Application Note

插板式模组应用

插板式 EtherCAT 系列从站

AN01010101 1.00 Date:2025/10/22

类别	内容
关键词	插板式EtherCAT模组、致远PCIe-2E主站卡
摘要	本文介绍了使用致远PCIe-2E主站卡对插板式电机驱动模块的应用示例

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	2025/10/21	•创建文档

目　录

插板式模组应用

插板式 EtherCAT 系列从站

1. 适用范围

本文档适用于使用致远 PCIe-2E 主站卡搭配插板式 EtherCAT 电机驱动模组的应用。

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

2.1 CiA402 简介

CiA402 是一种主流的电机控制通讯框架协议，使用主从模式，主站控制多台从站（电机）的位置、速度或转矩等。 CiA402 最初只在 CANopen 运行，目前已扩展到 Modbus RTU、Modbus TCP 和 EtherCAT 等实时总线。但不管任何总线，只要按 CiA402 流程操作寄存器，就能控制电机。

驱动器上电后，会尝试通过网口连接 EtherCAT 主站，或通过调试口连接 Modbus RTU 主站。如果已连接上 EtherCAT 主站，则先自动进入 EtherCAT 的 Operational 状态，之后按 CiA402 运行电机协议。如果未连接上 EtherCAT 主站，但已连接上 Modbus RTU 主站，则直接运行 CiA402。

2.2 CiA402 状态机

主站连通驱动器后，驱动器按图 2.1 运行 CiA402 工作流程。CiA402 定义了不同地址的对象，例如最重要的控制字和状态字（在 EtherCAT 中的地址分别是 6040h 和 6041h）。主站通过 SDO 或 PDO 写控制字，就能触发不同的跳转条件（例如 T2~T16），让驱动器在不同的状态中切换，同时主站通过读状态字，就能掌握驱动器的基本状态，例如是否有故障。在此基础上，还有了不同功能的对象，例如控制位置的目标位置对象（607Ah）。

2.2.1 状态

在 Start 到 Ready To Switch On 状态，只需接通控制电源、保持通讯即可，无需接通驱动电机的主电源，但电机是未通电使能的。Switched On 状态表示控制电源和主电源都已接通，但电机未能通电使能。Openeration Enabled 状态表示控制电源和主电源都已接通，电机已通电上锁。

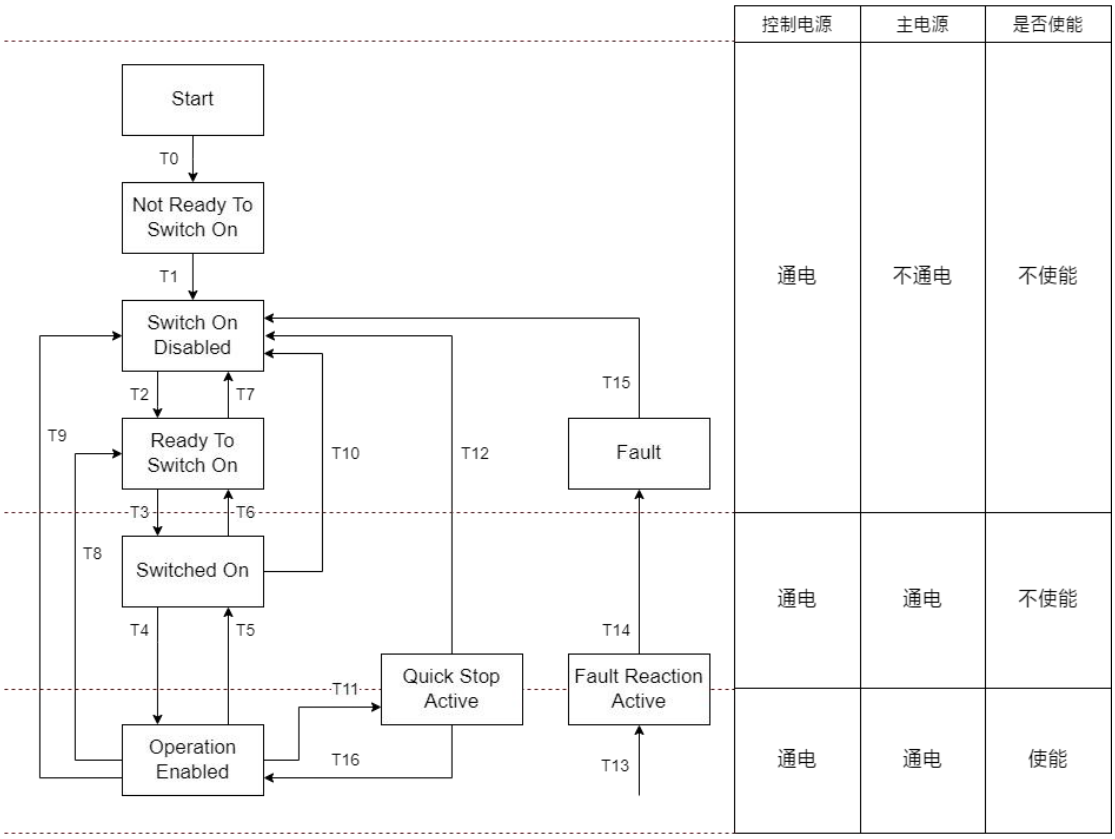


图 2.1　CiA402 工作流程

表 2.1　CiA402 状态

状态	驱动器动作
Not ready to switch on	控制电源已供电，开始初始化参数
Switch on disabled	初始化参数完毕
Ready to switch on	等待主电源供电
Switched on	控制电源和主电源都已供电，驱动器准备好，等待使能
Operation enabled	已被使能，电机可以运行
Quick stop active	电机快速减速停止
Fault reaction active	错误发生但未被处理，电机已停机
Fault	错误已被处理，等待切换到 Switch on disabled 状态

2.2.2 条件

图 2.1 各个跳转条件的说明如表 2.2 所示。控制字用于发送总线指令，状态字的 1~9 位表示跳转后的状态。另外本驱动器不含 STO 使能控制脚，表中 STO 固定为有效。

表 2.2　CiA402 跳转条件

条件	说明	控制字 6040h	状态字 6041h（Bit1~9）
T0	控制电源已供电	自动跳转	0x0000
T1	参数已成功初始化	自动跳转	0x0250
T2	总线发 Shutdown 指令，且 STO 有效	0x0006	0x0231
T3	总线发 Switch On 或 Enable Operation 指令	0x0007	0x0233
T4	自动跳转（T3 已发送 Enable Operation）或 总线发 Enable Operation 指令	0x000F	0x0237/0x1237
T5	总线发 Disable Operation 指令	0x0007	0x0233
T6	总线发 Shutdown 指令	0x0006	0x0231
T7	总线发送 Disable Voltage 或 Quick Stop 指令，或 STO 无效	0x0000 或 0x0002 或自动跳转	0x0250
T8	总线发送 Shutdown 命令	0x0006	0x0231
T9	总线发 Disable Voltage 指令或 STO 无效	0x0000 或自动跳转	0x0250
T10	总线发送 Disable Voltage 或 Quick Stop 指令或 STO 无效	0x0000 或 0x0002	0x0250
T11	总线发送 Quick Stop 指令	0x0002	0x217
T12	总线发送 Disable Voltage 指令或 STO 无效	0x0000 或自动跳转	0x0250
T13	发生 2 等级故障	自动跳转	0x021F
T14	发生 3 或 4 等级的故障级 或 Fault reaction active 完成退出	自动跳转	0x0218
T15	指令 Fault Reset 指令或 STO 无效	0x0080 或自动跳转	0x0250
T16	总线发送 Enable Operation 指令	0x000F	0x0237/0x1237

2.3　CiA402 控制字

2.3.1 位定义

控制字 Controlword 的地址位于 6040h，主站通过它给从站发出不同的指令组合，各个位的定义如表 2.3 所示。其中 Bit 8 Halt 为 1 时驱动器减速暂停，如果电机正在图 2.1 的 Operation enabled 状态运行，此位为 1 时电机将按当前模式下的减速度进行减速停止，并停

留在 Operation enabled 状态。Bit 7 Fault reset 用于错误复位，如果出现了错误，此位从 0 变为 1 将清除当前的错误，并跳转到图 2.1 的 Switch on disabled 状态，等待再次运行。Bit 15~9 是预留位。Bit 3~0 是组合指令。

表 2.3　控制字位定义

15~9	8	7	6~4	3	2	1	0
N / A	Halt	Fault reset	Operation mode specific	Enable operation	Quick Stop	Enable voltage	Switch on

Bit6~4 是运行模式控制位，仅在 PP 轮廓位置和 HM 归零模式使用，如表 2.4 所示。

表 2.4　运行模式控制位定义

位	PP 轮廓位置	HM 归零
4	上升沿时更新目标位置	上升沿时开始归零
5	0：到达当前的目标位置后，再往新的目标位置运行 1：立刻往新的目标位置运行	---
6	0：目标位置为绝对位置 1：目标位置为相对位置	---

2.3.2 指令编码

控制字的 Bit 3~0 以及 Bit 7 组成命令代码，实现图 2.1 的跳转条件，如表 2.5 所示。

表 2.5　指令编码

指令名	位组合					跳转条件
	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0	
Shutdown	0	X	1	1	0	T2,T6,T8
Switch on	0	0	1	1	1	T3
Switch on + enable operation	0	1	1	1	1	T3 + T4
Disable voltage	0	X	X	0	X	T7,T9,T10,T12
Quick stop	0	X	0	1	X	T7,T10,T11
Disable operation	0	0	1	1	1	T5
Enable operation	0	1	1	1	1	T4,T16
Fault reset	0->1	X	X	X	X	T15

