[概述 2](#_Toc508203548)

[应用层功能 2](#_Toc508203549)

[设备模型 4](#_Toc508203550)

[CAN-ID 5](#_Toc508203551)

[对象字典 6](#_Toc508203552)

[描述 6](#_Toc508203553)

[对象字典的分配 7](#_Toc508203554)

[数据类型 8](#_Toc508203555)

[通讯参数对象 10](#_Toc508203556)

[对象字典的实现 11](#_Toc508203557)

[网络管理对象NMT 12](#_Toc508203558)

[设备状态 12](#_Toc508203559)

[状态转换 13](#_Toc508203560)

[NMT通讯 14](#_Toc508203561)

[网络错误控制对象NEC 15](#_Toc508203562)

[NEC通讯 15](#_Toc508203563)

[NEC参数 16](#_Toc508203564)

[应急指示对象EMCY 17](#_Toc508203565)

[EMCY通讯 17](#_Toc508203566)

[EMCY参数 18](#_Toc508203567)

[服务数据对象SDO 20](#_Toc508203568)

[SDO报文 20](#_Toc508203569)

[SDO参数 21](#_Toc508203570)

[SDO快速传输数据场 23](#_Toc508203571)

[SDO分段传输数据场 24](#_Toc508203572)

[SDO块传输数据场 25](#_Toc508203573)

[SDO中断传输数据场 27](#_Toc508203574)

[同步对象SYNC 28](#_Toc508203575)

[SYNC通讯 28](#_Toc508203576)

[SYNC参数 29](#_Toc508203577)

[过程数据对象PDO 30](#_Toc508203578)

[PDO报文 30](#_Toc508203579)

[PDO通讯 31](#_Toc508203580)

[PDO通讯参数 32](#_Toc508203581)

[PDO映射参数 34](#_Toc508203582)

[时间戳TIME STAMP 35](#_Toc508203583)

[TIME报文 35](#_Toc508203584)

[TIME通讯 36](#_Toc508203585)

[TIME参数设置 37](#_Toc508203586)

# 概述

## 应用层功能

CAN总线协议只定义物理层和数据链路层，没有规定应用层，本身并不完整

CAN-OPEN协议就是用来定义CAN总线协议的 应用层

拿写信来说，

固定规格的纸张、打印机内的墨粉相当于物理层；

拉丁字母就是逻辑层（数据链路层）；

打印机以及其中放置的纸张就是CAN收发器和控制器；

语法好比应用层；

CANopen应用层的作用就是定义句子的成分（主语、谓语、宾语等）。

假如要写信给国外的一家餐厅订餐，我们不需要懂这个国家的语言，只要使用有常用短语的旅行用语手册即可。

把预设的短语稍加改动，把咖啡换成茶。

这个例子中，旅行用语手册相当于CANopen中的设备子规范、接口规范、应用规范。这些规范实现定义好了不同的场合下需要用到的语句。这样就可以减少写信花费的时间，相当于减少设计通信系统伊奥花费的时间。

这些规范可以由设备的制造商定义，也可以由CIA定义。

应用层（Application layer）：为网络中每一个有效设备都能够提供一组有用的服务与协议。

CANopen应用层详细定义了通信服务和其他相关的通信协议。

通信对象、过程参数、配置参数 保存在设备的对象字典中。

基本服务：

|  |  |
| --- | --- |
| 请求 | 应用程序请求CANopen软件的一种通信服务 |
| 指示 | CANopen软件向应用程序报告某一事件或执行的任务 |
| 响应 | 应用程序对CANopen软件报告的事件或任务作出应答 |
| 确认 | CANopenn软件向应用程序确认CANopen软件已经执行了任务 |

CANopen应用层的服务类型分为2种：

|  |  |
| --- | --- |
| 仅在设备中执行的服务 | 如：局部服务和提供启动的服务 |
| 多个设备通过网络进行通信的服务 | 如：确认的服务（握手）和未确认的服务（广播） |

通讯描述（Communication profile）：提供配置设备、通讯数据的含义，定义数据通讯方式。

设备描述（Device proflile）：为设备（类）增加符合规范的行为。

在CAN协议中，ID起着重要的作用，它决定着数据帧发送的优先级，也决定着其它节点是否会接收这个数据帧。

CAN协议不对挂载在它之上的节点分配优先级和地址，对总线的占有权是由信息的重要性决定的，即对于重要的信息，我们会给它打包上一个优先级高的ID，使它能够及时地发送出去。

这样的优先级分配原则，使得CAN的扩展性大大加强，在总线上增加或减少节点并不影响其它设备。

在CAN总线上数据是以广播的形式发送的，所有连接在CAN总线的节点都会收到所有其它节点发出的有效数据，因而CAN控

在之前CAN总线协议的基础上，设置两个地方：

|  |  |
| --- | --- |
| 仲裁段的 11/29位标识符 | 1.确定消息是来自哪个设备  2.确定消息的优先级  3.预设功能  4.做一些相关定义，防止出现功能不同惨CAN报文相同的情况，如定义标识符为0x181表示模拟量，0x182表示开关量，那么即使数据段相同，两者要表示的数据也不同 |
| 数据段的8字节 | 1.数据内容的传输格式  2.数据的读取规则 |
| 总线上的设备需要具有单个设备的控制机制和错误处理能力 |  |

## 设备模型

|  |  |  |
| --- | --- | --- |
| 通信单元 | CAN收发器 |  |
| CAN控制器 |  |
| CANopen协议栈 | 包含通信对象和状态机 |
| 对象字典 | 是设备的所有参数的列表；  应用单元和通信单元都可以访问对象字典，它们执行操作时所需要的参数都来源于 对象字典中的条目的值； | |
| 应用单元 | 当设备要实现某个功能时，就要使用应用单元，应用单元描述和定义了设备的功能；  应用单元在实现某个功能时，先访问对象字典，获取参数；然后会和通信单元进行交互：  例如在I/O设备中，访问设备的数字或模拟 输入/输出 接口；  在驱动系统中，可以实现轨迹发生器或速度控制模块的控制。 | |

通信单元提供数据传输的所有机制和通信对象，符合CANopen规范的数据可以利用这些机制通过CAN接口进行传输。

通信时，应用层向它的客户提供服务，这种能提供服务的设备称之为服务器

这些服务是服务器所允许的服务，也可以是一项功能或特定的服务。

客户端可以不用关心服务执行的详细过程，在未确认的通信服务中，服务器的应用层不会返回服务执行成功或失败的信息。而该信息必须由应用程序自己来发送，因此，在这种情况下客户端就不会知道应用程序什么时候执行了服务。

此外，在服务中还定义了服务原语，所谓的服务是指应用程序放给应用层的请求。服务器的应用层收到来自客户端的请求报文后，其就会向应用层程序发出服务请求命令，在执行完该请求服务后，应用程序就会想其应用层响应结果。然后再通过确认将产生的CAN报文发送给客户端。

在确认的通信服务中，无论在什么情况下，应答信号都是由服务器的应用层来发送的，收到应答信号的时间与执行的服务类型及其参数的关系很大。例如，写操作的事件比读取单个值的时间长。

## CAN-ID

通信对象，是CANopen协议定义的一种**标准数据帧**报文，报文格式可以查看 《CAN总线协议》

扩展报文格式仅用于系统研发者所配置的报文。

|  |  |  |
| --- | --- | --- |
| 报文的优先级通由4 位为功能码来确定。值越小，优先级越高。  CANopen协议为各类通信对象都预定义了功能码，可以直接使用  功能码将数据帧分为7个优先级的报文，按照**优先级从高至低**依次为： | | |
| NMT | 0000 | 0x0 |
| EMERGENCY | 0001 | 0x80 |
| SYNC | 0001 | 0x80 |
| TIME | 0010 | 0x100 |
| TPDO1 | 0011 | 0x180 |
| TPDO2 | 0101 | 0x280 |
| TPDO3 | 0111 | 0x380 |
| TPDO4 | 1001 | 0x480 |
| RPDO1 | 0100 | 0x200 |
| RPDO2 | 0110 | 0x300 |
| RPDO3 | 1000 | 0x400 |
| RPDO4 | 1010 | 0x500 |
| SDO（服务端🡪客户端） | 1011 | 0x580 |
| SDO（客户端🡪服务端） | 1100 | 0x600 |
| NEC | 1110 | 0x700 |

|  |  |
| --- | --- |
| 节点号  （低7位） | 区分优先级：  区分报文优先级时，先看功能代码，数值低的优先级高；  如果功能代码相同，再看节点ID，数值低的优先级高。  区分设备：  在CANopen网络中，每个设备有一个固定的唯一的节点ID。  设备的节点号由系统集成商定义，**一般在设备上有设置节点ID的机械开关；**  范围是1~127（0不允许被使用）；CANopen 网络最多可支持127个节点共存，即支持127个设备 |
| 预留的  CAN-ID | 如果要给报文配置CAN-ID，那么注意不要使用预留给某些特定功能的CAN标识符：  001h~~07Fh：预留给CANopenSafety  101h~~180h：预留给动态主机  6E0h~~6FFh：预留给CANopen管理器  780h~~7FFh：预留给LSS |

