

## lgh1.5.2 etherCAT接口分析教程



iiidd777

IT技术分享及教程

3 人赞同了该文章

### 1. 用户空间库

本地应用程序接口驻留在内核空间中，因此只能从内核访问。为了使应用程序接口在用户空间程序中可用，已经创建了一个用户空间库，该用户空间库可以根据LGPL版本2 [5]的条款和条件链接到程序。

这个库被命名为libethercat。它的源代码驻留在lib /子目录中，并且在使用make时默认生成。它被安装在安装前的lib /路径下，如libethercat.a（用于静态链接），libethercat.la（用于libtool）和libethercat.so（用于动态链接）。

### 2. 使用库

应用程序接口头文件ecrt.h可以在内核和用户上下文中使用。以下简单的显示了如何使用EtherCAT功能构建程序。整个示例可以在主源文件的examples / user /路径中找到。

```
#include <ecrt.h>

int main (void)

{

ec_master_t * master = ecrt_request_master (0);

if (! master )

return 1; // error

pause (); // wait for signal

return 0;

}
```

该程序可以编译并通过以下命令动态链接到库：

列表2.1：使用用户空间库的链接器命令

```
gcc ethercat .c -o ectest -I/opt/ etherlab / include -L/opt / etherlab /lib -lethercat \

-Wl ,-- rpath -Wl ,/ opt / etherlab /lib
```

该库也可以静态编译：

```
gcc -static ectest .c -o ectest -I/opt/ etherlab / include \

/opt/ etherlab /lib/ libethercat .a
```

### 3. 实现

内核API基本上是通过主字符设备传输到用户空间的。内核API的函数调用通过ioctl () 接口映射到用户空间。 用户空间API函数共享一组通用的ioctl () 调用。 接口的内核部分直接调用相应的API函数，导致最小的附加延迟。

出于性能方面的原因，实际的域进程数据不会在每次访问时在内核和用户内存之间进行复制：而是将数据映射到用户空间应用程序。主站配置和激活后，主模块会创建一个跨所有域的过程数据存储区，并将其映射到用户空间，以便应用程序可以直接访问过程数据。 因此，从用户空间访问过程数据时不会有额外的延迟。

内核/用户API差异由于过程数据的内存映射，内存由库函数内部管理。因此，为域提供外部内存是不可能的，例如在内核API中。相应的函数只能在kernel space中使用。这是在用户空间中使用应用程序接口时唯一的区别。

### 4. RTDM接口

当使用Xenomai或RTAI等实时扩展的用户空间接口时，不推荐使用ioctl () ，因为它可能会干扰实时操作。为此，实时设备模型 (RTDM) [17]已经被开发出来。主模块提供了一个RTDM接口，除了普通的字符设备，如果主源配置了-enable-rtdm。要强制应用程序使用RTDM接口而不是普通字符设备，它必须与libethercat\_rtdm库而不是libethercat链接。 libethercat\_rtdm的使用是透明的，所以具有完整API的EtherCAT头ecrt.h可以照常使用。

为了使代码清单2.1中的例子使用RTDM库，链接器命令必须改变如下：

```
gcc ethercat -with - rtdm .c -o ectest -I/opt/ etherlab / include \
-L/ opt/ etherlab / lib - lethercat_rtdm \
-Wl ,-- rpath -Wl ,/ opt/ etherlab / lib
```

### 5. 应用程序接口

应用程序接口为应用程序提供访问EtherCAT主站的功能和数据结构。接口的完整文档作为Doxygen [13]注释包含在头文件include/ecrt.h中。 它可以直接从文件注释中读取，也可以作为更舒适的HTML文档读取。

### 6. ecrt.h接口分析

应用程序使用IGH提供的接口时，需要包含“ecrt.h”头文件，该文件中包含了相应的数据结构和函数的声明。ecrt.h提供的接口分为以下几部分：全局函数、主站方法、从站配置方法、域方法、SDO请求方法、VOE处理方法、寄存器请求方法。

#### 6.1 全局函数

##### 6.1.1 ecrt\_version\_magic

功能：返回实时接口的版本信息。

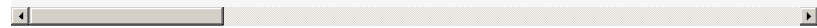
函数原型： unsigned int ecrt\_version\_magic(void);

参数： master\_index – 请求的主机的索引

返回值： 成功 – 指向主机的指针；失败 -- NULL

##### 6.1.2 ecrt\_request\_master

功能: 请求EtherCAT主机进行实时操作。在应用程序可以访问EtherCAT主程序之前, 它必须保留一个供独占使用



函数原型: `ec_master_t *ecrt_request_master(unsigned int master_index);`

参数: `master_index` – 请求的主机的索引

返回值: 成功 – 指向主机的指针; 失败 -- NULL

### 6.1.3 `ecrt_release_master`

功能: 释放请求的ethercat主机

函数原型: `void ecrt_release_master(ec_master_t *master);`

参数: `master` -- ethercat主机

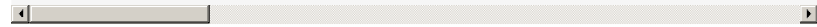
返回值: 无

## 6.2 主站方法

### 6.2.1 `ecrt_master_callbacks`

功能: 设置锁定回调。对于并发主访问, 即如果应用程序以外的其他实例希望在总线上发送和接收数据报, 应用程序

函数原型: `void ecrt_master_callbacks(ec_master_t *master,`



`void (*send_cb)(void *), void (*receive_cb)(void *),`

`void *cb_data);`

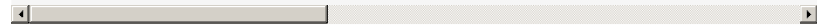
参数: `master` -- ethercat主机; `send_cb` -- 发送回调函数;