2.4　CiA402 状态字

2.4.1 位定义

状态字 Statusword 的地址位于 6041h，主站通过读取此对象获悉从站的当前状态，各个位的定义如表 2.6 所示。

表 2.6　状态字位定义

位号	名称	说明
0	Ready to switch on	准备启动
1	Switch on	启动，主电源需接通
2	Operation enabled	允许启动，电机已通电上锁
3	Fault	错误
4	Voltage enabled	为 1 表示已接通主电源
5	Quick stop	为 0 表示正在快速停止
6	Switch on disabled	未启动
7	Warning	为 1 表示发生了错误等级为 0 的错误，即发出警告
8	N / A	本产品不使用
9	Remote	为 1 表示控制字的指令已被执行
10	Target reached	为 1 表示已达到目标值，位置、速度、转矩等
11	Internal limit active	本产品不使用
12~15	N / A	本产品不使用

2.4.2 状态编码

状态字的 Bit 0~3 以及 Bit5~6 组成状态编码，每组编码对应图 2.1 的一种状态，如表 2.7 所示。主站查询这些编码器，就能知道从站的当前状态。

表 2.7　状态编码

状态位组合	CiA402 状态
xxxx xxxx x0xx 0000b	Not ready to switch on
xxxx xxxx x1xx 0000b	Switch on disabled
xxxx xxxx x01x 0001b	Ready to switch on
xxxx xxxx x01x 0011b	Switched on
xxxx xxxx x01x 0111b	Operation enabled
xxxx xxxx x00x 0111b	Quick stop active
xxxx xxxx x0xx 1111b	Fault reaction active
xxxx xxxx x0xx 1000b	Fault

2.5　CiA402 运行模式

在进入 Operation enabled 状态后，驱动器将根据运行模式字 Modes of operation（地址 6060h）按不同的模式运行，如表 2.8 所示。本产品支持+8 周期同步位置模式（简称 CSP），+3 轮廓速度模式（PV），+1 轮廓位置模式（PP），+6 归零模式（HM）。

表 2.8　运行模式字

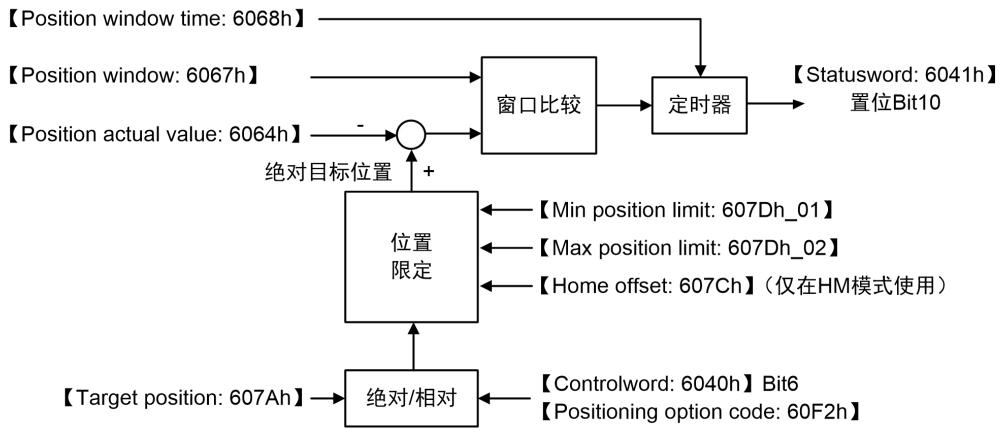
数值	定义
-128～-1	厂家自定义
0	不执行任务
+1	Profile position mode　PP 轮廓位置模式
+2	Velocity mode　V 速度模式
+3	Profile velocity mode　PV 轮廓速度模式
+4	Profile torque　mode　PT 轮廓转矩模式
+5	保留
+6	Homing mode　HM 归零模式
+7	Interpolated position mode　IP 插补位置模式
+8	Cyclic sync position mode　CSP 周期同步位置模式
+9	Cyclic sync velocity mode　CSV 周期同步速度模式
+10	Cyclic sync torque mode　CST 周期同步转矩模式
+11～+127	保留

2.5.1 到达和跟随判断

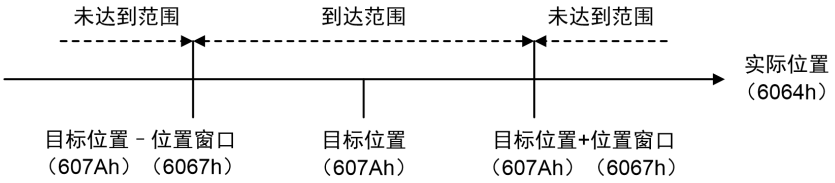
1. 位置到达和跟随

图 2.2 显示了在位置控制相关的模式下，位置到达的判断原理。【Target position: 607Ah】是目标位置，可以是绝对位置，如果控制字【Controlword: 6040h】的 Bit6 为 0 表示绝对位置，如果为 1 且【Positioning option code: 60F2h】为 0，相对上一个目标位置运行，如果【Positioning option code: 60F2h】为 2，相对当前实际位置运行。如果实际位置【Position actual value: 6064h】在设定的位置窗口【Position window: 6067h】范围内，且持续时间长于位置窗口时间【Position window time: 6068h】，就被判断为已到达目标位置，状态字的 Bit 10 被置 1。【Min position limit: 607Dh_01】、【Max position limit: 607Dh_02】用于限定最终的绝对目标位置的范围，【Home offset: 607Ch】是 HM 归零模式使用的偏移量。

在 CSP 周期同步位置模式下不支持位置到达的判断，仅支持绝对位置控制。



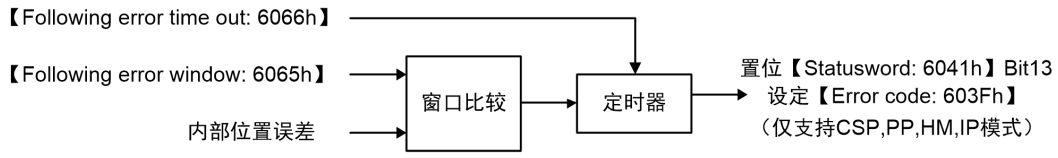
(a) 位置到达框图



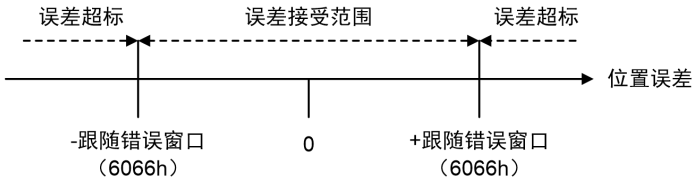
(b) 位置到达范围

图 2.2　位置到达判断

图 2.3 显示了 CSP 周期同步位置、PP 轮廓位置和 HM 归零等模式下位置跟随错误的判断原理。当驱动器内部位置误差大于跟随错误窗口【Following error window: 6065h】，且持续时间长于跟随错误超时实际【Following error time out: 6066h】，则就被判断为跟随错误，并且设定错误码到【Error code: 603Fh】。



(a) 位置跟随错误框图



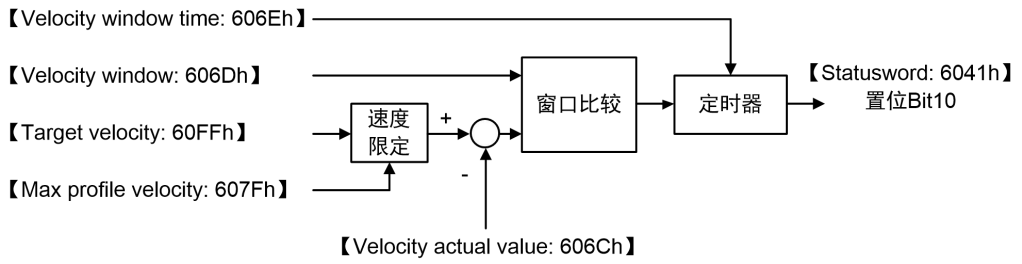
(b) 位置跟随错误范围

图 2.3　位置跟随错误判断

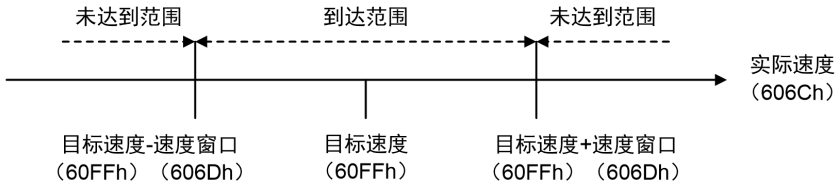
2.　速度到达和零速

图 2.4 显示了在 PV 轮廓速度模式下判断目标速度是否到达的原理。目标速度【Target velocity：60FFh】被限定在【Max profile velocity: 607Fh】的范围内。当目标速度和实际速

度【Velocity actual value: 606Ch】的差值，在速度窗口【Velocity window: 606Dh】的范围内，且持续时间长于【Velocity window time: 606Eh】，就被判断为速度到达，状态字的 Bit10 被置位。



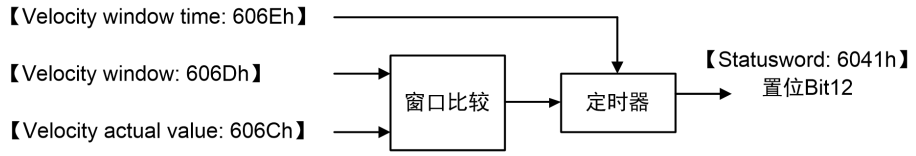
(a) 速度到达框图



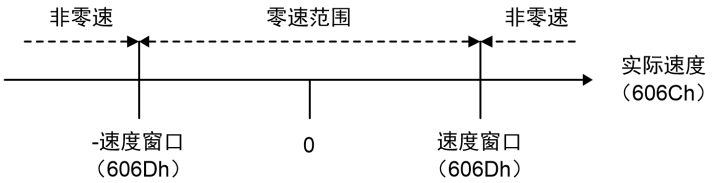
(b) 速度到达范围

图 2.4　速度到达判断

图 2.5 显示了在 PV 轮廓速度模式下零速度的判断原理。当实际速度【Velocity actual value: 606Ch】在速度窗口【Velocity window: 606Dh】的范围内，且持续时间长于【Velocity window time: 606Eh】，就被判断为零速度，状态字的 Bit12 被置位。