# 对象字典

## 描述

CANopen 的每一个站都配有一个对象字典（Object Dictionary，OD）。

对象字典是所有数据的集合，这些数据涉及到设备的应用程序、通信、状态机；包含描述这个设备和它的网络行为的所有参数。

CANopen网络中主机向某个设备的对象字典写入参数或者读取设备对象字典的某个参数，由此实现控制一个设备或获取一个设备的状态。

对于子索引00h，它是对象的第一个子索引，对应的子对象，总是一个UNSIGNED8的变量，其值总是表示这个对象含有的成员的数目，所以如果某个对象是一个变量，那么其Sub-index=0000 0001

|  |  |  |
| --- | --- | --- |
| Index | 对象的索引号—16位  对象字典中的每一个对象对应一个唯一的**16位**的索引，一个CANopen设备的对象字典中最多可以有65536个对象  用户通过该索引访问到对应的对象。 | |
| Sub-index | 对象的某个条目的子索引号—8位  对象至少有1个成员  不可能只访问对象而不访问成员：因为实际在使用索引访问某个对象时，会使用一个24位的编码，其中的16位用于Index，8位用于Sub-index，那么Sub-index=0000 0000 就表示访问这个索引的第一个子索引。  如果某个对象是一个数组或一个结构类型的变量，那么这个对象还有一个**8位**的子索引，用户通过子索引来访问这个对象的各个元素或成员。有时候就直接称某个元素或成员为子对象  对象的第一个条目的子索引是Sub-index1，第二个是Sub-index2  一个对象至少有一个子索引，一个对象的子索引最多为254个(去掉00h和FFh)  对于子索引FFh，一般不存在这个子索引，如果设备支持FFh，那么其对应的子对象往往是一个UNSIGNED32的变量，其值可以表示整个对象字典的结构？ | |
| Object code | DOMAIN | 包含大量数据的对象，如一段可执行程序代码 |
| DEFTYPE | 简单数据类型定义 |
| DEFSTRUCT | 复杂数据类型定义，如：record、array等 |
| VAR | 该对象仅有一个成员 |
| ARRAY | 该对象有多个成员，而且它们的数据类型相同 |
| RECORD | 该对象有多个成员，它们的数据类型可以不同 |
| Data type | 表示对象中条目变量的数据类型 | |
| Category | Optional：由生产商决定是否把该对象添加到设备的对象字典中；  Mandatory：设备的对象字典中必须要有这个对象；  Conditional：可能是Optional，也可能是Mandatory，取决于设备的其他属性。 | |
| Access | ro：只读，表示这个对象只能被读  rw：读/写，表示这个对象可读可写  wo：只写，表示这个对象只能被写  const ：常数，表示这个对象只能被引用，既不能读也不能写 | |
| PDO mapping | PDO映射  Optional： 这个对象可以被映射到PDO  Default： 这个对象已经被默认映射到PDO了的  TPDO： 这个对象可以被映射到TPDO但不能映射到RPDO  RPDO： 这个对象可以被映射到RPDO但不能映射到TPDO  No： 这个对象不能被映射 | |
| Value range | 该子索引号条目对应的数据的取值范围 | |
| Default value | 设备初始化后该子索引号对应的数据的值  No：没有默认值  Profile-specific： 在协议中定义  Manufacturer-specific： 由该设备的生产商定义 | |

## 对象字典的分配

在下面的索引划分中我们可以看到对象也有很多种类

由于是16bits的位索引，所以最多有65536个对象，协议对这65536个索引进行了划分，每块索引范围用于特定的对象类型。

|  |  |
| --- | --- |
| 索引(16进制) | 对象 |
| 0000 | 保留 |
| 0001~~001F | 静态数据类型 |
| 0020~~003F | 复杂数据类型 |
| 0040~~005F | 生产商针对它们的设备所设置的特定的数据类型。 |
| 0060~~025F | 针对特定的设备添加的基本数据类型或复杂数据类型 |
| 0260~~03FF | 保留 |
| 0400~~0FFF | 保留 |
|  |  |
| 1000~~1FFF | 通讯参数 |
|  |  |
| 2000~~5FFF | 生产商针对它们的设备所设置的特定对象 |
|  |  |
| 6000~~67FF | 驱动设备参数 |
| 6800~~6FFF | 标准设备参数和功能2 |
| 7000~~77FF | 标准设备参数和功能3 |
| 7800~~7FFF | 标准设备参数和功能4 |
| 8000~~87FF | 标准设备参数和功能5 |
| 8800~~8FFF | 标准设备参数和功能6 |
| 9000~~97FF | 标准设备参数和功能7 |
| 9800~~9FFF | 标准设备参数和功能8 |
|  |  |
| A000~~AFFF | 网络变量，包括输入变量和输出变量  属于可编程的设备 |
| B000~~BFFF | 系统变量，包括输入变量和输出变量 |

## 数据类型

对象字典中的0001h~~025Fh 索引范围的对象，都是数据类型。

这些对象不能读也不能写，只能被引用，作为参考来使用。

对象字典中的对象都是数据，这些数据的数据类型就是参考0001h~~025Fh 索引范围的对象所表示的数据类型的。

|  |  |
| --- | --- |
| 静态数据类型  如BOOLEAN、INTEGER、UNSIGNED、floating point、string 等  这些对象是const：只能被引用，不能读也不能写  对这些对象执行读操作，那么可以获得对象的长度（如UNSIGNED），举个例子：  如果在接收RPDO中配置这些对象（虚拟映射），那么相应的位和字节就会被隐藏或忽略。 | |
| NIL |  |
| Boolean |  |
| Void |  |
| Unsigned Integer |  |
| Signed Integer |  |
| Floating-Point Numbers |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| 预定义的复杂数据类型  由多个基本数据类型组成的数据结构类型（struct）、数组 等  这些对象是const：只能被引用，不能读也不能写  作用是为所有的设备描述一种预定义结构 | | |
| 索引(hex) | 名称 | 说明 |
| 0020 | PDO\_COMMUNICATION\_PARAMETER |  |
| 0021 | PDO\_MAPPING\_PARAMETER |  |
| 0022 | SDO\_PARAMETER |  |
| 0023 | IDENTITY |  |

|  |  |  |
| --- | --- | --- |
| 制造商定义的复杂数据类型  由多个基本数据类型组成的数据结构类型（struct）、数组 等  这些对象是const：只能被引用，不能读也不能写  作用是为所有的设备描述一种预定义结构 | | |
| 索引(hex) | 名称 | 说明 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 子索引结构  这些对象拥有相同的子索引结构： | | |
| 子索引(hex) |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| 扩展数据类型 | |
| Octet String |  |
| Visible String |  |
| Unicode String |  |
| Time of Day |  |
| Time Difference |  |
| Domain |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| Index(hex) | 对象类型 | 描述 |
| 0001 | DEFTYPE | BOOLEAN |
| 0002 | DEFTYPE | INTEGER8 |
| 0003 | DEFTYPE | INTEGER16 |
| 0004 | DEFTYPE | INTEGER32 |
| 0005 | DEFTYPE | UNSIGNED8 |
| 0006 | DEFTYPE | UNSIGNED16 |
| ………… | | |
| 000F | DEFTYPE | Domain |
| 0010 | DEFTYPE | INTEGER24 |
| 0011 | DEFTYPE | REAL64 |
| ………… | | |
| 0020 | DEFSTRUCT | 作为PDO传输参数的数据类型 |
| 0021 | DEFSTRUCT | PDO映射 |
| 0022 | DEFSTRUCT | SDO传输参数 |
| 0023 | DEFSTRUCT | ID |
| ………… | | |
| 0040~~005F | DEFSTRUCT | 生产商配置的复杂数据类型 |
| 0060~~007F | DEFTYPE | 表示标准设备参数和功能的数据1 的类型 |
| 0080~~009F | DEFSTRUCT | 复杂设备参数和功能1 |
| 00A0~~00BF | DEFTYPE | 标准设备参数和功能2 |
| 00C0~~00DF | DEFSTRUCT | 复杂设备参数和功能2 |
| 00E0~~00FF | DEFTYPE | 标准设备参数和功能3 |
| ………… | | |

我们已经知道DEFSTRUCT这种对象类型表明这个对象是有子索引的，以Index=0020为例：

## 通讯参数对象

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 | |
| Shutdown option code | VAR | 605B | INTEGER16 | RW | O | 驱动器发生状态转换：Operation Enable→Ready To Switch On.  实际执行的操作 | |
| -32768 ~~–1 | 设备制造商定义 |
| 0 | Disable drive function |
| 1 | Slow down with slow down ramp; disable of the drive function |
| 2~~32767 | 保留 |

## 对象字典的实现

为了能快速访问索引，将索引放在一个很大的表格中，然后只要通过索引就能访问所需的对象。

但是我们不能把所有的索引都放进去，因为2^16 个对象需要非常大的存储空间。

而事实对某一个设备而言，也并不是所有条目都会使用到，所以，只要把要用到的条目写到一个表格中。我们称这个表格为**对象列表**

但是，这样做又有一个问题：就是存在空隙，因此不能根据原来的索引来访问对象，而需要通过一种查找算法来找到合适的条目。

CANopen软件通过固定的索引/子索引 来访问对象列表中的条目。对象列表提供一个指向存储器中的某个变量的指针。应用程序可以直接通过变量名称访问所需的条目。对象列表就构成了 索引/子索引 与对应变量名称之间的接口。