`receive_cb` -- 接收回调函数; `cb_data` -- 传递给回调函数的任意指针

返回值: 无

### 6.2.2 `ecrt_master_create_domain`

功能: 创建新的流程数据域。对于过程数据交换, 至少需要一个过程数据域。此方法创建一个新的过程数据域, 并



函数原型: `ec_domain_t *ecrt_master_create_domain(ec_master_t *master);`

参数: `master` -- ethercat主机

返回值: 成功 -- 指向新域对象的指针; 失败 – NULL

### 6.2.3 `ecrt_master_slave_config`

功能: 获取从站的配置, 为给定的别名和位置元组创建从属配置对象并返回它。如果存在相同的配置, 则重新使用  
使用别名和位置参数进行寻址。如果别名为零, 则一个位置被解释为所需的从站的环位置。如果别名为非零, 则它



这个方法分配内存, 并应在`ecrt_master_activate` () 之前在非实时上下文中调用。

函数原型: ec\_slave\_config\_t \*ecrt\_master\_slave\_config(

ec\_master\_t \*master, uint16\_t alias, uint16\_t position,

uint32\_t vendor\_id, uint32\_t product\_code );

参数: master -- ethercat主机; alias – 从站别名; position – 从站位置;

vendor\_id – 厂商ID; product\_code – 产品ID

返回值: 成功 -- 指向从站配置结构的指针; 失败 -- NULL

#### 6.2.4 ecrt\_master\_select\_reference\_clock

功能: 为分布式时钟选择参考时钟, 如果某个主节点没有调用此方法, 或者配置指针为空, 则具有DC功能的第一个从机将提供参考时钟。

函数原型: int ecrt\_master\_select\_reference\_clock(

ec\_master\_t \*master, ec\_slave\_config\_t \*sc );

参数: master -- ethercat主机; sc -- 指向从站配置的指针

返回值: 成功 – 返回0; 失败 – 返回错误代码

#### 6.2.5 ecrt\_master

功能: 获取主站信息。此函数的堆上不分配内存。注意指向此结构的指针必须指向有效变量。

函数原型: int ecrt\_master(

ec\_master\_t \*master, ec\_master\_info\_t \*master\_info );

参数: master -- ethercat主机; master\_info -- 指向主机输出信息结构的指针

返回值: 成功 – 返回0; 失败 – 返回 < 0

#### 6.2.6 ecrt\_master\_get\_slave

功能: 获取从站的信息, 尝试找到具有给定环位置的从属对象。获得的信息存储在一个结构中。此函数的堆上不分配内存。

函数原型: int ecrt\_master\_get\_slave(ec\_master\_t \*master,

uint16\_t slave\_position, ec\_slave\_info\_t \*slave\_info);

参数: master -- ethercat主机; slave\_position –从站的位置; slave\_info -- 指向从站输出信息结构的指针

返回值: 成功 -- 指向主机的指针; 失败 -- NULL

#### 6.2.7 ecrt\_master\_sdo\_download

功能: 执行SDO下载请求以将数据写入从站设备。此请求由主状态机处理。此方法会阻塞, 直到请求被处理, 并且可能不会在实时上下文中调用。

函数原型: int ecrt\_master\_sdo\_download(ec\_master\_t \*master,

uint16\_t slave\_position, uint16\_t index, uint8\_t subindex, uint8\_t \*data, size\_t data\_size,  
uint32\_t \*abort\_code);

参数: master -- ethercat主机; slave\_position --从站的位置; index -- SDO索引; subindex -- SDO子索引; data -- 下载的缓冲区; data\_size -- 缓冲区的大小; abort\_code -- 中止SDO下载代码

返回值: 成功 – 返回0; 失败 – 返回 < 0

#### 6.2.8 ecrt\_master\_sdo\_download\_complete

功能: 执行SDO下载请求, 通过完全访问将数据写入从设备。此请求由主状态机处理。此方法会阻塞, 直到请求被处理, 并且可能不会在实时上下文中调用。

函数原型: int ecrt\_master\_sdo\_download\_complete(

ec\_master\_t \*master, uint16\_t slave\_position, uint16\_t index, uint8\_t \*data, size\_t data\_size,  
uint32\_t \*abort\_code);

参数: master -- ethercat主机; slave\_position --从站的位置; index -- SDO索引; data -- 下载的缓冲区; data\_size -- 缓冲区的大小; abort\_code -- 中止SDO下载代码

返回值: 成功 – 返回0; 失败 – 返回 < 0

#### 6.2.9 ecrt\_master\_sdo\_upload

功能: 执行SDO 上传请求以从服务器读取数据。此请求由主状态机处理。此方法会被阻塞, 直到请求被处理。

函数原型: int ecrt\_master\_sdo\_upload(ec\_master\_t \*master,

uint16\_t slave\_position, uint16\_t index, uint8\_t subindex, uint8\_t \*target, size\_t target\_size,  
size\_t \*result\_size, uint32\_t \*abort\_code);

参数: master -- ethercat主机; slave\_position --从站的位置; index -- SDO索引; subindex -- SDO子索引; target -- 上传的目标缓冲区; target\_size -- 目标缓冲区的大小; result\_size -- 已上传的数据的大小; abort\_code -- 中止SDO上传代码

