Application Note

插板式模组应用

插板式 EtherCAT 系列从站

AN01010101 1.00 Date:2025/10/24

类别	内容
关键词	插板式EtherCAT模组、模拟量、PCIe-EtherCAT主站卡、C/C++
摘要	本文介绍了使用致远PCIe主站卡C/C++编程搭配插板式模拟量模块的应用示例

插板式模组应用

插板式 EtherCAT 系列从站

修订历史

版本	日期	原因
V1.00	2025/10/24	•创建文档

插板式模组应用

插板式 EtherCAT 系列从站

目　录

插板式模组应用

插板式 EtherCAT 系列从站

1. 适用范围

本文档适用于搭配致远 PCIe 主站卡 C/C++编程搭配插板式模拟量从站模组的应用。

产品概述

为满足个性化大型系统的控制需求，致远电子推出了插板式模组。该系统采用 EtherCAT 总线，尺寸小巧，采用 40 脚标准排针接口。用户按需制作分线底板，从站板插在底板上通过 EtherCAT 网络级联，最大支持 255 个节点。

- **以不变应万变**，只需最少的工作量，按需制作分线底板；电机驱动、数字量、模拟量、编码器等多种模组选择组合。
- **小体积大系统**，在最小的体积下集成最多的从站，实现大型系统的控制。
- **高精度快布局**，基于 EtherCAT，实现高精度分布控制，以及即插即用快速布局。

产品应用

- 物流装备
- 电子制造
- 新能源
- 纺织
- 包装

2. 原理概述

PCIe-EtherCAT 通讯卡提供了 C/C++接口的 SDK 动态库。用户只需要调用动态库提供的接口编写程序，就可以控制板卡运行 EtherCAT 主站。在主站程序中操作 PDO 数据、请求 SDO 读写命令，就可以控制从站设备。

本文档介绍如何调用 PCIe 主站卡的 SDK 接口来控制 EtherCAT 总线。本示例采用主站的同步队列接口交换数据。同步队列的通信原理如图 2.1 所示。

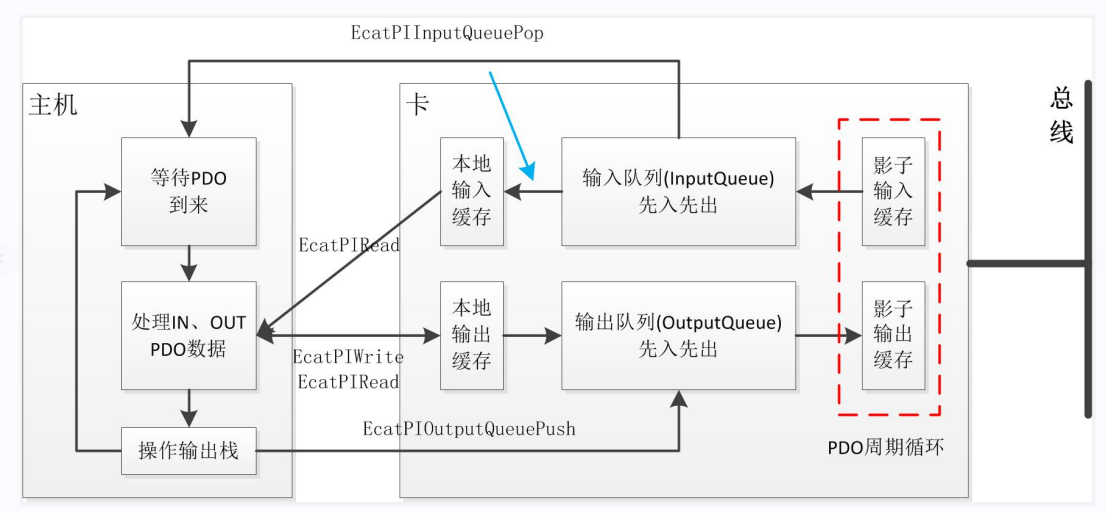

图 2.1　同步队列通信原理

3. 开发环境

（1）HPCIe-2E 主站卡及带 PCIe 卡槽的开发主机（Windows 或 Linux），主站卡如下图 3.1 所示。

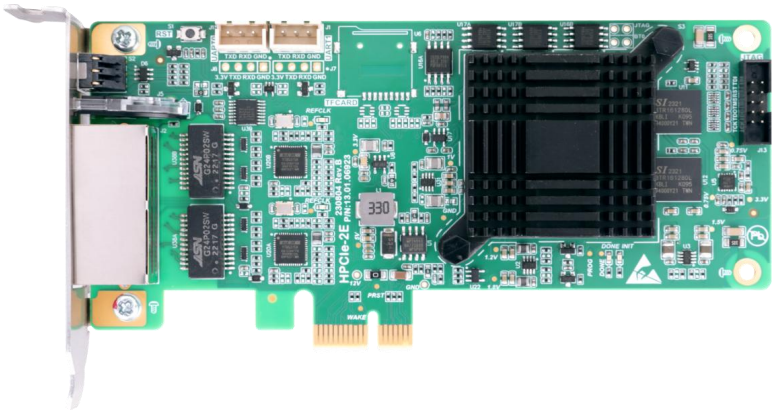

图 3.1 PCIe-2E 主站卡

（2）ZCPC-80801 耦合器、ZIOC-E0800AI（0-20mA 电流输入）、ZIOC-E0004AI（0-20mA 电流输出）、ZIOC-E0800AI1（4-20mA 电流输入）、ZIOC-E0004AI（4-20mA 电流输出）、ZIOC-E0800AU1（2x±10V+6x0-10V 电压输入）、ZIOC-E0008AU（0-10V 电压输出）插板式模块及 ZMB-IO-8 分线底板，如图 3.2 所示。

