学习笔记

用这个笔记来记录学习ethercat中遇到的问题，与对其了解的过程。

### Ether cat帧结构

#### 1.1以太网帧头

由于每一份资料中提及的内容详细程度不一样，所以，将遇到的相关内容粘贴到这个文档当中。免除了再次查找的时间。

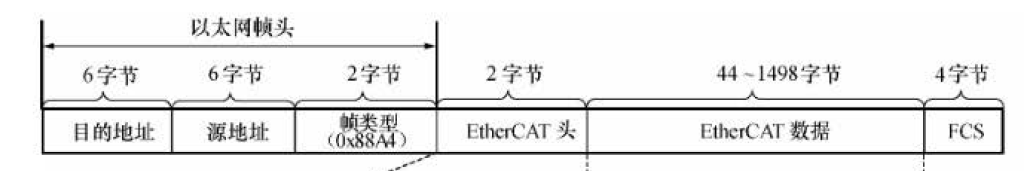


图1.1.

从图1中可以看出来，Ether Cat帧总体上是由以太网帧头，ether cat帧和FCS组成的。

其中：



表1.1.

从表1中可以知道，前面两个地址分别是接收方MAC地址和发送方MAC地址。

这里有个问题，如果是以主站发出数据来说，接受方是从站的MAC地址，源地址是主机MAC地址。那么在接收数据，就是RxPDO的时候，这两个MAC地址的位置是不是调换了？

第一个问题应该是不会交换。

还有，在主站中在文件中配置了主站的MAC地址，是不是在使用主站的接口时不需要填写这个以太网帧头？这个可能需要后面验证一下。如果需要的话，那么伺服驱动器的MAC地址去哪里查找。（插到电脑上会自动识别）。主站链接到从站时应该有打印相关信息。

第二个问题，后面在选择网段寻址时会有介绍。

这里面帧类型的0x88A4就定义了后面的数据帧是Ether Cat帧，后面就会继续分析Ether Cat帧。

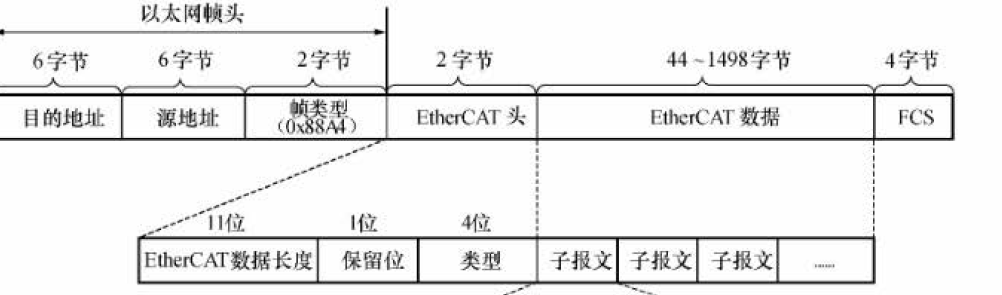


图1.2.