(a) 零速判断框图



(b) 零速判断范围

图 2.5　零速判断

2.5.2 CSP 周期同步位置模式

在 CSP 周期同步位置模式下，主站通过 PDO 定时更新目标位置。主站根据需要实现的加减速和匀速过程，精确计算好每个周期的目标位置，从站不负责加减速的处理。本驱动器 CSP 模式仅支持绝对运动，不支持相对运动。

1. SDO 初始化

先按下面的流程，通过 SDO 初始化必要的对象。其中带*的对象可使用默认值，不进行初始化。

1) 【Mode of operations: 6060h】设定为周期同步位置模式(0x08)。

2) 【Following error window: 6065h】*设定最大的位置误差，单位 PUU。

3) 【Following error time out: 6066h】*设定最大误差的允许时间，单位 ms。

4) 【Position window: 6067h】*设定目标位置到达范围，单位步脉冲。

5) 【Position window time: 6068h】*设定目标位置到达范围的允许时间，单位 ms。

6) 【Min position limit: 607Dh_01】*设定目标位置最小限定值，默认-0x7FFFFFFF。

7) 【Max position limit: 607Dh_02】*设定目标位置最大限定值，默认 0x7FFFFFFF。

其中【Following error window: 6065h】设定最大允许的误差，即当前给定位置和实际位置的差值，如果误差一直大于此值，并且持续时间长于【Following error time out: 6066h】，就报错停机，先后进入 Fault reaction active 和 Fault 状态，【Statusword：6041h】状态寄存器的 Fault 位置 1，【Error code: 603Fh】显示对应的错误码。往【Controlword: 6040h】写入 0x0080，触发 T15，返回 Switch on disabled 状态，可按上面流程重新开始执行。

2. PDO 控制

初始化完成后，再使用如下 PDO 控制电机，如表 2.9 所示。【Target position: 607Ah】，设定目标位置【Controlword: 6040h】依次写入 0x06->0x07->0x0F，进入 Operation enabled 状态。电机将运行到目标位置。

注：以整步角度 1.8°，256 细分为例：如果【Target position: 607Ah】设置为 51200（（360/1.8（整步角度）） * 256（细分）），即电机转动一圈。细分数可以在 2000h 对象组中修改。

表 2.9 CSP 模式 PDO

方向	对象	名称	数据类型	单位	备注
RxPDO 驱动接收	【Controlword: 6040h】	控制字	UINT16	---	**必选**
	【Target position: 607Ah】	目标位置	INT32	微步	**必选**
TxPDO 驱动发送	【Statusword: 6041h】	状态字	UINT16	---	**必选**
	【Error code: 603Fh】	错误码	UINT16	---	可选
	【Position actual value: 6064h】	实际位置	INT32	微步	**必选**
	【Digital inputs: 60FDh】	数字信号输入	UINT32	---	可选

2.5.3 PP 轮廓位置模式

在 PP 轮廓位置位置模式下，主站通过 PDO 更新目标位置，驱动器自行实现加减速和匀速过程，最终到达目标位置。PP 模式支持绝对运动和相对运动。

1. SDO 初始化

先按下面的流程，通过 SDO 初始化必要的对象。其中带*的对象可使用默认值，不进行初始化。

1) 【Mode of operations: 6060h】设定为轮廓位置模式模式(0x01)。

2) 【Following error window: 6065h】*设定最大的位置误差，单位 PUU。

3) 【Following error time out: 6066h】*设定最大误差的允许时间，单位 ms。

4) 【Position window: 6067h】*设定目标位置到达范围，单位步脉冲。

5) 【Position window time: 6068h】*设定目标位置到达范围的允许时间，单位 ms。

6) 【Profile jerk use: 60A4 】*设定 Jerk 参数，仅使能 S 曲线时有效，单位 RPM/s/s。

7) 【Max profile velocity: 607Fh】设定最高限定速度，单位 0.1RPM。

8) 【Positioning option code: 60F2】*设定相对位置的方式，【Controlword: 6040h】的 Bit6 为 1 时表示相对运动，本对象有效，否则无效；本对象默认为 0 表示相对前一个目标位置运行，为 1 表示相对当前位置运行。(暂不支持相对运动)

9) 【Min position limit: 607Dh_01】*设定目标位置最小限定值，默认-0x7FFFFFFF。

10) 【Max position limit: 607Dh_02】*设定目标位置最大限定值，默认 0x7FFFFFFF。

2. PDO 控制

初始化完成后，再使用如下 PDO 控制电机，如表 2.10 所示。【Controlword: 6040h】的 Bit6~4 需按表 2.4 设定，Bti6 和 Bit5 设定绝对运动和相对运动，【Target position: 607Ah】设定目标位置。【Profile velocity: 6081h】设定轮廓速度，【Profile acceleration：6083h】设定加速度，【Profile deceleration：6084h】设定减速度。最后【Controlword: 6040h】依次写入 0x06->0x07->0x0F，进入 Operation enabled 状态。

随后电机将以设定的加速度加速到目标速度，最终以设定的减速度停在目标位置。当【Controlword: 6040h】的 Bit4 出现 0->1，则按新的配置运行到新的目标位置；往整个字写入 0x10F 电机减速停止。

表 2.10 PP 模式 PDO

方向	对象	名称	数据类型	单位	备注
RxPDO 驱动接收	【Controlword: 6040h】	控制字	UINT16	---	必选
	【Target position: 607Ah】	目标位置	INT32	微步	必选
	【Profile velocity: 6081h】	轮廓速度	UINT32	0.1RPM	必选
	【Profile acceleration：6083h】	加速度	UINT32	RPM/s	必选
	【Profile deceleration：6084h】	减速度	UINT32	RPM/s	必选
TxPDO 驱动发送	【Statusword: 6041h】	状态字	UINT16	---	必选
	【Error code: 603Fh】	错误码	UINT16	---	必选
	【Position actual value: 6064h】	实际位置	INT32	微步	必选
	【Velocity actual value: 606C】	实际速度	INT32	0.1RPM	必选

2.5.4 PV 轮廓速度模式

在 PV 轮廓速度模式下，让电机匀速转动。

1. SDO 初始化

先按下面的流程，通过 SDO 初始化必要的对象。其中带*的对象可使用默认值，不进行初始化。

1) 【Mode of operations: 6060h】设定为轮廓速度模式(0x03)。
2) 【Velocity window: 606Dh】*设定目标速度到达范围，单位 0.1RPM。
3) 【Velocity window time: 606Eh】*设定目标速度到达范围的允许时间，单位 ms。
4) 【Profile jerk use: 60A4 】*设定 Jerk 参数，仅使能 S 曲线时有效，单位 RPM/s/s。
5) 【Max profile velocity: 607Fh】*设定最高限定速度，单位 0.1RPM。

2. PDO 控制

初始化完成后，再使用如下 PDO 控制电机，如表 2.11 所示。【Target velocity：60FFh】设定不同的目标速度，【Profile acceleration：6083h】设定加速度，【Profile deceleration：6084h】设定减速度。最后【Controlword: 6040h】控制字，依次写入 0x06->0x07->0x0F，进入 Operation enabled 状态。电机将以设定的加速度加速到目标速度，【Target velocity：60FFh】目标速度，设定为 0 时，电机以该减速度减速停止。

表 2.11 PV 模式 PDO

方向	对象	名称	数据类型	单位	备注
RxPDO 驱动接收	【Controlword: 6040h】	控制字	UINT16	---	必选
	【Target velocity：60FFh】	目标速度	INT32	0.1RPM	必选
	【Profile acceleration：6083h】	加速度	UINT32	RPM/s	可选
	【Profile deceleration：6084h】	减速度	UINT32	RPM/s	可选
TxPDO 驱动发送	【Statusword: 6041h】	状态字	UINT16	---	必选
	【Error code: 603Fh】	错误码	UINT16	---	可选
	【Position actual value: 6064h】	实际位置	INT32	微步	必选
	【Velocity actual value：606Ch】	实际速度	INT32	0.1RPM	可选

2.5.5 HM 归零模式

在 HM 归零模式下，让电机找到零点。

1. SDO 初始化

先按下面的流程，通过 SDO 初始化必要的对象。其中带*的对象可使用默认值，不进行初始化。

1) 【Mode of operations: 6060h】设定为归零模式(0x06)。
2) 【Following error window: 6065h】*设定最大的位置误差，单位 PUU。
3) 【Following error time out: 6066h】*设定最大误差的允许时间，单位 ms。
4) 【Position window: 6067h】*设定目标位置到达范围，单位步脉冲。
5) 【Position window time: 6068h】*设定目标位置到达范围的允许时间，单位 ms。
6) 【Controlword: 6040h】依次写入 0x06->0x07->0x0F，进入 Operation enabled 状态。

2. PDO 控制

初始化完成后，再使用如下 PDO 控制电机，如表 2.12 所示。设定【Homing speed switch: 6099h_01】寻找归零开关的最高速度，【Homing speed zero: 6099h_02】离开归零开关的最

高速度，【Homing acceleration: 609Ah】归零的加减速。【Controlword: 6040h】的 Bit4 需按表 2.4 设定，出现 0->1 变化开始运动，1->0 暂停运动。【Homing method：6098h】设定归零模式，17 使用正限位开关归零，18 使用负限位开关归零，其他设定不支持。

