CAN 知识简析及应用分享

前言

众所周知 CAN 通讯是一个很传统的通讯方式，并且在传感器通讯和一些其他的应用领域也还算是主流的通讯方式，加上其应用之广泛，还是有不少工程师对这方面的应用有兴趣，所以在这里将自己的经验总结并分享给大家。

个人之前并未涉足 CAN 通讯的领域，由于工作要求并在一些得力风电同事（例如梁德春，刘懿等）的帮助下，经过一段时间的积累和实践操作，成功从零基础转变到 CAN 通讯基本无大碍，但是由于没有经过专业的理论学习，大多的经验也都是基于实践中的总结，一些专业术语或者描述无法与专业的书籍比较，敬请见谅；另外这本书在很大程度上分享的是一些经验，并不会将每个细节上的 CAN 原理从专业书籍中照搬过来，如有需要，请参考专业书籍。

本书的内容安排将基于倍福的 CAN 通讯设备产品系列，结合相关的应用，从接线到应用，分享个人的一些学习心得：第一章首先介绍 CAN 总线线缆和 CAN 接线的 PIN 脚，并基于倍福的硬件对接头的 PIN 脚做了罗列方便大家以后查阅；第二章通过几个简单的实际应用将 COBID，PDO（过程数据），SDO（服务数据）和对象字典等 CAN 总线中常见的术语对大家进行介绍，结合实际的应用希望帮助读者能更好的熟悉概念；第三章从总线报文来进一步学习，认识总线数据，会分析，会找错，那应用应该就没有太大的问题了；接着的第四章按照常见的基本 CAN 总线的应用方式来举例说明，进一步阐述常见的配置上的多种可能；最后一章补充说明基于倍福 CAN 总线设备的另一种方式-ADS，通过 ADS 实现 CAN 设备的控制。

当然此书编写的过程中难免有一些笔误或者由于个人的见解和描述与读者有一些认知上的冲突，希望大家提出批评和建议！邮件请联系：jie.xu@beckhoff.com.cn

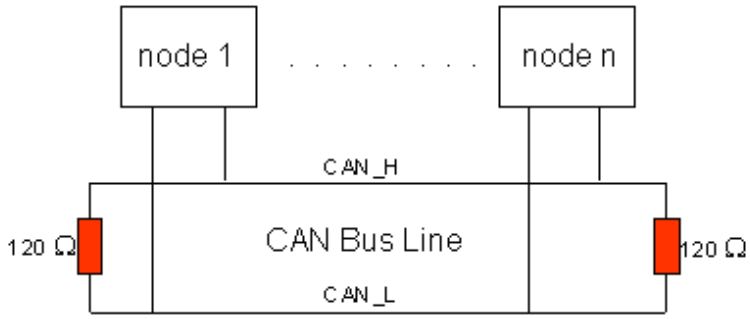
许洁 2018 年 4 月

第一章 CAN 硬件相关

相信很多做通讯的同仁会在日常工作中也有同感，通讯的很多问题往往在于通讯线缆。工欲善其事必先利其器，那么开篇就谈谈 CAN 通讯的接线吧。

第一节 主从站两端安装 120 Ohm 的电阻（后称“终端电阻”）

正常通讯都需要在 CAN-High 和 CAN-Low 信号线缆两端安装终端电阻，用来防止信号在线缆的传输过程中由于反射造成衰减（即使短距离也建议安装），CAN 总线基本无需中继。目前倍福的全线产品除 FC51xx 系列的板卡上默认加载终端电阻，其余都需在使用的过程中使用带终端电阻的线缆或者自己组装，如下图所示，需在主从站设备的两端都使用接有终端电阻的线缆。



第二节 线缆长度与波特率的关系

线缆的长度往往和波特率有密切的关系，但是这边提到的距离，例如以 1M 的波特率为例，并不保证 20m 是没有问题，其实有报告说对于 1M 的波特率来说 5m 的最保险的，但是也有一些实际的应用在 20m 的情况下也是通讯正常的，不过这是对于线性系统来说。如果有分支 1M 的话，带分支最多只能 0.3m，如果是 500K 的话，分支线长不能超过 0.5m，总线长不能超过 1.5m。

Baud Rate	Bus length
1 Mbit/s	< 20 m*
500 kbit/s	< 100 m
250 kbit/s	< 250 m
125 kbit/s	< 500 m
50 kbit/s	< 1000 m
20 kbit/s	< 2500 m
10 kbit/s	< 5000 m

倍福 CAN 线缆可选用的型号有：ZB5100 或者 ZB5200，接头的型号：ZS1052-3000，此接头均默认配备 120Ohm 终端电阻，并可通过拨片来控制电阻是否接合使用。

Pin 脚对应线缆颜色（以倍福为例）

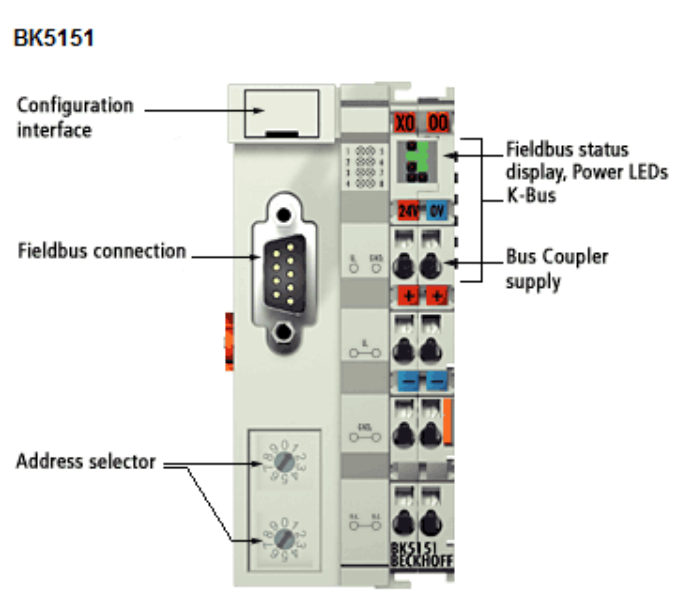
BK51x0 pin BC5150/BX5100	BK5151, CX805x, CX-B510/M510	Fieldbus Box pin	FC51xx pin/EL6751	Function	ZB5100 cable color	ZB5200 cable color
1	3	3	3	CAN Ground	**black**/ (red) 黑	**black** 黑
2	2	5	2	CAN Low CAN 低	**black**	**blue**
3	5	1	5	Screen	Filler strand	Filler strand
4	7	4	7	CAN high CAN 高	**white** 白	**white** 白
5	9	2	9	not used	**(red)**	**(red)**

CANOpen 一般要接 3 条线（2-CANLow 白线，3-地线(等电位线)黄线，7-CAN High 绿线，5-屏蔽线）

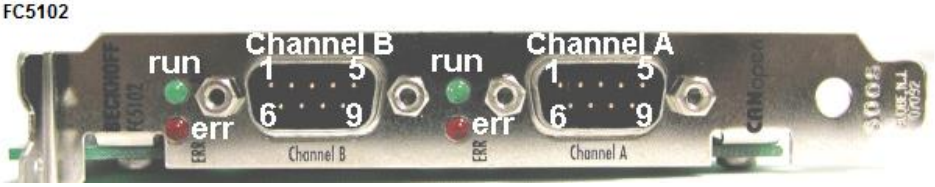
第三节 倍福硬件常见 Pin 脚排线

a) 9 针头 D-sub：

Pin	Assignment
2	CAN low (CAN-)
3	CAN ground (internally connected to pin 6)
6	CAN ground (internally connected to pin 3)
7	CAN high (CAN+)



FC5102



b)　5 针接头

5: reserved
4: CAN high
3: Shield
2: CAN low
1: CAN Ground

c)　M12 接头

1: Shield
2: reserved
3: CAN Ground
4: CAN high
5: CAN low

第二章 CAN 基础和基于倍福 CAN 硬件的初级应用举例

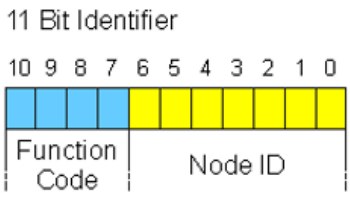
倍福基于 CAN 总线的硬件基本有下面几类：

- BK 系列总线耦合器：该款总线耦合器系列是倍福早期的 K-Bus 现场总线系列之一，它支持传统 CAN 波特率从 10K~1M，如果没有特殊要求一般采用自适应波特率的方式（也可通过拨码或者 KS2000[1]进行修改），常在 CAN 回路中作为一个 Node（节点/从站），采集 IO 信号及传感器的信号或者对一些执行器进行简单的控制。
- BC/BX 系列 PLC 总线控制器：该款为硬 PLC 并兼有耦合器的功能。
- EL6751-00x0 基于 EtherCAT 的 CAN 网关设备：归属于 EtherCAT IO，根据其在 CAN 网络的不同角色可以选择相应的主从模块。
- FC510x 等 CAN 总线板卡：用于 IPC 的扩展，接口常为 PCI 或者 PCI-express/Mini PCI.
- CXxxxx-B510 或者 CXxxxx-M510：用于 Embedded PC 总线扩展。

这边首先带大家熟悉一下 CAN 总线的一些基本概念，最后通过结合一些应用的分析，希望能更好的强化对于基本概念的理解。

第一节 节点号 COBID

COBID 是 CAN 总线的唯一的识别号，常见 11bit 即 CAN2.0A（CAN2.0B 为 29bit，与 A 的区别通常也就是最高位为 1. ），它集成了硬件的节点号[2]和功能码信息，一般数字越小，传输优先级越高。



CANopen功能符的分配表	
功能段FS	概念
0000	NMT 系统管理
0001 0x8x	SYNC同步标记
0011 0x18x	PD0(tx)过程数据对象发送包
0100 0x2xx	PD0(rx)过程数据对象接收包
1011 0x58x	SDO(tx)服务数据的发送包
1100 0x6xx	SD0(rx)服务数据的接收包

我们常用的 COBID 基本分为三类：

- 过程数据 PDO：用来传输和接受周期性数据，每个 PDO 可以包含的数据为 8byte。
- 启动和停止帧：此类数据帧一般优先级比较高，常用来控制类似启动，停止，复位，报错等数据的发布。
- 服务数据 SDO：常用来作为参数设置。Tx SDO→0x580+节点号；Rx SDO→0x600+节点号。SDO 传输的数据也是 8Byte。

[1] KS2000 是倍福的一个配置和修改参数的付费软件，对于早期的 K-Bus 等总线设备建议选择这款软件和配合使用相应的线缆 KS2000-Z2-USB。

[2] Node ID-节点号：即我们常说的站号。

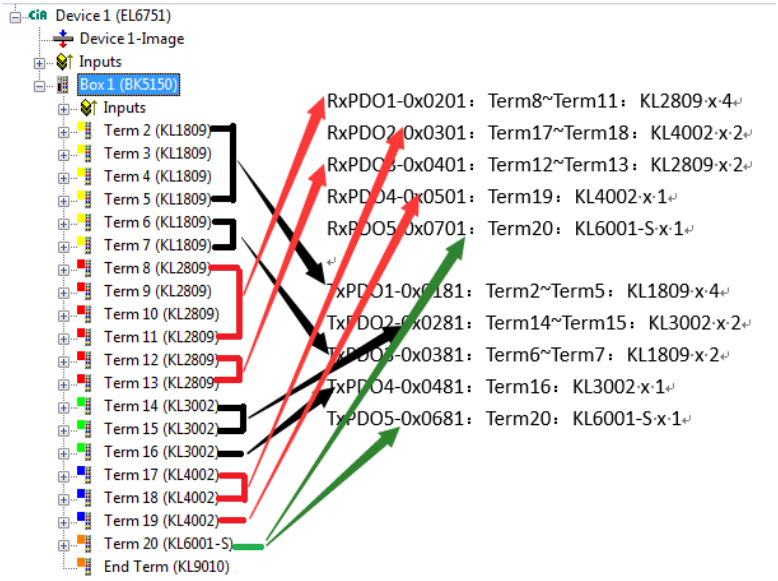
第二节 过程数据 PDO

对于 CAN 总线来说，过程数据即所谓的 PDO，是总线上的周期性数据，理论上只要配置好相应的 PDO 即可实现主从的数据交换，当然此处暂时不考虑时序等问题。过程数据 PDO，分为两类 TxPDO 和 RxPDO，TxPDO 即 transmit process data object，RxPDO 即 receive process data object，顾名思义，TxPDO 是要传输给主站的数据，常见为输入数据，例如传感器数据等；RXPDO 是接收主站的一些指令的输出数据，例如执行器开关量等。一般 TxPDO1 的 COBID 为 0x180+站号（例如 5 号站就是 0x185），依次分别为 0x28x,0x38x,0x48x...；一般 RxPDO1 的 COBID 为 0x200+站号（例如 5 号站就是 0x205），依次分别为 0x30x,0x40x,0x50x...通常每个主站可以添加的 TxPDO 和 RxPDO 一般不会也不建议超过 11 对，TwinCAT 软件从第五对 PDO 开始 COBID 就不会自动分配，均需用户自己手动修改，请注意一些特殊的 COBID 是不能占用的，需要保证唯一，例如网络管理 NMT，SYNC 相关和 SDO 相关的 COBID，过多的 PDO 会一定程度的增加总线负载，影响通讯的正常。

下面以倍福的 CAN 总线耦合器 BK5150 为例，来分析过程数据 PDO 的配置。

Object	Function	Function code	Resulting COB ID		Object for communication Parameter / mapping
			hex	dec	
Emergency	Status / error	1	0x81 - 0xFF	129 - 255	- / -
PDO1 (tx)	dig. inputs	11	0x181 - 0x1FF	385 - 511	0x1800 / 0x1A00
PDO1 (rx)	digital outputs	100	0x201 - 0x27F	513 - 639	0x1400 / 0x1600
PDO2 (tx)	analog inputs	101	0x281 - 0x2FF	641 - 767	0x1801 / 0x1A01
PDO2 (rx)	analog outputs	110	0x301 - 0x37F	769 - 895	0x1401 / 0x1601
PDO3 (tx)	analog inputs*	111	0x381 - 0x3FF	897 - 1023	0x1802 / 0x1A02
PDO3 (rx)	analog outputs*	1000	0x401 - 0x47F	1025 - 1151	0x1402 / 0x1602
PDO4 (tx)	analog inputs*	1001	0x481 - 0x4FF	1153 - 1279	0x1803 / 0x1A03
PDO4 (rx)	analog outputs*	1010	0x501 - 0x57F	1281 - 1407	0x1403 / 0x1603

用 TwinCAT 软件对 BK5150 进行配置，一般只需进行 IO 扫描，系统会自动识别出耦合器后面安装的 IO 模块，无须手动进行 PDO 配置。不过我们这边可以借助 TwinCAT 软件得知 PDO 是如何分配的，这种方法可以为第三方的 CAN 主站设备带 BK5150 的时候做为参考[3]。



