PCIE_ECAT 接口说明文档

本文档描述了基于 SG-PCIe-ECAT 平台的 PCIe EtherCAT 通信库的 API 接口。

1. 概述

该库提供了通过 PCIe 总线进行 EtherCAT 主站通信的功能,包括设备管理、总线控制、SDO 通信以及数据交换。

1.1 核心特性

- **PCIe 直连通信**: 底层 PCIe 驱动封装,提供高效的数据传输通道
- **EtherCAT 主站功能**: 支持设备发现、连接、配置管理及周期同步控制
- **高速数据交换**: 提供内存映射,实现低延迟控制
- **SDO 通信**: 完整的 CoE (CANopen over EtherCAT) 服务数据对象访问

1.2 技术架构

- **分层架构**: 应用层 -> 接口层 -> 核心逻辑层 -> 硬件抽象层
- **混合语言支持**: C 核心库,提供 C 接口供 C、C++、C# (.NET) 、python等调用

---

2. 调用流程说明

2.1 库函数调用流程

PCIE_ECAT 通信库的使用遵循标准的设备初始化、数据交互、资源释放流程。

流程图:

---

3. 数据类型与枚举

3.1 句柄类型

类型	描述
`ECAT_HANDLE`	EtherCAT 控制句柄,用于标识一个打开的设备会话

使用说明:

```
ECAT_HANDLE hHandle = NULL;
int32_t ret = EcatOpen(&hHandle, 0);
if (ret == ECAT_S_OK) {
    // 使用 hHandle 进行后续操作
    EcatClose(hHandle);
}
```

3.2 状态枚举 (EcatState)

枚举值	数值	描述
`EcatStateNotSet`	0	未设置,从站离线
`EcatStateINIT`	1	INIT 初始化状态
`EcatStatePREOP`	2	PREOP 预操作状态
`EcatStateBOOT`	3	BOOT 引导状态
`EcatStateSAFEOP`	4	SAFEOP 安全操作状态
`EcatStateOP`	8	OP 操作状态
`EcatStateMask`	15	状态掩码,用于提取状态位
`EcatStateErrorFlag`	16	状态错误标志位

状态转换流程:

```
INIT -> PREOP -> SAFE-OP -> OP (正常运行路径)
         |-> BOOT (固件升级路径)
```

使用示例:

```
uint16_t state = 0;
EcatGetMasterState(hHandle, &state);

// 检查是否处于 OP 状态
if (state == EcatStateOP) {
    printf("已全部进入 OP 状态\n");
}
if ((state&EcatStateMask) == EcatStateOP) {
    printf("有从站设备已进入 OP 状态\n");
}

// 检查是否有错误
if (state & EcatStateErrorFlag) {
    printf("主站状态异常\n");
}
```

3.3 输入输出区域索引

常量	数值	描述	特性
`PI_AREA_LOCAL_OUTPUT`	0	本地输出缓存	可读写,非线程安全,同步控制
`PI_AREA_LOCAL_INPUT`	1	本地输入缓存	可读写,非线程安全,同步控制

注意事项:

- 提供直接的内存映射访问,性能最优
- 非线程安全,多线程访问需自行加锁
- 使用前必须调用 EcatPINMap 获取地址

---

4. 核心接口函数

4.1 设备管理

EcatOpen

打开 ECAT 设备并获取控制句柄。

```
EXPORT int32_t EcatOpen(OUT ECAT_HANDLE* phHandle, IN uint8_t board_id);
```

参数:

- `phHandle`: 返回 ECAT 控制句柄 (输出参数)
- `board_id`: 对应板子端的拨码 ID (输入参数)，同时插入多个PCIE板时，每个板子只能对应一个ID

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
ECAT_HANDLE hHandle = NULL;
int32_t ret = EcatOpen(&hHandle, 0);  // 打开拨码为 0 的设备
if (ret != ECAT_S_OK) {
    printf("打开设备失败,错误码: %d\n", ret);
    return -1;
}
printf("设备打开成功,句柄: %p\n", hHandle);
```

注意事项:

- 每个设备只能打开一次,重复打开会返回错误
- 使用后必须调用 EcatClose 释放资源
- `board_id` 必须与硬件拨码开关一致

EcatClose

关闭 ECAT 设备并释放资源。

```
EXPORT int32_t EcatClose(IN ECAT_HANDLE hHandle);
```

参数:

- `hHandle`: 板卡的控制句柄 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
if (hHandle != NULL) {
    EcatClose(hHandle);
    hHandle = NULL;  // 避免悬空指针
}
```

注意事项:

- 关闭后句柄失效,不得再次使用
- 建议在程序退出或不再需要设备时立即关闭

EcatReset

复位 ECAT 主站设备。

```
EXPORT int32_t EcatReset(IN ECAT_HANDLE hHandle);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用场景:

- EtherCAT设备通信异常时尝试恢复

EcatSetTimeout

设置通信超时时间。

