band of the cyclic transmission mode. The C2 master can only receive this frame and shall not send this frame.

Either of the destination address or the source address of this frame shall be C1 master because this frame is exchanged between the C1 master and the slaves. The data length shall not be changed during normal operation.

Table 21 and Table 22 show the data format of this frame.

Table 21 – Data format of the Output data or the Input data frame

Octet number	Size	Contents
1 to n	Unsigned8n	Output data or Input data
(n+1) to (n+m)	Unsigned8m	Padding

Table 22 – The field list of the Output data or the Input data frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Output data or Input data	Output data from C1 master to slave or input data from slave to C1 master	0 to maximum value that can be represented with the octet length of the specified data length	–	
Padding	Area adjusted to become multiple in 32 bits in data length	–	0	

5.2.4 Delay measurement start frame

The C1 master uses this frame to specify the targeted station of the delay measurement to measure the transmission delay from the C1 master to each slave or C2 master. Only the C1 master transmits this frame. After the station receives this frame, the station returns the receipt frame till the number of times that specified with this frame.

Table 23 and Table 24 show detailed the data format of this frame.

Table 23 – Data format of Delay measurement start frame

Octet number	Size	Contents
1 to 2	Unsigned16	Measurement number
3 to 4	Unsigned16	Reserved

Table 24 – The field list of Delay measurement start frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Measurement number	Number of times for sending the transmission delay measurement frame	1 to 32	1	

5.2.5 Delay measurement frame

The C1 master uses this frame to measure the transmission delay and to notify the slaves and the C2 master of the result. Only the C1 master transmits this frame. The C1 master shall send this frame to the station to which the delay measurement start frame is sent, and send this frame till the times specified with the delay measurement start frame. The C1 master shall notify of the result of the measurement by using this frame.

The slaves and the C2 master that received delay measurement start frame shall receive and return this frame. They shall stop returning the receipt frame after receiving this frame till the number of times notified with the delay measurement start frame.

Table 25 and Table 26 show detailed the data format of this frame.

Table 25 – Data format of Delay measurement frame

Octet number	Size	Contents
1 to 4	Unsigned32	Timestamp
5 to 6	Unsigned16	Transmission delay
7 to 8	Unsigned16	Reserved

Table 26 – The field list of Delay measurement frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Transmission delay	Time from sending a frame by C1 master to reception of it by slave or C2 master	0 to $2^{16} - 1$	–	The unit is defined according to the setting of the time unit. The default unit is 10 ns.

5.2.6 Message token frame

The C1 master sends this frame to notify the C2 to permit for starting the C2 message communication before the C2 message communicationstart time. The C1 master shall send this frame only when the C1 master exchanges the command data/response data with all the slaves and the C1 message communication ends before the C2 message communication start time. However, the C1 master shall not send this frame when the time from the current time of the local clock to the start time of the C2 message communication is shorter than the transmission delay between the C1 master and the C2 master.

The C2 master that receives this frame starts the C2 message communication unless the C2 message communication start time has come.

Only the C1 master can send this frame. The destination and the source addresses are fixed because only the C2 master can receive this frame. This frame contains no data.

5.2.7 Status frame

The C1 master uses this frame to inquire the status of the slaves and the C2 master. The station that received this frame from the C1 master shall send this frame to notify of the current status. The slaves and the C2 master shall return this frame also when receiving a cycle information frame contains their own address as the destination. The slave and the C2 master can return this frame to request the C1 master to execute the transmission delay measurement even when they receive the other frame than the synchronous frame from the C1 master.

Broadcast address or multicast address shall not be specified as the destination of this frame.

Table 27 to Table 30 show detailed the data format of this frame.

Table 27 – Data format of Status frame

Octet number	Size	Contents
1 to 2	Unsigned16	Status
3 to 4	Unsigned16	Repeater status

Table 28 – The field list of Status frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Status	Current value of the DLE status	(See Table 29)	–	Refreshed by slave or C2 master
Repeater status	Current value of repeater status	(See Table 30)	–	Refreshed by slave or C2 master

Table 29 – The list of the DLE status

Status code	Description
0x0000	Station does not exist
0x0002	Waiting for the parameters setting by the DLS-user
0x0021	Searching for a packet with the duplicate Node_ID
0x0024	Waiting for the delay measurement request by the DLS-user
0x0025	Waiting for delay measurement start indication from C1 master
0x0026	Measuring transmission delay
0x0027	Waiting for cyclic information frame from C1 master
0x0030	Waiting for communication start request by the DLS-user
0x0031	Operating in cyclic transmission mode
0x0060	Operating in acyclic transmission mode

Table 30 – The list of Repeater status

Send status			
bit	**Symbol**	**Description**	**Initial**
b_0	–	Reserved	0
b_1	MII_RXTXE	Send request detected while receiving data from PhL	0
b_2	MII_TXRXE	Receive data from PhL detected while sending data to PhL	0
b_3	MII_RXRXE	Receive data from PhL detected while receiving data from PhL	0
b_4 to b_{15}	–	Reserved	0

5.2.8 Cycle Information frame

The C1 master uses this frame to notify the slave and the C2 master of the transmission mode. Only the C1 master can send this frame.

The C1 master can broadcast this frame or send it to the slaves or the C2 master individually. The slaves and the C2 master shall return a status frame when they receive this frame that contains their own address as the destination.

Table 31 and Table 32 show detailed the data format of this frame.

Table 31 – Data format of Delay measurement frame

Octet number	Size	Contents
1 to 2	Unsigned16	Transmission cycle
3 to 4	Unsigned16	C2 message delay
5 to 6	Unsigned16	Maximum delay
7	Unsigned8	Communication mode
8	Unsigned8	Time unit

Table 32 – The field list of Cycle Information frame

Field	Contents	Maximum and minimum value	Note
Transmission cycle	Transmission cycle of the cyclic transmission mode	3 125 to 64 000	[a]
C2 message delay	Delay time from the current time stored in the synchronous frame to the time of C2 message communication start	0 to $2^{16}-1$	
Maximum delay	Maximum value among the transmission delay measured by the C1 master.	0 to 2^{16}-1	[a]
Communication mode	Selection of transmission mode to be executed after initialization	0: Cyclic 1: Acyclic	
Time unit	Selection code of time unit	0: 10 ns 1: 100 ns 2: 1 μs	[b]

[a] The unit is defined by the value of the time unit field in this frame.

[b] When the transmission cycle Tcycle is 31,25 μs ≤ T_MCYC <= 500 μs, set to 10 ns.
In the case of 500 μs < T_MCYC <= 4 ms, set to 100 ns.
When 4 ms < T_MCYC <= 64 ms, set to 1 μs.

5.2.9 Message frame

This frame is used for message transmission. All stations can send this frame. When the DLSDU that requested to send is beyond the size that can be contained within one frame, the DLSDU shall be divided by using the message control field in this frame.

Table 33 and Table 34 show detailed the data format of this frame.

Table 33 – Data format of Message frame

Octet number	Size	Contents
1 to n	Unsigned8n	Message data
(n+1) to (n+m)	Unsigned8m	Padding

Table 34 – The field list of Message frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Message data	Arbitrary data except output data or input data	0 to maximum value that can be represented with the octet length of the specified data length	–	
Padding	Area adjusted to become multiple in 32 bits in data length	–	0	

5.3 Short format DLPDU structure

5.3.1 General

5.3.1.1 Short frame format

The short format DLPDU is shown in Figure 14. Transmission sequence shall be from left to right as shown in Figure 14, i.e. station address first, followed by control, DLS-user data and finally CRC. See 5.2.1.7 for frame type and data length field.

Preamble	Start flag	Station address	Control	DLS-user data	CRC	End flag
16 bit	8 bit	8 bit	8 bit	(8 x n) bit	16 bit	8 bit

IEC

Figure 14 – Short format DLPDU structure

5.3.1.2 Preamble

The preamble field is identical to ISO/IEC 13239:2002 (HDLC). The preamble field is a 16-bits field that is used to allow the physical signaling part circuitry to reach its steady state synchronization with the receiving frame timing.

The preamble pattern is:

"10101010 10101010."

5.3.1.3 Start flag

The start flag is identical to ISO/IEC 13239:2002 (HDLC). This field is the sequence of bit pattern "01111110". It immediately follows the preamble pattern and indicates the start of a frame.

Due to bit-stuffing, this bit sequence is prevented from occurring inside the DLSDU sequence. The DLE shall only accept a received signal burst as a DLPDU after verifying this sequence and shall remove this sequence before transferring the DLSDU sequence to the DLS-user.

5.3.1.4 Station address field

Table 35 shows the range of the station address field.

Table 35 – Range of Station address field

Station Address	Contents
0x00	Reserved
0x01	C1 master
0x02	C2 master
0x03 to 0xDE	Slaves
0xDF	Not connected (default)
0xE0 to 0xFE	Reserved
0xFF	Broadcast address

5.3.1.5 Control field (CTL)

The Control field is used to identify the frame type. This field has two formats. One is an I/O data exchange format and the other is a Message format. Table 36 to Table 38 show detailed this field.

Table 36 – Control field format (I/O data exchange format)

Octet number	Bit number	Size	Symbol	Contents
1	b_3.. to b_0	Unsigned4	CMD	Input data or Output data flag; 1: Input data (response) 3: Output data (command) 8: Synchronous frame
	b_7.. to b_4	Unsigned4	-	Reserved (0)[a]
[a] This bit shall be zero.				

Table 37 – Control field format (Message format)

Octet number	Bit number	Size	Symbol	Contents
1	b_3.. to b_0	Unsigned4	S(n)	Sequence number
	b_4	Unsigned1	-	Reserved (1)[a]
	b_5	Unsigned1	C1/C2	Primary node flag
	b_6	Unsigned1	END	End flag
	b_7	Unsigned1	S/D	Sync / data flag
[a] This bit shall be one.				

Table 38 – The field list of Message format

Symbol	Description
S(n)	Transmitting send or receive sequence number.
C1/C2	0: C1 master 1: C2 master
END	When S/D is 0, 0: Data transfer from primary node to secondary node 1: Data transfer from secondary node to primary node When S/D is 1, 0: Following data exists 1: This data is last data
S/D	0: Handshake frame 1: Data frame

5.3.1.6 DLS-user data field

The length of the DLS-user data field is from 8 octets to 64 octets.

5.3.1.7 CRC field

The CRC field uses a 16-bit CRC-CCITT. The CRC is calculated based on the following formula.

$$P(x)=X^{16}+X^{12}+X^{5}+1 \quad (17)$$

5.3.1.8 End flag

The sequence of symbols in this field is identical to the start flag (see 5.3.1.3).

5.3.2 Synchronous frame

The C1 master uses this frame to synchronize the slaves and the C2 master. Only the C1 master can send this frame. The C1 master shall set the station address to the broadcast address (0xFF), and the slaves and the C2 master shall receive this frame. When they receive this frame, they shall check the CRC field contained in this frame, and then issue a cyclic event if the result of the check has no error.

Table 39 and Table 40 show detailed the data format of this frame.

Table 39 – Data format of the Synchronous frame

Octet number	Size	Contents
1 to 2	Unsigned16	Transmission cycle
3 to 4	Unsigned16	Time slot width
5 to 16 or 31	-	Reserved

Table 40 – The field list of the Synchronous frame

Field	Contents	Maximum and minimum value	Note
Transmission cycle	Transmission cycle notified C2 master	0 to $2^{16}-1$	The unit is 0,25 μs
Time slot width	Time slot	0 to $2^{16}-1$	The unit is 0,25 μs

5.3.3 Output data or Input data frame

5.3.3.1 Output data frame

The C1 master uses this frame for the output data to be sent to slave within the I/O data exchange band of cyclic transmission mode. The C2 master can only receive this frame. The length of the output data shall be 16 or 31 octets.

Table 41 and Table 42 show detailed the data format of this frame.

Table 41 – Data format of the Output data frame

Octet number	Size	Contents
1 to 16 or 31	Unsigned8n	Output data

Table 42 – The field list of the Output data frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Output data	Output data from C1 master to slave	0 to Maximum value according to the data length	–	

5.3.3.2 Input data frame

Slaves use this frame for the input data to be sent to C1 master within the I/O data exchange band of cyclic transmission mode. The C2 master can only receive this frame. The length of the input data shall be 16 or 31 octets.

Table 43 and Table 44 show detailed the data format of this frame.

Table 43 – Data format of the Input data frame

Octet number	Size	Contents
1 to 16 or 31	Unsigned8n	Input data

Table 44 – The field list of the Input data frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Input data	Input data from slave to C1 master	0 to Maximum value according to the data length	–	

5.3.4 Message frame

This frame is used for message transmission. All stations can send this frame. When the DLSDU that requested to send is beyond the size that can be contained within one frame, the DLSDU shall be divided by using the message control field in this frame.

5.4 Short format II DLPDU structure

5.4.1 General

5.4.1.1 Short frame fortmat II

The short format II DLPDU is shown in Figure 15. Transmission sequence shall be from left to right as shown in Figure 15, i.e. station address first, followed by Transmission cycle counter, DLS-user data and finally CRC and DLS-user data can be omitted.

Preamble	Start flag	Station address	Transmission cycle counter	DLS-user data	CRC	End flag
4 or 16 bit	8 bit	8 bit	8 bit	(8 x n) bit	16 bit	8 bit

IEC

Figure 15 – Short format II DLPDU structure

5.4.1.2 Preamble (PA)

The preamble field is identical to ISO/IEC 13239:2002 (HDLC). The preamble field is a 4-bits or 16-bits field that is used to allow the physical signaling part circuitry to reach its steady state synchronization with the receiving frame timing.

The preamble pattern is:

"1010." or "10101010 10101010."

5.4.1.3 Start flag (SF)

The start flag is identical to ISO/IEC 13239:2002 (HDLC). This field is the sequence of bit pattern "01111110". It immediately follows the preamble pattern and indicates the start of a frame.

Due to bit-stuffing, this bit sequence is prevented from occurring inside the DLSDU sequence. The DLE shall only accept a received signal burst as a DLPDU after verifying this sequence and shall remove this sequence before transferring the DLSDU sequence to the DLS-user.

5.4.1.4 Station address field (SA)

Station address field is as described below. Figure 16 shows the station address field for acyclic transmission mode. As the station address field for acyclic transmission mode uses Bit4-0, Bit6-5 and Bit7 are reserved. Figure 17 shows the station address field for cyclic transmission mode. As the station address field for cyclic transmission mode uses Bit4-0 and Bit7, Bit6-5 is reserved. Bit7 is the Bit to identify whether the slave responds with the input data corresponding to the output data or it returns the input data.

Bit	7	6	5	4	3	2	1	0
	Reserved			A4	A3	A2	A1	A0

IEC

Figure 16 – Acyclic transmission frame address field

Bit	7	6	5	4	3	2	1	0
	A7	Reserved		A4	A3	A2	A1	A0

IEC

Figure 17 – Cyclic transmission frame address

Table 45 shows the range of the station address field.