EtherCat头：



图1.3

我的理解就是：长度，子报文字节数。

类型：1

保留位就不写

#### 1.2 子报文头



图2.1

子报文头中包含几个信息：

命令（可选的命令下面有列表，表2.1）。

索引：暂时认为是子报文的序号。从0开始，还是从1开始。

地址区：设备寻址，和命令是一一对应的。

长度，后面数据段的字节数。

图2.2中是松下手册中的子报文段的解释。

WKC的计算方法后面会有介绍。



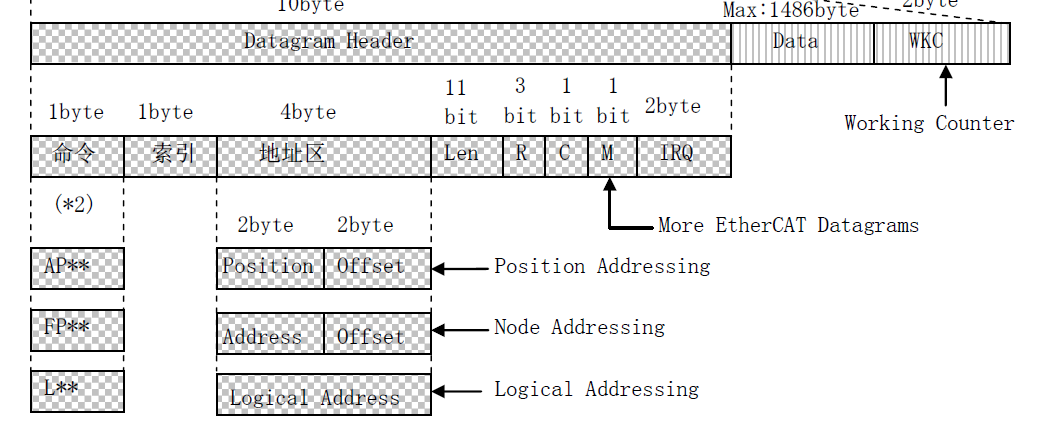


图2.2

从图2.2中可以看到，在命令中选择的模式不同（AP,FP,L），后面的寻址方式是不一样的。

其中cmd的详细描述：



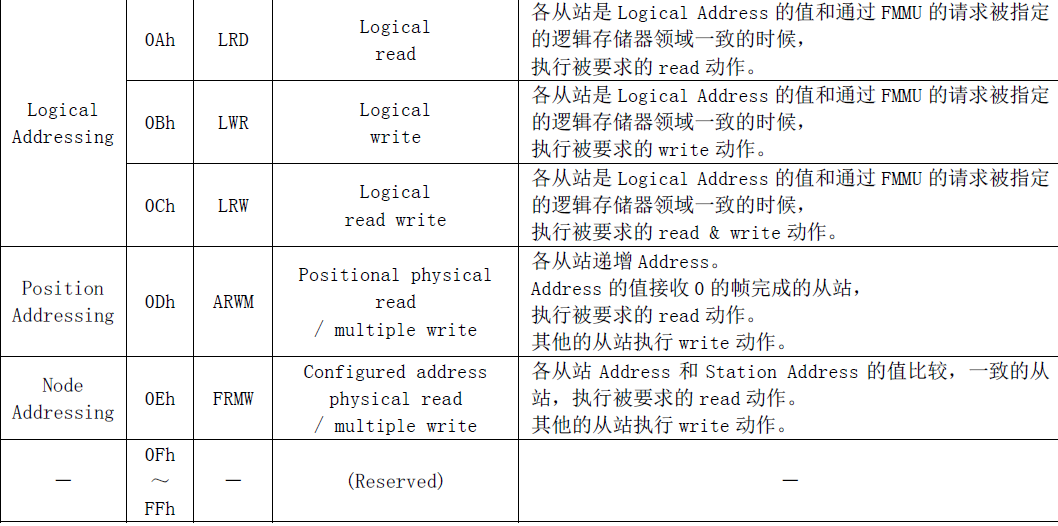
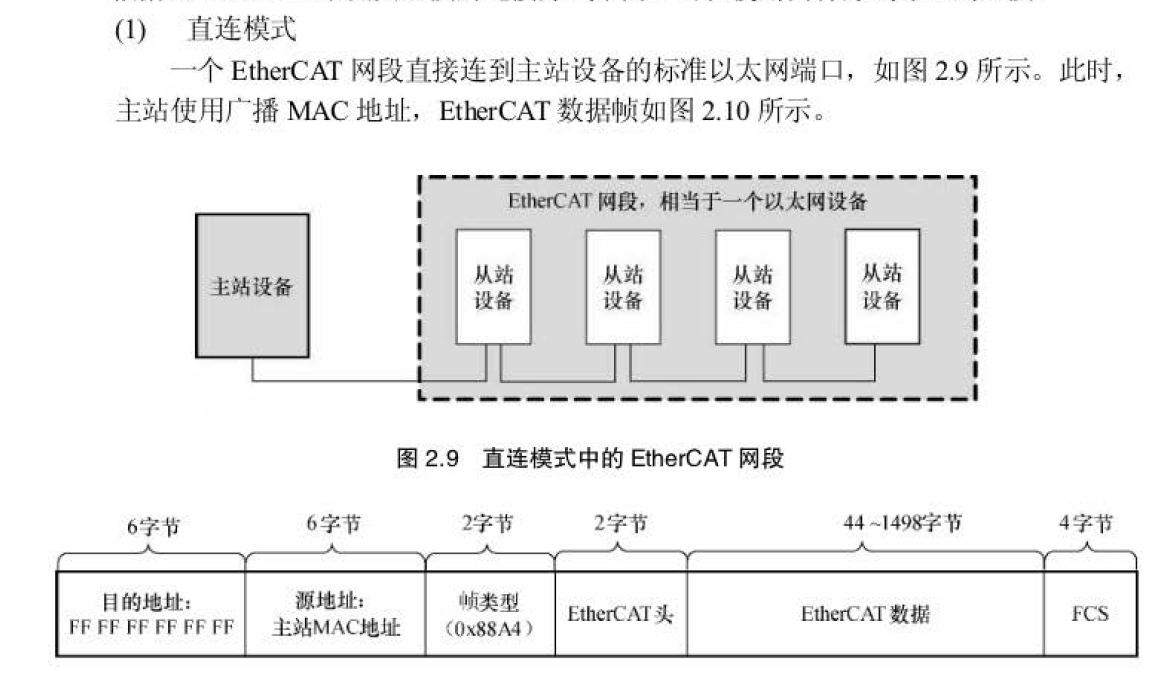