返回值: 成功 – 返回0; 失败 – 返回 < 0

#### 6.2.10 ecrt\_master\_write\_idn

功能: 执行SoE写入请求。开始写入IDN并阻塞, 直到处理完请求或发生错误。

函数原型: int ecrt\_master\_write\_idn(ec\_master\_t \*master,

uint16\_t slave\_position, uint8\_t drive\_no,

uint16\_t idn, uint8\_t \*data,

size\_t data\_size, uint16\_t \*error\_code);

参数: master -- ethercat主机; slave\_position -- 从站位置;

drive\_no -- 驱动号； idn – SOE IDN；

data -- 写入数据的缓冲区； data\_size -- 数据大小；

error\_code – 保存SOE返回的错误码

返回值：成功 – 返回0； 失败 – 返回 < 0

#### 6.2.11 ecrt\_master\_read\_idn

功能: 执行SoE读取请求。开始读取IDN并阻塞, 直到处理完请求或发生错误。

函数原型: int ecrt\_master\_read\_idn(ec\_master\_t \*master,

uint16\_t slave\_position, uint8\_t drive\_no,

uint16\_t idn, uint8\_t \*target,

size\_t target\_size, size\_t \*result\_size,

uint16\_t \*error\_code

);

参数: master -- ethercat主机; slave\_position -- 从站位置;

drive\_no -- 驱动号； idn – SOE IDN；

target -- 读取数据的缓冲区； target\_size -- 数据大小；

result\_size – 实际接收数据的大小； error\_code -- 保存SOE返回的错误码

返回值：成功 – 返回0； 失败 – 返回 < 0

#### 6.2.12 ecrt\_master\_activate

功能: 完成配置阶段并准备循环操作。此函数告诉主机配置阶段已完成, 实时操作将开始。函数为域分配内部内存。

注意: 调用此函数后, 实时应用程序负责循环调用ecrt\_master\_send () 和 ecrt\_master\_receive (), 以确保总线通信。在调用这个函数之前, 由主线程负责, 所以不能调用这些函数! 方法本身分配内存, 不应在实时上下文中调用。

函数原型: int ecrt\_master\_activate(ec\_master\_t \*master);

参数: master -- ethercat主机

返回值：成功 – 返回0； 失败 – 返回 < 0

#### 6.2.13 ecrt\_master\_set\_send\_interval

功能: 设置调用ecrt\_master\_send()的间隔。此信息帮助主状态机决定主状态机可以向帧追加多少数据。当主状态机

函数原型: int ecrt\_master\_set\_send\_interval(

ec\_master\_t \*master, size\_t send\_interval);

参数: master -- ethercat主机; send\_interval -- 发送时间间隔, us

返回值: 成功 – 返回0; 失败 – 返回 < 0

#### 6.2.14 ecrt\_master\_send

功能: 发送队列中的所有数据报。该方法接收所有排队等待传输的数据报, 将它们放入帧中, 并将它们传递给以太网设备进行发送。

函数原型: void ecrt\_master\_send(ec\_master\_t \*master);

参数: master -- ethercat主机

返回值: 无

#### 6.2.15 ecrt\_master\_deactivate

功能: 删除总线配置。创建的所有对象ecrt\_master\_create\_domain()、ecrt\_master\_slave\_config()、ecrt\_slave\_config\_create\_voe\_handler() 已释放, 因此指向它们的指针将无效。不应在实时上下文中调用此方法。。

函数原型: void ecrt\_master\_deactivate(ec\_master\_t \*master);

参数: master -- ethercat主机

返回值: 无

#### 6.2.16 ecrt\_master\_receive

功能: 从硬件获取接收到的帧并处理数据报。通过调用中断服务例程查询网络设备接收到的帧。提取接收到的数据报并放入队列中。接收到的数据报和超时的数据报将被标记并退出队列。

函数原型: void ecrt\_master\_receive(ec\_master\_t \*master);

参数: master -- ethercat主机

返回值: 无

#### 6.2.17 ecrt\_master\_state

功能: 读取当前主机的状态

函数原型: void ecrt\_master\_state(

const ec\_master\_t \*master, ec\_master\_state\_t \*state);

参数: master -- ethercat主机; state -- 保存主机的状态信息

返回值: 无

### 6.2.18 ecrt\_master\_link\_state

功能： 读取冗余链路的当前状态。

函数原型: int ecrt\_master\_link\_state(const ec\_master\_t \*master,


unsigned int dev\_idx, ec\_master\_link\_state\_t \*state );

参数: master -- ethercat主机; dev\_idx -- 设备索引; state -- 存储链路状态的信息;

返回值: 成功 – 返回0; 失败 – 返回 < 0

### 6.2.19 ecrt\_master\_application\_time

功能: 设置应用程序时间。在使用分布式时钟操作从机时, 主机必须知道应用程序的时间。时间不是由主机本身增



注意: 此方法的第一次调用用于计算从站SYNC0/1中断的相位延迟。必须在第一个实时周期中调用实时周期, 或者只能在第一个实时周期中调用。否则会出现同步问题。该时间用于设置从机的<tt>系统时间偏移量</tt>和<tt>循环操作开始时间</tt>寄存器, 以及通过

ecrt\_master\_sync\_reference\_clock () 将直流参考时钟与应用程序时间同步时使用。时间定义为2000-01-01 00:00的纳秒。转换历元时间可以用EC\_TIMEVAL2NANO () 宏完成。

函数原型: void ecrt\_master\_application\_time(

ec\_master\_t \*master, uint64\_t app\_time);

参数: master -- ethercat主机; app\_time -- 应用时间

返回值: 无

### 6.2.20 ecrt\_master\_sync\_reference\_clock

功能: 将DC参考时钟漂移补偿数据报排队发送, 参考时钟将与上次取消ecrt\_master\_application\_time () 提供的应用程序时间同步。

