EtherCAT 规范

第四部分：数据链路层协议规范

ETG.1000.4 S(R)V1.0.2

ETG 号	ETG.1000.4
类型	S (规范)
状态	R (颁布)
版本	V1.0.2
中文版本号	C01

作者	ETG
联系方式	info@ethercat.org
英文文件名	ETG1000_4_S_R_V1i0i2_EcatDLLServices.doc
日期	6.24.2013
中文文件名	ETG1000_4_CHN_EcatDLLServices_C01.doc

EtherCAT Technology Group

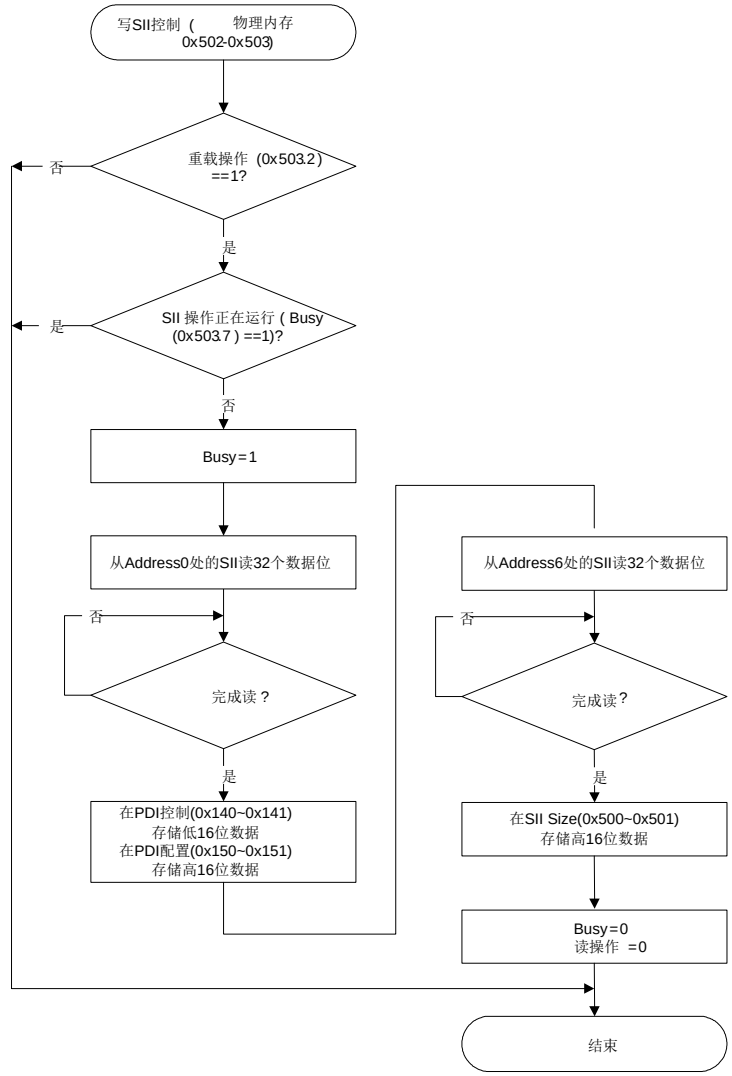
原文修改履历

版本号	注释
1.0	引用于只供 ETG 使用的 IEC-规范 61158-2 Type 12 部分
1.0.1	修正问题与澄清，请查看 ETG1000_ES_D_V0i6_EcatSpecErrata.pdf。该文件没有修改点。
1.0.2	修正问题与澄清，请查看 ETG1000_V1i0i2_ES_R_V0i7_EcatSpecErrata.doc 与 IEC SC65C MT 9 编辑注释。该文件没有修改点。

中文版修改履历

版本号	注释
C01	初版

目录



1 范围

1.1 本标准的范围以及所对应的 IEC 标准

ETG.1000 系列文件是在 EtherCAT Technology group 范围内对 EtherCAT Technology 详细说明。它分为以下几个部分：

- ETG.1000.2：物理层服务定义和协议规范。
- ETG.1000.3：数据链路层服务定义。
- ETG.1000.4：数据链路层协议规范。
- ETG.1000.5：应用层服务定义。
- ETG.1000.6：应用层协议规范。

以上各文件依赖于 IEC 61158 系列文件 Type12 中相一致的部分。为了回避商标使用权的问题，EtherCAT 在 IEC 61158 中被称作 Type12。

1.2 概要

在自动化环境中，数据链路层提供设备之间的基本的时间关键通讯。

本协议中所规范的通信将适用于所有参与活动的实体。

a) 同步启动的周期通信

b) 对应数据链路各实体每周期要求，周期或非周期异步通信。

从而可知，此协议具有以下特点：提供周期或非周期的异步访问，但每个周期有一个同步的重启。

1.3 规范

本标准的规定：

a) 从一个数据链路层上的用户实体到一个或多个实体传输数据或控制信息的过程；

b) 用于本标准协议传输数据和控制信息的 DLPDUs 的结构以及作为物理接口数据单元的表示。

1.4 规程

通过以下的方式来定义各规程

a) 因为交换 DLPDU 而产生的 DL-实体(DLE)间的交互；

b) 在同一系统中因为 DLS 原语的交换而产生的一个 DL-服务(DLS)提供者和一个 DLS-用户之间的交互；

c) DSL- 提供者和 ISO/IEC 8802-3 的 MAC 服务之间的交互。

1.5 适用性

这些规程适用于如下两个系统间的通信实例：在 OSI 模型数据链路层的支持时间关键通信服务的系统和在开放系统互连环境中需要互连能力的系统间通信实例。

行规提供了一个总结设备能力的简单的、多属性的方法，以此方式可应用于不同的时间关键的通信需求。

1.6 一致性

本标准为系统实现这些规程也规定了一致性的要求。这部分标准不包含证明符合这些要求的测试。

2 参考文献

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅所注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

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

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

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

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

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

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

ISO/IEC 9899, *Programming Languages – C.*

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

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

IETF RFC 768, *User Datagram Protocol*; available at http://www.ietf.org

IETF RFC 791, *Internet Protocol darpa internet program protocol specification*; available at http://www.ietf.org

3 关于术语、定义、符号和缩略语

基于文件化的目的，定义如下术语，定义，符号和缩略语。

3.1 参考模型术语和定义

本部分基于 ISO/IEC 7498-1 与 ISO/IEC 7498-3 中的概念，并使用下列定义的术语：

3.1.1	**DL-双工传输**	**DL-duplex-transmission**	[7498-1]
3.1.2	**DL-协议**	**DL-protocol**	[7498-1]
3.1.3	**DL-协议数据单元**	**DL-protocol-data-unit**	[7498-1]
3.1.4	**(N) –实体** **DL-实体** **Ph-实体**	**(N)-entity** **DL-entity** **Ph-entity**	[7498-1]
3.1.5	**(N)-接口数据单元** **DL-服务数据单元** **Ph-接口数据单元**	**(N)-interface-data-unit** **DL-service-data-unit (N=2)** **Ph-interface-data-unit (N=1)**	[7498-1]
3.1.6	**(N) 层** **DL 层(N=2)** **Ph 层(N=1)**	**(N)-layer** **DL-layer (N=2)** **Ph-layer (N=1)**	[7498-1]
3.1.7	**(N) 服务** **DL 服务(N=2)** **Ph 服务(N=1)**	**(N)-service** **DL-service (N=2)** **Ph-service (N=1)**	[7498-1]
3.1.8	**(N) 服务访问点** **DL 服务访问点(N=2)** **Ph 服务访问点(N=1)**	**(N)-service-access-point** **DL-service-access-point (N=2)** **Ph-service-access-point (N=1)**	[7498-1]
3.1.9	**(N)服务访问点地址** **DL 服务访问点地址(N=2)** **Ph 服务访问点地址(N=1)**	**(N)-service-access-point-address** **DL-service-access-point-address (N=2)** **Ph-service-access-point-address (N=1)**	[7498-1]
3.1.10	**对等实体**	**peer-entities**	[7498-1]
3.1.11	**Ph 接口数据**	**Ph-interface-data**	[7498-1]
3.1.12	**原语名**	**primitive name**	[7498-3]
3.1.13	**重组**	**reassembling**	[7498-1]
3.1.14	**合流**	**recombining**	[7498-1]
3.1.15	**复位**	**reset**	[7498-1]
3.1.16	**路由**	**routing**	[7498-1]
3.1.17	**分段**	**segmenting**	[7498-1]
3.1.18	**排序**	**sequencing**	[7498-1]
3.1.19	**分流**	**splitting**	[7498-1]
3.1.20	**系统管理**	**systems-management**	[7498-1]

3.2 服务约定术语和定义

本部分还使用了用于数据链路层的 ISO/IEC 10731 中的下列术语定义：

3.2.1 不对称服务 asymmetrical service

3.2.2 证实(原语) confirm (primitive);
请求者.交付(原语) requestor.deliver (primitive)

3.2.3 交付(原语) deliver (primitive)

3.2.4 DL-服务-原语 DL-service-primitive;
原语 primitive

3.2.5 DL-服务-提供者 DL-service-provider

3.2.6 DL-服务-用户 DL-service-user

3.2.7 指示(原语) indication (primitive)
接受者.交付(原语) acceptor.deliver (primitive)

3.2.8 请求(原语) request (primitive);
请求者.提交(原语) requestor.submit (primitive)

3.2.9 请求者 requestor

3.2.10 响应(原语) response (primitive);
接受者.提交(原语) acceptor.submit (primitive)

3.2.11 提交(原语) submit (primitive)

3.2.12 对称服务 symmetrical service

3.3 通用术语和定义

注：很多定义用于多个协议类型，但并非对所有协议都是必要的。

以下定义也适用于本部分。

3.3.1

帧 Frame
DLPDU 的同义语

3.3.2

组 DL 地址 Group DL-address
潜在指定多个含有扩展链接的 DLSAP 的 DL 地址。单个 DL 实体可以有多个与单个 DLSAP 相关的组 DL 地址。单个 DL 实体也可以有单个与多个 DLSAP 相关的组 DL 地址。

3.3.3

节点 Node
出现在本地链路上的单个 DL 实体

3.3.4

接收 DLS 用户 receiving DLS-user
作为 DLS 用户数据接收者的 DL 服务用户

注：一个 DL 服务用户可以同时是发送和接收 DLS-user

3.3.5

发送 DLS-user　　sending DLS-user
作为 DL 用户数据源的 DL 服务用户

3.4 附加的 EtherCAT 定义

3.4.1

应用　　Application
生产或消费数据的函数或数据结构

3.4.2

应用对象　　Application objects
通过网络间和网络设备内来管理和提供信息交换的运行时的多个对象类

3.4.3

基本型从站　　basic slave
只支持数据物理寻址的从站设备

3.4.4

位　　Bit
由 1 和 0 组成的信息单元，是可发送的最小数据单元

3.4.5

客户端　　Client
1) 使用另一个对象的服务来执行任务的对象

2) 服务器所响应的消息的发起者

3.4.6

连接　　Connection
在相同或不同设备中，两个应用对象之间的逻辑绑定

3.4.7

周期　　Cyclic
以定期和重复模式进行重复的事件

3.4.8

循环冗余检查　　Cyclic Redundancy Check (CRC)
从数据阵列中计算出来并用来作为该阵列的检验码的余数

3.4.9

数据　　Data
泛指现场总线上传送的任何信息

3.4.10

数据一致性　　Data consistency
实现客户端与服务器之间及其内部对输入输出数据对象的一致性传输和访问的方法

3.4.11

设备　　Device
连接到至少由一个通信部件（网络部件）组成的现场总线的物理实体，并且该物理实体可能包含一个控制部件和/或一个终端部件（变送器，执行器等）

3.4.12

分布式时钟　　Distributed clocks
同步从站和维持一个全局时基的方法

3.4.13

误差　　Error
计算、观察、测量值或状态与指定的或理论上正确的值或状态之间差异

3.4.14

事件　　Event
条件发生变化的实例

3.4.15

现场总线内存管理单元　　Fieldbus Memory Management Unit
现场总线内存管理单元拥有在逻辑地址和物理内存间建立一个或多个对应关系的功能

3.4.16

现场总线内存管理单元实体　　Fieldbus Memory Management Unit entity
现场总线的内存管理单元的单个元素：一个相干的逻辑地址空间和一个连贯的物理内存位置之间的对应关系

3.4.17

完整型从站　　Full slave
同时支持数据的物理和逻辑寻址的从属设备

3.4.18

接口　　Interface
由功能特性、信号特性或其他适当的特性定义的两个功能单元之间的共享边界

3.4.19

主站　　Master
控制网络上的数据传送，通过发送报文来初始化对从站的媒体访问，并构成控制系统的接口的设备

3.4.20

映射　　Mapping
两个对象间的对应关系，使得一个对象成为另一对象的一部分

3.4.21

媒体　　Medium
在两点或多点间传输通信信号的电缆、光纤或其他介质

注：“media”是指“medium”的复数形式

3.4.22

报文　　Message
用于传送信息的一系列有序的八位位组

注：通常用于在应用层实体之间传递信息

3.4.23

网络　　Network
所有的媒体、连接器、中继器、路由器、网关和相关节点通信部件，用以实现一组通信设备互连

3.4.24

节点　　Node
一个网络的链路端点或两个及以上链路的交汇点[见 IEC 61158-2]

3.4.25

对象　　Object
设备内某一特定组件的抽象描述

注：一个对象可以是:

a) 一个设备能力的抽象描述，它由以下任一或所有部分组成：

1) 数据（随时间改变的信息）；

2) 组态（行为的参数）；

3) 方法（用数据和组态可以做的事情）；或者

b) 相关数据（以变量的形式）和操作这些数据的方法（程序）的集合。这些方法具有明确定义的接口和行为。

3.4.26

过程数据　　Process data
以测量和控制为目的的数据对象，其中包含指定周期传输或非周期传输的应用对象

3.4.27

服务器　　Server
为另一个（客户）对象提供服务的对象

3.4.28

服务　　Service
对象和/或对象类的执行是建立在另一个对象和/或对象类的请求基础上的操作或功能

3.4.29

从站　　Slave
只有在被主站或前面的从站启动后，才能执行媒体访问的 DL 实体

3.4.30

同步管理器　　Sync manager
控制单元的集合，用来协调对同时使用的对象的访问

3.4.31

同步管理器通道　　Sync manager channel
用来协调对同时使用的对象的访问的单个控制单元

3.4.32

交换机　　Switch
在 IEEE 802.1D 中定义的 MAC 桥

3.5 通用符号和缩写

注：很多符号和缩写为不止一个协议类型所共用，但并非对所有协议都是必须的。

DL-	数据链层（作为前缀）
DLC	DL-连接
DLCEP	DL-连接端
DLE	DL-实体(数据链层当前活动体)
DLL	DL-层
DLPCI	DL-协议-控制- 信息
DLPDU	协议数据单元
DLM	DL-管理
DLME	DL-管理实体（DL-管理当前活动体）
DLMS	DL-管理服务
DLS	DL-服务
DLSAP	DL-服务访问点
DLSDU	DL-服务数据单元
FIFO	先进先出（队列方法）
OSI	开发系统互连
Ph-	物理层（作为前缀）
PhE	Ph-实体
PhL	Ph-层
QoS	服务质量

3.6 追加 EtherCAT 符号和缩写

AL	Application layer	应用层
DLSDU	Data-link protocol data unit	DL 服务数据单元
APRD	Auto increment physical read	自增式物理读
APRW	Auto increment physical read write	自增式物理读/写
APWR	Auto increment physical write	自增式物理写
ARMW	Auto increment physical read multiple write	自增式物理读/多重写
BRD	Broadcast read	广播读
BRW	Broadcast read write	广播读/写
BWR	Broadcast write	广播写
CAN	Controller area network	控制器局域网
CoE	CAN application protocol over EtherCAT services	基于 EtherCAT 服务的 CAN 应用协议
CSMA/CD	Carrier sense multiple access with collision detection	带冲突检测的载波监听多路访问
DC	Distributed clocks	分布式时钟
DCSM	DC state machine	DC 状态机
DHSM	(DL)PDU handler state machine	(DL) PDU 处理程序状态机
EtherCAT	Prefix for DL services and protocols	DL 服务和协议的前缀
E²PROM	Electrically erasable programmable read only memory	电可擦可编程只读存储器

EoE	Ethernet tunneled over EtherCAT services	基于 EtherCAT 服务的以太网隧道
ESC	EtherCAT slave controller	EtherCAT 从站控制器
FCS	Frame chec sequence	帧检验序列
FMMU	Fieldbus memory management unit	现场总线内存管理单元
FoE	File access with EtherCAT services	基于 EtherCAT 服务的文件访问
FPRD	Configured address physical read	配置的地址物理读
FPRW	Configured address physical read write	配置的地址物理读/写
FPWR	Configured address physical write	配置的地址物理写
FRMW	Configured address physical read multiple write	配置的地址物理读/多重写
HDR	Header	帧头
ID	Identifier	标识符
IP	Internet protocol	因特网协议
LAN	Local area network	局域网
LRD	Logical memory read	逻辑存储器读
LRW	Logical memory read write	逻辑存储器读/写
LWR	Logical memory write	逻辑存储器写
MAC	Media access control	媒体访问控制
MDI	Media dependent interface(specified in ISO/IEC 8802-3)	媒体相关接口（见 ISO/IEC 8802-3 中规定）
MDX	Mailbox data exchange	邮箱数据交换
MII	Media independent interface(specified in ISO/IEC 8802-3)	媒体无关接口（见 ISO/IEC 8802-3 中规定）
PDI	Physical device interface(a set of elements that allows to DL services from the DLS-user)	物理设备接口（允许从 DLS 用户访问 DL 服务的一组单元）
PDO	Process data object	过程数据对象
PHY	Physical layer device(specifed in ISO/IEC 8802-3)	物理层设备（见 ISO/IEC 8802-3 中规定 ）
PNV	Publish network variable	发布网络变量
RAM	Random access memory	随机存取存储器
RMSM	Resilient mailbox state machine	恢复邮箱状态机
Rx	Receive	接收
SDO	Service data object	服务数据对象
SII	Slave information interface	从站信息接口
SIISM	SII state machine	SII 状态机
SyncM	Synchronization manager	同步管理器
SYSM	Sync manager state machine	同步管理器状态机
TCP	Transmission control protocol	传输控制协议
Tx	Transmit	发送
UDP	User datagram protocol	用户数据包协议
WKC	Working counter	工作计数器

3.7 约定

3.7.1 基本概念

以下服务均已在 ETG.1000.3 中进行了详细说明。服务规范对 EtherCAT DL 所提供的服务进行了定义。这些服务到 ISO/IEC 8802-3 的映射在本部分中进行描述。

本标准采用 ISO/ IEC10731 中所记叙的的描述性约定。

3.7.1.1 抽象语法约定

与 PDU 结构相关的各 DL 语法元素在表 1 中列出并进行了描述。

帧部分(Frame part) 中的元素在规范中具体定义。

数据区域(Data field)是元素的名称。

数据类型(Data Type)表示终端符号的类型。

值/描述(Value/Description)包含了常数值或参数的意义。

表 1 – PDU 元素表述实例

帧部分	数据区域	数据类型	值/描述
EtherCAT xxx	CMD	Unsigned8	0x01
	IDX	Unsigned8	索引
	ADP	Unsigned16	自增地址
	ADO	Unsigned16	物理内存地址
	LEN	Unsigned11	YYY 的数据长度(以八位位组为单位)
	Reserved	Unsigned4	0x00
	NEXT	Unsigned1	0x00: 最终 EtherCAT PDU 0x01: 后续 EtherCAT PDU
	IRQ	Unsigned16	保留为将来使用
	YYY		下一个元素
	WKC	Unsigned16	工作计数器

由 C 语言符号（GB/T 15272）描述的属性类型见图 1。BYTE 和 WORD 是 unsigned char 和 unsigned short 类型。

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

图 1 – 类型描述实例

属性本身的描述如表 2 所示。

参数描述了一个单一元素的属性。

物理地址表示在物理地址空间中的位置。

数据类型指出该元素的类型。

访问类型 EtherCAT DL/ PDI 表示对元素的访问权。R 表示只读访问权，W 表示写入访问权。如果 EtherCAT DL 和 PDI 对其均不拥有写入访问权，则该变量将由 DL 自身进行初始化和维护。

值/描述包含了常数值和/或者参数的意义。

表 2 – 属性描述实例

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
State	0x0120	Unsigned4	RW	R	0x01:初始化请求 0x02:预运行请求 0x03:引导状态模式请求 0x04:安全运行请求 0x08:运行请求
Acknowledge	0x0120	Unsigned1	RW	R	0x00:无确认 0x01 确认(必须是一个正沿)
Reserved	0x0120	Unsigned3	RW	R	0x00
Application Specific	0x0121	Unsigned8	RW	R	

3.7.1.2 保留位和八位位组的编码约定

术语 Reserved 可用于描述八位位组的位或整个八位位组。所有保留位或八位位组在发送端都应设为 0，并且不得在接收端进行测试，除非它被显式声明或者保留位或八位位组被状态机检查。

术语 Reserved 也可用于指明一个参数范围内的特定值是用于将来扩展用的。在这种情况下，保留值不应用在发送端，并且不得在接收端进行测试，除非它被显式声明或保留值被状态机检查。

3.7.1.3 特定字段八位位组的通用编码约定

DLSDU 可包含载有原始和压缩信息的特定字段。这些字段应按图 2 所示的顺序编码。

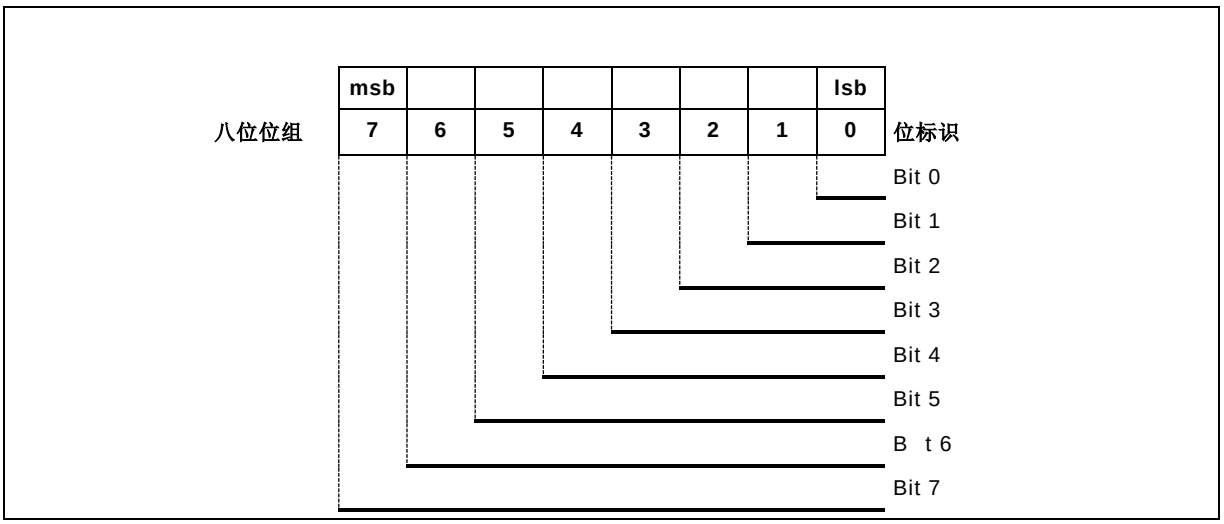


图 2 – 特定字段的公共结构

位可以组合成位组。每一个位或位组都应由位标识符寻址（如 Bit 0, Bit 1 到 4）。在八位位组内的位置应符合上图。位或位组可以用别名标识，也可标注为保留。分组数据位应无间隙升序。位组值可表示为二进制、十进制或十六进制值，此值仅对位组有效，如果所有的 8 位均在组中的话，则只能代表整个八位位组。十进制或十六进制的值应当转换为二进制值，这样，组中最高编号的位代表位组的 MSB。

例 1: 指定字段八位位组的描述和关系。

Bit 0: Reserved。

Bit 1-3: Reason_Code，十进制值 2 意味着一般错误。

Bit 4-7: 固定设置为 1。

根据上面的描述，构造的八位位组，如下所示：

(msb) Bit 7 = 1,

Bit 6 = 1,

Bit 5 = 1,

Bit 4 = 1,

Bit 3 = 0,

Bit 2 = 1,

Bit 1 = 0,

(lsb) Bit 0 = 0.