表 2.12　HM 模式 PDO

方向	对象	名称	数据类型	单位	备注
RxPDO 驱动接收	【Controlword: 6040h】	控制字	UINT16	---	必选
	【Homing method: 6098h】	归零模式	INT8	---	可选
	【Homing speed switch:　6099h_01】	寻找归零开关的最高速度	UINT32	0.1RPM	可选
	【Homing speed zero: 6099h_02】	离开归零开关的最高速度	UINT32	0.1RPM	可选
	【Homing acceleration: 609Ah】	归零的加减速	UINT32	RPM/s	可选
	【Homeing offset: 607Ch】	零点偏移量	INT32	微步	可选
TxPDO 驱动发送	【Statusword: 6041h】	状态字	UINT16	---	必选
	【Error code: 603Fh】	错误码	UINT16	---	可选
	【Position actual value: 6064h】	实际位置	INT32	微步	必选
	【Velocity actual value: 606C】	实际速度	INT32	0.1RPM	可选

2.6　CiA402 其他说明

2.6.1 CiA402 数字信号

【Digital inputs: 60FDh】反馈输入数字信号的状态，为 0 表示无效，为 1 表示有效。Bit 0 为 DI 1，Bit 1 为 DI 2，Bit 2 为 Latch，Bit 3~31 保留。当配置使能限位功能后，DI 2 和 DI 1 能分别触发正负限位的故障。

3. 开发环境

（1）个 ZCPC-80801 耦合器模组、1 个 ZMTC-EF1200 步进电机模组、一个小型步进电机、一台 24V 电源。还需要 1 个 ZMB-EF1200-4 分线底板用于连接模组。

（2）安装了致远 PCIe-2E 主站卡的 Windows 或 Linux 工控机一台。

（3）一台安装了 AWStudio 运动控制版上位机软件的 Windows 电脑，软件如下图 3.1 所示。

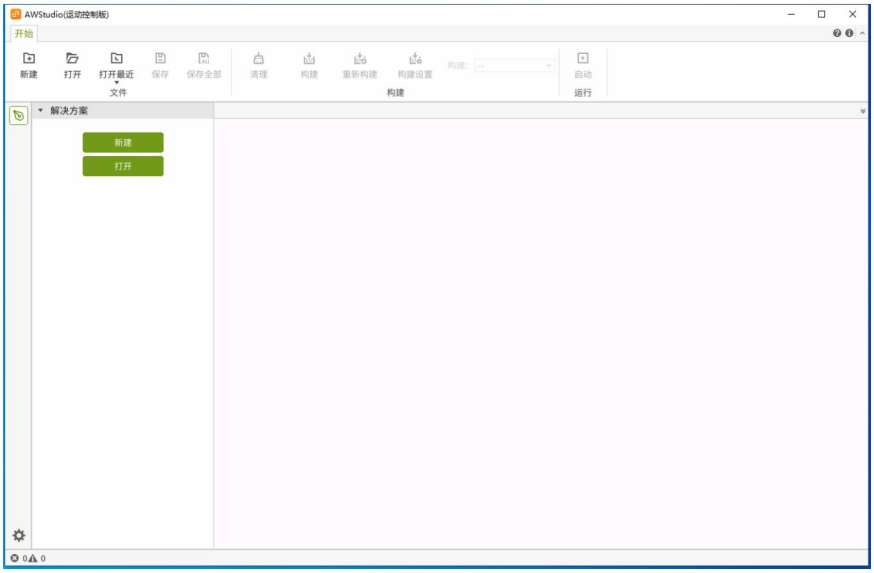


图 3.1 上位机软件

4. 实现过程

将 1 个 ZCPC-80801 耦合器、1 个 ZMTC-EF1200 步进电机驱动模组，从左到右分别插入 ZMB-EF1200-4 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。

用网线连接 ZCPC-80801 耦合器的“IN”口和 PCIe 主站卡的其中一个 ECAT 口，另外底板的 MOTOR1 端子上也需要接 1 路 24V 给 ZMTC-EF1200 步进电机驱动模组，用于驱动电机供电，如图 4.1 所示。供电接线请参考《ZMB-EF1200-4 分线底板数据手册》。

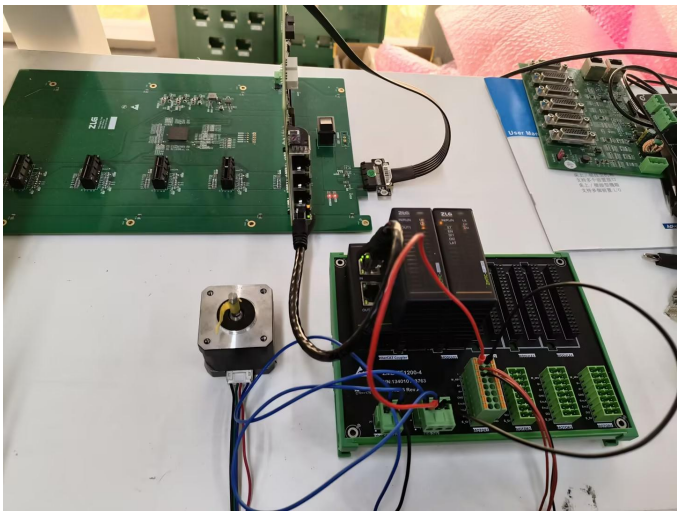


图 4.1 硬件环境

4.1 使用 AWStudio 上位机控制

（1）运行 ecat_server。在安装了 PCIe-2E 主站卡的工控机上，运行“光盘资料/Linux(或 Windows)/tools/ecat_server/ecat_server”工具，启动后会开启当前部署了 PCIe 主站卡的电脑下所有可用的 PCIe 主站卡的主站连接。

双击光盘资料 tools/ecat_server/ecat_server.exe 程序打开服务器。如下图 4.2 所示，上位机可以连接的通道端口号为 5010。

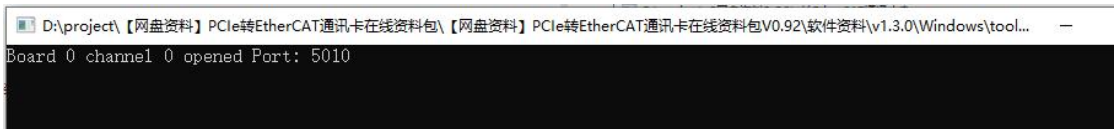


图 4.2 打开服务器

端口生成规则：　5　0　X　X
|　|-------通道号
|----------编码器 ID

运行成功如上图 4.2 运行结果所示，使用 PCIe-2E 扫描获得的端口号为 5010，端口号的数据构成规则约定为倒数第二位是编码器 ID，例如本通讯卡的编码 ID 为 1，倒数第一位为通道号，PCIe-2E 只有一个通道，则为 50X0，若是 PCIe-4E 系列的双通道，端口号具有

两个，分别为 50X0 和 50X1，开发者可以根据通道号来进行选择端口号。当出现端口被占用时，通道号的那一位会发生改变，根据 ecat_server 程序的提示连接对应的板卡即可。

（2）运行 AWStudio 上位机软件。在 Windows 电脑上打开 AWStudio 运动控制版。AWStudio 以解决方案的形式来管理各类文件，因此，第一步先新建或打开一个解决方案。

AWStudio 启动后默认没有解决方案，点击界面中”新建”按钮或者工具栏的”新建”按钮，可弹出”新建解决方案”对话框，如图 4.3 所示。

解决方案类型选择”主站控制器/卡配置”，解决方案名称和解决方案路径按需要修改，点击”创建”按钮即可创建一个新的解决方案。

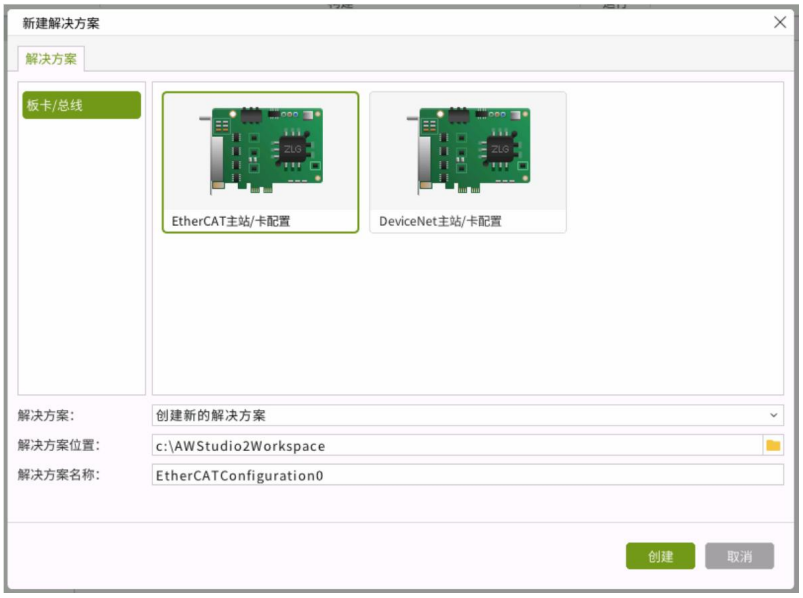


图 4.3 新建解决方案

（3）配置主站。填写 ecat_server 程序所在主机的 IP 地址(例如本主机 IP 地址为：192.168.1.200，如果 awstudio 和 ecat_server 在同一台主机，也可使用 127.0.0.1)。端口号使用上述 ecat_serverr 工具的结果，例如本通讯卡的端口号为 5010。

