MIC7001机器智能控制器

用 户 手 册

长沙赛搏机器智能有限公司

V2020.02

1. 简介

MIC7001机器智能控制器是长沙赛搏机器智能有限公司针对工程机械等行业自主研发的国内第一款高防护分布式实时控制产品，不仅能够适应恶劣的工作环境如：露天安装、极端温度、高振动、高冲击、强电磁干扰等，而且具备强大的计算和存储能力，还内置远程通信功能，可帮助主机厂家大幅度提升工程机械设备的智能化和信息化水平。

MIC7001机器智能控制器采用 IEC61131-3 标准开发环境，支持 IEC61131-3 的梯形图（LD）、结构化文本（ST）等五种标准开发语言，与国内广泛使用的 IFM、EPEC 等控制器的开发软件保持兼容，软件移植方便快捷。（详细内容参见MIC7001快速入门指南）

MIC7001机器智能控制器具备丰富的通信接口，包括实时以太网、WIFI、窄带物联网、USB、CAN、RS485 等，可以方便的进行软件开发、监控维护、系统扩展等工作。

MIC7001机器智能控制器可通过实时以太网和 CAN 总线扩展 IO 模块，为大型主机搭建分布式控制系统，支持自制扩展 IO 模块和第三方兼容产品。

1.1. 功能特点

CPU 采用 32 位 ARM 芯片，主频 600~1000MHz。

具备完整的实时 Linux 操作系统，可通过大量现有开源软件拓展通信和管理功能。

具备大容量内部存储器，可实时记录主机运行信息，用于故障分析和工艺改进。

具备丰富的通信（以太网）端口，可通过兼容部件轻松进行系统功能扩展。

具有电源短路、反接保护、通信线短路、输出短路保护等。

具有高精度 PWM 电流反馈采集功能。

具有高精度 PWM 端口恒流输出功能。

具有输出端口线路短路/断路故障自诊断功能。

具有 DO 端口降压输出功能。

具有 CAN 终端电阻软件可配置功能。

具有 IP67 的防护等级。

密封良好的连接器：两个 8 针和两个 35 针的安普防水接插件。

尺寸：200mm×170mm×40mm。

1.2. 输入输出端口数量及复用关系

MIC7001 端口复用表								
复用类型	端口数量	AI	DI	PI	AO	DO	PWM	PWM-i
AI/DI	18	18	18	/	/	/	/	/
AI/DI/DO/PWM	24	24	24	/	/	24	24	/
DO/PWM-i	4	/	/	/	/	4	4	4
DO_L/PWM	4	/	/	/	/	4	4	/
DI/PI	4	/	4	4	/	/	/	/
AO/AI/DI	2	2	2	/	2	/	/	/
合计：	56	44	48	4	2	32	32	4

1.3. 性能指标

1.3.1. 计算性能

整数计算： 不大于 60ns

浮点数计算： 不大于 200ns

三角函数计算：不大于 20us

1.3.2. 端口性能

序号	类型	测试项目	测试条件	指标
1	数字量输入	最高输入电压		32V
		最大电平变化速率		1.471KHz
		最小逻辑高电平电压	独立 DI	9.5V
			AI 复用	3.8V
			PI 复用	3.9V
		最大逻辑低电平电压	独立 DI	9.0V
			AI 复用	3.5V
			PI 复用	4.0V
2	数字量输出	单路最大持续电流		2.5A
3	模拟量输入	输入范围		0～5V
		采样精度	<1V	2%
			5V>Ui>1V	<1%
		分辨率		12 位
		线性度		<1%
		重复度		<1%
4	模拟量输出	输出范围		0～5V
		分辨率		16 位
		输出精度		1%
		线性度		1%
		重复度		1%
5	脉冲量采集	频率范围		1Hz - 8KHz
		幅值范围		9V - 24V
		频率测量精度		1Hz 或 0.2%
		输入阻抗		>1MΩ
		持续输出最大电流		2.5A
		控制精度		16 位

6	脉宽调制输出	电流反馈精度		12 位
		电流反馈范围		0～1.25A
		电流反馈响应时间		< 110ms
		输出频率范围		40Hz～1KHz
		输出频率精度		±1Hz