```
EXPORT int32_t EcatSetTimeout(IN ECAT_HANDLE hHandle, IN uint32_t timeout_ms);
```

参数:

- `hHandle`: 板卡的控制句柄 (输入参数)
- `timeout_ms`: 超时时间,单位为毫秒 (输入参数),默认值为 2000ms

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
// 设置超时时间为 5 秒
EcatSetTimeout(hHandle, 5000);

// SDO 通信将使用新的超时时间
EcatCoeSDOUpload(hHandle, 0, 0x1018, 1, &len, data, 5000);
```

注意事项:

- 超时时间影响所有阻塞式 API 调用
- 设置过短可能导致正常操作超时失败
- 设置过长会影响错误检测的及时性
- 建议根据网络状况和应用需求调整,通常 1000-5000ms

EcatSetBoardCommTimeout

设置板子端通信超时时间。

```
EXPORT int32_t EcatSetBoardCommTimeout(IN ECAT_HANDLE hHandle, IN uint32_t
timeout_ms);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)
- `timeout_ms`: 板子端通信超时时间,单位为毫秒 (输入参数),默认值为 0 (不超时)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

功能说明:

- 通过 PCIe 总线将超时时间配置发送到板子端固件
- 当设置为 0 时,板子端不会因超时而退出 EtherCAT 主站运行,会持续输出最后修改的数据

- 当设置为非 0 值时,板子端在指定时间内未收到主机通信数据将自动停止 EtherCAT 主站运行

使用场景:

- 防止主机程序异常时板子端持续输出旧数据
- 实现看门狗机制,确保通信链路健康
- 调试时控制板子端行为

使用示例:

```
// 示例 1: 设置 5 秒超时
int32_t ret = EcatSetBoardCommTimeout(hHandle, 5000);
if (ret == ECAT_S_OK) {
    printf("板子端超时时间设置为 5 秒\n");
}

// 示例 2: 禁用超时 (默认行为)
EcatSetBoardCommTimeout(hHandle, 0);
printf("板子端超时已禁用,将持续输出最后数据\n");
```

注意事项:

- 该函数影响的是板子端固件的行为,与 EcatSetTimeout 不同
- `EcatSetTimeout` 设置的是主机端 API 调用的超时时间
- `EcatSetBoardCommTimeout` 设置的是板子端等待主机通信的超时时间
- 建议在程序启动后尽早设置
- 超时后板子端会停止 EtherCAT 主站运行并停止PCIE通信，需要重新调用 EcatOpen 恢复

EcatGetApiVersion

获取 API 版本号字符串。

```
EXPORT int32_t EcatGetApiVersion(OUT char* pBuffer, IN uint32_t bufferSize);
```

参数:

- `pBuffer` : 存储版本号的缓冲区 (输出参数)
- `bufferSize` : 缓冲区大小 (输入参数),建议至少为 32 字节

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败,具体错误码参考 `pcie_errno.h`

功能说明:

- 返回格式化的版本号字符串,格式为 "主版本.次版本.修订号"
- 例如: "1.0.0"
- 可用于兼容性检查、日志记录、问题诊断等场景

使用示例:

```
// 示例 1: 基础用法
char version[32];
int32_t ret = EcatGetApiVersion(version, sizeof(version));
if (ret == ECAT_S_OK) {
    printf("API 版本: %s\n", version);  // 输出: API 版本: 1.0.0
}

// 示例 2: 版本比较
char version[32];
EcatGetApiVersion(version, sizeof(version));
if (strcmp(version, "1.0.0") >= 0) {
    printf("API 版本满足最低要求\n");
} else {
    printf("警告: API 版本过低,请升级库文件\n");
}

// 示例 3: 日志记录
char version[32];
EcatGetApiVersion(version, sizeof(version));
LogPrint("程序启动, PCIE_ECAT API 版本: %s\n", version);
```

C# 调用示例:

```
using System.Text;
using System.Runtime.InteropServices;

public class EcatBoardApi
{
    [DllImport("PCIE_ECAT.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int EcatGetApiVersion(StringBuilder pBuffer, uint
bufferSize);
}

// 使用示例
StringBuilder version = new StringBuilder(32);
int ret = EcatBoardApi.EcatGetApiVersion(version, (uint)version.Capacity);
if (ret == 0) {
    Console.WriteLine($"API 版本: {version.ToString()}");  // 输出: API 版本: 1.0.0
}
```

注意事项:

- 缓冲区大小建议至少为 32 字节,以容纳未来可能的长版本号
- 如果缓冲区太小,函数会返回 ECAT_E_FAIL 错误
- 传入 NULL 指针或 bufferSize 为 0 会返回 ECAT_E_INVALIDARG 错误
- 版本号遵循语义化版本规范 (Semantic Versioning)
- 建议在程序启动时调用并记录版本号,便于问题追踪

4.2 总线控制

EcatBusRun

启动 ECAT 总线运行。