对象列表分为几个不同的块，每块均由若干个具有连续索引范围的字段组成。

# 网络管理对象NMT

## 设备状态

|  |  |  |
| --- | --- | --- |
| 节点状态 | 支持的通讯 | |
| 初始化状态  Initialisation | 从机在初始化状态下**不接收任何网络报文，也不发送任何报文**  从机在初始化状态下，自动设置自身参数和CANopen 对象字典；  初始化完成后，自动进入预操作状态。  为了区分Reset\_Node和Reset\_Communication 这两个NMT指令，又把**初始化状态细分为3个状态**：  硬件开启（上电）或复位🡪**正在初始化🡪复位应用🡪复位通信**🡪预操作状态 | |
| 复位应用 | 设备自动设置自身参数和对象字典中的参数恢复到默认值：  有关制造商相关的参数：（2000h~~5FFF）、设备子协议的参数（6000h~~9FFFh、A000h~~AFFFh、B000h~~BFFFh）  恢复默认设置以后自动触发 复位应用到复位通信的转换，从而进入复位通信状态。 |
| 复位通信 | 只有通信子协议的参数（1000h~~1FFFh）会恢复到默认值。  有了这两种复位方法，NMT主机就可以用不同的方式复位CANopen设备：  要么复位整个应用程序，要么仅复位CANopen参数。 |
| 预操作状态  Pre-operational | NMT | 主机发送NMT指令来改变从机的工作状态 |
| boot\_up | 从机一旦从初始化状态进入预操作状态，就会自动发布启动消息报文（boot up），将自己的这一状态改变事件（即设备从初始化状态转变到预操作状态这一事件）通知其他设备；然后等待命令。 |
| NEC | 从机设备发送NEC报文，用于设备监控 |
| SYNC | 从机设备作为同步报文生产者，在发送启动报文消息之后马上开始循环发送同步报文，从而使其他设备同步。 |
| SDO | 主机通过SDO通讯读写从机对象字典的对象，  写对象可以配置从机的参数，读对象可以获取从机状态 |
| 运行状态  Operational | NMT | 主机发送NMT指令来改变从机的工作状态 |
| NEC | 从机设备发送NEC报文，用于设备监控 |
| SYNC | 从机设备作为同步报文生产者，在发送启动报文消息之后马上开始循环发送同步报文，从而使其他设备同步。 |
| SDO | 主机通过SDO通讯读写从机对象字典的对象，以此配置设备参数 |
| PDO |  |
| 停止状态  Stopped | NMT | 主机发送NMT指令来改变从机的工作状态 |
| NEC | 从机设备发送NEC报文，用于设备监控 |

## 状态转换

|  |  |  |
| --- | --- | --- |
| 状态机的状态的转变通常由：  1.从机设备的内部事件触发（如设备上电、复位、内部错误触发……）  2.主机的NMT指令。 | | |
| 起始状态 | 目标状态 | 触发源 |
| / | 初始化状态 | 1.硬件开启（上电）  2.复位 |
| 初始化状态 | 预操作状态 | 从设备初始化完成后，自动进入预操作状态。 |
| 预操作状态 | 运行状态 | 1.主机向该设备发送NMT报文，CS=1🡪Service start remote node；  2.从预操作状态转变到运行状态通常由NMT主机发起。但有些自行启动CANopen设备会自动进入运行状态 |
| 停止状态 | 主机向该设备发送NMT报文，CS=2🡪Service stop remote node |
| 初始化状态的复位应用状态 | 主机向该设备发送NMT报文，CS=81🡪Service reset node |
| 初始化状态的复位通信状态 | 主机向该设备发送NMT报文，CS=82🡪Service reset communication |
| 运行状态 | 预操作状态 | 主机向该设备发送NMT报文，CS=80🡪Service enter pre-operational |
| 停止状态 | 主机向该设备发送NMT报文，CS=2🡪Service stop remote node |
| 初始化状态的复位应用状态 | 主机向该设备发送NMT报文，CS=81🡪Service reset node |
| 初始化状态的复位通信状态 | 主机向该设备发送NMT报文，CS=82🡪Service reset communication |
| 停止状态 |  |  |

## NMT通讯

|  |  |
| --- | --- |
| 主机-从机模型 | 不是所有设备都可以做NMT主机，但是所有CANopen设备都可以做从机  只有在接通网络之后才能决定哪台设备作为NMT主机，裁决的方法见配置NMT主机详见CIA302  进行NMT通讯时，整个CAN网络中，只有一个主机，其他设备都是从机。  NMT报文**仅由主机发送，从机接收**；  是一对一或一对多通讯，一次传输的报文的数目等于指定接收消息的从机的数目 |

# 网络错误控制对象NEC

## NEC通讯

错误对象控制提供 **心跳报文**和 **节点保护**  2种监控服务，只能采**其中一种**来进行设备监控。

建议使用心跳报文。这种方法很灵活而且不需要用到远程帧。有些设备不支持远程帧。

|  |  |
| --- | --- |
| 通讯模型 | 心跳报文使用**生产者-消费者模型**  发送心跳报文的设备作为生产者，接收心跳报文的设备作为消费者。  进行心跳报文通讯时，只有一个生产者，可以有多个消费者；  心跳报文仅由生产者发送，消费者接收；  是一对一或一对多通讯，一次传输的报文数目等于接收者的数目；  节点保护使用**主机-从机模型**  不是所有设备都可以做主机，但是所有CANopen设备都可以做从机  只有在接通网络之后才能决定哪台设备作为主机，裁决的方法见配置主机详见CIA302  进行NEC通讯时，整个CAN网络中，只有一个主机，其他设备都是从机。  节点保护报文**仅由从机发送，主机接收**；  NEC通讯是是一对一通讯，一次传输1个报文 |

|  |  |
| --- | --- |
| 节点保护通讯 | 配置主机对象字典的Guard time 和Life time factor  **当且仅当Guard time和Life time factor的值都不为0时，主机采用节点保护机制**  主机自动周期性地发送节点保护远程帧（无数据），指定发送节点保护报文的从机设备的节点ID；  从机每次收到该远程帧就会发送节点保护报文给主机；  如果Guard time和Life time factor其中一个设置为0或两个都设置为0；那么就意味着不设置节点保护机制。  如果同时设置了心跳报文和节点保护报文，那么就使用心跳保护报文  注意节点保护功能基于CAN远程帧，不是所有的CAN设备都支持远程帧。 |
| 心跳报文通讯 | 一般由某个从机作为生产者，周期性的发送心跳报文；  一般从机会将该报文发送给主机，但是也可以同时发送给其他从机用于设备之间相互监视；  通过心跳报文可以让某个设备当前所处的状态让其他设备知道  Before the reception of the first heartbeat the status of the heartbeat producer is unknown.  If the heartbeat time is 0 or the node-ID is 0 or greater than 127 the corresponding object entry  shall be not used. The heartbeat time shall be given in multiples of 1ms.  An attempt to configure several heartbeat times unequal 0 for the same node-ID the CANopen  device shall be responded with the SDO abort transfer service (abort code: 0604 0043h). |

## NEC参数

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 | |
| Consumer heartbeat time | ARRAY | 1016-00 | \ | const | O | 子索引数目 | |
| 1016-01 | UNSIGNED32 | RW | M | bit | 描述 |
| 31：24 | 保留=0 |
| 23：16 | Node-ID  配置心跳报文消费者指定接收哪个设备的心跳报文  填入被监视设备（即心跳报文生产者）的Node-ID  范围：1~127 |
| 15：0 | 配置心跳报文消费者的心跳周期；  如果超过心跳周期消费者还是没有收到指定设备的心跳报文，那么消费者就知道这个生产者出问题了。  **消费者设备的心跳周期要大于生产者的心跳间隔；**  单位：ms |
| 1016-  02~07 | UNSIGNED32 | RW | O | 同上  一台设备可以接收来自多台设备的心跳报文 | |
|  |  |  |  |  |  |  | |
| Producer heartbeat time | VAR | 1017 | UNSIGNED16 | RW | Conditional  M if guarding not supported | 配置心跳报文生产者心跳间隔  配置心跳报文生产者每个多长时间发送一条心跳报文；  单位：ms  范围：0x00 00~~0xFF FF  0x00 00：不产生心跳报文 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 |
| Guard time | VAR | 100C | UNSIGNED16 | RW | Conditional;  M if node guarding is supported | 用于配置NEC主机相邻两次发送节点保护远程帧的时间间隔；  单位：ms  范围：0x00 00 ~~ 0xFF FF  00 00：取消节点保护 |
| Life  time factor | VAR | 100D | UNSIGNED8 | RW | Conditional;  M if node guarding is supported | 用于配置NEC主机查询设备的最迟时间；  寿命因子×保护时间=主机查询设备的最迟时间  从主机发送一个远程幁开始算起，如果经过了这个最迟时间主机还没有收到指定的从机的节点保护报文，那么主机就认为这个从机故障了。  单位：  范围：0x00 ~~ 0xFF  0x00：取消节点保护 |

# 应急指示对象EMCY

## EMCY通讯

|  |  |
| --- | --- |
| 通讯模型 | EMCY使用**生产者-消费者模型**  发送EMCY消息的设备作为生产者，接收EMCY消息的设备作为消费者。  进行EMCY通讯时，只有一个生产者，可以有多个消费者；  **EMCY报文仅由生产者发送，消费者接收；**  是一对零或一对一或一对多通讯，一次传输的报文数目等于接收者的数目 |