2. 硬件端口

MIC7001机器智能控制器具有四个防水接线端子，分别标注为A（左下，8 针黑色）、B（右下，8 针白色）、C（左上，35 针黑色）、D（右上，35 针白色）。

各自包含的功能如下：

A 端口：包含一个 USB 端口、一个 RS485 端口和一个 CAN2.0 接口；

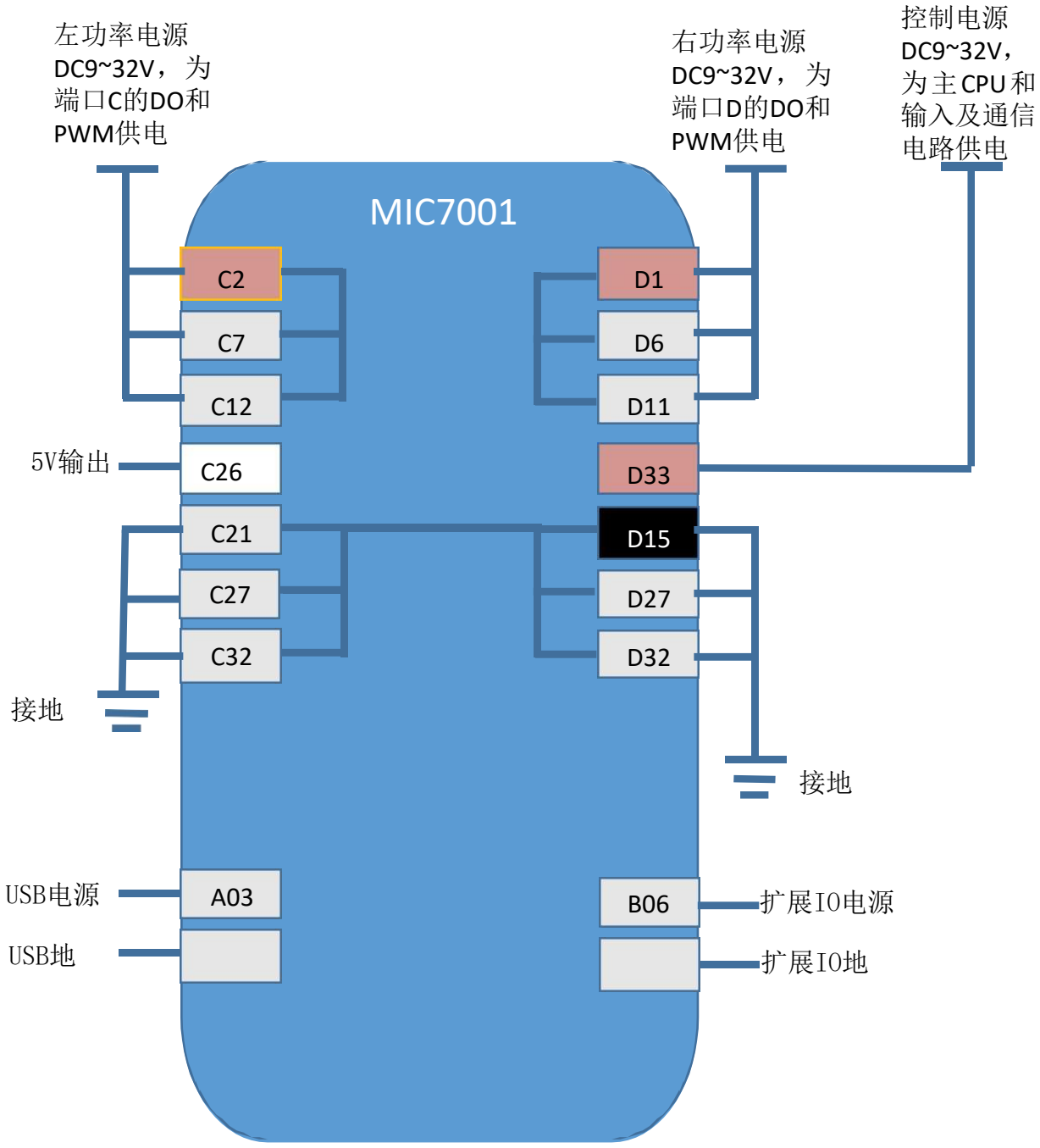
B 端口：包含一个 EtherCAT 主站接口和一个 CAN2.0 接口，专门用于连接扩展 IO 模块；