图 3.2 插板式模组

（3）安装 AWStudio 运动控制板软件的 PC 一台（Windows 系统）。

（4）网线及 24V 电源。

（5）Visual Studio 2015。

4. 实现过程

4.1 设备连接

如图 3.2 所示，将 ZCPC-80801、ZIOC-E0800AI、ZIOC-E0004AI、ZIOC-E0800AI1、ZIOC-E0004AI1、ZIOC-E0800AU1、ZIOC-E0008AU 模块从左到右分别插入到 ZMB-IO-8 分线底板。

分别在 EtherCAT 24V 端子和系统 24V 端子接上两路独立的 24V 电源，对于简单测试也可共用一组 24V 电源，底板的大地 EARTH 可以悬空。

将 ZIOC-E0004AI 的输出通道 1 接入到 ZIOC-E0800AI 输入 1、2 通道（两个输入通道约占一半电流），ZIOC-E0004AI 的通道 2 接入到 ZIOC-E0800AI 输入 3、4 通道，以此类推。ZIOC-E0004AI1 和 ZIOC-E0004AI 的一样。ZIOC-E0800AU1 和 ZIOC-E0008AU 分别对应连接了 1-4 通道，剩余接口悬空不接。

用网线连接 HPCIe-2E 主站卡的 ECAT 口和 ZCPC-80801 耦合器的 IN 口，给设备通上 24V 电源。

4.2 ENI 文件生成

本节介绍通过 AWStudio 运动控制版配置上位机生成 ENI 文件。

4.2.1 新建解决方案

AWStudio 启动后默认没有解决方案，需要先新建或打开一个解决方案，操作界面如图 4.1 所示。