[3]注意：TC3 暂时不能组态 BX/BC/BK 系列的设备，故所有的该系列设备都将以 TC2 为组态做实例。

倍福 CAN 耦合器的默认配置 PDO1 分配给数字量输入和输出，PDO2 分配给模拟量，PDO3 和更高的 PDO 是要根据耦合器后面的模块来分配的，如果第一对 PDO 没有分配完的数字量也从这里继续分配。通过上图的分析可以看到第一对 PDO 全是分配的数字量的 IO，然后第二对 PDO 分配的全是模拟量 IO，然后从第三对开始，陆续将没有分配完的数字量和模拟量依照顺序分配，并且在分配过程数据的时候有意的将不同型号的数据分配在不同的 PDO 中。一般情况 TwinCAT 软件中可以在 BK5120/BK5150 上添加 11 对 PDO 的过程数据，但是如果一定要使用 16 对的话，需要先手动将 BK5120/BK5150 配置成一个通用的 CanOpen 设备（general CanOpen Box），然后手动添加 PDO，并且手动配置 PDO 的节点号。不过这样的情况下更建议客户选用两个 BK5120/BK5150。

另外一个在使用第三方主站配置 BK 系列耦合器的时候常会发现这样几个问题:

1) 如果添加的模块太多，导致 PDO 超过默认的 4 对，经常发现只有前 4 对过程数据可以进行交换，其他都是无效的。
   这边需要在 0x5500 写 0xFFFF0000，倍福的 BK5150 在使用倍福的系统控制的时候会自动写 0xFFFF0000.但是如果是第三方的主站的时候是需要自己手动添加的（SDO 中添加）。
   下图是 Information 中关于 PDO activ 的说明：

Activate PDOs

Index	Sub-index	Name	Type	Attri-bute	Mapping	Default value	Meaning
0x5500	0	Activate PDO Defaults	Unsigned32	rw	N	0x00000000	sets PDO communication parameters for PDOs 2...11

CANopen defines default identifiers for 4 transmit (Tx) and 2 receive (Rx) PDOs, all other PDOs being initially deactivated after the nodes have started up. Index 0x5500 can activate all the PDOs that, in accordance with the terminals inserted, are filled with process data (manufacturer-specific default mapping). A manufacturer-specific default identifier allocation is carried out here for PDO5...11, while the transmission type and a uniform inhibit time is set for PDO2...11. PDOs that do not have process data (and which are thus superfluous in the present configuration) are not activated.

2) 在第三方主站带 BK5150 的时候，，模拟量输入的信号不能正常采集，其实是由于一开始默认的模拟量输入信号事件触发的这个功能默认是关闭的→开启即可：
   0x6423 写 0x1 即可开启。
   下图是 information 中关于开启模拟量输入信号事件触发采集的说明：

Event driven analog inputs

Index	Sub-index	Name	Type	Attri-bute	Mapping	Default value	Meaning
0x6423	0	Global Interrupt Enable	Boolean	rw	N	FALSE (0)	Activates the event-driven transmission of PDOs with analog inputs.

Although, in accordance with CANopen, the analog inputs in TxPDO2..4 are by default set to transmission type 255 (event driven), the event (the alteration of an input value) is suppressed by the event control in object 0x6423, in order to prevent the bus from being swamped with analog signals. It is recommended that the flow of data associated with the analog PDOs is controlled either through synchronous communication or through using the event timer. In event-driven operation, the transmission behavior of the analog PDOs can be parameterized before activation by setting the inhibit time (object 0x1800ff, sub-index 3) and/or limit value monitoring (objects 0x6424 + 0x6425) and/or delta function (object 0x6426).

第三节 服务数据 SDO 和对象字典 EDS

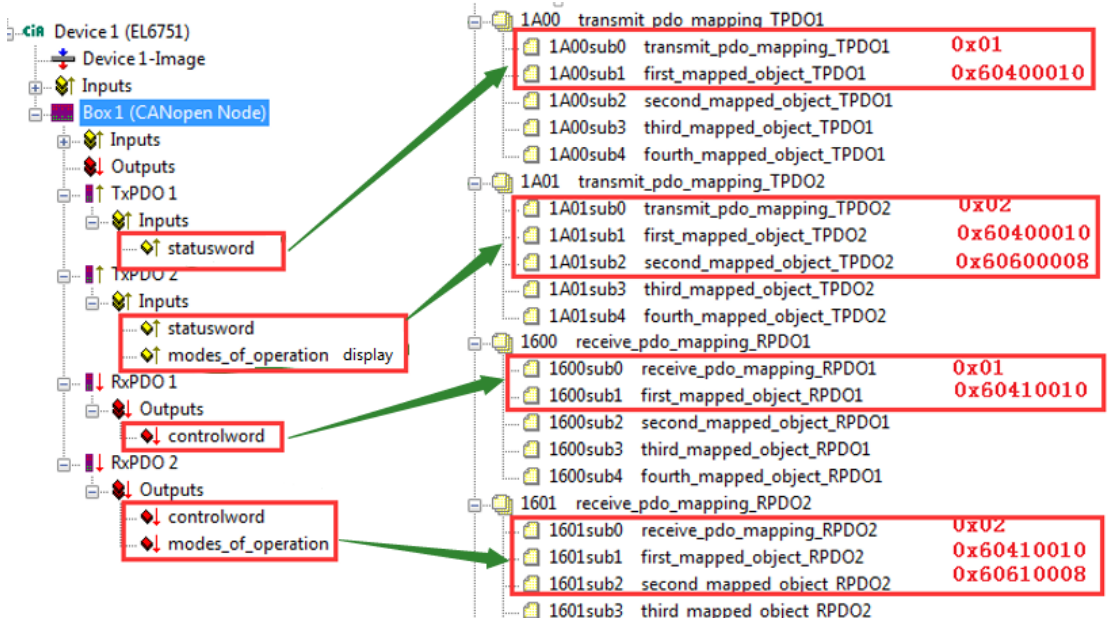
服务数据 SDO 和对象字典是 CANOpen 中才引入的内容：

- EDS，对象字典分为两部分，第一部分包括基本的设备信息，例如设备 ID，制造商，通信参数等等。第二部分描述了特殊的设备功能。一个 16 位的索引和一个 8 位的子索引唯一确定了对象字典的入口。通过对象字典的入口可以对设备的"应用对象"进行访问，设备的"应用对象"可以是输入输出信号，设备参数，设备功能和网络变量等，它相当于设备描述文件。如果在倍福的主站设备中配置第三方 CANOpen 设备，一般需要导入第

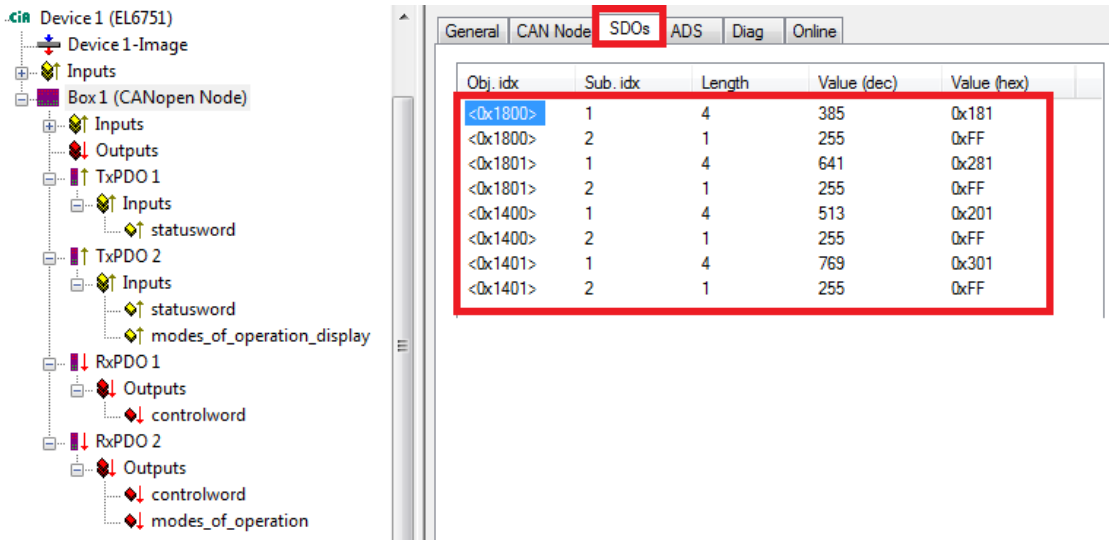
三方的 EDS 文件，从而获得设备的应用对象的信息。

- 服务数据 SDO，通过设定对象字典中的 PDO 的入口及索引和子索引，配置过程中传输参数以及传输大数据块，标示了 PDO 在对象字典中的映射关系，很类似倍福的 CoE 列表。SDO 通常做的更多的是参数的设置，但是我们也可以通过 SDO 来读取 PDO 的数据。通过修改 SDO 里面的 PDO 的映射关系可以实现对不同应用对象的调整，就像我们在 EtherCAT 中的 PDO 的 assignment。

下面以一个例子来分析说明，范例中使用的 EDS 是早年的一款摩力电机的 CANOpen 伺服驱动器，在 EDS 的描述文件中仅做了一部分 PDO 对象的映射（如下图），



- 0x160x: RxPDO 和输出应用对象的映射关系，子索引 sub0 一般表示的是该 PDO 下映射的变量的数量，之后子索引为实际映射的应用对象。以 0x1600 为例，首先它对应的是 IO 列表下 CAN 节点的第一个 RxPDO1，通过 eds 文件可以看到该 PDO 只包含一个变量（sub0 为 1），该变量对应的是驱动的控制字（CAN402 标准中对象 0x6041 即表示控制字，0x10 表示变量类型为两个字节），TwinCAT 中加载 EDS 文件即可看到 RxPDO 里面确实只有一个变量并且该变量的类型也是 2 个字节。同样分析 0x1601，即可得知 RxPDO2 含有两个变量（sub0 为 2），第一个变量为控制字（对应的应用对象为 0x6041 也是 2 个字节），第二个变量为操作模式（应用对象 0x6061，大小为一个字节 0x08）
- 0x1A0x：TxPDO 和输入应用对象的映射关系，同样子索引 sub0 表示的是该 PDO 下映射的变量的数量，之后子索引为与实际输入应用对象的映射。以 0x1A00 为例，首先它对应的是 CAN 节点的第一个 TxPDO1，该 PDO 只有一个变量（因为 sub0 为 1），对应的应用对象是驱动器的状态字（0x6040，0x10 表示大小也为 2 个字节）；同样 0x1A01 对应 TxPDO2，该 PDO 默认有两个变量（sub0 为 2），第一个变量同样也是状态字（0x60400010），第二个变量为模式显示（0x6060，0x08 表示大小为一个字节）。
- 在这边不对通讯的类型进行过多的介绍，通讯类型的定义一般在 0x1400（Rx）和 0x1800（Tx）里面，通常分为同步和异步两种



但是在应用中，客户需要增加更多的 PDO，所以需要重新修改 SDO 即 0x160x 和 0x1A0x 中和应用对象的映射关系。可以看到（上图）默认的 eds 文件的 SDO 里面仅有的是对 COBID 和通讯类型的说明，倍福 CANOpen 主站及软件均支持在软件中设置 SDO 的映射关系，而无需修改 EDS 文件。在和厂家的工程师沟通后，说可以参照 EtherCAT 接口的驱动器的 xml 文件去配置 PDO 中的变量（如下图）。

```
<RxPdo Sm="2" Fixed="0">
  <Index>#x1600</Index>
  <Name>Outputs</Name>
  - <Entry>
      <Index>#x6040</Index>
      <SubIndex>0</SubIndex>
      <BitLen>16</BitLen>
      <Name>Controlword</Name>
      <DataType>UINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x6060</Index>
      <SubIndex>0</SubIndex>
      <BitLen>8</BitLen>
      <Name>Mode_of_Operation</Name>
      <DataType>USINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x6098</Index>
      <SubIndex>0</SubIndex>
      <BitLen>8</BitLen>
      <Name>Homing_Method</Name>
      <DataType>USINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x607A</Index>
      <SubIndex>0</SubIndex>
      <BitLen>32</BitLen>
      <Name>Target_Position</Name>
      <DataType>DINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x60FF</Index>
      <SubIndex>0</SubIndex>
      <BitLen>32</BitLen>
      <Name>Target_Velocity</Name>
      <DataType>DINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x6071</Index>
      <SubIndex>0</SubIndex>
      <BitLen>16</BitLen>
      <Name>Target_Torque</Name>
      <DataType>UINT</DataType>
    </Entry>
</RxPdo>
```

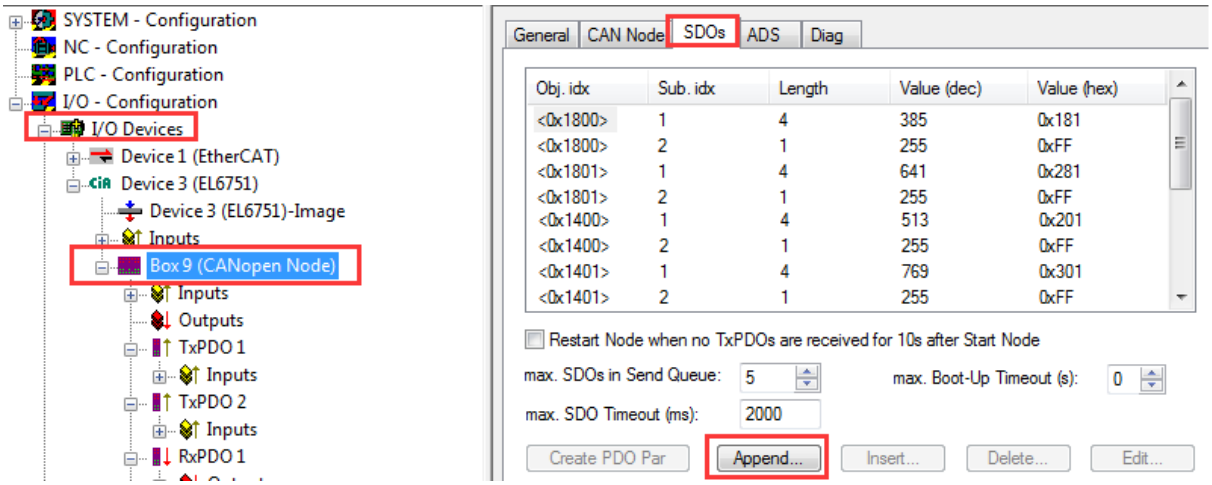
```
- <TxPdo Sm="3" Fixed="0">
    <Index>#x1A00</Index>
    <Name>Inputs</Name>
  - <Entry>
      <Index>#x6041</Index>
      <SubIndex>0</SubIndex>
      <BitLen>16</BitLen>
      <Name>Statusword</Name>
      <DataType>UINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x6064</Index>
      <SubIndex>0</SubIndex>
      <BitLen>32</BitLen>
      <Name>Position_Actual_Value</Name>
      <DataType>DINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x6061</Index>
      <SubIndex>0</SubIndex>
      <BitLen>8</BitLen>
      <Name>Modes_Of_Operation_Display</Name>
      <DataType>USINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x1001</Index>
      <SubIndex>0</SubIndex>
      <BitLen>8</BitLen>
      <Name>Error_Register</Name>
      <DataType>USINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x606C</Index>
      <SubIndex>0</SubIndex>
      <BitLen>32</BitLen>
      <Name>Velocity_Actual_Value</Name>
      <DataType>DINT</DataType>
    </Entry>
  - <Entry>
      <Index>#x6077</Index>
      <SubIndex>0</SubIndex>
      <BitLen>16</BitLen>
      <Name>Torque_Actual_Value</Name>
      <DataType>UINT</DataType>
    </Entry>
</TxPdo>
```

- 在客户 XML 文件中已经将应用对象的 PDO 详细的列举了，所以在配置 CANOpen 的 SDO 映射对象时候可以完全采纳，例如控制字即 0x6040001**0**，最后的低字节为 0x10 因为控制字为 16 位的 UINT。工作模式该对象为 8 位的参数，所以为 0x6060000**8**。对于 32 位的目标位置该对象 0x607A00**20**.