函数原型: void ecrt\_master\_sync\_reference\_clock(ec\_master\_t \*master);

参数: master -- ethercat主机

返回值: 无

### 6.2.21 ecrt\_master\_sync\_slave\_clocks

功能: 将DC时钟漂移补偿数据报排队发送, 所有的子时钟与基准时钟同步

函数原型: void ecrt\_master\_sync\_slave\_clocks(ec\_master\_t \*master);

参数: master -- ethercat主机

返回值: 无

### 6.2.21 ecrt\_master\_reference\_clock\_time

功能: 获取参考时钟系统时间的低32位, 可用于使主时钟与基准时钟同步。参考时钟系统时间通过ecrt\_master



注意：返回的时间是参考时钟的系统时间减去参考时钟的传输延迟。

函数原型：int ecrt\_master\_reference\_clock\_time(

ec\_master\_t \*master, uint32\_t \*time);

参数：master -- ethercat主机；time -- 指向查询的系统时间的指针

返回值：成功 – 返回0；失败 – 返回 < 0

#### 6.2.22 ecrt\_master\_sync\_monitor\_queue

功能：将DC同步监视数据报排队以进行发送。数据报广播读取所有“系统时差”寄存器(\a 0x092c)以获得直流同  
函数原型：void ecrt\_master\_sync\_monitor\_queue(ec\_master\_t \*master);

参数：master -- ethercat主机

返回值：无

#### 6.2.23 ecrt\_master\_sync\_monitor\_process

功能：处理DC同步监控数据报。如果同步监视数据报是在

ecrt\_master\_sync\_monitor\_queue ()，可以使用此方法查询结果。

函数原型：uint32\_t ecrt\_master\_sync\_monitor\_process(  
ec\_master\_t \*master);

参数：master -- ethercat主机

返回值：返回最大时差的上估计值，单位：ns

#### 6.2.24 ecrt\_master\_reset

功能：重新配置从站设备

函数原型：void ecrt\_master\_reset(ec\_master\_t \*master);

参数：master -- ethercat主机

返回值：无

### 6.3从站配置方法

#### 6.3.1 ecrt\_slave\_config\_sync\_manager

功能：配置同步管理器。设置同步管理器的方向。这将覆盖来自SII的默认控制寄存器的方向位。此方法之前必须

函数原型：int ecrt\_slave\_config\_sync\_manager(  
ec\_slave\_config\_t \*sc, uint8\_t sync\_index,

ec\_slave\_config\_t \*sc, uint8\_t sync\_index,

ec\_direction\_t direction, ec\_watchdog\_mode\_t watchdog\_mode);

参数: sc -- 从站配置; sync\_index -- 同步管理器索引;

direction -- 输入/输出; watchdog\_mode -- 看门狗模式

返回值: 成功 – 返回0; 失败 – 返回 < 0

### 6.3.2 ecrt\_slave\_config\_watchdog

功能: 配置从站的看门狗时。此方法之前必须在非实时上下文中调用ecrt\_master\_activate ()。

函数原型: void ecrt\_slave\_config\_watchdog(

ec\_slave\_config\_t \*sc, uint16\_t watchdog\_divider,

uint16\_t watchdog\_intervals);

参数: sc -- 从站配置; watchdog\_divider – 40ns间隔的数量。用作所有从站看门狗的基本单元; watchdog\_intervals -- 过程数据监视程序的基本间隔数;

返回值: 无

### 6.3.3 ecrt\_slave\_config\_pdo\_assign\_add

功能: 将PDO添加到同步管理器的PDO分配中。此方法之前必须在非实时上下文中调用ecrt\_master\_activate ()。

函数原型: int ecrt\_slave\_config\_pdo\_assign\_add(

ec\_slave\_config\_t \*sc, uint8\_t sync\_index,

uint16\_t index);

参数: sc -- 从站配置; sync\_index -- 同步管理器索引;

index -- 分配PDO索引

返回值: 成功 – 返回0; 失败 – 返回 < 0

### 6.3.4 ecrt\_slave\_config\_pdo\_assign\_clear

功能: 清除同步管理器的PDO配置。这可以在分配PDO之前通过调用ecrt\_slave\_config\_pdo\_assign\_add()。

此方法之前必须在非实时上下文中调用ecrt\_master\_activate ()。

函数原型: void ecrt\_slave\_config\_pdo\_assign\_clear(

ec\_slave\_config\_t \*sc, uint8\_t sync\_index );

参数: sc -- 从站配置; sync\_index -- 同步管理器索引;

返回值: 无

### 6.3.5 ecrt\_slave\_config\_pdo\_mapping\_add () 函数

功能： 在PDO映射中添加一个PDO条目。此方法之前必须在非实时上下文中调用  
ecrt\_master\_activate () 。

函数原型： int ecrt\_slave\_config\_pdo\_mapping\_add(

ec\_slave\_config\_t \*sc, uint16\_t pdo\_index,

uint16\_t entry\_index, uint8\_t entry\_subindex,

uint8\_t entry\_bit\_length);

参数： sc -- 从站配置； pdo\_index -- PDO索引；

entry\_index – 要添加到PDO映射的PDO条目的索引； entry\_subindex – 要添加到PDO映射的  
PDO项的子索引； entry\_bit\_length – PDO项的大小；

