DMC运动控制函数库说明

[1. 引言 3](#_Toc13442)

[2. EtherCAT 简介 4](#_Toc23669)

[2.1. EtherCAT特点 4](#_Toc10247)

[2.2. 状态机 4](#_Toc29700)

[2.3. EtherCAT 协议 5](#_Toc22818)

[2.3.1. On the FLY处理方式 5](#_Toc3832)

[2.3.2. EtherCAT帧结构 6](#_Toc15509)

[2.3.3. EtherCAT数据报结构 6](#_Toc2638)

[2.4. 从站地址 8](#_Toc4518)

[2.5. FMMU 9](#_Toc3463)

[2.6. SM 10](#_Toc31191)

[2.6.1. 两种缓存模式 11](#_Toc5118)

[2.6.2. 基于 EtherCAT 的 CAN 应用协议（CoE） 11](#_Toc27507)

[2.6.3. SDO的处理 12](#_Toc27076)

[2.6.4. PDO的处理 12](#_Toc15730)

[2.7. 分布式时钟与同步 14](#_Toc116)

[2.7.1. 分布式时钟DC 14](#_Toc16253)

[2.7.2. 传输时延的测量、偏移量补偿 15](#_Toc25507)

[2.7.3. 漂移补偿 16](#_Toc10014)

[2.7.4. 同步方式 17](#_Toc4567)

[3. DMC运动控制库 19](#_Toc9053)

[3.1. ECM主站 19](#_Toc26153)

[3.2. 连接方式 19](#_Toc12890)

[3.3. 配置文件 20](#_Toc24572)

[3.4. 接口函数 21](#_Toc28837)

[3.4.1. 初始化、关闭 21](#_Toc16755)

[3.4.2. 单轴位置 22](#_Toc6239)

[3.4.3. 直线插补 24](#_Toc649)

[3.4.4. Z轴拱门插补 24](#_Toc22128)

[3.4.5. 停止函数 26](#_Toc17913)

[3.4.6. 回原点运动 26](#_Toc3666)

[3.4.7. 指令脉冲计数 27](#_Toc9818)

[3.4.8. IO控制 27](#_Toc10557)

[4. 示例程序 27](#_Toc9492)

[4.1. 单轴点位 28](#_Toc13245)

[4.2. 直线插补 28](#_Toc4643)

[4.3. 回原点 29](#_Toc25853)

[4.4. IO模块 29](#_Toc15038)

[4.5. Z轴拱门插补 30](#_Toc17728)

[5. 编程举例 30](#_Toc24151)

[5.1. 开发环境 30](#_Toc5482)

[5.2. 运行环境 32](#_Toc9369)

# 引言

DMC运动控制函数库是基于讯成科技EtherCAT Master Control IC/Starter Kit封装的动态库，用于EtherCAT 主站的开发。函数库接口风格与DMC1000∕DMC1000B 类似，实现的功能包括：单轴位置（梯形或S型速度曲线）、回原点、减速停止、急停、多轴直线插补以及通用输入输出。

本函数库接口简单，易学易用，具备完善的日志记录功能，方便进行问题定位及排查。

# EtherCAT 简介

EtherCAT（以太网控制自动化技术）是一个开放架构，以以太网为基础的现场总线系统，其名称的CAT为控制自动化技术（Control Automation Technology）字首的缩写,最早是由德国的Beckhoff公司研发。

## EtherCAT特点

* 卓越的性能：总的来说 EtherCAT 是最快的工业以太网技术，同时它提供纳秒级精确的同步。
* 灵活的拓扑：在传统的工业以太网系统中，可安装的交换机和集线器的级联数量是有限的，这限制了整个的网络拓扑结构。而 EtherCAT 无需交换机或集线器，因此没有这样的局限性。
* 简单且耐用：EtherCAT 可以自动配置地址，而无需手动配置。在启动时，网络将目标拓扑与现实拓扑对比从而检测差异。
* 低成本易实现：主站设备硬件仅需要一个以太网端口，EtherCAT 从站控制器可以从很多供应商获得：ASIC芯片、FPGA。 EtherCAT 自身并不向从站设备 CPU 提出任何性能要求，从而降低了设备成本。

## 状态机

以下为EtherCAT状态转换框图，EtherCAT设备必须支持4种状态，负责协调主站和从站应用程序在初始化和运行时的状态关系。

Init：初始化，简写为I；

Pre-Operational:预运行，简写为P；

Safe-Operational:安全运行，简写为S;

Operational:运行，简写为O。

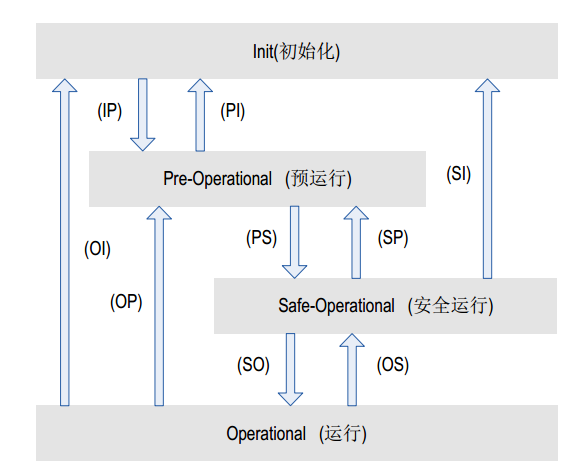


图 2-1 EtherCAT状态机

从初始化状态向运行状态转化时，必须按照“初始化→预运行→安全运行→运行”的顺序转化，不可以越级。从运行状态返回时可以越级转化。状态的转化操作和初始化过程如下表所示：

表 2-1 状态转换

|  |  |
| --- | --- |
| 状态和状态转化 | 操作 |
| 初始化(I) | 应用层没有通信，主站只能读写ESC寄存器 |
| IP | 主站配置从站站点地址；  配置邮箱通道；  配置DC分布时钟；  请求“预运行状态” |
| 预运行(P) | 应用层邮箱数据通信(SDO) |
| PS | 主站使用邮箱初始化过程数据；  主站配置过程数据通信使用的SM通道；  主站配置FMMU；  请求“安全状态” |
| 安全运行(S) | 有过程数据通信，但是只允许读输入数据，不产生输出信号 |
| SO | 主站发出有效的输出数据；  请求“运行状态” |
| 运行状态(O) | 输入和输出全部有效；  仍然可以使用邮箱通信 |

## EtherCAT 协议

### On the FLY处理方式

当报文传播到某个从站时，从站将处理结果填充到对应数据区，再将报文发送到下一个从站。最终报文返回主站，主站对数据区中的数据进行处理。

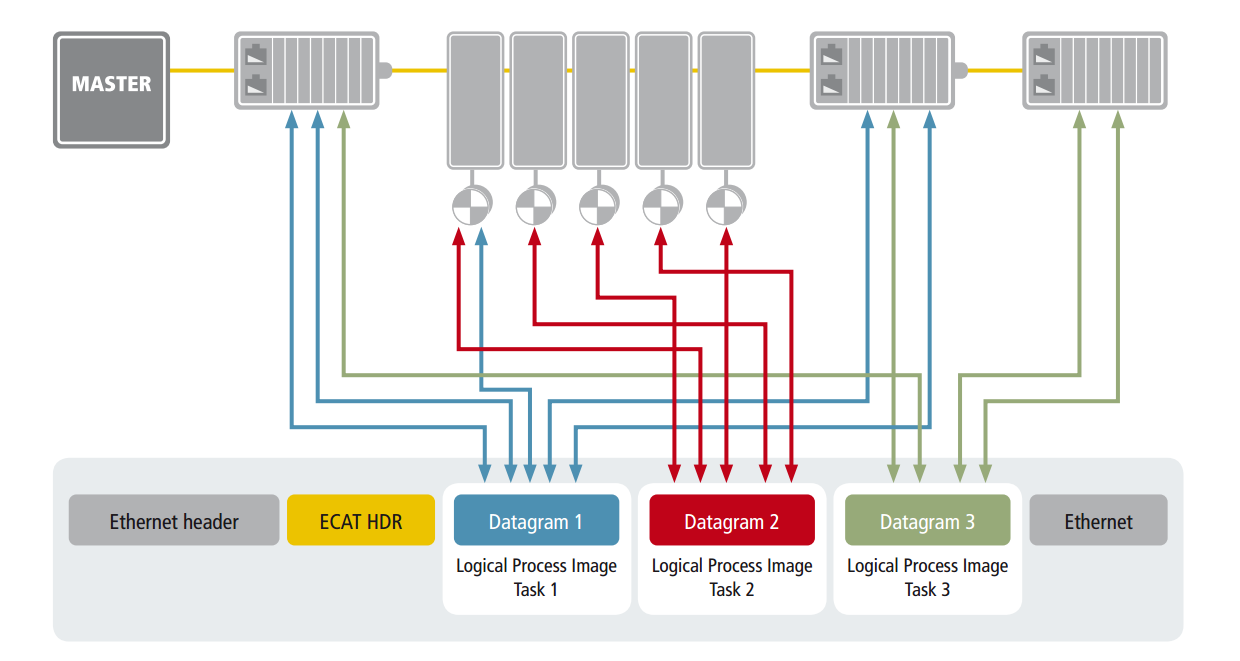


图 2-2 高速动态地插入过程数据（“on the fly”）

### EtherCAT帧结构

EtherCAT 将其报文嵌入到标准的以太网数据帧中（形成 EtherCAT 数据帧）。EtherCAT数据帧由EtherCAT帧头、数据报区组成，数据报区可能含有多个数据报。