此位组有一个八位位组的值 0xF4。

3.7.2 状态机约定

协议时序以状态机的方式描述。

在状态图中状态表示为框，状态转换表示为箭头，状态名和状态图迁移与在状态迁移表中的名称相对应。

状态转换文本列表结构如下，见表 3。

第一行包含转换的名称。

在第二行定义当前状态。

第三行包含一个可选事件，紧跟着以“/”作为开始字符的条件，最后是以“=>”作为开始字符的动作。

最后一行包含下一个状态。

如果事件发生并满足条件，转换激发，即：执行动作，并进入下一个状态。

状态机描述布局如表 3 所示。状态机描述的各个元素的含义如表 4 所示。

表 3 – 状态机描述元素

#	当前状态	事件 /条件 =>动作	下一个状态

表 4 – 状态机元素的描述

描述元素	含义
当前状态 下一个状态	状态的名称
#	状态转换的名称或编号。
事件	事件的名称或描述。
/条件	布尔表达式。前面的“\”不是条件的一部分。
=>动作	任务、服务或函数调用的列表。前面的“=>”不是动作的一部分

在状态机中使用的约定如表 5 所示。

表 5 – 状态机中使用的约定

约定	含义
=	左侧项的值被右侧项的值替换，如果右侧项的值是参数，来自于原语，可看作一个输入事件
axx	如果 a 是一个字母，则表示一个参数的名称。 例如： Identifier = reason 意思是 reason 参数值被赋值给叫 Identifier 的参数。
"xxx"	表示固定的可见字符串。 例如： Identifier = "abc" 意思是值"abc"被赋值给一个叫 Identifier 的参数
nnn	如果所有的元素都是数值，该项表示以十进制表示的一个常数
0xnn	如果所有的元素都是数值，该项表示以十六进制表示的一个常数
==	逻辑条件，指示左侧项等于右侧项
<	逻辑条件，指示左侧项小于右侧项
>	逻辑条件，指示左侧项大于右侧项
!=	逻辑条件，指示左侧项不等于右侧项
&&	逻辑“与”
||	逻辑“或”
!	逻辑“非”
+ - * /	算数运算符
;	表达式的分隔符

为了理解协议机，我们强烈建议读者属性定义、本地功能以及 FDL-PDU 定义相关的条。我们假设读者对于这些定义都有足够的认识并在阅读时不需要任何额外的解释。

以 C 语言符号(GB/T 15272)进行定义的其他结构可用来描述条件和动作。

4 DL 协议概述

4.1 工作原理

EtherCAT DL 是实时以太网技术，旨在最大限度地利用全双工以太网带宽。媒体访问控制采用主站/从站原则，主站节点（典型的控制系统）发送以太网帧给从站节点，从站节点从这些帧中提取和插入数据。

从以太网的角度看，一个 EtherCAT 网段就是一个单个的以太网设备，它接收和发送标准的 ISO/IEC 8802-3 以太网帧。但这种以太网设备并不局限于带后方的微处理器的单个以太网控制器，它可能还包含大量的 EtherCAT 从站设备。这些从站设备直接处理到来的以太网帧，从中读取数据和/或插入自己的数据，并把帧传给下一个从站设备。网段内的最后一个从站设备沿着设备链反向发送完全处理的以太网帧，并通过第一个从站设备把收集的信息返回给主站，主站接收信息做为以太网响应帧。

此方法采用以太网全双工的模式：双方向的通信都是独立执行的，主站设备和由一个或多个从站设备构成的 EtherCAT 网段直接通信不需要使用交换机。

4.2 拓扑

通信系统的拓扑结构对自动化的成功应用是一个非常重要的因素。拓扑结构对布线、诊断特性、冗余选项和热插拔特性都有很大影响。

Ethernet 常用的星型拓扑结构可导致布线以及基础结构成本的增加。所以，尤其是对自动化应用，往往优先考虑总线型或树型拓扑结构。

从站节点的布置构成一个开环总线，在开环的一端，主站设备通过直连方式或者交换机发送数据帧。在另一端接收被处理的数据帧。从一个节点到下一个节点数据帧传输都有延时，数据帧从最后一个节点返回 PDU 到主站。利用以太网全双工能力，由此产生的拓扑结构是一个物理线型。

原则上，分支在任何地方都是可以的，它可以用来把总线型结构提升为树型结构，而树型结构支持很简单的布线；比如，单个的分支可以拓展到控制柜或机器模块，而主干线却只能从一个模块到下一个模块。

4.3 帧处理原则

要实现最高的性能，应以“on the fly”方式直接处理以太网帧。如果以这种方式实现，从站节点在帧通过从站时识别并执行相应的指令。

注：EtherCAT DL 能够通过标准以太网控制器实现而不直接处理。传输机制对传输性能的影响在 IEC 61784-2 中有相关描述。

节点都有通过读或写服务访问的可寻址内存，这种读或写服务可以是单节点连续或多节点同步访问。多个 EtherCAT PDU 可以嵌入到一个以太网帧中，每个 PDU 寻址一个聚合的数据段。EtherCAT 的 PDU 传输如图 3 所示：

a) 直接在以太网帧的数据区内，

b) 通过 IP 传输的 UDP 报文的数据段内。

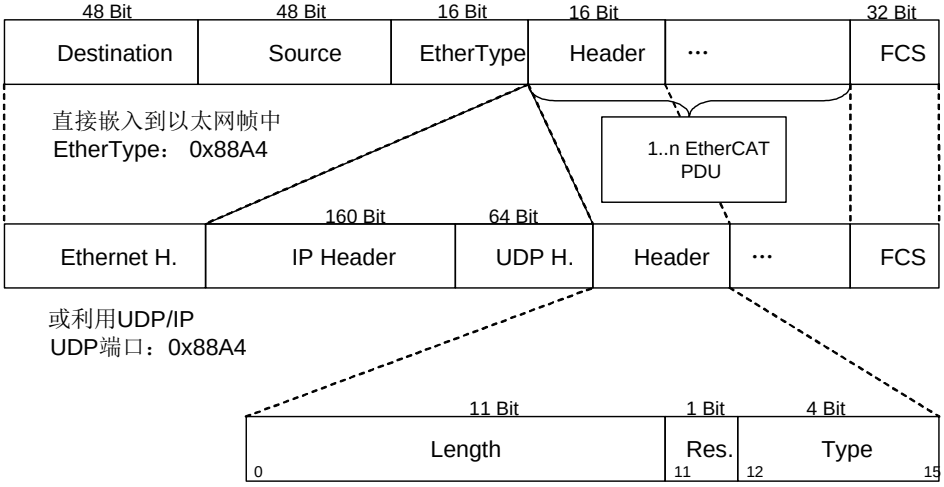


图 3 – 帧结构

变种 a) 只能在一个以太网子网中使用，因为组合的帧不能被路由器传送。在机械控制应用中，这种限制并不代表一种约束。多个 EtherCAT 网段被连接到一个或多个交换机。同一个网段的第一个节点的以太网 MAC 地址被用作 EtherCAT 网段寻址。

注意：更多关于寻址的详细描述请参考数据链路层服务定义（ETG.1000.3）

变种 b) 通过 UDP/IP 产生一个较长的协议头（IP 和 UDP 头），但对于楼宇自动化等非时间关键的应用允许使用 IP 路由选择。在主机端可以实现任何标准的 UDP/IP 传输协议。

4.4 数据链路层概述

通过一个以太网帧携带多个 EtherCAT PDU，多个节点可以被独立地寻址。被无间隙的打包到一个以太网帧中。最后一个 EtherCAT PDU 是帧的结尾，除非当帧的大小小于 64 字节时，在这种情况下帧必须填充到 64 字节。

相对于每个节点传输一帧数据，EtherCAT 能够更好的利用以太网的带宽。然而，例如一个只包含 2 比特用户数据的 2 通道数字输入节点，一个单个的 EtherCAT PDU 的开销仍然过多。

因此，从站节点同样可以支持逻辑地址映射。过程数据可以插入到逻辑地址空间内的任何地方。如果包含用于明确的过程数据映射区（位于相应逻辑地址）的读或写服务 EtherCAT PDU 被发送，而不是寻址特定节点，那么每个节点从正确的位置提取或插入过程数据，如图 4 所示。

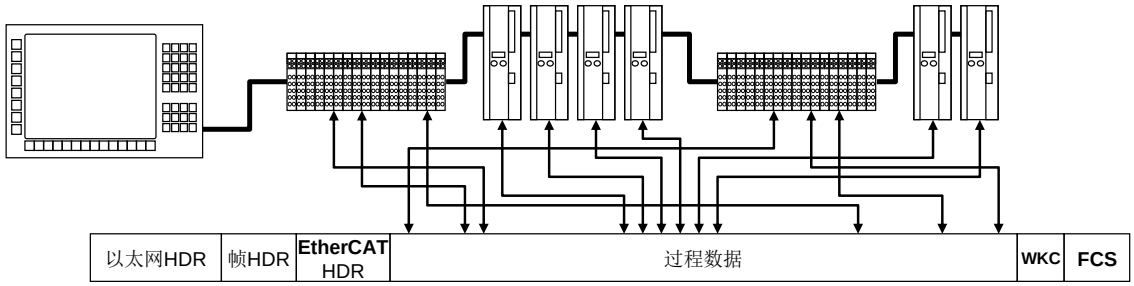


图 4 – 单个帧的数据映射

所有的节点都检测是否有地址与过程映像相匹配，如果匹配的话就插入它们的数据，这样多个节点可以被一个 EtherCAT PDU 同时寻址。主站可以通过单个的 EtherCAT PDU 得到完整的逻辑过程映像序列。主站不再需要额外的映射，所以过程数据可以直接分配到不同的控制任务中。每个任务都可以创建自己的过程映射并在自己的 Timeframe 内交换过程映像。节点的物理顺序完全是任意的，仅和首次初始化阶段相关。

逻辑地址空间大小为 2^{32} 八位位组（4GB）。EtherCAT 现场总线可以被认为一个用于自动化系统的串行背板，并使其在大的和非常小的自动化设备下都能连接到分布式过程数据。使用标准的以太网控制器和标准的以太网电缆，大量的 I/O 通道（无分配限制）可与自动化设备连接，因此 EtherCAT 具有高带宽，低延迟和最佳的有效可用数据传输率。同时，为了保留现存的技术和标准，像现场总线扫描仪这样的设备也可以连接。

4.5 错误检测概述

EtherCAT DL 通过以太网帧的检测序列(FCS)来判断一个帧是否被正确传输。由于一个或几个从站会在数据传输过程中修改以太网帧，因此在传播过程中，每个从站都全在接收时检查 FCS 并在发送时重新计算。如果检测到校验和错误，从站不进行 FCS 修改，而是通过增加错误计数来通知主站，确保在一个开环拓扑里能够精准地确定单一错误源的位置。

当对 EtherCAT DLPDU 读或写数据时，被寻址的从站要将位于 DLPDU 尾部的工作计数器(WKC)递加。通过分析 WKC，主站能检测出期望的节点号是否已经处理过相应的 DLPDU。

4.6 节点参考模型

4.6.1 映射到 OSI 基本参考模型

使用 ISO/IEC 7498 信息处理系统-开放系统互连-基本参考模型（OSI）中的原理、方法和模型描述了 EtherCAT DL。OSI 模型为通信标准提供了一种各个层可以独立开发和修改的分层方法。在 EtherCAT DL 规范中按自顶向下定义了完整的 OSI 协议栈的功能和协议栈的用户功能。OSI 的中间 3～6 层的功能被并入到 EtherCAT DL 数据链路层或 EtherCAT DL 应用层。同样的，那些现场总线应用层的用户常规特性由 EtherCAT 应用层提供，以便简化用户操作，如图 5 所示。

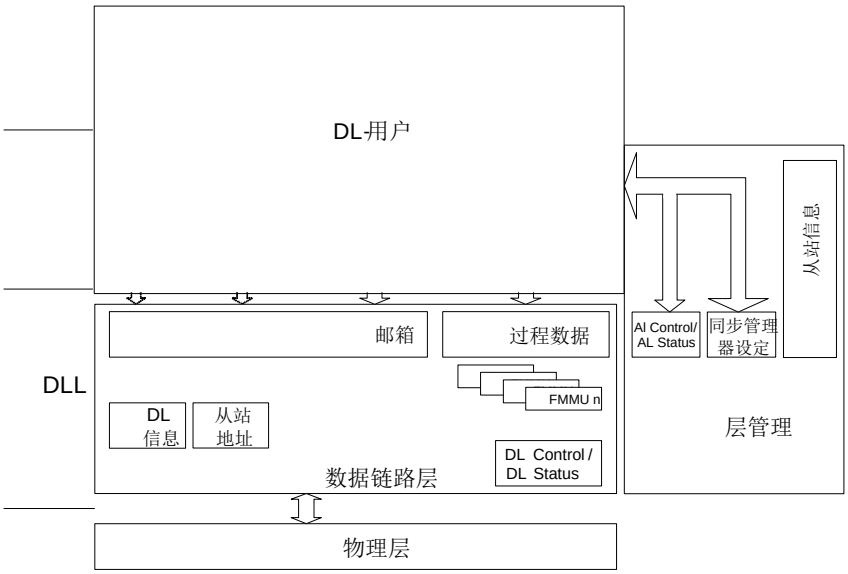


图 5 – 从站节点参考模型

4.6.2 数据链路层特征

数据链路层为通过 EtherCAT DL 连接的设备之间的数据通信提供基本时间关键的支持。“时间关键”一词用来描述带时窗的应用，在此时窗内，要求完成一个或多个有明确定义确定度的规定动作。在时窗内没有完成指定的动作，有可能造成需要该动作的应用的失败，甚至会影响设备，厂房及人身安全。

数据链路层的任务包括计算、比较、生成帧校验序列，并通过从以太网帧中提取或插入数据来实现通信。这些任务依据在被预先定义的内存位置中的数据链路层参数来实现。在物理内存中通过邮箱配置和过程数据部分使得应用层能够使用应用数据。

4.7 操作概述

4.7.1 与 ISO/IEC8802-3 的关系

这部分描述了 ISO/IEC8802-3 以外的数据链路层服务。

4.7.2 数据帧结构

EtherCAT 以太网帧包括一个或多个 EtherCAT PDU（如图 6），每个 PDU 寻址独立的设备或存储区。通过帧类型 0x88A4[1)]和 EtherCAT 帧头识别 EtherCAT 帧。或者，当采用符合 IETF RFC 791/IETF RFC 768 的 UDP/IP 传输时（如图 7），通过目的 UDP 的端口号 34980[2)]=0x88A4 和 EtherCAT 帧头识别 EtherCAT 帧。其他分散的 IP 数据包被忽略。如果 UDP 校验被从站设置成 0，也会被忽略。不检查 IP 服务类型，不校验 IP 协议头校验，但需要 IP 数据包长度和 UDP 的数据长度。

———————

1) IEEE 注册机构为 EtherCAT 分配的以太网帧类型为 0x88A4。
2) 互联网数字分配机构(IANA)为 EtherCAT 分配的 UDP 端口号是 34980。

每个 EtherCAT PDU 都包含一个 EtherCAT 头、数据域和相应的工作计数器。所有节点被 EtherCAT PDU 寻址并且交换相关的数据后，将增加工作计数器的值。

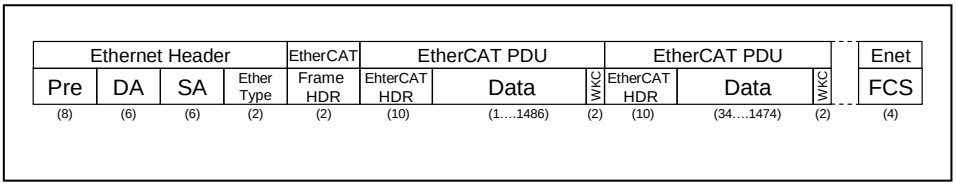


图 6 – EtherCAT PDU 嵌入式以太网帧

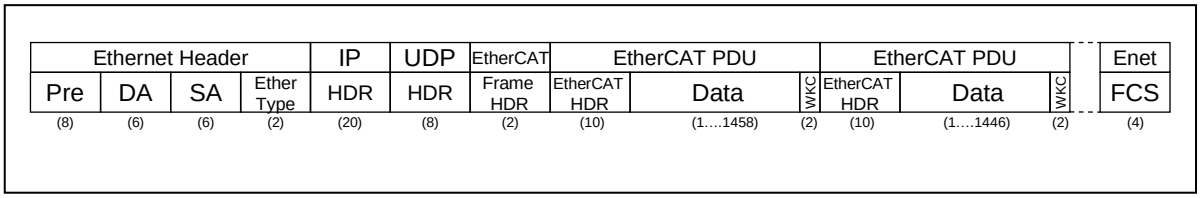


图 7 – EtherCAT PDUs 嵌入式 UDP/IP

5 帧结构

5.1 帧编码原则

EtherCAT DL 采用标准的 ISO/IEC 8802-3 以太网帧结构来传输 EtherCAT PDU。也可以选择通过 UDP/ IP 发送 PDU。 EtherCAT 的特定协议部分在这两种情况下是相同的。

5.2 数据类型和编码规则

5.2.1 数据类型和编码规则的一般描述

为了能够交换有意义的数据，数据格式和含义必须被生产者和消费者们知道。本规范通过数据类型的概念模型化以上需求。

编码规则定义了数据类型的数值的描述以及传输语法的描述。数值以位序列描述。位序列以八位位组为单位传输。数值的数据类型以小节字序风格编码，如表 6。

数据类型编码规则应该同时适用 DL 服务和协议以及 AL 服务和协议的规定。在 ISO/IEC 8802-3 中指定以太网帧的编码规则。以太网的 DLSDU 是一个八位位组串。八位位组的传送顺序取决于 MAC 和 PhL 的编码规则。

5.2.2 位序列的传送语法

为了通过 EtherCAT DL 传送，一个位序列会被重新编入一个八位位组序列中。在 GB/T 15272 中十六进制表示法同样可以用于八位位组。使 b = b0...bn-1 成为一个位序列。符号 k 是一个非负整数，比如 8(k - 1)< n ≤ 8k，那么 b 转换到 k 字节组就如表 6。位 bi, i ≥n 是八位位组的最高位数，是不关注位的。

八位位组 1 先被传送，八位位组 k 最后被传送。因此位序列传送依次通过网络（八位位组中的传送顺序由 ISO/IEC 8802-3 决定）：

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

表 6 – 位序列传送语法

字节数	1.	2.	k.
	b_7 .. b_0	b_{15} .. b_8	b_{8k-1} .. b_{8k-8}

例如

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

位序列 b = b0 .. b9 = 0011 1000 01b 表示一个值为 0x21c 的 Unsigned10，它被分成 2 个字节传送：第一个是 0x1C 然后是 0x02。

5.2.3 无符号整数

DATA 的基本数据类型 Unsignedn 的值是非负整数，值的范围是 0, ..., 2^n-1。数据表示为长度为 n 的位序列。位序列

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

被赋值

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

位序列的最低有效位从左边开始。

示例：值 266 = 0x10A 数据格式是 Unsigned16 进制数，它被分成 2 个八位位组传送，第一个是 0x0A 然后是 0x01。

Unsignedn 数据类型的传送见表 7。Unsigned 数据类型比如 Unsigned1～Unsigned7 和 Unsigned9～Unsigned15 也将被使用。在这种情况下，下一个单元将会从第一个自由位的位置开始，见 3.7.1。

表 7 – 无符号数据类型的传送语法

八位位组编号	1.	2.	3.	4.	5.	6.	7.	8.
Unsigned 8	$b_7..b_0$							
Unsigned16	$b_7..b_0$	$b_{15}..b_8$						
Unsigned32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Unsigned64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.2.4 带符号整数

DATA 的基本数据类型 Integern 的值是整数，值的范围是-2^{n-1} 到 $2^{n-1}-1$。数据表示长度为 n 的位序列。位序列

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

被赋值

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

以及，执行 2 的补码

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

注：位序列的最低有效位从左边开始。

示例：值–266 = 0xFEF6 的数据格式为 Integer16，它被分成 2 个八位位组发送，第一个是 0xF6 然后是 0xFE。

Integern 数据类型的传送详见表 8。Integern 数据类型比如 Integer1～Integer7 和 Integer9～Integer15 也将被使用。在这种情况下下一个单元将会从第一个自由位的位置开始。参考 3.7.1。

表 8 – 整数数据类型的传送语法

八位位组编号	1.	2.	3.	4.	5.	6.	7.	8.
Integer8	$b_7..b_0$							
Integer16	$b_7..b_0$	$b_{15}..b_8$						
Integer32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Integer64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.2.5 八位位组串

数据类型 OctetString*length* 的定义如下；length 是八位位组串的长度。

ARRAY [length] OF Unsigned8 OctetString*length*

5.2.6 可见字符串

VisibleString*length* 数据类型定义如下。*VISIBLE_CHAR* 数据类型的范围是 0_h，并从 0x20～0x7E。用 7-bit 编码符号表示其数据。*length* 是可见字符串的长度

Unsigned8 VISIBLE_CHAR

ARRAY [length] OF VISIBLE_CHAR VisibleString*length*

这里不需要 0x0 去终止此字符串。

5.3 DLPDU 结构

5.3.1 EtherCAT 帧嵌入以太网帧

帧结构包含以下数据项，详见表 9。

表 9 – EtherCAT 帧嵌套入以太网帧

帧部分	数据区域	数据类型	值/描述
Ethernet	Dest MAC	BYTE[6]	ISO/IEC 8802-3 规定的目的 MAC 地址
	Src MAC	BYTE[6]	ISO/IEC 8802-3 规定的源 MAC 地址
(optional)（可选的）	VLAN Tag	BYTE[4]	IEEE 802.1Q 规定的 0x81,0x00 和 2 字节标签控制信息
	Ether Type	BYTE[2]	0x88, 0xA4 (EtherCAT)
	EtherCAT frame		见 5.3.3
	Padding	BYTE[n]	ISO/IEC 8802-3 规定：如果 DL PDU 少于 64 个八位位组，则需要填充。
Ethernet FCS	FCS	Unsigned32	ISO/IEC 8802-3 规定的标准以太网校验和代码。

5.3.2 EtherCAT 帧嵌套 UDP 数据报

帧结构包括以下数据项。详见表 10。

表 10 – EtherCAT 帧嵌入 UDP PDU

帧部分	数据区域	数据类型	值/描述
Ethernet	Dest MAC	BYTE[6]	见表 9
	Src MAC	BYTE[6]	见表 9
(optional)（可选的）	VLAN Tag	BYTE[4]	见表 9
	Ether Type	BYTE[2]	0x08, 0x00 (IP)
IP	VersionHL	BYTE	0x45 (IP 版本(4) 头长度 (5*4 个八位位组))
	Service	BYTE	0x00 (IP 服务器类型)
	TotalLength	Unsigned16	(服务的 IP 总长度) – EtherCAT 段中不检查
	Identification	Unsigned16	(分段服务的 IP 标识) – EtherCAT 段中不检查
	Flags	BYTE	(IP 标志 – 他们将不被考虑但 EtherCAT 帧分段会导致错误) - EtherCAT 段中不检查
	Fragments	BYTE	(IP 分段数 - EtherCAT 帧分段会导致错误) - EtherCAT 段中不检查
	Ttl	BYTE	(IP 生存时间 – 只在路由器中被检测) - EtherCAT 段中不检查
	Protocol	BYTE	(IP 子协议 – 这个值为 UDP 保留)
	Header checksum	Unsigned16	(IP 头校验和) - EtherCAT 段中不检查
	Source IP address	BYTE[4]	(初始的 IP 源地址) - EtherCAT 段中不检查
	Destination IP address	BYTE[4]	(帧指向的 IP 目的地址 – 在 EtherCAT 中通常一个组播地址像单独地址一样需要地址解析协议 ARP) - EtherCAT 段中不检查
UDP	Src port	WORD	(UDP 源端口) - EtherCAT 段中不检查
	Dest port	WORD	0x88A4 (UDP 源端口)
	Length	WORD	(UDP 帧长度) - EtherCAT 段中不检查
	Checksum	WORD	(UDP 帧的校验和) – 被 EtherCAT 帧置 0 但不校验
	EtherCAT frame		参考 5.3.3
	Padding	BYTE[n]	ISO/IEC 8802-3 规定，如果 DLPDU 少于 64 个八位位组则需要填充
Ethernet FCS	FCS	Unsigned32	ISO/IEC 8802-3 指定标准 Ethernet 校验和代码。

注 1：IETF RFC 791 指定 IP 包结构和编码要求

注 2：IETF 协议(见 IETF RFC 768 和 RFC 791)的多八位位组值的编码八位位组顺序不同于 EtherCAT DL-协议。

5.3.3 EtherCAT 帧结构

EtherCAT 帧结构应由表 11 表 12 与表 13 中的结构之一构造。

表 11 – 包含 EtherCAT PDU 的 EtherCAT 帧结构

帧部分	数据区域	数据类型	值/描述
EtherCAT Frame	Length	unsigned11	帧的长度 (减去 2 个八位位组)
	Reserved	unsigned 1	0
	Type	unsigned4	协议类型 = EtherCATDLPDUs (0x01)
	EtherCAT PDU 1		见 5.4
	…		见 5.4
	EtherCAT PDU n		见 5.4

表 12 – 包含网络变量的 EtherCAT 帧结构

帧部分	数据区域	数据类型	值/描述
EtherCAT frame	Length	unsigned11	帧的长度 （减去 2 个八位位组）
	reserved	unsigned 1	0
	Type	unsigned4	协议类型 = network 变量 (0x04)
Publisher header	PubID	BYTE[6]	发布者 ID(Publisher ID)
	CntNV	Unsigned16	包括 EtherCAT 帧在内的网络变量数
	CYC	Unsigned16	发布者侧的周期数
	reserved	BYTE[2]	0x00, 0x00
	Network variable 1		见 5.5
	…		见 5.5
	Network variable n		见 5.5