```
EXPORT int32_t EcatBusRun(IN ECAT_HANDLE hHandle, IN const char* cEniFileName);
```

参数:

- `hHandle` : ECAT 控制句柄 (输入参数)
- `cEniFileName` : ENI 配置文件路径 (输入参数)

返回值:

- ECAT_S_OK (0): 成功

- 非 0: 失败

ENI 文件说明:

- ENI (EtherCAT Network Information) 是 EtherCAT 网络配置文件
- 包含从站拓扑、PDO 映射、同步管理器配置等信息
- 由 EtherCAT 配置工具生成

使用示例:

```
const char* eniFile = "config/network.eni";
int32_t ret = EcatBusRun(hHandle, eniFile);
if (ret != ECAT_S_OK) {
    printf("启动总线失败,错误码: %d\n", ret);
    return -1;
}
printf("总线启动成功\n");
```

注意事项:

- 必须先调用 EcatOpen 打开设备
- ENI 文件路径必须是绝对路径或相对于工作目录的有效路径
- 启动后从站会自动进入状态机转换流程
- 可通过 EcatGetMasterState 查询当前状态

EcatBusStop

停止 ECAT 总线运行。

```
EXPORT int32_t EcatBusStop(IN ECAT_HANDLE hHandle);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用场景:

- 程序正常退出前停止总线
- 切换配置文件前停止当前总线
- 紧急情况下停止所有通信

使用示例:

```
EcatBusStop(hHandle);
printf("总线已停止\n");
```

注意事项:

- 停止后所有从站会回到 INIT 状态
- 停止操作是阻塞的,会等待当前周期完成
- 停止后可以再次调用 EcatBusRun 重新启动

EcatMasterActivate

激活主站实时循环。

```
EXPORT int32_t EcatMasterActivate(IN ECAT_HANDLE hHandle);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

功能说明:

- 启动周期性数据交换循环
- 必须在 EcatBusRun 之后调用
- 激活后才能进行实时数据交换

使用示例:

```
// 1. 启动总线
EcatBusRun(hHandle, "network.eni");


// 2. 激活实时循环
EcatMasterActivate(hHandle);
printf("实时循环已激活\n");

// 3. 等待从站进入 OP 状态
uint16_t state = 0;
do {
    EcatGetMasterState(hHandle, &state);
    usleep(100000);  // 等待 100ms
} while (state != EcatStateOP);
```

注意事项:

- 激活后检查确保所有从站已进入 OP 状态
- 不要在激活后进行耗时的非实时操作

4.3 状态查询

EcatGetSlaveCount

获得当前 ENI 配置下从站的数量。

```
EXPORT int32_t EcatGetSlaveCount(IN ECAT_HANDLE hHandle, OUT uint16_t *pu16Count);
```

参数:

- `hHandle` : ECAT 控制句柄 (输入参数)
- `pu16Count` : 从站的个数 (输出参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
uint16_t slaveCount = 0;
int32_t ret = EcatGetSlaveCount(hHandle, &slaveCount);
if (ret == ECAT_S_OK) {
    printf("配置从站数量: %d\n", slaveCount);
}
```

注意事项:

- 返回的是 ENI 文件中配置的从站数量
- 不代表实际在线的从站数量
- 必须在 EcatBusRun 之后调用

EcatGetActiveSlaveCount

获取当前网络中活跃的从站数量(支持热插拔检测)。

```
EXPORT int32_t EcatGetActiveSlaveCount(IN ECAT_HANDLE hHandle, OUT uint16_t
*pu16Count);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)
- `pu16Count`: 活跃从站的个数 (输出参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

与 EcatGetSlaveCount 的区别:

- EcatGetSlaveCount: 返回 ENI 配置的理论数量
- EcatGetActiveSlaveCount: 返回实际在线的从站数量

使用场景:

- 检测从站是否全部上线
- 诊断网络连接问题

使用示例:

```c
uint16_t configuredCount = 0;
uint16_t activeCount = 0;

EcatGetSlaveCount(hHandle, &configuredCount);
EcatGetActiveSlaveCount(hHandle, &activeCount);

if (activeCount < configuredCount) {
    printf("警告: 有 %d 个从站离线\n", configuredCount - activeCount);
}
```

EcatGetMasterState

获取所有从站AL状态。

```c
EXPORT int32_t EcatGetMasterState(IN ECAT_HANDLE hHandle, OUT uint16_t* pu16State);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)
- `pu16State`: 从站状态,参考 `EcatState` 枚举