设备出现错误时，就会发送一个紧急报文，来告知网络中的其他设备自己所处的错误状态；

任何 支持EMCY并且使能了EMCY接收功能 的设备会接收并处理紧或报文

一个错误对应一次EMCY传输

有时某个设备会同时发送好几条紧急报文，这样优先级较低的紧急报文就会延迟发送。

为了防止这种情况发生，用户可以在对象字典中对紧急报文的标志符进行适当配置。

但是要注意：对于需要接受其他紧急报文的设备需要给予通知，否则可能丢失错误信息。

如果经过一段时间后错误消失，设备会发送一个紧急报文来通知其他设备错误解除这一事件。

当前的错误状态保存在对象字典的错误寄存器（索引1001h）中；

最近出现的错误保存在对象字典的预定义的错误场（索引1003h）中；

用户可以通过SDO读取这些信息。

|  |  |  |
| --- | --- | --- |
| 设备错误 | 通信错误 | 这类错误通常在利用CAN网络进行通信的时候出现，并由CAN控制器发送错误信号。如果通信错误频繁出现，则表示现有系统的物理链路存在问题。  CAN系统除草会改变通过CAN进行通信的应用程序的应答行为。通信软件中出现的错误也属于通信错误。  此外，用户要特别注意“心跳”或“寿命”保护失效的问题（见“服务数据对象”小节）。 |
| 应用错误 | 通常是指应用程序发现的硬件问题如：短路、超过温度上限 等  也有可能是某种设备特定的错误 |

## EMCY参数

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 | | |
| Error register | VAR | 1001 | UNSIGNED8 | RO | M | 当设备发生紧急错误时，该设备的对象字典Error register  相应的错误类型位置1  其中不管发生什么错误，Generic error总会置1  同时该设备发送紧急报文(如果支持EMCY)  该紧急报文中就包含Error register的数据 | | |
| bit | M/O | 错误类型 |
| 7 | O | 生产商定义 |
| 6 | 保留 |
| 5 | Device profile specific |
| 4 | Communication error (overrun, error state) |
| 3 | Temperature |
| 2 | Voltage |
| 1 | Current |
| 0 | M | Generic error |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pre-defined error field | ARRAY | 1003--00 | / | RW | O | 读操作：可获取本设备当前存在的错误的个数  写操作：只能写0，写0可以清空所有错误 | |
| 1003--01 | UNSIGNED32 | RO | 存储最新的错误的信息  每当有新的错误发生，就会储存在Sub-Index 01  older errors依次下移一个 | |
| bit | 描述 |
| 31:16 | additional information  生产商定义 |
| 15:0 | error code  本错误的错误代码 |
| 1003--  02~~FE | UNSIGNED32 | RO |  | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| COB-ID EMCY | VAR | 1014 | UNSIGNED32 | RW | Conditional  (M if  Emergency  is supported) | bit | 描述 |
| 31 | valid=0：本设备能发送紧急报文  valid=1：本设备不发送紧急报文 |
| 30 | 保留=0 |
| 29 | frame=0：标准帧；  frame=1：扩展帧；  CANopen设备都是使用标准帧  所以**frame总是为0** |
| 28:11 | 扩展CAN-ID；  CANopen设备都是使用标准帧  所以扩展CAN-ID**总是全为0** |
| 10:0 | 基础CAN-ID；  11位中的高4位是功能码  低7位是Node-ID；  功能码是默认为0001  用户可以在设备的预操作状态下修改；  但一般就使用默认的。  **valid=0时，不能做修改** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 |
| Inhibit time EMCY | VAR | 1015 | UNSIGNED16 | RW | O | 禁止时间  **单位：100us**  **0：没有时间限制**  **1~~0xFFFF FFFF：禁止时间的时间值** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 | |
| Emergency consumer | ARRAY | 1028--00 | UNSIGNED32 | const | M | 条目数  范围：1~127 | |
| 1028--01 | UNSIGNED32 | RW | M | bit | 描述 |
| 31 | valid=0：本设备能接收紧急报文  valid=1：本设备不接收紧急报文 |
| 30 | 保留=0 |
| 29 | frame=0：标准帧；  frame=1：扩展帧；  CANopen设备都是使用标准帧  所以**frame总是为0** |
| 28:11 | 扩展CAN-ID；  CANopen设备都是使用标准帧  所以扩展CAN-ID**总是全为0** |
| 10:0 | 基础CAN-ID；  11位中的高4位是功能码  低7位是Node-ID；  功能码是默认为0001  用户可以在设备的预操作状态下修改；  但一般就使用默认的。  **valid=0时，不能做修改** |
| 1028--02  ~~127 | UNSIGNED32 | O | 同上 | |

# 服务数据对象SDO

## SDO报文

|  |  |  |
| --- | --- | --- |
| 报文类型 | 快速传输下载报文 |  |
| 快速传输回复报文 |  |
| 分段传输下载报文 |  |
| 分段传输回复报文 |  |
| 块传输下载报文 |  |
| 块传输回复报文 |  |
| 传输次序 | 发送SDO报文时先发送高位，再发送低位； | |
| 报文结构 | SDO报文属于CAN**标准数据帧报文**；其数据结构参见《CAN总线协议》  SDO报文的**CAN-ID**、**Data Field、DLC**见下表： | |

|  |  |
| --- | --- |
| CAN-ID | 默认SDO高4位功能码：  SDO（服务端🡪客户端）1011（580）  SDO（客户端🡪服务端）1100（600） |
| 默认SDO低7位Node-ID：  SDO（服务端🡪客户端）Node-ID=发送该报文的服务端设备的Node-ID；  SDO（客户端🡪服务端）Node-ID=该报文发送的目标服务端设备的Node-ID； |
| DLC | DLC=8 |
| 数据场 | SDO报文的数据场的格式根据不同的传输方式有所不同  数据位字节**由低到高**：byte0、byte1、byte2、byte3、byte4、byte5、byte6、byte7、  先传输低字节，即先传输byte0，再传输byte1，……  详见下面的SDO数据场 |

|  |  |
| --- | --- |
| SDO  通讯模型 | **客户端-服务器**  发送SDO请求的设备称为客户端，对请求进行SDO回复的设备作为服务端。  SDO通讯过程中只有一个客户端，只有一个服务端，  **一对一通讯，一次SDO传输的报文数目根据具体的通讯方式而定，**  通常由主机作为客户端，从机作为服务端  客户端设备使用SDO通信来 **读/写**服务端设备的对象字典中的某个**索引的子索引所存储的数值**  把读操作称为“上传”，把写操作称为“下载” |
| 例如：  设备A作为SDO1 和 SDO2 和SDO2的服务端，设备B作为SDO1 的客户端，设备C作为SDO2的客户端；  那么客户端对设备A中对象字典的访问就不会一致：  设备B改变了设备A的对象字典中的某一个对象的某个成员的值，但设备C却不知道，这样设备A中就会出现不同的对象字典的映像。这个问题只会在设备中存在多个SDO服务器的情况下出现，要解决这个问题，就只能允许一个SDO服务器对设备中的对象字典进行读/写，而其他的SDO服务器只能对对象字典进行读访问。 |