表2.1

#### 1.3 EtherCat网段寻址

网段寻址分两种，一种是直连，第二种是通过路由器连接。

其中直连：



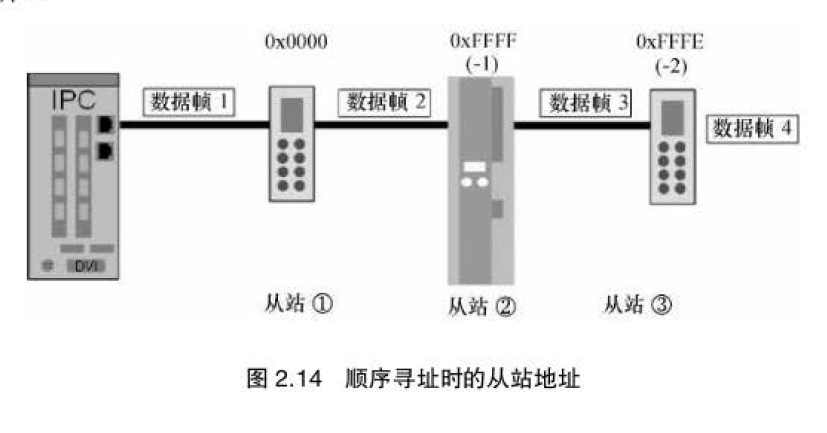
这就解释了前面的问题2，直连的话，目标地址写全F就可以了。

通过路由器连接的话：那就写路由器端口的MAC地址。

#### 1.4 设备寻址

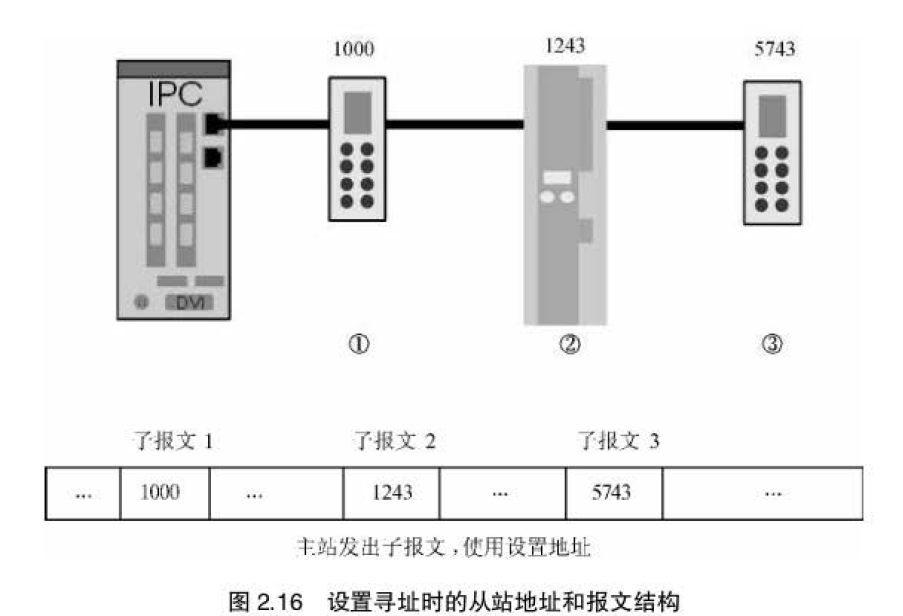
在每个网段内，都提供3种寻址方式。设备寻址-顺序寻址，设备寻址-设置寻址，逻辑寻址。

顺序寻址：第一个设备为0，后面一次减少1。





设置寻址：就是地址是固定的，

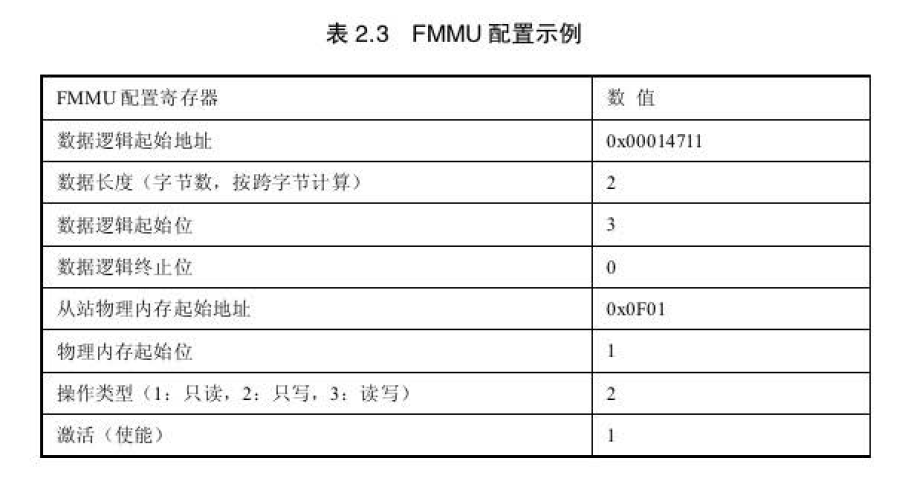


不过没找到如何设置从站地址。

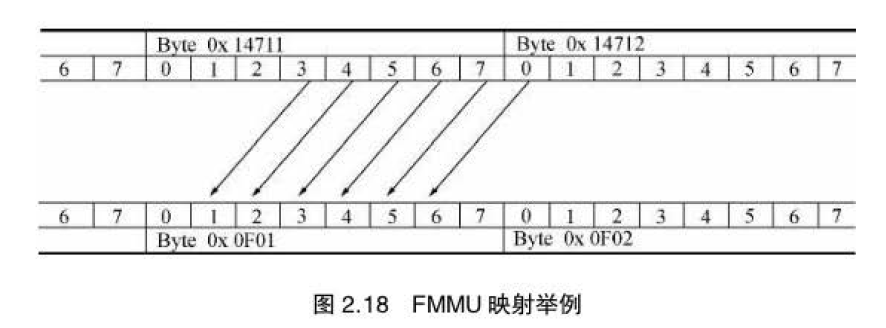
逻辑寻址与FMMU：

逻辑映射就是指将从站的内存中的物理地址中的内容映射到32位地址区中的某一些位当中。具体映射的方法是根据FMMU来决定的。

FMMU保护内容：



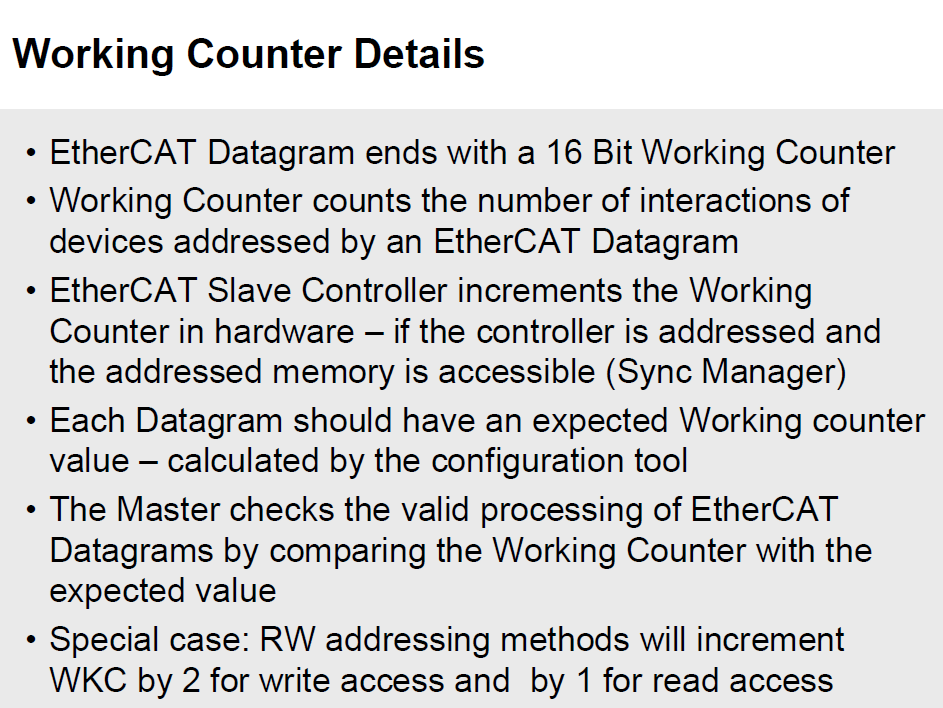
该FMMU映射对应的映射关系：



从图中可以看出，FMMU就是写明了映射的两端开始位置（包括字节号与位号）与终止位置（数据逻辑终止位与字节长度）。映射到的设备上物理地址的起始位置（0x0F01的1位）。