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

状态位解析:

```c
uint16_t state = 0;
EcatGetMasterState(hHandle, &state);

// 有设备处于 SAFE-OP 状态
bool haveSafeop = (state & EcatStateSAFEOP) != 0;
// 有设备处于 OP 状态
bool hasOp = (state & EcatStateOP) != 0;
// 所有设备都处于 OP 状态
bool allOP = state == EcatStateOP;
```

典型状态序列:

```
0 (NotSet) -> 1 (INIT) -> 2 (PREOP) -> 4 (SAFE-OP) -> 8 (OP)
```

EcatGetSlaveALStatusCode

获取指定从站的 AL (Application Layer) 状态码。

```
EXPORT int32_t EcatGetSlaveALStatusCode(IN ECAT_HANDLE hHandle, IN uint16_t
u16SlaveId, OUT uint16_t* pu16ALStatusCode);
```

参数:

- `hHandle` : ECAT 控制句柄 (输入参数)
- `u16SlaveId` : 从站位置索引,范围: 0 ~ SlaveCount-1 (输入参数)
- `pu16ALStatusCode` : 存储从站的 AL 状态码 (输出参数)，参考 `EcatState` 枚举

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
uint16_t alStatus = 0;
int32_t ret = EcatGetSlaveALStatusCode(hHandle, 0, &alStatus);
if (ret == ECAT_S_OK && alStatus == EcatStateOP) {
    printf("从站 0 已进入OP状态\n");
}
```

注意事项:

- 可用于检查从站状态

4.4 SDO 通信 (CoE)

EcatCoeSDODownload

写从站 SDO (Service Data Object,服务数据对象)。

```
EXPORT int32_t EcatCoeSDODownload(
    IN ECAT_HANDLE      hHandle,
    IN uint16_t         u16SlaveId,
    IN uint16_t         u16Index,
    IN uint8_t          u8SubIndex,
    IN uint32_t         u32DataLen,
    IN const void       *pvData,
    IN uint32_t         u32TimeOutMs);
```

参数:

- `hHandle` : ECAT 控制句柄 (输入参数)
- `u16SlaveId` : 从站位置索引,范围: 0 ~ SlaveCount-1 (输入参数)
- `u16Index` : SDO 索引 (输入参数)
- `u8SubIndex` : SDO 子索引 (输入参数)
- `u32DataLen` : 数据的字节长度,通常为 1, 2, 4 (输入参数)
- `pvData` : 写入的数据指针,多字节须小端排列 (输入参数)
- `u32TimeOutMs` : 超时时间,以毫秒为单位 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用场景:

- 配置从站参数
- 修改从站工作模式
- 写入控制命令

使用示例:

```
// 示例 1: 写入 8 位数据
uint8_t value8 = 0x01;
EcatCoeSDODownload(hHandle, 0, 0x6040, 0, 1, &value8, 1000);

// 示例 2: 写入 16 位数据 (小端)
uint16_t value16 = 0x0102;  // 低位在前
EcatCoeSDODownload(hHandle, 0, 0x6040, 0, 2, &value16, 1000);

// 示例 3: 写入 32 位数据
uint32_t value32 = 0x01020304;
EcatCoeSDODownload(hHandle, 0, 0x6040, 0, 4, &value32, 1000);
```

注意事项:

- SDO 通信用于非实时参数配置,不适合高频调用
- 数据必须以小端格式排列
- 超时时间应根据网络负载合理设置

EcatCoeSDOUpload

读从站 SDO (Service Data Object,服务数据对象)。

```
EXPORT int32_t EcatCoeSDOUpload(
    IN ECAT_HANDLE      hHandle,
    IN uint16_t         u16SlaveId,
    IN uint16_t         u16Index,
    IN uint8_t          u8SubIndex,
    IN OUT uint32_t    *pu32DataLen,
    OUT void           *pvData,
    IN uint32_t         u32TimeOutMs);
```

参数:

- `hHandle` : ECAT 控制句柄 (输入参数)
- `u16SlaveId` : 从站位置索引,范围: 0 ~ SlaveCount-1 (输入参数)
- `u16Index` : SDO 索引 (输入参数)
- `u8SubIndex` : SDO 子索引 (输入参数)
- `pu32DataLen` : 输入为缓冲区最大长度,输出为读取到的有效字节数 (输入输出参数)
- `pvData` : 存储读取到的数据,多字节须小端排列 (输出参数)
- `u32TimeOutMs` : 超时时间,以毫秒为单位 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用场景:

- 读取从站状态信息
- 查询从站配置参数
- 读取设备版本信息

使用示例:

```
// 示例 1: 读取 32 位数据
uint32_t data32 = 0;
uint32_t len = sizeof(data32);
int32_t ret = EcatCoeSDOUpload(hHandle, 0, 0x1018, 1, &len, &data32, 1000);
if (ret == ECAT_S_OK) {
    printf("读取成功,数据: 0x%08X, 长度: %d\n", data32, len);
}