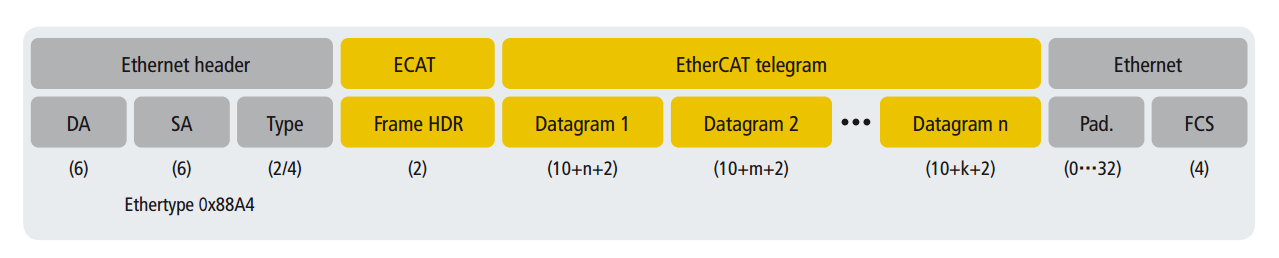


图 2-3 EtherCAT 报文结构

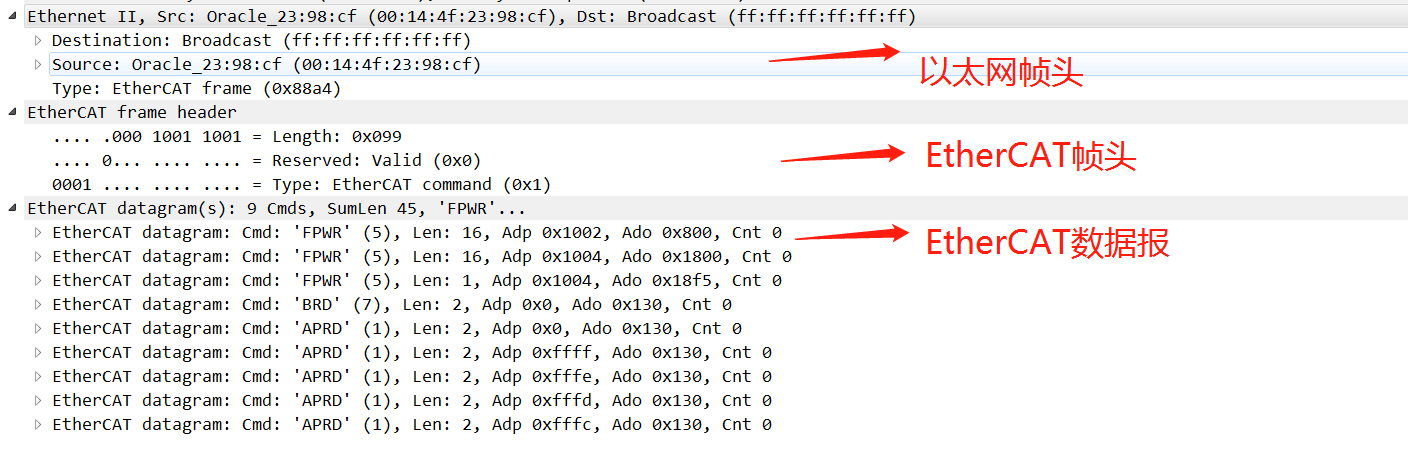


图 2-4 EtherCAT报文实例

### EtherCAT数据报结构

EtherCAT数据报由CMD及SlaveAddr决定寻址方式，各从站可以读取、写入Data部分。

表 2-2 数据报结构

|  |  |  |
| --- | --- | --- |
| 数据报 | 说明 | 长度（字节） |
| Cmd | 命令类型 | 1 |
| Index | 数据报索引，递增 | 1 |
| Slave Addr | 从站地址 | 2 |
| Offset Addr | 偏移量 | 2 |
| Length | 数据区长度、标志位 | 2 |
| Interrupt | 中断标志 | 2 |
| Data | 二进制数据区 | 可变 |
| WKC | WKC计数 | 2 |

表 2-3 命令类型

|  |  |  |  |
| --- | --- | --- | --- |
| CMD | **Abbr.** | **Name** | **Description** |
| 0 | NOP | No Operation | Slave ignores command |
| 1 | APRD | Auto Increment Read | Slave increments address. Slave puts read data into the EtherCAT datagram if received address is zero. |
| 2 | APWR | Auto Increment Write | Slave increments address. Slave writes data into memory location if received address is zero. |
| 3 | APRW | Auto Increment Read Write | Slave increments address. Slave puts read data into the EtherCAT datagram and writes the data into the same memory location if received address is zero. |
| 4 | FPRD | Configured Address Read | Slave puts read data into the EtherCAT datagram if address matches with one of its configured addresses |
| 5 | FPWR | Configured Address Write | Slave writes data into memory location if address matches with one of its configured addresses |
| 6 | FPRW | Configured Address Read Write | Slave puts read data into the EtherCAT datagram and writes data into the same memory location if address matches with one of its configured addresses. |
| 7 | BRD | Broadcast Read | All slaves put logical OR of data of the memory area and data of the EtherCAT datagram into the EtherCAT datagram. All slaves increment position field. |
| 8 | BWR | Broadcast Write | All slaves write data into memory location. All slaves increment position field. |
| 9 | BRW | Broadcast Read Write | All slaves put logical OR of data of the memory area and data of the EtherCAT datagram into the EtherCAT datagram, and write data into memory location. BRW is typically not used. All slaves increment position field. |
| 10 | LRD | Logical Memory Read | Slave puts read data into the EtherCAT datagram if received address matches with one of the configured FMMU areas for reading. |
| 11 | LWR | Logical Memory Write | Slaves writes data to into memory location if received address matches with one of the configured FMMU areas for writing. |
| 12 | LRW | Logical Memory Read Write | Slave puts read data into the EtherCAT datagram if received address matches with one of the configured FMMU areas for reading. Slaves writes data to into memory location if received address matches with one of the configured FMMU areas for writing |
| 13 | ARMW | Auto Increment Read Multiple Write | Slave increments address. Slave puts read data into the EtherCAT datagram if received address is zero, otherwise slave writes the data into memory location. |
| 14 | FRMW | Configured Read Multiple Write | Slave puts read data into the EtherCAT datagram if address matches with one of its configured addresses, otherwise slave writes the data into memory location. |
| 15-255 |  | reserved |  |

表 2-4 WKC计数规则

|  |  |  |
| --- | --- | --- |
| Cmd | Data Type | Increment |
| Read Command | No success | No change |
| Successful read | +1 |
| Write Command | No success | No change |
| Successful write | +1 |
| ReadWrite Command | No success | No change |
| Successful read | +1 |
| Successful write | +2 |
| Successful read and write | +3 |

**错误！未定义书签。**、**错误！未定义书签。**展示了主站广播读取0x130的实际报文，从站返回的报文中WC计数为5，代表共有5个从站，所有从站均处于操作模式，均未出现错误。



图 2-5 主站读取报文

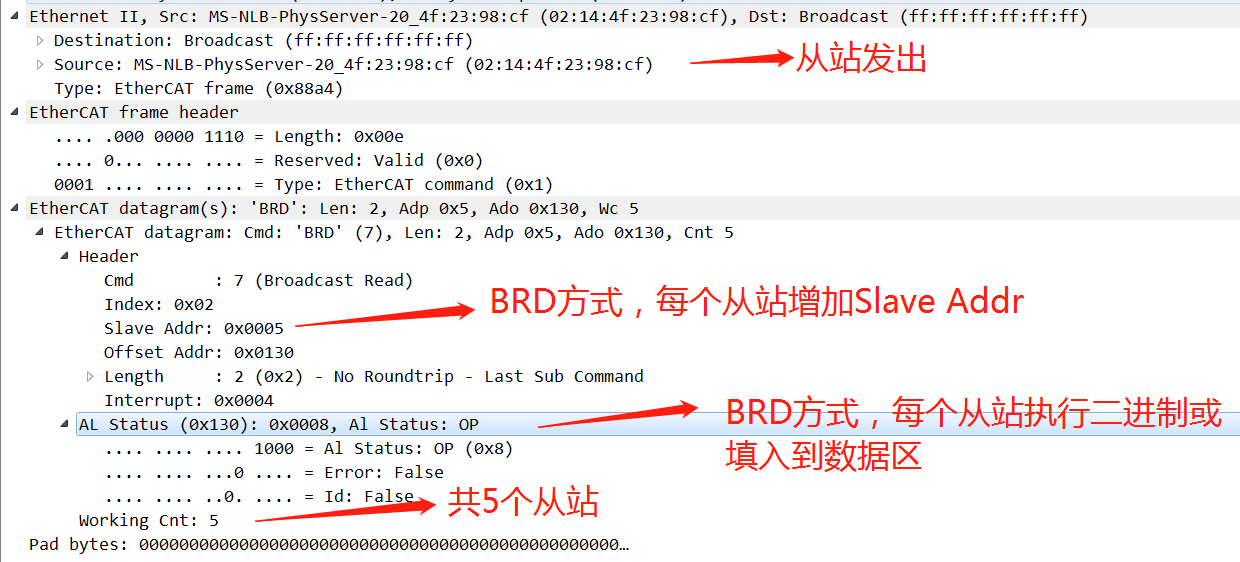


图 2-6 从站返回报文

## 从站地址

EtherCAT 可以自动配置地址，而无需手动配置。在检查完网络配置后， EtherCAT 主站为每个节点分配一个配置好的节点地址，并通过该固定的地址与节点进行通信。这使得主站可以有针对性地访问某个从站设备，即使网络拓扑结构在操作过程中发生改变，例如对于热插拔组。下面简单介绍自动配置地址的实现方式。

1. 主站BRD 0x130，返回的WKC计数代表当前总线上连接的从站数量；
2. 主站APWR 0x10配置从站地址，第一个从站的Slave Addr为0，第二个从站的Slave Addr为0xFFFF(-1)，第三个从站的Slave Addr为0xFFFE(-2)。



## FMMU

Fieldbus Memory Management Units (FMMU)通过内存地址映射的方式，将逻辑地址转换为物理地址，可以使多个从站共享一个逻辑地址空间。