FMMU是在主站的链路层初始化时配置好后发给从站的。这个和对象字典时一样的。暂时还不知道如何配置。

#### 1.5 关于WKC



翻译：

EtherCat数据帧以一个16位的工作计数结尾。

工作计数器记录和EtherCat数据帧寻址的交互设备。

EtherCat从站控制器如果被寻址并且寻址内存可用的话，在硬件中将其加1.

每个数据帧都应该有一个期望的工作计数值，它是由配置工具计算出来的。

主站通过对比WKC和预期值来检查合法的EtherCat数据帧的处理过程

有一个特例：RW定位时，WKC会在写操作时加2，读操作时加1.

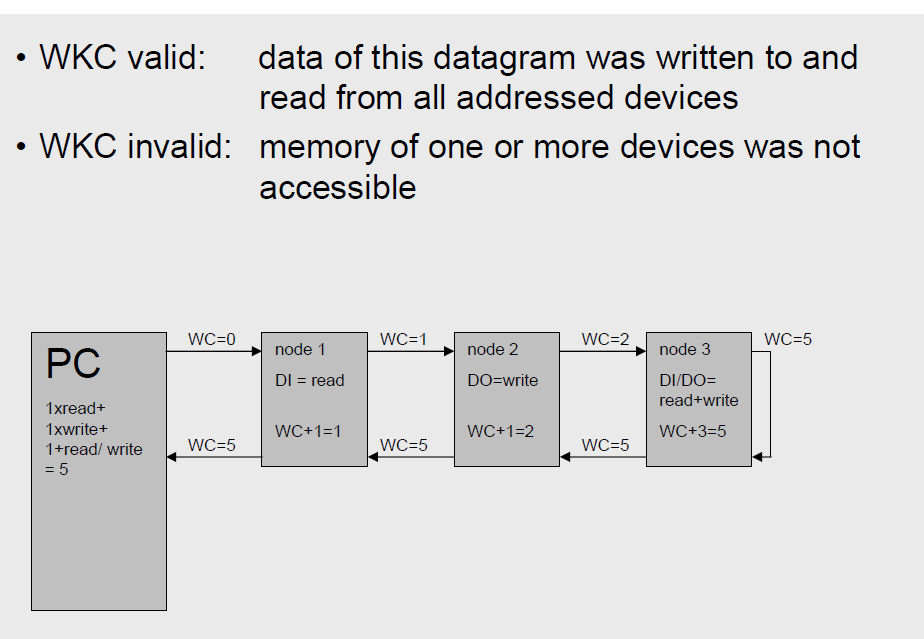


图1.5.1

图1.5.1中说明了WKC的增加过程，寻址模式为只读或者只写时，WKC每经过一个从站就会加1。在经过节点3时，由于是RW模式，所以加了3。然后返回。

#### 1.6 SyncManager

SM英文名字是同步管理器。和它的名字一样，它是控制从站刷新某一段内存行为的管理器。

它支持两种方式：

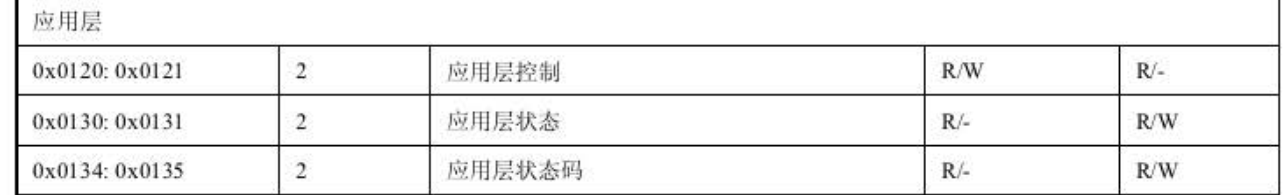
1．3段内存式。周期性过程数据。速度快，不保证数据安全。

2. 邮箱方式。一段内存，有握手，所以速度没保障。

DC与SM，中断的发生，有可能就是SM中设置的。暂时没完全了解。

#### 1.7 从站寄存器

首先，所有从站的行为都是通过写从站寄存器来实现的。ESC芯片中支持的寻址为 0x0000~0xFFFF，其中前4K（0x0000~0x0FFF）为寄存器空间。那么，后面就和厂商有关，就是厂商自己定义的。前4K可以看etherCat标准DATASHEET文档。后面看厂商给出的文档。