表 13 – 包含邮箱的 EtherCAT 帧结构

帧部分	数据区域	数据类型	值/描述
EtherCAT frame	Length	unsigned11	帧的长度 (减去 2 字节)
	reserved	unsigned 1	0
	Type	unsigned4	协议类型 = 邮箱 (0x05)
	Mailbox		见 5.6

5.4 EtherCAT DLPDU 结构

5.4.1 读

5.4.1.1 概述

读服务是一个主站从一个或多个从站的内存中读取数据。如果至少出现一个被寻址的属性，那么每个从站应增加工作计数器。

5.4.1.2 自增式物理读(Auto increment physical read, APRD)

自增式物理读（APRD）编码见表 14。每个从站增加地址 ADP。从站接收到一个值为 0 的自增式地址就执行读操作请求。

表 14 – 自增式物理读(APRD)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x01 (APRD 命令)
	IDX	Unsigned8	索引
	ADP	WORD	自增式地址
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0: 帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6，第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

每个从站递增该参数，当从站接收到该参数为 0 值时执行读访问。

注：这表示，该参数包含主站侧在逻辑环中从 0 开始的从站负数位置（如：-7 表示有 7 个从站在主站和被寻址的从站之间）。收到证实原语时，该参数包含被增加了所经过的从站个数的请求值。

ADO

该参数包含从站物理内存空间中被读取的数据存储区的首地址。

LEN

该参数包含读取数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

如果对寻址的从站访问有效，那么该参数包含读取的数据。否则发送请求的值保持不变。

WKC

如果数据读取成功该参数增加 1。

5.4.1.3 配置的地址物理读(Configured address physical read, FPRD)

配置地址物理读（FPRD）编码。见表 15。

表 15 – 配置的地址物理读(FPRD)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x04 (FPRD 命令)
	IDX	Unsigned8	索引
	ADP	WORD	配置的站地址或配置站别名
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

D_ address 值作为站地址或者站地址别名的从站执行读动作。

ADO

该参数包含从站物理内存空间中被读取的数据存储区的首地址。

LEN

该参数包含读取数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数规定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

如果对寻址的从站访问有效，那么该参数包含读取的数据。否则发送请求的值保持不变。

WKC

如果数据读取成功该参数增加 1。

5.4.1.4 广播读(Broadcast read, BRD)

广播读（BRD）编码见表 16。

表 16 – 广播读(BRD)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x07 (FPRD 命令)
	IDX	Unsigned8	索引
	ADP	WORD	每个转送 BRD PDU 的站使该参数增加 1
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0: 帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

该参数经过每个从站都应增加 1。

ADO

该参数包含从站物理内存空间中被读取的数据存储区的的首地址。支持要求物理内存区域的从站（物理内存地址和长度）都应响应此服务。

LEN

该参数包含读取数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

该参数包含进入前收集的读取数据或以前主机的条目或默认值。该参数应包含请求参数数据和从站中寻址数据间的按位或操作的结果。

WKC

每个与请求数据做按位或操作的从站都使该参数增加 1。

5.4.1.5 逻辑读(Logical read, LRD)

逻辑读（LRD）编码见表 17。从站只复制数据到那些通过 FMMU 实体将逻辑地址空间映射到物理地址空间的参数数据。

表 17 – 逻辑读(LRD)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x0A (LRD 命令)
	IDX	Unsigned8	索引
	ADR	DWORD	逻辑地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	为以后的使用保留
	DATA	OctetString LEN	数据，DLS 用户指定结构
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADR

该参数包含将被读取数据的逻辑内存首地址。所有的从站中有一个或多个在 FMMU 实体中匹配的请求逻辑内存区域（逻辑内存地址和长度），将请求数据映射到 FMMU 实体设定所描述的数据参数中，且工作计数器加 1。

LEN

该参数包含读取数据的八位位组数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

一旦得到证实，该参数表示从设备中读取的数据。每一个检测到逻辑内存区域地址匹配的从站，把对应的物理内存空间中的数据放到参数中合适的部分。

WKC

所有检测到被请求的逻辑内存区域地址匹配的从站都使该参数增加 1。

5.4.2 写

5.4.2.1 概述

写服务是主站向一个或多个从站的寄存器或内存写数据。如果存在被寻址的属性，那么工作计数器会增加。如果至少一个部分的数据能被写，工作计数器增加 1。

5.4.2.2 自增式物理写(Configured address physical read, FPRD)

自增式地址物理地址写（APWR）编码见表 18。每个从站地址加 1。从站接收到一个值为 0 的自增式地址物理地址参数就执行写操作请求。

表 18 – 自增式物理写(APWR)

帧部分	数据区域	数据类型	数值/描述
EtherCAT PDU	CMD	8Unsigned8	0x02 (APWR 命令)
	IDX	Unsigned8	索引
	ADP	WORD	自增加地址
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	数据区域的数据长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

从站按照其在段中的位置被寻址。每个从站应递增该参数。接收到此参数值为 0 的从站应响应此服务。

注：这表示，该参数包含主站侧在逻辑环中从 0 开始的从站负数位置（如：-7 表示有 7 个从站在主站和被寻址的从站之间）。收到证实原语时，该参数包含被增加了所经过的从站个数的请求值。

ADO

该参数包含从站物理内存空间中被写的数据存储区的首地址。

LEN

该参数包含写入数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

该参数包含写入的数据。

WKC

如果数据写入成功，该参数加 1。

5.4.2.3 配置的地址物理写(Configured address physical write, FPWR)

配置地址物理写（FPWR）编码见表 19。

表 19 – 配置的地址物理写(FPWR)

帧部分	数据区域	数据类型	数值/描述
EtherCAT PDU	CMD	Unsigned 8	0x05 (FPWR 命令)
	IDX	Unsigned 8	索引
	ADP	WORD	配置站地址或配置站别名
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned 11	数据区域的数据长度
	reserved	Unsigned 3	0x00
	C	Unsigned 1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned 1	0x00:EtherCAT 帧的最后一个 EtherCATPDU 0x01:EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6，第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

D_ address 值作为站地址或者站地址别名的从站执行写动作。

ADO

该参数包含从站物理内存空间中被写的数据存储区的首地址。

LEN

该参数包含写入数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）。

DATA

该参数包含写入的数据。

WKC

如果数据写入成功，参数加 1。

5.4.2.4 广播写(Broadcast write, BWR)

广播写（BWR）编码见表 20。

表 20 – 广播写(BWR)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x08 (BWR 命令)
	IDX	Unsigned8	索引
	ADP	WORD	每个转送 BRD PDU 的站使该参数增加 1
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

该参数经过每个从站都应增加 1。

ADO

该参数包含从站物理内存空间中被写的数据存储区的首地址。支持请求物理内存区域的从站（物理内存地址和长度）都应响应服务器。

LEN

该参数包含写入数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）。

DATA

该参数包含写入的数据。

WKC

把数据写入其物理内存的所有从站都将该参数加 1。

5.4.2.5 逻辑写(Logical write, LWR)

逻辑写（LWR）编码见表 21。从站只复制数据到那些通过 FMMU 实体将逻辑地址空间映射到物理地址空间的内存或者寄存器中。

表 21 – 逻辑写(LWR)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x0B (LWR 命令)
	IDX	Unsigned8	索引
	ADR	DWORD	逻辑地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00:帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	为以后使用保留
	DATA	OctetString LEN	数据，DLS 用户指定结构
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADR

该参数包含将被写入数据的逻辑内存首地址。所有的从站中有一个或多个在 FMMU 实体中匹配的请求逻辑内存区域（逻辑内存地址和长度），将请求数据映射到 FMMU 实体设定所描述的数据参数中，且工作计数器加 1。

LEN

该参数包含写入数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

该参数包含写入的数据。每一个检测到请求逻辑内存区域地址匹配的从站，将参数中符合的部分放到相应的物理内存空间。

WKC

当所有从站都检测到一个请求逻辑内存区域的地址并且数据写入成功，该参数应增加 1。

5.4.3 读写

5.4.3.1 自增式物理读写(Auto increment physical read write, APRW)

自增式物理读写（APRW）编码见表 22。每个从站地址递增。从站接收到自增式地址参数为 0 时，执行请求操作。

表 22 – 自增式物理读写(APRW)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x03 (APRW 命令)
	IDX	Unsigned8	索引
	ADP	WORD	自动增加地址
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

从站将会被按照其在段落中的位置分配地址。每个从站需要递增该参数。接收到参考值为 0 的从站将响应此服务。

注：这表示，该参数包含主站侧在逻辑循环中从 0 开始的从站负数位置（如：-7 表示有 7 个从站在主机和被分配地址的从站之间）。确认这个参数包含传送从站设备所请求增加的数量。

ADO

该参数包含从站物理内存空间中的首地址，被读取和写入的数据存储在这里。

LEN

该参数包含写入数据的八位位组数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

该参数包含写入的数据以及从被分配地址的从站读到的数据服务器是否执行完成。

WKC

该参数如果写数据成功增加二，数据成功读取增加一。

5.4.3.2 配置的地址物理读写(Configured address physical read write, FPRW)

配置地址物理读写（FPRW）编码见表23。

表 23 – 配置的地址物理读写(FPRW)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x06 (FPRW 命令)
	IDX	Unsigned8	索引
	ADP	WORD	配置站地址或配置站别名
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0 帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

站地址或者站地址别名为 D_address 值的从站，在读操作后执行写动作。

ADO

该参数包含从站物理存储空间中数据被读取和写入存储区的首地址。

LEN

该参数包含读写数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

如果服务执行成功，该参数应包含要写入的数据和从被寻址的从站读出的数据。

WKC

如果所有的从站写数据成功该参数增加 2，读数据成功再增加 1。

5.4.3.3 广播读写(Broadcast read write, BRW)

广播读写（BRW）编码见表 24。

表 24 – 广播读写(BRW)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x09 (BRW 命令)
	IDX	Unsigned8	索引
	ADP	WORD	每个转送 BRD PDU 的站使该参数增加 1
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADP

该参数经过每个从站都应增加 1。

ADO

该参数包含从站物理内存空间中读取和写入数据存储区的首地址。支持请求物理内存区域的从站（物理内存地址和长度）都应响应此服务。

LEN

该参数包含读写数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

该参数包含进入前的数据并被写入。允许在写之前执行一个读操作。该参数应包请求参数数据和从站中被寻址数据间的按位或操作的结果。

WKC

如果写数据成功应使该参数增加 2，按位或请求数据的所有从站使该参数再增加 1。

5.4.3.4 逻辑读写(Logical read write, LRW)

逻辑读写（LRW）编码见表 25。从站设备通过此服务(写操作)来取回数据以及通过此服务(读操作)来放置数据。从站仅将数据复制到参数数据或仅从参数数据复制出数据，这些参数数据被 FMMU 实体从逻辑地址映射到物理地址。

表 25 – 逻辑读写(LRW)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x0C (LRW 命令)
	IDX	Unsigned8	索引
	ADR	DWORD	逻辑地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	为以后使用保留
	DATA	OctetString LEN	数据，DLS 用户指定的结构
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是此服务中主站的本地标识符；它不能被从站修改。

ADR

该参数包含将被读写数据所在的逻辑内存的首地址。在 FMMU 实体中有一个或多个匹配的请求逻辑内存区域（逻辑内存地址和长度）的所有从站应响应此服务。

LEN

该参数包含读写数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

该参数包含被写入的数据。检测到与地址请求逻辑内存区域的匹配的从站，将正确数据放入相应的物理内存空间。通过证实原语，该参数应包含读到的数据。每一个从站检测到与逻辑内存区域地址匹配的地址，从站把对应的物理内存空间的数据放入参数中对应的区域。

WKC

如果一段数据被成功写入，该参数增加 2；如果一段数据被成功读出，该参数再加 1。

5.4.3.5 自增式物理读多次写(Auto increment physical read multiple write, ARMW)

自增式物理读多次写（ARMW） 编码见表 26。

表 26 – 自增式物理读多次写(ARMW)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x0D (ARMW 命令)
	IDX	Unsigned8	索引
	ADP	WORD	自增式地址或寄存器
	ADO	WORD	物理内存或寄存器地址
	LEN	Unsigned11	数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或者由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

参数索引是服务主站的本地标识符；它不能被从站修改。

ADP

从站按照其在段中的位置被寻址。每个从站应递增该参数。接收到此参数值为 0 的从站应执行一个读动作，其他的从站执行写动作。

注：这表示，该参数包含在逻辑环中主站侧从 0 开始的从站的负数位置（如：-7 表示有 7 个从站在主站和被寻址的从站之间）。收到证实原语时，该参数包含被增加了所经过的从站个数的请求值。

ADO

该参数包含从站物理内存空间中的首地址，被写入和读取的数据存储在这里。

LEN

该参数包含读写数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）。

DATA

如果服务能成功完成，该参数包含从从站地址中写入和读出的数据。

WKC

该参数被每个成功读或写数据的从站增加 1。

5.4.3.6 配置的地址物理读多次写(Configured address physical read multiple write, FRMW)

配置地址物理读多次写（FRMW）编码见表 27。

表 27 – 配置的地址物理读多次写(FRMW)

帧部分	数据区域	数据类型	值/描述
EtherCAT PDU	CMD	Unsigned8	0x0E (FRMW 命令)
	IDX	Unsigned8	索引
	ADP	WORD	配置站地址或配置站别名
	ADO	WORD	物理内存地址
	LEN	Unsigned11	DATA 数据区域的长度
	reserved	Unsigned3	0x00
	C	Unsigned1	循环帧 0:帧没有循环 1:帧已循环一次
	NEXT	Unsigned1	0x00: EtherCAT 帧的最后一个 EtherCAT PDU 0x01: EtherCAT 帧里还有后续 EtherCAT PDU
	IRQ	WORD	外部事件
	DATA	OctetString LEN	数据，结构按照 5.6、第 6 章或由 DLS 用户规定
	WKC	WORD	工作计数器

CMD

命令参数应包含服务命令。

IDX

索引参数是主站的本地标示符；它不能被从站修改。

ADP

具有值 D_ address 作为站地址或者站地址别名的从站执行读动作 – 其他从站执行写动作。

ADO

该参数包含从站物理内存空间中的首地址，被写入和读取的数据存储在这里。

LEN

该参数包含读写数据的八位位组个数。

C

该参数表示帧在网络中已经循环，不被转发。

NEXT

该参数指定帧中是否还有另一个 EtherCAT PDU。

IRQ

该参数包含被外部事件掩码（见表 39）所掩码的外部事件（见表 38）

DATA

如果服务能成功完成，该参数包含从从站地址中写入和读出的数据。

WKC

该参数被每个成功读或写数据的从站增加 1。

5.5 网络变量结构

网络变量编码见表 28。

表 28 – 网络变量

帧部分	数据区域	数据类型	值/描述
Network variable	Index	Unsigned16	DLS-user 对象的索引
	HASH	Unsigned16	全部数据结构的哈希算法，用以检测数据的变更
	LEN	Unsigned16	长度
	Q	Unsigned16	品质
	DATA	OctetString [LEN]	数据，结构由 DLS 用户指定

5.6 EtherCAT 邮箱结构

邮箱编码见表 29。邮箱编码应与 EtherCAT 邮箱内存元素结合使用或编码成通过以太网 DL 或 IP 传送邮箱的数据结构。

表 29 – 邮箱

帧部分	数据区域	数据类型	值/描述
Mailbox	Length	Unsigned16	邮箱服务数据的长度
	Address	WORD	如果一个主站是客户端，站地址是源，如果一个从站是客户端或数据被传输到目标 EtherCAT 段以外，那么站地址是目的
	Channel	Unsigned6	0x00（为以后保留）
	Priority	Unsigned2	0x00:（最低优先级） … 0x03:（最高优先级）
	Type	Unsigned4	0x00: 错误(ERR) 0x01: 保留 0x02: 基于 EtherCAT 服务的以太网隧道(EoE)； 0x03: 基于 EtherCAT 服务的 CAN 应用协议 (CoE)； 0x04: 基于 EtherCAT 服务的文件访问(FoE)； 0x05: 基于 EtherCAT 的伺服行规 (SoE)； 0x06 -0x0e: 保留； 0x0f: 供应商特定

	Cnt	Unsigned3	邮箱服务计数器（0 保留，1 是起始值，7 以后的下一个值是 1） 从站为每个新邮箱服务递增 Cnt 值。主站应检查它，以发现丢失的邮箱服务。主站应改变（递增）Cnt 值。从站应检查它，以发现重复写的服务。从站不应检查 Cnt 值的顺序。主站和从站的 Cnt 值是独立的。
	reserved	Unsigned1	0x00
	Service Data	OctetString [Length]	邮箱服务数据

.错误回复的服务数据编码见表 30。

表 30 – 错误回复服务数据

帧部分	数据区域	数据类型	值/描述
Service Data	Type	Unsigned16	0x01:邮箱命令
	Detail	Unsigned16	0x01: MBXERR_SYNTAX 6 个八位位组的邮箱头文语法错误； 0x02: MBXERR_UNSUPPORTEDPROTOCOL 不支持邮箱协议； 0x03: MBXERR_INVALIDCHANNEL Channel 字段包含错误值 (从站可以忽略 Channel 字段)； 0x04: MBXERR_SERVICENOTSUPPORTED 不支持邮箱协议中的服务； 0x05: MBXERR_INVALIDHEADER 邮箱协议的邮箱协议头错误 (不包括 6 个八位位组的邮箱头)； 0x06: MBXERR_SIZETOOSHORT 接收的邮箱数据长度太短； 0x07: MBXERR_NOMOREMEMORY 由于资源限制邮箱协议不能被处理； 0x08: MBXERR_INVALIDSIZE 数据长度不一致。

6 属性

6.1 管理

6.1.1 DL 信息

DL 信息寄存器包含从站控制器(ESC)的类型、版本和被支持的资源。

参数

Type

该参数应该包括这些从站控制器的类型。

Revision (主版本)

该参数应包含从站控制器的修订号。

Build (minor revision)

该参数应包含从站控制器的构建版本号。

Number of supported FMMU entities

该参数应包含从站控制器支持 FMMU 实体的数量。

Number of supported sync manager channels

该参数应包含从站控制器支持的同步管理器通道（或实体）的数量。

RAM size

该参数应包含从站控制器的 RAM 容量，单位是 Kbyte（容量小于偶数将会自动进位）。

Port descriptor 端口描述

Port 0 Physical Layer

该参数应表明端口 0 所使用的物理层。

Port 1 Physical Layer

该参数应表明端口 1 所使用的物理层。

Port 2 Physical Layer

该参数应表明端口 2 所使用的物理层。

Port 3 Physical Layer

该参数应表明端口 3 所使用的物理层。

Features supported

FMMU bit operation not supported

该参数应表明在从站控制器的 FMMU 是否支持位操作。

DC supported

如果支持分布式时钟该参数设置成 1

DC range

该参数应表明以 ns 为单位的时钟值的范围

Low jitter EBUS

该参数应表明低抖动功能特性有效

Enhanced link detection

该参数表明对 EBUS 口增强型链接检测有效

Enhanced link detection

该参数表明对 MII 口增强型链接检测有效

Separate Handling of FCS errors

该参数应表明由另外的 EtherCAT 从站控制器引起的误差将被单独计算

Special FMMU/Sync manager configuration

该参数应表明一个专用 FMMU(现场总线存储管理单元)/同步管理配置被使用：

FMMU 0 被用作 RxPDO（没有位映射）

FMMU 1 被用作 TxPDO（没有位映射）

FMMU 2 被用作同步管理器的邮箱写事件位

Sync manager 0 被用作写邮箱

Sync manager 1 被用作读邮箱

Sync manager 2 被用作 RxPDO 的缓存区

Sync manager 3 被用作 TxPDO 的缓存区

LRW not supported

该参数应表明是否支持 LRW。

BRW, APRW, FPRW not supported

该参数应表明支持 BRW, APRW, FPRW。

图 8 描述了 DL 信息的属性类型

```
typedef struct
{
  BYTE          Type;
  BYTE          Revision;
  WORD          Build;
  BYTE          NoOfSuppFmmuEntities;
  BYTE          NoOfSuppSyncManChannels;
  BYTE          RamSize;
  BYTE          PortDescr;
  unsigned      FmmuBitOperationNotSupp:  1;
  unsigned      Reserved2:                1;
  unsigned      DCSupp:                   1;
  unsigned      DCRange:                  1;
  unsigned      LowJEBUS:                 1;
  unsigned      EnhLDEBUS:                1;
  unsigned      EnhLDMII:                 1;
  unsigned      FCSsERR:                  1;
  unsigned      sFMMUSyMC:                1;
  unsigned      NotSuppLRW:               1;
  unsigned      NotSuppBAFRW:             1;
  unsigned      Reserved4:                5;
} TDLINFORMATION;
```

图 8 – DL 信息类型描述

DL 信息的编码见表 31

表 31 – DL 信息

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Type	0x0000	BYTE	R	R	
Revision	0x0001	BYTE	R	R	
Build	0x0002	WORD	R	R	
Number of supported FMMU entities	0x0004	BYTE	R	R	0x0001-0x0010
Number of supported Sync Manager channels	0x0005	BYTE	R	R	0x0001-0x0010
RAM Size	0x0006	BYTE	R	R	以 Koctet（1024 个八位位组）为单位的 RAM 大小 (1～60)
Port0 Descriptor	0x0007	unsigned2	R	R	可选项 00: 未应用 01: 未配置 10: EBUS 11: MII
Port1 Descriptor	0x0007	unsigned2	R	R	可选项 00: 未应用 01: 未配置 10: EBUS 11: MII
Port2 Descriptor	0x0007	unsigned2	R	R	可选项 00: 未应用 01: 未配置 10: EBUS 11: MII
Port3 Descriptor	0x0007	unsigned2	R	R	可选项 00: 未应用 01: 未配置 10: EBUS 11: MII
FMMU Bit Operation Not Supported	0x0008	unsigned1	R	R	0:支持位操作 1:不支持位操作
Reserved	0x0008	unsigned1	R	R	
DC Supported	0x0008	unsigned1	R	R	0: DC 不支持 1: DC 支持
DC Range	0x0008	unsigned1	R	R	0: 32 位 1: 64 位系统时间，系统时间偏移量和接收时间端口 0
Low Jitter EBUS	0x0008	unsigned1	R	R	0:无效 1: e 有效
Enhanced Link Detection EBUS	0x0008	unsigned1	R	R	0:无效 1:有效，如果在最近 256 位单元中发现超过 16 个错误则关闭链接
Enhanced Link Detection MII	0x0008	unsigned1	R	R	0:无效 1:有效，如果在最近 256 位单元中发现超过 16 个错误则关闭链接，

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Separate Handling of FCS errors	0x0008	unsigned1	R	R	0:无效 1:有效，帧和修改过的 FCS(最高有效位被取反) 将被单独在 RX- Error Previous counter 中计数
Reserved	0x0009	unsigned1	R	R	
LRW not supported	0x0009	unsigned1	R	R	0:支持 LRW 1:不支持 LRW
BRW, APRW; FPRW not supported	0x0009	unsigned1	R	R	0:支持 BRW, APRW; FPRW 1:不支持 BRW, APRW; FPRW
Special FMMU Syc manager configuration	0x0009	unsigned1	R	R	0:无效 1:, 有效 FMMU 0 被用作 RxPDO（没有位映射） FMMU 1 被用作 TxPDO（没有位映射） FMMU 2 被用作同步管理器 1 的邮箱写事件位 同步管理器 0 被用作写邮箱 同步管理器 1 被用作读邮箱 同步管理器 2 被用作流入数据的缓存区 同步管理器 0 被用作流出数据的缓存区
Reserved	0x0009	unsigned4	R	R	

6.1.2 站地址

配置的站地址寄存器包含从站的站地址，该从站控制器中的 FPRD、FPRW、FRMW 和 FPWR 服务被激活。

参数

Configured station address

该参数应包含被配置的从站控制器的站地址。这个站地址在启动时被主站设立。

Configured station alias

该参数应包含配置的从站控制器的站别名。这个站别名在启动时被 DL-user 设立。

站地址的属性类型 见图 9

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

图 9 – 地址类型描述

站地址的编码详见表 32

表 32 – 被配置的站地址

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Configured Station Address	0x0010	WORD	RW	R	
Configured Station Alias	0x0012	WORD	RW	R	

6.1.3 DL 控制

DL 控制寄存器通常被主站用来控制从站控制器 DL 端口的操作。

参数

Forwarding rule

该参数应使能直接转发或限制转发。限制转发将改变源 MAC 地址并删除非 EtherCAT 帧。

Loop control port 0

该参数应包括下列信息：该端口是处于物理连接时自动激活状态还是由主机的命令打开或关闭。

Loop control port 1

该参数应包括下列信息：该端口是处于物理连接时自动激活状态还是由主机的命令打开或关闭。

Loop control port 2

该参数应包括下列信息：该端口是处于物理连接时自动激活状态还是由主机的命令打开或关闭。

Loop control port 3

该参数应包括下列信息：该端口是处于物理连接时自动激活状态还是由主机的命令打开或关闭。

Transmit buffer size

该可选参数应当被用作优化站内的延时。如果这个站和它的邻站有一个稳定的速率传输，该参数或许会被减小。这个默认设置由 ISO/IEC 8802-3 的时钟精度要求所决定。