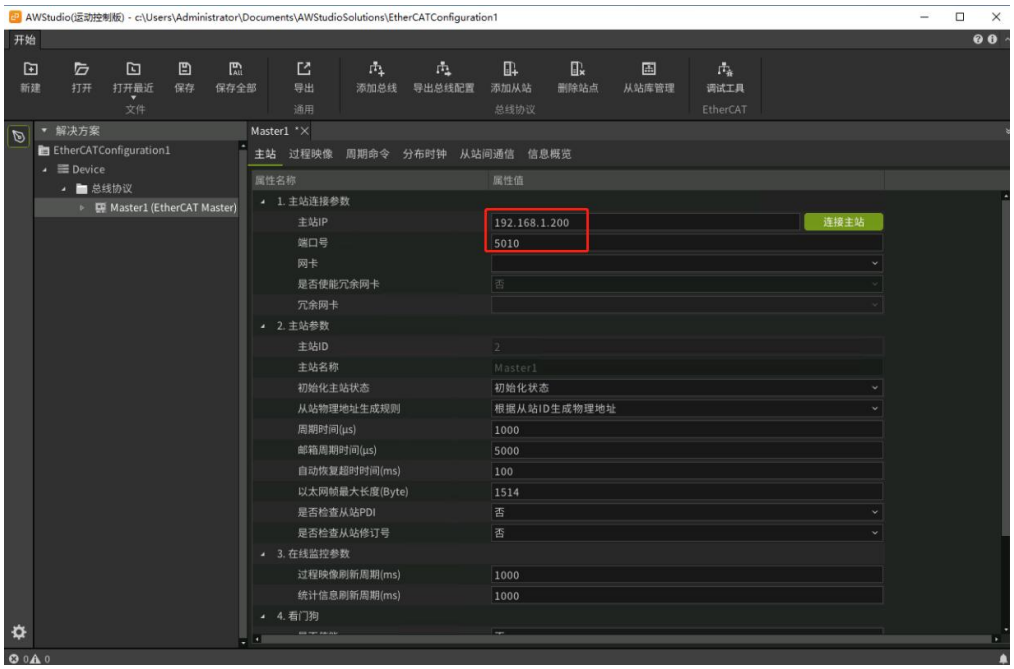


图 4.4 配置 IP 地址及端口号

（4）扫描从站设备，通过选中主站节点，右键菜单中“拓扑扫描”功能，如图 4.5 所示。扫描总线上的从站设备，扫描结果如下图 4.6 所示。

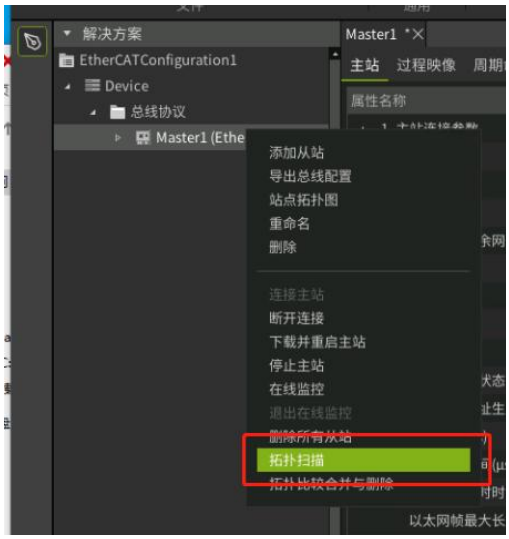


图 4.5 扫描从站

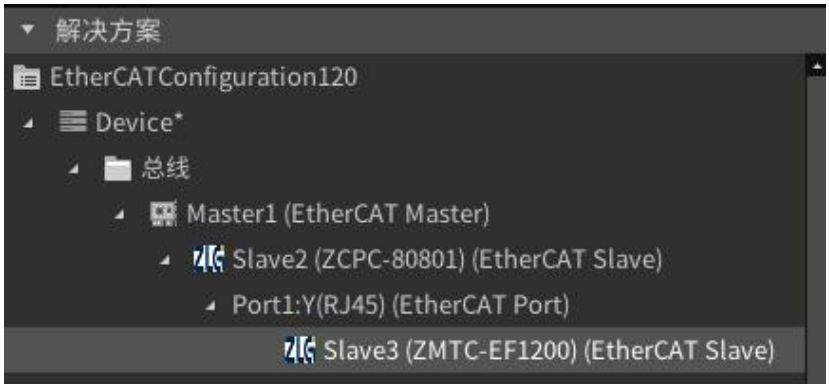


图 4.6 从站扫描结果

（5）导入从站 ESI 文件。如从站无法正确识别，可手动导入从站 ESI 文件，操作如下图 4.7 所示。

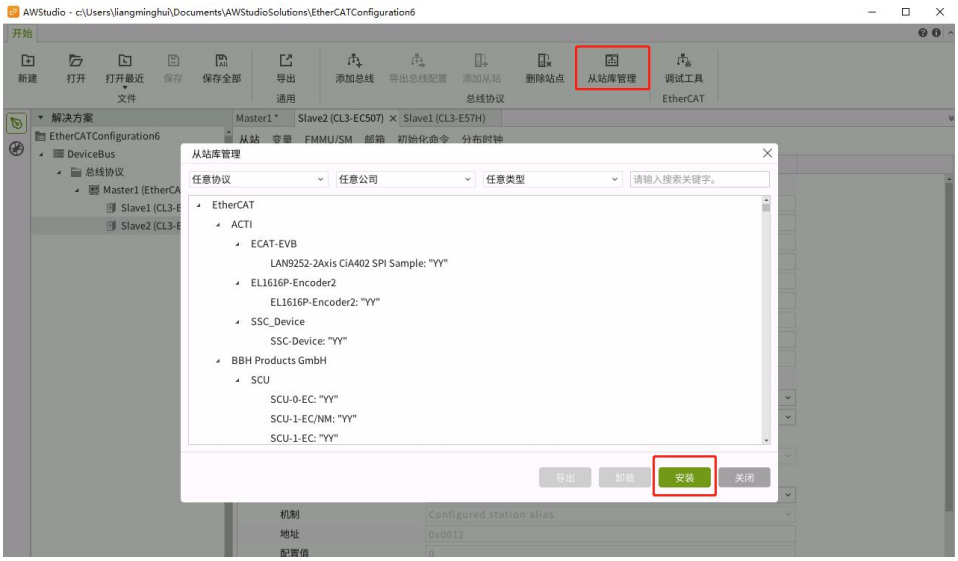


图 4.7 手动导入从站 ESI 文件

（6）PDO 过程数据配置。主要有从站 FMMU/SM 配置和 MDP 模块配置两种方式。

一般应用不需配置从站 FMMU/SM，使用从站默认配置即可。当需要改变输入输出变量时，可通过本配置项配置，如图 4.8 所示。

插板式模组应用

插板式 EtherCAT 系列从站

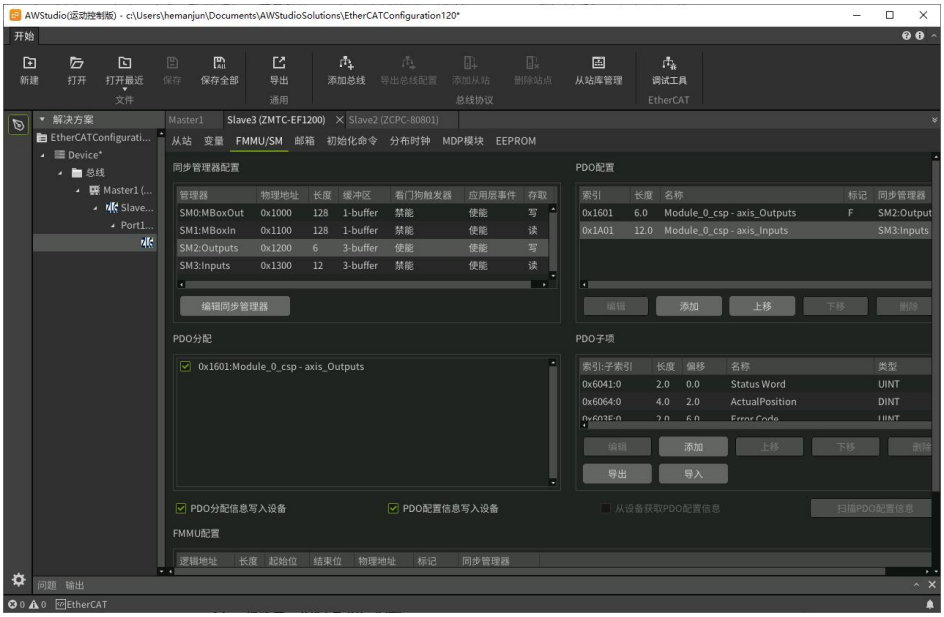


图 4.8 从站 FMMU/SM 配置

ZMTC-EF1200 支持通过 MDP 模块快速配置预定义的过程数据。在从站 MDP 模块页面，在右侧模块库提供了不同运行模式下使用的 PDO 配置。在左侧插槽配置，可以根据运行模式快速配置 PDO，如图 4.9 所示。