## SDO参数

|  |  |  |  |
| --- | --- | --- | --- |
| SDO服务器 | SDO服务器 | SDO server parameter(hex) | SDO client parameter(hex) |
| SDO1（默认） | 1200 | 1280 |
| SDO2 | 1201 | 1281 |
| SDO3 | 1202 | 1282 |
| …… | …… | …… |
| SDO128 | 127F | 12FF |
| 一个CANopen设备最多可以配置128个SDO服务器；  每一个SDO服务器可以提供两种SDO传输服务：  本设备作为SDO客户端：使用SDO client parameter 对象来配置相关参数  本设备作为SDO服务端：使用SDO server parameter 对象来配置相关参数  用户要配置某个SDO服务器时，就要同时配置该SDO服务器的这两个对象； | | |
| 一旦某个SDO服务器配置完毕，那么之后：每次设备使用该SDO服务器进行SDO传输时；  🡪设备要从两种服务中选择一种，即本次传输本设备是作为客户端还是作为服务端？  🡪如果选择作为客户端，那么就按照SDO client parameter 对象的配置来进行传输  如果选择作为服务端，那么就按照SDO server parameter 对象的配置来进行传输 | | |
| 一个CANopen设备至少有一个SDO服务器，即SDO1，用户可以自行添加SDO服务器 | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 | |
| SDO server parameter | RECORD | 1200~127F  —00 | / | RO | O | 条目数=3 | |
| 1200~127F  —01 | UNSIGNED32 | RW | O | bit | 描述 |
| 31 | valid=0：本SDO服务器作为**服务端**时**可以接收SDO**  valid=1：本SDO服务器作为**服务端**时**不接收SDO** |
| 30 | dyn=0：对本对象的设置会保存在驱动器的存储器中；  dyn=1：不保存对本对象的设置，本设备一旦断电，再次启动时，本对象的所有设置都回复默认 |
| 29 | frame=0：标准帧； frame=1：扩展帧；  CANopen设备都是使用标准帧，所以**frame总是为0** |
| 28:11 | 扩展CAN-ID；  CANopen设备都是使用标准帧，  所以**扩展CAN-ID总是全为0** |
| 10:0 | 基础CAN-ID；**注：valid=0时不能修改CAN-ID**  高4位是功能码，低7位是Node-ID  填入本SDO服务器作为**服务端**接收SDO时所接收的SDO报文的CAN-ID = 0x600+NodeID |
| 1200~127F  —02 | UNSIGNED32 | RW | O | bit | 描述 |
| 31 | valid=0：本SDO服务器作为**服务端**时**可以发送SDO**  valid=1：本SDO服务器作为**服务端**时**不发送SDO** |
| 30 | 同上 |
| 29 | 同上 |
| 28:11 | 同上 |
| 10:0 | 基础CAN-ID；**注：valid=0时不能修改CAN-ID**  高4位是功能码，低7位是Node-ID  填入本SDO服务器作为**服务端**发送SDO时所发送的SDO报文的CAN-ID= 0x580+NodeID |
| 1200~127F  —03 | UNSIGNED8 | RW | O | 当本设备选择作为服务端进行SDO传输，  对方设备(即客户端设备)是确定的，  本条目存储的就是对方设备的NodeID | |
|  | | | | | | | |
| SDO  client parameter | RECORD | 1280~12FF  —00 | / | RO | O | 条目数=3 | |
| 1280~12FF  —01 | UNSIGNED32 | RW | O | bit | 描述 |
| 31 | valid=0：本SDO服务器作为**客户端**时**可以发送SDO**  valid=1：本SDO服务器作为**客户端**时**不发送SDO** |
| 30 | 同上 |
| 29 | 同上 |
| 28:11 | 同上 |
| 10:0 | 基础CAN-ID；**注：valid=0时不能修改CAN-ID**  高4位是功能码，低7位是Node-ID  填入本SDO服务器作为**客户端**发送SDO时所发送的SDO报文的CAN-ID = 0x600+NodeID |
| 1280~12FF  —02 | UNSIGNED32 | RW | O | bit | 描述 |
| 31 | valid=0：本SDO服务器作为**客户端**时**可以接收SDO**  valid=1：本SDO服务器作为**客户端**时**不接收SDO** |
| 30 | 同上 |
| 29 | 同上 |
| 28:11 | 同上 |
| 10:0 | 基础CAN-ID；**注：valid=0时不能修改CAN-ID**  高4位是功能码，低7位是Node-ID  填入本SDO服务器作为**客户端**接收SDO时所接收的SDO报文的CAN-ID = 0x580+NodeID |
| 1280~12FF  —03 | UNSIGNED8 | RW | O | 当本设备选择作为客户端进行SDO传输，  对方设备(即服务端设备)是确定的，  本条目存储的就是对方设备的NodeID | |

## SDO快速传输数据场

|  |  |
| --- | --- |
| 快速  SDO  传输 | 只有在传输数据不超过4个字节的情况下才能使用。一次传输2条SDO报文；  客户端发送传输请求报文至服务端，服务端收到请求后向客户端发送一条回复报文；  适用于对象字典中大多数的对象，因为这些对象的成员的数据长度都在4字节以内； |
| SDO传输时间取决于：  SDO客户端与SDO服务器协议处理所需要的时间、使用的位速率、访问对象字典条目的时间  响应时间通常只有几毫秒，如果访问对象字典之后还要进行应答，那么访问时间有可能需要几百毫秒甚至几秒钟（最坏情况）。  在非易失性存储器中保存本地配置时尤其要注意这一点。 |
| 数据传输效率  数据传输效率E等于有效数据字节与待传输数据字节之比：  E=n/16 n=1~4 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 传输方式 | byte0  高4位 | byte0  低4位 | byte1~3 | byte4~7 | | | | |
| 下载请求 | 2 | 数据位中的有效字节数**n：**  3 🡪 n=4  7 🡪 n=3  B🡪 n=2  F🡪 n=1  需严格按照所下载的目标条目变量的数据类型来设置n的值 | byte1：索引值低8位  byte2：索引值高8位  byte3：8位子索引值  如目标索引是：  **60EFh** 索引的  **01h**子索引  byte1~3 填入：**0xEF 60 01** | byte4~7中：  低n 个字节是有效的；  高4-n个字节无效补0；  有效的n 个字节中：  如传输数据是0x657A00 | | | | |
| n | byte4 | byte5 | byte6 | byte7 |
| 4 | 00 | 7A | 65 | 00 |
| 3 | 00 | 7A | 65 | 00 |
| 2 | 无法将0x657A00完整传输 | | | |
| 下载回复 | 6 | 0 | 索引值 | 00 00 00 00 | | | | |
|  |  |  |  |  | | | | |
| 上传请求 | 4 | 0 | 索引值 | 00 00 00 00 | | | | |
| 上传回复 | 4 | 数据位无效字节数**n** | 索引值 | data | | | | |

## SDO分段传输数据场

|  |  |
| --- | --- |
| 分段  SDO  传输 | 在传输数据超过4个字节的情况下使用，最长不超过（2^32 -1）个字节；一次传输至少4条SDO报文；  先由客户端发送传输初始化请求至服务端，服务端收到请求后向客户端发送一条回复报文；  用于初始化的这两条报文指定了数据发送端和数据接收端  接下来数由数据发送端数据片段报文，每个数据片段报文包含最多7字节的数据；  数据接收端每接收到一个数据片段，就会进行回复；  数据发送端收到回复后发送下一个数据片段；  数据发送端在最后一个数据片段中会告诉接收端这是最后一个 |
| 数据传输效率：  假设具有7字节的数据分段利用率最大，g=n/7  n mod 7 ≠0 ：E=n/((g+2)\*16)  n mod 7 ＝0 ：E=n/((g+1)\*16)  n=7：E=0.218； n=8：E=0.166； n=64：E=363； n=256：E=0.421； n=1024：E=0.432 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SDO  下载 | 传输方式 | byte0高4位 | byte0低4位 | byte1~3 | byte4~7 |
| 下载初始化  请求 | 2 | 1 | 索引值 | 本次分段传输要下载的**总字节数;**  高字节用来存放所要传输的字节的低字节  如：byte4~7=17 00 00 00，  表明一共要下载23个字节 |
| 下载初始化回复 | 6 | 0 | 索引值 | 00 00 00 00 |
| 下载片段 | 0或1  下载的第一个片段→0  下载的第二个片段→1  下载的第三个片段→0  如此交替 | 偶数：后面还有片段  奇数：最后一个片段  数据位无效字节数n  n=本4位值/2 | 7字节全部用来存放数据：  高7-n 个字节是有效的；低n个字节无效补0；  有效的7-n 个字节中：  高字节用来存放所要传输的字节的低字节。  如要传输的数据是0x657A00  n=0时，CAN数据位是：0x00 7A 65 00 0000 00  n=3时，CAN数据位是：0x00 7A 65 00  n=5时，无法将0x657A00完整传输 | |
| 片段回复 | 2或3  回复第一个片段→2  回复第二个片段→3  回复第三个片段→2  如此交替 | 0 | 00 00 00 00 00 00 00 | |
| 下载片段 | 0或1 | 判断后面是否还有片段  数据位无效字节数n | data | |
| 片段回复 | 2或3 | 0 | 00 00 00 00 00 00 00 | |
| …… | …… | …… | …… | |
|  | | | | | |
| SDO  上传 | 传输方式 | byte0高4位 | byte0低4位 | byte1~3 | byte4~7 |
| 上传初始化请求 | 4 | 0 | 索引值 | 00 00 00 00 |
| 上传初始化回复 | 4 | 1 | 索引值 | 本次分段传输要上传的**总字节数** |
| 上传片段请求 | 6或7  第一个上传请求→6  第二个上传请求→7  第三个上传请求→6  如此交替 | 0 | 00 00 00 00 00 00 00 | |
| 片段回复 | 0或1  上传的第一个片段→0  上传的第二个片段→1  上传的第三个片段→0  如此交替 | 偶数：后面还有片段  奇数：最后一个片段  数据位无效字节数n  n=本4位值/2 | data | |
| 上传片段请求 | 6或7 | 0 | 00 00 00 00 00 00 00 | |
| 片段回复 | 0或1 | 判断后面是否还有片段  数据位无效字节数n | data | |
| …… | …… | …… | …… | |