Table 45 – Range of Station address field

Station Address	Contents
0x00	Common address
0x01 to 0x0F	Reserved
0x1F	Broadcast address
0x10 to 0x1E	C2 master, Slaves

5.4.1.5 Transmission cycle counter field (Tcc)

Transmission cycle counter (Tcc) field is used to indicate the timing of response by slaves. Table 46 shows the cycle scale counter field format.

Table 46 – Cycle scale counter field format

Octet number	Bit number	Size	Symbol	Contents
1	b_7.. to b_0	Unsigned8	CMD	Counter value that changes every transmission cycle. When this counter value matches the value of Tcc_mul and Tcc_pos (see IEC61158-3-24, 5.3.2.2.2, Table14) of each slave, the slave makes a response.

5.4.1.6 DLS-user data field

Octet number to indicate the DLS-user data field length is from 0 to 15.

5.4.1.7 CRC field

The CRC field uses a 16-bit CRC-CCITT. The CRC is calculated based on the following formula.

$$P(x)=X^{16}+X^{12}+X^5+1 \qquad (18)$$

5.4.1.8 End flag

The sequence of symbols in this field is identical to the start flag (see 5.3.1.3).

5.4.1.9 Frame type (FT)

There are three types of this frame type according to the command frame length. The frame length is determined by the number of bytes of SA, Tcc, DLS and CRC. Table 47 shows the list of frame types.

Table 47 – The list of frame type

Value	Symbol	Command	Description
0	FT_ASYNC	19 bytes	Asynchronous frame
1	FT_SYNC	4 bytes	Synchronous frame
2	FT_IO	6 to 20 bytes	Output data and Input data frame Excluding 19 bytes.
3 to 15	–	–	Reserved

5.4.2 Asynchronous frame

Figure 18 shows the asynchronous frame DLPDU format.

Preamble	Start flag	Station address	DLS-user data	CRC	End flag
4 or 16 bit	8 bit	8 bit	128 bit	16 bit	8 bit

IEC

Figure 18 – Asynchronous frame

5.4.3 Synchronous frame

Figure 19 shows the synchronous frame DLPDU format to be used by C1. Figure 20 shows the synchronous frame DLPDU format to be used by C2 or slave.

Preamble	Start flag	Station address	Transmission cycle counter	DLS–user data	CRC	End flag
4 or 16 bit	8 bit	8 bit	8 bit	(8 x n) bit	16 bit	8 bit

IEC

Figure 19 – Synchronous frame (to be used by C1)

Preamble	Start flag	Station address	DLS–user data	CRC	End flag
4 or 16 bit	8 bit	8 bit	(8 x n) bit	16 bit	8 bit

IEC

Figure 20 – Synchronous frame (to be used by C2 or slave)

5.4.4 Output data or Input data frame

5.4.4.1 Output data frame

Table 48 and Table 49 show detailed data format of this frame.

The octet number indicating the DLS-user data length of this output data frame is 0 to 15 when sending individual address. When sending the common address, the output data can be omitted.

Table 48 – Data format of the Output data frame

Octet number	Size	Contents
0 to 15	Unsigned8 x n	Output data

Table 49 – The field list of the Output data frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Output data	Output data from C1 master to C2 master and slave	0 to Maximum value according to the data length	–	

5.4.4.2 Input data frame

Table 50 and Table 51 show detailed the data format of this frame.

The octet number indicating the input data length of the input data frame is 0 to 15.

Table 50 – Data format of the Input data frame

Octet number	Size	Contents
0 to 15	Unsigned8 x n	Input data

Table 51 – The field list of the Input data frame

Field	Contents	Maximum and minimum value	Recommended value	Note
Input data	Input data from C2 master and slave to C1 master	0 to Maximum value according to the data length	–	

6 DLE element procedure

6.1 Overview

In following subclause, the operation of CTC and SRC is described. Although DL-management is a component of DLE, it is described in the next clause.

6.2 Cyclic transmission control sublayer

6.2.1 General

CTC provides the following with:

- Control of communication sequence and communication band (C1 master);
- Monitoring of all the I/O data exchanged between the C1 master and the slaves (C2 master);
- Sending response as a server (Slave);
- Message communication in cyclic communication;
- Sporadic data exchange.

6.2.2 DLS-user interface

6.2.2.1 Primitive definitions

The primitives exchanged between the DLS-user and CTC are shown below. Table 52 shows the primitives and parameters issued by the DLS-user and Table 53 shows the primitives and parameters issued by CTC..

Table 52 – Primitives and parameters for the DLS-user interface issued by the DLS-user

Primitive	Source	Parameter	Function
DL-WRITE-DATA.req	DLS-user	SAP_ID, DLS-user-data	Send data writing request
DL-READ-DATA.req	DLS-user	SAP_ID	Receive data reading request
DL-SDA.req	DLS-user	SAP_ID, Node_ID, Length, DLS-user-data	Send data request with transmission confirmation
DL-SDN.req	DLS-user	SAP_ID, Node_ID, Length, DLS-user-data	Send data request without transmission confirmation
DL_GET_STATUS.req	DLS-user	Sts_ID	DL parameter referring request

Table 53 – Primitives and parameters for the DLS-user interface issued by the CTC

Primitive	Source	Parameter
DL-WRITE-DATA.cnf	CTC	Result
DL-READ-DATA.cnf	CTC	Result, DLS-user-data
DL-SDA.ind	CTC	Node_ID, Length, DLS-user-data
DL-SDA.cnf	CTC	Result
DL-SDN.ind	CTC	Node_ID, Length, DLS-user-data
DL-SDN.cnf	CTC	Result
DL_GET_STATUS.cnf	CTC	Result, Cur_Status
DL_EVENT.ind	CTC	Event_ID

6.2.2.2 Overview of the interactions

See IEC 61158-3-24, 4.3.

6.2.2.3 Detailed definitions of the primitives and parameters

See IEC 61158-3-24, 4.4.

6.2.3 Protocol machines in CTC

6.2.3.1 Overview

There are three protocol machines in CTC:

- Cyclic transmission protocol machine.
- Message segmentation protocol machine.
- Acyclic transmission protocol machine.

The cyclic transmission protocol machine has three kinds of protocol machines corresponding to the adopted frame format. These protocol machines are available as time slot type or no time slot type. First, protocol machines of time slot type are available in the following two patterns. One is "fixed-width time slot type" whose transmission sequence is composed of same-width time slot, and the other is "configurable time slot type" whose transmission sequence for each station is different time slot. In addition, each of two protocol machines of "fixed-width time slot type" and "fixed-width time slot type" has three kinds of protocol machines corresponding to the station type (the C1 master, the C2 master, and the slave).

Next, protocol machines of no time slot type have three kinds of protocol machines corresponding to the station type (the C1 master, the C2 master, and the slave).

The cyclic communication protocol machine has two protocol machine, sender and receiver.

The message communication protocol machine works only while the cyclic transmission protocol machine is active.

6.2.3.2 Cyclic transmission protocol machine

6.2.3.2.1 Protocol machine for cyclic communication consists of fixed-width time slot

6.2.3.2.1.1 C1 master

Figure 21 and Table 54 show the state diagram and the state table of the C1 master that adopts fixed-width time slot.

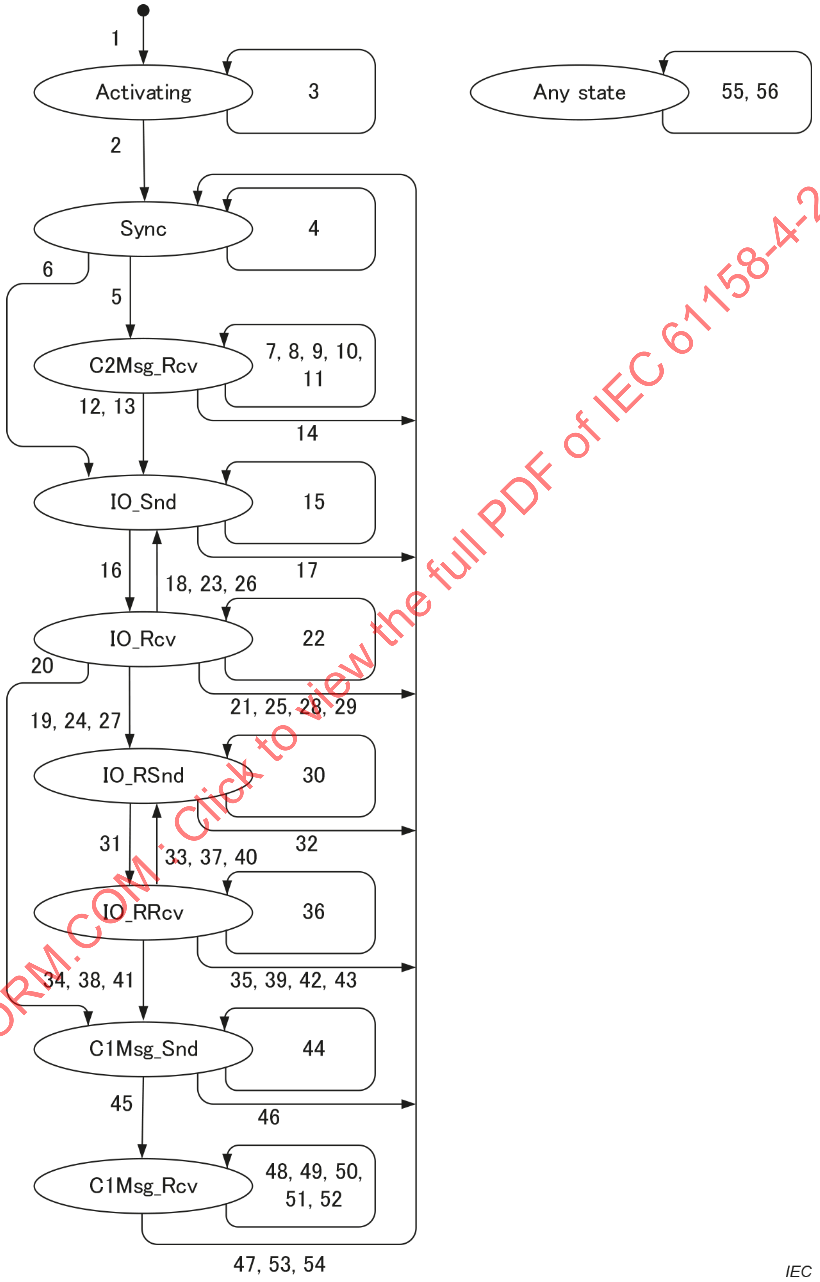


Figure 21 – The state diagram of the C1 master for fixed-width time slot

Table 54 – The state table of the C1 master for fixed-width time slot

No.	Current state	Event /condition =>action	Next state
1	Any state	Power on or CTC_Reset.req => (none)	Activating
2	Activating	CTC_Start.req => START_TIMER(T(Tcycle), V(Tcycle)) CTC_Start.cnf { OK }	Sync
3	Activating	CTC_Start.req => CTC_Start.cnf { NG }	Activating
4	Sync	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } SRCSDU:= BUILD_PDU_SYNC() Node_ID:= GET_DA(SRCSDU) Length:= GET_LEN(SRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU} START_TIMER(T(Tslot), V(Tslot)) V(Nslave):= 1 V(Nretry):= 0 V(Nrest_slot):= V(Nmax_retry)	Sync
5	Sync	SRC_Send_Frame.cnf { Result } / V(Fc2msg) = "True" => (none)	C2Msg_Rcv
6	Sync	SRC_Send_Frame.cnf { Result } / V(Fc2msg) = "False" => (none)	IO_Snd
7	C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec2msg)) => EXEC_MSPM_R(Ec2msg, Rcv_sts, RcvSRCSDU, SndSRCSDU) Node_ID:= GET_DA(SndSRCSDU) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	C2Msg_Rcv
8	C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec1msg)) => (none)	C2Msg_Rcv
9	C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) <> FT_MSG)) => (none)	C2Msg_Rcv
10	C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) <> V(MA))) => (none)	C2Msg_Rcv

No.	Current state	Event /condition =>action	Next state
11	C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / (Rcv_sts <> OK) => (none)	C2Msg_Rcv
12	C2Msg_Rcv	SRC_Send_Frame.cnf { Result } => (none)	IO_Snd
13	C2Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" => SRCSDU:= BUILD_PDU_OUT(V(Nslave)) Node_ID:= GET_DA(SndSRCSDU) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	IO_Snd
14	C2Msg_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot))	Sync
15	IO_Snd	EXPIRED_TIMER (T(Tslot)) = "True" => SRCSDU:= BUILD_PDU_OUT(V(Nslave)) Node_ID:= GET_DA(SndSRCSDU) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	IO_Snd
16	IO_Snd	SRC_Send_Frame.cnf { } => (none)	IO_Rcv
17	IO_Snd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot))	Sync
18	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = GET_NODE_ID(V(Nslave)) && (V(Nslave) < V(Nmax_slave)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) V(Nslave):= V(Nslave) + 1	IO_Snd
19	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = GET_NODE_ID(V(Nslave)) && (V(Nslave) >= V(Nmax_slave) && (V(Nrest_slot) > 0) && (V(Nretry) > 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	IO_RSnd
20	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = GET_NODE_ID(V(Nslave)) && (V(Nslave) >= V(Nmax_slave) && (V(Nrest_slot) > 0) && (V(Nretry) = 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	C1Msg_Snd
21	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = GET_NODE_ID(V(Nslave)) && (V(Nslave) >= V(Nmax_slave) && (V(Nrest_slot) <= 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	Sync

No.	Current state	Event /condition =>action	Next state
22	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) <> GET_NODE_ID(V(Nslave))) => (none)	IO_Rcv
23	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nslave) < V(Nmax_slave)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) PUT_RETRY_LIST(V(Nslave)) V(Nslave):= V(Nslave) + 1	IO_Snd
24	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nslave) >= V(Nmax_slave) && (V(Nrest_slot) > 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) PUT_RETRY_LIST(V(Nslave)) V(Nslave):= V(Nslave) + 1	IO_RSnd
25	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nslave) >= V(Nmax_slave) && (V(Nrest_slot) <= 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) PUT_RETRY_LIST(V(Nslave)) V(Nslave):= V(Nslave) + 1	Sync

26	IO_Rcv	EXPIRED_TIMER (T(Tslot)) = "True"
		/ V(Nslave) < V(Nmax_slave)
		=> STORE_PDU_IN(V(Nslave), SRCSDU, NG)
		If V(Nrest_slot) > 0 then
		PUT_RETRY_LIST(V(Nslave))
		endif
		V(Nslave):= V(Nslave) + 1
			IO_Snd

27	IO_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) > 0) => STORE_PDU_IN(V(Nslave), SRCSDU, NG) PUT_RETRY_LIST(V(Nslave)) V(Nslave):= GET_RETRY_LIST() SRCSDU:= BUILD_PDU_OUT(V(Nslave)) Node_ID:= GET_DA(SRCSDU) Length:= GET_LEN(SRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	IO_RSnd
28	IO_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) = 0) => STORE_PDU_IN(V(Nslave), SRCSDU, NG)	Sync
29	IO_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot) STORE_PDU_IN(V(Nslave), SRCSDU, NG)	Sync