C 端口：主要为电源和输入、输出端口；

D 端口：主要为电源和输入、输出端口；

2.1. 电源端口

图2-1 MIC7001电源接线



2.2. 输入输出端口表

MIC7001 插座 C（黑35针，左上）									
管脚	AI	DI	PI	AO	DO	PWM	PWM_I	Power	备注
1：	AI-21	DI-21			DO-03	PWM-03			
2：								VCC_L	功率电源-左
3：	AI-22	DI-22			DO-04	PWM-04			
4:	AI-23	DI-23			DO-05	PWM-05			
5:	AI-24	DI-24			DO-06	PWM-06			
6:	AI-25	DI-25			DO-07	PWM-07			
7:								VCC_L	功率电源-左
8:	AI-26	DI-26			DO-08	PWM-08			
9:	AI-27	DI-27			DO-09	PWM-09			
10:	AI-28	DI-28			DO-10	PWM-10			
11:	AI-29	DI-29			DO-11	PWM-11			
12:								VCC_L	功率电源-左
13:	AI-20	DI-20			DO-02	PWM-02			
14:		DI-49	PI-01						
15:		DI-48	PI-00						
16:	AI-00	DI-00							支持0~20mA
17:	AI-02	DI-02							
18:	AI-08	DI-08							
19:	AI-05	DI-05							
20:	AI-07	DI-07							
21:								GND	
22:	AI-46	DI-46		AO-00					DI低有效
23:	AI-42	DI-42			DO-24	PWM-24			低有效输出
24:	AI-19	DI-19			DO-01	PWM-01			
25:	AI-18	DI-18			DO-00	PWM-00			
26:								5V输出	电流最大2A
27:								GND	
28:	AI-01	DI-01							
29:	AI-03	DI-03							
30:	AI-04	DI-04							
31:	AI-06	DI-06							
32:								GND	
33:					DO-28	PWM-28	PWMi-00		
34:					DO-29	PWM-29	PWMi-01		
35:	AI-43	DI-43			DO-25	PWM-25			低有效输出

MIC7001 插座 D（白35针，右上）									
管脚	AI	DI	PI	AO	DO	PWM	PWM-i	Power	备注
1：								VCC_R	功率电源-右
2：	AI-41	DI-41			DO-23	PWM-23			
3：	AI-40	DI-40			DO-22	PWM-22			
4:	AI-39	DI-39			DO-21	PWM-21			
5:	AI-38	DI-38			DO-20	PWM-20			
6:								VCC_R	功率电源-右
7:	AI-37	DI-37			DO-19	PWM-19			
8:	AI-36	DI-36			DO-18	PWM-18			
9:	AI-35	DI-35			DO-17	PWM-17			
10:	AI-34	DI-34			DO-16	PWM-16			
11:								VCC_R	功率电源-右
12:	AI-33	DI-33			DO-15	PWM-15			
13:	AI-45	DI-45			DO-27	PWM-27			低有效输出
14:	AI-47	DI-47		AO-01					DI低有效
15:								GND	
16:	AI-17	DI-17							
17:	AI-15	DI-15							
18:	AI-09	DI-09							
19:	AI-12	DI-12							
20:	AI-10	DI-10							支持0~20mA
21:		DI-51	PI-03						
22:		DI-50	PI-02						
23:	AI-32	DI-32			DO-14	PWM-14			
24:	AI-44	DI-44			DO-26	PWM-26			低有效输出
25:					DO-30	PWM-30	PWMi-02		
26:					DO-31	PWM-31	PWMi-03		

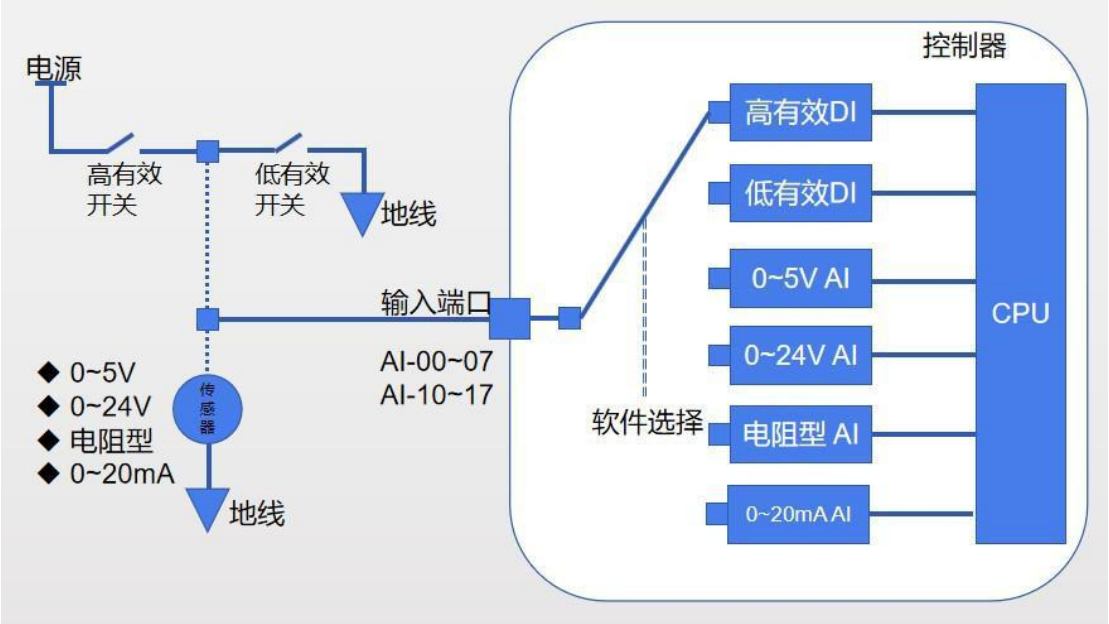
27:								GND	
28:	AI-16	DI-16							
29:	AI-14	DI-14							
30:	AI-13	DI-13							
31:	AI-11	DI-11							
32:								GND	
33:								VIN_C	控制电源
34:	AI-30	DI-30			DO-12	PWM-12			
35:	AI-31	DI-31			DO-13	PWM-13			