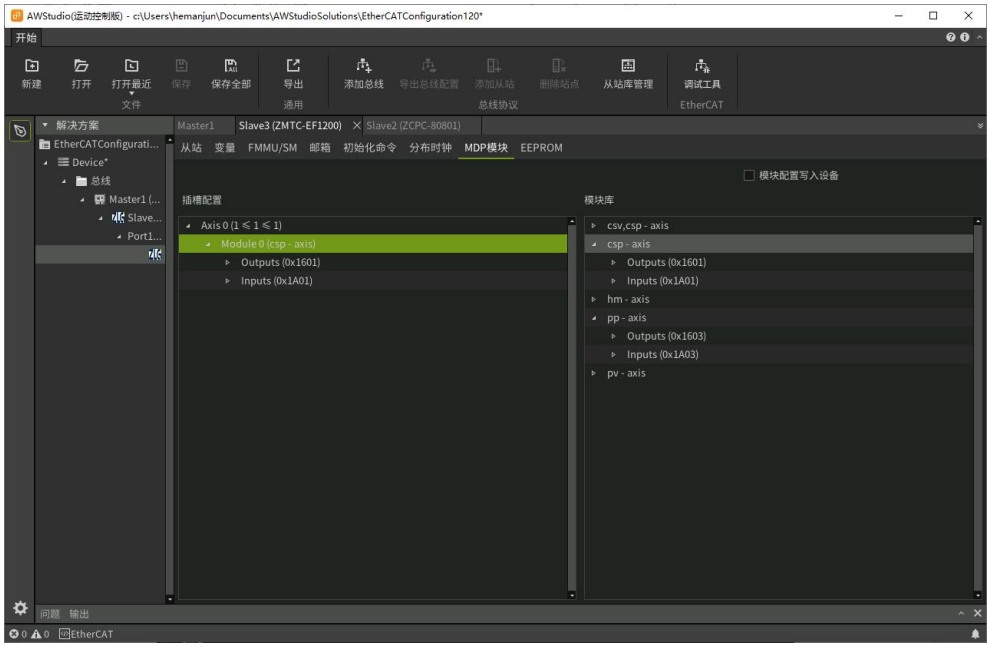


图 4.9 过 MDP 模块配置

（7）配置分布式时钟。支持自动配置分布时钟模式，默认以第一个支持分布时钟的从站作为时钟源，也可以手动配置，如图 4.10 所示。

插板式模组应用

插板式 EtherCAT 系列从站

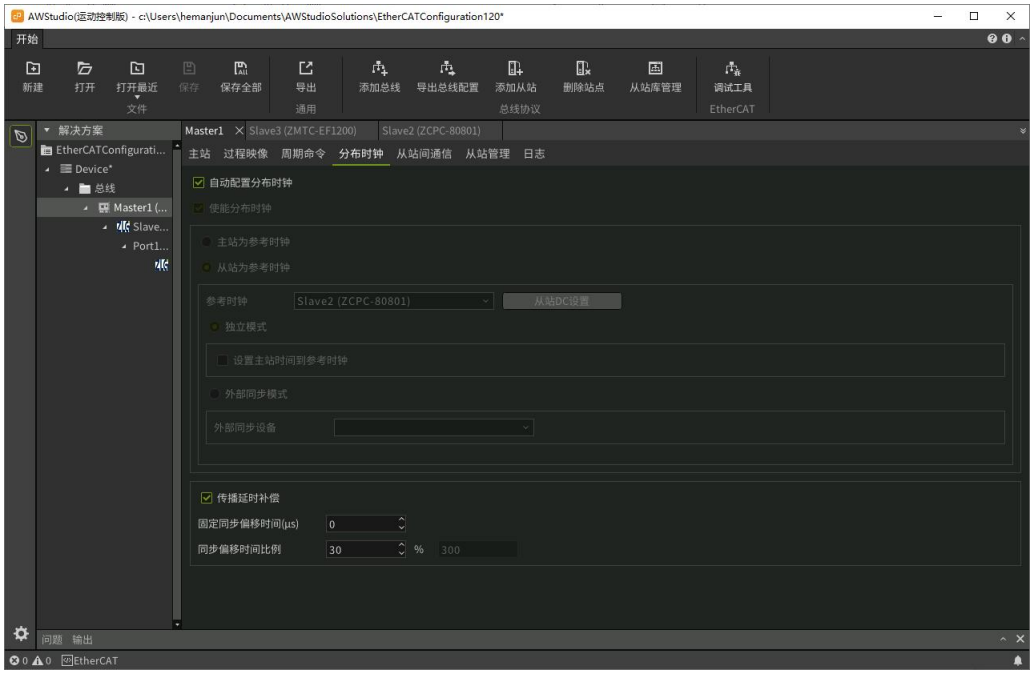


图 4.10 配置分布式时钟

（8）启动主站，并进入在线监控。完成以上配置后，右键“Master->下载并重启主站”，完成后右键“Master->在线监控”，如图 4.11 所示。

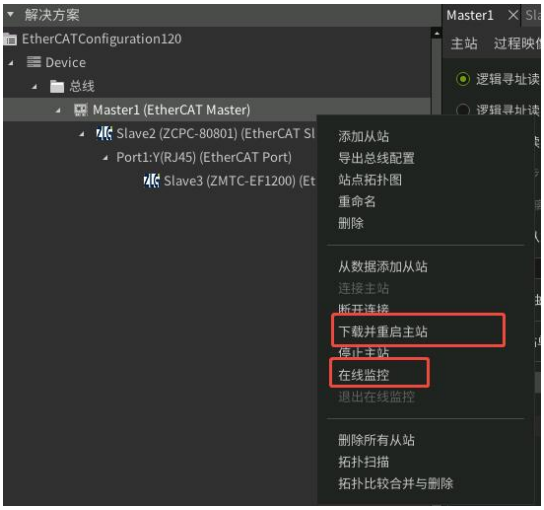


图 4.11 启动主站并监控

状态页需要先启动主站并进入在线监控状态才可用，可以查看并切换主站的状态（同时切换所有从站状态），可以查看诊断提示以及网络状态，如图 4.12 所示。读写 CoE 邮箱数据需要主站运行在“预操作状态”以上的运行级别，读写 PDO 数据需要主站运行在“操作状态”。

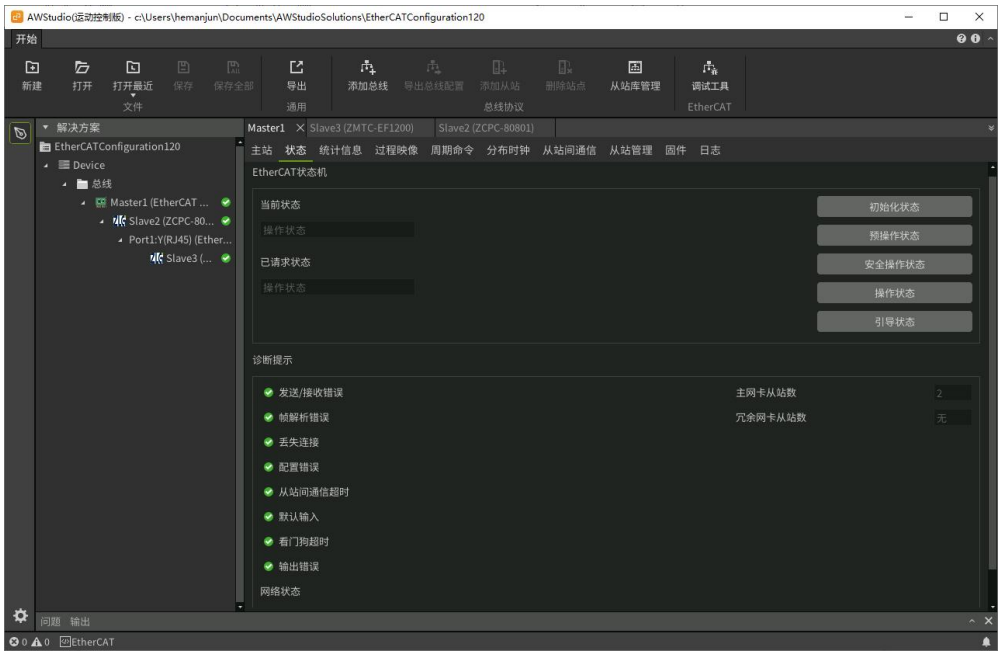


图 4.12 查看网络状态

（9）打开从站 ZMTC-EF1200 的“邮箱-CoE”页，找到 0x6060 对象，右键“设置值”，设置先切换到“预操作状态”，写电机运行模式为需要的模式，例如 CSP 模式为 8，如图 4.13 所示。

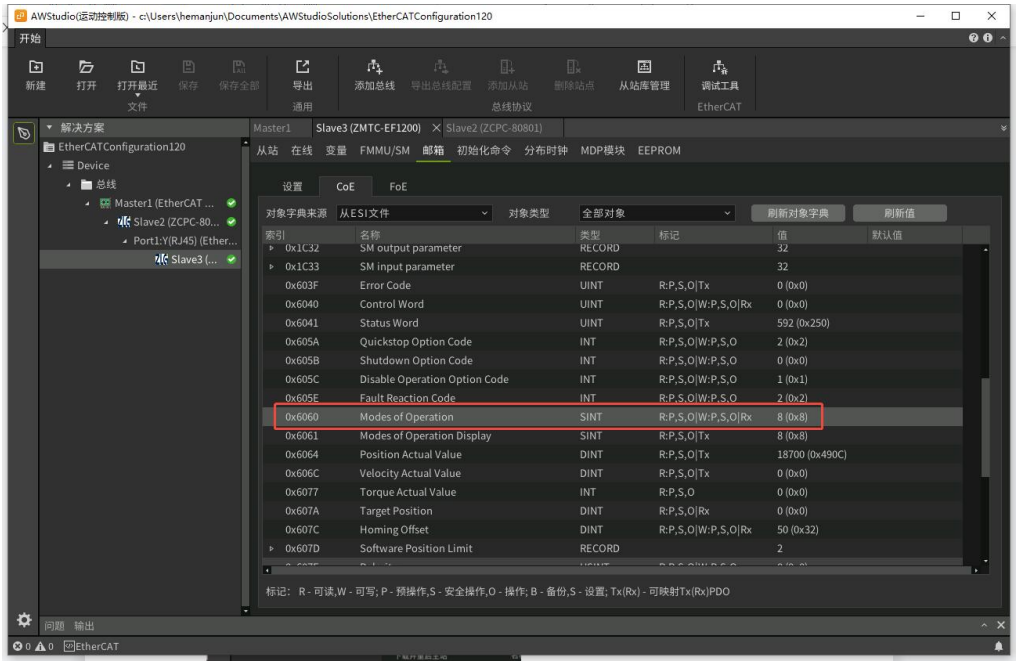


图 4.13 设置电机运行模式

（10）再切换到“操作模式”，通过写 PDO 数据控制电机。打开“Master-过程映像”页，右键要修改的变量，“强制设置值”即可改变值。

修改输出变量的 Control Word，按 0x6 0x7 0xF 依次写入来启动电机，启动成功后输入变量 Status Word 应为 0x1637。此时修改 TargetPosition 即可使电机转动，如图 4.14 所示。