No.	Current state	Event /condition =>action	Next state
30	IO_RSnd	EXPIRED_TIMER (T(Tslot)) = "True" => V(Nslave):= GET_RETRY_LIST() SRCSDU:= BUILD_PDU_OUT(V(Nslave)) Node_ID:= GET_DA(SRCSDU) Length:= GET_LEN(SRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	IO_RSnd
31	IO_RSnd	SRC_Send_Frame.cnf { } => (none)	IO_RRcv
32	IO_RSnd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)	Sync
33	IO_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = GET_NODE_ID(V(Nslave)) && (V(Nrest_slot) >= 1) && (V(Nretry) > 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) V(Nrest_slot):= V(Nrest_slot) – 1	IO_RSnd
34	IO_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = GET_NODE_ID(V(Nslave)) && (V(Nrest_slot) >= 1) && (V(Nretry) <= 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	C1Msg_Snd
35	IO_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = GET_NODE_ID(V(Nslave)) && (V(Nrest_slot) < 1)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	Sync
36	IO_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) <> GET_NODE_ID(V(Nslave))) => (none)	IO_RRcv
37	IO_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nrest_slot) >= 1) && (V(Nretry) > 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) V(Nrest_slot):= V(Nrest_slot) – 1	IO_RSnd
38	IO_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nrest_slot) >= 1) && (V(Nretry) <= 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) V(Nrest_slot):= V(Nrest_slot) – 1	C1Msg_Snd
39	IO_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nrest_slot) < 1)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	Sync

No.	Current state	Event /condition =>action	Next state
40	IO_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" / ((V(Nrest_slot) >= 1) && (V(Nretry) > 0)) => STORE_PDU_IN(V(Nslave), None, TOUT) V(Nrest_slot):= V(Nrest_slot) – 1 V(Nslave):= GET_RETRY_LIST() SRCSDU:= BUILD_PDU_OUT(V(Nslave)) Node_ID:= GET_DA(SRCSDU) Length:= GET_LEN(SRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	IO_RSnd
41	IO_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" / ((V(Nrest_slot) >= 1) && (V(Nretry) <= 0)) => STORE_PDU_IN(V(Nslave), None, TOUT) V(Nrest_slot):= V(Nrest_slot) – 1 EXEC_MSPM_IS(Ec1msg, SRCSDU) Node_ID:= GET_DA(SRCSDU) Length:= GET_LEN(SRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	C1Msg_Snd
42	IO_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nrest_slot) < 1) => STORE_PDU_IN(V(Nslave), None, TOUT)	Sync
43	IO_RRcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot) STORE_PDU_IN(V(Nslave), SRCSDU, TOUT)	Sync
44	C1Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" => EXEC_MSPM_IS(Ec1msg, SndSRCSDU) Node_ID:= GET_DA(SndSRCSDU) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU}	C1Msg_Snd
45	C1Msg_Snd	SRC_Send_Frame.cnf { } => (none)	C1Msg_Rcv
46	C1Msg_Snd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)	Sync
47	C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, RcvSRCSDU} / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec1msg)) => STOP_TIMER(T(Tslot)) EXEC_MSPM_IR(E1msg, Rcv_sts, RcvSRCSDU)	Sync
48	C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, RcvSRCSDU} / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec2msg)) => (none)	C1Msg_Rcv

No.	Current state	Event /condition =>action	Next state
49	C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, RcvSRCSDU} / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec2msg)) => (none)	C1Msg_Rcv
50	C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, RcvSRCSDU} / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) <> FT_MSG)) => (none)	C1Msg_Rcv
51	C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, RcvSRCSDU} / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) <> V(MA))) => (none)	C1Msg_Rcv
52	C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, RcvSRCSDU} / ((Rcv_sts <> OK) => (none)	C1Msg_Rcv
53	C1Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" => STOP_TIMER(T(Tslot))	Sync
54	C1Msg_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot))	Sync
55	Any state	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	Same state
56	Any state	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	Same state

6.2.3.2.1.2 C2 master

Figure 22 and Table 55 show the state diagram and the state table of the C2 master that adopts fixed-width time slot.

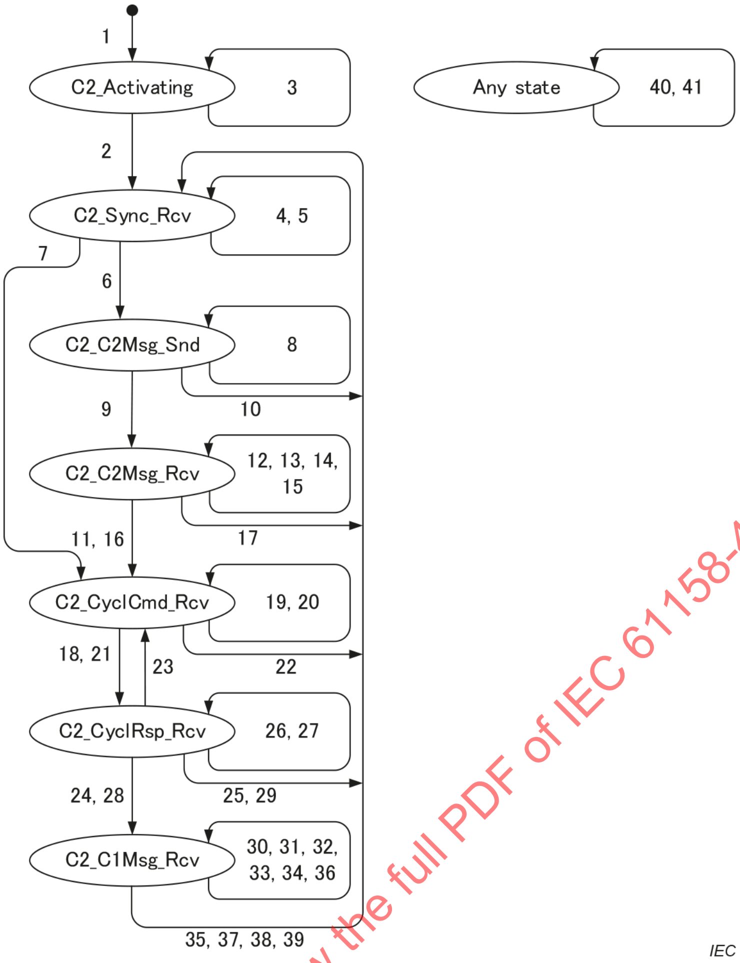


Figure 22 – The state diagram of the C2 master for fixed-width time slot

Table 55 – The state table of the C2 master for fixed-width time slot

No.	Current state	Event /condition =>action	Next state
1	Any state	Power on or CTC_Reset.req => (none)	C2_Activating
2	C2_Activating	CTC_Start.req => START_TIMER(T(Tcycle), V(Tcycle)) CTC_Start.cnf { OK }	C2_Sync_Rcv
3	C2_Activating	CTC_Start.req => CTC_Start.cnf { NG }	C2_Activating
4	C2_Sync_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) <> Cgaddr)) => (none)	C2_Sync_Rcv
5	C2_Sync_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / (Rcv_sts <> OK) => (none)	C2_Sync_Rcv

No.	Current state	Event /condition =>action	Next state
6	C2_Sync_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" / V(Fc2msg) = "True" => DL_Event.ind {DL_Ev_Tcycle } START_TIMER(T(Tslot), V(Tslot)) V(Nslave):= 1	C2_C2Msg_Snd
7	C2_Sync_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" / V(Fc2msg) = "False" => DL_Event.ind {DL_Ev_Tcycle } START_TIMER(T(Tslot), V(Tslot)) V(Nslave):= 1	C2_Out_Rcv
8	C2_C2Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" => EXEC_MSPM_IS(Ec2msg, SndSRCSDU) Node_ID:= GET_DA(SndSRCSDU) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU}	C2_C2Msg_Snd
9	C2_C2Msg_Snd	SRC_Send_Frame.cnf { } => (none)	C2_C2Msg_Rcv
10	C2_C2Msg_Snd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)	C2_Sync_Rcv
11	C2_C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec2msg)) => EXEC_MSPM_IR(E1msg, Rcv_sts, RcvSRCSDU)	C2_Out_Rcv
12	C2_C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec1msg)) => (none)	C2_C2Msg_Rcv
13	C2_C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) <> FT_MSG)) => (none)	C2_C2Msg_Rcv
14	C2_C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) <> V(MA))) => (none)	C2_C2Msg_Rcv
15	C2_C2Msg_Rcv	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / (Rcv_sts <> OK) => (none)	C2_C2Msg_Rcv
16	C2_C2Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" => (none)	C2_Out_Rcv
17	C2_C2Msg_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot))	C2_Sync_Rcv

No.	Current state	Event /condition =>action	Next state
18	C2_Out_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) = FT_OUT)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	C2_In_Rcv
19	C2_Out_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) <> FT_OUT)) => (none)	C2_Out_Rcv
20	C2_Out_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / (Rcv_sts <> OK) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	C2_Out_Rcv
21	C2_Out_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" => (none)	C2_In_Rcv
22	C2_Out_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot))	C2_Sync_Rcv
23	C2_In_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) = FT_IN) && (V(Nslave) < V(Nmax_slave))) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) V(Nslave):= V(Nslave) + 1	C2_Out_Rcv
24	C2_In_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) = FT_IN) && (V(Nslave) >= V(Nmax_slave)) && (V(Nmax_retry) > 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) V(Nslave):= V(Nslave) + 1 V(Nretry):= V(Nmax_retry)	C2_C1Msg_Rcv
25	C2_In_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) = FT_IN) && (V(Nslave) >= V(Nmax_slave)) && (V(Nmax_retry) <= 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) V(Nslave):= V(Nslave) + 1	C2_Sync_Rcv
26	C2_In_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) <> FT_IN)) => (none)	C2_In_Rcv
27	C2_In_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nslave) < V(Nmax_slave))) => (none)	C2_In_Rcv
28	C2_In_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nslave) >= V(Nmax_slave)) && (V(Nmax_retry) > 0)) => V(Nretry):= V(Nmax_retry)	C2_C1Msg_Rcv
29	C2_In_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts <> OK) && (V(Nslave) >= V(Nmax_slave)) && (V(Nmax_retry) <= 0)) => (none)	C2_Sync_Rcv

No.	Current state	Event /condition =>action	Next state
30	C2_C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && (GET_MC(RcvSRCSDU) = Ec1msg)) => EXEC_MSPM_R(Ec1msg, Rcv_sts, RcvSRCSDU, SndSRCSDU) Node_ID:= GET_DA(SndSRCSDU) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	C2_C1Msg_Rcv
31	C2_C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && (GET_MC(RcvSRCSDU) = Ec2msg)) => (none)	C2_C1Msg_Rcv
32	C2_C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) <> FT_MSG)) => (none)	C2_C1Msg_Rcv
33	C2_C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (GET_DA(SRCSDU) <> V(MA))) => (none)	C2_C1Msg_Rcv
34	C2_C1Msg_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / (Rcv_sts <> OK) => (none)	C2_C1Msg_Rcv
35	C2_C1Msg_Rcv	SRC_Send_Frame.cnf { } => (none)	C2_Sync_Rcv
36	C2_C1Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / V(Nretry) > 0 => V(Nretry):= V(Nmax_retry) – 1	C2_C1Msg_Rcv
37	C2_C1Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / V(Nretry) <= 0 => STOP_TIMER(T(Tslot)	C2_Sync_Rcv
38	C2_C1Msg_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot))	C2_Sync_Rcv
39	Any state	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = Cgaddr) && (GET_FT(RcvSRCSDU) = FT_SYNC)) => DL_Event.ind {DL_Ev_Tcycle } RESTART_TIMER(T(Tcycle)) START_TIMER(T(Tslot), V(Tslot)) V(Nslave):= 1	C2_Sync_Rcv
40	Any state	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = Cgaddr) && (GET_FT(RcvSRCSDU) <> FT_SYNC)) => (none)	Same state

No.	Current state	Event /condition =>action	Next state
41	Any state	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	Same state

6.2.3.2.1.3 Slave

Figure 23 and Table 56 show the state diagram and the state table of the slave that adopts fixed-width time slot.

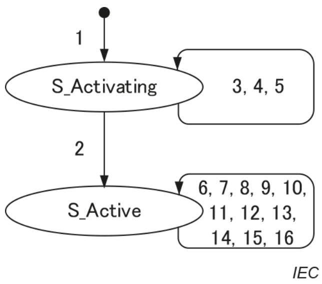


Figure 23 – The state diagram of the slave for fixed-width time slot

Table 56 – The state table of the slave for fixed-width time slot

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => (none)	S_Activating
2	S_Activating	CTC_Start.req => START_TIMER(T(Tcycle), V(Tcycle)) CTC_Start.cnf { OK }	S_Active
3	S_Activating	CTC_Start.req => CTC_Start.cnf { NG }	S_Activating
4	S_Activating	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	S_Activating
5	S_Activating	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	S_Activating
6	S_Active	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle }	S_Active
7	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = Cgaddr) && (GET_FT(RcvSRCSDU) = FT_SYNC)) => DL_Event.ind {DL_Ev_Tcycle } RESTART_TIMER(T(Tcycle))	S_Active

No.	Current state	Event /condition =>action	Next state
8	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = Cgaddr) && (GET_FT(RcvSRCSDU) <> FT_SYNC)) => (none)	S_Active
9	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_OUT)) => SndSRCSDU:= BUILD_PDU_IN(1) Node_ID:= V(MA) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU} STORE_PDU_OUT(1, RcvSRCSDU, Rcv_sts)	S_Active
10	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec1msg)) => EXEC_MSPM_R(Ec1msg, Rcv_sts, RcvSRCSDU, SndSRCSDU) Node_ID:= MA Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU}	S_Active
11	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) = FT_MSG) && GET_MC(RcvSRCSDU) = Ec2msg)) => EXEC_MSPM_R(Ec2msg, Rcv_sts, RcvSRCSDU, SndSRCSDU) Node_ID:= MA Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU}	S_Active
12	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = V(MA)) && (GET_FT(RcvSRCSDU) <> FT_MSG) && (GET_FT(RcvSRCSDU) <> FT_OUT)) => (none)	S_Active
13	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) <> V(MA))) => (none)	S_Active
14	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / (Rcv_sts <> OK) => (none)	S_Active
15	S_Active	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	S_Active
16	S_Active	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	S_Active

6.2.3.2.2 Protocol machine for cyclic communication consists of configurable time slot

6.2.3.2.2.1 C1 master

Figure 24 and Table 57 show the state diagram and the state table of C1 master that adopts configurable time slot.