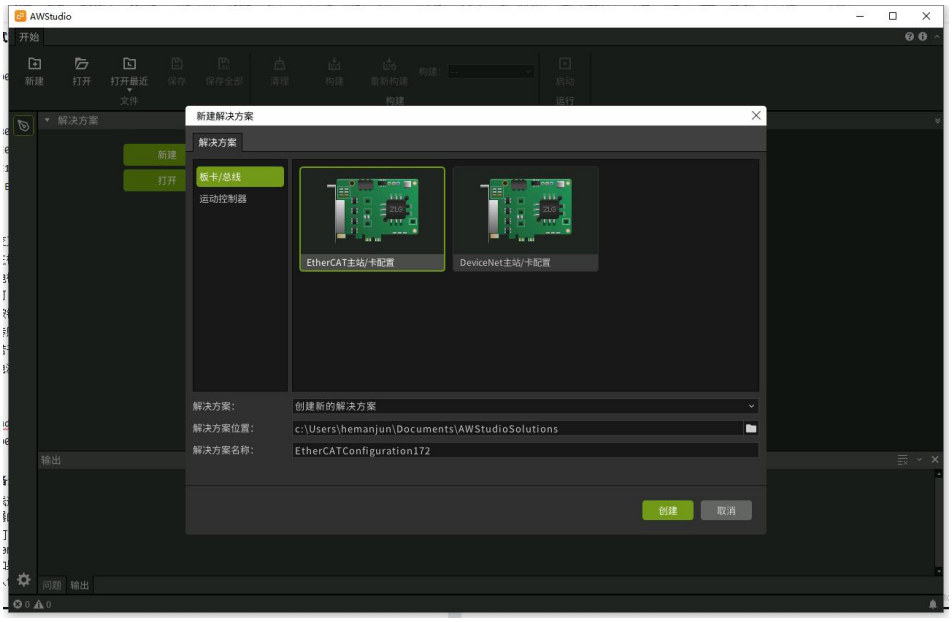

图 4.1 新建解决方案

4.2.2 连接主站

如果需要在线监控、在线扫描等功能，需要先连接主站，操作界面如图 4.2 所示。

图 4.2 连接主站

4.2.3 添加从站

可以通过在线扫描或手动添加的方式添加从站，通过在线扫描的方式，可以先连接主站，然后用拓扑扫描功能，操作界面如图 4.3 所示。

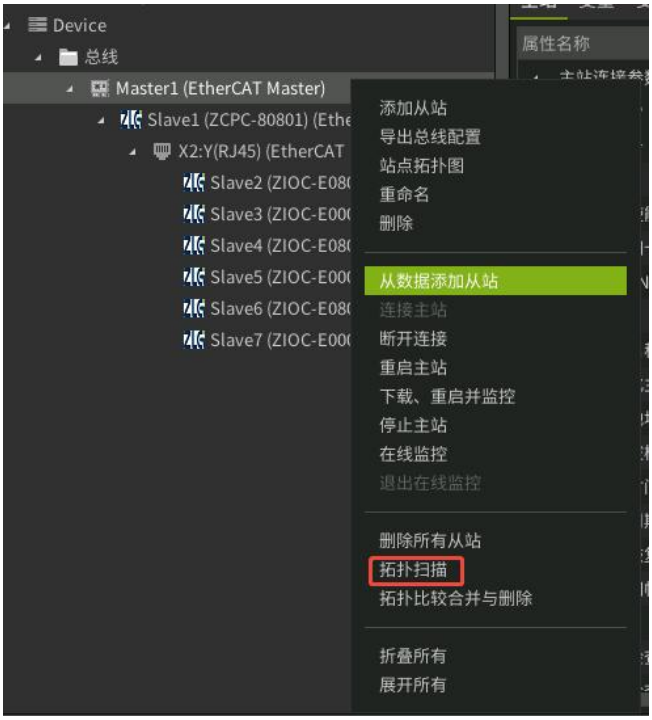

图 4.3 拓扑扫描

成功添加从站后如图 4.4 所示。

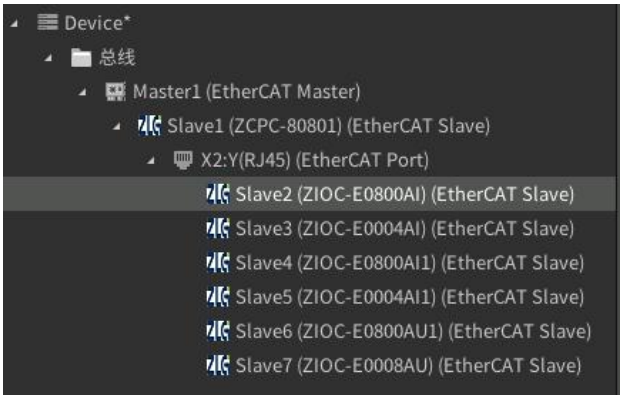

图 4.4 成功扫描从站

4.2.4 配置从站的 PDO 过程数据

双击各从站，打开“从站配置-变量页”，可以查看厂商 ESI 默认配置下的 PDO 变量，如图 4.5 所示。默认变量不一定能满足我们的输入输出需求。例如 ZIOC-E0800AI 默认输入变量是 Code 编码的形式。

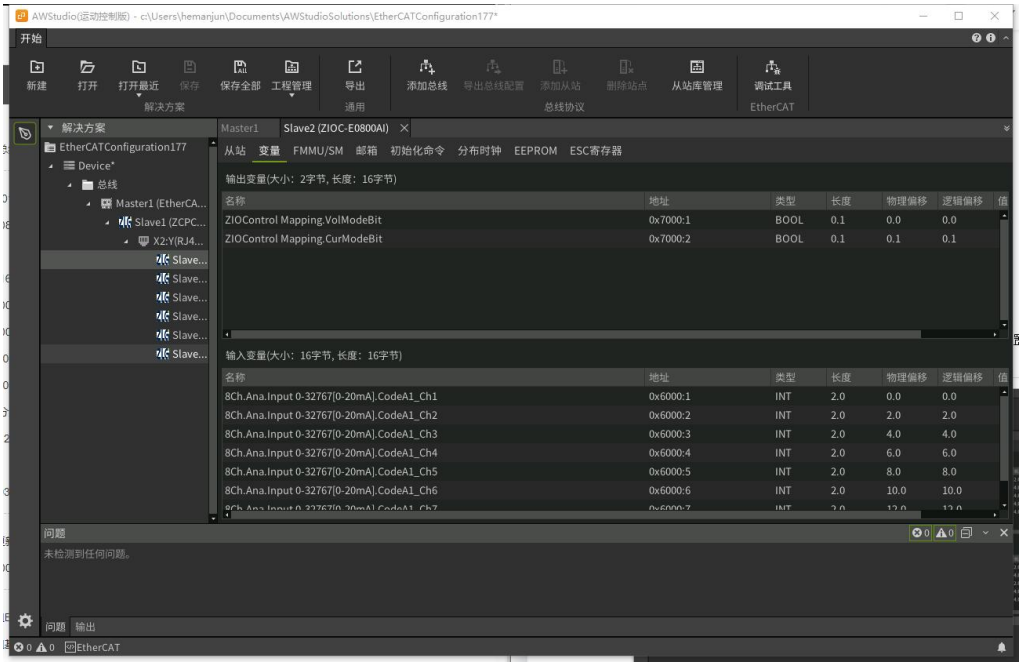

图 4.5 查看 PDO 变量

ZIOC-E0800AI 也支持内部将 Code 编码转换为浮点数，将浮点数返回给主站。

打开“从站配置-FMMU/SM”，画面大致分为了 4 个区域。单击左上“同步管理器配置-SM3:Inputs”，在左下“PDO 分配”可以看到输入 SM 允许的配置项，包括了 0x1A00 的 Code 形式输入，0x1A01 的浮点数形式输入，如图 4.6 所示。