Enable alias address

该可选参数用来使能别名地址。

DL 控制属性类型见图 10

```
typedef struct
{
  unsigned      ForwardingRule:         1;
  unsigned      Reserved0:              7;
  unsigned      LoopControlPort0:       2;
  unsigned      LoopControlPort1:       2;
  unsigned      LoopControlPort2:       2;
  unsigned      LoopControlPort3:       2;
  unsigned      TxBufferSize:           3;
  unsigned      Reserved3:              5;
  unsigned      EnableAliasAddress:     1;
  unsigned      Reserved4:              7;
} TDLCONTROL;
```

图10 – DL 控制类型描述

DL 控制编码见表 33。

表 33 – DL 控制

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Forwarding rule	0x0100	无符号 1	RW	R	0:不处理直接转发 1:改变源 MAC 地址和删除非 EtherCAT 帧
reserved	0x0100	无符号 7	RW	R	0x00
Loop control port 0	0x0101	无符号 2	RW	R	0: o 自动 => “未链接”时关闭“已链接”时打开 1:自动关 => “未链接”时关闭 “已链接”时打开并写 1 2: 常开 3: 常闭
Loop control port 1	0x0101	无符号 2	RW	R	0:自动 => “未链接”时关闭“已链接”时打开 1:自动关 => “未链接”时关闭 “已链接”时打开并写 1 2: 常开 3: 常闭
Loop control port 2	0x0101	无符号 2	RW	R	0:自动 => “未链接”时关闭“已链接”时打开 1:自动关 => “未链接”时关闭 “已链接”时打开并写 1 2: 常开 3: 常闭

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Loop control port 3	0x0101	Unsigned2	RW	R	0:自动 => “未链接”时关闭“已链接”时打开 1:自动关 => “未链接”时关闭 “已链接”时打开并写 1 2: 常开 3: 常闭
TransmitBufferSize	0x0102	Unsigned3	RW	R	在准备和发送之间的缓存区，如果缓存区充满一半（7）将被发送
reserved	0x0102	Unsigned5	RW	R	0x00
EnableAliasAddress	0x0103	Unsigned1	RW	R	0:未使能站别名地址 1:使能站别名地址
reserved	0x0103	Unsigned7	RW	R	0x00
注：回路打开的意思是使能通过这个端口发送并且在接收端口等待一个反应—接收数据将被转发到对等端口。回路关闭的意思是被转发的数据直接被镜像，因此他们将被转发到对等端口。一个关闭端口将摒弃所有接收的数据。					

6.1.4 DL 状态

DL 状态寄存器被用作指示 DL 端口的状态以及 DL 用户和 DL 之间接口的状态

参数

DL-user operational

该参数包含的信息为是否有一个 DL 用户被连接到了从站控制器过程数据接口。

DL-user watchdog status

该参数应包含进程数据接口看门狗的状态。

Extended link detection

该参数应包含扩展链接检测的使能状态。

Link status port 0

该参数指明这个端口上的的物理链接。

Link status port 1

该参数指明这个端口上的的物理链接。

Link status port 2

该参数指明这个端口上的的物理链接。

Link status port 3

该参数指明这个端口上的的物理链接。

Loop back port 0

该参数指明在同一端口上的转发，即回送。

Signal detection port 0

该参数指明在 RX-Port 口上是否检测到信号。

Loop back port 1

该参数指明在同一端口上的转发，即回送。

Signal detection port 1

该参数指明在 RX-Port 口上是否检测到信号。

Loop back port 2

该参数指明在同一端口上的转发，即回送。

Signal detection port 2

该参数指明在 RX-Port 口上是否检测到信号。

Loop back port 3

该参数指明在同一端口上的转发，即回送。

Signal detection port 3

该参数指明在 RX-Port 口上是否检测到信号。

DL 状态的属性类型见图 11

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

图 11 – DL 状态类型描述

DL 状态编码说明见表 34。

表 34 – DL 状态

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
DLS-user operational	0x0110	Unsigned1	R	R	0x00: DLS 用户不操作 0x01: DLS 用户操作
DLS-user watchdog status	0x0110	Unsigned1	R	R	0x00: DLS 用户看门狗失效 0x01: DLS 用户看门狗未失效
Extended link detection	0x0110	Unsigned1	R	R	0x00
Reserved	0x0110	Unsigned1	R	R	0x00
Link status port 0	0x0110	Unsigned1	R	R	0x00:在这个端口没有物理链接 0x01:在这个端口有物理链接
Link status port 1	0x0110	Unsigned1	R	R	0x00:在这个端口没有物理链接 0x01:在这个端口有物理链接
Link status port 2	0x0110	Unsigned1	R	R	0x00:在这个端口没有物理链接 0x01:在这个端口有物理链接
Link status port 3	0x0110	Unsigned1	R	R	0x00:在这个端口没有物理链接 0x01:在这个端口有物理链接
Loop status port 0	0x0111	Unsigned1	R	R	0x00:回路没有激活 0x01: 回路激活
Signal detection port 0	0x0111	Unsigned1	R	R	0x00:在 RX-port 上没检测到信号 0x01: 在 RX-port 上检测到信号
Loop status port 1	0x0111	Unsigned1	R	R	0x00: 回路没有激活 0x01: 回路激活
Signal detection port 1	0x0111	Unsigned1	R	R	0x00:在 RX-port 上没检测到信号 0x01: 在 RX-port 上检测到信号
Loop status port 2	0x0111	Unsigned1	R	R	0x00: 回路没有激活 0x01: 回路激活
Signal detection port 2	0x0111	Unsigned1	R	R	0x00:在 RX-port 上没检测到信号 0x01: 在 RX-port 上检测到信号
Loop status port 3	0x0111	Unsigned1	R	R	0x00: 回路没有激活 0x01: 回路激活
Signal detection port 3	0x0111	Unsigned1	R	R	0x00:在 RX-port 上没检测到信号 0x01: 在 RX-port 上检测到信号

6.1.5 DLS 用户特殊寄存器

6.1.5.1 DLS 用户控制寄存器

图 12 说明了主站，DL 和 DL 用户之间成功写 DL 用户控制寄存器(R1)的顺序的原语。

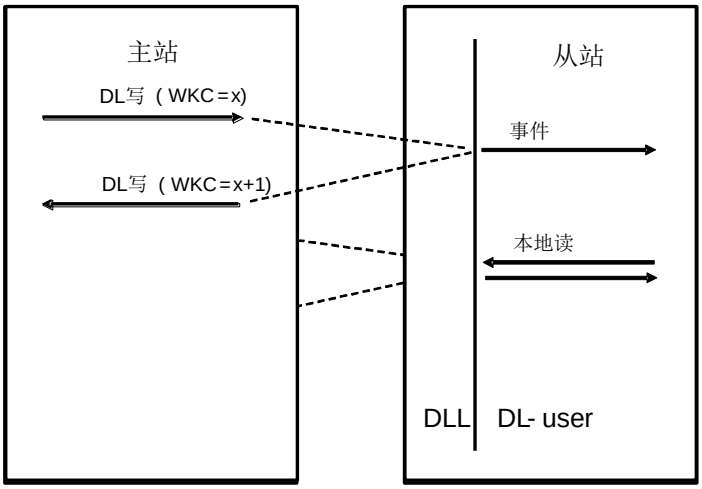


图 12 – 成功写 DL 用户控制寄存器的顺序

主站发送一个带工作计数器(WKC = x)写服务，从站 DL(从站控制器)把接收到的数据写入寄存器区域，增加工作计数器(WKC = x + 1)并产生一个事件同时 DL 用户读控制寄存器。如果控制寄存器没有读出，下次写这个寄存器会被忽略（不被改变）。

控制寄存器用来在主站和从站间控制信息。

6.1.5.2 DL 用户状态寄存器

图 13 说明了主站、DL 和 DL 用户之间成功读 DL 用户状态寄存器(R3)的原语序列。

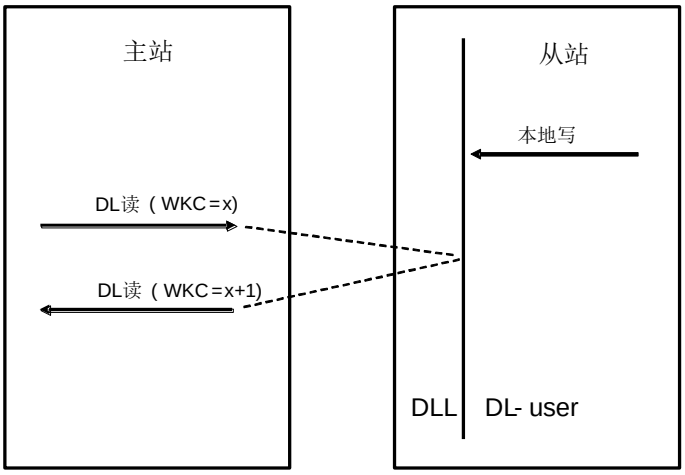


图 13 – 成功读 DL 用户状态寄存器的顺序

从站的 DL 用户以本地方式写 DL 用户状态寄存器。主机发送一个带工作计数器(WKC = x)的读服务，从站 DL(从控制器)从寄存器区域发送数据，并增加工作计数器(WKC = x + 1)

6.1.5.3 DL 用户特殊寄存器

DL 用户有一组特殊寄存器 R2, R4 到 R8。由 DL 用户定义其内容的含义。

6.1.5.4 DL 用户属性

DLS 用户特殊寄存器结构和访问类型描述见表 35。

表 35 – DLS 用户特殊寄存器

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
DLS-user R1	0x0120	Unsigned8	RW	R	0x01
DLS-user R2	0x0121	Unsigned8	RW	R	0x00
DLS-user R3	0x0130	Unsigned8	R	RW	0x01
DLS-user R4	0x0131	Unsigned8	R	RW	0x00
DLS-user R5	0x0132	Unsigned16	R	RW	0x00
DLS-user R6	0x0134	Unsigned16	R	RW	0x00
DLS-user R7	0x0140	Unsigned8	R	R	0x00
Copy	0x0141	Unsigned1	R	R	0x00:没有特殊动作 0x01:复制 DLS-user R1 到 DLS-user R3
reserved	0x0141	Unsigned7	R	R	0x00
DLS-user R8	0x0150	Unsigned32	R	R	0x00

6.1.6 事件参数

事件寄存器被用作指明对 DL 用户的一个事件，如果相应的事件源被读取，这个事件应当被确认，这些事件能够被屏蔽。

参数

DL-user Event

DL-user R1 Chg

如果一个向 DL 用户控制寄存器的写服务被调用，该参数被置位，并且如果 DL 用户控制寄存器被本地读，该参数被复位。

DC Event 0

如果 DC 事件 0 被激活，该参数被置位。

DC Event 1

如果 DC 事件 1 被激活，该参数被置位。

DC Event 2

如果 DC 事件 2 被激活，该参数被置位。

Sync manager change event

如果一个多亏同步管理器区的写服务发生，该参数被置位。如果 DL 用户读出事件寄存器，该参数被复位。

Sync manager channel access events (0 到 15)

如果接收到对应用内存区的写服务，该内存区被主站配置为写，或一个对应用内存区的读服务，该内存区被主站配置为读，该参数被置位，并且当应用内存区将被读(本地读)或写(本地写)，该参数被复位。

DL-user Event Mask

如果相应的属性被设定，DL 用户事件将被使能，否则不使能。

External Event

DC Event 0

如果 DC 事件 0 激活，该参数被置位。

DL Status Chg

如果 DL 状态寄存器被改变，该参数被置位，并且当一个对 DL 状态寄存器的 EtherCAT 读服务被调用，该参数被复位。

DL-user R3 Chg

如果一个对 DL 用户状态寄存器写本地服务被调用，该参数被置位，并且当一个对 DL 用户状态寄存器的 EtherCAT 读服务被调用，该参数被复位。

Sync manager channel access events (0 到 7)

如果接收到一个把应用内存区被从站配置为写的写服务或一个把应用内存区被从站配置为读的读服务，该参数被设置，并且当应用内存区将被读(本地读)或写(本地写)，该参数被复位。

Event Mask

如果相应的属性被设置，外部事件将被使能，否则不使能。

DLS 用户的事件结构和访问类型见表 36。

表 36 – DLS-user 事件

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
DLS-user R1 chg	0x0220	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活 R1（被写）
DC event 0	0x0220	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（至少一个锁存器事件发生）
DC event 1	0x0220	unsigned1	R	r	0x00:同步信号不激活 0x01:同步信号激活
DC event 2	0x0220	unsigned1	R	r	0x00:同步信号不激活 0x01:同步信号激活
Sync manager change event	0x0220	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（一个或者更多同步管理器通道被改变）
EEPROM Emulation	0x0220	unsigned1	R	r	0x00:没有命令待执行 0x01: EEPROM 命令待执行
Watchdog Process Data	0x0220	unsigned1	R	r	0x00 没有超时 0x01: 超时 通过读“Sync Manager Watchdog Status”清除位
reserved	0x0220	unsigned1	R	r	0x00
Sync manager channel 0 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 1 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 2 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 3 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 4 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 5 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 6 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Sync manager channel 7 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 8 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 9 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 10 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 11 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 12 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 13 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 14 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被访问）
Sync manager channel 15 event	0x0222	unsigned1	R	r	0x00:没有事件激活 0x01: 事件激活（同步管理器通道被访问）
reserved	0x0223	Unsigned8	R	r	0x00

与 DLS 用户事件相关的 DLS 用户事件掩码和编码见表 37。

表 37 – DLS-user 事件掩码

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Event mask	0x0204	array [0..31] of unsigned1	R	rw	对每一个元素 0:无效 1:使能事件

远程合作者的 Event 结构和访问类型描述见表 38。外部事件映射到访问此从站的所有 EtherCAT PDU 的 IRQ 参数。如果一个事件被置位并且和它相关的掩码被置位，PDU 的 IRQ 参数中的相关位也被置位。

表 38 – 外部事件

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
DC Event 0	0x0210	unsigned1	R	r	0x00:没有事件激活 0x01: DC 事件 0 激活
reserved	0x0210	unsigned1	R	r	0x00
DL Status change	0x0210	unsigned1	R	r	0x00:没有事件激活 0x01: DL 状态寄存器被改变激活
R3 Chg	0x0210	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（R3 被写）
Sync manager channel 0 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Sync manager channel 1 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Sync manager channel 2 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Sync manager channel 3 event	0x0221	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Sync manager channel 4 event	0x0211	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Sync manager channel 5 event	0x0211	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Sync manager channel 6 event	0x0211	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Sync manager channel 7 event	0x0211	unsigned1	R	r	0x00:没有事件激活 0x01:事件激活（同步管理器通道被从站访问）
Reserved	0x0211	Unsigned4	R	r	0x00

与外部事件相关的外部事件掩码以及编码的说明见表 39。

表 39 – 外部事件掩码

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Event mask	0x0200	array [0..15] of unsigned1	RW	r	对每一个元素 0:禁止事件 1:使能事件

6.2 统计

6.2.1 RX 错误计数器

RX 错误计数寄存器包含物理层错误和帧错误（例如 长度，FCS(帧检验序列)）信息。往任何一个计数器中写入数据将清零所有计数器。当计数器值到达最大计数值（255）时将停止计数。

参数

Port 0 physical layer error count

该参数计算（端口 0 的）MII（媒体独立接口）层 RX 错误发生次数。

Port 0 frame error count

该参数计算（端口 0 的）帧错误（包括帧内部的 RX 错误）的发生次数。

Port 1 physical layer error count

该参数计算（端口 1 的）MII 层 RX 错误发生次数。

Port 1 frame error count

该参数计算（端口 1 的）帧错误（包括帧内部的 RX 错误）的发生次数。

Port 2 physical layer error count

该参数计算（端口 2 的）MII（媒体独立接口层）层 RX 错误发生次数。

Port 2 frame error count

该参数计算（端口 2 的）帧错误（包括帧内部的 RX 错误）的发生次数。

Port 3 physical layer error count

该参数计算（端口 3 的）MII（媒体独立接口）层 RX 错误发生次数。

Port 3 frame error count

该参数计算（端口 3 的）帧错误（包括帧内部的 RX 错误）的发生次数。

注：由于帧是在转发的过程中被处理的，所以一个 RX 错误或帧错误将同时出现在出错的站点及其之后的任何站点。主站通过减去上一端口的错误计数值来获得真实信息。

RX 错误计数器的属性类型如图 14 所示。

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

图 14 – RX 错误计数器类型描述

RX 错误计数器的编码如表 40 所示。

表 40 – RX 错误计数器

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Frame error count port 0	0x0300	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Physical error count port 0	0x0301	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Frame error count port 1	0x0302	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Physical error count port 1	0x0303	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Frame error count port 2	0x0304	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Physical error count port 2	0x0305	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Frame error count port 3	0x0306	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Physical error count port 3	0x0307	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器

6.2.2 丢失链接计数器

可选的丢失链接计数寄存器包含了链接丢失序列的信息。往任何一个计数器中写入数据将清零所有计数器。当计数器值到达最大计数值（255）时将停止计数。

参数

Port 0 lost link count

该参数计算（端口 0 的）MII 层的链接丢失发生次数。

Port 1 lost link count

该参数计算（端口 2 的）MII 层的链接丢失发生次数。

Port 2 lost link count

该参数计算（端口 3 的）MII 层的链接丢失发生次数。

Port 3 lost link count

该参数计算（端口 4 的）MII 层的链接丢失发生次数。

丢失链接计数器的属性类型如图 15 所示。

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

图 15 – 丢失链接计数器类型描述

丢失链接计数器的编码如表 41 所示。

表 41 – 丢失链接计数器

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Lost link count port 0	0x0310	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Lost link count port 1	0x0311	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Lost link count port 2	0x0312	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Lost link count port 3	0x0313	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器

6.2.3 附加计数器

可选的先前错误计数器的寄存器包含指示了后继链接有问题的错误帧的信息。因为错误帧包含一个特殊类型的校验和，所有可以被发现和报告。往任何一个计数器中写入数据会清零所有计数器。当计数器值到达最大计数值（255）时将会停止计数。

参数

Port 0 previous error count

该参数计算（端口 0）由后继站点发现的错误发生次数。

Port 1 previous error count

该参数计算（端口 1）由后继站点发现的错误发生次数。

Port 2 previous error count

该参数计算（端口 2）由后继站点发现的错误发生次数。

Port 3 previous error count

该参数计算（端口 3）由后继站点发现的错误发生次数。

可选的错误 EtherCAT 帧计数器对 FCS 正确但是数据报文结构错误的帧进行计数。往任何一个计数器中写入数据将清零所有计数器。当计数器值到达最大计数值（255）时将停止计数。

参数

Wrong EtherCAT frame counter

该参数计算 EtherCAT 帧错误的发生次数。

可选的本地问题计数器对发生的本地问题进行计数。往任何一个计数器中写入数据会清零所有计数器。当计数器值到达最大计数值（255）时将会停止计数。

参数

Local problem counter

该参数计算发生在从站内的通信问题次数。

附加计数器的属性类型如图 16 所示。

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

图 16 – 附加属性计数器类型描述

附加计数器的编码如表 42 所示。

表 42 – 附加计数器

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Previous Error Count Port 0	0x0308	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Previous Error Count Port 1	0x0309	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Previous Error Count Port 2	0x030A	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Previous Error Count Port 3	0x030B	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Malformat frame Count	0x030C	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器
Local Problem Count	0x030D	unsigned8	RW	R	往任何一个计数器写入数据将复位所有计数器

6.3 看门狗

6.3.1 看门狗分频器

从站控制器的系统时钟由看门狗分频器分频。

参数

Watchdog divider

该参数包含代表基本看门狗增量的、间隔单位为 40 纳秒的数值(-2)。（默认值为 100us = 2498）

看门狗分频器的属性类型如图 17 所示。

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

图 17 – 看门狗分频器类型描述

看门狗分频器编码如表 43 所示。

表 43 – 看门狗分频器

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Watchdog divider	0x0400	WORD	RW	R	40ns 间隔用于其他看门狗定时器

6.3.2 DLS-用户看门狗

DL用户看门狗的数值用于监视DL用户。任何从DL用户到从站控制器的访问必须先复位这个看门狗。

参数

DL-user watchdog

该参数应包含用于监视 DL 用户的看门狗（以 100µs 为单位的看门狗分配器的默认值 1000 意味着看门狗的时间是 100 ms）

DLS-用户看门狗的属性类型如图 18 所示。

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

图 18 – DLS-用户看门狗类型描述

DLS-用户看门狗的编码如表 44 所示。

表 44 – DLS-用户看门狗

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
DLS-user watchdog	0x0410	WORD	RW	R	

6.3.3 同步管理器看门狗

同步管理器看门狗的值用于监视每个同步管理器实体。如果看门狗选项被同步管理器使能，则每次对由同步管理器配置过的 DL 用户内存区进行的写访问都应复位看门狗。

参数

Sync manager watchdog

该参数包含监视同步管理器的看门狗。

同步管理器看门狗的属性类型如图 19 所示。

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

图 19 – 同步管理器看门狗类型描述

同步管理器看门狗编码如表 45 所示。

表 45 – 同步管理器通道看门狗

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Sync manager watchdog	0x0420	WORD	RW	R	

6.3.4 同步管理器看门狗状态

同步管理器看门狗状态包含了每个同步管理器看门狗的状态。

参数

Sync manager watchdog status

该参数包含所有同步管理器看门狗的看门狗状态。

同步管理器看门狗状态的属性类型如图 20 所示。

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

图 20 – 同步管理器看门狗状态类型描述

同步管理器看门狗状态的编码如表 46。

表 46 – 同步管理器看门狗状态

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Sync manager channel watchdog status	0x0440	Unsigned1	R	R	所有同步管理器共用一个看门狗标志 0: 看门狗溢出 1: 看门狗正常或未使能
reserved	0x0440	Unsigned15	R	R	

6.3.5 看门狗计数器

该可选的参数对看门狗的溢出次数计数

参数

Sync manager watchdog counter

该参数对所有同步管理器看门狗的超时进行计数。

DL-user watchdog counter

该参数记录 DL 用户看门狗的超时次数。

看门狗计数器的属性类型如图 21 所示。

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

图 21 – 看门狗计数器类型描述

看门狗计数器的编码如表 47 所示。

表 47 – 看门狗计数器

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Sync manager WD count	0x0442	unsigned8	RW	R	对这个计数器写将复位它
PDI WD count	0x0443	unsigned8	RW	R	对这个计数器写将复位它

6.4 从站信息接口

6.4.1 从站信息接口区

从站信息接口区的编码是由 DLS 用户特定的。

6.4.2 从站信息接口访问

从站信息接口访问的属性类型如图 22 所示。

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

图 22 – 从站信息接口访问类型描述

从站信息接口访问的编码如表 48 所示。

表 48 – 从站信息接口访问

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Owner	0x0500	Unsigned1	RW	R	0: EtherCAT 数据链路 1: 过程数据接口

Lock	0x0500	Unsigned1	RW	R	复位对 SII 的访问 0: 无动作 1: 取消访问 对这个置位将会复位寄存器 501.0
reserved	0x0500	Unsigned6	RW	R	
Access PDI	0x0501	Unsigned1	R	RW	0: 无访问 1: PDI 访问正在进行
reserved	0x0501	Unsigned7	R	RW	

6.4.3 从站信息接口控制/状态

通过从站信息接口控制/状态寄存器可以控制向从站信息接口读或写的操作。

参数

Slave information interface assign

该参数包含了分配给 DL 或 DL 用户的接口信息。

Reset slave information interface access

该参数复位对从站信息接口的访问。

Slave information interface access

该参数包含了从站信息接口是否被激活的信息。

Slave information interface read size

该参数包含了单个命令可以读取的八位位组数（4 或 8）的信息。

Slave information interface write access

该参数包含了是否允许对从站信息接口进行写访问的信息。

Slave information interface address algorithm

该参数包含了从站信息接口的协议包含一或两个地址八位位组的信息。

Read operation

该参数被主站写入，用于启动通过从站信息接口进行的 32 位或 64 位的读操作。主站将读取该参数以便检查读操作是否完成。

Write operation

该参数被主站写入，用于启动通过从站信息接口进行的 16 位的写操作。主站将读取该参数以检查写操作是否完成。写操作的一致性是无法保证的。

Reload operation

该参数被主站写入，用于启动通过从站信息接口进行的 32 或 64 位的重载操作。主站应读该参数以检查重载操作是否完成。

SII error

该参数包含当启动失败时需要的对 SII 参数的读访问信息。

Error command

该参数包含了上一次对从站信息接口访问是否成功的信息。

Busy

该参数包含了是否有访问操作正在进行的信息。

从站信息接口控制/状态的属性类型如图 23 所示。

```
typedef struct
{
  unsigned      WriteAccess:              1;
  unsigned      Reserved1:                5;
  unsigned      ReadSize:                 1;
  unsigned      AddressAlgorithm:         1;
  unsigned      ReadOperation:            1;
  unsigned      WriteOperation:           1;
  unsigned      ReloadOperation:          1;
  unsigned      CheckSErrDLu:             1;
  unsigned      DeviceInfoError:          1;
  unsigned      CommandError:             1;
  unsigned      WriteError:               1;
  unsigned      Busy:                     1;
} TSIICONTROL;
```