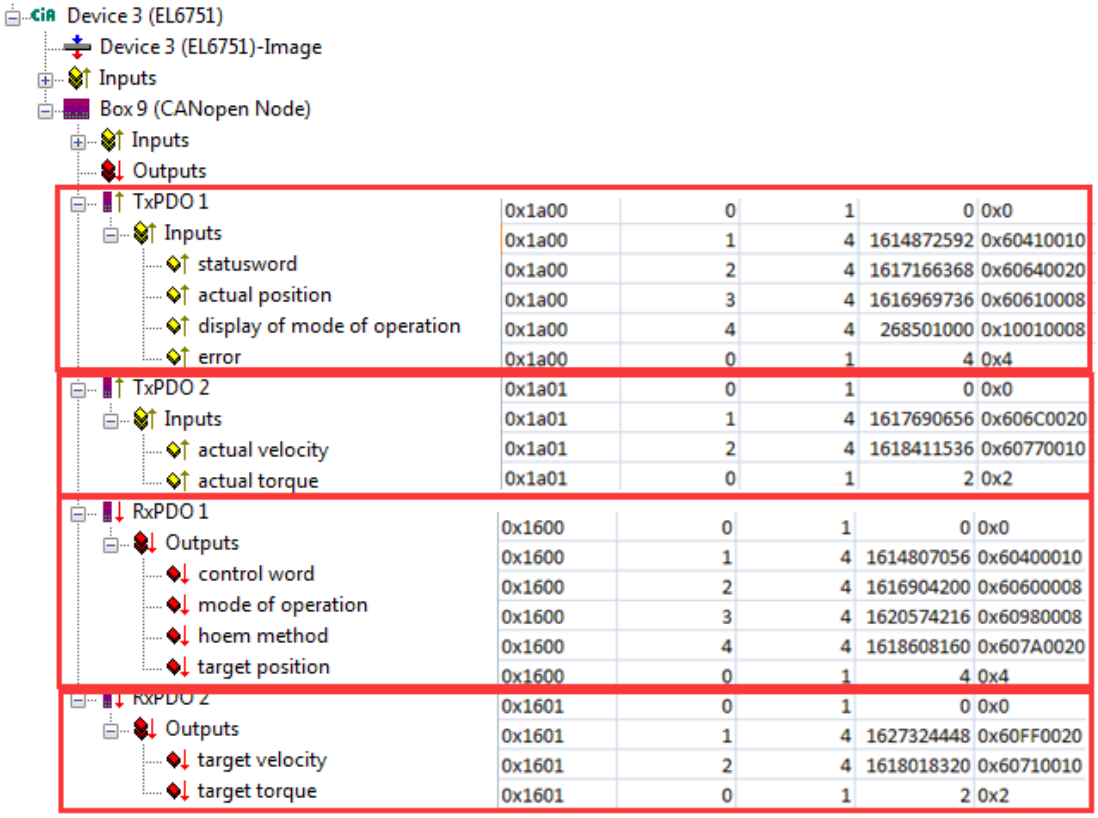
- 由于 CANOpen 中每个 PDO 可以映射的应用对象最多只能 8Byte，所以对照 EtherCAT 的 XML，XML 中的输入和输出变量均需要分配在两个 PDO 中，其中将控制字，控制模式，回参方式和目标位置配置在第一个 RxPDO1 中，将目标速度和目标扭矩配置在第二个 RxPDO2 中；将状态字，当前位置，模式显示和错误寄存器配置在第一个 TxPDO1 中，将当前速度和当前扭矩配置在第二个 TxPDO2 中。

0x1600	0	1	0	0x0
0x1600	1	4	1614807056	0x60400010
0x1600	2	4	1616904200	0x60600008
0x1600	3	4	1620574216	0x60980008
0x1600	4	4	1618608160	0x607A0020
0x1600	0	1	4	0x4
0x1601	0	1	0	0x0
0x1601	1	4	1627324448	0x60FF0020
0x1601	2	4	1618018320	0x60710010
0x1601	0	1	2	0x2
0x1a00	0	1	0	0x0
0x1a00	1	4	1614872592	0x60410010
0x1a00	2	4	1617166368	0x60640020
0x1a00	3	4	1616969736	0x60610008
0x1a00	4	4	268501000	0x10010008
0x1a00	0	1	4	0x4
0x1a01	0	1	0	0x0
0x1a01	1	4	1617690656	0x606C0020
0x1a01	2	4	1618411536	0x60770010
0x1a01	0	1	2	0x2

- 在 CANOpen 中，要修改 SDO 的映射，一般建议先将 SDO 中 Sub-Index0 改为 0（Sub-Index0 表示的该 PDO 中映射的应用对象的数量），先设无效，再修改，最后把 Sub Index 改回实际的应用对象的数量。
- 接着在 TwinCAT 软件中添加 SDO 相应的数据，点击 TwinCAT 软件中 I/O 设备中的第三方设备（CANOpen Node），点击右侧的“SDO“菜单，选择”Append...“（如下图），逐字逐条添加所有的应用对象（如上图）。



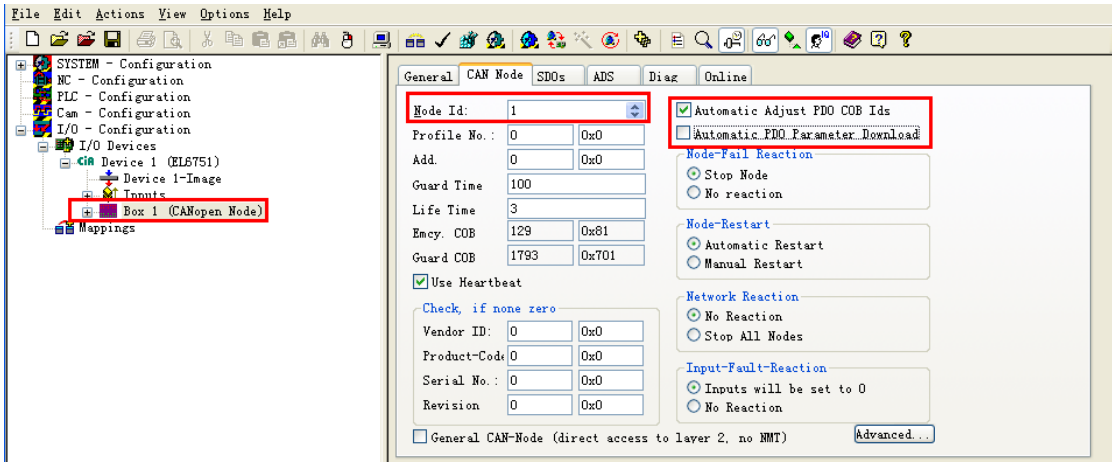
- 最后在 I/O 设备下面手动给每个 PDO 添加相应的变量并匹配变量类型和修改变量的名字以方便辨识。最后配置好的结果和下图一样。



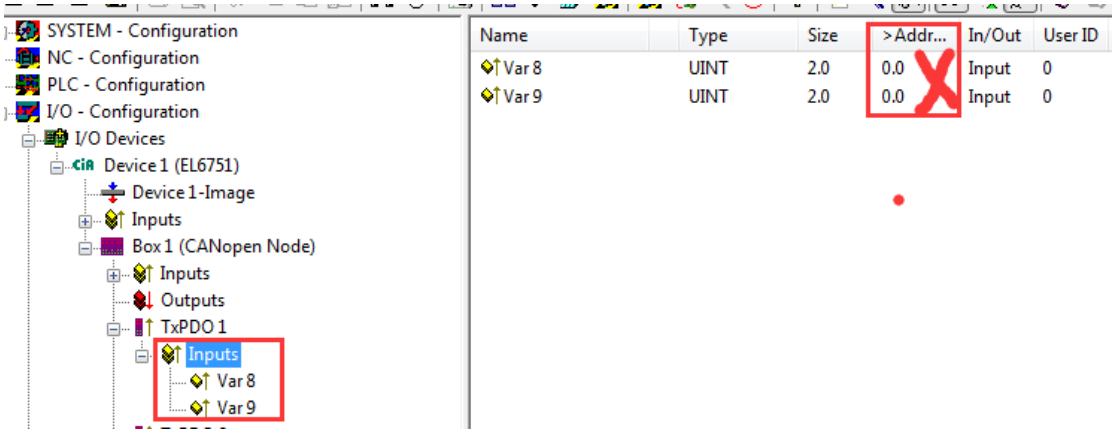
- 一般配置结束后，建议 reload IO device 几次，确认通讯的正常建立，因为 reload IO device 之后，硬件和软件的配置会进行对比，如果不匹配，在 TwinCAT 软件中会有相应的总线报错和诊断信息的。

注意：

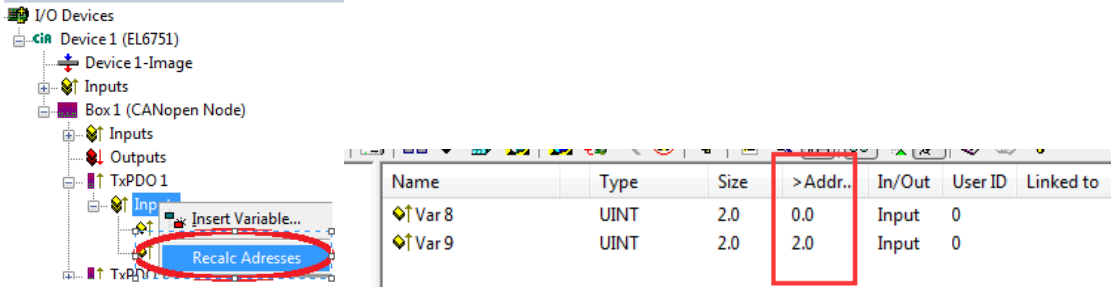
1) 如果出现说配置的数据和实际不匹配的情况（SDO 不匹配，或者 not all TxPDo are received），可以参考将 TwinCAT 中的 CAN Node 页面的一些设置做一些修改，不过注意不是所有的从站都支持手动配置 PDO 的映射。



2) 另外在手动在 PDO 中添加变量的时候，如果是手动一个一个添加的，务必注意，添加的变量的便宜地址不能重复第一个 PDO 的默认偏移地址为 0，第二个 PDO 的默认偏移地址为 8，依此类推。第二个及其后面的变量要设一个偏置移地址，否则读出来的数值和第一个相同。如第一个变量为 2BYTE 则第二个变量设置偏移地址为 0+2=2.如果在 PDO2 中进行改操作则偏移地址为 8+2=10。



或者采用系统的自动计算地址。



第四节 NMT message 网络管理服务和同步帧

- NMT 主要是提供网络管理（如初始化、启动和停止节点，侦测失效节点）等服务的功能。最常用的就是模块控制服务（NMT Module Control），它实现节点的启动和停止及错误的复位，例如总线出现故障时，可以辅助做节点的复位和启动，当然所有设备也必须支持 NMT 模块控制服务。NMT Module Control 消息不需要应答。NMT 消息格式如下：

NMT-Master → NMT-Slave(s)

COB-ID	Byte 0	Byte 1
0x000	CS	Node-ID

节点号如果特定表示操作仅对该站有效，当节点号为 0 的时候，则表示将对所有的 NMT 从站设备操作。

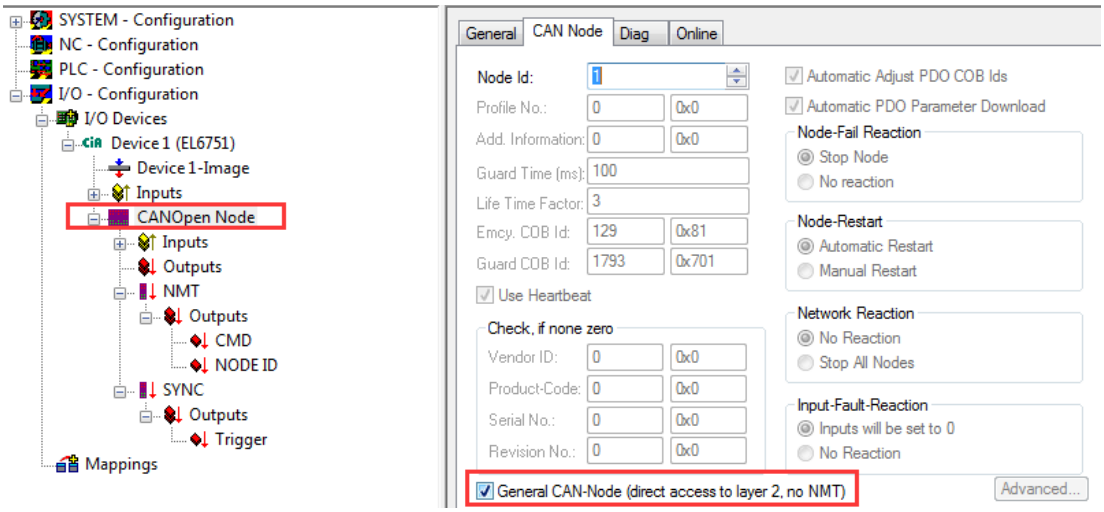
命令字 CS	NMT 服务
1	Start Remote Node
2	Stop Remote Node
128	Enter Pre-operational State
129	Reset Node
130	Reset Communication

- 同步帧，主要的目的就是根据前一个SYNC接收到的报文更新输出，SYNC报文常常不传送数据以保证报文尽量短。

CANopen 预定义主/从连接集的广播对象			
对象	功能码 （ID-bits 10-7）	COB-ID	通讯参数在 OD 中的索引
NMT Module Control	0000	000H	-
SYNC	0001	080H	1005H，1006H，1007H