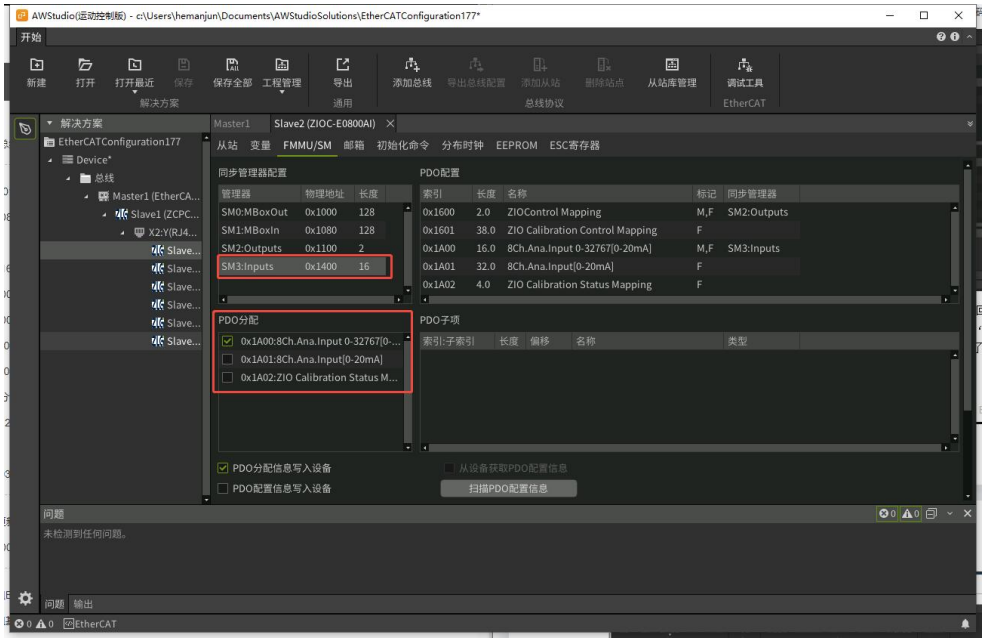

图 4.6 查看输入类型

将勾选改为 0x1A01，并勾选下方“PDO 分配信息写入设备”，即可将配置修改为浮点数输入，如图 4.7 所示。

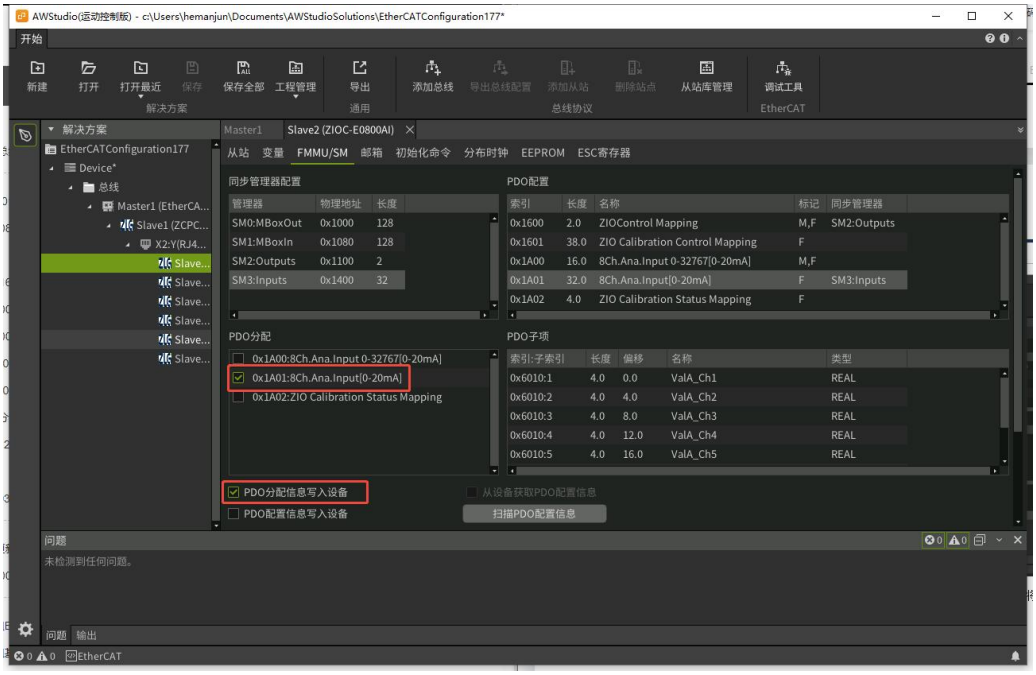

图 4.7 修改为浮点数输入

返回“变量页”查看从站数据，发现 PDO 数据结构已经发生了变化，如图 4.8 所示。

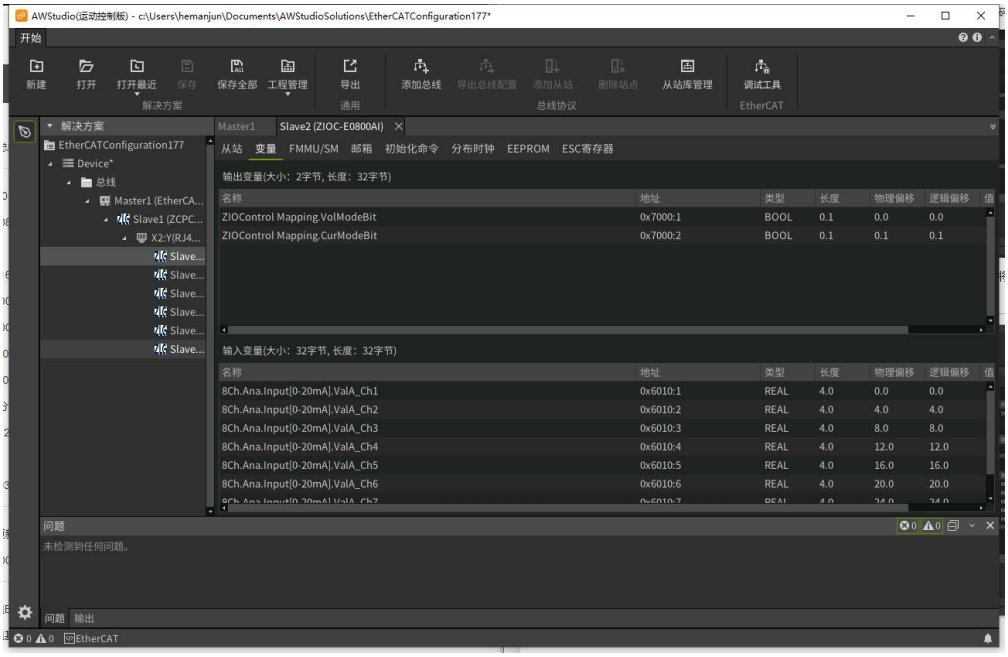

图 4.8 查看 PDO 数据结构已改变

为了减少用户使用困难，本示例介绍将所有从站输入输出修改为浮点数的方式，避免用户在程序中手动计算编码与电压/电流的关系。用户可以根据下表 4-1 所示手动修改所有从站的 FMMU/SM 配置。