图 23 – 从站信息接口控制/状态类型描述

从站信息接口控制/状态类型的编码见表 49。

表 49 – 从站信息接口控制/状态

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
SII write access	0x0502	Unsigned1	RW	R	0x00: 仅对 SII 读访问 0x01: 对 SII 执行读和写访问
reserved	0x0502	Unsigned4	R	R	0x00
EEPROM emulation	0x0502	Unsigned1	R	R	0x00: 正常操作（I²C 接口被使用） 0x01: PDI 仿真 EEPROM（I²C 没有使用）
SII Read Size	0x0502	Unsigned1	R	R	0x00: 每次执行读 4 个八位位组 0x01: 每次执行读 8 个八位位组
SII Address Algorithm	0x0502	Unsigned1	R	R	0x00: 1 个八位位组作为地址 0x01: 2 个八位位组作为地址
Read operation	0x0503	Unsigned1	RW	RW	0x00: 没有读操作请求（参数写）或读操作不忙碌（参数读） 0x01: 有读操作请求（参数写）或读操作忙碌（参数读） 当开始一次新的读操作时，这个位必须有一次上升沿跳变
Write operation	0x0503	Unsigned1	RW	RW	0x00: 没有写操作请求（参数写）或写操作不忙碌（参数读） 0x01: 有写操作请求（参数写）或写操作请求忙碌（参数读） 当开始一次新的写操作时，这个位必须有一次上升沿跳变
Reload operation	0x0503	Unsigned1	RW	RW	0x00: 没有重载操作请求（参数写）或写操作不忙碌（参数读） 0x01: 有重载操作请求（参数写）或写操作请求忙碌（参数读） 当开始一次新的写操作时，这个位必须有一次上升沿跳变
Checksum Error	0x0503	Unsigned1	R	R	0x00:启动时加载 DL 用户信息没有校验和错误 0x01:启动时加载 DL 用户信息校验和错误
Device info error	0x0503	Unsigned1	R	R	0x00: 启动时读设备信息无错误 0x01: 启动时读设备信息有错误
Command error	0x0503	Unsigned1	R	R	0x00: 上一个命令无错误 0x01: 上一个命令有错误
Write error	0x0503	Unsigned1	R	R	0x00: 上一次写操作无错误 0x01: 上一次写操作有错误
Busy	0x0503	Unsigned1	R	R	0x00: 操作完成 0x01: 操作正在进行

6.4.4 实际从站信息接口地址

实际从站信息接口地址寄存器包含了将由被下一次读或写操作访问（通过写从站信息接口控制/状态寄存器）的从站信息接口的实际地址信息。

参数

Address

该参数包含被下次读或写操作访问的 16 位字的地址。

从站信息接口地址的属性类型如图 24 所示。

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

图 24 – 从站信息接口地址类型描述

实际从站信息接口地址编码如表 50 所示。

表 50 – 实际从站信息接口地址

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Address	0x0504	DWORD	RW	RW	16 位的字地址

6.4.5 实际从站信息接口数据

实际从站信息接口数据寄存器包含了在下一次写操作将要被写入从站信息接口的数据（16 位）或上次读操作所读到的数据（32 位或 64 位）。

参数

Data

主站会往该参数写入数据（16 位），该数据在下次写操作时会被写到从站信息接口。当主站读该参数时，可以得到上次通过从站信息接口获得的数据（32 位/64 位）。

从站信息数据的属性类型如图 25 所示。

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

图 25 – 从站信息接口数据类型描述

实际从站信息接口数据的编码如表 51 所示。

表 51 – 实际从站信息接口数据

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Data	0x0508	DWORD	RW	RW	当写操作时，只使用低 16 位 (0x508-0x509)

6.5 媒体独立接口（MII）

6.5.1 MII 控制/状态

MII 管理包含一个可选的属性集。MII 控制/状态寄存器的读或写操作是可控制的。

参数

MII write access

该参数包含一个到 MII 的写访问是否被允许的信息。如果要支持 MII 管理，读应始终被使能。

Address offset

该参数包含端口号和 MII 地址之间的偏移信息。

Read operation

该参数将由主机写入，用于在 MII 中启动 8 位读操作。该参数将被主机写入。该参数将被从主机读取，以检查读操作是否完成。

Write operation

该参数由主机写入，用于在 MII 中启动 8 位写操作，该参数由主机写入。该参数将被从主机读取，以检查写操作是否完成。写操作的一致性是没有保证的。中断写过程中可能会导致不一致的值，应避免遗漏关键的操作。

Error command

该参数应包含上次访问 MII 是否成功的信息。

Busy

该参数包含一个访问操作是否正在进行的信息。

图 26 描述了 MII 控制/状态的属性类型。

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

图 26 – MII 控制/状态类型描述

MII 控制/状态编码表见表 52。

表 52 – MII 控制/状态

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Write access	0x0510	Unsigned1	RW	R	0x00:只读访问 MII 0x01:可读写访问 MII
reserved	0x0510	Unsigned6	RW	R	0x00
PHYoffset	0x0510	Unsigned1	R	R	0x00(默认值)偏移被添加到 MII 地址的第 4 位 由本地配置设置。
Read operation	0x0511	Unsigned1	RW	R	0x00:没有读操作请求（参数写）或读操作不忙（参数读） 0x01:有读操作请求（参数写）或读操作忙（参数读） 要开始一个新的读操作，此参数必须有一个上升沿。
Write operation	0x0511	Unsigned1	RW	R	0x00:没有写操作请求（参数写）或写操作不忙（参数读） 0x01: 有写操作请求（参数写）或写操作忙（参数读） 要开始一个新的读操作，此参数必须有一个上升沿。
reserved	0x0511	Unsigned4	R	R	0x00
Write error	0x0511	Unsigned1	R	R	0x00:上一次写操作无错误 0x01:上一次写操作有错误
Busy	0x0511	Unsigned1	R	R	0x00:操作完成 0x01:操作正在进行

6.5.2 实际 MII 地址

实际的 MII 地址寄存器包含从站的 MII 寄存器中的实际地址，这个实际地址会被下一次读或写操作（通过写 MII 控制/状态寄存器）访问。

参数

Address PHY

该参数应包含由下一个读或写操作访问 PHY 地址。

Address PHY register

该参数包含被下一个读或写操作访问的 PHY 寄存器的地址。PHY 寄存器见 ISO / IEC8802-3 第 22 条。

图 27 描述 MII 地址的属性类型。

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

图 27 – MII 地址类型描述

实际 MII 地址编码见表 53。

表 53 – 实际 MII 地址

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Address PHY	0x0512	Unsigned8	RW	R	PHY（0,1）地址
Address register	0x0513	Unsigned8	RW	R	PHY 寄存器地址

6.5.3 实际 MII 数据

实际 MII 数据寄存器包含在下次写操作中被写入 MII 的数据（8 位）或上次读操作读到的数据（8 位）。

参数

Data

主站往该参数写数据，该数据在下次写操作时将要被写入 MII。当主站正在读此参数时，将接收上一次从 MII 读取的数据。

MII 数据的属性类型见图 28。

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

图 28 – MII 数据类型描述

MII 实际数据编码见表 54。

表 54 – MII 实际数据

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
Data	0x0514	Unsigned8	RW	R	

6.6 现场总线的内存管理单元(FMMU)

6.6.1 概述

现场总线内存管理单元（FMMU）通过内部地址将逻辑地址转换成物理地址。因此，FMMU 可实现逻辑寻址一个跨越多个从站设备的数据段：一个 DLPDU 可寻址任意分布的几个设备内的数据。FMMU 支持位映射。一个 DLE 可能包含几个 FMMU 实体。每个 FMMU 实体将一个连贯的逻辑地址空间映射到一个连贯的物理地址空间。

FMMU 最多包含 16 个实体。每个实体描述了 EtherCAT 通信网络的逻辑内存和从站物理内存之间的内存转换。

图 29 显示了一个例子，将逻辑地址 0x14711.3-14712.0 映射到八位位组存储器 1.1-1.6。

注：从左边最高位到右边最低位的值所代表的含义不意味着在传输线上的排序。

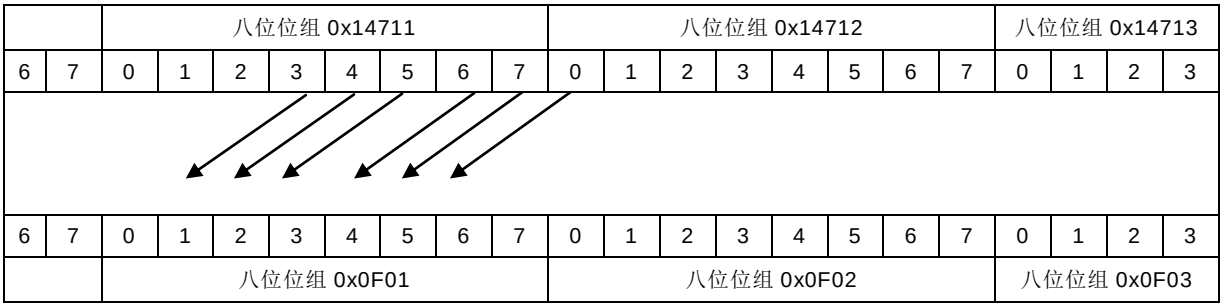


图 29 – FMMU 映射示例

6.6.2 FMMU 属性

参数

Logical start address

该参数应包含以八位位组表示的内存转换的逻辑存储区起始地址。

Logical start bit

该参数应包含逻辑起始地址的位偏移。

Logical end bit

该参数应包含逻辑结束地址的位偏移。

Physical start address

该参数应包含以八位位组表示的内存转换的物理存储区起始地址。

Physical start bit

该参数应包含物理起始地址的位偏移。

Length

该参数应包含内存转换的从第一个八位位组的逻辑地址空间（长度为 2 的映射）到最后一个八位位组的字节大小。

Read enable

该参数应包含一个读操作（物理存储器是源，逻辑存储器是目的地）是否被使能的信息。

Write enable

该参数应包含一个写操作（逻辑存储器是源，物理存储器是目的地）是否被使能的信息。

Enable

该参数应包含内存转换是否被激活的信息。

图 30 显示 FMMU 实体的属性类型。

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

图 30 – FMMU 实体类型描述

表 55 显示 FMMU 实体。表 56 显示 FMMU 结构。

表 55 – 现场总线内存管理单元(FMMU)实体

参数	相对地址（偏移）	数据类型	访问类型	访问类型 PDI	值/描述
Logical start address	0x0000	DWORD	RW	R	
Length	0x0004	WORD	RW	R	
Logical start bit	0x0006	Unsigned3	RW	R	
reserved	0x0006	Unsigned5	RW	R	0x00
Logical end bit	0x0007	Unsigned3	RW	R	
reserved	0x0007	Unsigned5	RW	R	0x00
Physical start address	0x0008	WORD	RW	R	
Physical start bit	0x000A	Unsigned3	RW	R	
reserved	0x000A	Unsigned5	RW	R	0x00
Read enable	0x000B	Unsigned1	RW	R	0x00: 读服务时忽略实体 0x01: 读服务时使用实体
Write enable	0x000B	Unsigned1	RW	R	0x00: 读服务时忽略实体 0x01: 读服务时使用实体
reserved	0x000B	Unsigned6	RW	R	0x00
Enable	0x000C	Unsigned1	RW	R	0x00:实体没激活 0x01: 实体激活
reserved	0x000C	Unsigned15	RW	R	0x0000
reserved	0x000E	WORD	RW	R	0x0000

表 56 – 现场总线内存管理单元(FMMU)

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/描述
FMMU entity 0	0x0600	TFMMU	RW	R	
FMMU entity 1	0x0610	TFMMU	RW	R	
FMMU entity 2	0x0620	TFMMU	RW	R	
FMMU entity 3	0x0630	TFMMU	RW	R	
FMMU entity 4	0x0640	TFMMU	RW	R	
FMMU entity 5	0x0650	TFMMU	RW	R	
FMMU entity 6	0x0660	TFMMU	RW	R	
FMMU entity 7	0x0670	TFMMU	RW	R	
FMMU entity 8	0x0680	TFMMU	RW	R	
FMMU entity 9	0x0690	TFMMU	RW	R	
FMMU entity 10	0x06A0	TFMMU	RW	R	
FMMU entity 11	0x06B0	TFMMU	RW	R	
FMMU entity 12	0x06C0	TFMMU	RW	R	
FMMU entity 13	0x06D0	TFMMU	RW	R	
FMMU entity 14	0x06E0	TFMMU	RW	R	
FMMU entity 15	0x06F0	TFMMU	RW	R	

6.7 同步管理器

6.7.1 同步管理器概述

同步管理器控制 DL 用户存储器的访问。每个通道定义一致的 DL 用户内存区。

主站和 PDI 之间的数据交换有 2 种方法：

- 握手模式（邮箱）：一个实体填充数据到 DL-用户内存区后，直到数据被其他实体读出，才能再次访问该区。
- 缓存模式：生产者的数据和消费者的数据之间的相互作用是不相关的 – 每个实体期望在任何时间进行访问，总是提供给消费者最新数据。

握手模式通过一个缓存区来实现：一个中断或一个状态标志表示缓存区是空还是满。

只有当携带读或写命令帧的 FCS 有效时，一个缓存区的交换才是有效的。如图 31 所示的交互原则。

缓存区的交换动作在第一个八位数组和最后一个八位数组处耦合：

- 如果缓存区是空的，在第一个八位位组处写数据能够使能对缓存区的写操作
- 通过写缓存区的最后一个八位位组，缓存区的状态将被设定为满
- 读出第一个八位位组的数据，为读操作准备缓存区
- 通过读出缓存区的最后一个八位位组来设定缓存区的状态为空

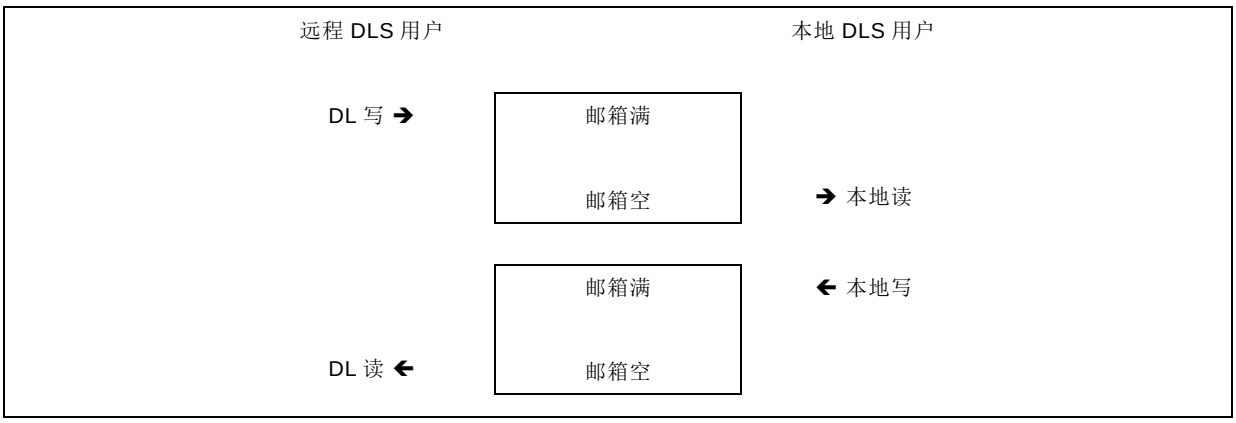


图 31 – 同步管理器邮箱交互

如果邮箱是满的，直到其被读出（即邮箱的最后一个八位位组将被读出）前都不能被写入。不管读出数据（可能被上层的时序约束）需要多长时间。

对于周期数据有不同的概念被实施，以确保数据的一致性和可用性。这是由一组缓存区完成，它允许写入和读取数据同时完成而不受干扰。两个缓存区被分配给发送方和接收方，一个备用缓存区作为中间存储区。

这意味着，在这种模式下，缓存区需要被扩充 3 倍。图 32 给出了一个开始地址为 0x1000，长度为 0x100 的配置。实际上其他缓存基本不可用。访问总是在缓存区 1 的寻址范围内进行。读最后一个八位位组或写最后一个八位位组将导致自动缓存交换（DL 侧：只有当缓存区的数据帧被正确接收时）。

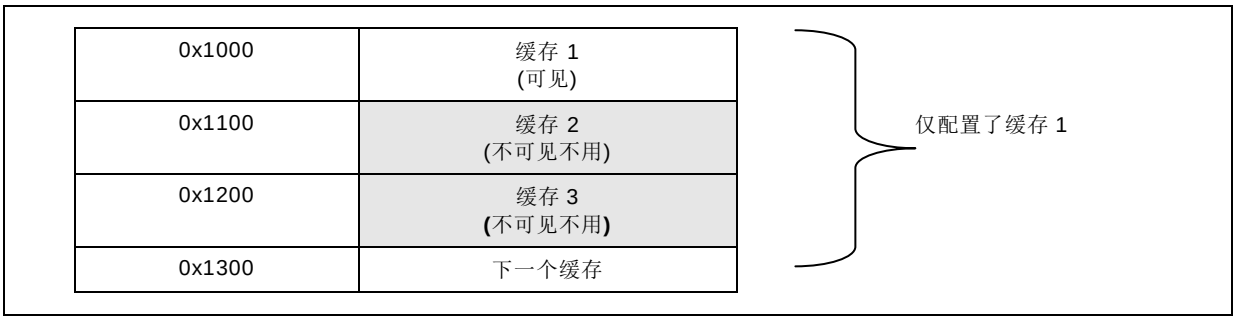


图 32 – SyncM 缓存区分配

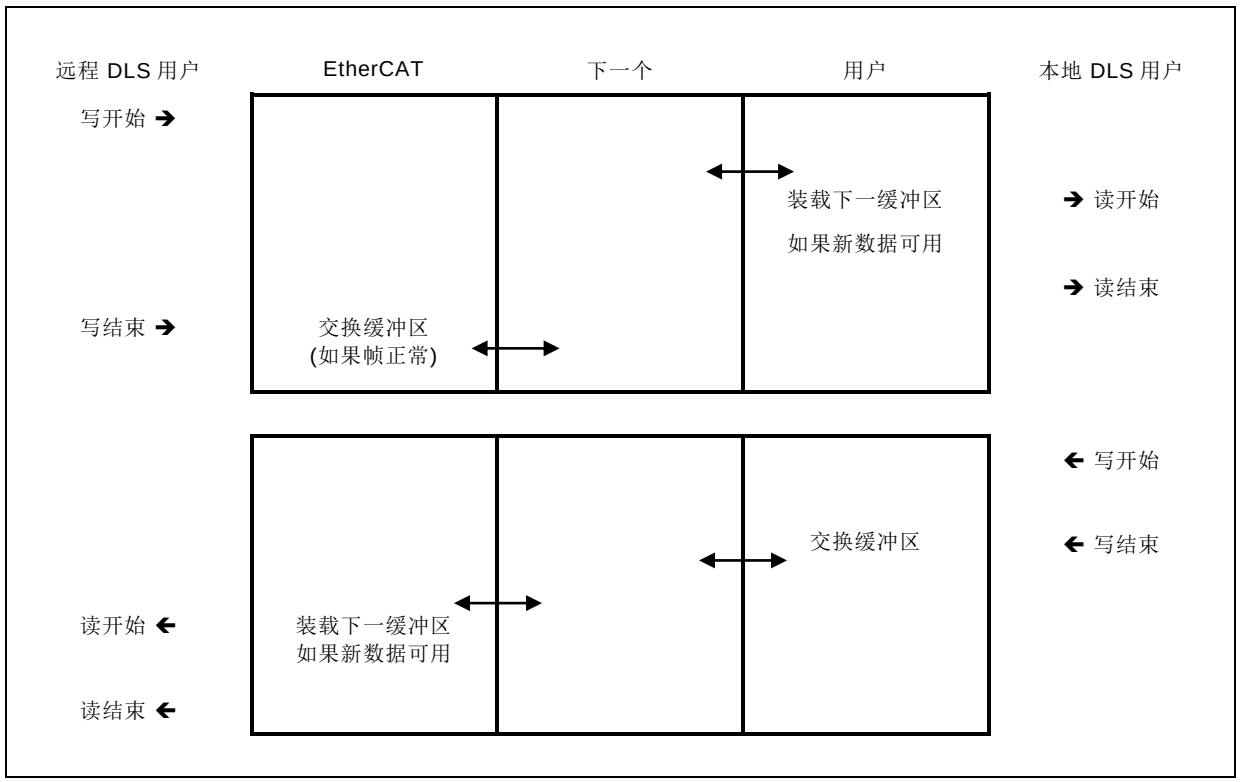


图 33 – SyncM 缓存区相互作用

该方案允许不受读写频率控制地访问缓存区。所以，主站的速度不影响从站的实现。

图 34 给出了一个读邮箱错误的交互示例。

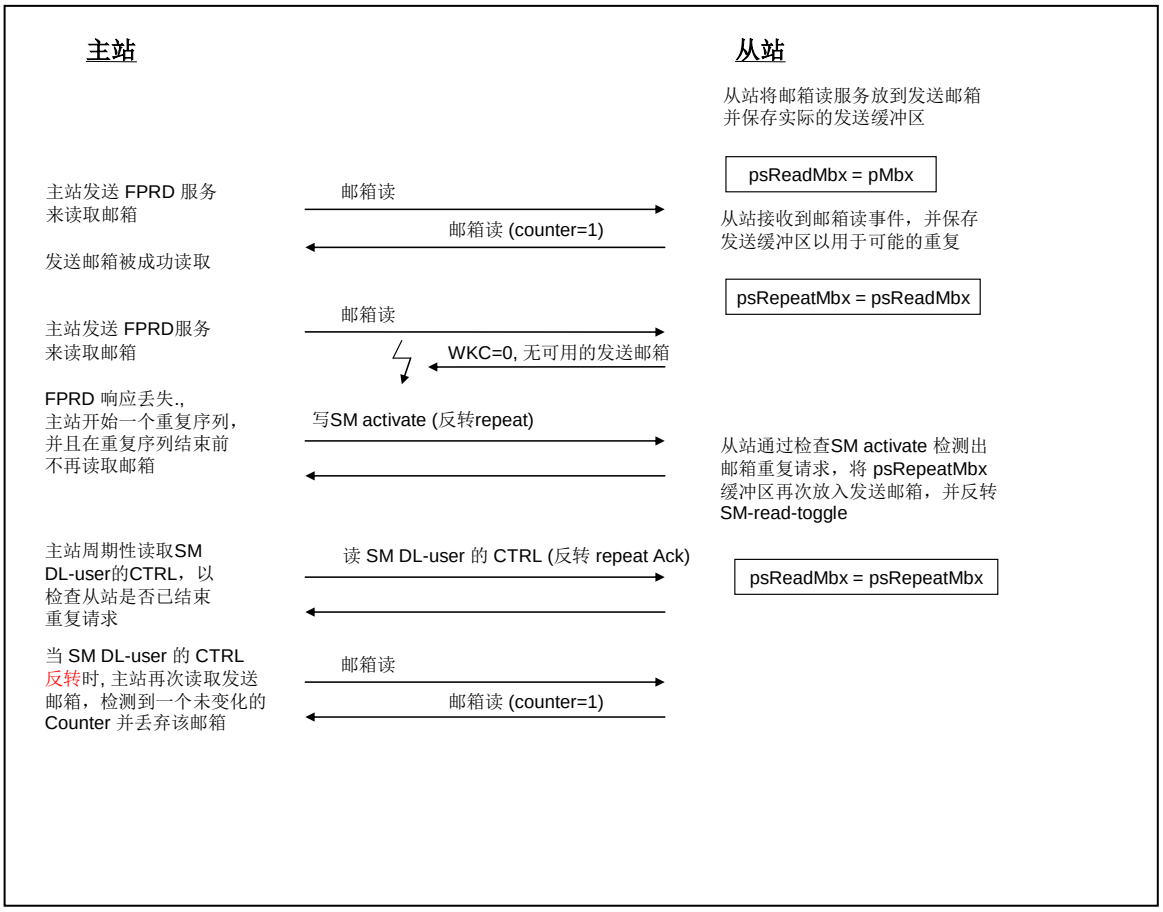


图 34 – 读邮箱的写/读切换处理

如果一个 EtherCAT DLPDU 丢失，这个切换位被用来重新同步邮箱通信，即以前丢失的只读邮箱入口将重新加载。

6.7.2 同步管理器属性

参数

Physical start address

该参数应包含相应的 DL 用户内存区的物理内存中以八位位组表示的起始地址。

Length

该参数包含相应的 DL 用户内存区的八位位组个数。

Operation mode

该参数应包含的信息为：相应的 DL 用户内存区访问类型是邮箱访问还是缓存访问。

Direction

该参数应包含的信息为：相应的 DL 用户内存区是被主站读还是写。

Event enable

该参数应包含的信息为：如果在相应的 DL 用户内存区中有一个被主站写入的新的有效数据（写方向）或者主机从 DL 用户读出一个新的数据（读方向）时，是否会有一个事件产生。