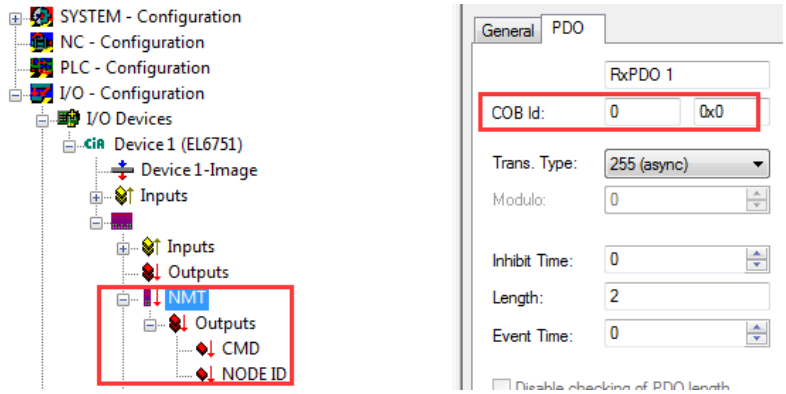
下面用一个例子来实现一个从站的 reset。

1) 首先在 I/O 设备，CANOpen 主站下面添加一个从站（CANOpen Node），勾选 General CAN-NODE（direkt access to layer 2, No NMT），如果已有某个从站已勾选该选项，则可以跳过直接进入第 2 步。



NodeID 可以是任何值，这个从站仅是一个虚拟的从站，主要是为了发 CAN NMT 的控制服务命令。

2) 在一个空的 RxPDO 中添加两个字节类型变量，这样可以通过 PLC 来发命令和节点号。修改该 PDO 的 COBID 为 0，这样才实现 NMT 模块控制服务: 0x00 <CS><Node-ID>



3) 接着添加 SYNC 报文，它理解起来就像一个 trigger，保持通讯的建立，如果没有这个信号，很可能通讯会断开。所以我们需要再建立一个 RxPDO，修改 COBID 为 0x80，并添加一个字节的数据，链接 PLC 变量，在 PLC 程序中将该变量周期性累加 1。

4) 这样 PLC 程序代码如果运行起来，CMD 给 0x01，Node-ID 给 0x01 的话，就表示将启动 CAN 总线中的 1 号站；如果 CMD 给 0x81，Node-ID 给 0x01 的话，就表示将 reset CAN 总线中 1 号站的通讯。如果 Node-ID 不给特定的值（即 0x00），NMT 操作将对所有的 CAN 总线上的从站有效。SYNC 的信号是保证总线的通讯的长期有效。

第三章 CAN 总线数据分析

前一章节和大家简单讨论和认识了一些 CAN 中的常见的术语，但是个人认为只有熟悉总线数据报文才更重要，因为从个人理解上来说，总线上的数据是原始数据，CANOpen 或者其他 CAN 的应用是将总线数据更像是以一种特定的方式包装数据而已，当然这边不考虑时序等方面的其他和协议相关的内容。并且在应用的过程中，个人发现很多情况下如果没有过往通讯经验的情况下，经常是一开始并不能决定用何种方式去配置总线的数据，往往客户也会和你说总线的数据是这么样的，或者说他的总线数据可能不是规范的种种，所以最好的方式就是往往是先通过分析总线数据，抽丝剥茧，最后找到最适合的应用方式。

第一节 过程数据 PDO 数据分析

如果客户的应用仅是对于一些过程数据进行读取或者设置，一般对总线中的过程数据 PDO 抓包，便可以直接观察到需要发送和接受的数据，通过简单的数据分析基本可以确定用何种方式去建立通讯。

常见 PDO 报文的基本格式为：**<PDO > <data 1~8 byte>**

下面以两次实际应用为例，曾经一客户自主研发变流器 CAN 的板卡，设备曾与贝加莱 CAN 设备成功通讯，客户初期仅提供我们抓包数据，希望通过倍福的主站设备也能通讯，总线数据举例如下

0x00000000	接收	0	0x0059c0c4	00000481	数据帧	标准帧	0x06	28	00	00	00	00	00		
0x00000001	接收	0	0x0059c0de	00000181	数据帧	标准帧	0x06	00	04	00	00	00	00		
0x00000002	接收	0	0x0059c0e2	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x00000003	接收	0	0x0059c0f7	00000281	数据帧	标准帧	0x06	00	00	00	00	00	00		
0x00000004	接收	0	0x0059c111	00000381	数据帧	标准帧	0x06	00	00	06	01	08	fd		
0x00000005	接收	0	0x0059c11f	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x00000006	接收	0	0x0059c12b	00000481	数据帧	标准帧	0x06	28	00	00	00	00	00		
0x00000007	接收	0	0x0059c145	00000181	数据帧	标准帧	0x06	00	04	00	00	00	00		
0x00000008	接收	0	0x0059c15d	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x00000009	接收	0	0x0059c15f	00000281	数据帧	标准帧	0x06	00	00	00	00	00	00		
0x0000000a	接收	0	0x0059c178	00000381	数据帧	标准帧	0x06	00	00	06	01	04	fd		
0x0000000b	接收	0	0x0059c192	00000481	数据帧	标准帧	0x06	28	00	00	00	00	00		
0x0000000c	接收	0	0x0059c1a5	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x0000000d	接收	0	0x0059c1b0	00000181	数据帧	标准帧	0x06	00	04	00	00	00	00		
0x0000000e	接收	0	0x0059c1ca	00000281	数据帧	标准帧	0x06	00	00	00	00	00	00		
0x0000000f	接收	0	0x0059c1e3	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x00000010	接收	0	0x0059c1e5	00000381	数据帧	标准帧	0x06	00	00	06	01	fc	fc		
0x00000011	接收	0	0x0059c203	00000481	数据帧	标准帧	0x06	28	00	00	00	00	00		
0x00000012	接收	0	0x0059c21d	00000181	数据帧	标准帧	0x06	00	04	00	00	00	00		
0x00000013	接收	0	0x0059c221	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x00000014	接收	0	0x0059c237	00000281	数据帧	标准帧	0x06	00	00	00	00	00	00		
0x00000015	接收	0	0x0059c250	00000381	数据帧	标准帧	0x06	00	00	06	01	03	fd		
0x00000016	接收	0	0x0059c268	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x00000017	接收	0	0x0059c26b	00000481	数据帧	标准帧	0x06	28	00	00	00	00	00		
0x00000018	接收	0	0x0059c287	00000181	数据帧	标准帧	0x06	00	04	00	00	00	00		
0x00000019	接收	0	0x0059c2a1	00000281	数据帧	标准帧	0x06	00	00	00	00	00	00		
0x0000001a	接收	0	0x0059c2a6	00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01	44
0x0000001b	接收	0	0x0059c2bb	00000381	数据帧	标准帧	0x06	00	00	06	01	03	fd		

1) 首先过滤重复的数据，选择部分需要仔细分析的基本数据

00000481	数据帧	标准帧	0x06	28	00	00	00	00	00	
00000181	数据帧	标准帧	0x06	00	04	00	00	00	00	
00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01 44
00000281	数据帧	标准帧	0x06	00	00	00	00	00	00	
00000381	数据帧	标准帧	0x06	00	00	06	01	08	fd	
00000201	数据帧	标准帧	0x08	00	00	00	00	00	00	01 44

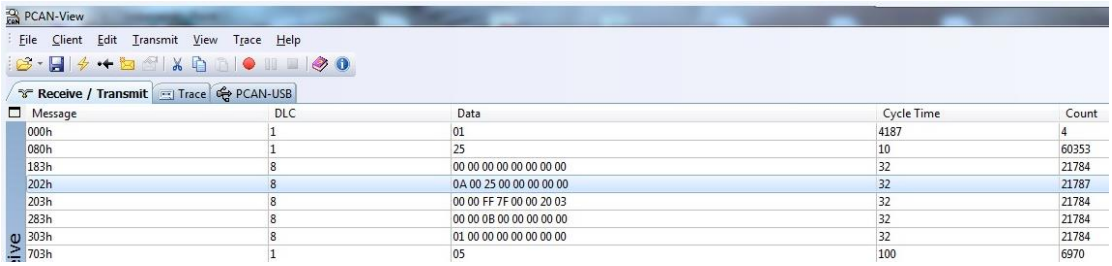
2) 接着看上面的数据根据 CAN 的规则可以知道基本分成两类：
   - RxPDO 只有 1 个→0x201 这个数据是主站发送给从站的。
   - TxPDO 有 4 个→0x181,0x281,0x381,0x381 这些数据是从从站接收到的。
3) 但是由于这边只有一些 PDO 的数据，并且没有 EDS 文件，并且也没有任何类似同步帧的总线数据，考虑到后期的程序的逻辑的灵活性，所以基本我们可以确定这种类型的总

线数据用 CAN Interface 基本没有问题。

4) 接着最后用 CAN Interface 实现了数据的发送和接受，程序部分基本如下（PLC 部分的详细说明在下一章第四章会有详细的介绍）：

```
IF NOT CfgFlag THEN
    stTxMessage.CobId:=16#4028;  (*for COB ID 16#201; bit 0-bit 4: data length; bit 5-bit 15:cob ID*)
    CfgFlag:=TRUE;

END_IF

IF TxCounter_output= TxCounter_input  THEN
    NoOfRxMessages_output := NumberOfMessagesToSend;
    TxCounter_output := TxCounter_output + 1;
END_IF


IF RxCounter_output<> RxCounter_input THEN
    RxCounter_output:= RxCounter_output+1;
END_IF

(*Example code: Message send to CANOpen slave*)
stTxMessage. TxMessage[6]:=16#01;
stTxMessage. TxMessage[7]:=16#44;
```

还有一次应用：柯蒂斯 curtis 的驱动器，从 PCAN 的抓包可以看到如果 0x203 发送数据，驱动器会反馈 0x283，所以可以用 general CAN Node 的方式，手动发同步帧即可，具体的实现见下一章的第三节。



Message	DLC	Data	Cycle Time	Count
000h	1	01	4187	4
080h	1	25	10	60353
183h	8	00 00 00 00 00 00 00 00	32	21784
202h	8	0A 00 25 00 00 00 00 00	32	21787
203h	8	00 00 FF 7F 00 00 20 03	32	21784
283h	8	00 00 0B 00 00 00 00 00	32	21784
303h	8	01 00 00 00 00 00 00 00	32	21784
703h	1	05	100	6970

第二节 服务数据 SDO 数据分析

上一节和大家一起分析了总线中的 PDO 的数据，这边再带大家一起看一下总线中的 SDO 报文， 通常可以通过 SDO 进行一些例如参数的修改，也可以用 SDO 来读取或者控制 PDO。访问者被称作客户 (client)，对象字典被访问且提供所请求服务的 CANopen 设备别称作服务器 (server)。客户的 CAN 报文和服务器的应答 CAN 报文总是包含 8 字节数据（尽管不是所有的数据字节都一定有意义），并且一个客户的请求一定有来自服务器的应答，否则就会有报错。常见 SDO 报文的基本格式如下：

11Bit 标识 + 8 Byte 数据

其中 11bit 的基本分为下面两类：

- 0x600 + Node ID: SDO Rx Client -> Server Request
- 0x580 + Node ID: SDO Tx Client -> Server Response

8 个字节的数据中，左边的第一个 byte 命令字，表示是什么操作，基本有下面几类：

- 0x40: Client -> Server, Upload Request 上传请求（读请求）
- 0x4x: Client -> Server, Upload Response 上传应答（读应答）

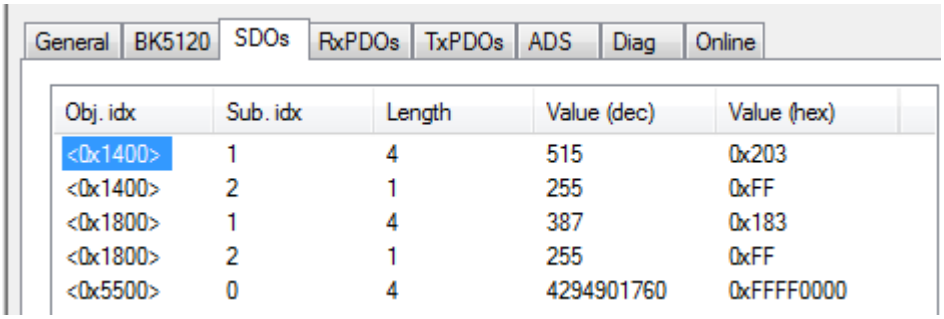
- 0x2x: Client -> Server, Download Request 下载请求（写请求）
- 0x60: Client -> Server, Download Response 下载应答（写应答）
- 0x80： Breakdown of Parameter Communication 报错

其中有效数据的长度 0x4x 或者 0x2x 的“X”来决定

数据长度	1字节	2字节	3字节	4字节
0x2x	0x2F	0x2B	0x27	0x23
0x4x	0x4F	0x4B	0x47	0x43

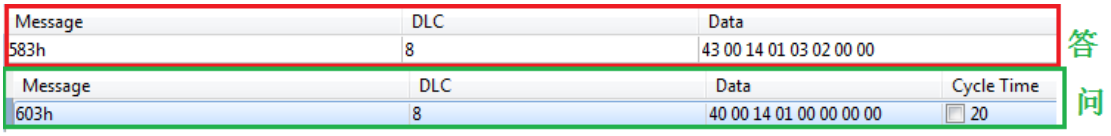
下面以 BK5120 为例，通过 PCAN 去发报文，用 SDO 来读取一些对象字典的数据。

在 BK5120 的 SDO 中有下面的一些内容



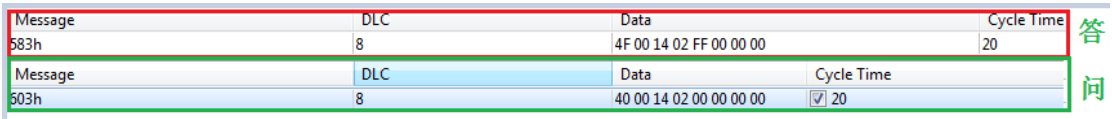
通过 PCAN 发 SDO 的包去读取相应的内容

- 读取对象字典 0x1400 的子索引 0x01 内容，即 RxPDO1 的 COBID 0x203



  - 40 表示是读取
  - 00 14 对应对象字典的 0x1400
  - 01 对应子索引 0x01
  - 00 00 00 00 不使用
  - 43 表示是应答 长度为 8 个字节
  - 00 14 表示对应 0x1400
  - 01 对应子索引 0x01
  - 03 02 00 00 对应 0x203

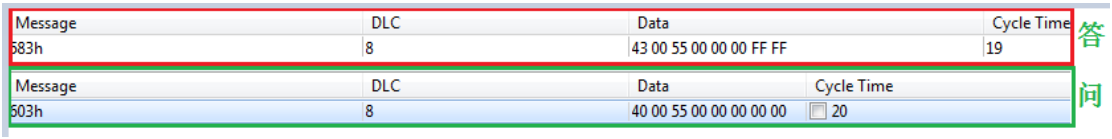
- 读取对象字典 0x1400 的子索引 0x02 内容，即 RxPDO1 的传输率



  - 40 表示是读取
  - 00 14 对应对象字典的 0x1400
  - 02 对应子索引 0x02
  - 00 00 00 00 不使用
  - 4F 表示是应答 长度为 1 个字节
  - 00 14 表示对应 0x1400
  - 02 对应子索引 0x02

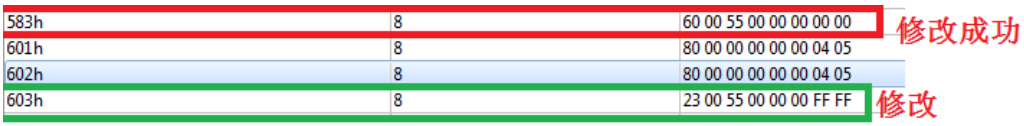
- FF 00 00 00 对应波特率 255.