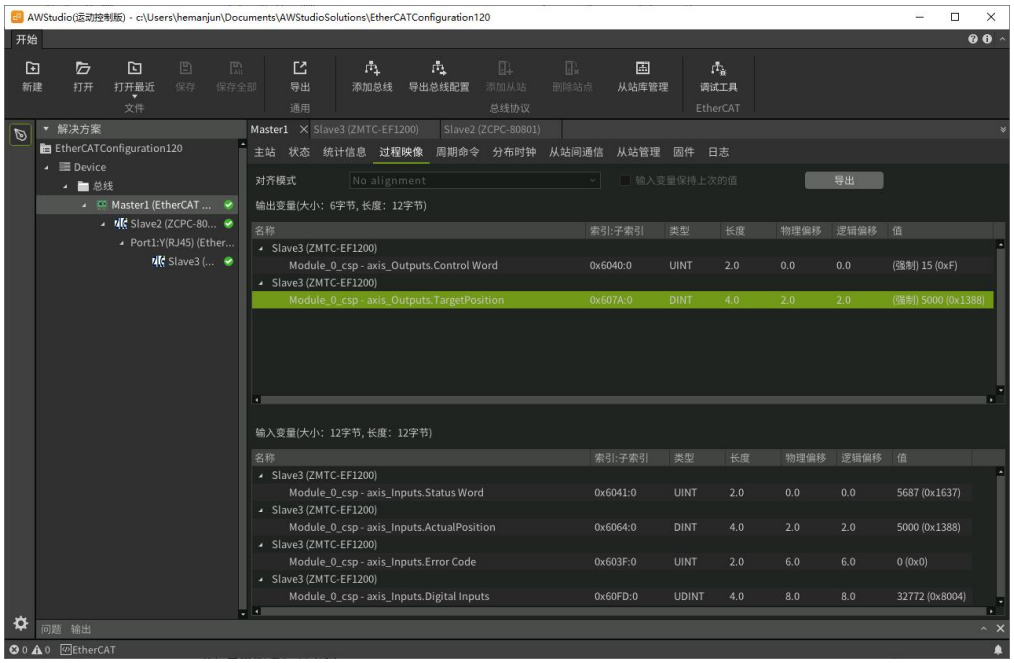


图 4.14 修改输出变量

4.2 通过编写 C 程序控制

（1）导出配置信息 ENI。在 4.1 示例中，总线配置完成后，可以右键“Master->导出总线配置”，将配置导出到 xml 文件，如图 4.15 所示。

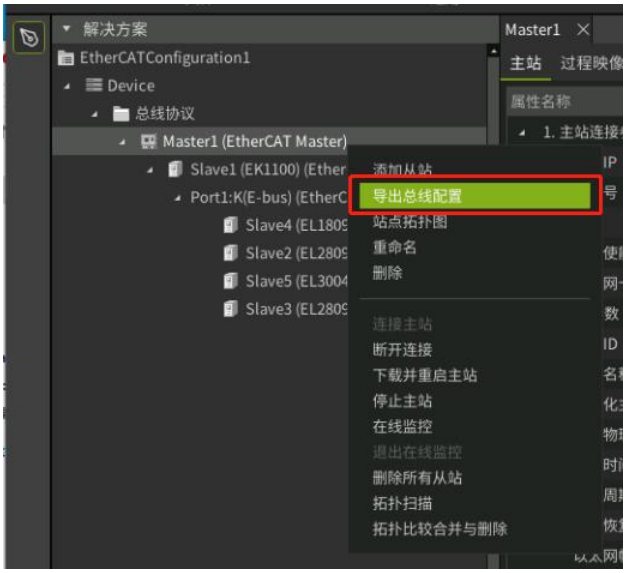


图 4.15 导出总线配置

（2）编写控制程序。本文以异步控制示例为例，电机将工作在 CSP 模式，本例程使用的 eni 文件为 eni.xml，用户需要生成自己的 ENI 文件进行替换。

PDO 是根据 ENI 配置的时间周期进行数据传输控制，而异步则是根据用户的实际应用时间周期进行数据传输。

相关函数说明：

1、EcatOpen() 根据通道号打开设备，获得句柄。

2、EcatBusRun() 启动并传入 ENI 文件。

3、EcatRequestMasterState() 请求切换状态。

4、EcatGetMasterState() 获得主站状态。

5、EcatPIEnable（）使能 PI 镜像缓存跟总线进行数据交换。

6、EcatPIRead（）从指定的缓存读取镜像输入数据。

7、EcatPIWrite（）写镜像输出数据到指定的缓存。

本例程实现电机的往返循环运动控制，执行程序同样需要输入板卡的索引 ID 和通道作为参数，ID 由别名编码器决定，例如本例程 ID 设置为为 1，通道根据产品型号决定，PCIe_2E 型号只有 1 个通道，参数为 0，而 PCIe_4E 型号是双主站，参数 0 或 1 可以选择相应的通道进项控制，例如使用通道 0 进行数据传输，则运行指令为：

```
make
sudo ./async_control_test 1 0
```

指令运行结果如下图 4.16 所示。

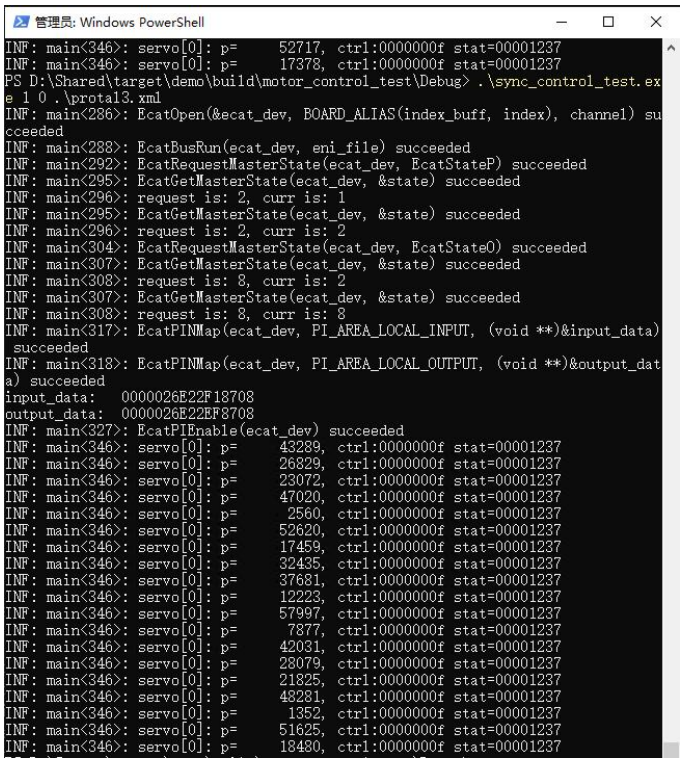


图 4.16 指令运行结果

（3）代码解析如下：

```
#define NUM_SERVOS   1              // 总电机数
```

宏 NUM_SERVOS 定义总电机数

```
// CiA402 协议电机控制模式
enum {
    OP_MODE_CSP = 8,
    OP_MODE_CSV = 9,
    OP_MODE_CST = 10,
};
static U32 g_op_mode = OP_MODE_CSP;
```

g_op_mode 设置电机运行模式，本示例为 CSP 模式。

```
////////////////////////////////////////////////////////////////////////////
//  过程映像
// CiA402 电机控制字
typedef union {
    struct {
        U16 switch_on: 1;
        U16 enable_voltage: 1;
        U16 quick_stop: 1;
        U16 enable_operation: 1;
        U16 pad0: 3;
        U16 fault_reset: 1;
        U16 halt: 1;
        U16 pad1: 1;
        U16 pad2: 6;
    } b;
    U16 v;
} servo_ctrl_word;
// CiA402 电机状态字
typedef union {
    struct {
        U16 ready_to_switch_on: 1;
        U16 switched_on: 1;
        U16 operation_enabled: 1;
        U16 fault: 1;
        U16 voltage_enabled: 1;
        U16 quick_stop: 1;
        U16 switch_on_disabled: 1;
        U16 warning: 1;
        U16 pad0: 1;
        U16 remote: 1;
        U16 target_reached: 1;
        U16 internal_limit_active: 1;
        U16 pad1: 4;
    } b;
    U16 v;
} servo_stat_word;
```

配置过程映像，首先定义了 CiA402 协议控制字和状态字的含义。

```
// 存储电机控制变量指针，指向 PDO Output 对应变量
typedef struct {
    uint8_t                 *mode;
    servo_ctrl_word         *ctrl;
    I32                      *p;
} servo_ctrl;
// 存储电机状态变量指针，指向 PDO Input 对应变量
typedef struct {
    uint8_t                 *mode;
    servo_stat_word         *stat;
    I32                      *p;
} servo_stat;
typedef struct {
    servo_ctrl servos[NUM_SERVOS];
} proc_img_o;
typedef struct {
    servo_stat servos[NUM_SERVOS];
} proc_img_i;
// 存储用户配置电机控制变量在过程映像中的偏移量
typedef struct {
    int32_t                 po_mode_offset;
    int32_t                 po_ctrl_word_offset;
    int32_t                 po_p_offset;
    int32_t                 pi_mode_offset;
    int32_t                 pi_stat_offset;
    int32_t                 pi_p_offset;
} pi_offset_config;
static pi_offset_config   __pi_config[] = {
    {
        .po_mode_offset = -1,
        .po_ctrl_word_offset = 0,
        .po_p_offset = 2,
        .pi_mode_offset = -1,
        .pi_stat_offset = 0,
        .pi_p_offset = 2,
    },
};
```

servo_ctrl 和 servo_stat 两个结构体存储指向过程映像中电机控制相关变量的指针。指针通过 pi_offset_config 存储的偏移量在 PI 映像中求出，通过偏移求指针在 pi_get_output 和 pi_get_input 函数中获得。

```
proc_img_o __pi_o;
proc_img_i  __pi_i;
// 根据偏移量计算获取过程映像中变量对应的指针
static proc_img_o *pi_get_output(const pi_offset_config *config, void *output) {
    for (uint32_t i = 0; i < NUM_SERVOS; i++) {
        servo_ctrl *ctrl = &__pi_o.servos[i];
        const pi_offset_config *curr = &config[i];
        ctrl->mode = (uint8_t *)((curr->po_mode_offset != -1) ? (char *)output + curr->po_mode_offset : 0);
        ctrl->ctrl = (servo_ctrl_word *)((curr->po_ctrl_word_offset != -1) ? (char *)output + curr->po_ctrl_word_offset : 0);
        ctrl->p = (I32 *)((curr->po_p_offset != -1) ? (char *)output + curr->po_p_offset : 0);
    }
    return &__pi_o;
}
static proc_img_i *pi_get_input(const pi_offset_config *config, const void *output) {
    for (uint32_t i = 0; i < NUM_SERVOS; i++) {
        servo_stat *stat = &__pi_i.servos[i];
        const pi_offset_config *curr = &config[i];
        stat->mode = (uint8_t *)((curr->pi_mode_offset != -1) ? (char *)output + curr->pi_mode_offset : 0);
        stat->stat = (servo_stat_word *)((curr->pi_stat_offset != -1) ? (char *)output + curr->pi_stat_offset : 0);
        stat->p = (I32 *)((curr->pi_p_offset != -1) ? (char *)output + curr->pi_p_offset : 0);
    }
    return &__pi_i;
}
```