返回值： 成功 – 返回0； 失败 – 返回 < 0

### 6.3.6 ecrt\_slave\_config\_pdo\_mapping\_clear

功能： 清除给定的PDO映射。此方法之前必须在非实时上下文中调用  
ecrt\_master\_activate () 。

函数原型： void ecrt\_slave\_config\_pdo\_mapping\_clear(

ec\_slave\_config\_t \*sc, uint16\_t pdo\_index);

参数： sc -- 从站配置； pdo\_index -- PDO索引；

返回值： 无

### 6.3.7 ecrt\_slave\_config\_pdos

功能： 指定完整的PDO配置。此函数是函数的方便包装器

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

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

ecrt\_slave\_config\_pdo\_mapping\_add () , 它们更适合自动代码生成。

下面的示例演示如何指定完整的配置，包括PDO映射。有了这些信息，主机可以保留完整的过程  
数据，即使在配置时从机不存在：

```
ec_pdo_entry_info_t el3162_channel1[]={
```

```
{0x3101, 1, 8}, //状态
```

```
{0x3101, 2, 16};//值
```

```
};
```

```
ec_pdo_entry_info_t el3162_channel2[]={
```

```
{0x3102, 1, 8}, //状态
```

```
{0x3102, 2, 16};//值
```

```
};
```

```
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) ) {
```

```
//处理错误
```

```
}
```

下一个示例显示如何仅配置PDO分配。每个分配的PDO的条目取自PDO的默认映射。请注意，如果PDO配置为空并且从机离线，PDO条目注册将失败。

```
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
```

```
}
```

如果-已处理的项目数达到syncs，或-ec sync\_info\_t项的索引成员为0xff，则同步的处理将停止。在这种情况下，同步应设置为大于列表项数的数字；建议使用EC\_END命令。

此方法之前必须在非实时上下文中调用ecrt\_master\_activate () 。

函数原型: int ecrt\_slave\_config\_pdos(

ec\_slave\_config\_t \*sc, unsigned int n\_syncs,

const ec\_sync\_info\_t syncs[]);

参数: sc -- 从站配置; n\_syncs – 同步管理器的配置数;

syncs[] -- 同步管理器配置数组

返回值: 成功 – 返回0; 失败 – 返回 < 0

### 6.3.8 ecrt\_slave\_config\_reg\_pdo\_entry

功能: 为域中的过程数据交换注册PDO条目。在指定的PDO中搜索给定的PDO条目。如果给定的条目未映射, 则引发

如果还没有完成。返回请求的PDO条目数据在域的进程数据中的偏移量。可选地, 可以通过\abit\_position输出参数检索PDO入口位位置 (0-7)。此指针可能为NULL, 在这种情况下, 如果PDO项没有字节对齐, 则会引发错误。此方法之前必须在非实时上下文中调用ecrt\_master\_activate ()。

函数原型: int ecrt\_slave\_config\_reg\_pdo\_entry(

ec\_slave\_config\_t \*sc, uint16\_t entry\_index,

uint8\_t entry\_subindex, ec\_domain\_t \*domain,

unsigned int \*bit\_position

);

参数: sc -- 从站配置; entry\_index – 要注册的PDO项的索引;

entry\_subindex --要注册的PDO项的子索引; domain – 域;

bit\_position -- 如果需要位寻址, 则选择地址

返回值: 成功 – 返回 >= 0; 失败 – 返回 < 0

### 6.3.9 ecrt\_slave\_config\_reg\_pdo\_entry\_pos

功能: 使用其位置注册PDO条目, 类似于ecrt\_slave\_config\_reg\_pdo\_entry(), 但不使用pdo索引, 而是在p

函数原型: int ecrt\_slave\_config\_reg\_pdo\_entry\_pos(

ec\_slave\_config\_t \*sc, uint8\_t sync\_index,

unsigned int pdo\_pos, unsigned int entry\_pos,

ec\_domain\_t \*domain,unsigned int \*bit\_position);

参数: sc -- 从站配置; sync\_index –同步管理器索引;

pdo\_pos – PDO在SM中的位置; entry\_pos -- PDO中入口的位置;

domain -- 域; bit\_position -- 如果需要位寻址, 则选择地址

返回值: 成功 – 返回 >= 0; 失败 – 返回 < 0

### 6.3.10 ecrt\_slave\_config\_dc () 函数

功能: 配置分布式时钟。为同步信号设置AssignActivate字以及周期和移位时间。AssignActivate单词是特

函数原型: void ecrt\_slave\_config\_dc(

ec\_slave\_config\_t \*sc, uint16\_t assign\_activate,

uint32\_t sync0\_cycle, int32\_t sync0\_shift,

uint32\_t sync1\_cycle, int32\_t sync1\_shift);

参数: sc -- 从站配置; assign\_activate -- 赋值激活字;

sync0\_cycle -- SYNC0周期时间ns; sync0\_shift -- SYNC0转移时间;

sync1\_cycle -- SYNC0周期时间ns; sync1\_shift -- SYNC0转移时间;

返回值: 成功 -- 指向主机的指针; 失败 -- NULL

### 6.3.11 ecrt\_slave\_config\_sdo

功能: 添加SDO配置, SDO配置存储在从配置对象中, 并在主配置从机时下载到从机。这通常在主设备激活时发生

注意: PDO分配 (\p 0x1C10-\p 0x1C2F) 和PDO映射 (\p 0x1600-\p 0x17FF和\p 0x1A00-\p 0x1BFF) 的sdo不应配置为此函数, 因为它们是主配置完成的从配置的一部分。请使用 ecrt\_slave\_config\_pdos ()。这是用于添加SDO配置的通用函数。请注意, 此函数不会进行任何端接校正。如果需要特定于数据类型的函数 (自动更正endianness), 请查看 ecrt\_slave\_config\_sdo8 ()、ecrt\_slave\_config\_sdo16 () 和ecrt\_slave\_config\_sdo32 ()。此方法之前必须在非实时上下文中调用ecrt\_master\_activate ()。