// 示例 2: 读取字符串 (厂商名称)
char vendorName[64] = {0};
uint32_t strLen = sizeof(vendorName);
EcatCoeSDOUpload(hHandle, 0, 0x1008, 0, &strLen, vendorName, 1000);
printf("厂商名称: %s\n", vendorName);
```

注意事项:

- 调用前必须初始化 `*pu32DataLen` 为缓冲区大小
- 返回后 `*pu32DataLen` 会被更新为实际读取的字节数
- 读取的数据以小端格式存储
- 不要超过缓冲区大小,避免内存溢出

4.5 数据交换

EcatPINMap

获取镜像缓存的内存映射地址,用于高速访问。

```
EXPORT int32_t EcatPINMap(
    IN ECAT_HANDLE      hHandle,
    IN uint32_t         u32PiArea,
    OUT void** ppvData,
    OUT uint32_t* size);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)
- `u32PiArea`: 镜像的区域索引,支持 `PI_AREA_LOCAL_OUTPUT`、`PI_AREA_LOCAL_INPUT` (输入参数)
- `ppvData`: 返回 输入或输出的地址 (输出参数)
- [size]: 返回 输入或输出的大小 (输出参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
void* pOutput = NULL;
void* pInput = NULL;
uint32_t outputSize = 0;
uint32_t inputSize = 0;

// 映射输出缓存
EcatPINMap(hHandle, PI_AREA_LOCAL_OUTPUT, &pOutput, &outputSize);

// 映射输入缓存
EcatPINMap(hHandle, PI_AREA_LOCAL_INPUT, &pInput, &inputSize);

printf("输出缓存: %p, 大小: %d 字节\n", pOutput, outputSize);
printf("输入缓存: %p, 大小: %d 字节\n", pInput, inputSize);

// 直接读写数据
if (pOutput != NULL) {
    EC_WRITE_U32((uint8_t*)pOutput + 0, 1000);    // 写入位置 0
    EC_WRITE_U16((uint8_t*)pOutput + 4, 500);     // 写入位置 4
}
```

注意事项:

- **非线程安全**: 多线程访问必须加锁
- 必须在 EcatPIInputQueuePop 之后调用

- 返回的指针在设备关闭前一直有效
- 不要越界访问,否则会导致崩溃
- 建议使用辅助宏 (EC_READ_XXX, EC_WRITE_XXX) 进行数据读写

EcatPIInputQueuePop

从ECAT主站获取输入缓存。

```
EXPORT int32_t EcatPIInputQueuePop(
    IN ECAT_HANDLE      hHandle,
    IN bool             bReqClearRemain,
    IN uint32_t         u32TimeOutMs);
```

参数:

- `hHandle` : ECAT 控制句柄 (输入参数)
- `bReqClearRemain` : 保留
- `u32TimeOutMs` : 超时时间,以毫秒为单位 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
while (EcatPIInputQueuePop(dev, 0, 1000) != 0);
// 处理 输入输出 中的数据
processData();
```

注意事项:

- 先通过 EcatPINMap 获取的指针读取数据
- 超时时间内无数据会返回超时错误

EcatPIOutputQueuePush

更新输出缓存到ECAT主站。

```
EXPORT int32_t EcatPIOutputQueuePush(
    IN ECAT_HANDLE      hHandle,
    IN bool             bReqClearQueue,
    IN uint32_t         u32TimeOutMs);
```

参数:

- `hHandle`: ECAT 控制句柄 (输入参数)
- `bReqClearQueue`: 保留
- `u32TimeOutMs`: 超时时间,以毫秒为单位 (输入参数)

返回值:

- ECAT_S_OK (0): 成功
- 非 0: 失败

使用示例:

```
// 准备输出数据
if (pOutput != NULL) {
    EC_WRITE_U32((uint8_t*)pOutput + 0, position);
    EC_WRITE_U16((uint8_t*)pOutput + 4, velocity);
}

// 推送到输出队列
EcatPIOutputQueuePush(hHandle, false, 500);
```