Watchdog enable

该参数应包含的信息为：访问相应的 DL 用户内存区域的监测是否被使能。

Write event

该参数应包含的信息为：是否连续的 DL 用户内存（方向写）已被主站写入，该事件使能参数已设置。

Read event

该参数应包含的信息为：是否相应的 DL 用户内存（方向读）已被主站读取，该事件使能参数已设置。

Mailbox access type state

该参数应包含的信息为：相应的 DL 用户内存状态（读缓存，写缓存）是否为邮箱访问类型。

Buffered access type state

该参数应包含的信息为：相应的 DL 用户内存状态（缓存号，是否锁定）是否为缓存访问类型。

Channel enable

该参数应包含的信息为：同步管理器通道是否激活。

Repeat

该参数的变化表明一个重复请求。这主要是用来重复上一个邮箱的相互作用。

DC Event 0 with EtherCAT write

该可选参数应包含的信息为：在一个 EtherCAT 写的情况下 DC0 事件是否会被请求。

DC Event 0 with local write

该可选参数应包含的信息为：在一个 EtherCAT 本地写的情况下 DC0 事件是否会被请求。

Channel enable PDI

该参数应包含的信息为：同步管理通道是否激活。

Repeat Ack

在该参数的变化表明一个重复请求确认。在参数中 Repeat 的值被设置后重复确认。

图 35 描述同步管理通道的属性类型

```
typedef struct
{
  WORD          PhysicalStartAddress;
  WORD          Length;
  unsigned      OperationMode:           2;
  unsigned      Direction:               2;
  unsigned      Reserved1:               1;
  unsigned      DLSuserEventEnable:            1;
  unsigned      WatchdogEnable:          1;
  unsigned      Reserved2:               1;
  unsigned      WriteEvent:              1;
  unsigned      ReadEvent:               1;
  unsigned      Reserved3:         1;
  unsigned      mailboxState:            1;
  unsigned      bufferState:             2;
  unsigned      Reserved4:               2;
  unsigned      ChannelEnable:           1;
  unsigned      Repeat:                  1;
  unsigned      Reserved5:               4;
  unsigned      DCEvent0wBusw:          1;
  unsigned      DCEvent0wlocw:           1;
  unsigned      ChannelEnablePDI:        1;
  unsigned      RepeatAck:               1;
  unsigned      Reserved6:               6;
} TSYNCMAN;
```

图 35 – 同步管理通道类型描述

表 57 规定同步管理器通道。

表 58 给出同步管理器结构。

表 57 – 同步管理器通道

参数	相对地址（偏移）	数据类型	访问类型	访问类型 PDI	值/参数
Physical start address	0x0000	WORD	RW	R	
Length	0x0002	WORD	RW	R	
Buffer type	0x0004	Unsigned2	RW	R	0x00: 缓存 0x02: 邮箱
Direction	0x0004	Unsigned2	RW	R	0x00: 被主站读的区域 0x01: 被主站写的区域
reserved	0x0004	Unsigned1	RW	R	0x00
DLS-user event enable	0x0004	Unsigned1	RW	R	0x00: DLS 用户事件没有激活 0x01: DLS 用户事件激活（此时存储区被访问不再被锁定）
Watchdog enable	0x0004	Unsigned1	RW	R	0x00: 看门狗未使能 0x01: 看门狗使能
reserved	0x0004	Unsigned1	RW	R	0x00
Write event	0x0005	Unsigned1	R	R	0x00: 没有写事件 0x01: 写事件
Read event	0x0005	Unsigned1	R	R	0x00: 没有读事件 0x01: 读事件

reserved	0x0005	unsigned1	R	R	0x00
Mailbox state	0x0005	Unsigned1	R	R	0x00: 邮箱空 0x01: 邮箱满
Buffered state	0x0005	Unsigned2	R	R	0x00: 第一缓存 0x01: 第二缓存 0x02: 第三缓存 0x03: 缓存锁定
reserved	0x0005	Unsigned2	R	R	0x00
Channel enable	0x0006	Unsigned1	RW	R	0x00:通道未使能 0x01: 通道使能
Repeat	0x0006	Unsigned1	RW	R	
reserved	0x0006	Unsigned4	RW	R	0x00
DC Event 0 with Bus write	0x0006	Unsigned1	RW	R	0x00:没事件 0x01:主站完成写缓存，DC 事件发生
DC Event 0 with local write	0x0006	Unsigned1	RW	R	0x00:没事件 0x01: DL 用户完成写缓存，DC 事件发生
Channel enable PDI	0x0007	Unsigned1	R	RW	0x00: 通道未使能 0x01: 通道使能
RepeatAck	0x0007	Unsigned1	R	RW	数据恢复后，应当遵循重复
reserved	0x0007	Unsigned6	R	RW	0x00

表 58 – 同步管理器结构

参数	物理地址	数据类型	访问类型	访问类型 PDI	值/参数
Sync manager channel 0	0x0800	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 1	0x0808	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 2	0x0810	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 3	0x0818	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 4	0x0820	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 5	0x0828	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 6	0x0830	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 7	0x0838	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 8	0x0840	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 9	0x0848	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 10	0x0850	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 11	0x0858	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 12	0x0860	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 13	0x0868	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 14	0x0870	TSYNCMAN	RW	R	最后一个字节 PDI 可写
Sync manager channel 15	0x0878	TSYNCMAN	RW	R	最后一个字节 PDI 可写

注：同步管理通道应以下列方式使用：

同步管理通 0：邮箱写
同步管理通 1：邮箱读
同步管理通 2：过程数据写
（如果没有过程的数据读取支持，仍可用于过程数据的读取）
同步管理通 3 过程数据读

如果邮箱不被支持，可使用以下方式：
同步管理通 0：过程数据写
（如果没有过程的数据读取支持，仍可用于过程数据的读取）
同步管理通 1：过程数据读

6.8 分布式时钟

6.8.1 概要

DC 被用于非常精确的时钟需求和产生不受通信周期控制的时钟信号。没有高同步要求的系统可以通过共享服务（更可取的有 LRW、LDR、LWR）或使用相同以太网帧来访问缓存器。

6.8.2 延迟测量

延迟测量需要关联到一个独立帧的时间戳信息。从站只是提供时间戳的方法，延时的计算是主站的任务。

参数

Receive time port 0

该参数包含一个特定数据报在端口 0 开始的接收时间。特定的数据报应写访问此参数。如果接收正确，此帧的接收时间将被写入在此数据报末尾的参数。此外，寄存器接收时间端口 1，2 和 3 的锁存将被相同的数据报使能。

Receive time port 1

该参数包含一个特定数据报在端口 1 开始的接收时间。特定的数据报是对寄存器接收时间端口 0 寄存器的写访问。如果接收正确，此帧的接收时间将被写入在此数据报末尾的参数。

Receive time port 2

该参数包含一个特定数据报在端口 2 开始的接收时间。特定的数据报是对寄存器接收时间端口 0 寄存器的写访问。如果接收正确，此帧的接收时间将被写入在此数据报末尾的参数。

Receive time port 3

该参数包含一个特定数据报在端口 3 开始的接收时间。特定的数据报是对寄存器接收时间端口 0 寄存器的写访问。如果接收正确，此帧的接收时间将被写入在此数据报末尾的参数。

6.8.3 本地时间参数

本地时间参数包含本地系统的时间和控制循环的参数，该参数被用来实现一个协调本地系统时间与全局时间的控制循环。

参数

Local system time

该参数包含的是，收到一个数据报时被锁存的本地系统时间。对此参数的写访问应启动锁定本地系统时间和写入参考系统时间的比较。比较的结果应作为本地系统时间 PLL 的输入。

System time offset

该参数包含本地系统的时间和全局时间之间的偏移量。

System time transmission delay

该参数包含从具有参考系统时间的从站控制器到本地控制器的传输延迟。

System time difference

该参数包含本地系统时间和上一次一个时间点之间的最新比较结果，这个时间点是上次写入的时间减去系统时间偏移再减去系统时间传输延迟得到的。

Control loop parameter base value

该参数表示控制循环算法的基值。该算法取决于执行的类型和调整速度的测量。值越小意味着调整速度越快但时钟越不稳定；值越大意味着时钟越稳定但调整越慢。

6.8.4 DL 用户时间参数

DL 用户时间参数包含 DL 用户的本地时间参数 DC-user P1 到 P12。

注：该参数的含义并不在此定义。它的访问权限在 ETG.1000.5 中指定。

6.8.5 DC 属性

分布式时钟延迟测量的属性类型包括本地时间参数中，其描述在图 36 中。

```
typedef struct
{
  DWORD          ReceiveTimePort0;
  DWORD          ReceiveTimePort1;
  DWORD          ReceiveTimePort2;
  DWORD          ReceiveTimePort3;
  UINT64         LocalSystemTime;
  BYTE           Reserved2[8];
  UINT64         SystemTimeOffset;
  DWORD          SystemTimeTransmissionDelay;
  DWORD          SystemTimeDifference;
  BYTE           Reserved3[5];
  DWORD          ControlLoopBaseValue;
  BYTE           Reserved3[74];
} TDCTRANSMISSION;
```

图 36 – 分布式时钟本地时间参数类型描述

表 59 规定分布式时钟本地时间参数。

表 59 – 分布式时钟本地参数

参数	物理地址（偏移）	数据类型	访问类型	访问类型 PDI	值/描述
Receive time port 0	0x0900	DWORD	RW	R	一个写访问在该参数（如果 PDU 被正确接收）的 PDU 端口 0 开始接收（开始报文头的第一部分）时应锁存本地时间（ns），同时使能端口 1 – 3 的锁存
Receive time port 1	0x0904	DWORD	RW	R	一个写访问在该参数的 PDU 的端口 1 开始接收时应锁存本地时间（ns）（前提是端口 1 锁存使能，PDU 被正确接收）
Receive time port 2	0x0908	DWORD	RW	R	一个写访问在该参数的 PDU 的端口 2 开始接收时应锁存本地时间（ns）（前提是端口 2 锁存使能，PDU 被正确接收）
Receive time port 3	0x090C	DWORD	RW	R	一个写访问在该参数在 PDU 的端口 3 开始接收时会锁存本地时间（ns）（前提是端口 3 锁存使能，PDU 被正确接收）
System time	0x0910	UINT64	RW	R	写访问比较 PDU 的端口 0 开始接收时锁存的本地系统时间（ns）和被写入的值（如果 PDU 被正确接收），比较结果将是 DC PLL 的输入
reserved	0x0918	BYTE[8]	RW	R	
System time offset	0x0920	UINT64	RW	R	本地时间（ns）和本地系统时间（ns）的偏移量
System time transmission delay	0x0928	DWORD	RW	R	参考系统时间（ns）和本地系统时间（ns）的偏移量
reserved	0x092C	BYTE[4]	RW	R	

表 60 描述了分布式时钟 DLS 用户参数的编码。

表 60 – 分布式时钟 DLS 用户参数

参数	物理地址（偏移）	数据类型	访问类型	访问类型 PDI	值/描述
reserved	0x0980	BYTE	R	R	0
DC user P1	0x0981	BYTE	RW	R	0
DC user P2	0x0982	Unsigned16	R	RW	0
reserved	0x0984	BYTE[10]	RW	R	0
DC user P3	0x098E	Unsigned16	R	R	0
DC user P4	0x0990	DWORD	RW	R	0
reserved	0x0994	BYTE[12]	R	R	0
DC user P5	0x09A0	DWORD	RW	R	0
DC user P6	0x09A4	DWORD	RW	R	0
DC user P7	0x09A8	Unsigned16	RW	R	0
reserved	0x09AA	BYTE[4]	R	R	0
DC user P8	0x09AE	Unsigned1	RW	R	0
DC user P9	0x09B0	DWORD	R	R	0
reserved	0x09B4	BYTE[4]	R	R	0
DC user P10	0x09B8	DWORD	R	R	0
reserved	0x09BC	BYTE[4]	R	R	0
DC user P11	0x09C0	DWORD	R	R	0
reserved	0x09C4	BYTE[4]	R	R	0
DC user P12	0x09C8	DWORD	R	R	0
reserved	0x09CC	BYTE[4]	R	R	0

7 DL 用户内存区

7.1 概述

系统复位后，理论上内存区可以在没有任何限制条件下用于通信和本地 DL 用户。即这个区域可有通信，但是通过这一机制数据处理可能不一致。

通过 SYNC 管理器 SM（SYNC manager）可以用协调的方式使用这个内存区。因为 SYNC 管理器是由主站创建, 而从站不能使用这个通信专用区域。

支持以下 2 种通信处理模式。

- 缓存模式，始终允许双向读写的操作。这种模式需要三个内存区域支持。本地的刷新率和通信周期都可以单独设定。
- 邮箱模式： 使用一个缓存区，实现了带有握手机制的数据交换。一个实体(通信或 DL 用户)写入数据，然后这个存储区被锁定直到被另一个实体读取数据。

7.2 邮箱访问类型

7.2.1 邮箱传送

邮箱传送服务是从主站关于读写方向(写操作是指由主站写入数据，读操作是指由主站读出数据)和从站关于服务描述的角度介绍的。这个操作包含了握手协议，也就是说，主站必须在发出服务请求之后等待从站的确认动作，反之亦然。

数据链路层指定了读写每一帧数据的复原服务。

写服务时，主站（客户端模式）向从站发出修改从站内存区的请求。如果这个被寻址的从站是可用的，并且邮箱是空的话，写服务将被确认。然后检查操作数是否重复。如果连续写入相同的值将只被执行一次。

这个数据将被更新并保存到有下一次数据更新。

7.2.2 主站写

图 37 表示了在主站，DLL 与 DL 用户之间成功写的操作原语。

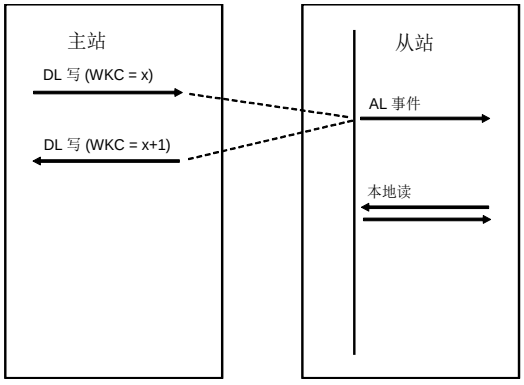


图 37 – 对邮箱的成功写过程

主站发送一个含工作计数器(WKC = x)的写服务， 然后从站的 DLL (从站控制器) 在 DL 用户的内存区写入该接收到的馈数据, 同时工作计数器加 1 (WKC = x + 1) 并生成一个事件。相应的 SM 通道锁定 DL 用户的内存区域直到被 DL 用户读取。因为 WKC 加 1，所以主站收到此次写操作成功成功的响应。 在 DL 用户读取 DL 用户内存区域时，相应的 SM 通道解锁该内存区域，保证了主站下一次的写入操作。

图 38 表示了在主站，DLL 与 DL 用户之间失败写的操作原语。

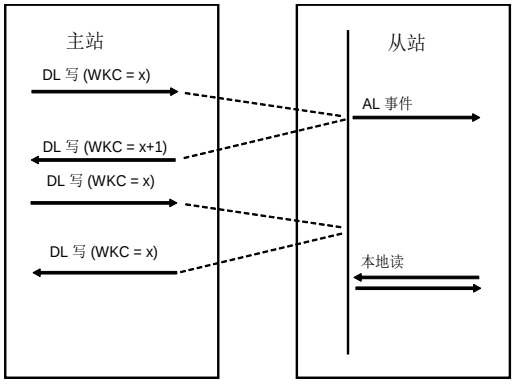


图 38 – 对邮箱的失败写过程

主站发送一个含工作计数器(WKC = x)的写指令， 然后从站的 DLL (从站控制器) 在 DL 用户的内存区写入一个反馈数据, 同时工作计数器加一 (WKC = x + 1)并生成一个事件. 相应的 SM 通道锁定 DL 用户的内存区域直到被 DL 用户读取。因为 WKC 加一，则表示主站此次操作成功. 在 DL 用户读取 DL 用户内存区域前，主站再次对该区域进行写入操作。因为这时 DL 用户内存区域仍然被锁定，从站的 DLL 将忽略主站这个操作，工作计数器将不再增加。这时主站将收到一个失败的反馈信息。之后在 DL 用户读取存储区域时，相应的 SM 通道解锁该内存区域，保证了主站下一次的写入操作。

7.2.3 主站读

图 39 表示了在主站，DLL 与 DL 用户之间成功读的操作原语。

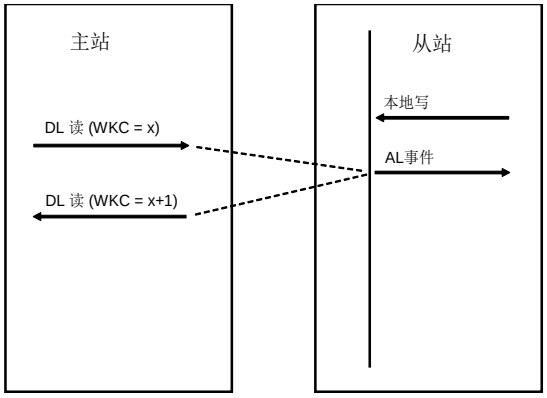


图 39 – 对邮箱的成功读过程

DL 用户更新了 DL 用户内存区域。相应的 SM 通道锁定 DL 用户的内存区域直到被主站读取。主站发送一个读取指令，从站的 DLL（从站控制器）发送 DL 用户内存区的数据，工作计数器自加一 (WKC = x + 1)，并对 DL 用户产生一个事件。因为 WKC 加一，表示主站此次操作成功 相应的 SM 通道解锁 DL 用户内存区域，保证了 DL 用户下一次的写入操作。

图 40 表示了在主站，DLL 与 DL 用户之间失败读的操作原语。

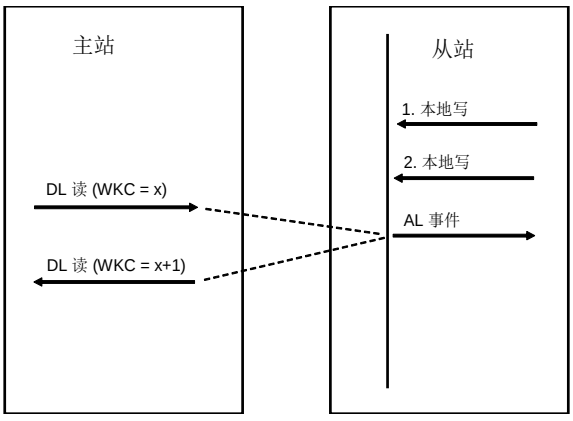


图 40 – 对邮箱的失败读过程

DL 用户更新了 DL 用户内存区域。相应的 SM 通道锁定 DL 用户的内存区域直到被主站读取。主站发送一个带工作计数器的读取指令，从站的 DLL（从站控制器）发送 DL 用户内存区的数据，工作计数器自加 1(WKC = x + 1)，并对 DL 用户产生一个事件。因为 WKC 加 1，主站收到此次操作成功的响应，相应的 SM 通道解锁 DL 用户内存区域，保证了 DL 用户下一次的写入操作。

7.3 缓存访问类型

7.3.1 主站写

图 41 表示了在主站、DLL 和 DL 用户之间连续写操作原语。该图表示了快速主站向较慢速率从站写操作的示例。

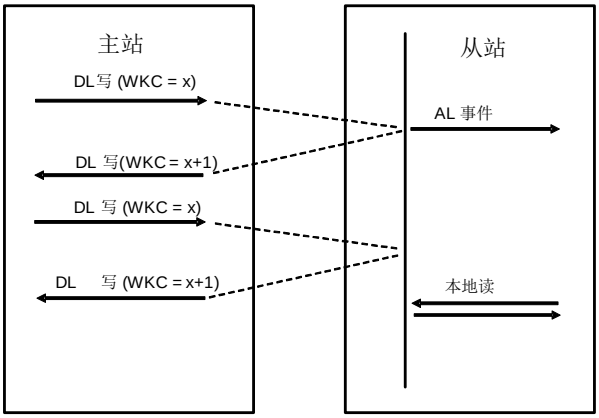


图 41 – 成功的写缓存过程

主站发送一个含工作计数器(WKC = x)的写请求，从站的 DLL（从站控制器）在 DL 用户内存区写入一个接收数据，工作计数器自加 1(WKC = x + 1)并对 DL 用户产生一个事件。因为 WKC 加 1，主站收到此次操作成功的响应。在 DL 用户读取内存区之前，主站再次执行写入指令，因为 缓存型的 DL 用户内存区域从来不锁定，从站的 DLL（从站控制器）重新在同一个 DL 用户内存区写入一个接收数据，工作计数器自加 1(WKC = x + 1)，同时再次对 DL 用户产生一个事件。因为 WKC 加 1，主站收到此次操作成功的响应。然后 DL 用户读取内存区域。

7.3.2 主站读

图 42 表示了在主站、DLL 和 DL 用户之间连续读操作原语。该图表示了从站在主站读取数据前多次更新数据的示例。

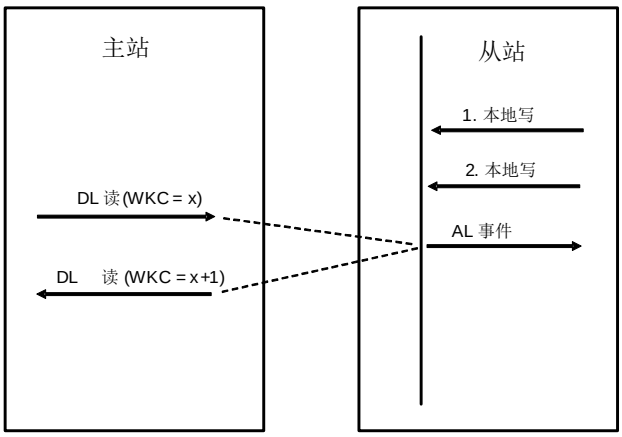


图 42 – 成功的读缓存过程

DL 用户更新内存区（1.本地写）。DL 用户用新值再次更新内存区（2.本地写），因为缓存型的 DL 用户内存区域从来不锁定，相应的 SM 通道就覆盖掉旧数据，主站发送一个带工作计数器的读指令，从站的 DLL（从站控制器）发送 DL 用户内存区的数据，工作计数器自加 1(WKC = x + 1)，并对 DL 用户产生一个事件。因为 WKC 加 1，主站收到此次操作成功的响应。

8 EtherCAT:FDL 协议状态机

8.1 从站 DL 状态机概述

图 43 一个 DL 从站的大致结构以及和各个状态机之间的交互。

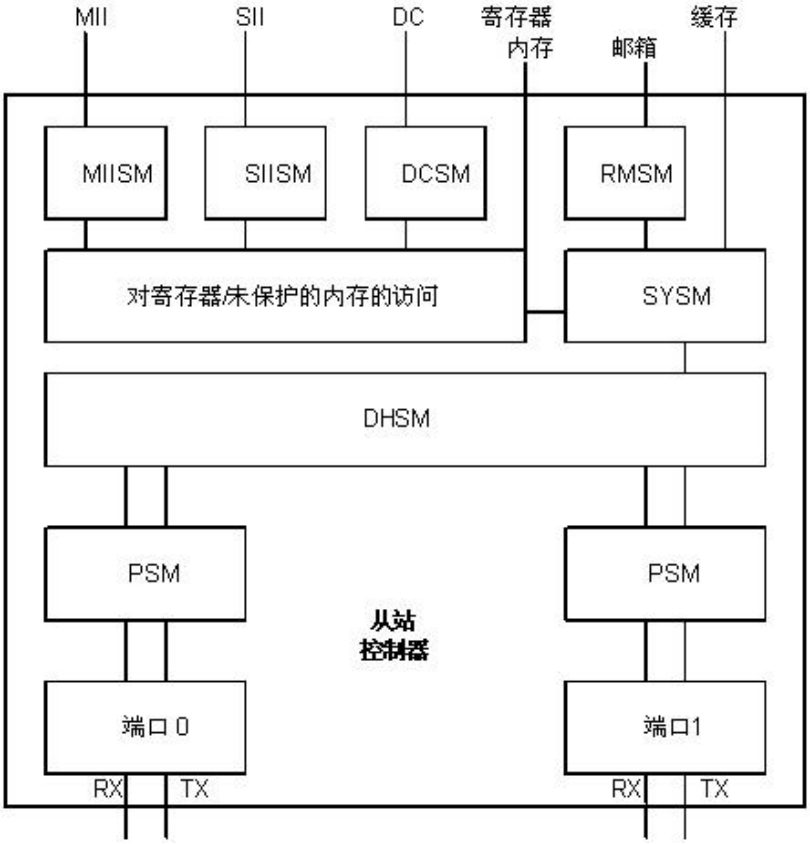


图 43 – 从站的协议机的结构

8.2 状态机描述

8.2.1 端口状态机(Port state machine, PSM)

PSM 协调下层端口状态机与 PDU 处理机，下层端口状态机用于处理 MAC 帧并将其以八位位组的长度传送到 PDU 处理程序。对于具有 2 个或更多 DL 接口的 DL 都分配一个状态机。然而对于没有明确定义的状态机端口可以参照以下示例并参照 ISO/IEC 8802-3。

a) 信息从 DL 接口以的传输是一个八位位组紧接着一个八位位组传输，而不是传输整个帧。

b) 如果一个端口没有链接 Tx.req 原语，那么将导致一个 Rx.ind 原语（在自动模式下使端口在自动状态，或者由指令关闭该回路)。

另外，ETG.1000.3 定义的统计计数器可以被 PSM 处理。

8.2.2 PDU 处理状态机(PDU handler state machine, DHSM)

DHSM 的处理方式是在第一个端口分拆以太网帧给单独的 EtherCAT PDU，在第二个端口 Receive Time 0 写请求，并将其映射到单独的寄存器或 SYSM（同步管理状态机）或 DCSM（DC 状态机）。FMMU 把全局地址映射到物理地址上，通过操作位于 DHSM 上的寄存器，激活 SIISM 和 MIISM。关于 DHSM 更多的细节可以参考附录 A.1。