函数原型: int ecrt\_slave\_config\_sdo(ec\_slave\_config\_t \*sc,

uint16\_t index, uint8\_t subindex,

const uint8\_t \*data, size\_t size);

参数: sc -- 从站配置; index -- SDO索引;

entry\_index --SDO子索引; data -- 指向data的指针;

size -- data的大小

返回值: 成功 – 返回0; 失败 – 返回 < 0

### 6.3.12 ecrt\_slave\_config\_complete\_sdo

功能: 为完整的SDO添加配置数据。SDO数据通过CompleteAccess传输。必须包括第一个子索引 (0) 的数据。此方法之前必须在非实时上下文中调用ecrt\_master\_activate ()。

函数原型: int ecrt\_slave\_config\_complete\_sdo(

ec\_slave\_config\_t \*sc, uint16\_t index,

const uint8\_t \*data, size\_t size);

参数: sc -- 从站配置; index -- SDO索引;

data -- 指向data的指针; size -- data的大小

返回值: 成功 – 返回0; 失败 – 返回 < 0

### 6.3.13 ecrt\_slave\_config\_create\_sdo\_request

功能: 在实时操作期间创建一个SDO请求来交换SDO。释放主节点时, 创建的SDO请求对象将自动释放。此方法之前必须在非实时上下文中调用ecrt\_master\_activate ()。

函数原型: ec\_sdo\_request\_t \*ecrt\_slave\_config\_create\_sdo\_request(

ec\_slave\_config\_t \*sc, uint16\_t index,

uint8\_t subindex, size\_t size);

参数: sc -- 从站配置; index -- SDO索引;

subindex -- SDO子索引; size -- data的大小

返回值: 成功 -- 指向新的SDO请求指针; 失败 – NULL

### 6.3.14 ecrt\_slave\_config\_create\_voe\_handler

功能: 创建一个VoE处理程序, 以便在实时操作期间交换特定于供应商的数据。每个从配置的VoE处理程序的数量不能超过10。

函数原型: ec\_voe\_handler\_t \*ecrt\_slave\_config\_create\_voe\_handler(

ec\_slave\_config\_t \*sc, size\_t size);

参数: sc -- 从站配置; size – 数据大小

返回值: 成功 -- 指向新的VOE处理指针; 失败 – NULL

### 6.3.15 ecrt\_slave\_config\_create\_reg\_request

功能: 创建一个寄存器请求, 在实时操作期间交换EtherCAT寄存器内容。此接口不应用于接管主功能, 而是用于请求寄存器内容。

函数原型: ec\_reg\_request\_t \*ecrt\_slave\_config\_create\_reg\_request(

ec\_slave\_config\_t \*sc, size\_t size);

参数: sc -- 从站配置; size – 数据大小

返回值: 成功 -- 指向新的寄存器请求指针; 失败 – NULL

### 6.3.16 ecrt\_slave\_config\_state

功能: 输出从站的配置状态

函数原型: void ecrt\_slave\_config\_state(

const ec\_slave\_config\_t \*sc, ec\_slave\_config\_state\_t \*state);

参数: sc -- 从站配置; state -- 需要写入的state对象

返回值: 无

## 6.4域方法

### 6.4.1 ecrt\_domain\_reg\_pdo\_entry\_list

功能: 为一个域注册一组PDO条目, 此方法之前必须在非实时上下文中调用 ecrt\_master\_activate () 。

函数原型: int ecrt\_domain\_reg\_pdo\_entry\_list(ec\_domain\_t \*domain,

const ec\_pdo\_entry\_reg\_t \*pdo\_entry\_regs);

参数: domain -- 域; pdo\_entry\_regs -- 注册的PDO数组

返回值: 成功 – 返回0; 失败 – 返回 < 0

### 6.4.2 ecrt\_domain\_size

功能: 返回域中process数据的大小

函数原型: size\_t ecrt\_domain\_size(const ec\_domain\_t \*domain);

参数: domain -- 域

返回值: 成功 –process数据映像的大小; 失败 --返回 < 0

### 6.4.3 ecrt\_domain\_data

功能: 返回域的进程数据。- 在内核上下文中: 如果外部内存与ecrt\_domain\_external\_memory()一起提供, 则

否则它将指向内部分配的内存。在这种情况下, 不能在调用master () 之前调用后者。—在用户空间上下文中: 必须在ecrt\_master\_activate () 以获取映射的域进程数据内存。

函数原型: uint8\_t \*ecrt\_domain\_data(ec\_domain\_t \*domain);

参数: domain -- 域

返回值: 成功 – 进程数据内存的指针; 失败 -- NULL

### 6.4.4 ecrt\_domain\_process

功能: 确定域数据报的状态。评估接收到的数据报的工作计数器, 并在必要时输出统计信息。必须在ecrt\_maste

函数原型: void ecrt\_domain\_process(ec\_domain\_t \*domain);

参数: domain -- 域



返回值: 无

#### 6.4.5 ecrt\_domain\_queue

功能: 将主数据报队列中的所有域数据报排队。调用此函数可标记域的数据报以便在 ecrt\_master\_send () 的下次调用。

函数原型: void ecrt\_domain\_queue(ec\_domain\_t \*domain);