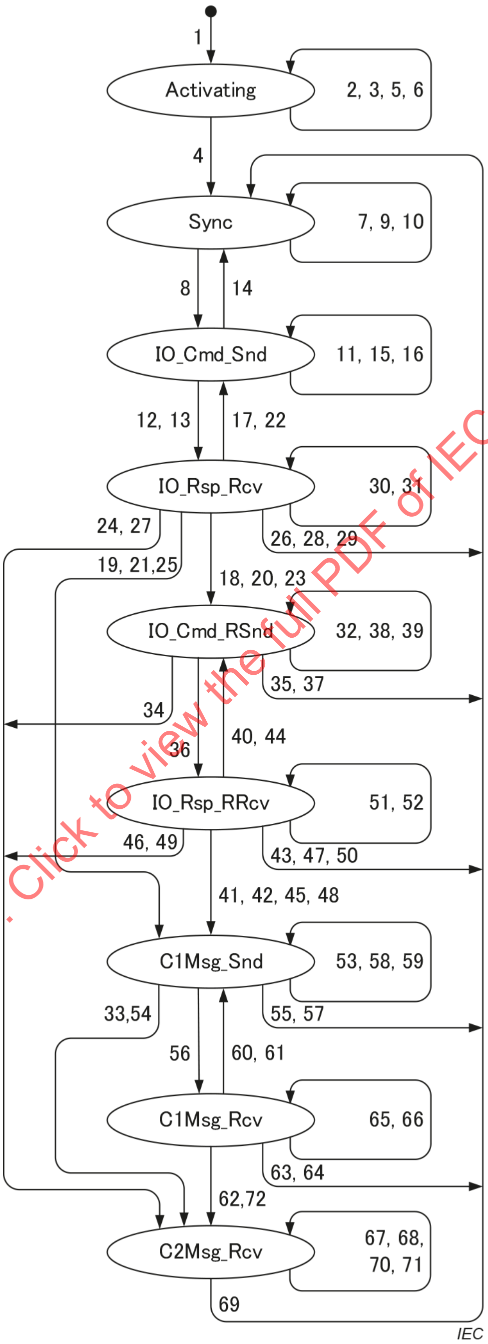


Figure 24 – The state diagram of the C1 master for configurable time slot

Table 57 – The state table of the C1 master for configurable time slot

No.	Current state	Event /condition =>action	Next state
1	Any state	Power on or CTC_Reset.req => (none)	Activating
2	Activating	CTC_Set_Par.req { Par_ID, Val } => CTC_SET_PAR(Par_ID, Val, Result) CTC_Set_Par.cnf { Result }	Activating
3	Activating	CTC_Get_Par.req { Par_ID } => CTC_GET_PAR(Par_ID, Val, Result) CTC_Get_Par.cnf { Result, Par_ID, Val }	Activating
4	Activating	CTC_Start.req => START_TIMER(T(Tcycle), V(Tcycle)) CTC_Start.cnf	Sync
5	Activating	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	Activating
6	Activating	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	Activating
7	Sync	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync
8	Sync	SRC_Send_Frame.cnf { } => V(Nslave):= 1 V(Nretry):= 0 V(Nrest_slot):= 0	IO_Cmd_Snd
9	Sync	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	Sync
10	Sync	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	Sync
11	IO_Cmd_Snd	EXPIRED_TIMER (T(Tslot)) = "True" => SRCSDU:= BUILD_PDU_OUT(V(Nslave)) SRC_Send_Frame.req {SRCSDU} STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	IO_Cmd_Snd
12	IO_Cmd_Snd	SRC_Send_Frame.cnf { } / V(Nslave) < V(Nmax_slave) => (none)	IO_Rsp_Rcv

No.	Current state	Event /condition =>action	Next state
13	IO_Cmd_Snd	SRC_Send_Frame.cnf { } / V(Nslave) >= V(Nmax_slave) => V(Nrest_slot):= V(Nmax_retry)	IO_Rsp_Rcv
14	IO_Cmd_Snd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle} STOP_TIMER(T(Tslot)) SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync
15	IO_Cmd_Snd	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	IO_Cmd_Snd
16	IO_Cmd_Snd	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	IO_Cmd_Snd

17	IO_Rsp_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU}
		/ ((Rcv_sts = OK) && V(Nslave) < V(Nmax_slave))
		=> STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)
		If ((Rcv_sts <> OK) && (V(Nrest_slot) > 0)) then
		PUT_RETRY_LIST(V(Nslave))
		Endif
		V(Nslave):= V(Nslave) + 1
			IO_Cmd_Snd

18	IO_Rsp_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU}
		/ ((Rcv_sts = OK) && (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) > 0) && (V(Nretry) > 0))
		=> STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)
		If (Result <> OK) then
		PUT_RETRY_LIST(V(Nslave))
		Endif
			IO_Cmd_RSnd

19	IO_Rsp_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) > 0) && (V(Nretry) = 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts)	C1Msg_Snd
20	IO_Rsp_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = NG) && (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) > 0) && (V(Nretry) = 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Rcv_sts) PUT_RETRY_LIST(V(Nslave))	IO_Cmd_RSnd
21	IO_Rsp_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / ((Rcv_sts = OK) && (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) <= 0)) => STORE_PDU_IN(V(Nslave), SRCSDU, Result) SND_MSG_TOKEN()	C2Msg_Rcv

No.	Current state	Event /condition =>action	Next state

22	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tslot)) = "True"
		/ V(Nslave) < V(Nmax_slave)
		=> STORE_PDU_IN(V(Nslave), SRCSDU, NG)
		If V(Nrest_slot) > 0 then
		PUT_RETRY_LIST(V(Nslave))
		Endif
		V(Nslave):= V(Nslave) + 1
		SRCSDU:= BUILD_PDU_OUT(V(Nslave))
		SRC_Send_Frame.req {SRCSDU}
		STOP_TIMER(T(Tslot))
		START_TIMER(T(Tslot), V(Tslot(V(Nslave))))
			IO_Cmd_Snd

23	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) > 0) && CMP_RTIM(T(Tcycle), V(Tslot(V(GET_CUR_RETRY_LIST))) + V(Tc2_dly)) > 0 => STORE_PDU_IN(V(Nslave), SRCSDU, NG) PUT_RETRY_LIST(V(Nslave)) V(Nslave):= GET_RETRY_LIST() SRCSDU:= BUILD_PDU_OUT(V(Nslave)) SRC_Send_Frame.req {SRCSDU} STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot), V(Tslot(V(slave))))	IO_Cmd_RSnd
24	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) > 0) && CMP_RTIM(T(Tcycle), V(Tslot(V(GET_RETRY_LIST))) + V(Tc2_dly)) = 0 && CHECK_MSG_EN(C2) <> 0 => STORE_PDU_IN(V(Nslave), SRCSDU, NG)	C2Msg_Rcv
25	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) <= 0) && CHECK_MSG_EN(C1) <> 0 => STORE_PDU_IN(V(Nslave), SRCSDU, NG) EXEC_MSPM_IS(Ec1msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C1) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot)) SND_MSG_TOKEN()	C1Msg_Snd
26	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) > 0) && CMP_RTIM(T(Tcycle), V(Tslot(V(GET_RETRY_LIST))) + V(Tc2_dly)) = 0 && CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) = 0 => STORE_PDU_IN(V(Nslave), SRCSDU, NG)	Sync

No.	Current state	Event /condition =>action	Next state
27	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) <= 0) && CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) <> 0 => STORE_PDU_IN(V(Nslave), SRCSDU, NG) SND_MSG_TOKEN()	C2Msg_Rcv
28	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nslave) >= V(Nmax_slave)) && (V(Nrest_slot) <= 0) && CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) = 0 => STORE_PDU_IN(V(Nslave), SRCSDU, NG)	Sync
29	IO_Rsp_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)) STORE_PDU_IN(V(Nslave), SRCSDU, NG) SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync
30	IO_Rsp_Rcv	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	IO_Rsp_Rcv
31	IO_Rsp_Rcv	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	IO_Rsp_Rcv
32	IO_Cmd_RSnd	EXPIRED_TIMER (T(Tslot)) = "True" / CMP_RTIM(T(Tcycle), V(Tslot(V(GET_CUR_RETRY_LIST))) + V(Tc2_dly)) > 0 => V(Nslave):= GET_RETRY_LIST() SRCSDU:= BUILD_PDU_OUT(V(Nslave)) SRC_Send_Frame.req {SRCSDU} STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot), V(Tslot(V(slave))))	IO_Cmd_RSnd
33	C1Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) <> 0 && CMP_RTIM(T(Tcycle), V(Tc2_dly)) = 0 => (none)	C2Msg_Rcv
34	IO_Cmd_RSnd	EXPIRED_TIMER (T(Tslot)) = "True" / CMP_RTIM(T(Tcycle), V(Tslot(V(GET_RETRY_LIST))) + V(Tc2_dly)) = 0 && CHECK_MSG_EN(C2) <> 0 => SND_MSG_TOKEN()	C2Msg_Rcv

No.	Current state	Event /condition =>action	Next state
35	IO_Cmd_RSnd	EXPIRED_TIMER (T(Tslot)) = "True" / CMP_RTIM(T(Tcycle), V(Tslot(V(GET_RETRY_LIST))) + V(Tc2_dly)) = 0 && CHECK_MSG_EN(C2) = 0 =>(none)	Sync
36	IO_Cmd_RSnd	SRC_Send_Frame.cnf { } => (none)	IO_Rsp_RRcv
37	IO_Cmd_RSnd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)) SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync
38	IO_Cmd_RSnd	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	IO_Cmd_RSnd
39	IO_Cmd_RSnd	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	IO_Cmd_RSnd
40	IO_Rsp_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / (V(Nrest_slot) > 1) && (V(Nretry) > 0) => STORE_PDU_IN(V(Nslave), SRCSDU, Result) V(Nrest_slot):= V(Nrest_slot) – 1	IO_Cmd_RSnd
41	IO_Rsp_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / (V(Nrest_slot) > 1) && (V(Nretry) <= 0) => STORE_PDU_IN(V(Nslave), SRCSDU, Result)	C1Msg_Snd
42	IO_Rsp_RRcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / (V(Nrest_slot) <= 1) => STORE_PDU_CYCR(V(Nslave), SRCSDU, Result)	C1Msg_Snd
43	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot) STORE_PDU_IN(V(Nslave), SRCSDU, NG) SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync

No.	Current state	Event /condition =>action	Next state
44	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nrest_slot) > 1) && (V(Nretry) > 0) && CMP_RTIM(T(Tcycle), V(Tslot(V(GET_CUR_RETRY_LIST))) + V(Tc2_dly)) > 0 => STORE_PDU_IN(V(Nslave), SRCSDU, NG) V(Nrest_slot):= V(Nrest_slot) – 1 V(Nslave):= GET_RETRY_LIST() SRCSDU:= BUILD_PDU_OUT(V(Nslave)) SRC_Send_Frame.req {SRCSDU}	IO_Cmd_RSnd
45	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nrest_slot) <= 1) && CHECK_MSG_EN(C1) <> 0 => EXEC_MSPM_IS(Ec1msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C1) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C1Msg_Snd
46	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nrest_slot) <= 1) && CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) <> 0 => SND_MSG_TOKEN()	C2Msg_Rcv
47	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" / (V(Nrest_slot) <= 1) && CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) = 0 =>(none)	Sync
48	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" /(V(Nrest_slot) > 1) && (V(Nretry) <= 0) && CMP_RTIM(T(Tcycle), V(Tslot(V(GET_RETRY_LIST))) + V(Tc2_dly)) = 0 && CHECK_MSG_EN(C1) <> 0 => EXEC_MSPM_IS(Ec1msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C1) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C1Msg_Snd
49	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" /CMP_RTIM(T(Tcycle), V(Tslot(V(GET_RETRY_LIST))) + V(Tc2_dly)) = 0 && CHECK_MSG_EN(C2) <> 0 => SND_MSG_TOKEN()	C2Msg_Rcv

No.	Current state	Event /condition =>action	Next state
50	IO_Rsp_RRcv	EXPIRED_TIMER (T(Tslot)) = "True" /CMP_RTIM(T(Tcycle), V(Tslot(V(GET_RETRY_LIST))) + V(Tc2_dly)) = 0 && CHECK_MSG_EN(C2) = 0 =>(none)	Sync
51	IO_Rsp_RRcv	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	IO_Rsp_RRcv
52	IO_Rsp_RRcv	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	IO_Rsp_RRcv
53	C1Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) <> 0 && CMP_RTIM(T(Tcycle), V(Tc2_dly)) > 0 => EXEC_MSPM_IS(Ec1msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C1) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C1Msg_Snd
54	C1Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) <> 0 => SND_MSG_TOKEN()	C2Msg_Rcv
55	C1Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) = 0 =>(none)	Sync
56	C1Msg_Snd	SRC_Send_Frame.cnf { } => (none)	C1Msg_Rcv
57	C1Msg_Snd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot) SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync
58	C1Msg_Snd	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	C1Msg_Snd
59	C1Msg_Snd	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	C1Msg_Snd
60	C1Msg_Rcv	SRC_Recv_Frame.ind {Result, RcvSRCSDU} => STOP_TIMER(T(Tslot)) EXEC_MSPM_IR(Ec1msg, Result, RcvSRCSDU)	C1Msg_Snd

No.	Current state	Event /condition =>action	Next state
61	C1Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) <> 0 && CMP_RTIM(T(Tcycle), V(Tc2_dly)) > 0 => EXEC_MSPM_IS(Ec1msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C1) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C1Msg_Snd
62	C1Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) <> 0 => SND_MSG_TOKEN()	C2Msg_Rcv
63	C1Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) = 0 && CHECK_MSG_EN(C2) = 0 =>(none)	Sync
64	C1Msg_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)) SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync
65	C1Msg_Rcv	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	C1Msg_Rcv
66	C1Msg_Rcv	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	C1Msg_Rcv
67	C2Msg_Rcv	SRC_Recv_Frame.ind { Result, RcvSRCSDU } => EXEC_MSPM_R(Ec2msg, Result, RcvSRCSDU, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU }	C2Msg_Rcv
68	C2Msg_Rcv	SRC_Send_Frame.cnf { } => (none)	C2Msg_Rcv
69	C2Msg_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)) SRCSDU:= BUILD_PDU_SYNC() SRC_Send_Frame.req {SRCSDU} V(Nslave):= 0 START_TIMER(T(Tslot), V(Tslot(V(Nslave))))	Sync
70	C2Msg_Rcv	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	C2Msg_Rcv

No.	Current state	Event /condition =>action	Next state
71	C2Msg_Rcv	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	C2Msg_Rcv
72	C1Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C1) <> 0 && CMP_RTIM(T(Tcycle), V(Tc2_dly)) = 0 => (none)	C2Msg_Rcv

6.2.3.2.2.2 C2 Master

Figure 25 and Table 58 show the state diagram and the state table of the C2 master that adopts configurable time slot.