## SDO块传输数据场

|  |  |
| --- | --- |
| 块传输 | 分段SDO传输每一段都要进行一次确认，不仅占用网络资源，而且浪费时间，效率不高。  块传输将数据划分成几块单一的包，在连续的请求或应答中逐块传输这些包。一次传输至少6条SDO报文；  先由客户端发送初始化传输请求至服务端，服务端收到请求后向客户端发送一条回复报文；  用于初始化的这两条报文指定了数据发送端和数据接收端  数据发送端发送第一个数据块块；即连续地依次发送数据片段报文，每个数据段报文包含最多7字节的数据，一个数据块最多连续发送127个数据片段，一个数据块的最后一个片段包含有它是最后一个的信息  数据接收端确认收到最后一个数据片段后，发送一个回复报文，一个数据块传输结束；  接下来数据发送端发送下一个数据块的第一个数据片段；  最后一个数据块传输结束后，数据发送端发送一个块传输结束报文，接端收到该结束报文后发送一个回复报文，由此，一次块传输过程结束。 |
| 通信对象之间要规定好每个块的消息数量，为安全起见，用户可以在块传输的最后一段中发送CRC校验和  发送完所有的块之后，客户端就会请求CRC校验和，传输就会以含有CRC校验和的结束消息而终止。  服务端确认结束消息，然后整个传输过程结束  在块传输过程中，数据的发送可能会造成总线在短时间内有极高的负载。因此，数据的接收方(块下载时是服务端，块上传时是客户端)必须具有缓冲整个块的能力；否则，数据接收方就会发出消息通知给数据发送端，这个块太大我接收不了，我现在已经接收到哪一个块中的哪一段，同时给出新的数据块大小；发送端就会按照新指定的块大小来重新传输整个块，而之前成功传输的块段也不需要再发送一遍。 |
| 数据传输效率  假设块大小是127字节，g=n/7 ，h=n/890 ； 写访问时i=0、读访问时i=1  n mod 7 ≠0 ：E=n/((g+h+i+5)\*8)  n mod 7 ＝0 ：E=n/((g+h+i+4)\*8)  n=7：E=0.145； n=8：E=0.142； n=64：E=533； n=256：E=0.761； n=1024：E=0.836 |

|  |  |  |  |
| --- | --- | --- | --- |
| 传输方式 | byte0高4位 | byte0低4位 | byte1~7 |
| 块下载  初始化请求 | C | 2：不支持CRC，下载结束请求报文的byte1~2为00 00  6：支持CRC，下载结束请求报文的byte1~2为CRC值 | byte1~3：索引值  byte4~7：将要下载的字节总数，高字节用来存放所要传输的字节的低字节。  如：下载的字节总数是16711685个字节  即FF 00 05，即00 FF 00 05，那么byte4~7=05 00 FF 00 |
| 块下载  初始化回复 | A | 0：不支持CRC  2：支持CRC | byte1~3：索引值  byte4：下一个数据块的允许的最大片段数目  byte5~7：00 00 00 |
| 下载片段 | 0：后面还有片段  8：最后一个片段 | 片段序号（0~F）  更大的序号需要修改高4位值 | data  CAN报文的数据位中，7个字节都是有效的  如传输的数据是0x657A00  CAN数据位就是：0x00 7A 65 00 00 00 00 |
| **bit7=1→最后一个片段，bit7=0→后面还有片段**  **bit6~0→片段序号** | |
| 下载片段 | 0或8 | 片段序号（0~F） | data |
| …… | …… | …… | …… |
| 块下载回复 | A | 2 | byte1：服务端成功收到的最后一个片段的序号  byte2：下一个数据块的允许的最大片段数目  byte3~7：00 00 00 00 00 |
| 下载片段 | 0或8 | 片段序号（0~F） | data |
| …… | …… | …… | …… |
| 块下载回复 | A | 2 | 同上 |
| 下载结束请求 | C | 最后一个数据块的最后一个片段的数据位中的无效字节数**n**  1→无效字节数=0  5→无效字节数=1  9→无效字节数=2  D→无效字节数=3 | byte1~2：CRC  byte3~7：00 00 00 00 00 |
| D | 1→无效字节数=4  5→无效字节数=5  9→无效字节数=6  D→无效字节数=7 |
| 下载结束回复 | A | 1 | 00 00 00 00 00 00 00 |

|  |  |  |  |
| --- | --- | --- | --- |
| 传输方式 | byte0高4位 | byte0低4位 | byte1~7 |
| 块上传  初始化请求 | A | 0：不支持CRC  4：支持CRC | byte1~3：索引值  byte4：客户端提出要求，在接下来的一个数据块的传输中，服务端上传的每个数据块中片段的数目  byte5：pst  byte6~7：00 00 |
| 块上传  初始化回复 | C | 2：不支持CRC，上传结束回复报文的byte1~2为00 00  6：支持CRC，长传结束回复报文的byte1~2为CRC值 | byte1~3：索引值  byte4~7：将要上传的字节总数，高字节用来存放所要传输的字节的低字节。  如：上传的字节总数是16711685个字节  即FF 00 05，即00 FF 00 05，那么byte4~7=05 00 FF 00 |
| 上传片段请求 | A | 3 | 00 00 00 00 00 00 00 |
| 上传片段 | 0：后面还有片段  8：最后一个片段 | 片段序号（0~F）  更大的序号需要修改高4位值 | data |
| 上传片段 | 0或8 | 片段序号（0~F） | data |
| …… | …… | …… | …… |
| 上传片段请求 | A | 2 | byte1：客户端成功收到的最后一个片段的序号  byte2：要求下一个数据块的片段的数目  byte3~7：00 00 00 00 00 |
| 上传片段 | 0或8 | 片段序号（0~F） | data |
| …… | …… | …… | …… |
| 上传片段请求 | A | 2 | 同上 |
| 块上传结束  回复 | C | 最后一个数据块的最后一个片段的数据位中的无效字节数**n** | byte1~2：CRC  byte3~7：00 00 00 00 00 |
| D |
| 收到，本次块上传就此结束 | A | 1 | 00 00 00 00 00 00 00 |

## 

## SDO中断传输数据场

|  |  |
| --- | --- |
| 中断  传输 | 仅由服务端发送，客户端接收，一次传输一条报文；  上述三种传输方式都是最先由客户端发送请求开始的；如果有错误，服务端无法回复客户端想要的报文，那么服务端就会回复一条中断传输报文，中断本次传输。 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 传输方式 | byte0高4位 | byte0低4位 | byte1~7 | |
| 中断传输回复 | 8 | 0 | byte1~3：索引值 | byte4~7：中断码  收到的CAN报文的数据位中，4个字节都是有效的  如收到的数据位是0x00 00 02 06  数据就是是：0x06 02 00 00 |

|  |  |
| --- | --- |
| 中断码 | 错误情况 |
| 05 03 00 00h | 替换位没有交替转变 |
| 05 04 00 00h | SDO协议超时 |
| 05 04 00 01 h | 非法或未知的协议字(SDO请求报文的高8位) |
| 05 04 00 02 h | 无效的块大小（仅针对块传输模式） |
| 05 04 00 03 h | 无效的序号（仅针对块传输模式） |
| 05 04 04 00 h | CRC错误（仅针对块传输模式） |
| 05 04 05 00h | 内存溢出 |
|  |  |
| 06 01 00 00 h | 对象不支持访问 |
| 06 01 00 01h | 试图读只写对象 |
| 06 01 00 02h | 试图写只读对象 |
| 06 02 00 00h | 对象字典中的对象不存在 |
| 06 04 00 41h | 对象不能够映射到PDO |
| 06 04 00 42h | 映射的对象的数目和长度超出了PDO长度 |
| 06 04 00 43h | 一般性参数内部不兼容 |
| 06 04 00 47h | 一般性设备内部不兼容 |
| 06 06 00 00h | 硬件错误导致访问对象失败 |
| 06 07 00 10h | 数据类型不匹配，服务参数长度不匹配 |
| 06 07 00 12h | 数据类型不匹配，服务参数长度太长 |
| 06 07 00 13h | 数据类型不匹配，服务参数长度太短 |
| 06 09 00 11h | 子索引不存在 |
| 06 09 00 30h | 超出参数的值范围(写访问时) |
| 06 09 00 31h | 写入参数的值太大(写访问时) |
| 06 09 00 32h | 写入参数的值太小(写访问时) |
| 06 09 00 36h | 最大值小于最小值 |
|  |  |
| 08 00 00 00h | 一般性错误 |
| 08 00 00 20h | 数据不能传送或保存到应用 |
| 08 00 00 21h | 数据不能传送或保存到应用，原因是：本地控制 |
| 08 00 00 22h | 数据不能传送或保存到应用，原因是：当前设备状态 |
| 08 00 00 23h | 对象字典动态产生错误或对象字典不存在 |
| 08 00 00 24h | 没有数据 |

# 同步对象SYNC

## SYNC通讯

SYNC生产者定期自动发送SYNC报文，

同步报文是可选项，并不是每台设备有具备。

|  |  |
| --- | --- |
| 通讯模型 | SYNC使用**生产者-消费者模型**  同步报文应用于同步模式的PDO通讯，所以也使用生产者-消费者模型  当进行同步通讯时，CANopen网络中**有且仅有一个同步生产者** 和 **1~~126个同步消费者**；  是一对一通讯或一对多通讯，一次传输的报文的数目等于消费者的数目。 |