应用层控制，应该就是控制从站状态机的寄存器。

在wireshark上读取到的地址0x0130，data为02,那么意思就是说，读取的是应用层状态，结果为PREOP。具体内容看下表。



那么相反的，控制从站应用层状态，就是控制伺服上使能可以通过写寄存器0x120



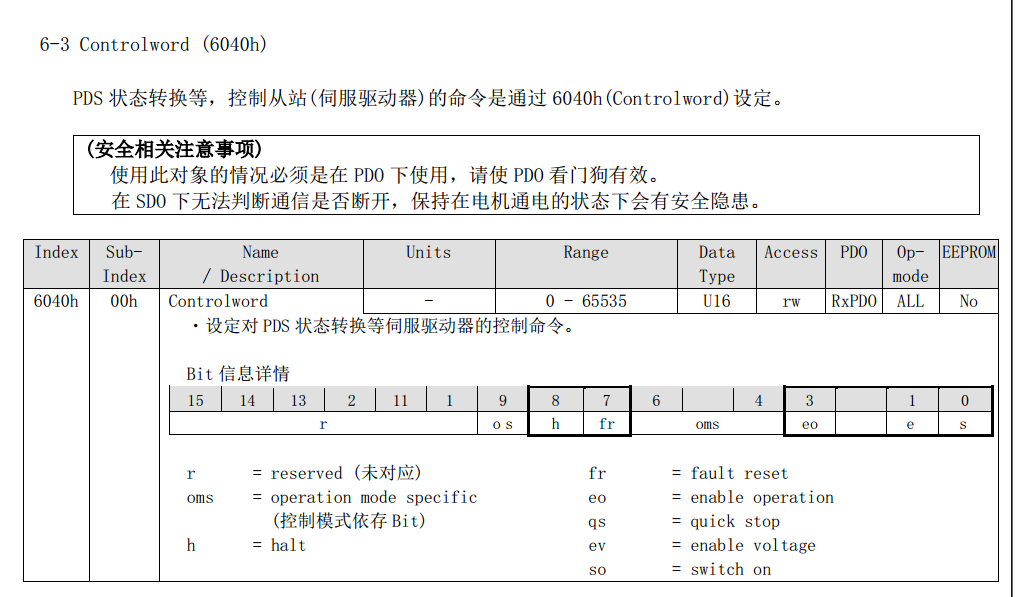
那么，还是要看igh的主站有哪些 接口来控制读写这些寄存器。

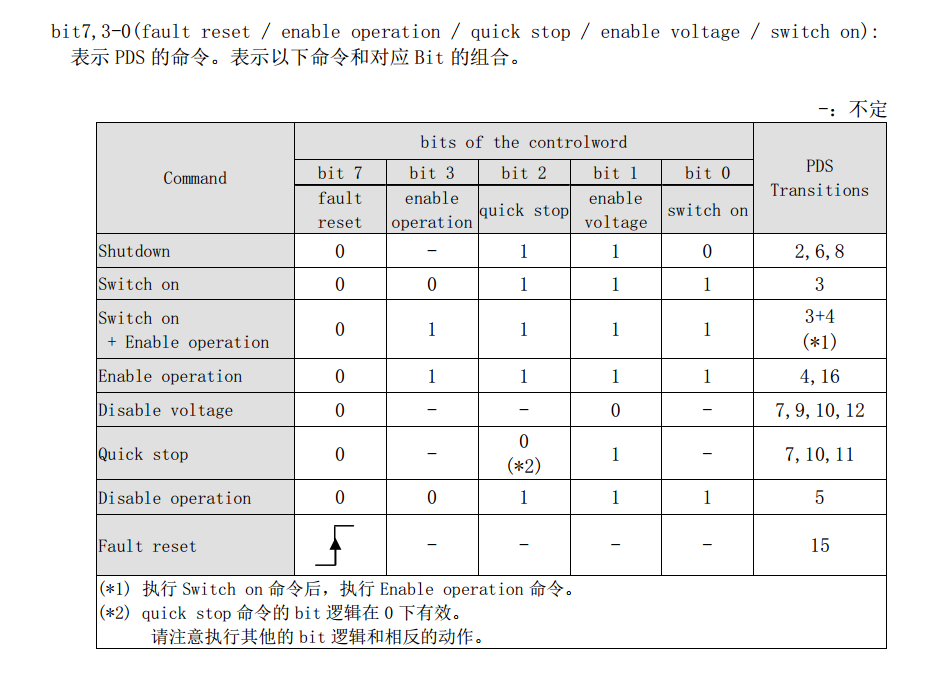
这里没有举例所有的寄存器功能。具体查看ethercat Datasheet手册，或者是《工业以太网现场总线ethercat》。