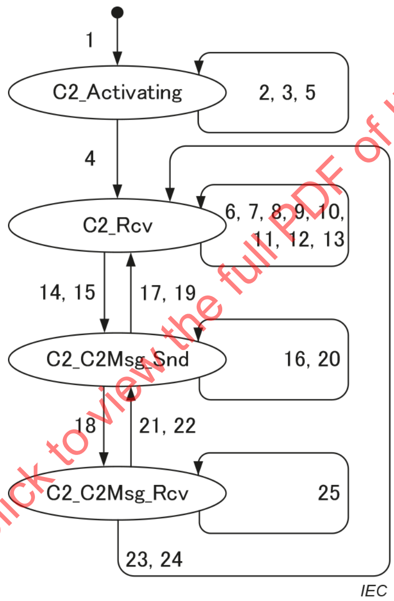


Figure 25 – The state diagram of the C2 master for configurable time slot

Table 58 – The state table of the C2 master for configurable time slot

No.	Current state	Event /condition =>action	Next state
1	Any state	Power on or CTC_Reset.req => (none)	C2_Activating
2	C2_Activating	CTC_Set_Par.req { Par_ID, Val } => CTC_SET_PAR(Par_ID, Val, Result) CTC_Set_Par.req { Result }	C2_Activating
3	C2_Activating	CTC_Get_Par.req { Par_ID } => CTC_GET_PAR(Par_ID, Val, Result) CTC_Get_Par.cnf { Result, Par_ID, Val }	C2_Activating

No.	Current state	Event /condition =>action	Next state
4	C2_Activating	CTC_Start.req => START_TIMER(T(Tcycle), V(Tcycle)) START_TIMER(T(Tmsg), V(Tmsg)) CTC_Start.cnf	C2_Rcv
5	C2_Activating	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	C2_Activating
6	C2_Rcv	SRC_Recv_Frame.ind { Result, SRCSDU } / (SRCSDU.TYPLEN = TYP1) => DL_Event.ind {DL_Ev_Tcycle } RESTART_TIMER(T(Tcycle)) RESTART_TIMER(T(Tmsg))	C2_Rcv
7	C2_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tcycle)) STOP_TIMER(T(Tmsg)) START_TIMER(T(Tcycle), V(Tcycle)) START_TIMER(T(Tmsg), V(Tmsg))	C2_Rcv
8	C2_Rcv	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / RcvSRCSDU. SA = C1 RcvSRCSDU. TYPLEN = TYP2 => V(Nslave):= EX_ADD(DA) STORE_PDU_OUT(V(Nslave), SRCSDU, Result)	C2_Rcv
9	C2_Rcv	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / RcvSRCSDU. SA = Slave RcvSRCSDU. TYPLEN = TYP2 => V(Nslave):= EX_ADD(DA) STORE_PDU_IN(V(Nslave), SRCSDU, Result)	C2_Rcv
10	C2_Rcv	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / SRCSDU. SA = C1 SRCSDU.TYPLEN = TYP12 => EXEC_MSPM_R(Ec1msg, Result, RcvSRCSDU, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU }	C2_Rcv
11	C2_Rcv	SRC_Send_Frame.cnf { } => (none)	C2_Rcv
12	C2_Rcv	EXPIRED_TIMER (T(Tmsg)) = "True" / CHECK_MSG_EN(C2) = 0 => (none)	C2_Rcv
13	C2_Rcv	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	C2_Rcv

No.	Current state	Event /condition =>action	Next state
14	C2_Rcv	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / RcvSRCSDU. SA = V(MA) RcvSRCSDU. TYPLEN = TYP5 => EXEC_MSPM_IS(Ec2msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C2) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C2_C2Msg_Snd
15	C2_Rcv	EXPIRED_TIMER (T(Tmsg)) = "True" / CHECK_MSG_EN(C2) <> 0 => EXEC_MSPM_IS(Ec2msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C1) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C2_C2Msg_Snd
16	C2_C2Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C2) <> 0 => EXEC_MSPM_IS(Ec2msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C2) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C2_C2Msg_Snd
17	C2_C2Msg_Snd	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C2) = 0 => (none)	C2_Rcv
18	C2_C2Msg_Snd	SRC_Send_Frame.cnf { } => (none)	C2_C2Msg_Rcv
19	C2_C2Msg_Snd	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)) STOP_TIMER(T(Tcycle)) STOP_TIMER(T(Tmsg)) START_TIMER(T(Tcycle), V(Tcycle)) START_TIMER(T(Tmsg), V(Tmsg))	C2_Rcv
20	C2_C2Msg_Snd	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	C2_C2Msg_Snd
21	C2_C2Msg_Rcv	SRC_Recv_Frame.ind { Result, RcvSRCSDU } => EXEC_MSPM_IR(Ec2msg, Result, RcvSRCSDU)	C2_C2Msg_Snd

No.	Current state	Event /condition =>action	Next state
22	C2_C2Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C2) <> 0 => EXEC_MSPM_IS(Ec2msg, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU } slot:= GET_MSG_SLOT(C2) STOP_TIMER(T(Tslot)) START_TIMER(T(Tslot),slot))	C2_C2Msg_Snd
23	C2_C2Msg_Rcv	EXPIRED_TIMER (T(Tslot)) = "True" / CHECK_MSG_EN(C2) = 0 => (none)	C2_Rcv
24	C2_C2Msg_Rcv	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle } STOP_TIMER(T(Tslot)) STOP_TIMER(T(Tcycle)) STOP_TIMER(T(Tmsg)) START_TIMER(T(Tcycle), V(Tcycle)) START_TIMER(T(Tmsg), V(Tmsg))	C2_Rcv
25	C2_C2Msg_Rcv	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	C2_C2Msg_Rcv

6.2.3.2.2.3 Slave

Figure 26 and Table 59 show the state diagram and the state table of slave which adopts configurable time slot.

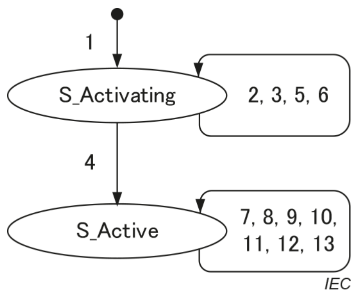


Figure 26 – The state diagram of slave for configurable time slot

Table 59 – The state table of slave for configurable time slot

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => (none)	S_Activating
2	S_Activating	CTC_Set_Par.req { Par_ID, Val } => CTC_SET_PAR(Par_ID, Val, Result) CTC_Set_Par.cnf { Result }	S_Activating
3	S_Activating	CTC_Get_Par.req { Par_ID } => CTC_GET_PAR(Par_ID, Val, Result) CTC_Get_Par.cnf { Result, Par_ID, Val }	S_Activating
4	S_Activating	CTC_Start.req => START_TIMER(T(Tcycle), V(Tcycle)) CTC_Start.cnf	S_Active
5	S_Activating	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	S_Activating
6	S_Activating	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	S_Activating
7	S_Active	EXPIRED_TIMER (T(Tcycle)) = "True" => DL_Event.ind {DL_Ev_Tcycle }	S_Active
8	S_Active	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / (SRCSDU.TYPLEN = TYP1) => DL_Event.ind {DL_Ev_Tcycle } RESTART_TIMER(T(Tcycle))	S_Active
9	S_Active	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / RcvSRCSDU.DA = V(MA) && RcvSRCSDU. TYPLEN = TYP2 => STORE_PDU_OUT(1, SRCSDU, Result) SndSRCSDU:= BUILD_PDU_IN(V(MA)) SRC_Send_Frame.req { SndSRCSDU}	S_Active
10	S_Active	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / RcvSRCSDU.DA = V(MA) RcvSRCSDU.SA = C1 && RcvSRCSDU. TYPLEN = TYP12 => EXEC_MSPM_R(Ec1msg, Result, RcvSRCSDU, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU }	S_Active
11	S_Active	SRC_Recv_Frame.ind { Result, RcvSRCSDU } / RcvSRCSDU.DA = V(MA) RcvSRCSDU.SA = C2 && RcvSRCSDU. TYPLEN = TYP12 => EXEC_MSPM_R(Ec2msg, Result, RcvSRCSDU, SndSRCSDU) SRC_Send_Frame.req { SndSRCSDU }	S_Active
12	S_Active	DL-WRITE-DATA.req(SAP_ID, DLSDU) => result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	S_Active
13	S_Active	DL-READ-DATA.req(SAP_ID) => result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	S_Active

6.2.3.2.3 Functions used by cyclic transmission machine

All the functions used by the fixed-width time slot protocol machine are summarized in Table 60.

Table 60 – The list of functions used by cyclic transmission machine

Function name	Parameter		Return	Operation
	Input	Output	Value	
BUILD_PDU_SYNC	none	None	SRCSDU	This function builds SRCSDU to request SRC to send the synchronous frame.
BUILD_PDU_OUT	Slave_index,	None	SRCSDU	This function builds SRCSDU to request SRC to the output data frame. Slave_Index is the number of slave and its range is 1 to V(Nmax_slave).
BUILD_PDU_IN	Slave_index,	None	SRCSDU	This function builds SRCSDU to request SRC to the input data frame. Slave_Index is the number of slave and its range is 1 to V(Nmax_slave).
STORE_PDU_OUT	Slave_index, SRCSDU, Rcv_Sts	None	None	This function stores SRCSDU retrieved from SRC and receives status Rcv_Sts as received the output data. Slave_Index is the number of slave and its range is 1 to V(Nmax_slave).
STORE_PDU_IN	Slave_index, SRCSDU, Rcv_Sts	None	None	This function stores SRCSDU retrieved from SRC and receives status Rcv_Sts as received the input data. Slave_Index is the number of slave and its range is 1 to V(Nmax_slave).
GET_DA	SRCPDU	None	Node_ID	This function takes out the value of the destination address field of the specified SRCSDU.
GET_FT	SRCSDU	None	frame_type	This function takes out the value of the type field of the specified SRCSDU.
GET_MC	SRCSDU	None	Flag	This function takes out the flag of the C1/C2 in the message control field of the specified SRCSDU.
GET_LEN	SRCSDU	None	Length	This function takes out the value of the length field of the specified SRCSDU.
GET_NODE_ID	Slave_index	None	Node_ID	This function retrieves a Node_ID corresponding to slave number.
PUT_RETRY_LIST	Slave_index	None	None	This function registers the slave number specified with Slave_index to retry list, and increments the value of V(Nretry) that contains the number of registered slave number.
GET_RETRY_LIST	none	None	Slave_index	This function retrieves a slave number from retry list, and decrements the value of V(Nretry) that contains the number of registered slave number.
GET_CUR_RETRY _LIST	None	None	Slave_index	This function retrieves a slave number from retry list.

Function name	Parameter		Return Value	Operation
	Input	Output		
EXEC_MSPM_R	Fc1c2, Rcv_sts, RcvSRCSDU	SndSRCSDU	none	This function calls the message segmentation machine for the responder of message communication. The parameter Fc1c2 is flag to specify C1 message or C2 message. Rcv_sts and RcvSRCSDU are receive status and received SRCSDU respectively that has been passed by SRC. SndSRCSDU is SRCSDU to be sent.
EXEC_MSPM_IS	Ec1c2	SndSRCSDU	None	This function calls the message segmentation protocol machine for the initiator of message communication to process a sending frame. The parameters of this function are same as the parameters with the same name of the function EXEC_MSPM_R.
EXEC_MSPM_IR	Ec1c2, Rcv_sts, RcvSRCSDU	None	None	This function calls the message segmentation protocol machine for the initiator of message communication to process a received frame. The parameters of this function are same as the parameters with the same name of the function EXEC_MSPM_R.
START_TIMER	Tim_ID, Val	None	None	This function starts the timer specified with Tim_ID in set value Val. The timer is auto-reload type, which load the set value again when the timer is timed up.
STOP_TIMER	Tim_ID	None	None	This function stops the timer specified with Tim_ID.
RESTART_TIMER	Tim_ID	None	None	This function restarts the timer specified with Tim_ID with the set value specified by START_TIMER.
EXPIRED_TIMER	Tim_ID	None	Flag	This function indicates the status of the timer specified with Tim_ID.
CTC_SET_PAR	Var_ID, Val	Result	None	This function updates the variable specified with Var_ID in the DLE in the value specified with Val.
CTC_GET_PAR	Var_ID	Val, Result	None	This function reads the current value of the variable specified with Var_ID in the DLE.
SET_CYC_DATA	SAP_ID, Node_ID, DLSDU	None	None	This function updates the send buffer for the I/O data exchange that exists in CTC in the specified data.
GET_CYC_DATA	SAP_ID, Node_ID	DLSDU	Result	This function updates the receive buffer for the I/O data exchange that exists in CTC in the specified data.

Function name	Parameter		Return	Operation
	Input	Output	Value	
CMP_RTIM	Tim_ID, Val	None	Flag	This function compares the remainder time of the timer specified with Tim_ID with set value Val. If the remainder time is large, True is returned and if it is small, False is returned.
CHECK_MSG_EN	Msg_mst	None	Enable_flag	This function judges whether the message communication that the station specified with Msg_mst operates as an initiator can be executed. When the following three conditions are satisfied, enable_flag is turned on: - The message communication band is configured; - DLS-user issued a request; - Time for message communication remains.
GET_MSG_SLOT	Msg_mst	None	msg_slot	This function outputs the period of time slot for the message communication that the station specified with Msg_mst operates as an initiator.
EX_ADD	Node_address	None	Slave_index	This function converts the station address to slave number.
SND_MSG_TOKEN	None	None	None	This function transmits the token frame when time that Message Token frame can be transmitted remains by the C2 message beginning time.

6.2.3.2.4 Protocol machine for cyclic communication consists of no time slot type

6.2.3.2.4.1 C1 master

Figure 27 and Table 61 show the state diagram and the state table of the C1 master to which no time slot type is applied.

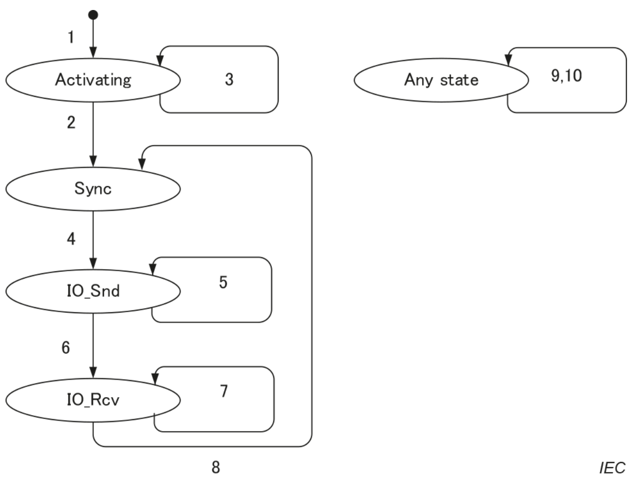


Figure 27 – The state diagram of the C1 master for no time slot type

Table 61 – The state table of the C1 master for no time slot type

No.	Current state	Event /condition =>action	Next state
1	Any state	Power on or CTC_Reset.req => (none)	Activating
2	Activating	CTC_Start.req => START_TIMER(T(Tcycle), V(Tcycle)) CTC_Start.cnf { OK }	Sync
3	Activating	CTC_Start.req => CTC_Start.cnf { NG }	Activating
4	Sync	EXPIRED_TIMER (T(Tcycle)) = "True" => START_TIMER(T(Tlatch))	IO_Snd
5	IO_Snd	SRCSDU:= BUILD_PDU_OUT(V(Nslave)) Node_ID:= GET_DA(SndSRCSDU) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SRCSDU}	IO_Snd
6	IO_Snd	SRC_Send_Frame.cnf { } => (none)	IO_Rcv
7	IO_Rcv	SRC_Recv_Frame.ind {Rcv_sts, Length, SRCSDU} / (Rcv_sts = OK) => STORE_PDU_IN(GET_SA(RcvSRCSDU), SRCSDU, Rcv_sts)	IO_Rcv
8	IO_Rcv	EXPIRED_TIMER (T(Tlatch)) = "True" => STOP_TIMER (T(Tlatch))	Sync
9	Any state	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	Same state
10	Any state	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, Node_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	Same state

6.2.3.2.4.2 C2 master

Figure 28 and Table 62 show the state diagram and the state table of the C2 master to which no time slot type is applied.