注意每个从站都要勾选“PDO 分配信息写入设备”。

表 4-1 各从站信息

名称	SM2:Outputs	SM3:Inputs	功能
ZCPC-80801	-	-	耦合器
ZIOC-E0800AI	0x1600	0x1A01	8 通道电流型模拟输入（0~20mA）
ZIOC-E0004AI	0x1601	0x1A00	4 通道电流型模拟输出（0~20mA）
ZIOC-E0800AI1	0x1600	0x1A01	8 通道电流型模拟输入（4~20mA）
ZIOC-E0004AI1	0x1601	0x1A00	4 通道电流型模拟输出（4~20mA）
ZIOC-E08000AU1	0x1600	0x1A0A	8 通道电压型模拟输入（+/-10V 模式）
ZIOC-E0008AU	0x160A	0x1A00	8 通道电压型模拟输出（0~10V 模式）

4.2.5 导出 ENI

同样配置其他主站和从站参数，完成后导出 ENI 文件，操作如图 4.9 所示。

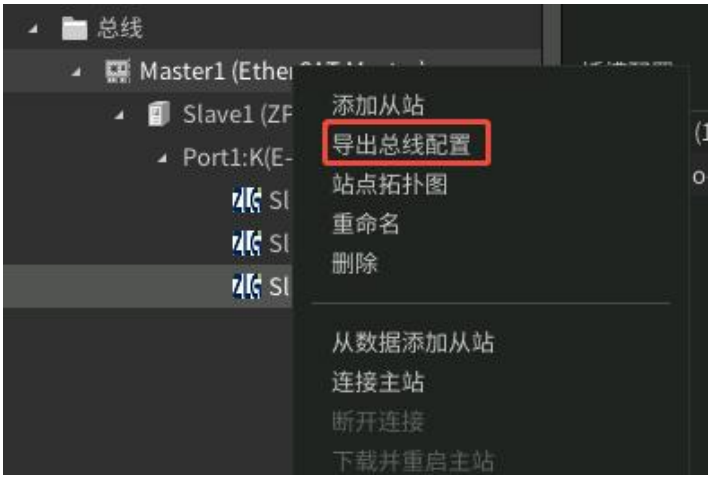

图 4.9 导出总线配置

4.3 过程数据 PDO

根据主站配置和网络拓扑结构，我们需要定义当前拓扑下使用的 PDO 数据结构。打开“主站配置-变量”页，查看当前网络拓扑下的数据结构，如图 4.10 所示。

插板式模组应用

插板式 EtherCAT 系列从站

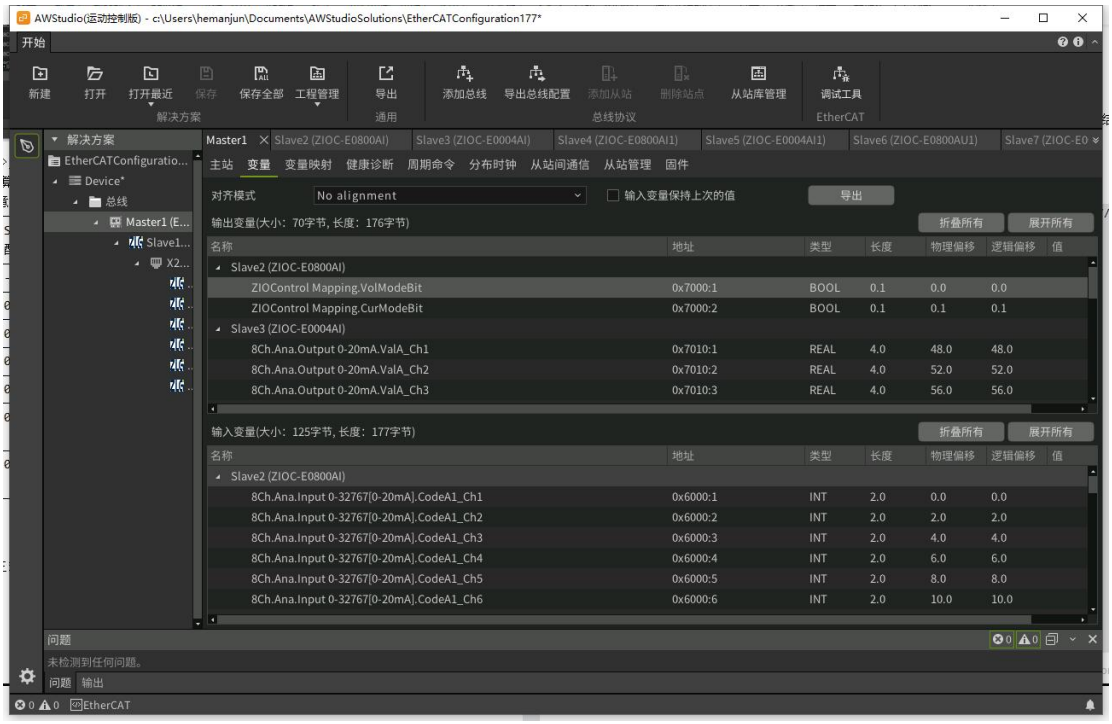

图 4.10 查看当前拓扑数据结构

结合从站手册的定义以及 AWStudio 配置上位机的配置，可以定义为以下的结构。

```
////////////////////////////////////////////////////////////////////////////////
// 过程映像
#pragma pack(push, 1)

typedef struct {
    uint8_t mapping[2];
    uint8_t pad[30];
} ZIOC_E0800_o;

typedef struct {
    float val[8];
} ZIOC_E0800_i;

typedef struct {
    uint8_t mapping[4];
    uint8_t pad[28];
} ZIOC_E0008_i;

typedef struct {
    float val[8];
} ZIOC_E0008_o;

typedef struct {
```

```
    uint8_t mapping[4];
    uint8_t pad[12];
} ZIOC_E0004_i;

typedef struct {
    float val[4];
} ZIOC_E0004_o;

typedef struct {
    ZIOC_E0800_o zioc_e0800ai;
    ZIOC_E0004_o zioc_e0004ai;
    ZIOC_E0800_o zioc_e0800ai1;
    ZIOC_E0004_o zioc_e0004ai1;
    ZIOC_E0800_o zioc_e0800au1;
    ZIOC_E0008_o zioc_e0008au;
} proc_img_o;

typedef struct {
    ZIOC_E0800_i zioc_e0800ai;
    ZIOC_E0004_i zioc_e0004ai;
    ZIOC_E0800_i zioc_e0800ai1;
    ZIOC_E0004_i zioc_e0004ai1;
    ZIOC_E0800_i zioc_e0800au1;
    ZIOC_E0008_i zioc_e0008au;
} proc_img_i;

#pragma pack(pop)
```

4.4 主站控制

4.4.1 初始化主站

首先启动 EtherCAT，其中 ALIAS 需要填入主站卡的别名编码器值，“protal3.xml”需要替换为用户通过 AWStudio 导出到 ENI.xml 文件。

```
//PCIe-2E 只有一个通道,PCIe-4E 有两个通道