---

5. 辅助宏定义

为了方便对 EtherCAT 数据进行读写,库提供了一系列宏定义。

5.1 数据读取宏

宏名称	功能	数据类型	示例
`EC_READ_U8(DATA)`	读取 8 位无符号数	`uint8_t`	`uint8_t val = EC_READ_U8(ptr);`
`EC_READ_S8(DATA)`	读取 8 位有符号数	`int8_t`	`int8_t val = EC_READ_S8(ptr);`
`EC_READ_U16(DATA)`	读取 16 位无符号数	`uint16_t`	`uint16_t val = EC_READ_U16(ptr);`
`EC_READ_S16(DATA)`	读取 16 位有符号数	`int16_t`	`int16_t val = EC_READ_S16(ptr);`
`EC_READ_U32(DATA)`	读取 32 位无符号数	`uint32_t`	`uint32_t val = EC_READ_U32(ptr);`
`EC_READ_S32(DATA)`	读取 32 位有符号数	`int32_t`	`int32_t val = EC_READ_S32(ptr);`
`EC_READ_U64(DATA)`	读取 64 位无符号数	`uint64_t`	`uint64_t val = EC_READ_U64(ptr);`
`EC_READ_S64(DATA)`	读取 64 位有符号数	`int64_t`	`int64_t val = EC_READ_S64(ptr);`

5.2 数据写入宏

宏名称	功能	数据类型	示例
`EC_WRITE_U8(DATA, VAL)`	写入 8 位无符号数	`uint8_t`	`EC_WRITE_U8(ptr, 0xFF);`
`EC_WRITE_S8(DATA, VAL)`	写入 8 位有符号数	`int8_t`	`EC_WRITE_S8(ptr, -1);`
`EC_WRITE_U16(DATA, VAL)`	写入 16 位无符号数	`uint16_t`	`EC_WRITE_U16(ptr, 1000);`
`EC_WRITE_S16(DATA, VAL)`	写入 16 位有符号数	`int16_t`	`EC_WRITE_S16(ptr, -500);`
`EC_WRITE_U32(DATA, VAL)`	写入 32 位无符号数	`uint32_t`	`EC_WRITE_U32(ptr, 0x12345678);`
`EC_WRITE_S32(DATA, VAL)`	写入 32 位有符号数	`int32_t`	`EC_WRITE_S32(ptr, -10000);`
`EC_WRITE_U64(DATA, VAL)`	写入 64 位无符号数	`uint64_t`	`EC_WRITE_U64(ptr, 0xFFFFFFFFFFFFFFFF);`
`EC_WRITE_S64(DATA, VAL)`	写入 64 位有符号数	`int64_t`	`EC_WRITE_S64(ptr, -1LL);`

5.3 使用示例

```c
#include "pcie_ecat.h"

void processPIData(void* pInput, void* pOutput) {
    // 从输入缓存读取数据
    uint16_t status = EC_READ_U16((uint8_t*)pInput + 0);
    int32_t position = EC_READ_S32((uint8_t*)pInput + 2);
    int16_t velocity = EC_READ_S16((uint8_t*)pInput + 6);

    printf("状态: %d, 位置: %d, 速度: %d\n", status, position, velocity);

    // 计算控制量
    int32_t targetPosition = position + 100;
    int16_t targetVelocity = 500;

    // 写入输出缓存
    EC_WRITE_S32((uint8_t*)pOutput + 0, targetPosition);
    EC_WRITE_S16((uint8_t*)pOutput + 4, targetVelocity);
    EC_WRITE_U16((uint8_t*)pOutput + 6, 0x000F);  // 控制字
}
```

注意事项:

- 确保指针偏移量不超出缓存边界

---

6. 错误码参考

具体的错误码定义请参考 `pcie_errno.h` 文件。

6.1 成功状态码 (0~999)

错误码	名称	描述
0	`ECAT_S_OK`	操作成功
1	`ECAT_S_SIZE_LIMIT`	数据大小限制
2	`ECAT_S_NOITEMS`	无数据项
5	`ECAT_S_TIMEOUT`	操作超时

6.2 PCIe 驱动层错误 (10000~10999)

错误码	名称	描述
10000	`ECAT_E_TASKID`	任务 ID 错误
10001	`ECAT_E_TASK_INIT`	任务初始化失败
10010	`ECAT_E_NOT_FOUND_BOARD`	未找到板卡

6.3 EtherCAT 主站层错误 (11000~11999)

错误码	名称	描述
11000	`ECAT_E_MASTER_NONE_DOMAIN`	主站无域
11002	`ECAT_E_INVALID_PARAM`	无效参数
11005	`ECAT_E_SLAVE_NOT_FOUND`	未找到从站

6.4 通用系统错误 (32768+)

错误码	名称	描述
32768	`ECAT_E_OUTOFMEMORY`	内存不足
32769	`ECAT_E_INVALIDARG`	无效参数
32770	`ECAT_E_FAIL`	一般性失败
32784	`ECAT_E_SEND_FRAME_FAILED`	发送帧失败
32785	`ECAT_E_RECV_FRAME_FAILED`	接收帧失败
32848	`ECAT_E_OPEN_FILE`	打开文件失败
32850	`ECAT_E_READ_FILE`	读取文件失败