- 读取 0x5500 的值

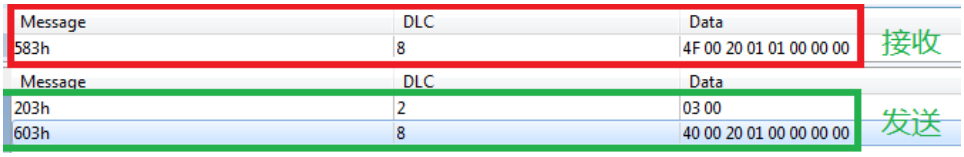


  - 40 表示是读取
  - 00 55 对应对象字典的 0x5500
  - 00 无子索引
  - 00 00 00 00 不使用
  - 43 表示是应答 长度为 8 个字节
  - 00 55 表示对应 0x5500
  - 00 无子索引
  - 00 00 FF FF 对应修改的值 0xFFFF0000（释放 4 对以上的过程数据，见第二章第二节的补充部分）.

并且倍福的主站在带 BK5120 这样的的从站的时候在一开始添加从站设备的时候就会将这个对象字典修改，所以第三方从站如果在配置的时候建议也修改。

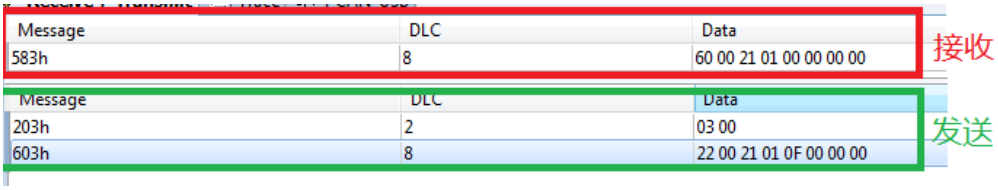


- 通过 SDO 读取数字量的输入值



  - 40 表示是读取
  - 00 20 对应对象字典 0x2000，即数字量输入
  - 01 对应 0x2000 的子索引 0x01
  - 00 00 00 00 读取的时候这边的 data 值默认值
  - 4F 表示是读取成功，返回一个字节
  - 00 20 对应 0x2000
  - 01 对应 0x2000 的子索引 0x01
  - 01 00 00 00 对应最低字节的值为 1，即数字量输入的第一个 bit 为 1

- 通过 SDO 修改数字量输出的值



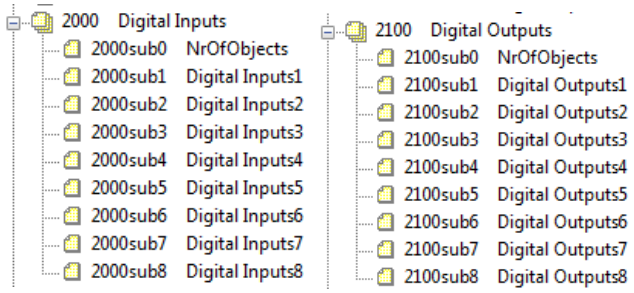
  - 22 表示是修改/写
  - 00 21 对应 0x2100 即数字量输出

- 01 对应 subIndex 0x01
- 0F 00 00 00 对应最低位开始 4 个 bit 均置 1
- 60 表示写成功
- 00 21 对应 0x2100
- 01 对应 subIndex 0x01
- 00 00 00 00 表示写值成功后的缺省值

下图为倍福耦合器的对象列表中对应的输入输出的位置。注意 subIndex0 为数据的长度，实际的值对应 subIndex1，

Index	Meaning
0x2000	Digital inputs (function identical to object 0x6000)
0x2100	Digital outputs (function identical to object 0x6100)
0x2200	1-byte special terminals, inputs (at present no terminals corresponding to this type are included in the product range)
0x2300	1-byte special terminals, outputs (at present no terminals corresponding to this type are included in the product range)
0x2400	2-byte special terminals, inputs (at present no terminals corresponding to this type are included in the product range)
0x2500	2-byte special terminals, outputs (at present no terminals corresponding to this type are included in the product range)
0x2E00	7-byte special terminals, inputs (at present no terminals corresponding to this type are included in the product range)
0x2F00	7-byte special terminals, outputs (at present no terminals corresponding to this type are included in the product range)

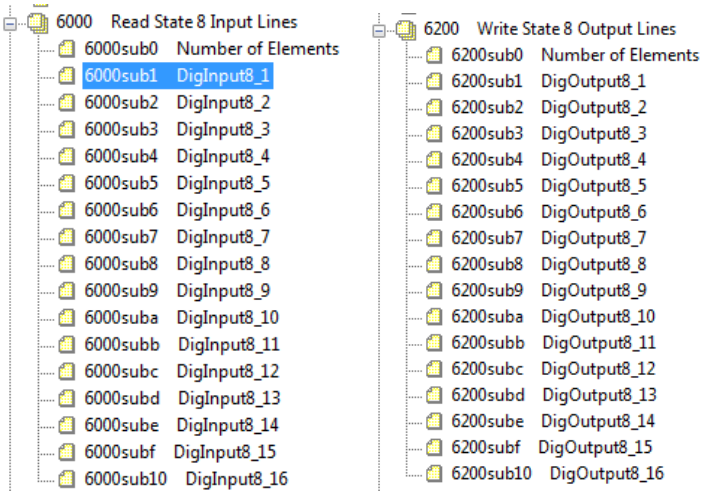
对象字典中的说明如下：



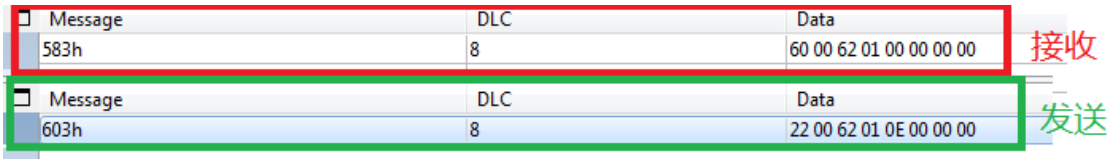
不过输入和输出也可以对应对象字典中 0x6000 之后的对象。同样 subIndex0 通常对应均为数据长度，subIndex1 才为实际对应的数据。

Parameters	Object directory address
Number of digital input bytes	Index 0x6000, sub-index 0
Number of digital output bytes	Index 0x6200, sub-index 0
Number of analog inputs	Index 0x6401, sub-index 0
Number of analog outputs	Index 0x6411, sub-index 0

对象字典 0x6000 和 0x6200 中的描述如下

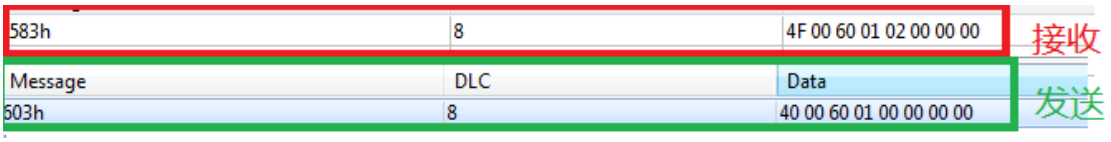


- 所以上面的通过 SDO 修改数字量输出的值也可以通过写 0x6200 的 subIndex1



Message	DLC	Data	
583h	8	60 00 62 01 00 00 00 00	接收
Message	DLC	Data	
603h	8	22 00 62 01 0E 00 00 00	发送

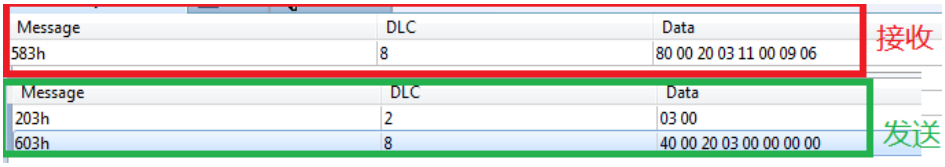
- 同样通过 SDO 读取数字量的输入的值也可以通过读 0x6000 的 subIndex1



583h	8	4F 00 60 01 02 00 00 00	接收
Message	DLC	Data	
603h	8	40 00 60 01 00 00 00 00	发送

第三节 SDO 通讯报错和报警数据（应急报文）分析

上一节和大家进行了 SDO 的通讯的操作，想必在操作的过程中也会碰到一些错误，下面我们和大家一起来分析一下错误代码。如下面的例子中通过 SDO 来读取 0x2000 的 subIndex3 的值，但是其实 subIndex3 并不存在



Message	DLC	Data	
583h	8	80 00 20 03 11 00 09 06	接收
Message	DLC	Data	
203h	2	03 00	发送
603h	8	40 00 20 03 00 00 00 00	

基本的错误的格式请参考下面的内容

11 bit identifier	8 byte user data							
0x580 (client) or 0x600(server) + node ID	0x80	Index0	Index1	SubIdx	Error0	Error1	Error2	Error3

Parameter	Explanation
Index0	Index low byte (Unsigned16, LSB)
Index1	Index high byte (Unsigned16, MSB)
SubIdx	Sub-index (Unsigned8)
Error0	SDO error code low low byte (LLSB)
Error3	SDO error code high high byte (MMSB)

80 00 20 03 11 00 09 06

- 0x80 表示这是 SDO 的报错
- 00 20 对应 0x2000
- 03 对应 subIndex 0x03
- 11 00 09 06 对应错误代码 0x06 09 00 11 即子索引不存在

对应的错误代码在 BECKHOFF information system 中也有说明

SDO error code	Explanation
0x06 02 00 11	Invalid table: Table or channel not present
0x06 02 00 10	Invalid register: table not present
0x06 01 00 22	Write protection still set
0x06 07 00 43	Incorrect number of function arguments
0x06 01 00 21	Function still active, try again later
0x05 04 00 40	General routing error
0x06 06 00 21	Error accessing BC table
0x06 09 00 10	General error communicating with terminal
0x05 04 00 47	Time-out communicating with terminal

SDO error code	Explanation
0x05 03 00 00	Toggle bit not changed
0x05 04 00 01	SDO command specifier invalid or unknown
0x06 01 00 00	Access to this object is not supported
0x06 01 00 02	Attempt to write to a Read_Only parameter
0x06 02 00 00	The object is not found in the object directory
0x06 04 00 41	The object cannot be mapped into the PDO
0x06 04 00 42	The number and/or length of mapped objects would exceed the PDO length
0x06 04 00 43	General parameter incompatibility
0x06 04 00 47	General internal error in device
0x06 06 00 00	Access interrupted due to hardware error
0x06 07 00 10	Data type or parameter length do not agree or are unknown
0x06 07 00 12	Data type does not agree, parameter length too great
0x06 07 00 13	Data type does not agree, parameter length too short
0x06 09 00 11	Sub-index not present
0x06 09 00 30	General value range error
0x06 09 00 31	Value range error: parameter value too great
0x06 09 00 32	Value range error: parameter value too small
0x06 0A 00 23	Resource not available
0x08 00 00 21	Access not possible due to local application
0x08 00 00 22	Access not possible due to current device status

紧接着我们看看如果总线数据有错即命令字是 0x80 这样的应急报文怎么分析，应急报文由设备内部出现的致命错误触发，由应用设备已最高优先级发送的数据。

一个应急报文也通常由 8byte 数据组成：

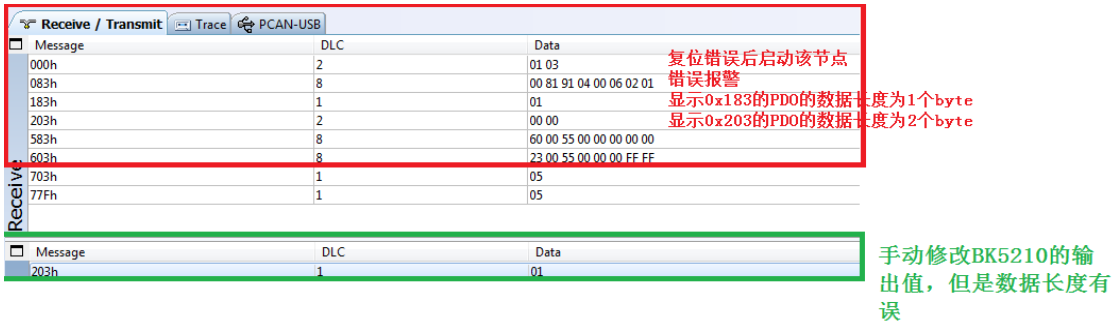
COBID（0x80+NodeID）+ 8 Byte 错误代码

Structure of the emergency message

The emergency object is always 8 bytes long; it contains first the 2-byte error code, then the 1-byte error register, and finally the additional code of 5 bytes. This is divided into a 2-byte bit field and a 3-byte parameter field:

11 bit identifier	8 bytes of user data							
0x80 (=128dec) + node-ID	EC0	EC1	EReg	Bit field 0: Comm	Bit field 1: DevErr	EMCY Trigger	Info 0	Info 1

例如基于上面 SDO 的硬件 BK5120，如果我通过发 PDO 的方式给 0x203 的输出赋值，但是由于赋值的数据格式出错（实际 PDO 长度为 2 个字节，我发送的是一个字节），就会触发硬件的错误报警。



发送→ 0x203 01

(给 0x203 这个 PDO 的第一个字节的第一位发 1)

接收到报警信息为 ←00 81 91 04 00 06 02 01

00 81 91： 前 3 个 bye 说明这是一个通讯错误。

04 00 06： CAN 的错误计数器超出警报范围

02:　　总线中该 PDO 实际的参数长度

01：　　设置的错误长度

Parameters	Explanation	
EC0	Error Code Low-Byte. Not used (always zero)	
EC1	Error Code High-Byte. 0x50 = device error, 0x81 = communication error, 0x00 = error reset	
EReg	Error register. 0x81 = device error, 0x91 = communication error	
Bit field 0: Comm	Bit field communication error:	
	0x01	Guarding delayed or failed
	0x02	Sync delayed or failed
	0x04	Incorrect PDO length parameterized
	0x08	Event timer timeout: RxPDO not received in time
	0x10	Receive queue overrun
	0x20	Transmit queue overrun
	0x40	CAN bus OFF
	0x80	CAN warning limit exceeded
Bit field 1: DevErr	Bit field device error:	
	0x01	Terminal error
	0x02	K-Bus error / IP-Link error
	0x03	-
	0x04	EEPROM error
	0x10	Unsupported terminal plugged in (BK5110, LC5100)
	0x80	Altered HW configuration.
EMCY trigger	The *emergency trigger* byte contains the code for the particular error that has triggered the emergency telegram. If an error has been rectified, an emergency telegram with the error code 0x0000 is sent, and the emergency trigger contains the description of the error that has been corrected. Errors that are still current are signaled here in the bit fields. Once the Bus Coupler is free of errors, it sends an emergency telegram containing zeros everywhere other than in the emergency trigger.	
	0x01	CAN warning limit exceeded (too many error frames)
	0x02	CAN bus OFF state has been reached. Since the coupler can no longer send an emergency telegram, an emergency telegram with trigger 0x40 is sent when the bus leaves the "off" state (a new CAN controller initialization).
	0x03	Transmit queue overrun: CAN messages are being lost
	0x04	Transmit queue overrun: CAN messages are being lost
	0x06	Incorrect PDO length parameterized (check mapping). Info 0: parameterized (expected) PDO length in bytes Info 1: current PDO length (results from the added lengths of the mapped objects)

同样的例子如果发送 0x203 03 00 00 00 00 00 00 00(发送 8 个 byte 的数据给 0x203)也会有错误报警。其中最后的 info1 就说我实际发送了 8 个字节的数据，其实该 PDO 只有 2 个 byte。

083h	8	00 81 91 04 00 06 02 08
Message	DLC	Data
203h	8	03 00 00 00 00 00 00 00

注意：由于各个厂家的报警处理和报文可能有所区别，所以第三方的错误报警请仔细参考第三方的产品说明书，倍福的则可在 BECKHOFF information system 中查找该硬件部分的 Emergency Object.