## SYNC参数

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 | |
| COB-ID SYNC message | VAR | 1005 | UNSIGNED32 | RW | Conditional  (M for SYNC producers) | COB-ID(不能PDO映射) | |
| bit | 描述 |
| 31 | 填充位 x=0 |
| 30 | gen.=1：本设备产生SYNC报文  gen.=0：本设备不产生SYNC报文  注意并不是所有设备都有SYNC generate 的功能，如果设备不具备此功能却将它的gen.参数设为1，那么就会报错。 |
| 29 | frame=0：标准帧；  frame=1：扩展帧；  CANopen设备都是使用标准帧  所以**frame总是为0** |
| 28:11 | 扩展CAN-ID；  CANopen设备都是使用标准帧  所以扩展CAN-ID**总是全为0** |
| 10:0 | 基础CAN-ID；11位中  高4位是功能码，功能码是默认为0001  低7位是Node-ID；任意配置，因为同步报文是广播发送的；NodeID只是作为一个标志；  但一般会默认设置为0x80  **gen.=1时，不能做修改** |
| Communication cycle period | VAR | 1006 | UNSIGNED32 | RW | Conditional  (M for SYNC producers) | 循环周期(不能PDO映射)  SYNC生产者每经过一个循环周期所指定的时间发送一个同步报文；  仅对于SYNC生产者有效  **单位：us**  **0：不使用同步传输**  **1~~0xFFFF FFFF：循环周期的时间值** | |
| Synchronous window length | VAR | 1007 | UNSIGNED32 | RW | O | 同步窗口长度(不能PDO映射)  PDO生产者收到指定个数的同步报文后，需要在同步窗口长度指定的时间内发送PDO；  **单位：us**  **0：发送PDO没有时间限制**  **1~~0xFFFF FFFF：同步窗口长度的时间值**  同步窗口长度的数值不能超过循环周期；  一般就设置同步窗口长度=循环周期； | |
| Synchronous counter overflow value | VAR | 1019 | UNSIGNED8 | RW | M | 同步计数器的溢出值(不能PDO映射)  计数器从1开始递增计数，  递增到溢出值后返回到1重新计数；  仅对于SYNC生产者有效  **0：没有同步计数器**  **2~240：溢出值**  设置溢出值时，要根据PDO通讯参数的传输类型；  如果一个设备每3个同步信息发送一个PDO，另一个每设备隔4个同步信息发送一个PDO，那么计数器就必须在读数为12的倍数（不超过240）时溢出。 | |

# 过程数据对象PDO

## PDO报文

|  |  |
| --- | --- |
| 传输次序 | 发送PDO报文时先发送高位，再发送低位； |
| 报文结构 | PDO报文属于CAN**标准数据帧报文**；其数据结构参见《CAN总线协议》  PDO报文的**CAN-ID**、**Data Field、DLC**见下表： |

|  |  |  |  |
| --- | --- | --- | --- |
| CAN-ID | PDO生产者使用其某个TPDO发送PDO报文🡪PDO消费者的某个RPDO  要成功实现数据的传输，那么要求：  该PDO报文的CAN-ID 与 这个RPDO的通讯参数的CAN-ID 相同 | | |
| 如果先配置消费者的某个RPDO，那么配置时：  (1)功能码选择默认的RPDO功能码  (2)Node-ID配置为该消费者设备的Node-ID  再配置PDO报文的CAN-ID：  与该RPDO的CAN-ID相同 | | RPDO1：0100 （CAN-ID = 0x200 + Node-ID）  RPDO2：0110 （CAN-ID = 0x300 + Node-ID）  RPDO3：1000 （CAN-ID = 0x400 + Node-ID）  RPDO4：1010 （CAN-ID = 0x500 + Node-ID） |
| 如果先配置要传输的PDO报文，那么配置时：  (1)功能码选择默认的TPDO功能码  (2)Node-ID配置为该生产者设备的Node-ID  再配置消费者的某个RPDO：  与PDO报文的CAN-ID相同 | | TPDO1：0011 （CAN-ID = 0x180 + Node-ID）  TPDO2：0101 （CAN-ID = 0x280 + Node-ID）  TPDO3：0111 （CAN-ID = 0x380 + Node-ID）  TPDO4：1001 （CAN-ID = 0x480 + Node-ID） |
| DLC | PDO报文 | DLC=8，RTR=0 | |
| PDO远程帧 | DLC=0，RTR=1 | |
| 数据场 | PDO远程帧 | 数据场为空 | |
| PDO报文 | PDO报文的8位数据位全部用来填写要发送的数据；  这些数据由消费者的某个RPDO接收后，会存储到该RPDO的映射参数所指定的对象条目中；  一个RPDO可以设置多个映射条目；每个对象都相应设置了位宽；  比如该消费者设备使用RPDO3接收PDO报文，RPDO3的映射参数对象设置了4个条目：  第一个条目(子索引00h) **=3，**表示映射条目的个数 第二个条目(子索引01h)，设置位宽=8bit，那么就接收PDO报文的高8位数据  第三个条目(子索引02h)，设置的位宽=32bit，那么就接收PDO报文的接下来的32数据  第四个条目(子索引03h)，设置的位宽=16bit，那么就接收PDO报文的接下来的16位数据，  **注意如果某个条目接受的数据超过8bit，那么接收时低字节在前，高字节在后** | |

## PDO通讯

|  |  |
| --- | --- |
| 生产者-消费者模型 | 发送PDO消息的设备作为生产者，接收PDO消息的设备作为消费者。  进行PDO通讯时，只有一个生产者，可以有多个消费者；  **PDO报文仅由生产者发送，消费者接收；**  是一对一或一对多通讯**一次传输的报文数目等于接收者的数目**  通常主机作为PDO消费者，从机作为PDO生产者，从机设备就可以定时传输过程数据给主机，  过程数据是**实时数据**，是需要高频率交换的数据。为了实现定时实时传输，就要求从机发送PDO报文并不是由设备控制的而由某些条件触发传输，一旦条件满足就立即进行传输；传输的内容、方式等都预先设定好；PDO整个的8个字节的数据空间都用来传输数据，能充分地进行数据传输。 |
| PDO同步传输 | 使用PDO同步传输时，设备收到指定个数的SYNC报文后，必须保证在**同步时间窗口范围内**发送PDO报文；即要在下一个同步报文到来之前把PDO发送出去；  因为有时碰巧收到某个SYNC报文后，要发送好多个PDO，那么后面几个PDO可能还没来得及发送出去，下一下同步报文已经到来了；  🡪一方面要避免这样的情况，如使用同步初始值，让几个PDO的发送的时机错开  🡪另一方面，如果发生了这样的情况，那么那几个滞后的PDO要不要发送了呢？  🡪如果要发送，那么PDO的接收方也要配置好在同步时间窗口之外收到PDO的情况  🡪还应该考虑另外一种极为罕见的情况，就是重复发送2次同步对象，因此设备制造商必须定义在这种情况下的同步PDO。当本地发送方的EOF字段的最后一位受到干扰时，这种情况就有可能发生。 🡪用户必须注意同步对象是否涉及接收PDO(RPDO)或者发送PDO(TPDO)。  🡪同步时间窗口的长度对SYNC通信是没有影响的；但是位速率和循环周期这两个参数是要考虑的  实际运作时，其实并不会严格地按照对象字典循环周期来发送SYNC报文；很容易延迟发送，  尤其在发送了一条SYNC报文之后还要进行异步传输时（NMT指令、保护监视、SDP访问、非同步PDO等）  SYNC传输延时（抖动）的时间约等于从接收到同步报文再到传输完一帧报文所花费的时间，最长为109个位时间加上填充位（平均为两个位时间）。在这种情况下，位速率将起到一定的作用，见下表   |  |  |  | | --- | --- | --- | | 位速率 | 循环周期 /ms | 8字节消息引起的延时/% | | 125 kbit/s | 10 | 约10 | | 500 kbit/s | 10 | 约2.5 | | 1 Mbit/s | 10 | 约1 | |
| 同步PDO传输举例 | 两个设备通过一个同步PDO进行通信。  设备A发送PDO，设备B接收PDO；  两个设备中的PDO都为同步PDO（传输类型1~240）。两个设备都接收来自主机的SYNC  时刻1：主机发送同步对象  时刻2：设备A、设备B都收到同步对象  时刻3：发送方（设备A）发送同步PDO  时刻4：接收方（设备B）接收到来自设备A的PDO  时刻5：主机发送同步对象  时刻6：接收方（设备B）接收到的同步PDO数据进行接收处理  设备B在时刻4收到同步PDO，但不会马上处理收到的数据，设备B在时刻6才处理收到的PDO。  也就是说，在时刻3和时刻6之间至少有一个同步周期才能对同步PDO数据进行同步。  在某些情况下，PDO数据在这段时间内会失效。  如果只将发送方的TPDO配置成同步传输，且把接收方的RPDO定义为事件驱动类型，就可以避免这个问题。这样就可以大大的改变时间特性。  时刻3和时刻6之间的持续时间等于由事件驱动的传输持续时间。 |