参数: domain -- 域

返回值: 无

#### 6.4.6 ecrt\_domain\_state

功能: 读取域中的状态。利用该方法, 可以实时监控过程数据交换。

函数原型: void ecrt\_domain\_state(

const ec\_domain\_t \*domain, ec\_domain\_state\_t \*state);

参数: domain -- 域; state -- 域的状态信息

返回值: 无

### 6.5 SDO请求方法

#### 6.5.1 ecrt\_sdo\_request\_index

功能: 设置SDO索引和子索引。

注意: 如果SDO索引和/或子索引在ecrt\_sdo\_request\_state () 返回EC\_request\_BUSY, 这可能会导致意外结果。

函数原型: void ecrt\_sdo\_request\_index(ec\_sdo\_request\_t \*req,

uint16\_t index, uint8\_t subindex);

参数: req -- SDO请求; index -- SDO 索引;

Subindex -- SDO子索引

返回值: 无

#### 6.5.2 ecrt\_sdo\_request\_timeout

功能: 设置SDO请求超时。如果无法在指定时间内处理请求, 则If将标记为失败。超时永久存储在请求对象中, 并

函数原型: void ecrt\_sdo\_request\_timeout(

ec\_sdo\_request\_t \*req, uint32\_t timeout);

参数: req -- SDO请求; timeout -- 超时时间, ms;

返回值: 无

### 6.5.3 ecrt\_sdo\_request\_data

功能： 访问SDO请求的数据。-读取操作成功后，可以像往常一样使用EC\_READ\_\* () 宏计算整数数据。例子：

```
uint16_t value=EC_READ_U16(ecrt_sdo_request_data(sdo));
```

-如果要触发写入操作，则必须将数据写入内存。如果要写入整型数据，请使用EC\_WRITE\_x\* () 宏。确保数据能放入内存中。内存大小是ecrt\_slave\_config\_create\_sdo\_request () 的参数。EC\_WRITE\_U16 (ecrt\_sdo\_request\_data (sdo) , 0xFFFF) ；

注意：在读取操作期间，返回值可能无效，因为如果读取的SDO数据不在内部，则可能会重新分配内部SDO数据内存

函数原型：uint8\_t \*ecrt\_sdo\_request\_data(

ec\_sdo\_request\_t \*req);

参数： req -- SDO请求；

返回值： 成功 -- 指向请求内部SDO数据内存的指针； 失败 -- NULL

### 6.5.4 ecrt\_sdo\_request\_data\_size

功能： 返回当前SDO数据大小。当创建SDO请求时，数据大小被设置为保留内存的大小。在读取操作之后，大小被



函数原型：size\_t ecrt\_sdo\_request\_data\_size(

const ec\_sdo\_request\_t \*req);

参数： req -- SDO请求；

返回值： 成功 -- 指向主机的指针； 失败 -- NULL

### 6.5.5 ecrt\_sdo\_request\_state

功能： 返回当前SDO数据的大小

函数原型：ec\_request\_state\_t ecrt\_sdo\_request\_state(

const ec\_sdo\_request\_t \*req);

参数： req -- SDO请求；

返回值： SDO数据的大小

### 6.5.6 ecrt\_sdo\_request\_write

功能： 调度SDO写操作。注意，当ecrt\_sdo\_request\_state()返回EC\_REQUEST\_BUSY时，可能不会调用此方法。

函数原型：void ecrt\_sdo\_request\_write(ec\_sdo\_request\_t \*req);

参数： req -- SDO请求；

返回值: 无

### 6.5.7 ecrt\_sdo\_request\_read

功能: 安排SDO读操作。注意, 当ecrt\_sdo\_request\_state()返回EC\_REQUEST\_BUSY时, 可能不会调用此方法。

函数原型: void ecrt\_sdo\_request\_read(ec\_sdo\_request\_t \*req);

参数: req -- SDO请求;

返回值: 无

## 6.6 VOE处理方法

### 6.6.1 ecrt\_voe\_handler\_send\_header

功能: 为将来的发送操作设置VoE头。VoE消息应包含一个4字节的供应商ID, 后跟as报头处的2字节供应商类型。i

这些值是有效的, 将用于以后的发送操作, 直到下次调用此方法为止。

函数原型: void ecrt\_voe\_handler\_send\_header(ec\_voe\_handler\_t \*voe,

uint32\_t vendor\_id, uint16\_t vendor\_type);

参数: voe -- VOE 句柄; vendor\_id – VID;

vendor\_type -- 厂商指定的类型

返回值: 无

### 6.6.2 ecrt\_voe\_handler\_received\_header

功能: 读取接收到的VoE消息的头数据。此方法可用于在读取操作成功后获取接收到的VoE报头信息。头信息存储7  
函数原型: void ecrt\_voe\_handler\_received\_header(  
const ec\_voe\_handler\_t \*voe, uint32\_t \*vendor\_id,

uint16\_t \*vendor\_type);

参数: voe -- VOE 句柄; vendor\_id – VID;

vendor\_type -- 厂商指定的类型;

返回值: 无

### 6.6.3 ecrt\_voe\_handler\_data

功能: 访问VoE处理程序的数据。此函数返回指向VoE处理程序的内部内存的指针, 该指针指向VoE标题后面的实际  
注意: 返回的指针不一定是持久的: 在读取之后操作时, 内存可能已重新分配。这可能是通过的size参数保留足够  
函数原型: uint8\_t \*ecrt\_voe\_handler\_data(ec\_voe\_handler\_t \*voe);

参数: voe -- VOE 句柄;