用户适配、增加和减少电机时，主要修改__pi_config 变量即可，增删对应变量的偏移。

```
// 电机模式初始化，向 0x6060 写模式
static int motor_mode_init(ECAT_HANDLE ecat_dev,uint32_t op_mode) {
    int ret = -1; uint32_t mode = 0; uint32_t len; int fail_try = 100; int i;
    for(i = 0; i < NUM_SERVOS && fail_try>0; ) {
        len = 1;
        EXIT_IF_FAILED(EcatCoeSDODownload(ecat_dev,i,0x6060,0x0,1,&op_mode,100));
        usleep(1000*100);
        EXIT_IF_FAILED(EcatCoeSDOUpload(ecat_dev,i,0x6060,0x0,&len,&mode,100));
        printf("motor[%d] set_mode:%d mode:%d\r\n",i,op_mode,mode);
        if(mode != op_mode) {
            usleep(1000*100);
            fail_try --;
            continue;
        }
        fail_try = 100;
        i++;
    }
    return ret;
}
```

motor_mode_init 初始化运行模式，主要是通过 SDO 协议写邮箱 0x6060 对象，设置运

行模式。

```
// 电机状态初始化，初始化 PDO 变量中的速度、位置、力矩等变量
static int motor_state_init(ECAT_HANDLE ecat_dev,ECAT_ENI_INFO ptInfo) {
    int    i = 0;
    int fail_try = 1000;
    bool run = false;
    uint32_t max_speed = 5000;
    for (i = 0; i < NUM_SERVOS && fail_try > 0;) {
        servos_t *servo = &__servos[i];
        if(EcatPIRead(ecat_dev,PI_AREA_SHADOW_INPUT,0,ptInfo.u32PIInputByteSize,input_buff)) {
            printf("EcatPIRead Fail\r\n");
        }
        run = state_control(i, &servo->ss, pi->servos[i].stat, &servo->sc, false);
        usleep(1000);
        // 更新输出变量
        *po->servos[i].ctrl = servo->sc;
        *po->servos[i].p = 0;
        if (EcatPIWrite(ecat_dev,PI_AREA_FAST_OUTPUT,0,ptInfo.u32PIOutputByteSize,output_buff) !=
0) {
            printf("EcatPIWrite Fail\r\n");
        }
        if(!run) {
            fail_try--;
            continue;
        }
        //初始位置初始化
        servo->out_p = *pi->servos[i].p;
        servo->zero_p = (float)servo->out_p;
        servo->offset_p = 0;
        servo->out_v = 0;
        servo->out_t = 0;
        servo->reverse = false;
        //初始化最高速度
        EcatCoeSDODownload(ecat_dev,i,0x6081,0x0,4,&max_speed,100);
        fail_try = 1000;
        i++;
    }
    return run? 0 : -1;
}
```

motor_mode_init 电机状态初始化，初始化 PDO 变量中的速度、位置、力矩等变量。

插板式模组应用

插板式 EtherCAT 系列从站

```
// CiA402 协议实现，根据状态字修改控制字
static bool motor_control(int axis, servo_stat_word *s0, const servo_stat_word *s1, servo_ctrl_word *c, bool show) {
    servos_t *servo = &__servos[axis];
    do {
        c->v = 0;
        c->b.enable_voltage = 1;
        c->b.quick_stop = 1;
        c->b.enable_operation = s1->b.switched_on || s1->b.operation_enabled;
        c->b.switch_on = s1->b.ready_to_switch_on;
        if (s1->b.fault && !servo->fault_reset_delay)  {
            // 延时 1 秒再尝试错误恢复，防止伺服在错误和恢复中反复快速切换损坏电路
            _DBG_("axis %d: delay fault reset, ctrl:%d, stat:%x", axis, c->v, (int)s1->v);
            servo->fault_reset_delay = 2000; // 2s(即 2000 个周期)
            servo->run_delay = 2000; // 当错误恢复成功伺服出力后，延时 2s 再开始加速(根据抱闸动作时间适当调整)
            break;
        }
        // 伺服发生错误，且在延时等待处理中
        if (servo->fault_reset_delay)  {
            servo->fault_reset_delay--;
            if (!servo->fault_reset_delay)  {
                // 延时减到 0，尝试进行错误恢复
                _DBG_("axis %d: start fault reset", axis);
                c->b.fault_reset = 1;
            }
            break;
        }
    } while (0);
    s0->v = s1->v;
    return s1->b.operation_enabled;
}
```

CiA402 协议需要维护状态机的切换，在 motor_control 函数中实现，包括电机启动和错误响应及处理。

```
RETURN_IF_FAILED(EcatOpen(&ecat_dev, BOARD_ALIAS(index_buff, index), channel));
RETURN_IF_FAILED(EcatBusRun(ecat_dev, “protal3.xml”));
```

然后是 EtherCAT 控制的部分。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。

```
// 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(500000);
    if (state == EcatStateO) break;
}
```

然后请求 EtherCAT 状态机切换到操作状态。

```
EcatGetEniInfo(ecat_dev,&ptInfo);
printf("InputByteSize:%d\r\n",ptInfo.u32PIInputByteSize);
printf("OutputByteSize:%d\r\n",ptInfo.u32PIOutputByteSize);
input_buff    = malloc(1024);
output_buff = malloc(1024);
memset(input_buff, 0, 1024);
memset(output_buff, 0, 1024);
if(input_buff == NULL || output_buff == NULL)
{
    printf("malloc fail\r\n");
    return 0;
}
pi = pi_get_input(__pi_config,input_buff);
po = pi_get_output(__pi_config,output_buff);
```

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。

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

在控制电机和初始化电机状态之前，需要先使能 PI 镜像交换

```
    while (1) {
        loops++;
        // PI Read，获取过程变量 Input 变量
        if(!EcatPIRead(ecat_dev,PI_AREA_SHADOW_INPUT,0,ptInfo.u32PIInputByteSize,input_buff)) {
            for (int i = 0; i < NUM_SERVOS; i++) {
                servos_t *servo = &__servos[i];
                if (loops % 10 == 0)  {
                    _INF_("IN  servo[%d]: p=%10d, ctrl:%08x stat=%08x", i, *pi->servos[i].p,
po->servos[i].ctrl->v, pi->servos[i].stat->v);
                    _INF_("OUT servo[%d]: p=%10d", i,  *po->servos[i].p);
                }
                // 电机控制实现，计算位置、速度、力矩
                // 伺服已出力，延时结束，进行正常控制
                // 不同轴单位可能不一样，下表定义各轴位置范围及增量
                I32 range_min[] = { -60000 };
                                    I32 range_max[] = { 60000 };
                float step_size[] = { 50 };
                if (servo->offset_p < range_min[i]) servo->reverse = false;
                if (servo->offset_p > range_max[i]) servo->reverse = true;
                servo->offset_p += (servo->reverse ? -1 : 1) * step_size[i];
                servo->out_p = (I32)(servo->zero_p + servo->offset_p);
                servo->out_p = MIN_LIMIT(servo->out_p ,range_min[i]);
                servo->out_p = MAX_LIMIT(servo->out_p ,range_max[i]);
                // 更新输出变量
                po->servos[i].ctrl->v = 0x2F;
                *po->servos[i].p = servo->out_p;
            }
            // PI Write，修改过程变量 Output 变量
            if
(EcatPIWrite(ecat_dev,PI_AREA_FAST_OUTPUT,0,ptInfo.u32PIOutputByteSize,output_buff) != 0)  {
                _ERR_("write PI_AREA_FAST_OUTPUT error");
                break;
            }
            usleep(1000*2);
            //pp mode need 0x6040: 0x2f -> 0x3f
            for (int i = 0; i < NUM_SERVOS; i++) {
                po->servos[i].ctrl->v = 0x3F;
                servos_t *servo = &__servos[i];
            }
            // PI Write，修改过程变量 Output 变量
            if
(EcatPIWrite(ecat_dev,PI_AREA_FAST_OUTPUT,0,ptInfo.u32PIOutputByteSize,output_buff) != 0) {
                _ERR_("write PI_AREA_FAST_OUTPUT error");
                break;
            }
        }
        usleep(1000);
    }
```

控制电机往复运动的主要逻辑在 main 函数中循环实现。在 CSP 模式下，程序通过计算

得到的电机位置后，写到 PDO 变量中。然后主站卡在下一次周期将更新后的电机位置发送给从站，从站就能控制电机转动。

由于异步模式下，主站使用独立的循环周期，所以程序可任意时刻读取从站的数据并输出数据到从站。

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**