## PDO通讯参数

**注意，一定要在设备的NMT预操作状态下才可以设置其通讯参数**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TPDO | TPDO1 | TPDO2 | TPDO3 | TPDO4 | TPDO5 | TPDO6 | TPDO7 | TPDO8 |
| Index | 1800h | 1801h | 1802h | 1803h | 1804h | 1805h | 1806h | 1807h |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RPDO | RPDO1 | RPDO2 | RPDO3 | RPDO4 | RPDO5 | RPDO6 | RPDO7 | RPDO8 |
| Index | 1400h | 1401h | 1402h | 1403h | 1404h | 1405h | 1406h | 1407h |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Sub-Index  (hex) | Data Type | Access | Category | 描述 | |
| 通讯  参数 | ARRAY | 00h | UNSIGNED8 | RO | M | 子索引数目，范围：2~~5 | |
| 01h | UNSIGNED32 | RW | M | bit | Value |
| 31 | 0：本通讯参数对象对应的PDO 有效  1：本通讯参数对象对应的PDO 无效 |
| 30 | 0：数据帧  1：远程帧 |
| 29 | 0：标准帧  1：扩展帧； |
| 28:11 | 扩展CAN-ID； |
| 10:0 | **11位中的高4位是功能码，低7位是Node-ID；**  PDO生产者使用它的某个TPDO发送PDO报文，  就需要配置该TPDO的通讯参数  一旦满足触发条件，该TPDO就会自动发送PDO报文，  报文的CAN-ID 就等于该TPDO通讯参数的CAN-ID |
| 02h | UNSIGNED32 | RW | M | Value  (desimal) | 传输类型(触发条件) |
| 0 | 同步非循环  先是规定的事件发生，然后收到一个同步对象，然后：  TPDO：发送相应的PDO报文  RPDO：检查总线上是否有符合要求的PDO报文，如果有则进行接收 |
| 1~240 | 同步循环 ，其具体数值表示**同步溢出值n**  触发条件是设备收到n个同步报文 |
| 252 | 同步远程帧  每接收到1个同步报文，本PDO进行一次采样，但不发送  接收到请求该数据的远程帧后发送本PDO；  要求该远程帧的的CAN-ID等于本PDO的COB-ID。 |
| 253 | 事件远程帧  每发生一次事件，本PDO进行一次采样，但不发送；  接收到请求该数据的远程帧后发送本PDO；  要求该远程帧的的CAN-ID等于本PDO的COB-ID。 |
| 254 | 制造商事件  每发生一次事件，设备对本PDO进行采样并发送 |
| 255 | CIA协议事件  每发生一次事件，设备对本PDO进行采样并发送 |
| 03h | UNSIGNED32 | RW | O | 禁止时间 | |
| 04h | 保留 | | | | |
| 05h | UNSIGNED32 | RW | O | 事件计时器 | |
| 06h | UNSIGNED32 | RW | O | 同步初始值 | |

|  |  |
| --- | --- |
| 禁止时间 | 用于定义发送具有相同CAN-ID的两个PDO之间至少要间隔的时间；  比如：某个PDO的触发事件第一次发生时设备直接发送该PDO，在这之后禁止时间所规定的时间内，该事件再次发生则不会触发PDO的发送。  如果将最长的禁止时间给予最高优先级的TPDO，将第二长的禁止时间给予第二高优先级的TPDO，以此类推，那么PDO数据的发送就会有规律的进行。在不允许发送最高优先级PDO的这一段时间内，便可以发送次高优先级的TPDO。 |
| 事件计时器 | 为异步传输所设置（254、255）  事件计时器溢出可以作为一个触发事件：计时器计数达到所设置的的时间，可以触发TPDO的发送，之后计数器立即复位。  事件计时器溢出可以可以用来监控PDO：在定义的时间内没有收到PDO，则通知应用程序进行相应的处理。 |
| 同步初始值 | 对于同步循环的通讯方式；  比如设备配置成每收到2条同步消息发送一个PDO，同步初始值设置为1，那么设备在收到第3、5、7这类奇数编号的同步消息之后就会发送数据；  这样可以实现总线负载的合理分配分配：比如设置另一台设备也是每收到2条同步消息发送一个PDO，但同步初始值为2，那么设备在收到4、6、8这类奇数编号的同步消息之后就会发送数据；  两台设备发送PDO的时机就错开了； |
| 触发事件 | 可以是PDO采样的数据(某个条目变量)发生变化、设备内部的时间、周期性运行的定时器行为；或者是接收到一个特定的消息。 除此以外，许多设备子协议还把设备从“预操作”状态转变到“操作”状态也视为一种触发事件，这样启动时就可以获得全部PDO数据。  消费者发送远程帧到生产者回复PDO数据帧之间的响应时间取决于所使用的CAN控制器。  好一点的控制器可以直接回复远程帧，差一点的可能响应时间较长  由于控制器的好坏对于用户而言并不透明，而且一些CAN控制器不支持远程帧，所以不推荐远程帧触发。 |

## PDO映射参数

**注意，一定要在设备的NMT预操作状态下才可以设置其映射参数**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TPDO | TPDO1 | TPDO2 | TPDO3 | TPDO4 | TPDO5 | TPDO6 | TPDO7 | TPDO8 |
| Index | 1A00h | 1A01h | 1A02h | 1A03h | 1A04h | 1A05h | 1A06h | 1A07h |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RPDO | RPDO1 | RPDO2 | RPDO3 | RPDO4 | RPDO5 | RPDO6 | RPDO7 | RPDO8 |
| Index | 1600h | 1601h | 1602h | 1603h | 1606h | 1605h | 1606h | 1607h |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Sub-Index  (hex) | Data Type | Access | Category | 描述 |
| 映射参数 | ARRAY | 00h | UNSIGNED8 | RO | M | 条目的数目；即映射对象的数目  范围：0~40；  0表示这个PDO不传输任何数据 |
| 01h | UNSIGNED32 | RW | O | 31:16---被映射的的对象的索引；  高8位存储索引值低8位，低8位存储索引值高8位；  15:8----被映射的的对象的子索引  7:0----被映射的的条目的位宽，范围：0x08、0x10、0x18、0x20  （根据该条目在对象字典中的描述） |
| 02h | UNSIGNED32 | RW | O |  |
| …… | …… | …… |  | …… |

|  |  |
| --- | --- |
| TPDO  映射参数 | **设备A使用TPDO1发送PDO报文，设置设备A的TPDO1的映射参数：**  设置设备A想要发送的**对象的索引的子索引**，以及该对象所存储的数据的**位宽**；  一旦设备A的TPDO1触发条件满足，设备A就自动发送一个PDO报文到总线中去，该报文的8个字节的数据位全部用来填装数据，所填装的数据就是TPDO1映射参数所设置对象所存储的数据。  可以在TPDO1中设置多个对象，即一个PDO可以发送多个对象的数据；  限制1：最多映射40个对象  限制2：所有映射对象数据加起来不能超过8字节。 |
| RPDO  映射参数 | **设备B使用RPDO1接收PDO报文，设置设备B的RPDO1的映射参数，**  RPDO1的映射参数中设置设备B的对象字典的某个索引的子索引；  一旦设备B的RPDO1接收数据，设备B就会把接收的数据存放在其RPDO1的映射参数所指定的设备B的对象字典的索引的子索引中。  因为设备B的RPDO1是用来接收设备A的TPDO1的；  所以**配置设备B的RPDO1的映射参数时时，子索引的数目要等于设备A的TPDO1的映射参数**  **传输时，设备A的TPDO1的第一个映射对象的数据将传输到**  **设备B的RPDO1的第一个对象的索引中;以此类推；** |
| 配置  映射参数  步骤 | 使用SDO在设备的**预操作状态**配置各个PDO的映射参数  1.先禁止该RPDO的映射：使用SDO在 00h 子索引写0  2.配置映射对象：使用SDO在其他子索引写入映射对象  3.使能RPDO的映射：使用SDO在 00h 子索引写入在第2步中写入的对象的个数 |

# 时间戳TIME STAMP

## TIME通讯

|  |  |
| --- | --- |
| 生产者-消费者模型 | 发送TIME消息的设备作为生产者，接收TIME消息的设备作为消费者。  进行TIME通讯时，只有一个生产者，可以有多个消费者；是一对零一对一或一对多通讯  **TIME报文仅由生产者发送，消费者接收；一次传输的报文数目等于接收者的数目** |

The TIME producer broadcasts the time stamp object. This TIME provides the simple network clock.

There may be a time jitter in transmission by the TIME producer corresponding approximately to the

latency due to some other message being transmitted just before the TIME.

In order to guarantee timely access to the network the TIME is given a very high priority CAN-ID (see

sub-clause 7.5.2.15). CANopen devices that operate a local clock may use the TIME object to adjust

their own time base to that of the time stamp object producer. The details of this mechanism are

implementation specific and do not fall within the scope of this specification.

## TIME参数设置

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 | |
| COB-ID  time stamp object | VAR | 1012 | UNSIGNED32 | RW | Conditional  (M if  Emergency  is supported) | COB-ID(不能PDO映射) | |
| bit | 描述 |
| 31 | consume=0：本设备不接收time stamp报文  consume=1：本设备可以接收time stamp报文 |
| 30 | produce=0：本设备不发送time stamp报文  produce=1：本设备可以发送time stamp报文  不是所有设备都能够发送time stamp报文，如果设备不具备此功能而将她的produce置1，那么就会返回错误 |
| 29 | frame=0：标准帧；  frame=1：扩展帧；  CANopen设备都是使用标准帧  所以**frame总是为0** |
| 28:11 | 扩展CAN-ID；  CANopen设备都是使用标准帧  所以扩展CAN-ID**总是全为0** |
| 10:0 | 基础CAN-ID；  11位中的高4位是功能码  低7位是Node-ID；  功能码是默认为0001  用户可以在设备的预操作状态下修改；  但一般就使用默认的。  **consume=1或produce=1时，不能修改CAN-ID** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | Object code | Index  (hex) | Data Type | Access | Category | 描述 |
| High resolution time stamp | VAR | 1013 | UNSIGNED32 | RW | O | This object shall indicate the configured high resolution time stamp. It may be mapped into a PDO in  order to exchange a high resolution time stamp message. Further application specific use is  encouraged.  VALUE DEFINITION  单位：us |