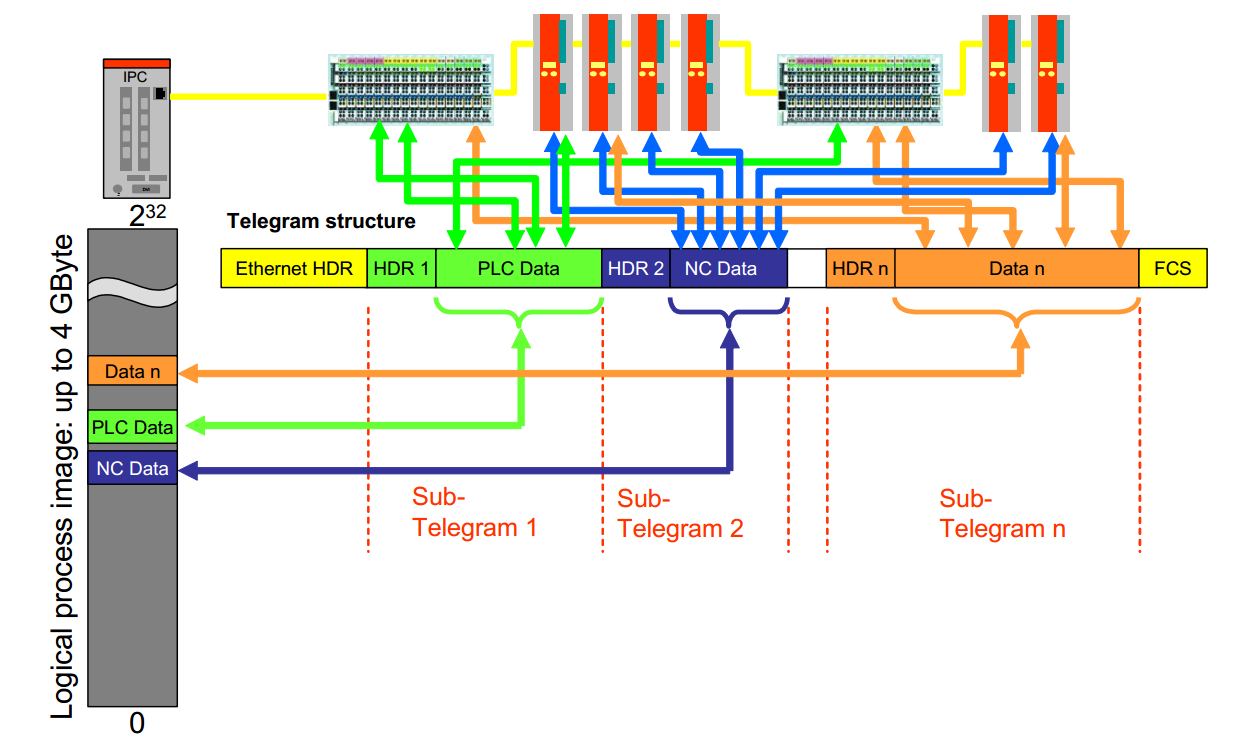


图 2-7 FMMU映射示意图

假设需要将逻辑地址从0x00010011[3] 到 0x00010013[0]，共14比特，映射到物理寄存器0x0F01[1]：0x0F02[6]。

表 2-5 FMMU实例

|  |  |  |
| --- | --- | --- |
| FMMU寄存器 | 寄存器内偏移 | 值 |
| Logical Start Address | 0x0:0x3 | 0x00010011 |
| Length (Bytes) | 0x4:0x5 | 3 |
| Logical Start bit | 0x6 | 3 |
| Logical Stop bit | 0x7 | 0 |
| Physical Start Address | 0x8:0x9 | 0x0F01 |
| Physical Start bit | 0xA | 1 |
| Type | 0xB | read and/or write |
| Activate | 0xC | 1 (enabled) |

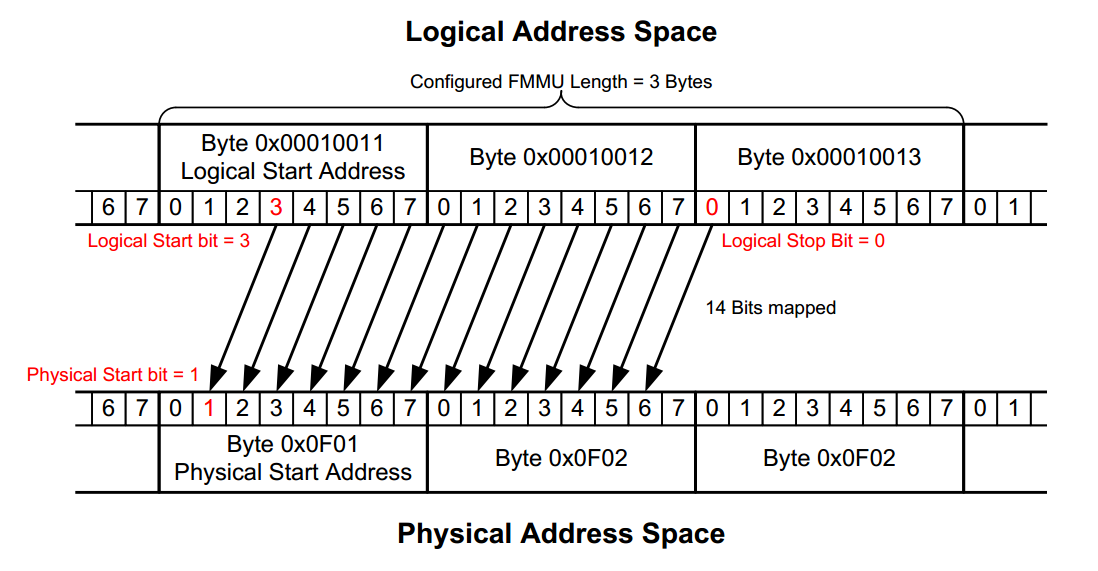


图 2-8 FMMU映射实例

## SM

SM（Sync Manager）用于保证从站控制器ESC内存被主站通过ECAT接口，以及从站本地控制器通过PDI接口同时访问的数据一致性。

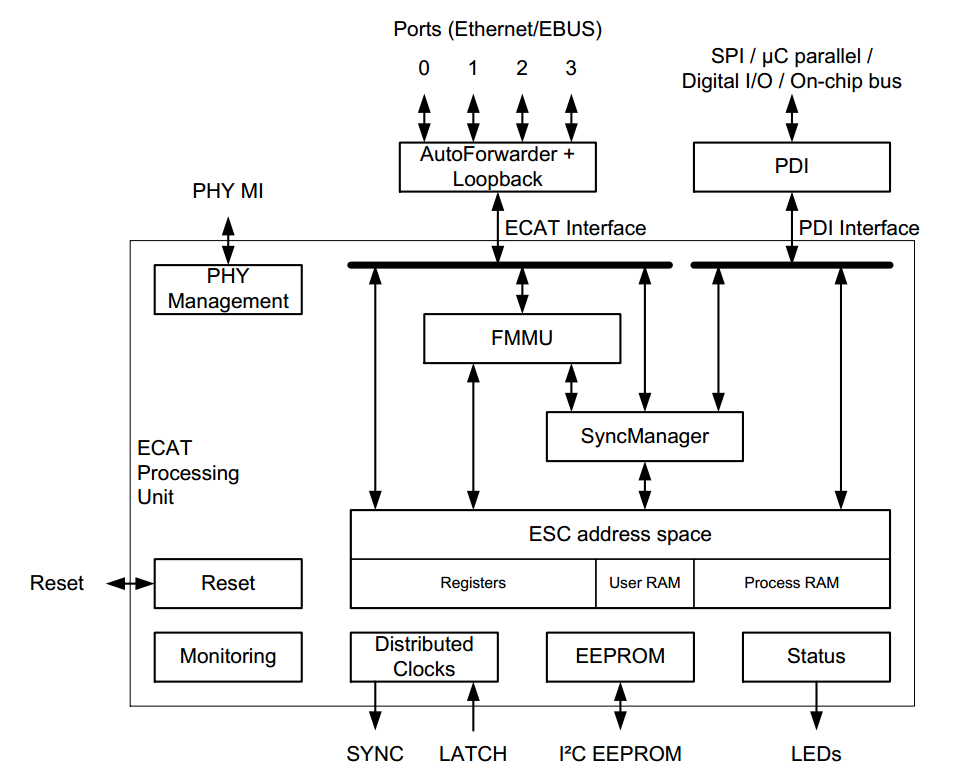


图 2-9 SM

SM保证了主站与从站本地应用操作内存数据的一致性、安全性，当数据发生变化时，SM会产生中断通知双方。主张可以对SM进行配置，SM管理了内部的一片缓存用于交换数据。访问SM管理的内存必须首先访问物理开始地址，否则访问会被拒绝。之后才能访问其它缓存位置，访问最后一个字节后，缓存的状态发生变化。

表 2-6 SM寄存器

|  |  |
| --- | --- |
| SM | 寄存器内偏移 |
| Physical Start Address | 0x0:0x1 |
| Length | 0x2:0x3 |
| Control Register | 0x4 |
| Status Register | 0x5 |
| Activate | 0x6 |
| PDI Control | 0x7 |

### 两种缓存模式

1. 3-buffer-mode,在任意时刻主站和从站本地均能访问缓存，消费者总能获得最后一次生产者生产的数据，生产者总能够对缓存进行更新。如果生产者的速度比消费者快，旧的数据会被新的数据覆盖。通常应用于周期性过程数据。
2. 1-buffer-mode，也称为邮箱模式，可以保证数据不会丢失。生产者产生了数据之后，只有消费者读取完成后，生产者才能继续生产数据。通常用于实现应用层协议。

### 基于 EtherCAT 的 CAN 应用协议（CoE）

使用 CoE 协议， EtherCAT 可提供与 CANopen®标准 EN 50325-4 相同的通信机制：包括对象字典、过程数据对象映像（PDO）以及服务数据对象（SDO），甚至相似的网络管理。因此，在已经实施了 CANopen 的设备中，仅需稍加变动即可轻松实现 EtherCAT，而且绝大部分的 CANopen 固件甚至都能得以重复利用。可以突破 8 字节的 PDO 限制，并可使用 EtherCAT增强的带宽资源实现整个对象字典的上传。另外，设备行规，如驱动器行规 CiA 402，也可以被重复应用于 EtherCAT。

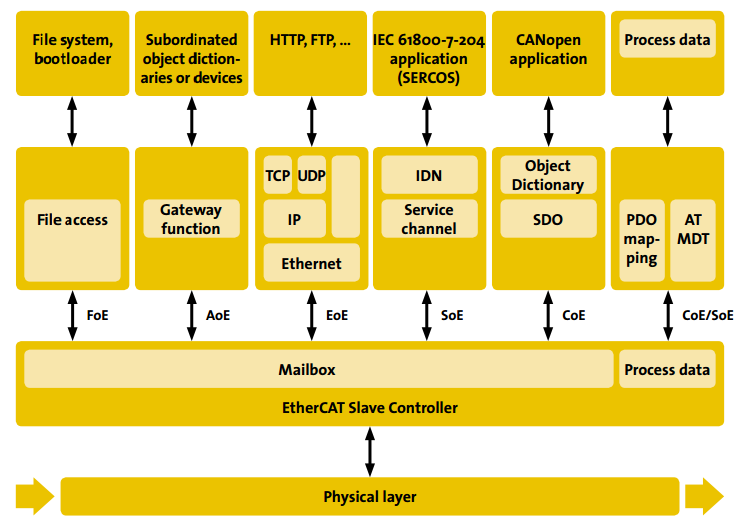


图 2-10 EtherCAT支持多种上层行规

### SDO的处理

假设初始化阶段SM0(0x800)配置为管理物理起始地址为0x1000,长度为0x40的内存区域，写模式。SM1(0x808)配置为管理物理起始地址为0x1100，长度为0x40的内存区域,读模式。主站侧处理流程如下：

1. FPWR 0x1000, 写入SDO请求;
2. FPRD 0x808,读取SM1的1-buffer-state，检查是否为written?如果不为written，重复2；
3. FPRD 0x1100，读取SDO响应。

从站侧处理流程如下：

1. SDO请求完整写入0x1000后，SM0产生中断；
2. 从站本地处理器，对SDO请求进行处理，将SDO响应写入0x1100；
3. SDO响应完整写入0x1100后，SM1的1-buffer-state变为written;



### PDO的处理

主站可通过LRW命令，周期性地与从站进行过程数据交换。适用于驱动器位置规划、IO模块输入输出等操作。

#### PDO配置

每个PDO包含一个或多个PDO条目，PDO条目可由index, subindex, bitlen定义。例如需将PDO(index=0x1600)配置为包含如下条目，主站按如下步骤进行：