6.5 错误处理最佳实践

```c
int32_t ret = EcatOpen(&hHandle, 0);
if (ret != ECAT_S_OK) {
    if (ECAT_IS_PCI_ERROR(ret)) {
        printf("PCIe 驱动层错误: %d\n", ret);
    } else if (ECAT_IS_MASTER_ERROR(ret)) {
        printf("EtherCAT 主站层错误: %d\n", ret);
    } else {
        printf("通用错误: %d\n", ret);
    }
    return -1;
}
```

---

7. 完整使用示例

7.1 基础通信流程

```c
#include <stdio.h>
#include <stdint.h>
#include "pcie_ecat.h"

int main() {
    ECAT_HANDLE hHandle = NULL;
    int32_t ret = 0;

    // 1. 打开设备
    ret = EcatOpen(&hHandle, 0);
    if (ret != ECAT_S_OK) {
        printf("打开设备失败: %d\n", ret);
        return -1;
    }
    printf("设备打开成功\n");

    // 2. 设置超时时间
    EcatSetTimeout(hHandle, 2000);

    // 3. 启动总线
    ret = EcatBusRun(hHandle, "config/network.eni");
    if (ret != ECAT_S_OK) {
        printf("启动总线失败: %d\n", ret);
        EcatClose(hHandle);
        return -1;
    }
    printf("总线启动成功\n");

    // 4. 激活实时循环
    ret = EcatMasterActivate(hHandle);
    if (ret != ECAT_S_OK) {
        printf("激活实时循环失败: %d\n", ret);
        EcatBusStop(hHandle);
        EcatClose(hHandle);
        return -1;
    }
    printf("实时循环已激活\n");

    // 5. 等待从站进入 OP 状态
    uint16_t state = 0;
    int timeout = 50;  // 最多等待 5 秒
    while (timeout-- > 0) {
        EcatGetMasterState(hHandle, &state);
        if (state == EcatStateOP) {
            printf("所有从站已进入 OP 状态\n");
            break;
        }
        usleep(100000);  // 等待 100ms
    }
```

```
    if (state != EcatStateOP) {
        printf("警告: 从站未完全进入 OP 状态\n");
    }



    // 6. 获取映射
    void* pOutput = NULL;
    void* pInput = NULL;
    uint32_t outputSize = 0;
    uint32_t inputSize = 0;

    EcatPINMap(hHandle, PI_AREA_LOCAL_OUTPUT, &pOutput, &outputSize);
    EcatPINMap(hHandle, PI_AREA_LOCAL_INPUT, &pInput, &inputSize);

    printf("输出缓存: %p (%d 字节)\n", pOutput, outputSize);
    printf("输入缓存: %p (%d 字节)\n", pInput, inputSize);

    // 7. 实时数据交换循环
    int running = 1;
    while (running) {
        // 获取 输入输出数据
         while (EcatPIInputQueuePop(dev, 0, 5000) != 0);

        // 读取输入数据
        int32_t actualPos = EC_READ_S32((uint8_t*)pInput + 0);
        int16_t actualVel = EC_READ_S16((uint8_t*)pInput + 4);
        uint16_t status = EC_READ_U16((uint8_t*)pInput + 6);

        printf("位置: %d, 速度: %d, 状态: 0x%04X\n",
                actualPos, actualVel, status);
        EC_WRITE_S32((uint8_t*)pOutput + 0, 1000);   // 目标位置
        EC_WRITE_S16((uint8_t*)pOutput + 4, 500);    // 目标速度

        // 推送输入输出数据
        EcatPIOutputQueuePush(dev, 0, 500);
    }

    // 8. 停止总线
    EcatBusStop(hHandle);
    printf("总线已停止\n");

    // 9. 关闭设备
    EcatClose(hHandle);
    hHandle = NULL;
    printf("设备已关闭\n");

    return 0;
}
```

7.2 SDO 配置示例

```c
#include "pcie_ecat.h"

// 配置驱动器参数
int configureDrive(ECAT_HANDLE hHandle, uint16_t slaveId) {
    int32_t ret = 0;

    // 1. 读取驱动器版本信息
    uint32_t version = 0;
    uint32_t len = sizeof(version);
    ret = EcatCoeSDOUpload(hHandle, slaveId, 0x1018, 4, &len, &version, 1000);
    if (ret != ECAT_S_OK) {
        printf("读取版本失败: %d\n", ret);
        return ret;
    }
    printf("驱动器版本: 0x%08X\n", version);

    // 2. 配置控制字
    uint16_t controlWord = 0x0006;  // Shutdown
    ret = EcatCoeSDODownload(hHandle, slaveId, 0x6040, 0, 2, &controlWord, 1000);
    if (ret != ECAT_S_OK) {
        printf("写入控制字失败: %d\n", ret);
        return ret;
    }

    // 3. 配置目标位置模式
    uint32_t modesOfOperation = 1;  // Profile Position Mode
    ret = EcatCoeSDODownload(hHandle, slaveId, 0x6060, 0, 1, &modesOfOperation, 1000);
    if (ret != ECAT_S_OK) {
        printf("配置操作模式失败: %d\n", ret);
        return ret;
    }

    printf("驱动器配置完成\n");
    return ECAT_S_OK;
}
```