代码的笔记放到第二章。

1.8 松下的从站 PDO映射。

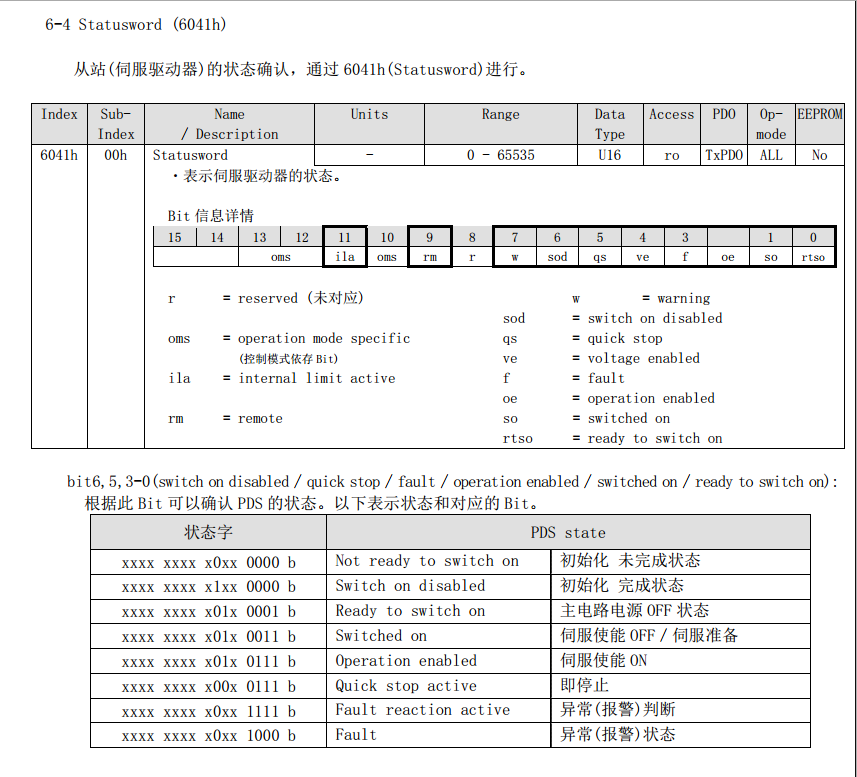
6040h控制字





这个控制字是用来控制伺服电机上使能的。

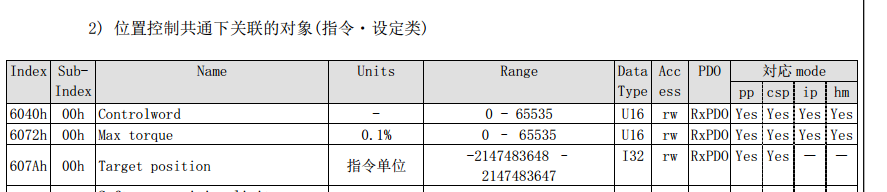
6041状态字



这个状态字读取伺服电机的状态。



控制模式选择，这里选择CSP模式。模式切换必须在disable的时候。并且会有2ms的延时。



607A写入目标位置。

### IGH主站学习。

#### 2.1 主站工具。

主站工具是调试程序非常好用的东西。其中可以利用reg\_read 与reg\_write对从站寄存器进行直接的读写。寄存器的定义可以参考工业以太网那本书中的定义。

还可以使用upload和download对4k后面的地址进行读写。

其中最有用的是slave和cstruct两个工具。其中slave可以读出从站的厂商id和设备id。后面在代码中注册主站的时候是可以用到的。后面也可以查看这两个函数的写法，写入到自己的程序中自动匹配从站。

Cstruct工具可以将从站的pdo映射相关信息写成c语言代码的格式，直接在控制主站的代码中应用。这个应该是不能自动匹配的。不过也可以生产xml文件，然后解析xml文件。这个可以暂时不考虑。

#### 2.2 主站lib的接口

现在使用的用户空间接口，因为rtdm接口编译没有通过。

后面xenomai进程可以测试看看效果。

用户端库的使用，就和文档中写的一样，编译生成动态库。然后，使用编译选项-lethercat就可以添加动态库进来。制定头文件的包含路径，-I/opt/etherlab/include。

在ecrt.h头文件中定义了用户空间中需要使用的所有接口函数。其中每个接口函数都会使用ioctrl来对应操作驱动提供的一个内核态的接口。

其中几个比较重要的函数：

##### 请求主站：

ecrt\_request\_master(0);

这个函数就这么调用就可以了。整个系统中允许申请多个主站，不过其实申请一个就够了。

其中包含了第一个从站配置的指针。而从站配置结构中也是个链表，它将后面所有的从站信息串联起来。

##### 创建域：

ecrt\_master\_create\_domain(master)

在整个主站中。所有的数据都是通过这个域来串联到一起的。看数据结构可以看出来，这个是单向的链表结构，包含了pdo映射的指针。

配置从站：

ec\_slave\_config\_t \*ecrt\_master\_slave\_config(

ec\_master\_t \*master, /\*\*< EtherCAT master \*/

uint16\_t alias, /\*\*< Slave alias. \*/

uint16\_t position, /\*\*< Slave position. \*/

uint32\_t vendor\_id, /\*\*< Expected vendor ID. \*/

uint32\_t product\_code /\*\*< Expected product code. \*/

);

这个函数就是创建从站配置的函数。其中master就填前面使用ecrt\_request\_master(0);函数返回的指针。然后，使用别名和位置对从站进行寻址。如果alias写0的话，那么就使用的是顺序寻址。Position直接写从0开始的值就可以。厂商id和设备id。可以通过读寄存器的值来获取。或者也可以使用主站提供的工具slave –v 获取所有的这些信息。

##### 配置从站pdo:

配置从站的pdo，需要多个函数。首先是配置sm寄存器。

/\*\* Specify a complete PDO configuration.

\*

\* This function is a convenience wrapper for the functions

\* ecrt\_slave\_config\_sync\_manager(), ecrt\_slave\_config\_pdo\_assign\_clear(),

\* ecrt\_slave\_config\_pdo\_assign\_add(), ecrt\_slave\_config\_pdo\_mapping\_clear()

\* and ecrt\_slave\_config\_pdo\_mapping\_add(), that are better suitable for

\* automatic code generation.

\*

\* The following example shows, how to specify a complete configuration,

\* including the PDO mappings. With this information, the master is able to

\* reserve the complete process data, even if the slave is not present at

\* configuration time:

\*

\* \code

\* ec\_pdo\_entry\_info\_t el3162\_channel1[] = {

\* {0x3101, 1, 8}, // status

\* {0x3101, 2, 16} // value

\* };

\*

\* ec\_pdo\_entry\_info\_t el3162\_channel2[] = {

\* {0x3102, 1, 8}, // status

\* {0x3102, 2, 16} // value

\* };

\*

\* ec\_pdo\_info\_t el3162\_pdos[] = {

\* {0x1A00, 2, el3162\_channel1},

\* {0x1A01, 2, el3162\_channel2}

\* };

\*

\* ec\_sync\_info\_t el3162\_syncs[] = {

\* {2, EC\_DIR\_OUTPUT},

\* {3, EC\_DIR\_INPUT, 2, el3162\_pdos},

\* {0xff}

\* };

\*

\* if (ecrt\_slave\_config\_pdos(sc\_ana\_in, EC\_END, el3162\_syncs)) {

\* // handle error

\* }

\* \endcode

\*

\* The next example shows, how to configure the PDO assignment only. The

\* entries for each assigned PDO are taken from the PDO's default mapping.

\* Please note, that PDO entry registration will fail, if the PDO

\* configuration is left empty and the slave is offline.

\*

\* \code

\* ec\_pdo\_info\_t pdos[] = {

\* {0x1600}, // Channel 1

\* {0x1601} // Channel 2

\* };

\*

\* ec\_sync\_info\_t syncs[] = {

\* {3, EC\_DIR\_INPUT, 2, pdos},

\* };

\*

\* if (ecrt\_slave\_config\_pdos(slave\_config\_ana\_in, 1, syncs)) {

\* // handle error

\* }

\* \endcode

\*

\* Processing of \a syncs will stop, if

\* - the number of processed items reaches \a n\_syncs, or

\* - the \a index member of an ec\_sync\_info\_t item is 0xff. In this case,

\* \a n\_syncs should set to a number greater than the number of list items;

\* using EC\_END is recommended.