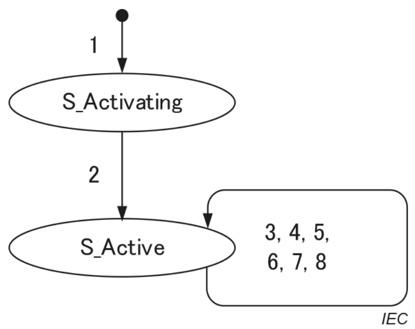


Figure 28 – The state diagram of the C2 master for no time slot type

Table 62 – The state table of the C2 master for no time slot type

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => (none)	S_Activating
2	S_Activating	CTC_Start.req => CTC_Start.cnf { OK }	S_Active
3	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) = FT_SYNC)) => (none)	S_Active
4	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = MA) && (GET_FT(RcvSRCSDU) =FT_ASYNC)) => SndSRCSDU:= BUILD_PDU_IN(1) Node_ID:= V(MA) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU} STORE_PDU_OUT(1, RcvSRCSDU, Rcv_sts)	S_Active
5	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = Broadcast) && (GET_FT(RcvSRCSDU) = FT_ASYNC)) => STORE_PDU_OUT(0, RcvSRCSDU, Rcv_sts)	S_Active
6	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_SA(RcvSRCSDU) = MonTarget[a])) => STORE_PDU_IN(GET_SA(RcvSRCSDU), RcvSRCSDU, Rcv_sts)	S_Active
7	S_Active	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	S_Active
8	S_Active	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	S_Active
[a] Address of the target slave to be monitored by the C2			

6.2.3.2.4.3 Slave

Figure 29 and Table 63 show the state diagram and the state table of the slave to which no time slot type is applied.

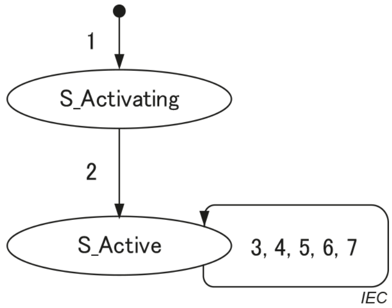


Figure 29 – The state diagram of the Slave for no time slot type

Table 63 – The state table of the Slave for no time slot type

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => (none)	S_Activating
2	S_Activating	CTC_Start.req => CTC_Start.cnf { OK }	S_Active
3	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_FT(RcvSRCSDU) = FT_SYNC)) => SndSRCSDU:= BUILD_PDU_IN(1) Node_ID:= V(MA) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU}	S_Active
4	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = MA) && (GET_FT(RcvSRCSDU) = FT_ASYNC)) => SndSRCSDU:= BUILD_PDU_IN(1) Node_ID:= V(MA) Length:= GET_LEN(SndSRCSDU) SRC_Send_Frame.req { Node_ID, Length, SndSRCSDU} STORE_PDU_OUT(0, RcvSRCSDU, Rcv_sts)	S_Active
5	S_Active	SRC_Recv_Frame.ind { Rcv_sts, Length, RcvSRCSDU } / ((Rcv_sts = OK) && (GET_DA(RcvSRCSDU) = Broadcast) && (GET_FT(RcvSRCSDU) = FT_ASYNC)) => STORE_PDU_OUT(0, RcvSRCSDU, Rcv_sts)	S_Active
6	S_Active	DL-WRITE-DATA.req(SAP_ID, DLSDU) => Result:= SET_CYC_DATA(SAP_ID, DLSDU) DL-WRITE-DATA.cnf (Result)	S_Active
7	S_Active	DL-READ-DATA.req(SAP_ID) => Result:= GET_CYC_DATA(SAP_ID, DLSDU) DL-READ-DATA.cnf (Result, DLSDU)	S_Active

6.2.3.2.5 Functions used by cyclic transmission machine for no time slot type

See Table 60.

6.2.3.3 Message segmentation protocol machine

6.2.3.3.1 General

Message segmentation and the assembly specification of the message for the message segmentation machine have two types of frame formats. They are described in the following subclause.

The message communication is classified into two types according to the allocated band. The first type is C1 message communication in which the C1 master operates as primary station (initiator) and the slave or the C2 master operates as secondary station (responder). The second type is C2 message communication in which the C2 master operates a primary station and the slave or the C1 master becomes a secondary station. The operation of the primary station and the secondary station is the same although they are different with respect to the allocated band.

Message segmentation protocol machine is executed by the cyclic transmission protocol machine with the function call whose name is prefixed by "EXEC_MSPM_". In the state table of the following subclause, the event "Call" means the function calls.

6.2.3.3.2 Segmentation for basic format DLPDU

6.2.3.3.2.1 Initiator for basic format DLPDU

Figure 30 and Table 64 show the state diagram and the state table of an initiator of the message segmentation machine when DLE adopts the basic format DLPDU.

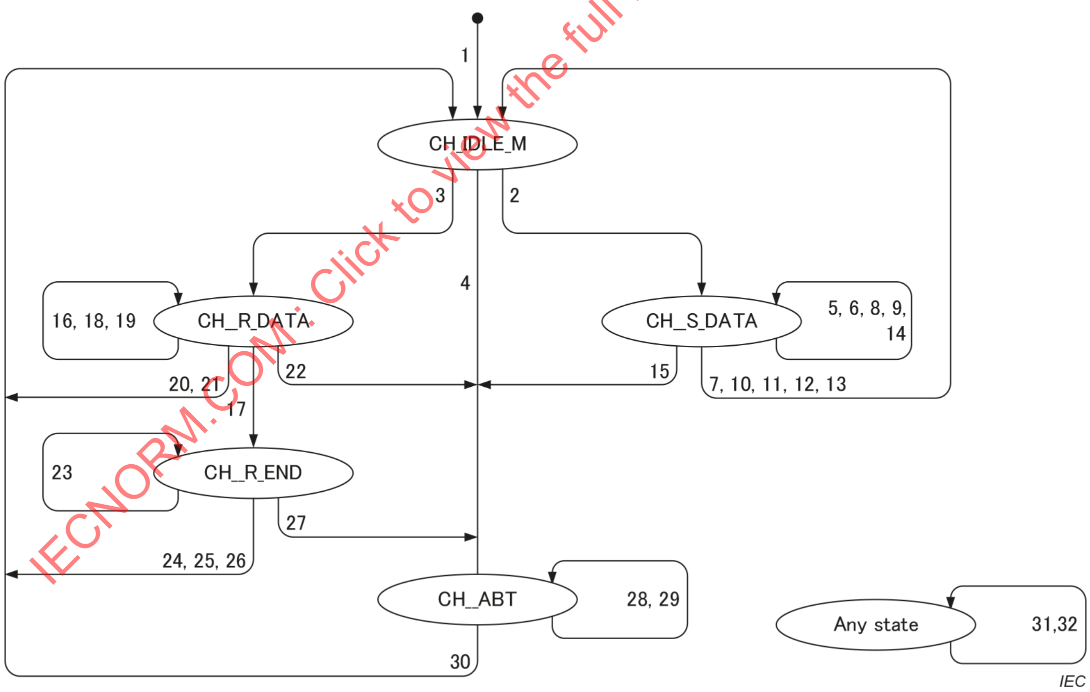


Figure 30 – The state diagram of message initiator for basic format

Table 64 – The state table of message initiator for basic format

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => V(Nms_n):= 0 V(Nmr_n):= 0	CH_IDLE_M
2	CH_IDLE_M	DL-SDA.req{SAP_ID,Node_ID,Length,SndDLSDU} => V(Nms_n):= 0 STORE_DLSDU(Ec1c2, Node_ID, SndDLSDU) V(Nmsg_len_n):= Length V(Nmsg_rem_len_n):= Length V(Fmsg_sending_n):= True	CH_S_DATA
3	CH_IDLE_M	EXEC_MSPM_IS(Ec1c2, SndSRCSDU) => V(Nmr_n):= 0 V(Fmsg_sending_n):= False V(Nmp_n):= 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RR, 0, V(Nmr_n), V(Nmp_n))	CH_R_DATA
4	CH_IDLE_M	DL-ABORT-SDA.req { SAP_ID } => V(Fmsg_sending_n):= False	CH_ABT
5	CH_S_DATA	EXEC_MSPM_IS(Ec1c2, SndSRCSDU) / V(Npkt_len_n) >= V(Nmsg_rem_len_n) => V(Nmp_n) = 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_I, V(Nms_n), V(Nmr_n), V(Nmp_n))	CH_S_DATA
6	CH_S_DATA	EXEC_MSPM_IS(Ec1c2, SndSRCSDU) / V(Npkt_len_n) < V(Nmsg_rem_len_n) => V(Nmp_n) = 0 SndSRCSDU:= BUILD_PDU_BMSG(MF_I, V(Nms_n), V(Nmr_n),V(Nmp_n)) V(Nmsg_rem_len_n):= V(Nmsg_rem_len_n) - V(Npkt_len_n)	CH_S_DATA
7	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR && GET_MC_NR(RcvSRCSDU) = V(Nms_n)+1 && V(Nmp_n) = 1 => V(Nms_n):= V(Nms_n) + 1 DL-SDA.cnf{ SND_OK }	CH_IDLE_M
8	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR && GET_MC_NR(RcvSRCSDU) = V(Nms_n)+1 && V(Nmp_n) = 0 => V(Nms_n):= V(Nms_n) + 1	CH_S_DATA
9	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR && GET_MC_NR(RcvSRCSDU) <> V(Nms_n)+1 => (none)	CH_S_DATA

No.	Current state	Event /condition =>action	Next state
10	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RNR && GET_MC_NR(RcvSRCSDU) = V(Nms_n)+1 && V(Nmp_n) = 1 => V(Nms_n):= V(Nms_n) + 1 DL-SDA.cnf{ SND_OK }	CH_IDLE_M
11	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RNR && GET_MC_NR(RcvSRCSDU) = V(Nms_n)+1 && V(Nmp_n) = 0 => DL-SDA.cnf{ SND_BUSY }	CH_IDLE_M
12	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RNR && GET_MC_NR(RcvSRCSDU) <> V(Nms_n)+1 => DL-SDA.cnf{ SND_BUSY }	CH_IDLE_M
13	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_REJ => V(Nms_n):= 0, V(Nmr_n):= 0 DL-SDA.cnf{ SND_ABT }	CH_IDLE_M
14	CH_S_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I => (none)	CH_S_DATA
15	CH_S_DATA	DL-ABORT-SDA.req { SAP_ID } => (none)	CH_ABT
16	CH_R_DATA	EXEC_MSPM_IS(Ec1c2, SndSRCSDU) => V(Nmp_n):= 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RR,0,V(Nmr_n),V(Nmp_n))	CH_R_DATA
17	CH_R_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I && GET_MC_NS(RcvSRCSDU) = V(Nmr_n) && GET_MC_F(RcvSRCSDU) = 1 => V(Nmr_n):= V(Nmr_n) + 1 STORE_PDU_MSG(Ec1c2, RcvSRCSDU)	CH_R_END
18	CH_R_DATA	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I && GET_MC_NS(RcvSRCSDU) = V(Nmr_n) && GET_MC_F(RcvSRCSDU) = 0 => V(Nmr_n):= V(Nmr_n) + 1 STORE_PDU_BMSG(RcvSRCSDU)	CH_R_DATA

<table>
<tr><th>No.</th><th>Current state</th><th>Event
/condition
=>action</th><th>Next state</th></tr>
<tr><td>19</td><td>CH_R_DATA</td><td>EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)

/ Rcv_sts = OK
&& GET_FT(RcvSRCSDU) = FT_MSG
&& GET_MC_FMT(RcvSRCSDU) = MF_I
&& GET_MC_NS(RcvSRCSDU) <> V(Nmr_n)

=> (none)</td><td>CH_R_DATA</td></tr>
<tr><td>20</td><td>CH_R_DATA</td><td>EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)

/ Rcv_sts = OK
&& GET_FT(RcvSRCSDU) = FT_MSG)
&& { (GET_MC_FMT(RcvSRCSDU) = MF_S_RR) ||
(GET_MC_FMT(RcvSRCSDU) = MF_S_RNR) }

=> (none)</td><td>CH_IDLE_M</td></tr>
<tr><td>21</td><td>CH_R_DATA</td><td>EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)

/ Rcv_sts = OK
&& GET_FT(RcvSRCSDU) = FT_MSG
&& GET_MC_FMT(RcvSRCSDU) = MF_S_REJ

=> V(Nms_n):= 0, V(Nmr_n):= 0</td><td>CH_IDLE_M</td></tr>
<tr><td>22</td><td>CH_R_DATA</td><td>DL-ABORT-SDA.req { SAP_ID }

=> (none)</td><td>CH_ABT</td></tr>
<tr><td>23</td><td>CH_R_END</td><td>EXEC_MSPM_IS(Ec1c2, SndSRCSDU)

=> V(Nmp_n) = 1
SndSRCSDU:=
BUILD_PDU_BMSG(MF_S_RR,0,V(Nmr_n),V(Nmp_n))</td><td>CH_R_END</td></tr>
<tr><td>24</td><td>CH_R_END</td><td>EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)

/ Rcv_sts = OK
&& GET_FT(RcvSRCSDU) = FT_MSG
&& { (GET_MC_FMT(RcvSRCSDU) = MF_S_RR) ||
(GET_MC_FMT(RcvSRCSDU) = MF_S_RNR) }

=> Node_ID:= GET_SA(RcvSRCSDU)
DLSDU:= GET_DLSDU(Ec1c2)
Length:= GET_DLSDU_LEN(DLSDU)
DL-SDA.ind{ SAP_ID, Rcv_sts , Node_ID, Length, DLSDU}</td><td>CH_IDLE_M</td></tr>
<tr><td>25</td><td>CH_R_END</td><td>EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)

/ Rcv_sts = OK
&& GET_FT(RcvSRCSDU) = FT_MSG
&& GET_MC_FMT(RcvSRCSDU) = MF_S_REJ

=> V(Nms_n):= 0, V(Nmr_n):= 0</td><td>CH_IDLE_M</td></tr>
<tr><td>26</td><td>CH_R_END</td><td>EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)

/ Rcv_sts = OK
&& GET_FT(RcvSRCSDU) = FT_MSG
&& GET_MC_FMT(RcvSRCSDU) = MF_I

=> (none)</td><td>CH_R_END</td></tr>
<tr><td>27</td><td>CH_R_END</td><td>DL-ABORT-SDA.req { SAP_ID }

=> (none)</td><td>CH_ABT</td></tr>
<tr><td>28</td><td>CH_ABT</td><td>EXEC_MSPM_IS(Ec1c2, SndSRCSDU)

=> V(Nmp_n) = 1
SndSRCSDU:= BUILD_PDU_BMSG(MF_S_REJ,0,0,V(Nmp_n))</td><td>CH_ABT</td></tr>
<tr><td>29</td><td>CH_ABT</td><td>EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)

/ Rcv_sts = OK
&& GET_FT(RcvSRCSDU) = FT_MSG
&& { (GET_MC_FMT(RcvSRCSDU) = MF_S_RR) ||
(GET_MC_FMT(RcvSRCSDU) = MF_I) }

=> (none)</td><td>CH_ABT</td></tr>
</table>

No.	Current state	Event /condition =>action	Next state

30	CH_ABT	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU)
		
		/ Rcv_sts = OK
		&& GET_FT(RcvSRCSDU) = FT_MSG
		&& { (GET_MC_FMT(RcvSRCSDU) = MF_S_RNR) ||
		(GET_MC_FMT(RcvSRCSDU) = MF_S_REJ) }
		
		=> V(Nms_n):= 0, V(Nmr_n):= 0
		DL-ABORT-SDA.cnf
		
		if (V(Fmsg_sending_n) = True) then
		DL-SDA.cnf { SND_ABT }
		endif
			CH_IDLE_M

31	Any state	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) <> FT_MSG => (none)	Same state
32	Any state	EXEC_MSPM_IR(Ec1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts <> OK => (none)	Same state

6.2.3.3.2.2 Responder for basic format DLPDU

Figure 31 and Table 65 show the state diagram and the state table of a responder of the message segmentation machine when DLE adopts basic format DLPLDU.