1. SDO Download Index=1600, subindex = 0, value=2,写入PDO包含的条目个数；
2. SDO Download Index=1600, subindex = 1, value=0x60600010,写入条目1的定义；
3. SDO Download Index=1600, subindex = 2, value=0x607a0020,写入条目2的定义。

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Subindex | Bitlen | 描述 |
| 0x6040 | 0 | 16 | 控制字 |
| 0x607a | 0 | 32 | 目标位置 |

之后主站需对RXPDO分配进行配置,(index =0x1C12，对应SM2)

1. SDO Download Index=1C12,subindex = 0, value=1, 写入RXPDO包含的PDO个数；
2. SDO Download Index=1C12,subindex = 1, value=0x1600, 写入PDO的index；

类似的可将PDO(index=0x1601)配置为包含如下条目,将TXPDO（index=1C13）配置为仅包含0x1601。

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Subindex | Bitlen | 描述 |
| 0x6041 | 0 | 16 | 状态字 |
| 0x607a | 0 | 8 | 操作模式显示 |

#### SM配置

SM2对应RXPDO(index=0x1C12)，起始Physical Start Address保持不变(0x1200)，Length设置为6(RXPDO包含一个PDO,该PDO包含所有条目数据总长为6字节)。

SM3对应TXPDO(index=0x1C13)，起始Physical Start Address保持不变(0x1400)，Length设置为3(TXPDO包含一个PDO,该PDO包含所有条目数据总长为3字节)。

#### FMMU配置

FMMU0(0x600)配置为逻辑地址0->物理地址0x1200(SM2物理起始地址)，长度为6,写方式。

FMMU1(0x601)配置为逻辑地址3->物理地址0x1400(SM3物理起始地址)，长度为3，读方式。

#### LRW命令

所有配置完成后，主站周期性的发送LRW命令，进行周期性数据交换。



## 分布式时钟与同步

### 分布式时钟DC

精确同步对于同时动作的分布式过程而言尤为重要。例如，对于执行协同运动的多个  
伺服轴的应用便是如此。对于完全同步的通信，通信错误会立即影响其同步品质，而与其相比，分布式同步时钟对于通信系统的抖动具有很好的容错性。因此，EtherCAT 采用分布式时钟（DC）的方式同步节点。各个节点的时钟校准完全基于硬件。第一个具有分布时钟功能的从站设备的时间被周期性地发布给系统中的其他设备。采用这样的机制，其它从站时钟可以根据参考时钟精确地进行调整。整个系统的抖动远小于 1µs。

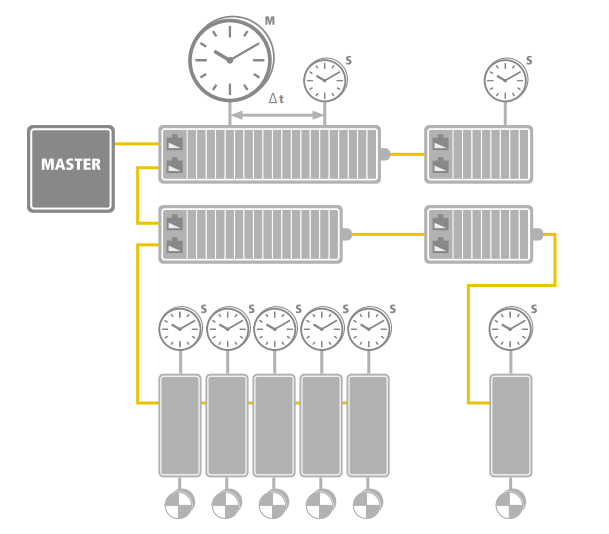


图 2-11 完全基于硬件并带有传输延迟补偿的同步

### 传输时延的测量、偏移量补偿

* 一般将第一个从站选定为参考时钟从站；
* T1为报文到达参考时钟本地时间，T4为报文返回参考时钟本地时间；
* T2为报文到达从站N本地时间，T3为报文返回从站N本地时间；
* 则报文从参考时钟传输到从站N的传输时延:



* 从站本地时钟与参考时钟的偏移量:





图 2-12示意图



### 漂移补偿

每个时钟由于石英晶振的差异，长时间运行会导致时钟差异。定期进行漂移补偿能够有效降低影响。

* 系统时间差定义为：



* 代表从站本地时钟比系统时钟快运行快，从站需要拨慢；
* 代表从站本地时钟比系统时钟快运行快，从站需要拨快；
* 寄存器0x092C:0x092F存储了从站的系统时间差；



### 同步方式

关 于 从 站 应 用 相 对 主 站 循 环 的 时 间 关 系 ， EtherCAT 定 义 了 3 种 主 要 的 同 步 模 式：

* Free Run (非同步): 从站的过程数据处理，由内部事件触发，与主站循环无关。
* SM-Synchronous: 从站的过程数据处理，由接收到携带过程数据的周期性数据帧时所产生的硬件中断触发。
* DC-Synchronous: 从站的过程数据处理，由基于分布时钟和系统时间的硬件中断触发。

#### Free Run

从站的过程数据处理，由**内部事件触发:**

* 不用定义周期性数据帧与从站本地应用程序之间的时间关系
* 各个“Free Run” 模式的从站之间，其时间偏移量不固定
* 适用于处理信号变化缓慢的 I/O 设备 (比如温度信号 …)

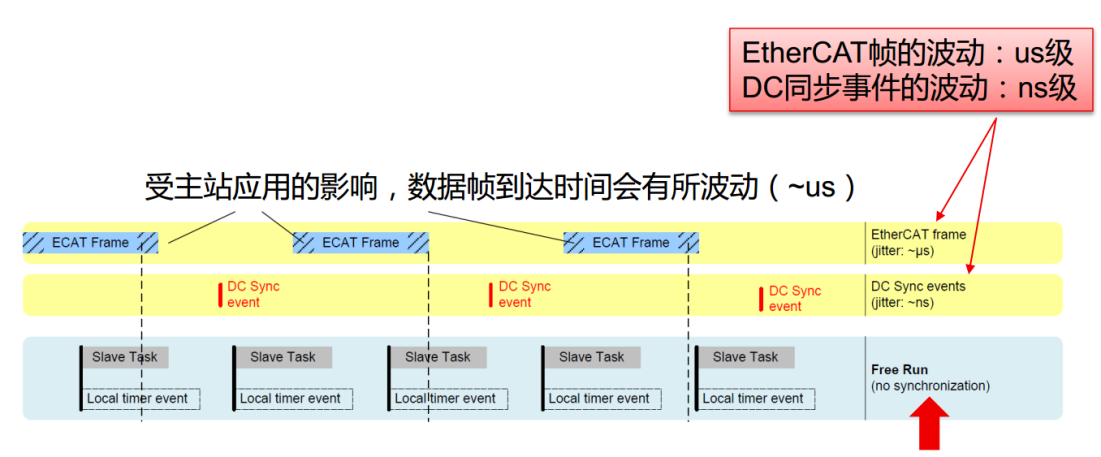


图 2-13 Free Run模式

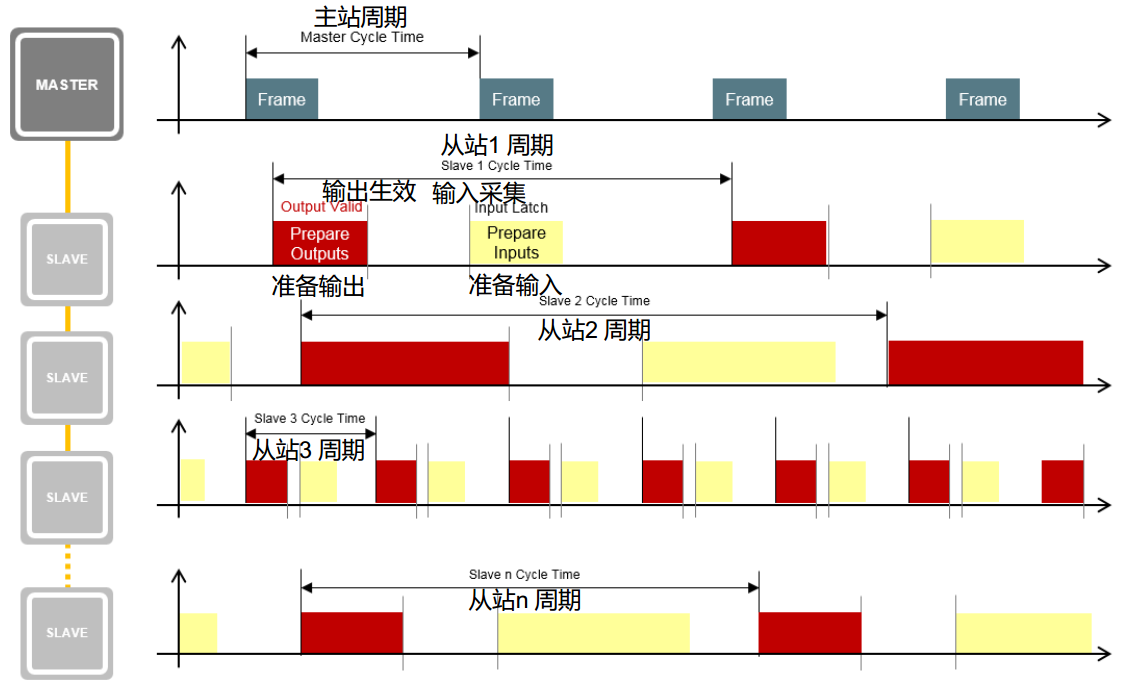


图 2-14 Free Run模式多个从站之间

#### DC同步

DC-Synchronous. 从站过程数据的处理，由从站中基于DC系统时间产生的硬件同步事件（SYNC Event）触发。

* 基于分布时钟的系统时间（DC System Time）, 在每个DC从站的内部产生硬件同步事件。
* 每个从站中的触发事件不受主站抖动和传播延时的影响。
* 适用于伺服驱动，以及IO模块。