\*

\* This method has to be called in non-realtime context before

\* ecrt\_master\_activate().

\*

\* \return zero on success, else non-zero

\*/

int ecrt\_slave\_config\_pdos(

ec\_slave\_config\_t \*sc, /\*\*< Slave configuration. \*/

unsigned int n\_syncs, /\*\*< Number of sync manager configurations in

\a syncs. \*/

const ec\_sync\_info\_t syncs[] /\*\*< Array of sync manager

configurations. \*/

);

从说明可以看出来，这个函数包装了几个函数的功能。将3个结构体串联到一起。

其中：

/\*\* Sync manager configuration information.

\*

\* This can be use to configure multiple sync managers including the PDO

\* assignment and PDO mapping. It is used as an input parameter type in

\* ecrt\_slave\_config\_pdos().

\*/

typedef struct {

uint8\_t index; /\*\*< Sync manager index. Must be less

than #EC\_MAX\_SYNC\_MANAGERS for a valid sync manager,

but can also be \a 0xff to mark the end of the list. \*/

ec\_direction\_t dir; /\*\*< Sync manager direction. \*/

unsigned int n\_pdos; /\*\*< Number of PDOs in \a pdos. \*/

ec\_pdo\_info\_t \*pdos; /\*\*< Array with PDOs to assign. This must contain

at least \a n\_pdos PDOs. \*/

ec\_watchdog\_mode\_t watchdog\_mode; /\*\*< Watchdog mode. \*/

} ec\_sync\_info\_t;

其中这个结构体就是保存sm信息的。看说明也说了，还包含了pdo映射的信息，还是是否使能看门狗。

代码举例：

static ec\_sync\_info\_t slave\_0\_syncs[] = {

{0, EC\_DIR\_OUTPUT, 0, NULL, EC\_WD\_DISABLE},

{1, EC\_DIR\_INPUT, 0, NULL, EC\_WD\_DISABLE},

{2, EC\_DIR\_OUTPUT, 1, slave\_0\_pdos + 0, EC\_WD\_ENABLE},

{3, EC\_DIR\_INPUT, 1, slave\_0\_pdos + 1, EC\_WD\_DISABLE},

{0xff}

};

这是松下伺服的sm配置参数。其中有输入方向的宏。Pdo映射地址是一个结构体的指针。后面看门狗的配置也是宏。最后是广播0xff.

这就引出了下面一个结构：

/\*\* PDO configuration information.

\*

\* This is the data type of the \a pdos field in ec\_sync\_info\_t.

\*

\* \see ecrt\_slave\_config\_pdos().

\*/

typedef struct {

uint16\_t index; /\*\*< PDO index. \*/

unsigned int n\_entries; /\*\*< Number of PDO entries in \a entries to map.

Zero means, that the default mapping shall be

used (this can only be done if the slave is

present at bus configuration time). \*/

ec\_pdo\_entry\_info\_t \*entries; /\*\*< Array of PDO entries to map. Can either

be \a NULL, or must contain at

least \a n\_entries values. \*/

} ec\_pdo\_info\_t;

这个就是pdo映射的内容。其中index就是ESC中，默认pdo映射信息的地址。这个是厂商指定下面的指针就是后面使用pdo的条目的结构体指针。Entries指的是标目的个数。

那么这就引出了下面一个结构体。Pdo条目信息（也就是对象字典）。

/\*\* PDO entry configuration information.

\*

\* This is the data type of the \a entries field in ec\_pdo\_info\_t.

\*

\* \see ecrt\_slave\_config\_pdos().

\*/

typedef struct {

uint16\_t index; /\*\*< PDO entry index. \*/

uint8\_t subindex; /\*\*< PDO entry subindex. \*/

uint8\_t bit\_length; /\*\*< Size of the PDO entry in bit. \*/

} ec\_pdo\_entry\_info\_t;

包含了index，subindex,和length这些信息。具体这些index对应的是什么意思，就要查看厂商手册中的对象字典了。

代码举例：

static ec\_pdo\_entry\_info\_t slave\_0\_pdo\_entries[] = {

{0x6040, 0x00, 16}, /\* Controlword \*/

{0x6060, 0x00, 8}, /\* Modes of operation \*/

{0x607a, 0x00, 32}, /\* Target position \*/

{0x60b8, 0x00, 16}, /\* Touch probe function \*/

{0x603f, 0x00, 16}, /\* Error code \*/

{0x6041, 0x00, 16}, /\* Statusword \*/

{0x6061, 0x00, 8}, /\* Modes of operation display \*/

{0x6064, 0x00, 32}, /\* Position actual value \*/

{0x60b9, 0x00, 16}, /\* Touch probe status \*/

{0x60ba, 0x00, 32}, /\* Touch probe pos1 pos value \*/

{0x60f4, 0x00, 32}, /\* Following error actual value \*/

{0x60fd, 0x00, 32}, /\* Digital inputs \*/

};

static ec\_pdo\_info\_t slave\_0\_pdos[] = {

{0x1600, 4, slave\_0\_pdo\_entries + 0}, /\* Receive PDO mapping 1 \*/

{0x1a00, 8, slave\_0\_pdo\_entries + 4}, /\* Transmit PDO mapping 1 \*/

};

上面就是伺服从站中默认pdo1的内容。不过pdo可以不使用默认的，不过这就需要对从站中1600,和1a00的内容进行修改。由于大部分功能默认pdo都是可以覆盖的。暂时没有去做对应的研究。

完成上面的配置以后，在调用函数ecrt\_master\_activate(master) 后，主站就会自己发送数据帧控制从站进入到op状态。就是说这个状态就主站是写好的。其实同样写好的还有sdo的状态机。不过，现在可以对这些不多做研究。可以后面再说。

不过只是这样，虽然可以进行注册。不过还是不能对这些pdo标目中的内容进行操作。还要获取对应条目的映射地址。

那么就需要下面这个函数：

/\*\* Registers a bunch of PDO entries for a domain.