8.2.3 同步管理器状态机(Synch manager state machine, SYSM)

同步信号管理器状态机处理被 SynchM 用作邮箱和缓存存储器的存储区域。邮箱服务被转发到一个处理重试的状态机 (恢复邮箱状态机 – RMSM)。对于每个 SM（SYNC Manager）都存在一个 SYSM 。访问内存时，只要没有激活的 SYSM 对应该地址，则访问将从一个 SYSM 转移到另一个 SYSM。 如果对应一个特殊的内存地址没有激活的 SYSM，则将有一个对存储区域或寄存器的请求。一些特定的访问规则适用于寄存器–关于这些寄存器的描述在 GB/T XXXXX.3 里可以找到。关于 SYSM 规格更多的细节可以参考附录 A.2。

8.2.4 恢复邮箱状态机(Resilient mailbox state machine, RMSM)

RMSM 是负责在操作读邮箱过程中进行重试操作和检查写邮箱指令的序列号。一个写入邮箱的重试操作就是将同一个序列号再次写入的过程。通过写入一个非零序列号可激活重试机制。。

读邮箱的重试机制使用 SM(SYNC manager)通道的 Repeat 和 RepeatAck 的参数，Repeat 参数中的 toggle 触发从站进行最后一次读取的重试操作。A.3 有对读邮箱属性的具体描述。

8.2.5 SII 状态机（SIISM）

8.2.5.1 从站信息交互接口操作流程图

SIISM 负责访问 SII。在这个端口上有指定的读取、写入、重新加载操作。主站可以根据特定程序来激活这些操作。

8.2.5.2 读操作

图 44 示出了读操作流程。

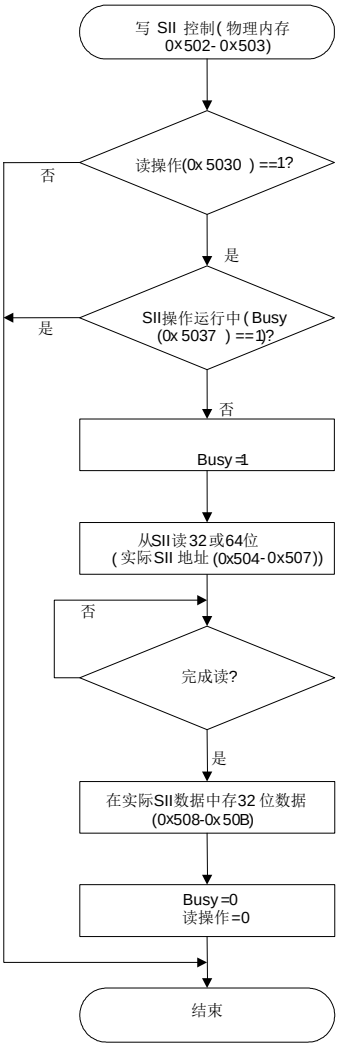


图 44 – SII 读操作

8.2.5.3 写操作

图 45 示出了写操作流程。

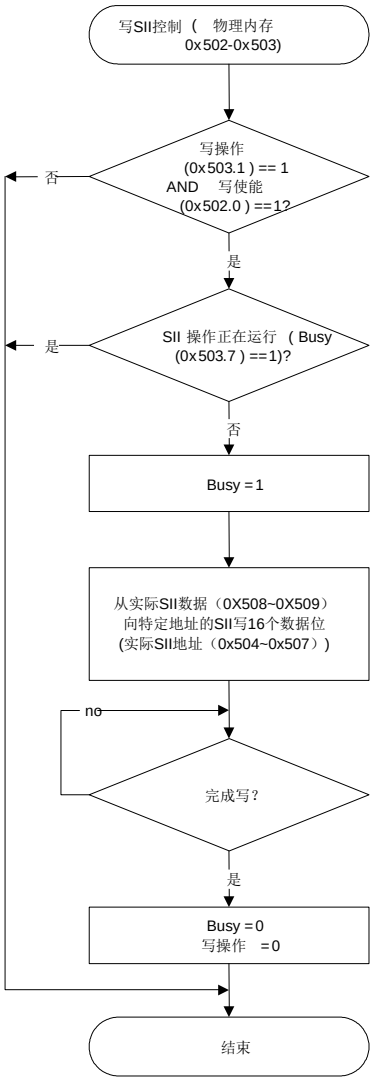


图 45 – SII 写操作

8.2.5.4 重新加载操作

图 46 示出了重新加载流程

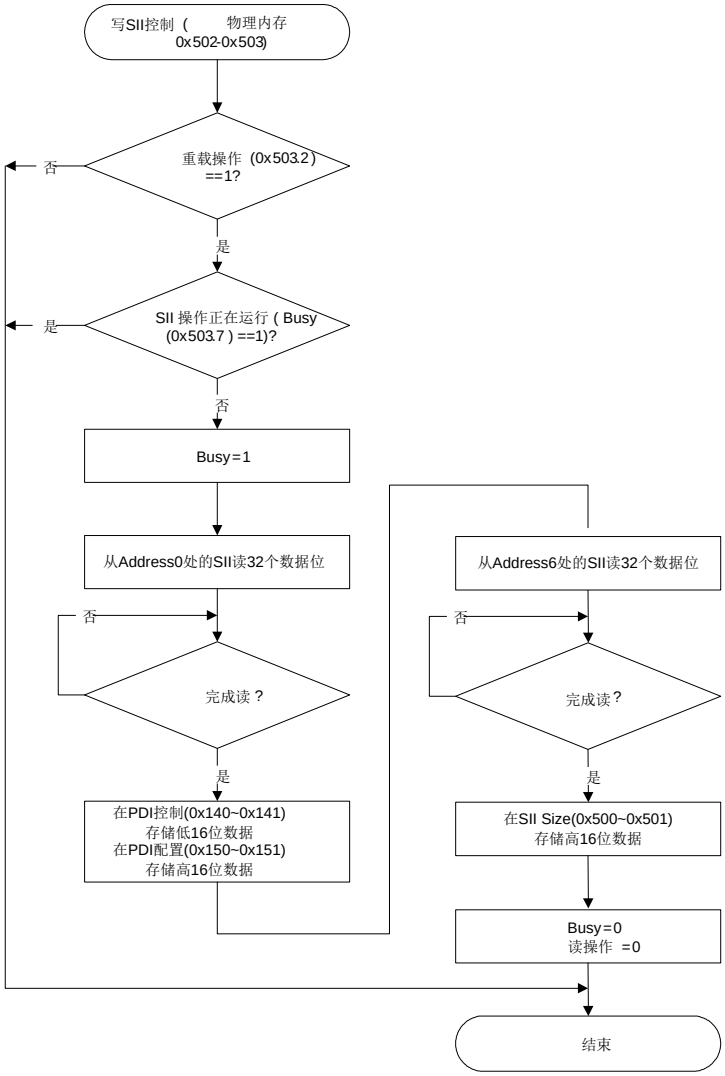


图 46 – SII 重新加载操作

8.2.6 MII 状态机（MIISM）

MIISM 负责访问 MII（基于 ISO/IEC 8802-3 的媒体独立接口）。流程遵循 8.2.5 中规定的结构，其命令、地址和数据缓存区都有具体的地址。

8.2.7 DC 状态机（DCSM）

8.2.7.1 DC 结构概述

DCSM 处理协调本地时钟、本地时钟同步和时间戳。DC 寄存器在 ETG.1000.3 中有介绍。

分布式时钟可以使所有从站设备具有相同的时间。网段内的第一个从站设备所包含的时钟作为参考时钟。其作用是同步其他从站设备的从站时钟与主站设备的时钟。主站设备每隔一定时间发送一个同步 PDU(为了避免从站时钟超出应用规定的范围)，有参考时钟的从站设备将自身的当前时间写入该同步 PDU。然后带有从站时钟的其他从站设备通过 ARMW 服务从同一 PDU 读取时间。由于逻辑环结构，这种情况是有可能发生的，因为参考时钟位于该网段的其他从站时钟之前。

由于每个从站设备在收发时都会产生一个很小的延时 (包括设备内部和物理连接的延时)，所以在同步从站时钟时，应该考虑参考时钟与各个从站时钟之间的传输延时。为了测量传输延时，主站设备向一个特定的地址发送一个广播写入命令 （端口 0 的接收时间寄存器），这将使每个从站设备在传输过程中一旦接收到该 PDU（或本地时钟时间），就保存那个时间，主站可以读取这些保存的时间并以此来设置一个延时寄存器。

参考时钟定义

一个从站的时钟将被用来作为参考时钟。这个参考时钟是位于主站和所有要被同步的从站之间的第一个时钟。参考时钟用 ARMW 或 FRMW 命令周期地发布它的时钟，参考时钟可以根据像 ISO/IEC 61588 这样的“全球”参考时钟进行调整。

前提

目前没有一个机制用来计算驻留时间。因此，在所有 EtherCAT 的从站设备上都不允许驻留时间大幅抖动。

关键点

- 参考时钟偏移补偿
- 传输延时测量

  每个从站控制器测量 2 个方向上每一帧的延时

  主站则计算所有从站之间的传输延时
- 参考时钟偏移补偿(系统时间)

  在所有设备上有唯一的绝对时间 (误差小于 1µs)

图 47 给出一个 DC 单元的结构概述。假设本地时钟频率为 100MHz，时钟的分辨率为 1ns。这允许以很小的时间间隔调整本地时钟去接近全局时钟频率，以避免在时间刻度上出现大的跳动。

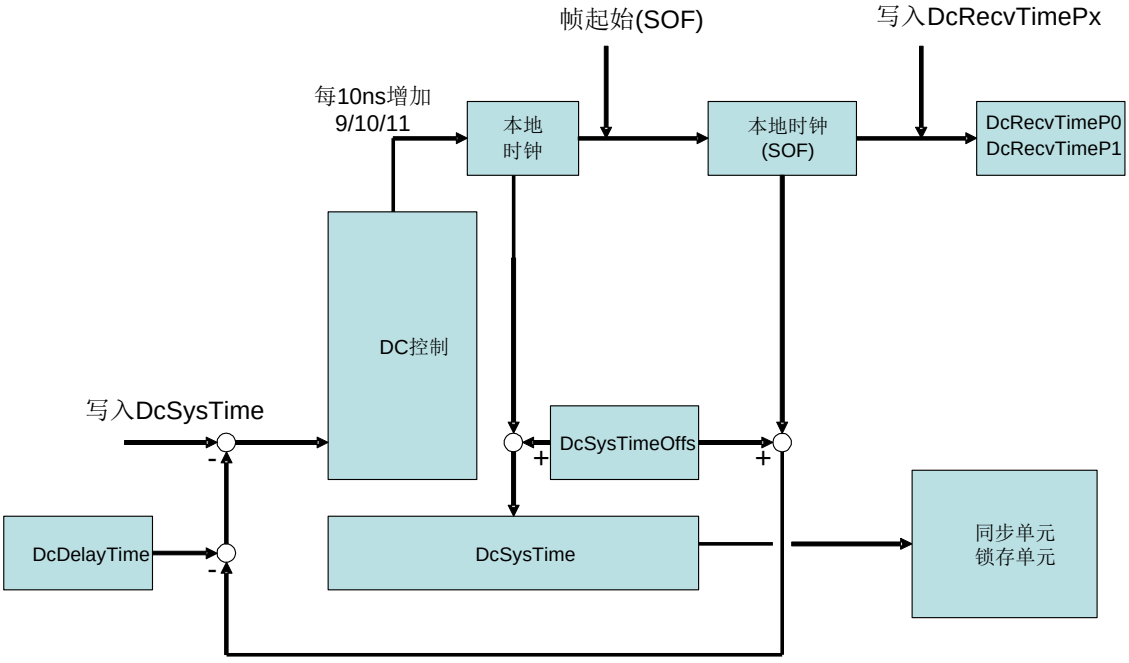


图 47 – 分布式时钟

SOF 是每一个以太网帧的前导码的起始。

本地时钟每 10ns 递增 10，取决于时钟的偏移，每 10ns 递增 9 到 11 个 DC 控制的常规时基（用于漂移补偿）。.

SysTimeOffset 允许在不改变本地时钟的运行的前提下进行调整。延迟作为第二偏移用于补偿参考时钟到从站时钟的延迟。

外部同步是由 GB/T 25931 规定的机制完成的。任何有外部通信接口的设备都包含一个边界时钟。从站用主站时钟来同步到边界时钟。每个 EtherCAT 网段应在任何时候只有一个活动的边界时钟，以满足 IEC 61588 拓扑要求。

DC 用于非常精确的定时要求。系统同步需要在 10µs 范围内，或采用其他延时补偿的方法，可以是通过共享 EtherCAT 的 PDU，访问需要同步的设备写缓存区来同步。

8.2.7.2 延时测量

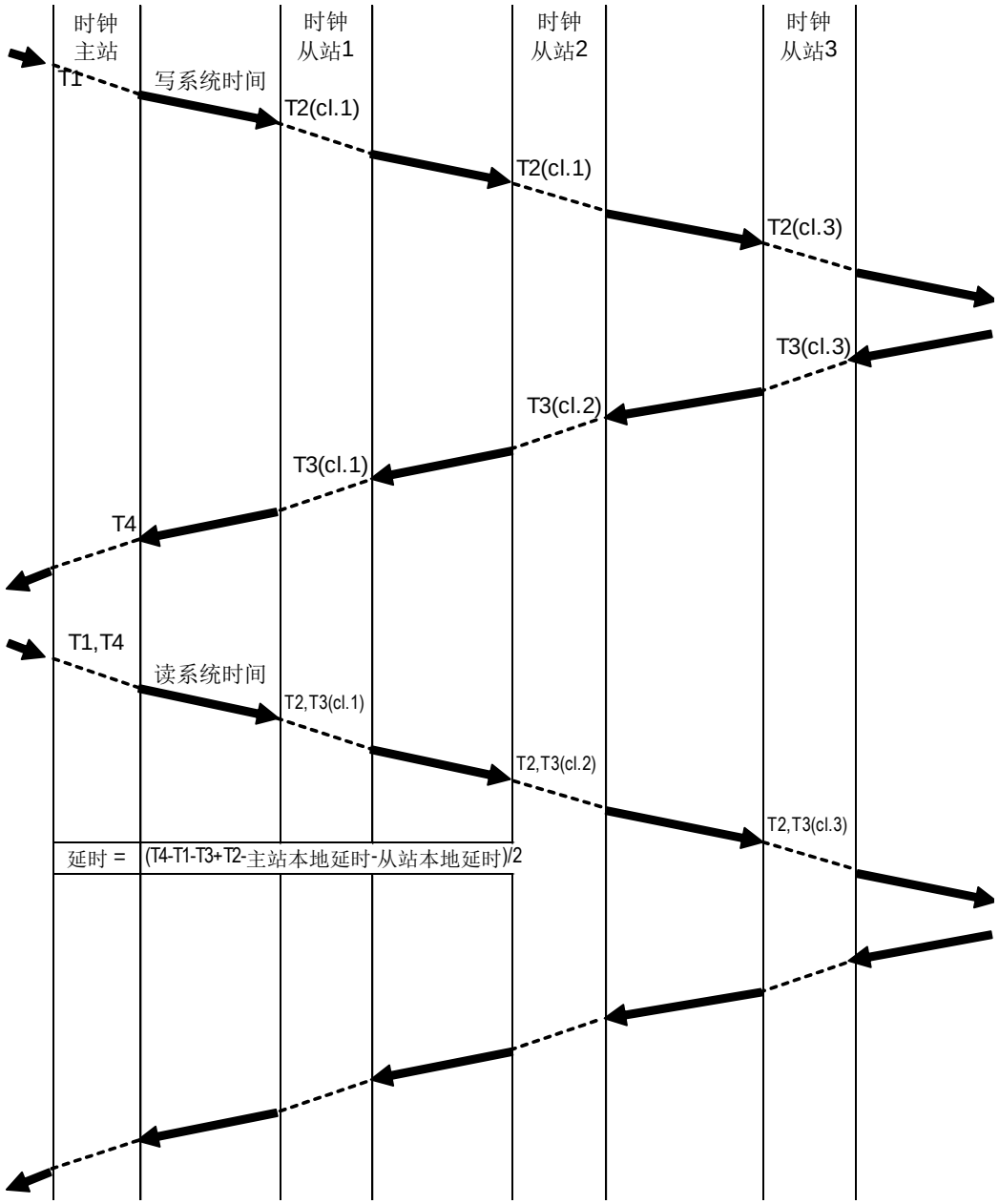


图 48 – 延时测量序列

T1(主站时钟)和 T2(各从站时钟 1,2,3 表示为 cl.1, cl.2 和 cl.3) 参考接收时间端口 0，T3(各从站时钟 1,2,3 表示为 cl.1,cl.2 和 cl.3) 和 T4 参考接收时间端口 1。最后的从站时钟 T2=T3。该模型假定对称连接即路径从 A 到 B 和 B 到 A 是等长的。不同线缆之间的延时和在以太网物理层的不同传输延时可产生一个时间偏移常数。

附件 A
(资料性附录)
–
EtherCAT:DL 协议状态机的附加规范

注 1：本附录规定了很多由 DLE 使用的提供低层和高层协议功能的有限状态机。本规范是本标准正文规范的补充，在冲突的情况下优先考虑正文规范。

注 2：这里给出的有限状态机描述是其具体实现的完整描述的简化。附加要求和要考虑的事项可以查看正文规范。

A.1 DHSM

A.1.1 原语定义

A.1.1.1 PSM 和 DHSM 之间的原语交换

表 A.1 给出了由 DHSM 到 PSM 的原语。

表 A.1 – 由 DHSM 到 PSM 的原语

原语名称	相关参数
Tx request	Port, Byte, Data

表 A.2 给出了由 PSM 到 DHSM 的原语。

表 A.2 – 由 PSM 到 DHSM 的原语

原语名称	相关参数
Rx indication	Port, Byte, Data

A.1.1.2 PSM 原语的参数

表 A.3 给出了 DHSM 和 PSM 之间的原语所使用的所有参数。

表 A.3 – DHSM 和 PSM 之间原语交换的所有参数

参数名称	描述
Port	本地端口标识符，作为主端口从 0 开始
Byte	接收/发送字节的标识符，在表 A.4 中指定
Data	接收/发送的八位位组的值

表 A.4 – 以太网帧八位位组的标识符

参数	描述
0	前导码的第一个八位位组
1	前导码的后继八位位组
2	DA 的第一个八位位组
3	DA 的后继八位位组
4	DA 的后继八位位组
5	SA 的后继八位位组
6	VLAN 的第一个八位位组
7	VLAN 的后继八位位组
8	Ethertype 的第一个八位位组
9	Ethertype 的后继八位位组
10	Ethernet SDU 的第一个八位位组
10+ n	Ethernet SDU 第 n+1 八位位组
0xffff (END)	带正确 FCS 校验的帧尾
0xfffe (ERR)	有差错的中止帧

A.1.2 状态机描述

每个从设备都有一个 DHSM。

DHSM 构成远程互动和本地内存之间的接口。每帧的八位位组将由 DHSM 从端口传送到端口。

如果 EtherCAT 的命令被识别且本地内存是可寻址的，将产生基于 SYM 状态机的交互。如果端口 0 发出指示，本地动作将被调用。其他端口上发出的惟一的本地动作是入向帧的时间戳。

状态机描述具有特定的实时 EtherType 或特定的 UDP 目的端口的以太网帧的解释。特定的 EtherCAT 处理循环帧的的检测和增量自动增量的地址，更新 WKC 和 FCS 检测将由 DHSM 完成。

错误处理在逻辑层被描述。为更好地定位错误链接，检测到一个错误的站将转发损坏的 FCS，4 位的物理符号将引发一个定位错。这一定位错与最后 2 比特取反的 FCS 相结合，将发出在一个差分链接上出现问题的信号。每个检测到的错误将导致在合适的统计属性中产生一个附加的登录项。

本地常量

LASTP
最末的以太网端口的标识符。端口编号从 0 到 LASTP。

END
以太网帧结束的指示符。

ERR
由于差错导致以太网帧结束的指示符。

本地变量

CMD
EtherCAT PDU 的命令标识符。

Etype1
Ethertype 的第一个八位位组。

Length
EtherCAT PDU 的长度。

MF
表示在以太网帧中的最后一个 EtherCAT PDU。

RxTimeLatch[0..LASTP]
EtherCAT PDU 的长度。

AdL
EtherCAT PDU 的第一个地址八位位组。

AdH
EtherCAT PDU 的第二个地址八位位组。

DaL
EtherCAT PDU 的第三个地址八位位组。

DaH
EtherCAT PDU 的第四个地址八位位组。

LeL
EtherCAT PDU 的第一个长度八位位组。

LeH
EtherCAT PDU 的第二个长度八位位组。

wkc
在 EtherCAT 的 PDU 中，被添加到 WKC 的本地添加因素。

RData
存储器元件的本地数据八位位组。

WData
被写入的一个 EtherCAT PDU 的数据八位位组。

Overflow
指示一个双八位位组的无符号整数的加法溢出。

状态表术语

标准后缀“.req”, “.cnf” and “.ind”是分别用来表示请求，证实和指示原语。

A.1.3 DHSM 表

表 A.5 是 DHSM 状态表。

表 A.5 – DHSM 状态表

#	当前状态	事件/状况⇒动作	下一状态
1	ETH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ETH
2	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 0 => RxTimeLatch[Port] = CT Port = 1 Tx.req(Port,Byte,Data)	ETH
3	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 1 => Port = 1 Tx.req(Port,Byte,Data)	ETH
4	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 2 => Port = 1 INIT_FCS(Data) Tx.req(Port,Byte,Data)	ETH
5	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 3 => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
6	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 4 => Port = 1 Data = Data | 2 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
7	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 4 && Byte < 8 => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
8	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 8 => Port = 1 Etype1 = Data UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
9	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data== 0xA4 && Etype1== 0x88) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECAT

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
10	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data== 0x00 && Etype1== 0x08) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIP
11	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte = 9 && (Data != 0xA4 || Etype1 != 0x88) && (Data != 0x00 || Etype1 != 0x08) => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ETH
12	ETH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 9 => Port = 1 Tx.req(Port,Byte,Data)	ETH
13	EIP	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIP
14	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte < 10 => Port = 1 Byte = ERR Tx.req(Port,Byte,Data)	ETH
15	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && (Byte == END || Byte == ERR) => Port = 1 Tx.req(Port,Byte,Data)	ETH
16	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 && Data == 0x45 => Port = 1 Tx.req(Port,Byte,Data)	EIP
17	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 && Data !=0x45 => Port = 1 Tx.req(Port,Byte,Data)	ETH
18	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 10 && Byte < 19 => Port = 1 Tx.req(Port,Byte,Data)	EIP
19	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 19 && Data == 0x11 => Port = 1 Tx.req(Port,Byte,Data)	EIP
20	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 19 && Data !=0x11 => Port = 1 Tx.req(Port,Byte,Data)	ETH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
21	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 19 && Byte < 32 => Port = 1 Tx.req(Port,Byte,Data)	EIP
22	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 32 && Data == 0x88 => Port = 1 Tx.req(Port,Byte,Data)	EIP
23	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 32 && Data != 0x88 => Port = 1 Tx.req(Port,Byte,Data)	ETH
24	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 33 && Data == 0xA4 => Port = 1 Tx.req(Port,Byte,Data)	EIP
25	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 33 && Data != 0xA4 => Port = 1 Tx.req(Port,Byte,Data)	ETH
26	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte > 33 && Byte < 36 => Port = 1 Tx.req(Port,Byte,Data)	EIP
27	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 36 => Port = 1 Data = 0 Tx.req(Port,Byte,Data)	EIP
28	EIP	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 37 => Port = 1 Data = 0 Tx.req(Port,Byte,Data)	ECAT
29	ECAT	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECAT
30	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte < 10 => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
31	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 10 => Len = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECAT

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
32	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 11&& (Data & 0xf0 == 0x10) => Len = Len + (Data*256 & 0x0f) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ECMD
33	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == 11&& (Data & 0xf0 != 0x10) => Port = 1 Tx.req(Port,Byte,Data)	ETH
34	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
35	ECAT	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
36	ECMD	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECMD
37	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 => CMD = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIDX
38	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
39	ECMD	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
40	EIDX	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIDX

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
41	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 => Cmd = Data Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADL
42	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
43	EIDX	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
44	EADL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EADL
45	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == BRD, BWR, BRW, APRD, APWR, APRW,ARMW => AdL = Data Data = Data + 1 Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADH
46	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == other => AdL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EADH
47	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
48	EADL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
49	EADH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EADH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
50	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == BRD, BWR, BRW => AdH = Data if AdL == 0xff then Data = Data + 1 Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
51	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == APRD, APWR, APRW,ARMW => AdH = Data if AdL == 0xff then Data = Data + 1 if Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
52	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == FPRD, FPWR, FPRW,FRMW => AdH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
53	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == LRD, LWR, LRW => AdH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
54	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMD == other => Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAL
55	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
56	EADH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
57	EDAL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDAL
58	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 => DaL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDAH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
59	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
60	EDAL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
61	EDAH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDAH
62	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 => DaH = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ELEL
63	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
64	EDAH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
65	ELEL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ELEL
66	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 => LeL = Data Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	ELEH
67	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
68	ELEL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
69	ELEH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ELEH
70	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && (!Closed[Port] || Data & 0x40 == 0) => if Closed[Port] then LeH = Data | 0x40 else LeH = Data MF = LeH & 0x80 Length = LeL + 256 * (LeH & 0x0f) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIRL
71	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && (Closed[Port] && Data & 0x40 != 0) => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
72	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
73	ELEH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
74	EIRL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIRL
75	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 => if Ena then Data == Data | (EventH & EventMskH) Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EIRH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
76	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
77	EIRL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
78	EIRH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EIRH
79	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length == 0 => wkc = 0 Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
80	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 && !Ena => wkc = 0 Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTI
81	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 && Ena => wkc = 0 Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
82	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
83	EIRH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
84	EDTI	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDTI
85	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length == 0 => Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
86	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Length != 0 => Length -- Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTI
87	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
88	EDTI	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
89	EDTA	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EDTA
90	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && R_FMMUMATCH(LOGADD) => MemoryAddress = RFMMUMAP (LOGADD) WKC = 0 WData= Data Read.ind (Address, Data, WKC)	WSYLR
91	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && W_FMMUMATCH(LOGADD) => MemoryAddress = WFMMUMAP (LOGADD) WKC = 0 RData= Data Write.ind (Address, Data, WKC)	WSYLW
92	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && N_FMMUMATCH(LOGADD) && Length == 0 => INC(LOGADD) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
93	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDL && N_FMMUMATCH(LOGADD) && Length != 0 => Length -- INC(LOGADD) Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
94	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDPR => WKC = 0 RData= Data Read.ind (Address, Data, WKC)	WSYPR
95	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && CMDPW => WKC = 0 WData= Data Write.ind (Address, Data, WKC)	WSYPW
96	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
97	EDTA	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
98	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /W_FMMUMATCH(LOGADD) => wkc = wkc | WKC MemoryAddress = WFMMUMAP (LOGADD) RData = Data Data = WData WKC = 0 Write.ind (Address, Data, WKC)	WSYLW
99	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /!W_FMMUMATCH(LOGADD) && Length == 0 => wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
100	WSYLR	**Read.rsp (MemoryAddress, Data, WKC)** /!W_FMMUMATCH(LOGADD) && Length != 0 => Length -- INC(LOGADD) wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
101	WSYLW	**Write.rsp (MemoryAddress, Data, WKC)** /Length == 0 => INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
102	WSYLW	**Write.rsp (MemoryAddress, Data, WKC)** /Length != 0 => Length -- INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
103	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /CMDPW => wkc = wkc | WKC if OR then Data = WData | Data RData = Data Data = WData WKC = 0 Write.ind (Address, Data, WKC)	WSYPW
104	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /!CMDPW && Length == 0 => if OR then Data = WData | Data wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
105	WSYPR	**Read.rsp (MemoryAddress, Data, WKC)** /!CMDPW && Length != 0 => Length -- if OR then Data = WData | Data wkc = wkc | WKC Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA
106	WSYPW	**Write.rsp (MemoryAddress, Data, WKC)** /Length == 0 => INC(LOGADD) wkc = wkc | (WKC * CMDPRMW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
107	WSYPW	**Write.rsp (MemoryAddress, Data, WKC)** /Length != 0 => Length -- INC(LOGADD) wkc = wkc | (WKC * CMDLRW) Data = RData Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EDTA

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
108	EWKL	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EWKL
109	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 => Overflow, Data = Data + wkc Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKH
110	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
111	EWKL	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
112	EWKH	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	ECMD
113	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && MF => Data = Data + Overflow Port = 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EWKL
114	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && !MF => Data = Data + Overflow Port 1 UPD_FCS(Data) Tx.req(Port,Byte,Data)	EFCS1
115	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
116	EWKH	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
117	EFCS1	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS1
118	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS1 Port 1 Tx.req(Port,Byte,Data)	EFCS2
119	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
120	EFCS1	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
121	EFCS2	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS2
122	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS2 Port 1 Tx.req(Port,Byte,Data)	EFCS3
123	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
124	EFCS2	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
125	EFCS3	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS3

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
126	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS3 Port 1 Tx.req(Port,Byte,Data)	EFCS4
127	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
128	EFCS3	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
129	EFCS4	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS4
130	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 => Data = FCS4 Port 1 Tx.req(Port,Byte,Data)	EFCS5
131	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
132	EFCS4	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == ERR => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH
133	EFCS5	**Rx.ind(Port,Byte,Data)** /Port != 0 => if Byte == 0 then RxTimeLatch[Port] = CT Port = (Port +1) MOD (LASTP +1) Tx.req(Port,Byte,Data)	EFCS5
134	EFCS5	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte == END => Port = 1 Byte = END Success = TRUE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

表 A.5 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
135	EFCS5	**Rx.ind(Port,Byte,Data)** /Port == 0 && Byte != END => Port = 1 Byte = ERR Success = FALSE Tx.req(Port,Byte,Data) Terminate.ind (success)	ETH

A.1.4 函数

表 A.6 是 DHSM 函数的汇总。

表 A.6 – DHSM 函数表

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

A.2 SYSM

A.2.1 原语定义

A.2.1.1 DHSM 和 SYSM 之间的原语交换

表 A.7 给出了由 SYSM 发到 DHSM 的原语

表 A.7 – 由 SYSM 发到 DHSM 的原语

原语名称	相关参数
Read response	Address, Data, WKC
Write response	Address, Data, WKC

表 A.8 给出了由 DHSM 发到 SYSM 的原语。

表 A.8 – 由 DHSM 发到 SYSM 的原语

原语名称	相关参数
Read indication	Address, Data, WKC
Write indication	Address, Data, WKC
Terminate indication	Success

A.2.1.2 DL 用户和 SYSM 之间的原语交换

表 A.9 给出了由 DL 用户发到 SYSM 的原语。

表 A.9 – 由 DL 用户发到 SYSM 的原语

原语名称	相关参数
DL-Read local request	Address, Length
DL-Write local request	Address, Length, Data

表 A.10 给出了由 SYSM 发到 DL 用户的原语。

表 A.10 – 由 SYSM 发到 DL 用户的原语

原语名称	相关参数
DL-Read local confirmation	L-Status, Data
DL-Write local confirmation	L-Status

注：本地事件不会模型化，因为他们不影响 SYSM。

A.2.1.3 DHSM 原语的参数

表 A.11 给出了 SYSM 和 DHSM 之间的原语所使用的所有参数。

表 A.11 – 用于 SYSM 和 DHSM 之间交换的原语所使用的参数

参数名称	描述
WKC	本地访问的工作计数器
Address	物理存储区的标识符
Data	接收/发送的八位位组值

A.2.2 状态机描述

每个同步管理器通道都有一个 SYSM。抽象模型为：如果有一个地址匹配，EtherCAT 读/写指示原语和读/写本地请求原语传递给第一个 SYSM 并被执行，否则被传递到下一个 SYSM。如果没有一个 SYSM 响应该服务原语，且存储器或寄存器是空闲的，该服务类型对这一寄存器是使能的，则物理存储器的处理程序将执行该服务请求。如果以太网帧被 DHSM 成功解析，寄存器的写访问将被执

行。对字或双字寄存器的第一个八位位组读访问以原子方法执行，即在后续的访问中，被读取的第一个八位位组的值将被冻结。

这里介绍 SYSM 的缓存类型和邮箱类型。本地请求按照这种方法模型化，读/写访问边界内或同步管理器区边界外都是完整的。

当关联的同步管理器通道寄存器区被写入时，产生 SM 事件。

本地变量

eact
被主站激活的缓存。

uact
被 DLS-user 激活的缓存。

Terminate
终止邮箱或缓存事务的指示符。

User
包含用户缓存。

Buffer
包含用于通信的缓存。

Next
包含下次使用的缓存。

Free
包含空闲缓存。

p1, p2, p3
三个缓存区的内存位置，第一个缓存是本地的，由 SM 指定，其他都按顺序排列。

act
包含邮箱的活动代码。

状态表术语

标准后缀：“.req”, “.cnf” 和 “.ind” 分别用来表示请求，证实和指示原语。

A.2.3 SYSM 表

表 A.12 给出了 SYSM 状态表。

表 A.12 – SYSM 状态表

#	当前状态	事件/状况⇒动作	下一状态

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

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

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

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

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

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
9	OFF	**Read.ind (Address, Data, WKC)** => pass next	OFF
10	OFF	**Terminate.ind(Success)** => pass next	OFF
11	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && !uact => (user. Address) = Data L-Status = OK uact = TRUE SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	BR-IDLE
12	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && uact => (user. Address) = Data L-Status = OK SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	BR-IDLE
13	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && !uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
14	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK uact = FALSE SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
15	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** /NA&&AE && !uact => L-Status = WRNGSEQ DL-Write Local.cnf (L-Status)	BR-IDLE
16	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&E && uact => (user. Address) = Data if next == NIL then next = user, user = free, free = NIL else user <=> next SM.bufferedState=next buffer number L-Status = OK uact = FALSE SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	BR-IDLE
17	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&NE && uact => (user. Address) = Data L-Status = OK DL-Write Local.cnf (L-Status)	BR-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
18	BR-IDLE	**DL-Write Local.req (Address, Length, Data)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BR-IDLE
19	BR-IDLE	**DL-Read Local.req (Address, Length)** => pass next	BR-IDLE
20	BR-IDLE	**Write.ind (Address, Data, WKC)** => pass next	BR-IDLE
21	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && !eact => if next != NIL then free = buffer, buffer = next, next = NIL Data = (buffer. Address) eact = TRUE WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
22	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && eact => Data = (buffer. Address) WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
23	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && !eact => if next != NIL then free = buffer, buffer = next, next = NIL Data = (buffer. Address) WKC = WKC +1 eact = TRUE Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
24	BR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && eact => /*Buffer exchange possible*/ Data = (buffer. Address) WKC = WKC +1 Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	BR-IDLE
25	BR-IDLE	**Read.ind (Address, Data, WKC)** /NA&&AE && !eact => Read.rsp (Address, Data, WKC)	BR-IDLE
26	BR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&E && eact => Data = (buffer. Address) WKC = WKC +1 Terminate = TRUE Read.rsp (Address, Data, WKC)	BR-IDLE
27	BR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&NE && eact => Data = (buffer. Address) WKC = WKC +1 Read.rsp (Address, Data, WKC)	BR-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
28	BR-IDLE	**Read.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BR-IDLE
29	BR-IDLE	**Terminate.ind(Success)** /Success && Terminate => eact = FALSE Terminate = FALSE SM.ReadEvent = 1 pass next	BR-IDLE
30	BR-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && eact) then eact = FALSE pass next	BR-IDLE
31	BR-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE eact = FALSE pass next	BR-IDLE
32	BW-IDLE	**DL-Write Local.req (Address, Length, Data)** => pass next	BW-IDLE
33	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && !uact => if next != NIL then free = user, user = next, next = NIL Data = (user. Address) L-Status = OK uact = TRUE SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
34	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && uact => Data = (user. Address) L-Status = OK SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
35	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && !uact => if next != NIL then free = user, user = next, next = NIL Data = (user. Address) L-Status = OK SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
36	BW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && uact => Data = (user. Address) L-Status = OK uact = FALSE SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
37	BW-IDLE	**DL-Read Local.req (Address, Length)** /NA&&AE && !uact => L-Status = WRNGSEQ DL-Read Local.cnf (L-Status, Data)	BW-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
38	BW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&E && uact => Data = (user. Address) L-Status = OK uact = FALSE SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	BW-IDLE
39	BW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&NE && uact => Data = (user. Address) L-Status = OK DL-Read Local.cnf (L-Status, Data)	BW-IDLE
40	BW-IDLE	**DL-Read Local.req (Address, Length)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BW-IDLE
41	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && !eact => (buffer. Address) = data eact = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
42	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && eact => (buffer. Address) = data WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
43	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && !eact => (buffer. Address) = data WKC = WKC +1 eact = TRUE Terminate = TRUE SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
44	BW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && eact => (buffer. Address) = data Terminate = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	BW-IDLE
45	BW-IDLE	**Write.ind (Address, Data, WKC)** /NA&&AE && !eact => Write.rsp (Address, Data, WKC)	BW-IDLE
46	BW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&E && eact => (buffer. Address) = data WKC = WKC +1 Terminate = TRUE Write.rsp (Address, Data, WKC)	BW-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
47	BW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&NE && eact => (buffer. Address) = data WKC = WKC +1 Write.rsp (Address, Data, WKC)	BW-IDLE
48	BW-IDLE	**Write.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	BW-IDLE
49	BW-IDLE	**Read.ind (Address, Data, WKC)** => pass next	BW-IDLE
50	BW-IDLE	**Terminate.ind(Success)** /Success && Terminate => eact = FALSE Terminate = FALSE if next == NIL then next = buffer, buffer = free, free = NIL else user <=> next SM.bufferedState=next buffer number SM.WriteEvent = 1 pass next	BW-IDLE
51	BW-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && eact) then eact = FALSE pass next	BW-IDLE
52	BW-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE eact = FALSE pass next	BW-IDLE
53	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&NE && act ==0 => (user. Address) = Data act = 1 L-Status = OK SM.ReadEvent = 0 DL-Write Local.cnf (L-Status)	MR-IDLE
54	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&AE && act != 0 => L-Status = NODATA DL-Write Local.cnf (L-Status)	MR-IDLE
55	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** /A&&E && act == 0 => (user. Address) = Data act = 2 SM.mailboxState=1 L-Status = OK SM.ReadEvent = 0 SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	MR-IDLE
56	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&AE && && act != 1 => L-Status = NODATA DL-Write Local.cnf (L-Status)	MR-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
57	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&E && act == 1 => (user. Address) = Data act = 2 SM.mailboxState=1 L-Status = OK SM.WriteEvent = 1 DL-Write Local.cnf (L-Status)	MR-IDLE
58	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / NA&&NE && act == 1 => (user. Address) = Data L-Status = OK DL-Write Local.cnf (L-Status)	MR-IDLE
59	MR-IDLE	**DL-Write Local.req (Address, Length, Data)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MR-IDLE
60	MR-IDLE	**DL-Read Local.req (Address, Length)** => pass next	MR-IDLE
61	MR-IDLE	**Write.ind (Address, Data, WKC)** => pass next	MR-IDLE
62	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&NE && act ==2 => Data = (buffer. Address) act = 3 WKC = WKC +1 SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	MR-IDLE
63	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&AE && act != 2 => Read.rsp (Address, Data, WKC)	MR-IDLE
64	MR-IDLE	**Read.ind (Address, Data, WKC)** /A&&E && act == 2 => Data = (buffer. Address) WKC = WKC +1 act = 4 Terminate = TRUE SM.WriteEvent = 0 Read.rsp (Address, Data, WKC)	MR-IDLE
65	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&AE && && act != 3 => Read.rsp (Address, Data, WKC)	MR-IDLE
66	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&E && act == 3 => Data = (buffer. Address) WKC = WKC +1 act = 4 Terminate = TRUE Read.rsp (Address, Data, WKC)	MR-IDLE
67	MR-IDLE	**Read.ind (Address, Data, WKC)** / NA&&NE && act == 3 => Data = (buffer. Address) WKC = WKC +1 Read.rsp (Address, Data, WKC)	MR-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
68	MR-IDLE	**Read.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MR-IDLE
69	MR-IDLE	**Terminate.ind(Success)** /Success && Terminate => Terminate = FALSE act = 0 SM.mailboxState=0 SM.ReadEvent = 1 pass next	MR-IDLE
70	MR-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && act == 3) then act = 2 pass next	MR-IDLE
71	MR-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE act = 2 pass next	MR-IDLE
72	MW-IDLE	**DL-Write Local.req (Address, Length, Data)** => pass next	MW-IDLE
73	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&NE && act ==3 => Data = (User. Address) L-Status = OK act = 4 SM.WriteEvent = 0 DL-Read Local.cnf (L-Status, Data)	MW-IDLE
74	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&AE && act != 3 => L-Status = NODATA DL-Read Local.cnf (L-Status, Data)	MW-IDLE
75	MW-IDLE	**DL-Read Local.req (Address, Length)** /A&&E && act == 3 => Data = (User. Address) L-Status = OK act = 0 SM.mailboxState=0 SM.WriteEvent = 0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	MW-IDLE
76	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&AE && && act != 4 => L-Status = NODATA DL-Read Local.cnf (L-Status, Data)	MW-IDLE
77	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&E && act == 4 => Data = (User. Address) L-Status = OK act = 0 SM.mailboxState=0 SM.ReadEvent = 1 DL-Read Local.cnf (L-Status, Data)	MW-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
78	MW-IDLE	**DL-Read Local.req (Address, Length)** / NA&&NE && act == 4 => Data = (User. Address) L-Status = OK DL-Read Local.cnf (L-Status, Data)	MW-IDLE
79	MW-IDLE	**DL-Read Local.req (Address, Length)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MW-IDLE
80	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&NE && act ==0 => (buffer. Address) = Data act = 1 WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	MW-IDLE
81	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&AE && act != 0 => Write.rsp (Address, Data, WKC)	MW-IDLE
82	MW-IDLE	**Write.ind (Address, Data, WKC)** /A&&E && act == 0 => (buffer. Address) = Data act = 2 Terminate = TRUE WKC = WKC +1 SM.ReadEvent = 0 Write.rsp (Address, Data, WKC)	MW-IDLE
83	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&AE && && act != 1 => Write.rsp (Address, Data, WKC)	MW-IDLE
84	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&E && act == 1 => (buffer. Address) = Data act = 2 Terminate = TRUE WKC = WKC +1 Write.rsp (Address, Data, WKC)	MW-IDLE
85	MW-IDLE	**Write.ind (Address, Data, WKC)** / NA&&NE && act == 1 => (buffer. Address) = Data WKC = WKC +1 Write.rsp (Address, Data, WKC)	MW-IDLE
86	MW-IDLE	**Write.ind (Address, Data, WKC)** / Address < SM.PhysicalStartAddress || Address >= (SM.PhysicalStartAddress+SM.Length) => pass next	MW-IDLE
87	MW-IDLE	**Read.ind (Address, Data, WKC)** => pass next	MW-IDLE

表 A.12 *(续)*

#	当前状态	事件/状况⇒动作	下一状态
88	MW-IDLE	**Terminate.ind(Success)** /Success && Terminate => Terminate = FALSE act = 3 SM.mailboxState=1 SM.WriteEvent = 1 pass next	MW-IDLE
89	MW-IDLE	**Terminate.ind(Success)** /!Terminate => if (!success && act == 1) then act = 0 pass next	MW-IDLE
90	MW-IDLE	**Terminate.ind(Success)** /(!Success && Terminate) => Terminate = FALSE act = 0 pass next	MW-IDLE

A.2.4 函数

表 A13 是 SYSM 函数的汇总。

表 A.13 – SYSM 函数表

函数名称	操作
A	Address == SM.PhysicalStartAddress
NA	Address > SM.PhysicalStartAddress
E	Address + Length == SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter
NE	Address + Length < SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter
AE	Address + Length =< SM.PhysicalStartAddress + SM.Length // Length = 1 if Length no service primitive parameter

A.3 RMSM

A.3.1 原语定义

A.3.1.1 SYSM 和 RMSM 之间的原语交换

表 A.14 给出了由 RMSM 发到 SYSM 的原语。

表 A.14 – 由 RMSM 发到 SYSM 的原语

原语名称	相关参数
DL-Write local request	Address, Length, Data

表 A.15 给出了由 SYSM 发到 RMSM 的原语。

表 A.15 – 由 SYSM 发到 RMSM 的原语

原语名称	相关参数
DL-Write local confirmation	L-Status

注意：本地事件不会模型化，因为他们不影响 RMSM。

A.3.1.2 SYSM 原语的参数

表 A.16 给出了 RMSM 和 SYSM 之间原语所使用的所有参数。

表 A.16 – RMSM 和 SYSM 之间原语所使用的所有参数

参数	描述
Address	物理存储区的标识符
Length	传输的八位位组的长度
Data	传输的八位位组的值

A.3.2 状态机描述

每个读邮箱同步管理器通道都有一个 RMSM。

RMSM 在用户请求时写读邮箱。只有一个用户请求(Read Upd)被接受。如果数据被主站读取，RMSM 将把数据保存在辅助缓存中。

切换时的改变将导致旧的邮箱的数据恢复（即使其间有一个新的邮箱更新）。

本地变量

Tggl
包含上次 SM 事件的切换标志。

Boxstate
邮箱的信号状态

BackupBox
邮箱 PDU 的备份存储单元。

ActualBox
实际邮箱 PDU 的虚拟存储。

SeqN
包含下一个服务的序列号。

状态表术语

标准后缀“.req”, “.cnf” 和 “.ind” 分别用来表示请求，确认和指示原语。

A.3.3 RMSM 表

表 A.17 给出了 RMSM 状态表。

表 A.17 – RMSM 状态表

#	当前状态	事件/状况⇒动作	下一状态
1	OFF	**Enable SM Event (Toggle)** => Tggl=Toggle Boxstate = 0 SeqN = 0 BackupBox = ERR PDU with no Error(0,0,0,0)	IDLE
2	OFF	**Disable SM Event** =>	OFF
3	IDLE	**Enable SM Event (Toggle)** =>	IDLE
4	IDLE	**Disable SM Event** => Terminate Segmented Services	OFF
5	IDLE	**Toggle SM Event (Toggle)** /Tggl != Toggle =>	IDLE
6	IDLE	**Toggle SM Event (Toggle)** /Tggl == Toggle => ActualBox = BackupBox Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read Boxstate = 1 Tggl = Toggle DL-Write Local.req(Address, Length, Data) write SM status(Toggle)	SEND

#	当前状态	事件/状况⇒动作	下一状态
7	IDLE	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** => ActualBox = Parameter from event service primitive Update (SeqN) Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read Boxstate = 0 DL-Write Local.req(Address, Length, Data)	SEND
8	IDLE	**DL-Write Local.cnf (L-Status)** => ignore	IDLE
9	SEND	**Enable SM Event (Toggle)** =>	IDLE
10	SEND	**Disable SM Event** => Terminate Segmented Services	OFF
11	SEND	**Toggle SM Event (Toggle)** /Tggl != Toggle =>	SEND
12	SEND	**Toggle SM Event (Toggle)**/Tggl == Toggle && Boxstate == 0=>ActualBox <=> BackupBox (Exchange) Address = SM1.PhysicalStartAddress Length = SM1.LengthData = encode Mailbox ReadBoxstate = 2DL-Write Local.req(Address, Length, Data)write SM status(Toggle)	SEND
13	SEND	**Toggle SM Event (Toggle)** /Tggl == Toggle && Boxstate == 1, 2 => write SM status(Toggle)	SEND
14	SEND	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** /Boxstate == 1 => BackupBox = Parameter from event service primitive Update (SeqN) Boxstate = 2	SEND
15	SEND	**DL-Mailbox Read Upd.req (Length, D_address, Channel, Priority, Type, Service Data Unit)** /Boxstate == 0, 2 => invalid user sequence	SEND
16	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 0 => DL_status = L-Status BackupBox =ActualBox DL-Mailbox Read Upd.cnf (DL_status)	IDLE
17	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 1 => DL_status = L-Status DL-Mailbox Read Upd.cnf (DL_status)	IDLE
18	SEND	**DL-Write Local.cnf (L-Status)** /Boxstate == 2 => ActualBox = BackupBox Address = SM1.PhysicalStartAddress Length = SM1.Length Data = encode Mailbox Read DL_status = L-Status Boxstate = 0 DL-Mailbox Read Upd.cnf (DL_status) DL-Write Local.req(Address, Length, Data)	SEND

A.3.4 函数

表 A.18 是 RMSM 函数的汇总。

表 A.18 – RMSM 功能表

函数名称	操作
Update (SeqN)	if (SeqN < 7) then SeqN = SeqN + 1 else SeqN = 1

参考文献

IEC 60559, *Binary floating-point arithmetic for microprocessor systems*

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

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

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

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

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

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

ISO/IEC TR 8802-1,*Information technology — Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 1: Overview of Local Area Network Standards*

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

ISO/IEC 15802-1, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Common specifications – Part 1: Medium Access Control (MAC) service definition*

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

ITU-T V.41, *Code-independent error-control system*

ANSI X3.66 (R1990), *Advanced data communication control procedures (ADCCP)*

IEEE 802.1D, *IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Bridges;* available at <http://www.ieee.org>

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

RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets : MIB-II*
RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types*

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

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

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

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

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

ETG.1100 EtherCAT Specification Communication profiles

附录

EtherCAT Technology Group (ETG)

EtherCAT Technology Group

Ostendstrasse 196
90482 Nuremberg, Germany

phone: +49 (9 11) 5 40 56 20
fax: +49 (9 11) 5 40 56 29
e-mail: info@ethercat.org
web: www.ethercat.org