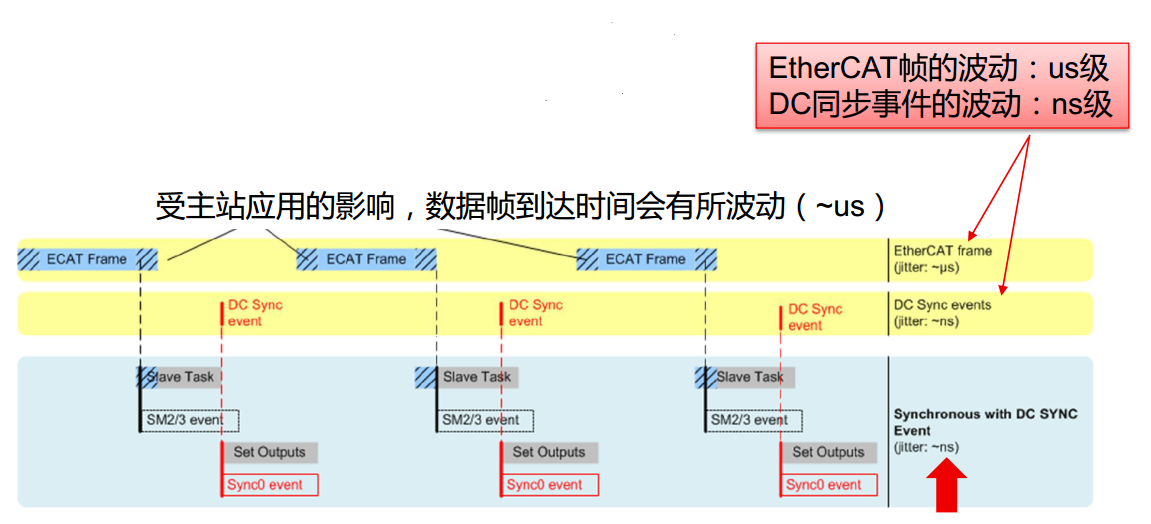


图 2-15 DC同步模式

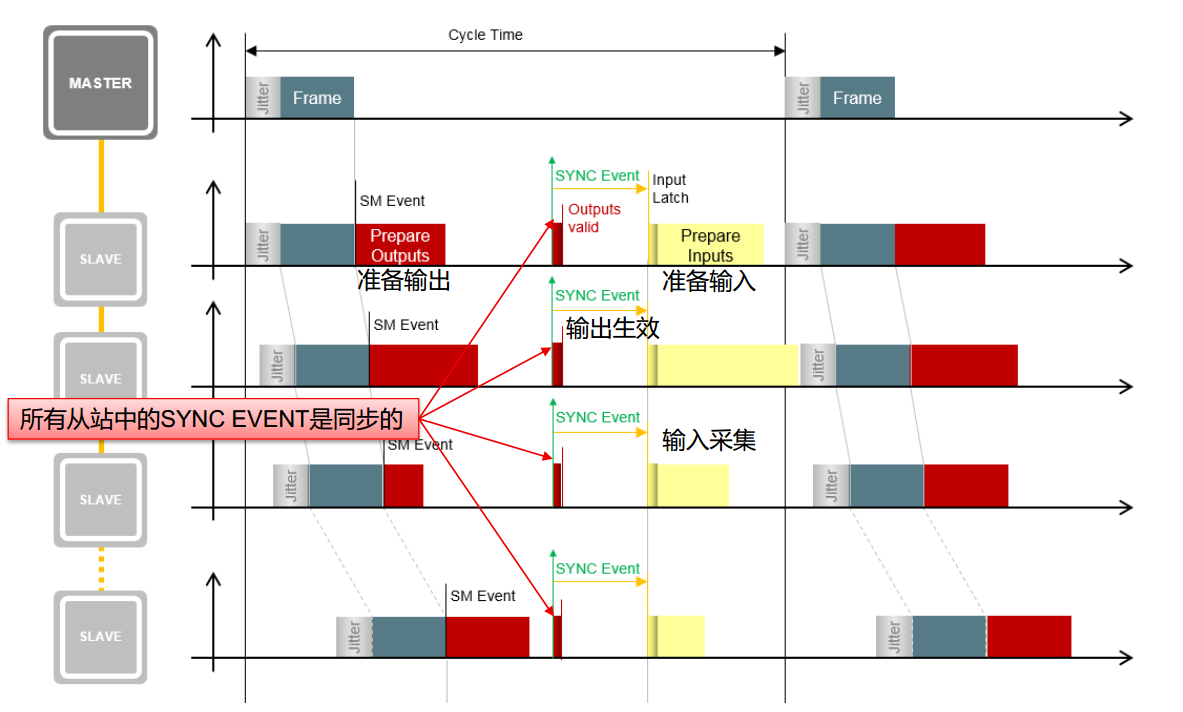


图 2-16 DC同步模式，多个从站之间

# DMC运动控制库

## ECM主站

EtherCAT Master Control IC/Starter Kit是讯成科技开发的主站开发工具集，安装Windows操作系统的PC可使用USB端口与ECM连接，弥补Windows操作系统非实时特点。ECM最多支持连接40个从站。使用前需连接跳线10(CONFIG0)与跳线26(GND)，将ECM配置为USB模式。三个指示灯（Mode Indicator）用于指示当前主站状态。

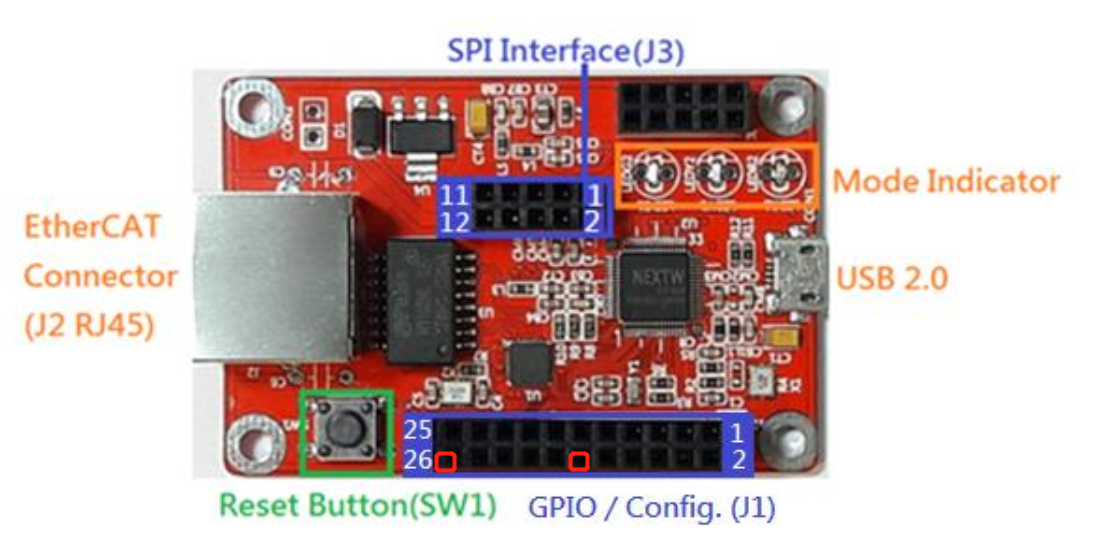


图 3-1 ECM

表 3-1 LED指示灯

|  |  |  |
| --- | --- | --- |
| LEDY1(黄色) | LEDG1(绿色) | 状态 |
| 亮 ON | 亮 ON | 未侦测到 EtherCAT 子站 或  子站未达预期阶段 |
| 暗 OFF | 暗 OFF | Init State 或 Pre-Operational State |
| 亮 ON | 暗 OFF | Safe-Operational State |
| 暗 OFF | 亮 ON | Operational State |

## 连接方式

Windows PC使用USB线连接至ECM,ECM通过双绞线连接各个从站。直接与ECM连接的从站的地址为1，顺序增加。连接示例图 3-2所示：

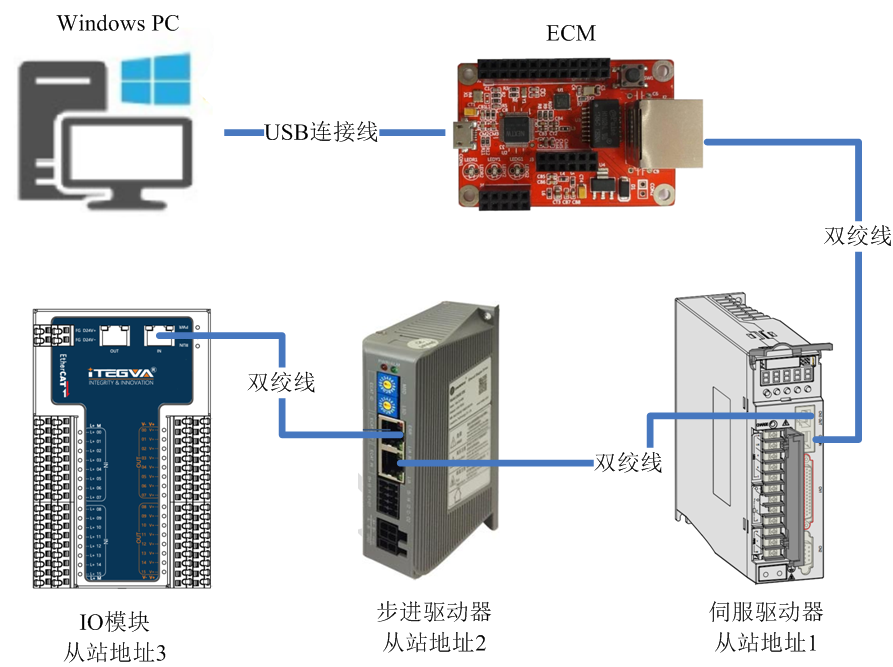


图 3-2 连接示意图

## 配置文件

Master.xml配置文件用于存储主站功能，子站信息，子站初始配置信息。程序初始化时加载配置信息，执行对应的初始化工作。配置文件大小写敏感，使用文本编辑器进行编辑。

主站配置在Mater下的各子节点中进行：

表 3-2 主站配置内容

|  |  |
| --- | --- |
| 子节点名称 | 描述 |
| LogLevel | 日志记录等级（日志文件存储在程序运行目录下,日志文件名：DMC.log）  0关闭 3错误 4 告警 6信息, 缺省值4 |
| HomeMethod | 回零方式， 缺省值 19 |
| HomeTimeout | 回零超时时间,单位s, 缺省值30 |
| ServoPosBias | 伺服位置达到检测允许误差范围，单位脉冲, 缺省值10 |

子站相关配置在Slave节点中进行：

表 3-3 Slave节点属性配置

|  |  |
| --- | --- |
| 属性名 | 描述 |
| desc | 从站描述 |
| index | 从站地址 |
| type | 从站类型：  Drive 伺服驱动器  Step 步进驱动器  IO IO模块 |

Slave节点下的Sdo节点可配置从站初始化阶段需进行的SDO Download操作。

表 3-4 Sdo节点属性配置

|  |  |
| --- | --- |
| 属性名 | 描述 |
| desc | SDO对象字典名称 |
| index | 对象索引，必须十六进制数以0x开头 |
| subindex | 对象子索引，必须十六进制数以0x开头 |
| size | 对象数据字节大小：十进制数 |
| value | 对象数据：十进制数 |