第四章 CAN 总线应用汇总

上两章通过一些基本应用解释了 CAN 总线的一些常用的术语和一些总线的知识，接着本章节将重点基于不同的应用解决方案的探讨，经过近些年的一些应用，感觉 CAN 总线的解决方案其实有很多，并且没有绝对的方法，但是基本可以分成下面几大类：

- 导入 CANOpen 设备提供 EDS 文件，便可直接进行 PDO 通讯。
- 导入 CANOpen 设备提供 EDS 文件，需重新修改 SDO 对应的 PDO 映射
- 不导入 EDS 文件，采用 general CAN Node 的方式通讯
- 不导入 EDS 文件，采用 CAN Interface 通讯
- 第三方主站设备，只能通过 EDS 文件配置，必须修改 EDS 文件

第一节 无需任何配置的 CANOpen 通讯的场合

这种情况是 CAN 总线中最简单的一个应用情况，例如倍福的从站，无论是 BK/BC/BX 设备，还是 EL6751-0010 均为该种方式，从站均可以通过扫描的方式被主站识别和配置（参考上一章的第二节的内容）。当然也有不少的第三方设备也可以这样，只需第三方提供设备的 EDS 文件，在 EDS 中已经完整的配置好所有的 PDO 的映射关系。由于这种应用基本和端子模块的使用方法类似，这边就不多加说明。需要注意的是线缆和波特率的一些设置，请务必参考设备说明书的波特率去设置，过长的线缆请一定程度上降低波特率。

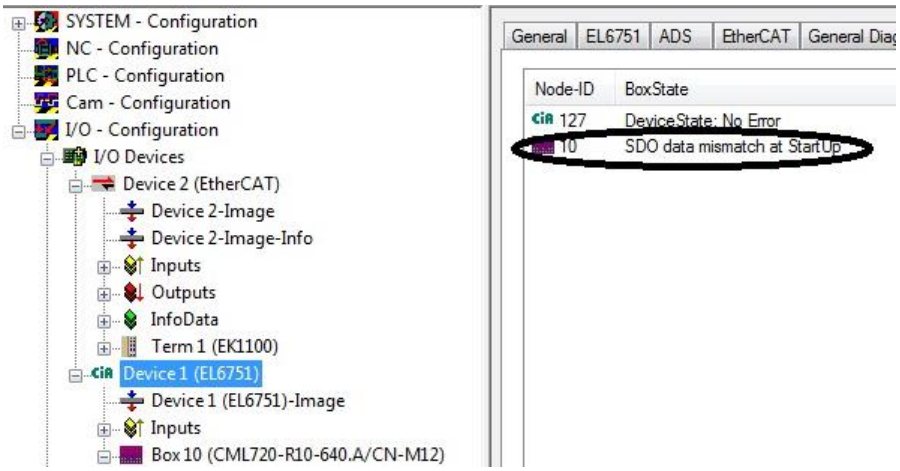
第二节 使用 EDS 文件，但 PDO 还需额外配置的场合

这种情况在 CAN 总线中，也比较常见，在上一章的第三节已经举过一个例子，基本的添加的步骤也雷同，需要注意的就是该操作需要参考第三方的使用说明书或者得到第三方的技术人员确认，因为不是所有的第三方设备都支持手动修改 PDO 的映射，并且以何种方式组合等一些问题都需有技术人员确认。

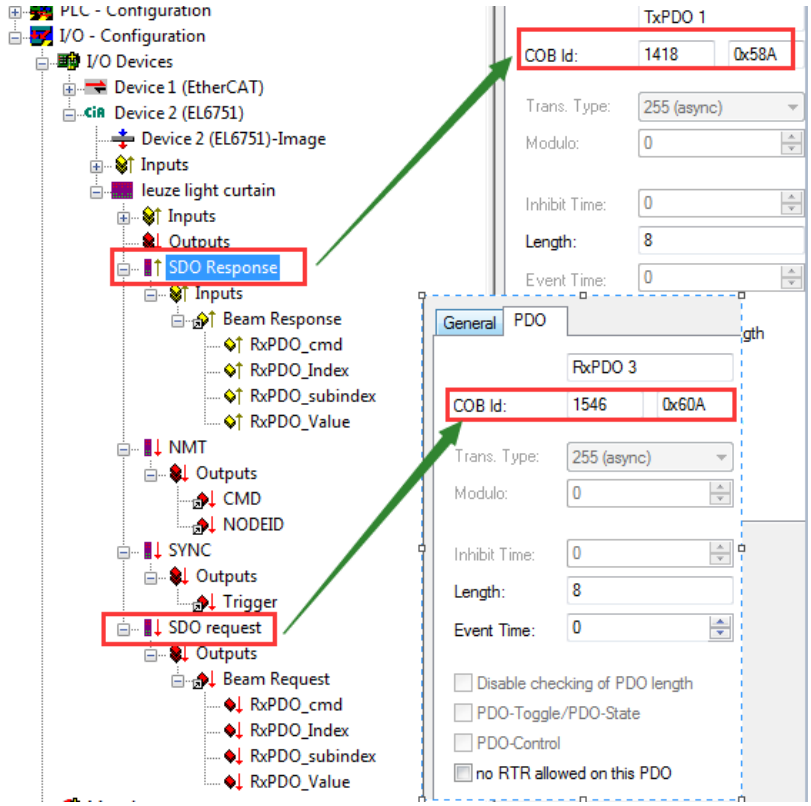
第三节 通过通用 CAN 节点的方式通讯的场合

这种方式在更多情况下应该理解成 CAN2.0 的通讯，无需 EDS，无需映射 PDO，只需要知道需要读取 PDO 有那些即可，在很多应用下均可行。例如有的时候一些设备没有很好的第三方技术支持，或者设备不是严格的遵循 CAN 总线的规范，但是 CAN 总线数据可以通过抓包获得，并且数据也比较简单和规律可以进行分析和归纳。还有一种情况也适用，例如当用 CANOpen 通讯的时候 SDO 映射 PDO 的时候总是报错，也可以用这种方式尝试。下面将用两个例子来说明这种应用方式。

1) 例如早期的一次和劳易测 Leuze 的光栅尺 CANOpen 通讯，初期按照他们的配置说明始终会报 SDO data mismatch at startup 的报错。

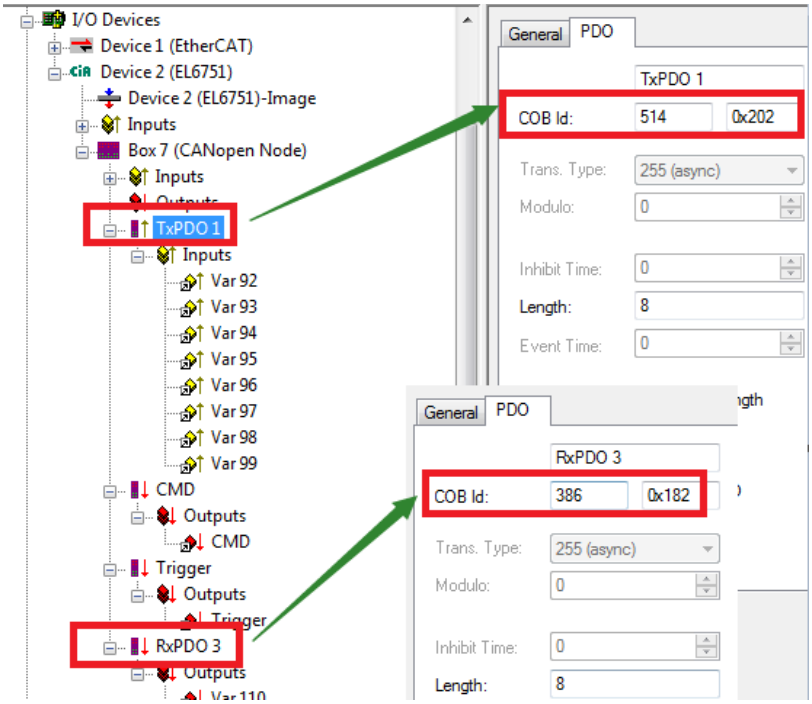


后来项目初期采用通用 CAN 节点的方式，通过 SDO 来读取相应的过程数据便没有问题（见下图）

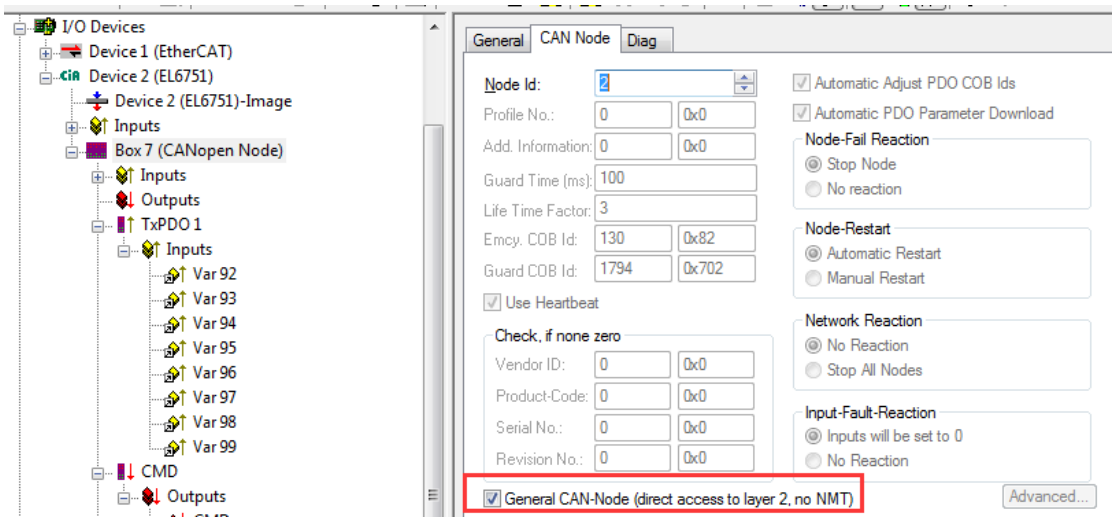


这样操作的原因初期的考虑是因为用抓包工具发 SDO 的报文，设备可以应答的，但是由于后来客户觉得用 SDO 的这种方式不是实时的过程数据，所以最后还是采用了 CANOpen 的方式，并发现之前 CANOpen 一直报错的原因也是由于硬件的故障。

2) 另一次应用，例如柯蒂斯 curtis 的驱动器，发现客户的 TxPDO 和 RxPDO 的 COBID 都是反的，TxPDO 应该 180+NodeID 的，变成 200+NodeID 了，对于这种非标准的 COBID 的类型一般也建议通过 CAN Node 的方式通讯，通过发启动命令和同步帧来启停通讯（参考第二章的第四节），过程数据可以根据客户的自定义来安排。



- 首先注意在使用通用 CAN 节点的这种情况下，一般都需要用 NMT 和 SYNC 同步帧。
- 由于应用的要求 TxPDO 和 RxPDO 的 COBID 有特别的要求，所以此时需忽略 CAN 的标准，如上图，TxPDO1 的 COBID 为 0x202，RxPDO3 为 0x182。
- 但是请务必勾选 General-Node（direkt access to layer 2, No NMT）



不过就如本节开始提过，此种解决方案适用于数据有规律可寻，也没有时序上的要求，例如仅对某些特定 PDO 进行周期性的监控。

第四节 CAN Interface 的应用场合

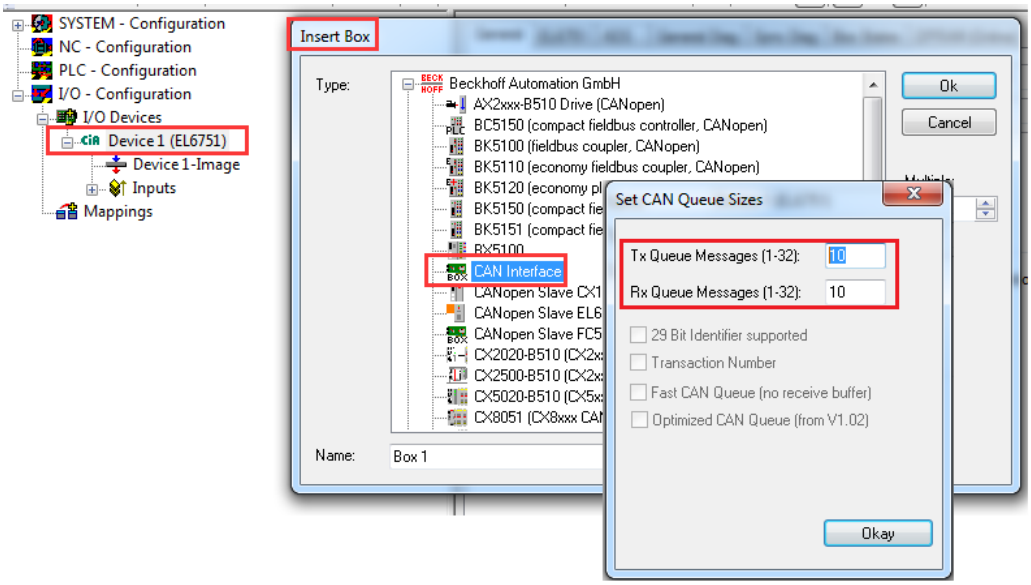
个人认为 CAN Interface 是 CAN 总线通讯的终极方案，这种方案可以解决所有问题，不过也是最复杂的一种方案，因为所有的内容和时序的操作均需通过 PLC 程序来实现，简单的理解这就是一个总线抓包器，总线上的数据需要自己通过 PLC 程序来过滤和筛选，然后对应到相