\*

\* This method has to be called in non-realtime context before

\* ecrt\_master\_activate().

\*

\* \see ecrt\_slave\_config\_reg\_pdo\_entry()

\*

\* \attention The registration array has to be terminated with an empty

\* structure, or one with the \a index field set to zero!

\* \return 0 on success, else non-zero.

\*/

int ecrt\_domain\_reg\_pdo\_entry\_list(

ec\_domain\_t \*domain, /\*\*< Domain. \*/

const ec\_pdo\_entry\_reg\_t \*pdo\_entry\_regs /\*\*< Array of PDO

registrations. \*/

);

这个函数就是注册pdo内容到指定地址当中的。后面这个pdo\_entry\_reg\_t就是保存这个信息的结构体。其定义如下:

typedef struct {

uint16\_t alias; /\*\*< Slave alias address. \*/

uint16\_t position; /\*\*< Slave position. \*/

uint32\_t vendor\_id; /\*\*< Slave vendor ID. \*/

uint32\_t product\_code; /\*\*< Slave product code. \*/

uint16\_t index; /\*\*< PDO entry index. \*/

uint8\_t subindex; /\*\*< PDO entry subindex. \*/

unsigned int \*offset; /\*\*< Pointer to a variable to store the PDO entry's

(byte-)offset in the process data. \*/

unsigned int \*bit\_position; /\*\*< Pointer to a variable to store a bit

position (0-7) within the \a offset. Can be

NULL, in which case an error is raised if the

PDO entry does not byte-align. \*/

} ec\_pdo\_entry\_reg\_t;

这个结构就指定了需要访问的pdo条目的完整信息。其中我们需要返回使用的，就是这个offset和bit\_position。这个bit\_position是当需要的信息为某几个位而不是完整字节的时候使用。不过这里用不到。

所以最重要的就是这个返回的offset。它是一个主站domain PDO数据首地址后面的一个偏移地址。

代码举例：

const static ec\_pdo\_entry\_reg\_t domain1\_regs[] = {

{0,0, Beckhoff\_Slave\_0, 0x6040, 0, &off\_slave\_0\_Controlword},

{0,0, Beckhoff\_Slave\_0, 0x6060, 0, &off\_slave\_0\_operation\_mode},

{0,0, Beckhoff\_Slave\_0, 0x607a, 0, &off\_slave\_0\_target\_position},

{0,0, Beckhoff\_Slave\_0, 0x6041, 0, &off\_slave\_0\_Statusword},

{0,0, Beckhoff\_Slave\_0, 0x6061, 0, &off\_slave\_0\_Modes},

{0,0, Beckhoff\_Slave\_0, 0x6064, 0, &off\_slave\_0\_position\_actual\_value},

// {AnaInSlavePos, Beckhoff\_EL3102, 0x3101, 2, &off\_ana\_in\_value},

// {AnaOutSlavePos, Beckhoff\_EL4102, 0x3001, 1, &off\_ana\_out},

// {DigOutSlavePos, Beckhoff\_EL2032, 0x3001, 1, &off\_dig\_out},

{}

};

这里面off后面的这些变量就是需要系统返回给我们的偏移量。后面会在发送与接受数据的时候用到。

##### 配置DC:

/\*\* Configure distributed clocks.

\*

\* Sets the AssignActivate word and the cycle and shift times for the sync

\* signals.

\*

\* The AssignActivate word is vendor-specific and can be taken from the XML

\* device description file (Device -> Dc -> AssignActivate). Set this to zero,

\* if the slave shall be operated without distributed clocks (default).

\*

\* This method has to be called in non-realtime context before

\* ecrt\_master\_activate().

\*

\* \attention The \a sync1\_shift time is ignored.

\*/

void ecrt\_slave\_config\_dc(

ec\_slave\_config\_t \*sc, /\*\*< Slave configuration. \*/

uint16\_t assign\_activate, /\*\*< AssignActivate word. \*/

uint32\_t sync0\_cycle, /\*\*< SYNC0 cycle time [ns]. \*/

int32\_t sync0\_shift, /\*\*< SYNC0 shift time [ns]. \*/

uint32\_t sync1\_cycle, /\*\*< SYNC1 cycle time [ns]. \*/

int32\_t sync1\_shift /\*\*< SYNC1 shift time [ns]. \*/

);

从这个函数的说明中可以看出来，这个assign\_activate是在厂商xml文件中定义的。Sync0\_cycle就是DC通信周期。Shift就是偏移时间。

代码举例：

ecrt\_slave\_config\_dc(sc\_ana\_in, 0x0300, PERIOD\_NS, 4400000, 0, 0);

##### 收发PDO：

收发PDO函数相对比较简单：

// receive process data

ecrt\_master\_receive(master);

ecrt\_domain\_process(domain1);

// send process data

ecrt\_domain\_queue(domain1);

ecrt\_master\_send(master);

不过这只是更新了主站域中的数据，需要使用主站提供的宏来将其读出来或者写进去。这个时候就需要用到上面注册的PDO偏移。

data->statusword= EC\_READ\_U16(domain1\_pd+off\_slave\_0\_Statusword);

data->mode\_display = EC\_READ\_U8(domain1\_pd+off\_slave\_0\_Modes);

data->actual\_position=EC\_READ\_S32(domain1\_pd+off\_slave\_0\_position\_actual\_value);

EC\_WRITE\_U16(domain1\_pd + off\_slave\_0\_Controlword, data->controlword);

EC\_WRITE\_U8(domain1\_pd + off\_slave\_0\_operation\_mode, 0x08);

EC\_WRITE\_S32(domain1\_pd + off\_slave\_0\_target\_position, data->target\_position);