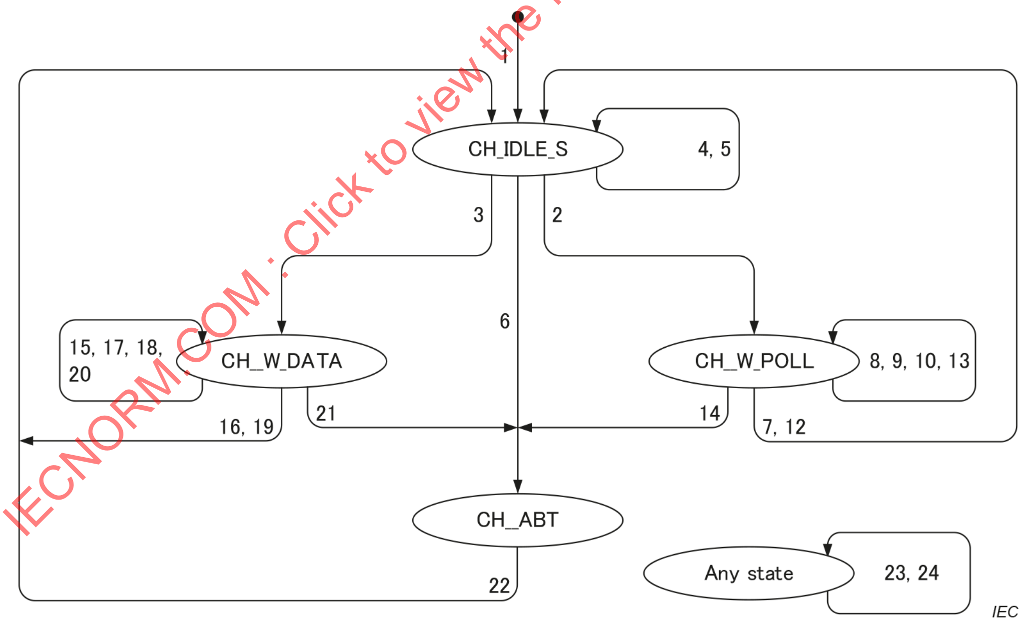


Figure 31 – The state diagram of message responder for basic format

Table 65 – The state table of message responder for basic format

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => V(Nms_n):= 0 V(Nmr_n):= 0	CH_IDLE_S
2	CH_IDLE_S	DL-SDA.req{SAP_ID,Node_ID,Length,SndDLSDU} => V(Nms_n):= 0 STORE_DLSDU(Ec1c2, Node_ID, SndDLSDU) V(Nmsg_len_n):= Length V(Nmsg_rem_len_n):= Length V(Fmsg_sending_n):= True	CH_W_POLL
3	CH_IDLE_S	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I && GET_MC_NS(RcvSRCSDU) = 0 && GET_MC_F(RcvSRCSDU) = 0 => V(Nmr_n):= 1 V(Fmsg_sending_n):= False STORE_PDU_BMSG(Ec1c2, RcvSRCSDU) V(Nmr_n):= 1 V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RR,0,V(Nmr_n),V(Nmf_n))	CH_W_DATA
4	CH_IDLE_S	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I && GET_MC_NS(RcvSRCSDU) = 0 && GET_MC_F(RcvSRCSDU) = 1 => V(Nmr_n):= 1 V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RNR,0,V(Nmr_n),V(Nmf_n)) STORE_PDU_MSG(Ec1c2, RcvSRCSDU) Node_ID:= GET_SA(RcvSRCSDU) DLSDU:= GET_DLSDU(Ec1c2) Length:= GET_DLSDU_LEN(DLSDU) DL-SDA.ind{ SAP_ID, Rcv_sts , Node_ID, Length, DLSDU}	CH_IDLE_S
5	CH_IDLE_S	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && CHK_MCTL_IS(RcvSRCSDU,MF_I) <> OK => V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RNR,0,V(Nmr_n),V(Nmf_n))	CH_IDLE_S
6	CH_IDLE_S	DL-ABORT-SDA.req { SAP_ID } => V(Fmsg_sending_n):= False	CH_ABT
7	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR && GET_MC_NR(RcvSRCSDU) = V(Nms_n) && V(Nmsg_rem_len_n) = 0 => V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RNR,0,V(Nmr_n),V(Nmf_n)) DL-SDA.cnf{ SND_OK }	CH_IDLE_S

No.	Current state	Event /condition =>action	Next state
8	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR && GET_MC_NR(RcvSRCSDU) = V(Nms_n) && V(Npkt_len_n) >= V(Nmsg_rem_len_n) => V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_I,V(Nms_n),V(Nmr_n),V(Nmf_n)) V(Nmsg_rem_len_n):= 0	CH_W_POLL
9	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR && GET_MC_NR(RcvSRCSDU) = V(Nms_n) && V(Npkt_len_n) < V(Nmsg_rem_len_n) => V(Nmf_n) = 0 SndSRCSDU:= BUILD_PDU_BMSG(MF_I,V(Nms_n),V(Nmr_n),V(Nmf_n)) V(Nms_n) = V(Nms_n) + 1 V(Nmsg_rem_len_n):= V(Nmsg_rem_len_n) - V(Npkt_len_n)	CH_W_POLL
10	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR && GET_MC_NR(RcvSRCSDU) <> V(Nms_n) => SndSRCSDU:= BUILD_PDU_BMSG(MF_I,V(Nms_n)-1,V(Nmr_n),V(Nmf_n))	CH_W_POLL
11	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RNR => SndSRCSDU:= BUILD_PDU_BMSG(MF_I,V(Nms_n)-1,V(Nmr_n),V(Nmf_n))	CH_W_POLL
12	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_REJ => V(Nms_n):= 0, V(Nmr_n):= 0, V(Nmf_n) = 1 DL-SDA.cnf{ SND_ABT } SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RNR,0,V(Nmr_n),V(Nmf_n))	CH_IDLE_S
13	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I => V(Nmf) = 1 SndSRCSDU:= BUILD_PDU_BMSG(MF_S_RR,0,V(Nmr_n),V(Nmf_n))	CH_W_POLL
14	CH_W_POLL	DL-ABORT-SDA.req { SAP_ID } => (none)	CH_ABT

No.	Current state	Event /condition =>action	Next state
15	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I && GET_MC_NS(RcvSRCSDU) = V(Nmr_n) && GET_MC_F(RcvSRCSDU) = 0 => STORE_PDU_BMSG(Ec1c2, RcvSRCSDU) V(Nmr_n):= V(Nmr_n) + 1 V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_MSG(MF_S_RR,0,V(Nmr_n),V(Nmf_n))	CH_W_DATA
16	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I && GET_MC_NS(RcvSRCSDU) = V(Nmr_n) && GET_MC_F(RcvSRCSDU) = 1 => V(Nmr_n):= V(Nmr_n) + 1 V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_MSG(MF_S_RNR,0,V(Nmr_n),V(Nmf_n)) STORE_PDU_MSG(Ec1c2, RcvSRCSDU) Node_ID:= GET_SA(RcvSRCSDU) DLSDU:= GET_DLSDU(Ec1c2) Length:= GET_DLSDU_LEN(DLSDU) DL-SDA.ind{ SAP_ID, Rcv_sts , Node_ID, Length, DLSDU}	CH_IDLE_S
17	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_I && GET_MC_NS(RcvSRCSDU) <> V(Nmr_n) =>V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_MSG(MF_S_RR,0,V(Nmr_n),V(Nmf_n))	CH_W_DATA
18	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RR => V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_MSG(MF_S_RR,0,V(Nmr_n),V(Nmf_n))	CH_W_DATA
19	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_REJ => V(Nms_n):= 0, V(Nmr_n):= 0, V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_MSG(MF_S_RNR),0,V(Nmr_n),V(Nmf_n))	CH_IDLE_S
20	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) = FT_MSG && GET_MC_FMT(RcvSRCSDU) = MF_S_RNR => V(Nms_n):= 0, V(Nmr_n):= 0, V(Nmf_n) = 1 SndSRCSDU:= BUILD_PDU_MSG(MF_S_RR),0,V(Nmr_n),V(Nmf_n))	CH_IDLE_S
21	CH_W_DATA	DL-ABORT-SDA.req { SAP_ID } => (none)	CH_ABT

No.	Current state	Event /condition =>action	Next state

22	CH_ABT	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU)
		/ Rcv_sts = OK
		=> V(Nms_n):= 0, V(Nmr_n):= 0, V(Nmf_n) = 1
		DL-ABORT-SDA.cnf
		if (V(Fmsg_sending_n) = True) then
		DL-SDA.cnf{SND_ABT}
		endif
		SndSRCSDU:= BUILD_PDU_MSG(MF_S_REJ,0,0,V(Nmf_n))
			CH_IDLE_S

23	Any state	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && GET_FT(RcvSRCSDU) <> FT_MSG => (none)	Same state
24	Any state	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts <> OK => (none)	Same state

6.2.3.3.3 Segmentation for short format DLPDU

6.2.3.3.3.1 Initiator for short format DLPDU

Figure 32 and Table 66 show the state diagram and the state table of an initiator of the message segmentation machine when DLE adopts the short format DLPDU.

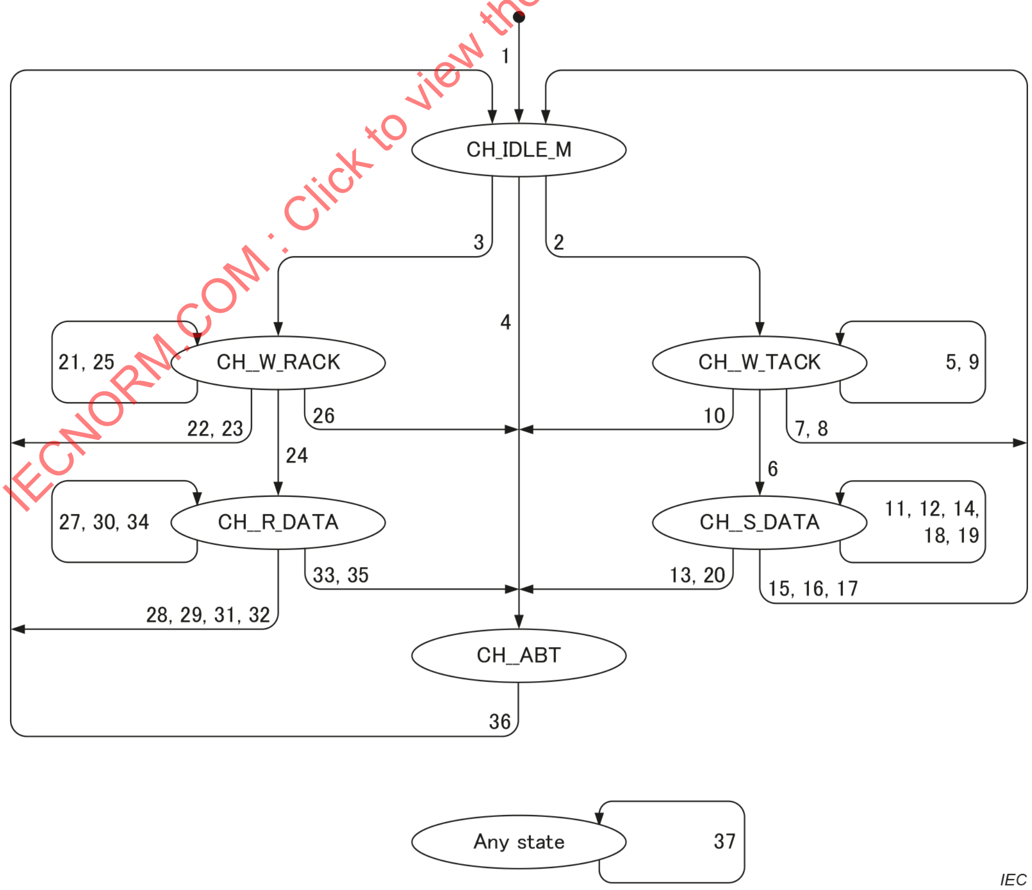


Figure 32 – The state diagram of message initiator for short format

Table 66 – The state table of message initiator for short format

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => (none)	CH_IDLE_M
2	CH_IDLE_M	DL-SDA.req{SAP_ID,Node_ID,Length,SndDLSDU} => V(Nmno_n) = GET_SMSG_NUM(Length) STORE_DLSDU(Ec1c2, Node_ID, SndDLSDU) V(Fmsg_sending_n):= True	CH_W_TACK
3	CH_IDLE_M	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) V(Fmsg_sending_n):= False	CH_W_RACK
4	CH_IDLE_M	DL-ABORT-SDA.req{SAP_ID,Node_ID} => V(Fmsg_sending_n):= False	CH_ABT
5	CH_W_TACK	EXEC_MSPM_IS(Fc1c2, SndSRCSDU) => V(Nmsd_n):= 0, V(Nmend_n):= 0, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_W_TACK
6	CH_W_TACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => (none)	CH_S_DATA
7	CH_W_TACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = ABT_NUM => DL-SDA.cnf{ SND_ABT }	CH_IDLE_M
8	CH_W_TACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) <> ABT_NUM => DL-SDA.cnf{ SND_ERR }	CH_IDLE_M
9	CH_W_TACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) => (none)	CH_W_TACK
10	CH_W_TACK	DL-ABORT-SDA.req{SAP_ID,Node_ID} => (none)	CH_ABT
11	CH_S_DATA	EXEC_MSPM_IS(Fc1c2, SndSRCSDU) / (V(Nmno_n)) > 1 => V(Nmsd_n):= 1, V(Nmend_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_S_DATA
12	CH_S_DATA	EXEC_MSPM_IS(Fc1c2, SndSRCSDU) / (V(Nmno_n)) = 1 => V(Nmsd_n):= 1, V(Nmend_n):= 1 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_S_DATA