## 接口函数

接口函数仿照雷赛公司 DMC1000∕DMC1000B 卡函数库，接口主要差异如下，请注意对比。函数库的DC cycle固定为2000us。

1. 从站使用从站地址进行指定；
2. 运动初始速率为0，运动相关函数中起始速度StrVel参数无效；
3. 回原点函数d1000\_home\_move，参数指定高速以及低速；
4. 停止函数decel\_stop，增加参数tDec用以指定减速时间；
5. 删除设置指令脉冲的接口d1000\_set\_command\_pos；
6. 输入输出接口d1000\_in\_bit，d1000\_out\_bit需指定IO模块从站地址，所有位作为一个无符号整数进行参数传递。

### 初始化、关闭

表 3-5 初始化、关闭相关函数说明

|  |  |  |
| --- | --- | --- |
|  | 名称 | 功能 |
| 1 | d1000\_board\_init | 初始化 |
| 2 | d1000\_board\_close | 关闭 |

在调用其它接口函数之前，必须调用d1000\_board\_init函数进行初始化工作,该函数执行下列操作：

1. 初始化日志功能；
2. 加载Master.xml配置文件；
3. 打开ECM；
4. 设置DC同步，同步周期；
5. 所有伺服、步进驱动器操作模式设置为CSP；
6. 状态切换为Operational State;
7. 清除各从站当前告警；
8. 执行各从站初始化SDO Download配置；
9. 所有伺服、步进驱动器使能；

当结束程序时，必须调用d1000\_board\_close函数释放对应资源，该函数执行下列操作：

1. 状态切换为Init State;
2. 关闭ECM;
3. 关闭日志功能；
4. 释放资源

### 单轴位置

表述运动轨迹时可以用绝对坐标和相对坐标这两种模式，如**错误！未定义书签。**所示。这两种模式各有优点，如：在绝对坐标模式中用一系列坐标点定义一条曲线，如果要修改中间某点坐标时，不会影响后续点的坐标；在相对坐标模式中，用一系列坐标点定义一条曲线，用循环命令可以重复这条曲线轨迹多次。在 DMC函数库中距离或位置的单位为脉冲；速度单位为脉冲/秒；时间单位为秒。

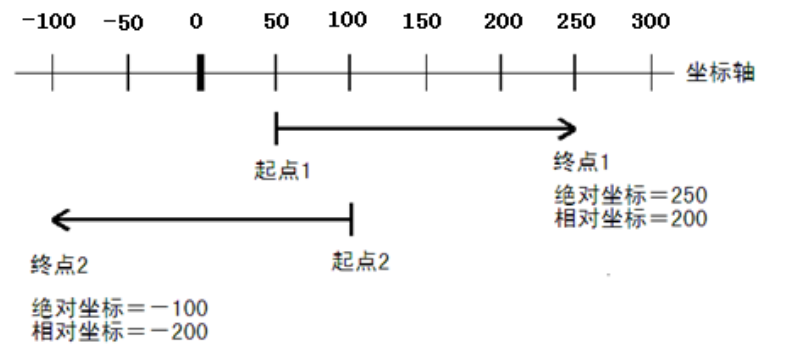


图 3-3 绝对坐标与相对坐标中轨迹终点的不同表达方式

#### 梯形速度曲线运动模式

通常位置控制采用这种速度控制模式。电机在运动一段指定距离时，其运动速度按梯形曲线变化，如图 3-4所示。**考虑实际应用场合，起始速度为0**。运动速度之所以要按梯形曲线变化，是因为：电机转子和被拖动的物体具有惯性，不可能在瞬间达到指定速度，因此应该有一定的加速过程。减速时亦是类似，否则电机会因为瞬间力矩不足而出现丢步、过冲（步进电机系统）或振荡（伺服电机系统）现象。

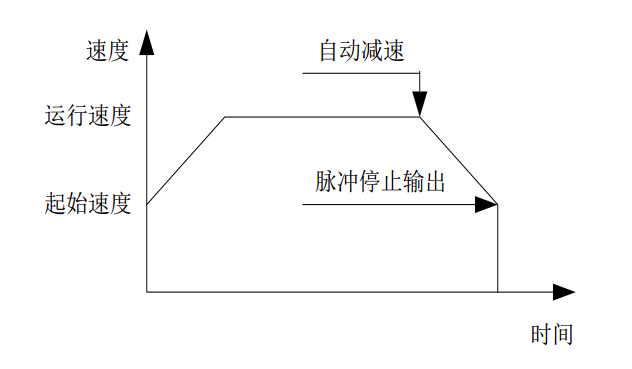


图 3-4 梯形速度曲线

表 3-6 梯形点位控制相关函数说明

|  |  |  |
| --- | --- | --- |
|  | 名称 | 功能 |
| 1 | d1000\_start\_t\_move | 以梯形速度曲线控制相对坐标的点位运动 |
| 2 | d1000\_start\_ta\_move | 以梯形速度曲线控制绝对坐标的点位运动 |

#### S形速度曲线运动模式

梯形速度曲线虽然实现起来简单，但它的加速度有突变，速度曲线不平滑，因而运动中有冲击现象，容易引起机器噪声和传动机构的磨损。在梯形速度曲线上（参见图 3-4），运动的不平滑主要表现在四个瞬间的速度转折及相对应的加速度突变，这四个瞬间分别是：起始时、升至最高速度时、从最高速度下降时和最后停止时。

若将加速度改为线性变化，则速度曲线相应将变得光滑，如图 3-5所示。升速和减速阶段均变得像 S 的形状。采用此种速度曲线，运动更平稳，且有助于缩短加速过程、降低运动装置的振动和噪声，同时还可以延长机械传动部分的寿命。

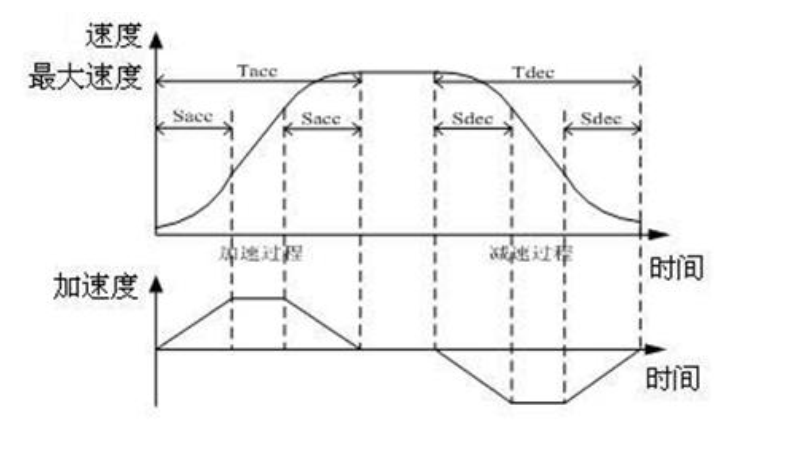


图 3-5 S 形速度曲线及其加速度曲线

表 3-7 S形点位控制相关函数说明

|  |  |  |
| --- | --- | --- |
|  | 名称 | 功能 |
| 1 | d1000\_start\_s\_move | 以 S 形速度曲线控制相对坐标的点位运动 |
| 2 | d1000\_start\_sa\_move | 以 S 形速度曲线控制绝对坐标的点位运动 |

#### 例程

DWORD ret;

if( ERR\_NOERR == (ret = d1000\_start\_t\_move(1, 10000, 0, 400000, 0.2)))

{

while(1)

{

ret = d1000\_check\_done(1);

if (MOVESTATE\_BUSY != ret)

break;

}

if (MOVESTATE\_STOP == ret)

printf("位置已到达.\n");

else

printf("运动失败.\n");

}

### 直线插补

可以指定任意轴进行直线插补， 插补运动的计算由专用函数完成， 用户只需调用相应的运动函数并设置运动速度、加速度、终点位置等参数，不需要介入插补过程的计算工作。

表 3-8 多轴直线插补相关函数说明

|  |  |  |
| --- | --- | --- |
|  | 名称 | 功能 |
| 1 | d1000\_start\_t\_line | 启动多轴相对坐标的直线插补（梯形速度曲线） |
| 2 | d1000\_start\_ta\_line | 启动多轴绝对坐标的直线插补（梯形速度曲线） |
| 3 | d1000\_start\_s\_line | 启动多轴相对坐标的直线插补（S形速度曲线） |
| 4 | d1000\_start\_sa\_line | 启动多轴绝对坐标的直线插补（S形速度曲线） |

例程如下:

DWORD ret;

short axisArray[] = {1, 2};

long distArray[] = {50000, 10000};

if (ERR\_NOERR == (ret = d1000\_start\_t\_line(2, axisArray, distArray,

100000, 0.2)))

{

while(1)

{

ret = d1000\_check\_done(1);

if (MOVESTATE\_BUSY != ret)

break;

}

if (MOVESTATE\_STOP == ret)

printf("直线插补完成.\n");

else

printf("直线插补失败.\n");

}

### Z轴拱门插补

沿Z轴做拱门运动，其它轴为直线插补运动。Z轴从起点处垂直提升hu高度后，开始直线插补，运动到最高绝对位置hh处后，开始下落，直线插补终止后垂直下落hd高度。如图 3-6所示。