应的 PDO 中去。不过如果对于通讯和数据的获取均通过不同的 PDO 来获得，例如要求先发什么 PDO 什么数据，接着发什么 PDO 的数据才能读取什么参数，这类还是建议用 CAN Interface 来配置。下面以 HAWK 的比例阀的应用来说明。HAWK 的比例阀需要走 CAN2.0 的方式，并且目前只能用 CAN Interface 的方式，因为他们有一些时序上的要求

1) 只发一次 0x70x→类似 NMT 的东西
2) 发 COBID：0x00 data：0x01 0x00→ 类似 NMT 的启动报文，启动所有结点
3) 接着每 100ms 读 COBID：0x18X 的内容。如果是 08 00 00 00 其中 00 80 是未工作的状态
4) 接着可以发第一次 COBID：0x20X，内容：0F 00 00 00
5) 设备如果正常的话应该返回 0x18X：0F 00 00 00，这说明状态对了
6) 接着可以继续发第二次的 0x20X：0F 00 xx xx，例如某个设定点-1000~1000 之间。
7) 如果设定正确应该是 0x18X：0F 00 xx xx 说明 OK 了
8) 如果出现错误（一级错误）：会返回 0x18X：09 00 xx xx
9) 这样需要再发一次 0x20X： 0F 00 00 00 直到设备返回的也是 0x18X：0F 00 00 00，才说明状态对
10) 如果出现错误（二级错误）：会返回 0x18X：01 00 xx xx
11) 这样需要首先发：0x20X： 0D 00 00 00 让设备返回 0x18X：09 00 00 00，然后再发 0x20X： 0F 00 00 00 直到设备返回的也是 0x18X：0F 00 00 00。也就是需要从二级的错误先回到一级然后再回复状态！

不过在这边我们将不会将如此复杂的逻辑在这边说明，仅将 CAN Interface 在使用时候 PLC 逻辑的大的框架和需要注意的地方和大家进行说明。

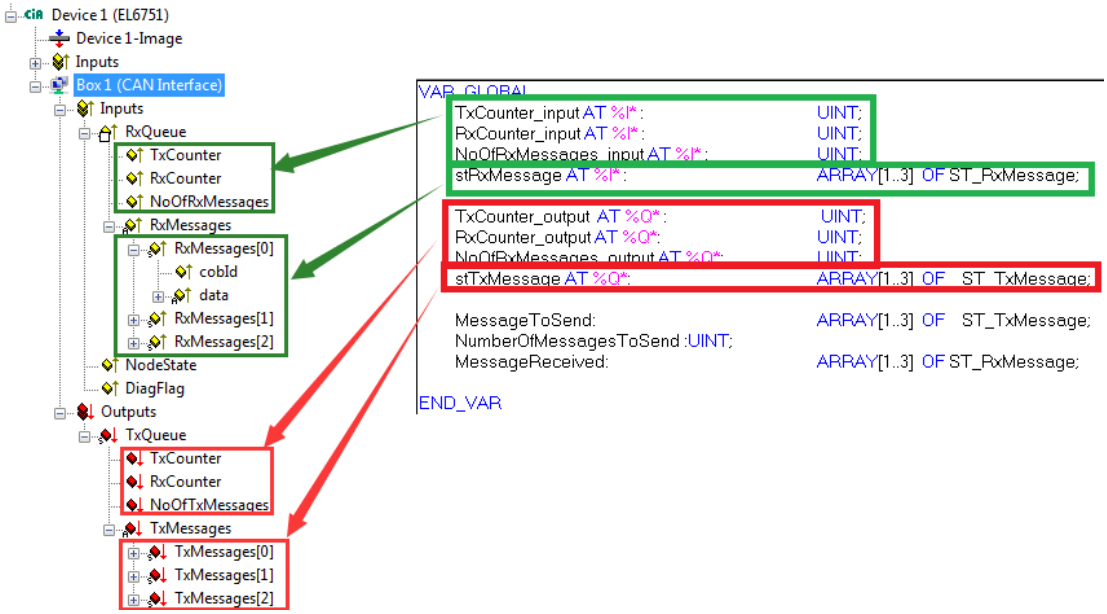
- 首先 CAN Interface 使用的时候在选择 IO 的时候就需要添加 CAN Interface，并选择合适的 Message Queue。



一般 Message Queue 的选择可以根据实际的要求，做不同情况的配合，例如如果有 3 个站一共有 10 个 PDO，可以用 10 个 Message Queue 来分别抓取 10 个 PDO 的信息，也可以只用 3 个 Message Queue 的方式以从站为基础来轮训获取信息，因为对于从站的信息的获取其实都是通过 PLC 程序的逻辑编写来实现。

- 接着在 PLC 中定义变量，首先需要的 TxCounter 的输入输出变量，RxCounter 的输入输出变量及 NoOfTxMessages 输出和 NoOfRxMessages 的输入及对应 MessageQueue 类型的

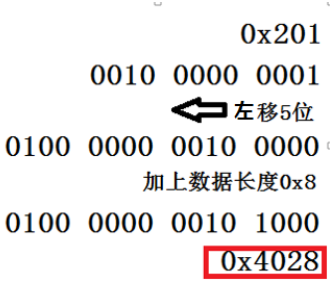
数组。因为这些 PLC 变量均需要与 CAN Interface 对应的做变量映射。



- 然后就是程序部分，当然程序部分的内容也可以依据实际需求有所变化，这里仅供参考。
  - 首先是初始化过程，例如这里将需要读取的从站的 COBID 准备好

```
IF NOT CfgFlag THEN（*CfgFlag 仅执行一次*）
    FOR i:=1 TO NoOfRxMessages_output DO
        MessageToSend [i].CobId:= xx;
    END_FOR
CfgFlag:=TRUE;
END_IF
```

注意 COBID 这边有些变化，不单纯是 CAN 标准的 COBID 前 4 位功能码后 7 位节点号，而是将标准的 COBID 按位左移 5 位，并增加了传输字节数信息，例如如果是传输 8 个字节即 0x8（参考下图例子中的转换关系）.



  - 接着就是更新发送区，因为 TxConter_Input 是反聩回来的已发送的记数，如果 input 和 output 一样，说明发送过了，因为从站已经收到发回的反馈信息，这时如果加 1，说明现在有新的数据了，会触发发送一次。

```
IF TxCounter_output= TxCounter_input    THEN
    NoOfRxMessages_output := NumberOfMessagesToSend;
    TxCounter_output := TxCounter_output + 1;
END_IF
```

注意：在 CAN Interface 中 TxPDO 即是输出，是要 transmit 给主站的；RxPDO 即是收到的，是从主站发出，从站收到的。这个和之前的几个应用 CANOpen 或者 general CAN Node 都有区别，注意一下去理解，不过其实这个从站的角度去理解比较容易，

一个站在主站的角度，一个是站在从站的角度。

- 同样对于接收区则需要首先去筛选你要读取的从站的 COBID 和比对是否数据是你需要的即可。可以参考下面的框架

```
IF RxCounter_output<> RxCounter_input THEN
    FOR j := 1 TO NoOfRxMessages_input    DO
        IF stRxMessage[j].CobId=16#4048 THEN
            MessageReceived[1].CobId:= stRxMessage[j].CobId;
            MessageReceived[1].RxMessage:=stRxMessage[j].RxMessage;
        END_IF
    END_FOR
RxCounter_output:= RxCounter_output+1;
END_IF
```

- 不过注意 Rx 和 TX 的 Messages Queue 的大小只跟 TX 数量有关，而与 RX 数据无关。如果发送的数据大于传输的数据，建议把 Tx 设置大点，浪费点就浪费点，并不影响使用。
- 另外 CAN Interface 和 General CAN Node 和 CANOpen 在 system manager 中的配置还是有区别的：一个主站设备中 CANOpen 和 General CAN Node 可以并存，但是一个主站只能有一个 CAN Interface 接口。
- 对于 CAN2.0B- 29bit 这种类型的 CAN 总线通讯只能用 CAN Interface 进行通讯，在 system manager 中看到一个 filter，这个 filter 其实是关闭的

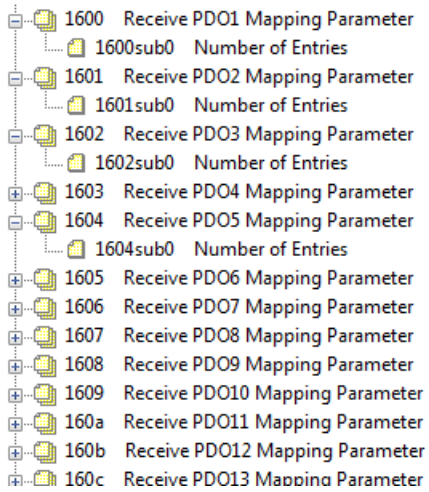
如果需要支持 CAN2.0B，则需注意 COBID 的最高位 MSB（bit31）置“1”，

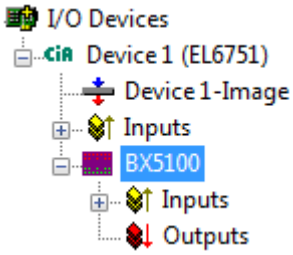
```
Message-Struktur with 29-Bit
-Length (0..8)
    - CobId
        Bit 0-28: 29 Bit-Identifier
        Bit 30: RTR
        Bit 31: 0: normal Message (11 Bit Identifier), 1: extended Message (29 Bit-
Identifier)
- Data[8]
```

第五节 需要修改 EDS 文件的场合

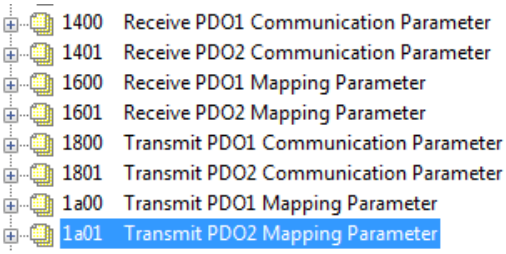
之前四节内容是针对于倍福 CAN 主站的应用，但是与第三方主站读取通讯的时候，第三方主站常要求导入 EDS，并且无法手动配置 PDO，倍福从站 EDS 文件并不描述 PDO 的映射关系，所以只能手动修改 EDS 文件。下面通过一个范例来简要的说明倍福 CAN 从站 EDS 文件修改的方式。例子中将手动修改 BX5100 的 EDS 文件，然后基于修改好的 EDS 文件，用 EL6751 来做主站。

1) BX5100 是倍福一款 CAN 总线的硬 PLC，本身可以独立编程和控制少量 IO，作为 CAN 总线从站时，总线部分 PDO 的 COBID 及 PDO 的通讯方式虽然已经在 BX5100 的 0x14xx 和 0x18xx 中定义好了，在这边对于这方面无须再做配置，但是从官网上下载 BX5100 的 EDS 文件，打开后可以看到 EDS 文件中默认预定义了 32 对 PDO，但是 PDO 中（0x16xx 和 0x1Axx）没有映射任何变量。如果直接在 EL6751 中添加 BX5100 并导入 EDS 文件，可以看到没有任何过程数据变量，所以必须手动添加。

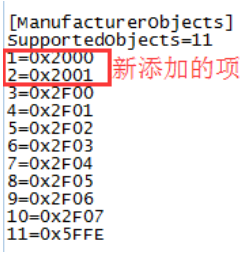




- 在本次举例中将只用到 TxPDO1，TxPDO2 和 RxPDO1， RxPDO2，所以首先手动删除 BX5100 里面 0x16xx 和 0x1Axx 中标配的剩下的 30 对 PDOs，为了清晰起见也可以删除 0x14xx 和 0x18xx 里面多余的关于 PDO 的通讯参数的对象。最后只在 EDS 文件中保留下图中的这些内容。



- 接着在对象列表中添加 0x2000 的对象列表和自定义的一些变量，从 0x2000~0x2F00 这段对是可以自由定义变量，我们首先添加两组新项，分别用作输入变量和输出变量。



➢ 0x2000 中自定义 4 个输出变量，均为 1 个字节的数据

```
[2000]
SubNumber=5
ParameterName=Self defined Output PDO
ObjectType=0x8

[2000sub0]
ParameterName=NrOfObjects
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=4
PDOMapping=1

[2000sub1]
ParameterName=Self definded Out_Var1
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=1
PDOMapping=1
```

一共有5个子索引

4个数据

数据1的名字

数据类型usint

```
[2000sub2]
ParameterName=Self definded Out_Var2
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=2
PDOMapping=1

[2000sub3]
ParameterName=Self definded Out_Var3
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=3
PDOMapping=1

[2000sub4]
ParameterName=Self definded Out_Var4
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=4
PDOMapping=1
```

数据2的名字

➢ 0x2001 中自定义 4 个输入变量，也为一个字节的数据。

```
[2001]
SubNumber=5
ParameterName=Self defined Input PDO2
ObjectType=0x8

[2001sub0]
ParameterName=NrOfObjects
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=4
PDOMapping=1

[2001sub1]
ParameterName=Self definded In_Var1
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=1
PDOMapping=1
```

一共5个子索引

包含4个数据

第一个数据

数据类型usint

```
[2001sub2]
ParameterName=Self definded In_Var2
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=2
PDOMapping=1

[2001sub3]
ParameterName=Self definded In_Var3
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=1
PDOMapping=1

[2001sub4]
ParameterName=Self definded In_Var4
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=2
PDOMapping=1
```

- 最后就是在 0x1600 和 0x1A00 中映射上面自定义的变量即可

```
[1600]
ParameterName=Receive PDO1 Mapping Parameter
ObjectType=0x9
SubNumber=3

[1600sub0]
ParameterName=Number of Entries
ObjectType=0x7
DataType=0x0005
AccessType=rol
Defaultvalue=2
PDOMapping=0
LowLimit=0
HighLimit=8

[1600sub1]
ParameterName=PDO mapping var1
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20000108
PDOMapping=1
LowLimit=0
HighLimit=8

[1600sub2]
ParameterName=PDO mapping var2
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20000208
PDOMapping=1
LowLimit=0
HighLimit=8
```