RETURN_IF_FAILED(EcatOpen(&ecat_dev, BOARD_ALIAS(index_buff, index), channel));
RETURN_IF_FAILED(EcatBusRun(ecat_dev, "protal3.xml"));
```

4.4.2 EtherCAT 状态切换

请求 EtherCAT 总线切换状态。

```
// 请求切换状态
uint8_t state;
RETURN_IF_FAILED(EcatRequestMasterState(ecat_dev, EcatStateO));
for (uint32_t i = 0; ; i++)
{
```

```
        RETURN_IF_FAILED(EcatGetMasterState(ecat_dev, &state));
        _INF_("request is: 8, curr is: %d", state);
        usleep(500*1000);
        if (state == EcatStateO) break;
    }
```

4.4.3 获取 PI 镜像指针

读写 PDO 变量，首先需要通过接口获取 PI 镜像。为了加快用户对输入输出缓存操作的速度，可使用内存映射的方式来直接操作缓存，以减少 IO 的操作时间。然后通过 pi_get_input 和 pi_get_output 获取变量指针。

```
// 清空并预计填入数据
    proc_img_i *pi;
    proc_img_o *po;
    RETURN_IF_FAILED(EcatPINMap(ecat_dev, PI_AREA_LOCAL_INPUT, (void **)&pi));
     _INF_("EcatPINMap done");

    RETURN_IF_FAILED(EcatPINMap(ecat_dev, PI_AREA_LOCAL_OUTPUT, (void **)&po));
     _INF_("EcatPINMap done");

     _INF_("pi: 0x%llx", pi);
     _INF_("po: 0x%llx", po);
```

4.4.4 使能 PI 总线数据交换

在模块状态之前，需要先使能 PI 镜像交换

```
RETURN_IF_FAILED(EcatPIEnable(ecat_dev));
```

4.4.5 预填充 PDO 队列

为了减少用户逻辑和系统抖动对 EtherCAT 总线造成的影响，可以向 PDO 队列里预填充几帧数据作为缓冲。

```
    for (uint32_t i = 0; i < 2; i++)
    {
        EcatPIOutputQueuePush(ecat_dev, false, 100);
    }
```

4.4.6 EtherCAT 周期处理主循环

主循环中，EcatPIInputQueuePop 等待队列接收可用数据。用户逻辑更新数据后，通过 EcatPIOutputQueuePush 函数将数据填入队列中。

```
    for (uint32_t i = 0; i < 2; i++)
    {
        EcatPIOutputQueuePush(ecat_dev, false, 100);
```

```
}
_INF_("EcatPINMap done");
RETURN_IF_FAILED(EcatPIEnable(ecat_dev));
while (1)
{
    while (EcatPIInputQueuePop(ecat_dev, false, 100) != 0);
      // 用户逻辑
    if (EcatPIOutputQueuePush(ecat_dev, 0, 100) != 0)
    {
        _ERR_("push pi output error");
        break;
    }
}
```

4.4.7 退出

当应用程序需要退出时，需要销毁 ECAT 主站。

```
// 退出
RETURN_IF_FAILED(EcatBusStop(ecat_dev));
RETURN_IF_FAILED(EcatClose(ecat_dev));
```