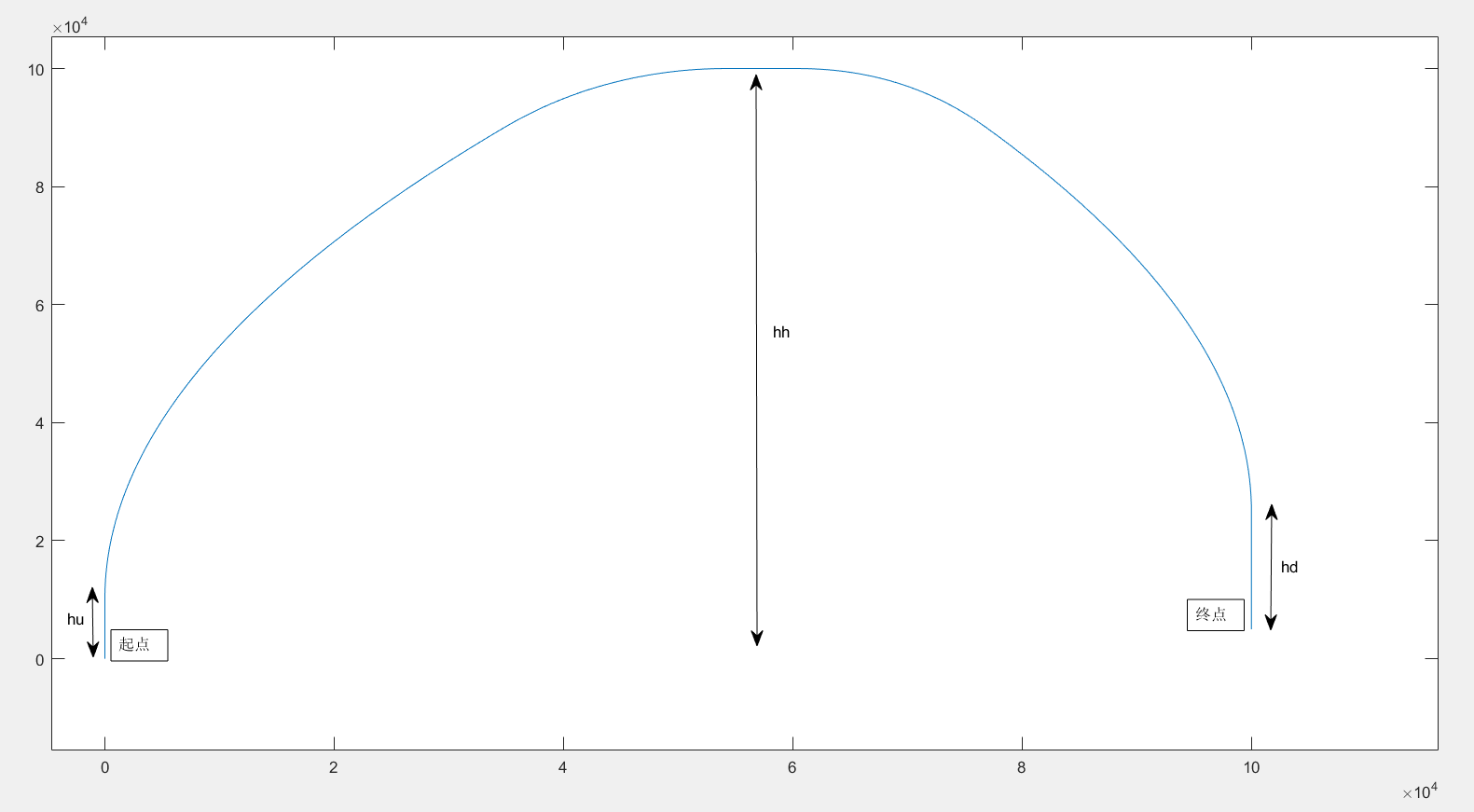


图 3-6 Z轴拱门运动示意图

表 3-9 Z轴拱门插补相关函数说明

|  |  |  |
| --- | --- | --- |
|  | 名称 | 功能 |
| 1 | d1000\_start\_t\_archl | 启动Z轴相对坐标拱门插补（梯形速度曲线） |
| 2 | d1000\_start\_ta\_archl | 启动Z轴绝对坐标拱门插补（梯形速度曲线） |

例程如下：

DWORD ret;

int hh = 100000;

int hu = 10000;

int hd = 20000;

short axisArray[2] = {2,1}; //2号轴为Z轴

long distArray[2] = {5000, 100000}; //位移

if (ERR\_NOERR == (ret=d1000\_start\_t\_archl(2, axisArray, distArray, 100000, 0.2, hh, hu, hd)))

{

while(1)

{

ret = d1000\_check\_done(1);

if (MOVESTATE\_BUSY != ret)

break;

}

if (MOVESTATE\_STOP == ret)

printf("Z轴拱门插补完成.\n");

else

printf("Z轴拱门插补失败.\n");

}

### 停止函数

当某个轴在单轴位置运动、直线插补过程中时，可急停或减速停止某个轴。d1000\_immediate\_stop将以固定时间停止，停止后关闭电机使能。d1000\_decel\_stop将T形减速停止对应轴。**当某个轴属于多轴插补运动中的某一个轴时，对任意轴调用停止函数，其它轴也会停止。**

表 3-10 停止相关函数说明

|  |  |  |
| --- | --- | --- |
|  | 名称 | 功能 |
| 1 | decel\_stop | 以梯形减速停止一个轴 |
| 2 | d1000\_immediate\_stop | 以梯形减速停止一个轴，并关闭电机使能 |

例程如下：

DWORD ret;

if (ERR\_NOERR == (ret = d1000\_immediate\_stop(1)))

{

while(1)

{

ret = d1000\_check\_done(1);

if (MOVESTATE\_BUSY != ret)

break;

}

if (MOVESTATE\_CMD\_STOP == ret)

printf("已停止.\n");

else

printf("停止失败.\n");

}

### 回原点运动

原点方法由Master.xml的HomeMethod节点指定。d1000\_home\_move的参数与回原点的相关对象对应关系如表 3-11所示。回原点超时返回MOVESTATE\_TIMEOUT，回原点成功会自动将命令位置重置为0，调用d1000\_get\_command\_pos返回0。

表 3-11 回原点相关参数

|  |  |  |  |
| --- | --- | --- | --- |
| Index | SubIndex | 字典对象名称 | 值 |
| 0x6099 | 1 | 搜寻原点高速 | HighVel |
| 0x6099 | 2 | 搜寻原点低速 | LowVel |
| 0x609A | 0 | 原点减速度 | HighVel/Tacc |

例程如下：

DWORD ret;

if (ERR\_NOERR == (ret = d1000\_home\_move(1, 50000, 5000, 0.2)))

{

while(true)

{

ret = d1000\_check\_done(1);

if (MOVESTATE\_BUSY != ret)

break;

}

if (MOVESTATE\_O\_STOP == ret)

printf("回原点成功.\n");

else

printf("回原点失败.\n");

}

### 指令脉冲计数

函数库内部维护了当前轴的指令脉冲绝对位置，回零成功后置0.可以通过d1000\_get\_command\_pos来读取该值，该值不提供接口进行修改。

例程如下：

long cmdpos = d1000\_get\_command\_pos(1);

printf("当前命令位置,%d.\n", cmdpos);

### IO控制

对于IO类型的子站，提供对应的接口函数进行输入输出。

|  |  |  |
| --- | --- | --- |
|  | 名称 | 功能 |
| 1 | d1000\_out\_bit | 写入输出状态值 |
| 2 | d1000\_in\_bit | 读取输入状态值 |

例程如下：

DWORD ret;

unsigned int bits;

if (ERR\_NOERR == (ret = d1000\_in\_bit(2, &bits)))

printf("当前输入=0x%x.\n", bits);

if (ERR\_NOERR == (ret = d1000\_out\_bit(2, 0xFF)))

printf("设置输出成功.\n");