7.3 C# 调用示例

```
using System;
using System.Runtime.InteropServices;

public class EcatBoardApi
{
    // 导入动态库
    const string DLL_NAME = "PCIE_ECAT.dll";

    [DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
    public static extern int EcatOpen(out IntPtr phHandle, byte board_id);

    [DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
    public static extern int EcatClose(IntPtr hHandle);

    [DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
    public static extern int EcatBusRun(IntPtr hHandle, string cEniFileName);

    [DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
    public static extern int EcatGetApiVersion(System.Text.StringBuilder pBuffer, uint
bufferSize);

    [DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
    public static extern void EcatSetLogEnabled(bool enable);

    [DllImport(DLL_NAME, CallingConvention = CallingConvention.Cdecl)]
    public static extern bool EcatIsLogEnabled();
}

public class Program
{
    public static void Main(string[] args)
    {
        // 获取并显示 API 版本
        System.Text.StringBuilder version = new System.Text.StringBuilder(32);
        int verRet = EcatBoardApi.EcatGetApiVersion(version, (uint)version.Capacity);
        if (verRet == 0) {
            Console.WriteLine($"PCIE_ECAT API 版本: {version.ToString()}");
        }

        // 启用日志
        EcatBoardApi.EcatSetLogEnabled(true);
        Console.WriteLine($"日志状态: {EcatBoardApi.EcatIsLogEnabled()}");

        // 打开设备
        IntPtr hHandle;
        int ret = EcatBoardApi.EcatOpen(out hHandle, 0);
        if (ret != 0)
        {
            Console.WriteLine($"打开设备失败: {ret}");
            return;
```

```
        }

        Console.WriteLine("设备打开成功");

        // 启动总线
        ret = EcatBoardApi.EcatBusRun(hHandle, "config/network.eni");
        if (ret != 0)
        {
            Console.WriteLine($"启动总线失败: {ret}");
            EcatBoardApi.EcatClose(hHandle);
            return;
        }

        Console.WriteLine("总线启动成功");

        // ... 业务逻辑 ...

        // 关闭设备
        EcatBoardApi.EcatClose(hHandle);
        Console.WriteLine("设备已关闭");
    }
}
```

---

8. 常见问题与排查

8.1 设备无法打开

症状: EcatOpen 返回错误码 10010 (ECAT_E_NOT_FOUND_BOARD)

可能原因:

1. PCIe 驱动未正确安装
2. 硬件连接异常
3. `board_id` 与拨码开关不一致

解决方法:

- 检查设备管理器中是否识别到 PCIe 设备
- 确认拨码开关设置与代码中的 `board_id` 一致
- 重新安装 PCIe 驱动程序

8.2 从站无法进入 OP 状态

症状: 调用 EcatGetMasterState 返回的状态长时间停留在 PREOP 或 SAFE-OP

可能原因:

1. ENI 配置文件错误
2. 从站硬件故障
3. 接线问题

解决方法:

- 使用 EcatGetSlaveALStatusCode 查询从站 AL 状态码
- 检查 ENI 文件中的 PDO 映射是否正确
- 检查 EtherCAT 网线连接和终端电阻
- 查看日志输出中的错误信息

8.3 SDO 通信超时

症状: EcatCoeSDOUpload 或 EcatCoeSDODownload 返回超时错误

可能原因:

1. 从站地址错误
2. SDO 索引/子索引不存在

解决方法:

- 确认 `u16SlaveId` 在有效范围内
- 查阅从站手册确认 SDO 索引有效性
- 增加超时时间 `u32TimeOutMs`
- 降低 SDO 访问频率

8.4 数据异常

症状: 读取到的数据与实际不符

可能原因:

1. 偏移量计算错误
2. 字节序问题
3. 未正确调用 Pop/Push

解决方法:

- 仔细核对 PDO 映射表中的偏移量
- 使用辅助宏 (EC_READ_XXX, EC_WRITE_XXX) 避免字节序问题
- 确保每次循环都调用了 EcatPlInputQueuePop 和 EcatPlOutputQueuePush
- 检查是否越界访问

---

9. 附录

9.1 相关文档

- `pcie_errno.h` : 错误码定义

9.2 技术支持

如遇问题,请提供以下信息:

1. 错误码及调用堆栈
2. 日志输出内容
3. 硬件配置 (拨码开关、从站型号等)
4. ENI 配置文件
5. 复现步骤

文档版本: v1.1
最后更新: 2026-04-23
适用库版本: PCIE_ECAT v1.0.1