4.5 PDO 数据处理

PDO 数据的读写可以在主循环里进行。由于我们配置为浮点数模式，因此回调函数里不需要做复杂的编码转换，直接按浮点数形式读写数据即可。

```
while (1)
{
    while (EcatPIInputQueuePop(ecat_dev, false, 100) != 0);
    loops++;

    for(int i = 0; i < 4; i++) {
        po->zioc_e0004ai.val[i] = 20.;
    }
    for(int i = 0; i < 4; i++) {
        po->zioc_e0004ai1.val[i] = 20.;
    }
    for(int i = 0; i < 8; i++) {
        po->zioc_e0008au.val[i] = 0.8 * (i + 1);
    }

    if(loops % 1000 == 0) {
        // 打印输入模拟量
        printf("Analog inputs:\r\n");
        for(int i = 0; i < 8; i++) {
```

```
                printf("ZIOC-E0800AI ch[%d] = %.6f\r\n", i, pi->zioc_e0800ai.val[i]);
            }
            for(int i = 0; i < 8; i++) {
                printf("ZIOC-E0800AI1 ch[%d] = %.6f\r\n", i, pi->zioc_e0800ai1.val[i]);
            }
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0800AU1 ch[%d] = %.6f\r\n", i, pi->zioc_e0800au1.val[i]);
            }
            // 打印输出变量
            printf("Analog outputs:\r\n");
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0004AI ch[%d] = %.6f\r\n", i, po->zioc_e0004ai.val[i]);
            }
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0004AI1 ch[%d] = %.6f\r\n", i, po->zioc_e0004ai1.val[i]);
            }
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0008AU ch[%d] = %.6f\r\n", i, po->zioc_e0008au.val[i]);
            }
        }
        if (EcatPIOutputQueuePush(ecat_dev, 0, 100) != 0)
        {
            _ERR_("push pi output error");
            break;
        }
    }
```

4.6 完整示例代码

```
#include <stdio.h>

#include "pci_errno.h"
#include "pci_zecm.h"
#include "pci_dbg.h"

#ifdef VXWORKS
#define main SYNC_CONTROL_TEST
#endif

#define RETURN_IF_FAILED(expr)          { int ret = (expr); if (ret != 0) { _ERR_(#expr " failed: err=%d", ret);
return -1; } else { _INF_(#expr " succeeded"); } }

////////////////////////////////////////////////////////////////////////////
// 过程映像
```

插板式模组应用

插板式 EtherCAT 系列从站

```
#pragma pack(push, 1)

typedef struct {
    uint8_t mapping[2];
    uint8_t pad[30];
} ZIOC_E0800_o;

typedef struct {
    float val[8];
} ZIOC_E0800_i;

typedef struct {
    uint8_t mapping[4];
    uint8_t pad[28];
} ZIOC_E0008_i;

typedef struct {
    float val[8];
} ZIOC_E0008_o;

typedef struct {
    uint8_t mapping[4];
    uint8_t pad[12];
} ZIOC_E0004_i;

typedef struct {
    float val[4];
} ZIOC_E0004_o;

typedef struct {
    ZIOC_E0800_o zioc_e0800ai;
    ZIOC_E0004_o zioc_e0004ai;
    ZIOC_E0800_o zioc_e0800ai1;
    ZIOC_E0004_o zioc_e0004ai1;
    ZIOC_E0800_o zioc_e0800au1;
    ZIOC_E0008_o zioc_e0008au;
} proc_img_o;

typedef struct {
    ZIOC_E0800_i zioc_e0800ai;
    ZIOC_E0004_i zioc_e0004ai;
    ZIOC_E0800_i zioc_e0800ai1;
    ZIOC_E0004_i zioc_e0004ai1;
    ZIOC_E0800_i zioc_e0800au1;
```

```
    ZIOC_E0008_i zioc_e0008au;
} proc_img_i;

#pragma pack(pop)

#ifdef _WIN32
#include <windows.h>
void usleep(int64_t usec){
    HANDLE timer;
    LARGE_INTEGER ft;
    ft.QuadPart = -(10*usec);
    timer = CreateWaitableTimer(NULL, TRUE, NULL);
    SetWaitableTimer(timer, &ft, 0, NULL, NULL, 0);
    WaitForSingleObject(timer, INFINITE);
    CloseHandle(timer);
}
#else
#include <unistd.h>
#endif

int main(int argc, char* argv[])
{
    char index_buff[10];
    uint32_t index = 0;
    uint64_t loops = 0;
    uint32_t channel = 0;
    char *eni_file;
    ECAT_HANDLE ecat_dev;

    if (argc != 4)
    {
       _INF_("usage: ./program encoder_id chnnal eni.xml\r\n");
       return 0;
    }

    index = atoi(argv[1]);
    channel = atoi(argv[2]);
    if(channel > 1) channel = 1;
    eni_file = argv[3];

    //PCIe-2E 只有一个通道,PCIe-4E 有两个通道
    RETURN_IF_FAILED(EcatOpen(&ecat_dev, BOARD_ALIAS(index_buff, index), channel));

    RETURN_IF_FAILED(EcatBusRun(ecat_dev, eni_file));
```

```
// 请求切换状态
uint8_t state;
RETURN_IF_FAILED(EcatRequestMasterState(ecat_dev, EcatStateO));
for (uint32_t i = 0; ; i++)
{
    RETURN_IF_FAILED(EcatGetMasterState(ecat_dev, &state));
    _INF_("request is: 8, curr is: %d", state);
    usleep(500*1000);
    if (state == EcatStateO) break;
}

_INF_("EcatRequestMasterState done");

// 清空并预计填入数据
proc_img_i *pi;
proc_img_o *po;
RETURN_IF_FAILED(EcatPINMap(ecat_dev, PI_AREA_LOCAL_INPUT, (void **)&pi));
_INF_("EcatPINMap done");

RETURN_IF_FAILED(EcatPINMap(ecat_dev, PI_AREA_LOCAL_OUTPUT, (void **)&po));
_INF_("EcatPINMap done");

_INF_("pi: 0x%llx", pi);
_INF_("po: 0x%llx", po);

for (uint32_t i = 0; i < 2; i++)
{
    EcatPIOutputQueuePush(ecat_dev, false, 100);
}
_INF_("EcatPINMap done");

RETURN_IF_FAILED(EcatPIEnable(ecat_dev));

while (1)
{
    while (EcatPIInputQueuePop(ecat_dev, false, 100) != 0);
    loops++;

    for(int i = 0; i < 4; i++) {
        po->zioc_e0004ai.val[i] = 20.;
    }
    for(int i = 0; i < 4; i++) {
        po->zioc_e0004ai1.val[i] = 20.;
```

```
        }
        for(int i = 0; i < 8; i++) {
            po->zioc_e0008au.val[i] = 0.8 * (i + 1);
        }

        if(loops % 1000 == 0) {
            // 打印输入模拟量
            printf("Analog inputs:\r\n");
            for(int i = 0; i < 8; i++) {
                printf("ZIOC-E0800AI ch[%d] = %.6f\r\n", i, pi->zioc_e0800ai.val[i]);
            }
            for(int i = 0; i < 8; i++) {
                printf("ZIOC-E0800AI1 ch[%d] = %.6f\r\n", i, pi->zioc_e0800ai1.val[i]);
            }
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0800AU1 ch[%d] = %.6f\r\n", i, pi->zioc_e0800au1.val[i]);
            }
            // 打印输出变量
            printf("Analog outputs:\r\n");
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0004AI ch[%d] = %.6f\r\n", i, po->zioc_e0004ai.val[i]);
            }
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0004AI1 ch[%d] = %.6f\r\n", i, po->zioc_e0004ai1.val[i]);
            }
            for(int i = 0; i < 4; i++) {
                printf("ZIOC-E0008AU ch[%d] = %.6f\r\n", i, po->zioc_e0008au.val[i]);
            }
        }

        if (EcatPIOutputQueuePush(ecat_dev, 0, 100) != 0)
        {
            _ERR_("push pi output error");
            break;
        }

    }
    RETURN_IF_FAILED(EcatBusStop(ecat_dev));
    RETURN_IF_FAILED(EcatClose(ecat_dev));
    return 0;
}
```