# 示例程序

为了方便用户尽快熟悉 DMC运动控制库，编写了示例程序供参考。

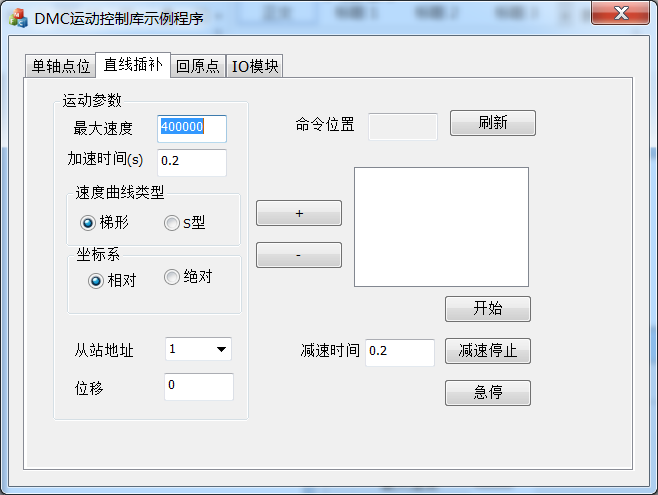
## 单轴点位

此选项卡中可对单轴运动相关函数进行测试。



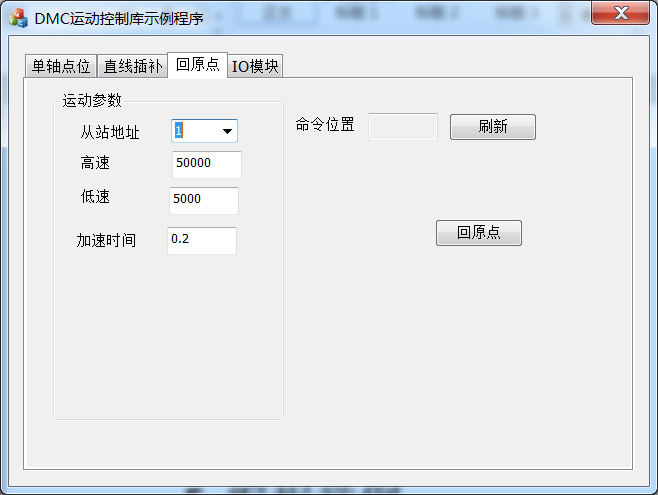
## 直线插补

此选项卡中可对直线插补相关函数进行测试。



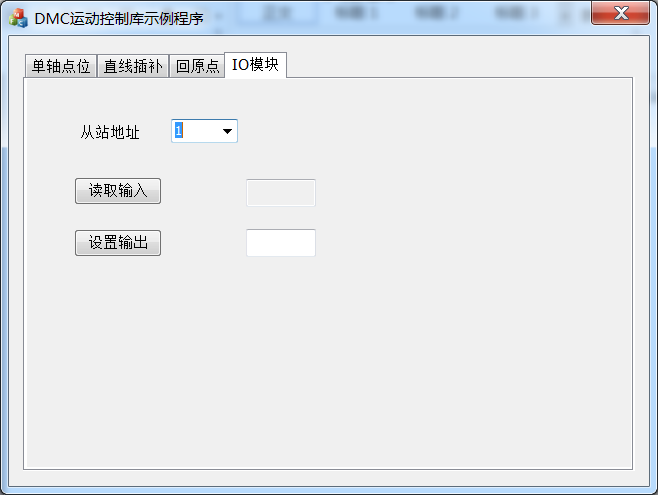
## 回原点

此选项卡中可对回原点相关函数进行测试。



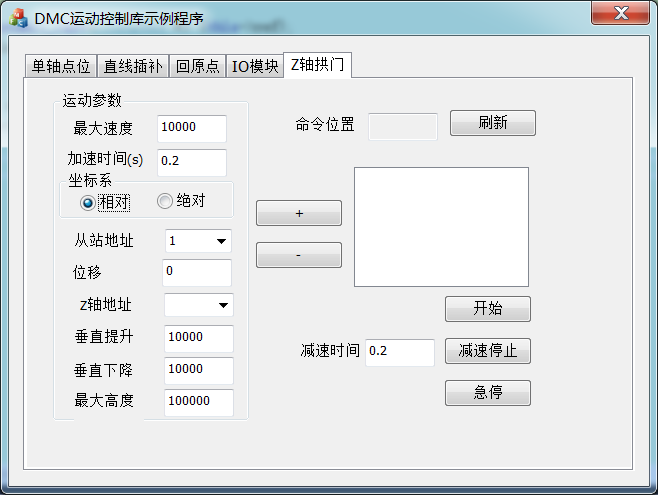
## IO模块

此选项卡中可对IO模块相关函数进行测试。



## Z轴拱门插补

此选项卡中可对Z轴拱门插补相关函数进行测试。

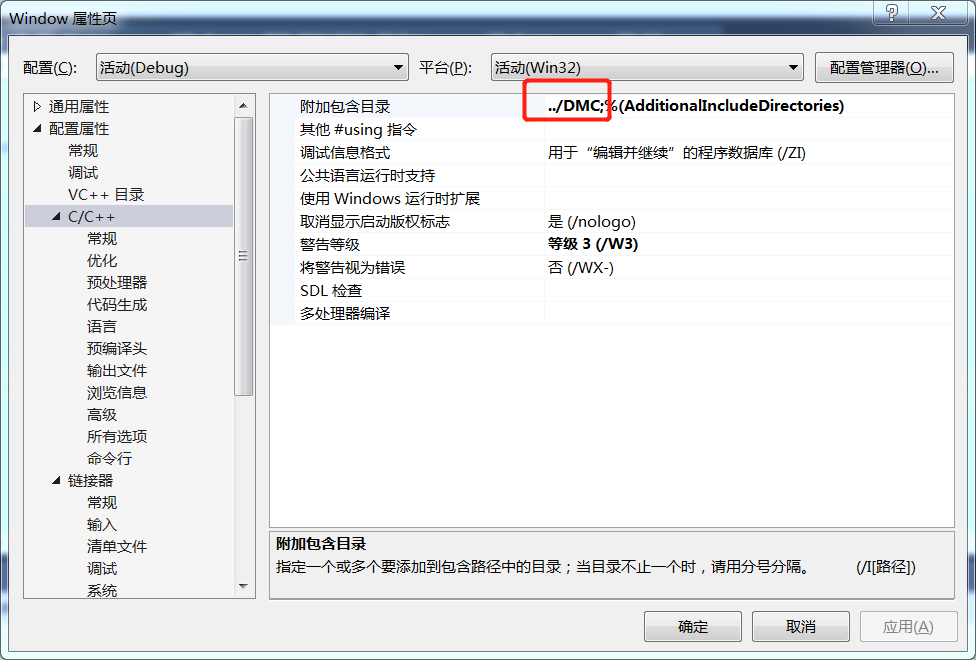


# 编程举例

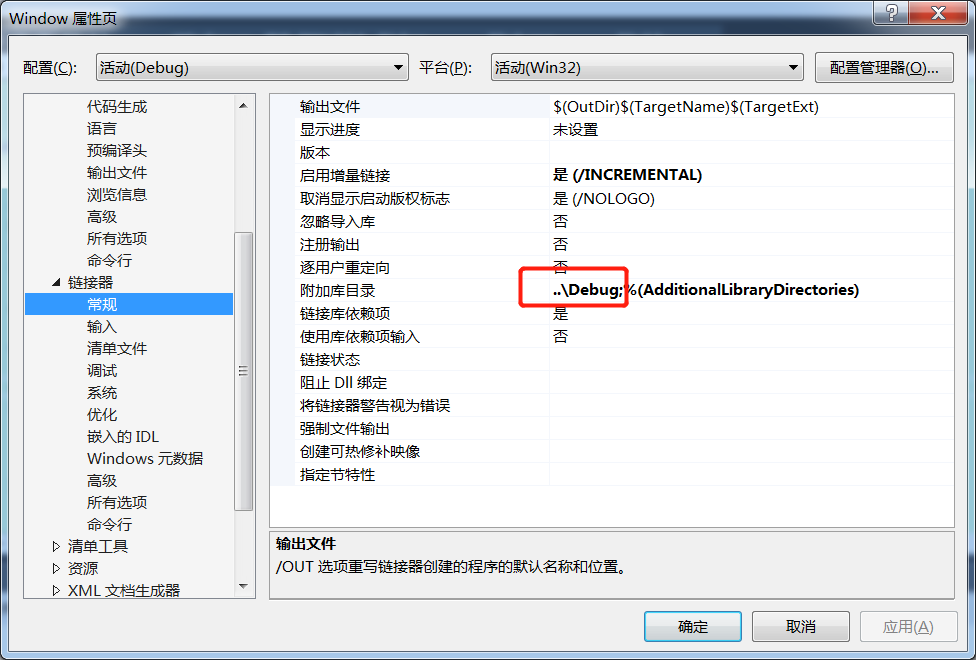
本章对开发运行环境配置进行简单介绍。

## 开发环境

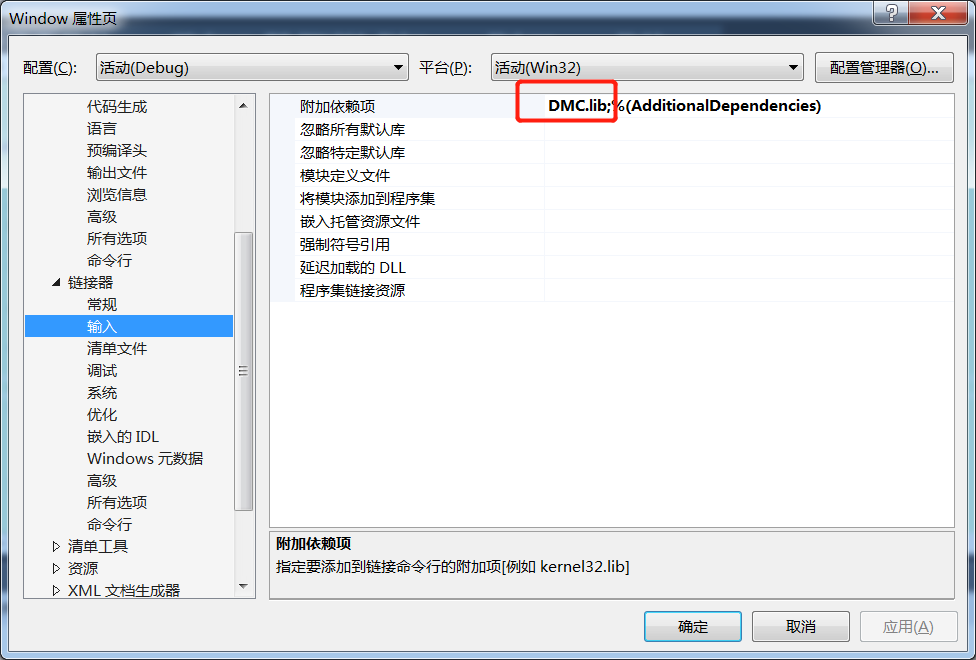
在附加包含目录中添加DMC.h所在目录。



在附加库目录中增加DMC.lib所在目录。



附加依赖项中增加DMC.lib



## 运行环境

程序运行目录下文件如表 5-1，注意Debug和Release环境下存在的差异。

表 5-1 运行目录下文件列表

|  |  |  |
| --- | --- | --- |
| Debug | Release | 描述 |
| DMC.dll | | DMC运动控制库 |
| NEXTWUSBLib.dll | | ECM库 |
| PocoXMLd.dll | PocoXML.dll | XML解析库 |
| PocoFoundationd.dll | PocoFoundation.dll | 基础库 |
| Master.xml | | 主站配置文件 |
| DMC.log | | 运行日志文件 |

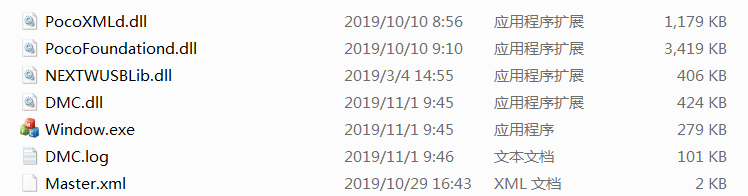


图 5-1 Debug环境

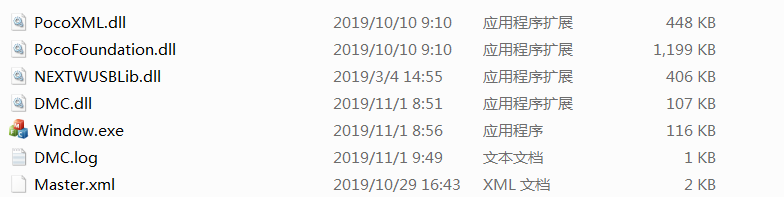


图 5-2 Release环境

DMC.log为程序运行过程记录日志，需关注是否产生下列记录：

* Fatal: 运行过程中产生严重错误
* Error：运行过程中产生错误
* Warning:运行过程中产生告警

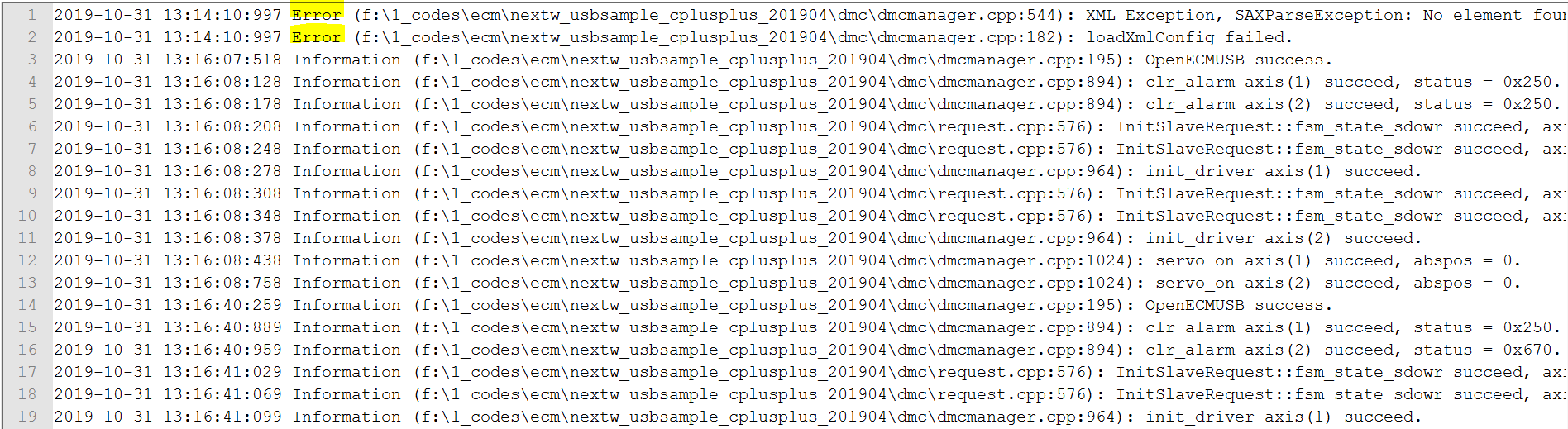


表 5-2 DMC.log文件示意