返回值：返回指向VoE处理程序的内部内存的指针。

#### 6.6.4 ecrt\_voe\_handler\_data\_size

功能：返回当前数据大小。数据大小是没有头的VoE数据的大小（请参见 ecrt\_VoE\_handler\_send\_header（））。创建VoE处理程序时，数据大小设置为保留内存的大小。在写入操作中，数据大小设置为要写入的字节数。读取操作后，大小设置为读取数据的大小。在任何其他情况下都不会修改大小。

```
函数原型: size_t ecrt_voe_handler_data_size(
```

```
const ec_voe_handler_t *voe);
```

参数：voe -- VOE 句柄；

返回值：当前数据的大小

#### 6.6.5 ecrt\_voe\_handler\_write

```
功能: 启动VoE写入操作。调用此函数后, ecrt_voe_handler_execute()方法必须在每个总线周期中调用, 只要  
函数原型: void ecrt_voe_handler_write(
```

```
ec_voe_handler_t *voe, size_t size);
```

参数：voe -- VOE 句柄； size -- 写入的字节数；

返回值：无

#### 6.6.6 ecrt\_voe\_handler\_read

功能：启动VoE读取操作。调用此函数后，ecrt\_voe\_handler\_execute（）方法。必须在每个总线周期中调用，只要它返回EC\_REQUEST\_BUSY。处理程序繁忙时，不能启动其他操作。状态机查询从服务器的发送邮箱，以获取要发送到主服务器的新数据。如果在EC\_VOE\_RESPONSE\_TIMEOUT（在master/VOE\_handler.c中定义）内没有数据出现，则操作失败。成功后，可以通过ecrt\_voe\_handler\_data\_size（），而接收数据的voe报头可以使用ecrt\_voe\_handler\_received\_header（）检索。

```
函数原型: void ecrt_voe_handler_read(
```

```
ec_voe_handler_t *voe);
```

参数：voe -- VOE 句柄；

返回值：无

#### 6.6.7 ecrt\_voe\_handler\_execute

```
功能: 执行处理程序。此方法执行VoE处理程序。它必须在每个总线周期中调用, 只要它返回EC_REQUEST_BUSY。  
函数原型: ec_request_state_t ecrt_voe_handler_execute(
```

```
ec_voe_handler_t *voe);
```

参数: req -- SDO请求;

返回值: 返回处理程序状态

## 6.7寄存器请求方法

### 6.7.1 ecrt\_reg\_request\_data

功能: 访问寄存器请求的数据。此函数返回指向请求内部内存的指针。

-读取操作成功后, 可以像往常一样使用EC\_READ\_\* () 宏计算整数数据。例子: uint16\_t value=EC\_READ\_U16(ecrt\_reg\_request\_data(reg\_request));

-如果要触发写入操作, 则必须将数据写入内存。如果要写入整型数据, 请使用EC\_WRITE\_x\* () 宏。确保数据能放入内存中。内存大小是ecrt\_slave\_config\_create\_reg\_request () 的参数。

EC\_WRITE\_U16 (ecrt\_reg\_request\_data(reg\_request), 0xFFFF) ; 。

函数原型: uint8\_t \*ecrt\_reg\_request\_data(ec\_reg\_request\_t \*req );

参数: req -- 寄存器请求;

返回值: 返回指向内存的指针。

### 6.7.2 ecrt\_reg\_request\_state

功能: 获取寄存器请求的当前状态。

函数原型: ec\_request\_state\_t ecrt\_reg\_request\_state(

const ec\_reg\_request\_t \*req);

参数: req -- 寄存器请求;

返回值: 返回请求状态。

### 6.7.3 ecrt\_reg\_request\_write

功能: 安排寄存器写入操作。

注意: 当ecrt\_reg\_request\_state () 返回EC\_request\_BUSY时, 不能调用此方法。

size参数被截断为请求创建时给定的大小。

函数原型: void ecrt\_reg\_request\_write(ec\_reg\_request\_t \*req,

uint16\_t address, size\_t size);

参数: req -- 寄存器请求; address -- 寄存器地址;

size – 写入的大小

返回值: 无

### 6.7.4 ecrt\_reg\_request\_read

功能：安排寄存器读取操作。

注意：当ecrt\_reg\_request\_state () 返回EC\_request\_BUSY时，不能调用此方法。

size参数被截断为请求创建时给定的大小。。

```
函数原型: void ecrt_reg_request_read(ec_reg_request_t *req,
```

uint16\_t address, size\_t size);


参数：req -- SDO请求;

返回值：无

发布于 2021-05-28 15:02

- 开放式 IEC 61131 控制系统设计 (书籍)
- CAT
- 接口设计

文章被以下专栏收录



IT技术专栏  
IT技术分享及教程

推荐阅读



AXI协议详解0：介绍与资料梳理

hal3515



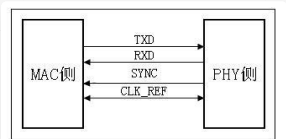
基于postman测试接口(整套接口测试)

铅华

股票数据接口

前言：最近在思考平时在网上看到的股票信息，能否通过接口获取到。今天在家一直在思考这个问题和找些信息，现阶段找到一些信息，先记录下。腾讯股票接口获取最新行情， http://qt.gtimg....

空空




媒体独立接口 (MII, Meida Independent Interface)

碎碎思 发表于FPGA之...



还没有评论

写下你的评论...



▲ 赞同 3



● 添加评论

🔗 分享

♥ 喜欢

★ 收藏

📄 申请转载