4.7 程序运行结果

启动程序后，主站切换到 Operation 操作状态，开始打印数据。如图 4.11 所示，最终结果符合预期。

其中，前两组模拟电流输入和输出模组因为是 1 个输出电流端口连接到 2 个输入电流端口，因此会分流，每两个输入端口读数和等于输出配置。

ZIOC-E0800AU1 与 ZIOC-E0008AU 为电压输入输出模组，端口输出与读数一对一，所以输出配置和输入度数一致。

```
Analog inputs:
ZIOC-E0800AI ch[0] = 9.608473
ZIOC-E0800AI ch[1] = 10.424618
ZIOC-E0800AI ch[2] = 9.870866
ZIOC-E0800AI ch[3] = 10.161506
ZIOC-E0800AI ch[4] = 9.554768
ZIOC-E0800AI ch[5] = 10.481608
ZIOC-E0800AI ch[6] = 9.790624
ZIOC-E0800AI ch[7] = 10.241784
ZIOC-E0800AI1 ch[0] = 10.017651
ZIOC-E0800AI1 ch[1] = 10.034880
ZIOC-E0800AI1 ch[2] = 10.021569
ZIOC-E0800AI1 ch[3] = 10.033358
ZIOC-E0800AI1 ch[4] = 10.022114
ZIOC-E0800AI1 ch[5] = 10.018919
ZIOC-E0800AI1 ch[6] = 10.016552
ZIOC-E0800AI1 ch[7] = 10.021931
ZIOC-E0800AU1 ch[0] = 0.800367
ZIOC-E0800AU1 ch[1] = 1.600509
ZIOC-E0800AU1 ch[2] = 2.398982
ZIOC-E0800AU1 ch[3] = 3.201007
Analog outputs:
ZIOC-E0004AI ch[0] = 20.000000
ZIOC-E0004AI ch[1] = 20.000000
ZIOC-E0004AI ch[2] = 20.000000
ZIOC-E0004AI ch[3] = 20.000000
ZIOC-E0004AI1 ch[0] = 20.000000
ZIOC-E0004AI1 ch[1] = 20.000000
ZIOC-E0004AI1 ch[2] = 20.000000
ZIOC-E0004AI1 ch[3] = 20.000000
ZIOC-E0008AU ch[0] = 0.800000
ZIOC-E0008AU ch[1] = 1.600000
ZIOC-E0008AU ch[2] = 2.400000
ZIOC-E0008AU ch[3] = 3.200000
```

图 4.11 数据打印结果

5. 免责声明

本着为用户提供更好服务的原则，广州致远电子股份有限公司（下称“致远电子”）在本手册中将尽可能地为用户呈现详实、准确的产品信息。但介于本手册的内容具有一定的时效性，致远电子不能完全保证该文档在任何时段的时效性与适用性。致远电子有权在没有通知的情况下对本手册上的内容进行更新，恕不另行通知。为了得到最新版本的信息，请尊敬的用户定时访问致远电子官方网站或者与致远电子工作人员联系。感谢您的包容与支持！

诚信共赢，持续学习，客户为先，专业专注，只做第一

广州致远电子股份有限公司

更多详情请访问 **www.zlg.cn**

欢迎拨打全国服务热线 **400-888-4005**