No.	Current state	Event /condition =>action	Next state
13	CH_S_DATA	EXEC_MSPM_IS(Fc1c2, SndSRCSDU) / (V(Nmno)) < 1 => (none)	CH_ABT
14	CH_S_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => (none)	CH_S_DATA
15	CH_S_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = ABT_NUM => DL-SDA.cnf{ SND_ABT }	CH_IDLE_M
16	CH_S_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) <> ABT_NUM => DL-SDA.cnf{ SND_ERR }	CH_IDLE_M
17	CH_S_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) = V(Nms_n) && V(Nmno_n) = 1 => DL-SDA.cnf{ SND_OK }	CH_IDLE_M
18	CH_S_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) = V(Nms_n) => V(Nms_n):= V(Nms_n) + 1 V(Nmno_n):= V(Nmno_n) – 1	CH_S_DATA
19	CH_S_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) <> V(Nms_n) => (none)	CH_S_DATA
20	CH_S_DATA	DL-ABORT-SDA.req{SAP_ID,Node_ID} => (none)	CH_ABT
21	CH_W_RACK	EXEC_MSPM_IS(Fc1c2, SndSRCSDU) => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_W_RACK
22	CH_W_RACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => (none)	CH_IDLE_M

No.	Current state	Event /condition =>action	Next state
23	CH_W_RACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) = ABT_NUM => (none)	CH_IDLE_M
24	CH_W_RACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) <> ABT_NUM => (none)	CH_R_DATA
25	CH_W_RACK	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) => (none)	CH_W_RACK
26	CH_W_RACK	DL-ABORT-SDA.req{SAP_ID,Node_ID} => (none)	CH_ABT
27	CH_R_DATA	EXEC_MSPM_IS(Fc1c2, SndSRCSDU) => V(Nmsd_n):= 0, V(Nmend_n):= 1 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_R_DATA
28	CH_R_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => (none)	CH_IDLE_M
29	CH_R_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) = ABT_NUM => (none)	CH_IDLE_M
30	CH_R_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) <> ABT_NUM => (none)	CH_R_DATA
31	CH_R_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) = V(Nms_n) && (CHK_END(RcvSRCSDU) = 1) => Node_ID:= RcvSRCSDU.SA STORE_PDU_MSG(Ec1c2, RcvSRCSDU) DLSDU:= GET_DLSDU(Ec1c2)) Length:= GET_DLSDU_LEN(DLSDU) DL-SDA.ind{ SAP_ID, Rcv_sts , Node_ID, Length, DLSDU}	CH_IDLE_M

No.	Current state	Event /condition =>action	Next state
32	CH_R_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) = V(Nms_n) && (CHK_END(RcvSRCSDU) = 0) && CHK_MRBUF() = OK => V(Nms_n):= V(Nms_n) + 1 STORE_PDU_MSG(Ec1c2, RcvSRCSDU)	CH_IDLE_M
33	CH_R_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) = V(Nms_n) && (CHK_END(RcvSRCSDU) = 0) && CHK_MRBUF() <> OK => (none)	CH_ABT
34	CH_R_DATA	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1) && CHK_S(RcvSRCSDU) <> V(Nms_n)) => (none)	CH_R_DATA
35	CH_R_DATA	DL-ABORT-SDA.req{SAP_ID,Node_ID} => (none)	CH_ABT

36	CH_ABT	EXEC_MSPM_IS(Fc1c2, SndSRCSDU)
		=> V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM
		SndSRCSDU:=
		BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))
		DL-ABORT-SDA.cnf
		if (V(Fmsg_sending_n) = True) then
		DL-SDA.cnf{SND_ABT}
		endif
			CH_IDLE_M

37	Any state	EXEC_MSPM_IR(Fc1c2, Rcv_sts, RcvSRCSDU) / Rcv_sts <> OK => (none)	Same state

6.2.3.3.3.2 Responder for short format DLPDU

Figure 33 and Table 67 show the state diagram and the state table of a responder of the message segmentation machine when DLE adopts short format DLPDU.

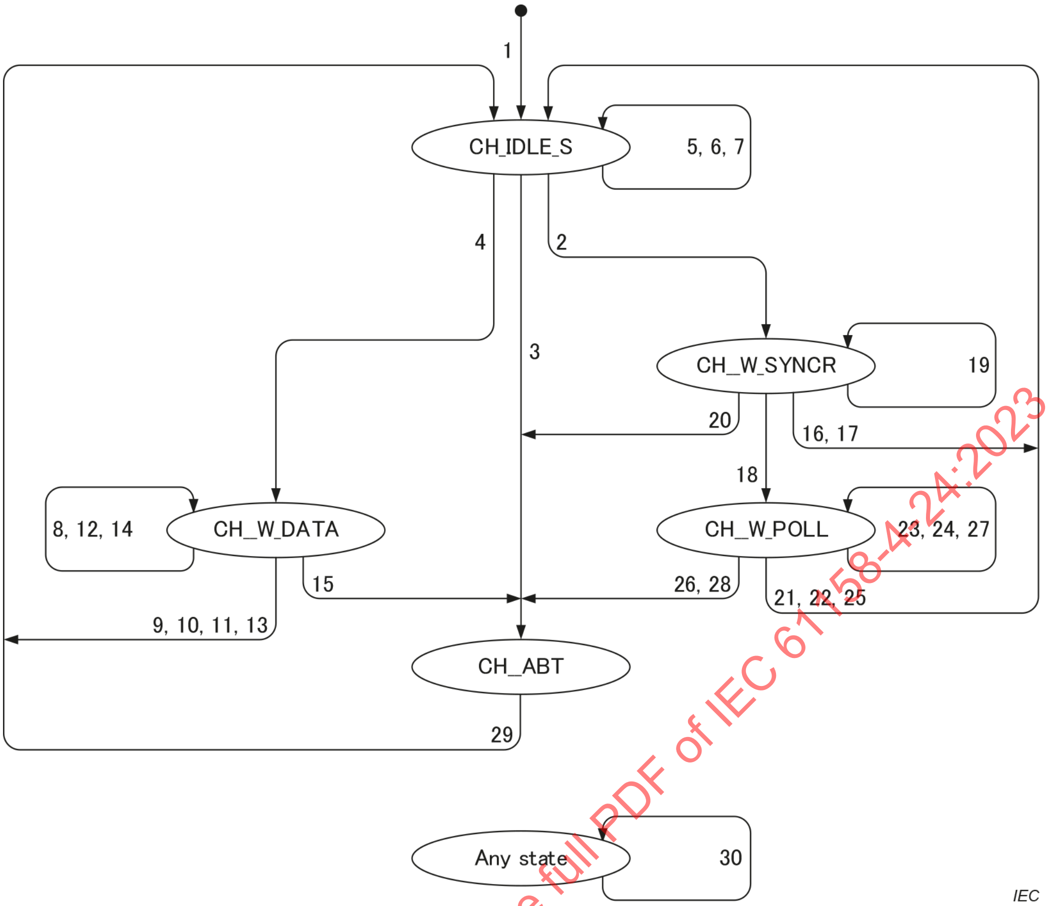


Figure 33 – The state diagram of message responder for short format

Table 67 – The state table of message responder for short format

No.	Current state	Event /condition =>action	Next state
1	Any states	Power on or CTC_Reset.req => (none)	CH_IDLE_S
2	CH_IDLE_S	DL-SDA.req{SAP_ID,Node_ID,Length,SndDLSDU} => V(Nmno) = GET_SMSG_NUM(Length) STORE_DLSDU(Ec1c2, Node_ID, SndDLSDU) V(Fmsg_sending_n):= True	CH_W_SYNCR
3	CH_IDLE_S	DL-ABORT-SDA.req{SAP_ID,Node_ID} => V(Fmsg_sending_n):= False	CH_ABT
4	CH_IDLE_S	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => V(Nmsd_n):= 0, V(Nmend_n):= 0, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) V(Fmsg_sending_n):= False	CH_W_DATA

No.	Current state	Event /condition =>action	Next state
5	CH_IDLE_S	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = ABT_NUM => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_IDLE_S
6	CH_IDLE_S	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) <> ABT_NUM => V(Nmsd_n):= 0, V(Nmend_n):= 0, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_IDLE_S
7	CH_IDLE_S	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 => V(Nmsd_n):= 0, V(Nmend_n):= 0, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_IDLE_S
8	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => SndSRCSDU:= BUILD_PDU_LAST_SMSG()	CH_W_DATA
9	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = ABT_NUM => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_IDLE_S
10	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) <> ABT_NUM => SndSRCSDU:= BUILD_PDU_LAST_SMSG()	CH_IDLE_S
11	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = V(Nms_n) && CHK_END(RcvSRCSDU) = 1 => V(Nmsd_n):= CHK_SD(RcvSRCSDU) V(Nmend_n):= CHK_END(RcvSRCSDU) V(Nms_n):= CHK_S(RcvSRCSDU) SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) STORE_PDU_MSG(Ec1c2, RcvSRCSDU) Node_ID:= RcvSRCSDU.SA DLSDU:= GET_DLSDU(Ec1c2)) Length:= GET_DLSDU_LEN(DLSDU) DL-SDA.ind{ SAP_ID, Rcv_sts , Node_ID, Length, DLSDU}	CH_IDLE_S

No.	Current state	Event /condition =>action	Next state
12	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = V(Nms_n) && CHK_END(RcvSRCSDU) = 0 && CHK_MRBUF() = OK => V(Nmsd_n):= CHK_SD(RcvSRCSDU) V(Nmend_n):= CHK_END(RcvSRCSDU) V(Nms_n):= CHK_S(RcvSRCSDU) SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) V(Nms_n):= V(Nms_n) + 1 STORE_PDU_MSG(Ec1c2, RcvSRCSDU)	CH_W_DATA
13	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = V(Nms_n) && CHK_END(RcvSRCSDU) = 0 && CHK_MRBUF() <> OK => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_IDLE_S
14	CH_W_DATA	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) <> V(Nms_n) => SndSRCSDU:= BUILD_PDU_LAST_SMSG()	CH_W_DATA
15	CH_W_DATA	DL-ABORT-SDA.req{SAP_ID,Node_ID} => (none)	CH_ABT
16	CH_W_SYNCR	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => SndSRCSDU:= BUILD_PDU_LAST_SMSG() DL-SDA.cnf{ SND_ERR }	CH_IDLE_S
17	CH_W_SYNCR	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = ABT_NUM => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) DL-SDA.cnf{ SND_ABT }	CH_IDLE_S
18	CH_W_SYNCR	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) <> ABT_NUM => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_W_POLL
19	CH_W_SYNCR	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= 0 SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))	CH_W_SYNCR

No.	Current state	Event /condition =>action	Next state
20	CH_W_SYNCR	DL-ABORT-SDA.req{SAP_ID,Node_ID} => (none)	CH_ABT
21	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 0 => SndSRCSDU:= BUILD_PDU_LAST_SMSG() DL-SDA.cnf{ SND_ERR }	CH_IDLE_S
22	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && (CHK_S(RcvSRCSDU) = ABT_NUM => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) DL-SDA.cnf{ SND_ABT }	CH_IDLE_S
23	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 0 && CHK_END(RcvSRCSDU) = 1 && (CHK_S(RcvSRCSDU) <> ABT_NUM => SndSRCSDU:= BUILD_PDU_LAST_SMSG()	CH_W_POLL
24	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = V(Nms_n) && V(Nmno_n) > 1 => V(Nmsd_n):= CHK_SD(RcvSRCSDU) V(Nmend_n):= CHK_END(RcvSRCSDU) V(Nms_n):= CHK_S(RcvSRCSDU) BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) V(Nms_n):= V(Nms_n) + 1 V(Nmno_n):= V(Nmno_n) – 1	CH_W_POLL
25	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = V(Nms_n) && V(Nmno_n) = 1 => V(Nmsd_n):= CHK_SD(RcvSRCSDU) V(Nmend_n):= 0 V(Nms_n):= CHK_S(RcvSRCSDU) BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) DL-SDA.cnf{ SND_OK }	CH_IDLE_S
26	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) = V(Nms_n) && V(Nmno_n) < 1 => V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM SndSRCSDU:= BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n)) DL-SDA.cnf{ SND_ERR }	CH_ABT
27	CH_W_POLL	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts = OK && CHK_SD(RcvSRCSDU) = 1 && CHK_S(RcvSRCSDU) <> V(Nms_n) => SndSRCSDU:= BUILD_PDU_LAST_SMSG()	CH_W_POLL

No.	Current state	Event /condition =>action	Next state
28	CH_W_POLL	DL-ABORT-SDA.req{SAP_ID,Node_ID} => (none)	CH_ABT

29	CH_ABT	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU)
		=> V(Nmsd_n):= 0, V(Nmend_n):= 1, V(Nms_n):= ABT_NUM
		SndSRCSDU:=
		BUILD_PDU_SMSG(V(Nmsd_n),V(Nmend_n),V(Nms_n))
		DL-ABORT-SDA.cnf
		if (V(Fmsg_sending_n) = True) then
		DL-SDA.cnf{SND_ABT}
		endif
			CH_IDLE_S

30	Any state	EXEC_MSPM_R(Fc1c2, Rcv_sts, RcvSRCSDU, SndSRCSDU) / Rcv_sts <> OK => (none)	Same state

6.2.3.3.4 Functions used by message segmentation protocol machine

Table 68 shows the list of functions which is used by the message segmentation machine.

Table 68 – List of functions used by the message segmentation machine

Function name	Parameter		Return	Operation
	Input	Output	Value	
STORE_DLSDU	Ec1c2, Node_ID, DLS-user_message	None	none	This function stores the DLS-user data of the message into the internal buffer.
GET_DLSDU	Ec1c2	None	DLS-user_message	This function retrieves the DLS-user message received from the remote station.
GET_DLSDU_LEN	DLS-user_message	None	length	This function retrieves the data length of the DLS-user message received from the remote station.
BUILD_PDU_BMSG	mframe_type, Ns, Nr, flag	None	SRCSDU	This function builds the SRCSDU to be sent. The input parameters are the value that to be contained the MCTL field of the message frame.
STORE_PDU_BMSG	Ec1c2, SRCSDU	None	none	This function stores the received SRCSDU into the internal buffer to assemble DLSPDU.
GET_SA	SRCPDU	None	Node_ID	This function takes out the value of the source address field of the specified SRCSDU.
GET_MC_FMT	SRCSDU	None	mframe_type	This function takes out the message type from the MCTL field of the specified SRCSDU.
GET_MC_NR	SRCSDU	None	Nr	This function takes out the value of Nr from the MCTL field of the specified SRCSDU.