一共三个索引

两个映射的变量

映射对应0x2000：01的数据量为一个字节的变量

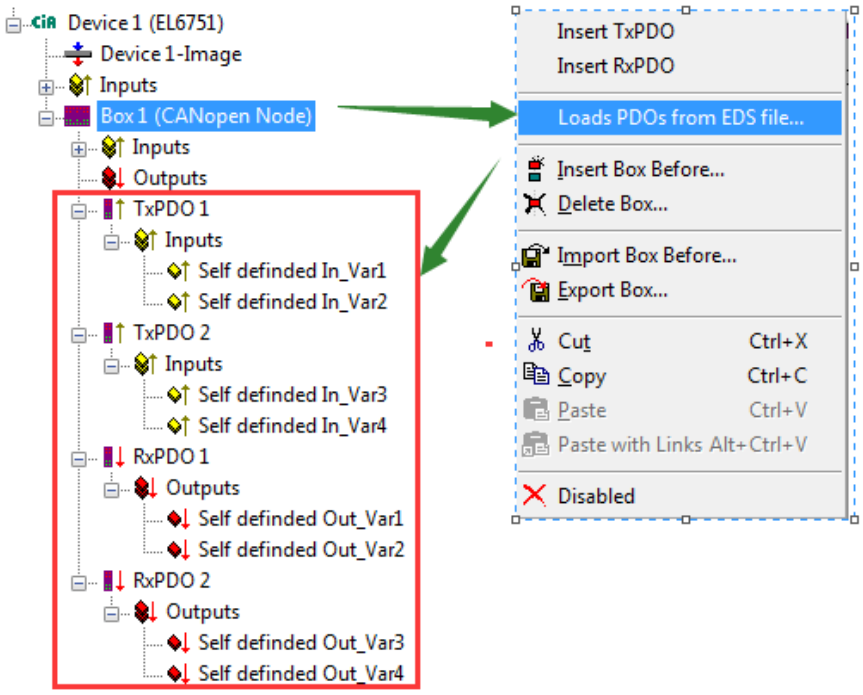
映射对应0x2000:02的数据量为一个字节的变量

```
[1601]
ParameterName=Receive PDO2 Mapping Parameter
ObjectType=0x9
SubNumber=3

[1601sub0]
ParameterName=Number of Entries
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=2
PDOMapping=0
LowLimit=0
HighLimit=8

[1601sub1]
ParameterName=PDO mapping var3
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20000308
PDOMapping=1
LowLimit=0
HighLimit=8

[1601sub2]
ParameterName=PDO mapping var4
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20000408
PDOMapping=1
LowLimit=0
HighLimit=8
```

```
[1a00]
ParameterName=Transmit PDO1 Mapping Parameter
ObjectType=0x9
SubNumber=3                    该PDO有3个索引

[1a00sub0]
ParameterName=Number of Entries
ObjectType=0x7
DataType=0x0005                一共映射2个变量
AccessType=ro
DefaultValue=2
PDOMapping=0
LowLimit=0
HighLimit=8

[1a00sub1]
ParameterName=PDO mapping var1
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20010108        对应0x2001:01的变量，类型为1个字节
PDOMapping=1
LowLimit=0
HighLimit=8

[1a00sub2]
ParameterName=PDO mapping var2
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20010208        对应0x2001:02的变量，类型为1个字节
PDOMapping=1
LowLimit=0
HighLimit=8
```

```
[1a01]
ParameterName=Transmit PDO2 Mapping Parameter
ObjectType=0x9
SubNumber=3                    该PDO有3个索引

[1a01sub0]
ParameterName=Number of Entries
ObjectType=0x7
DataType=0x0005                一共映射2个变量
AccessType=ro
DefaultValue=2
PDOMapping=0
LowLimit=0
HighLimit=8

[1a01sub1]
ParameterName=PDO mapping var1
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20010308        对应0x2001:03的变量，类型为1个字节
PDOMapping=1
LowLimit=0
HighLimit=8

[1a01sub2]
ParameterName=PDO mapping var2
ObjectType=0x7
DataType=0x0007
AccessType=rw
DefaultValue=0x20010408        对应0x2001:04的变量，类型为1个字节
PDOMapping=1
LowLimit=0
HighLimit=8
```

- 测试一下在 TwinCAT system manager 中添加 El6751 的主站，然后添加 general CAN Node，然后鼠标右键选择 load PDOS from EDS file，可以看到在 EDS 文件中添加的变量已经配好。

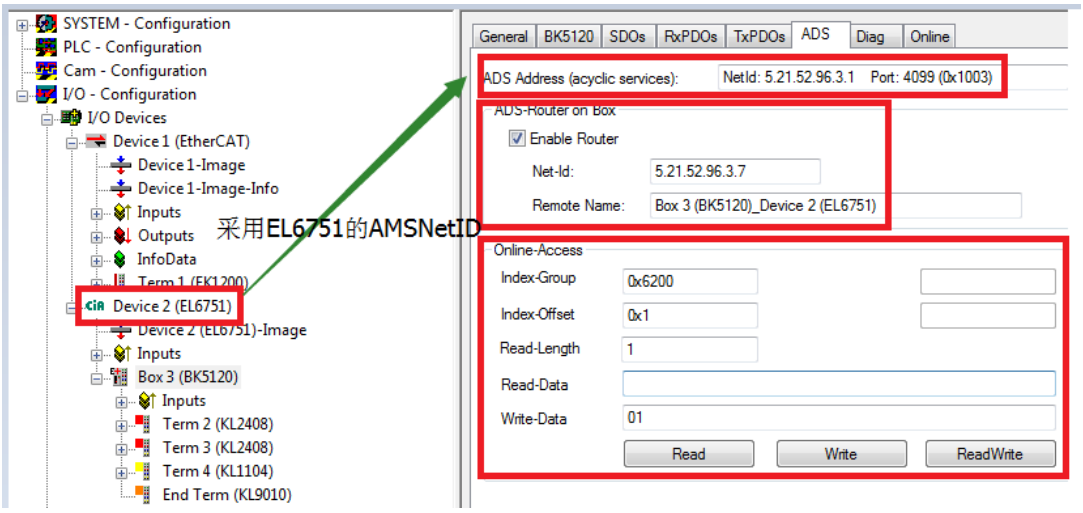


第五章 倍福基于 CAN 设备的 ADS 应用举例

ADS 是倍福的基础，基本所有软件硬件，应用层或者内核层，均直接或者间接的通过该方式进行数据交换，同样倍福的 CAN 设备也支持 ADS 的访问，下面将一些常见的应用总结如下。

第一节 通过 ADS 读取 SDO 对象列表

这个功能在使用 TwinCAT system manager 做配置的时候可以看见。



上面这个就实现了通过 ADS 写 BK5120 的输出（一个数字量的第一个通道）的功能。

倍福 CAN 耦合器上的数字量和模拟量的 IO 分别定义在下图中相应的对象字典中

Parameters	Object directory address
Number of digital input bytes	Index 0x6000, sub-index 0
Number of digital output bytes	Index 0x6200, sub-index 0
Number of analog inputs	Index 0x6401, sub-index 0
Number of analog outputs	Index 0x6411, sub-index 0

Index	Meaning
0x2000	Digital inputs (function identical to object 0x6000)
0x2100	Digital outputs (function identical to object 0x6100)
0x2200	1-byte special terminals, inputs (at present no terminals corresponding to this type are included in the product range)
0x2300	1-byte special terminals, outputs (at present no terminals corresponding to this type are included in the product range)
0x2400	2-byte special terminals, inputs (at present no terminals corresponding to this type are included in the product range)
0x2500	2-byte special terminals, outputs (at present no terminals corresponding to this type are included in the product range)
0x2E00	7-byte special terminals, inputs (at present no terminals corresponding to this type are included in the product range)
0x2F00	7-byte special terminals, outputs (at present no terminals corresponding to this type are included in the product range)

所以上面的例子分析如下：

- 首先 ADS 需要找到连接路由的设备，这边由于是 EL6751 下面带的耦合器 BK5120，所以路由设备的 AMSNetID 采用 EL6751 的，即 5.21.52.96.3.1
- （特定）端口 0x1003.
- Index-Group：对应该对象在对象字典中的索引，例如第一个数字量输出即 0x6200

- Index-Offset：对应该对象在对象字典中的偏移量，第一个数字量的第一个字节即 0x01
- Read/Write Length：读取或者写入的长度。此处为 1 个字节。
- 点击 Write 按钮即可看到 BK5120 的数字量输出的第一个通道的灯亮了。如果有错误即会有错误弹出窗口。实际的错误还请参考 ADS 的相应报错信息。

除了在 system Manger 中的 ADS 读写的调试方式，所有的也可以通过 PLC 程序的方式来实现。通过调用 ADSREAD 和 ADSWRITE 的功能块，加上合适的目标设备和端口，便可以实现对于 SDO 对象字典的自定义读取或者修改，具体的实现这边不做详细介绍。

Input parameters	Description
NETID	AMS NetID of BX or CAN Master
Port number	$0x1000_{hex}$ + NodeId (slave number)
IDXGRP	SDO Index
IDXOFFS	SDO Subindex
LEN	Length of SDO data (1...4)

第二节 通过 ADS 切换从站的状态机

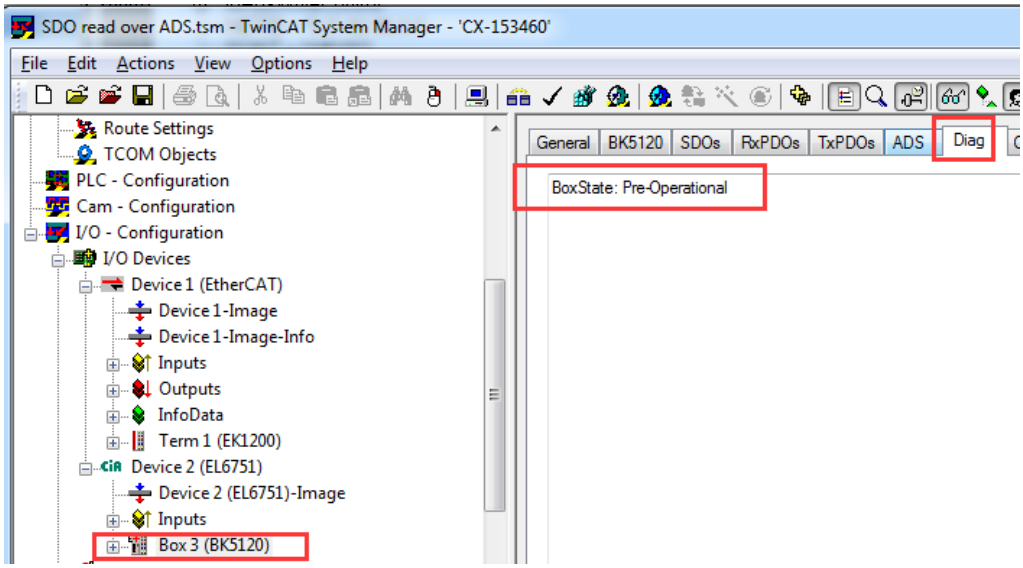
通过 ADS 还可以实现单个或者所有从站的状态机的切换，需要用到 ADSWRTCTL 功能块，参数的描述请参考下面的列表。

Input parameters	Description
NETID	local NetId of the BX, or leave empty, e.g. with ''
Port number	$0x1000_{hex}$ + NodeId (slave number) / 153_{dec} (all nodes)
ADSSTATE	ADSSTATE_RUN
DEVSTATE	1 - Pre / 0 - Operational
LEN	0
SRCADDR	0

下面的程序通过 ADS 修改从站的状态从 OP 到 pre-Op

```
fbAdsWrtCtl(
    NETID:='5.21.52.96.3.1',(*AmsNetID der CAN Master or BX*)
    PORT:=16#1003,  (* Port 1000+ Node:1 *)
    ADSSTATE:=ADSSTATE_RUN,
    DEVSTATE:=0,    (*0-per / 1-Operational *)
    LEN:=0,
    SRCADDR:=0,
    WRITE:=bStartChange,(* start the functionblock *)
    TMOUT:= t#1s );
```

成功后可以看到从站的状态已经成功切换到 Pre-Op 的状态，并且从站 BK5120 的 Run 灯切换为闪烁



此时如果观察总线的报文，可以看到下面的信息：

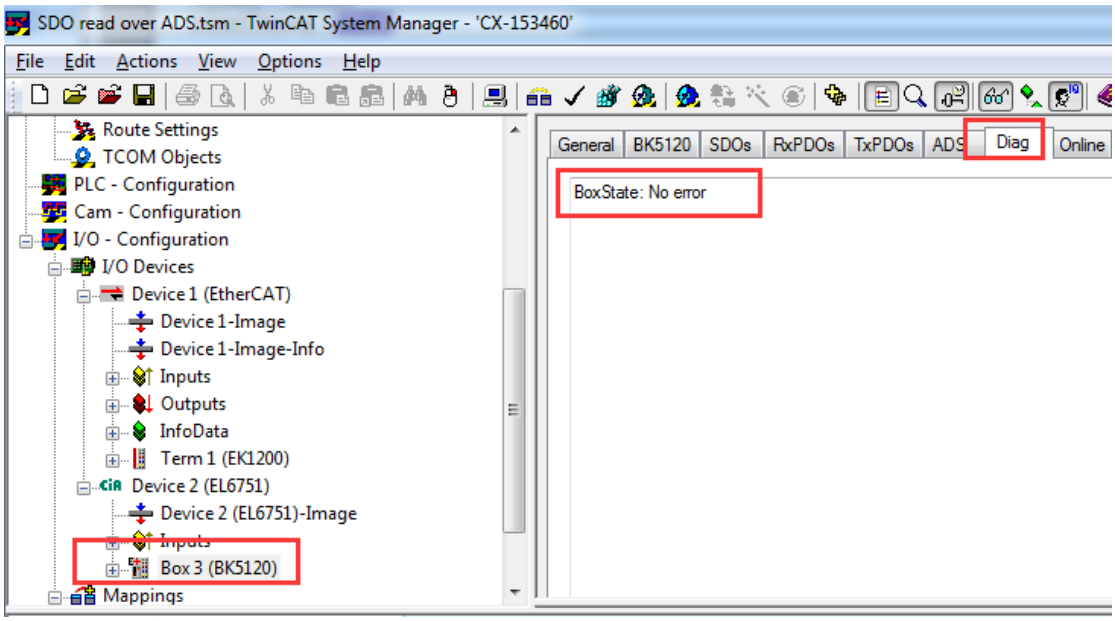
Message	DLC	Data
000h	2	80 03

这个是调用了 NMT 的 enter Pre-Operation state.

命令字	NMT 服务
1	Start Remote Node
2	Stop Remote Node
128	Enter Pre-operational State
129	Reset Node
130	Reset Communication

如果从 Pre-Op 切换到 Op 则其实是调用了 NMT 的 Start Remote Node 功能

000h	2	01 03



第三节 通过 ADS 发送任意 CAN message

倍福的从站 BX 还支持通过 ADS 发送任意 CAN Message

Input parameters	Description
NETID	local NetId of BX
Port number	153
IDXGRP	16#0000F921
IDXOFFS	0
LEN	11 bytes
SRCADDR	Pointer to an 11 byte ARRAY

例如下图中可以通过 ADSWRITE 功能块发送 SDO 来控制 BX5100 上的数字量输出。

Byte	Description	Example Node 3 SDO 0x603 Len 8 Download Request 0x2100 (Index) Sub Index 1 - Value "1"
1	COB-ID LowByte	0x06 (SDO Low Byte)
2	COB-ID HighByte	0x03 (SDO High Byte)
3	LEN (length)	0x08 (LEN, may be 5 in this case)
4	Data[1]	0x22 (Download Request)
5	Data[2]	0x00 (Index Low Byte)
6	Data[3]	0x21 (Index High Byte)
7	Data[4]	0x01 (Sub Index)
8	Data[5]	0x01 (Value "1")
9	Data[6]	0x00
10	Data[7]	0x00
11	Data[8]	0x00