MIC7001 内部 AI：		
名称	编号	备注
Ai48_Pi02	AI48	备用
Vout5	AI49	传感器电源 5V
Vc	AI50	控制电源
VCC_L	AI51	功率电源左
VCC_R	AI52	功率电源右
PA1	AI53	PWM 电流反馈 1
PA2	AI54	PWM 电流反馈 2
PA3	AI55	PWM 电流反馈 3
PA4	AI56	PWM 电流反馈 4
CPU_T	AI57	CPU温度
G_X	AI58	安装角度 X
G_Y	AI59	安装角度 Y
G_Z	AI60	安装角度 Z
RES	AI61	备用
IO-cntr	AI62	备用
CDS-cntr	AI63	备用

2.3. 输入端口特性

MIC7001机器智能控制器的输入端口可以通过软件灵活配置其实际功能，可以作为模拟量输入（0~20mA 电流、0~5V 电压、0~36V 电压、0~20kΩ电阻）或开关量输入（低电平有效、高电平有效）。

图2-2 MIC7001输入端口类型



当作为AI输入时，控制器的采集值按照实际的物理意义设置，电压型的单位是毫伏， 电流型单位是微安，电阻型单位是欧姆。例如输入为2.3V，控制器采集到的值为2300； 如果输入电流13.45mA，控制器采集到的值为13450； 如果可变电阻为51.5K，控制器采集到的值为51500.

MIC7001 控制器具有 48 路AI 输入端口，其中 18 路可与 DI 复用，28 路可与 DI/DO/PWM复用，2 路可与 AO/DI 复用。

AI-00和AI-10具备上述的全部六种功能，AI-1~AI-7和AI-11~AI-17具备除0~20mA外的五种功能，AI-56和AI-47只有0~36V、电阻输入和低有效开关量输入三种功能， 其它 AI 端口具备除电阻和电流输入外的四种功能（参见表2-1）。

MIC7001 控制器具备 4 路 PI 输入端口，可对外部脉冲进行计数。

表2-1 输入端口工作模式

输入端口序号	配置类型					
	0	1	2	3	4	5
	0~5V电压	0~36V电压	0~20mA电流	0~36K电阻	高有效开关	低有效开关
0	√	√	√	√	√	√
1~7	√	√		√	√	√
8~9	√	√			√	√
10	√	√	√	√	√	√
11~17	√	√		√	√	√
18~45	√	√			√	√
46~47		√		√		√

2.4. 输出端口特性

MIC7001 控制器具有 2 路 AO 输出端口，可输出 0~13V 模拟电压，精度为 12 位，最大输出电流 20mA。设定值单位为毫伏，例如：若要输出7.45V电压，设定相应的AO寄存器值为7450。

MIC7001 控制器具有 28 路 DO/PWM 输出端口，单路最大输出电流 2.5A，具备短路保护功能。

MIC7001 控制器具有 4 路电流型 PWM 输出端口，输出电流范围 0~1000mA，精度位 12位。设定值单位为毫安，例如：若要输出745mA电流，可设定相应的PWMi寄存器值为745。根据不同型号电子比例阀的特性，实际输出电流可能会有偏差，若要精确输出电流，需要连接实际的电磁阀对输出设定值进行校准。

2.5. 通信端口

2.5.1. EtherCAT

接线：

B7: T+ 发 送 正
B8: T- 发 送 负
B5: R+ 接 收 正
B3: R- 接 收 负

EtherCAT 接线可按照普通以太网规范，收发端相同标号的线直连，其中 T+和 T-使用一对双绞线，R+和 R-使用一对双绞线。

MIC7001以太网接口的出厂默认IP地址为： 192.168.1.253

2.5.2. CAN

接线：

B2: CANOH
B4: CANOL
A1: CAN1H
A4: CAN1L

CAN 电缆使用双绞线，两端设备的 CANH 和 CANL 直连。

2.5.3. USB

接线：

A5: USBO+
A8: USBO-
A3: USB-5V
A2: USB-GND

应用：

MIC7001机器智能控制器的 USB 接口可支持 U 盘文件传输、摄像头视频采集、话筒音频采集、WIFI 等丰富的外围功能拓展。具体使用方法较灵活，若有该方面需求，可咨询赛搏机器智能的技术支持人员。

2.5.4. RS485

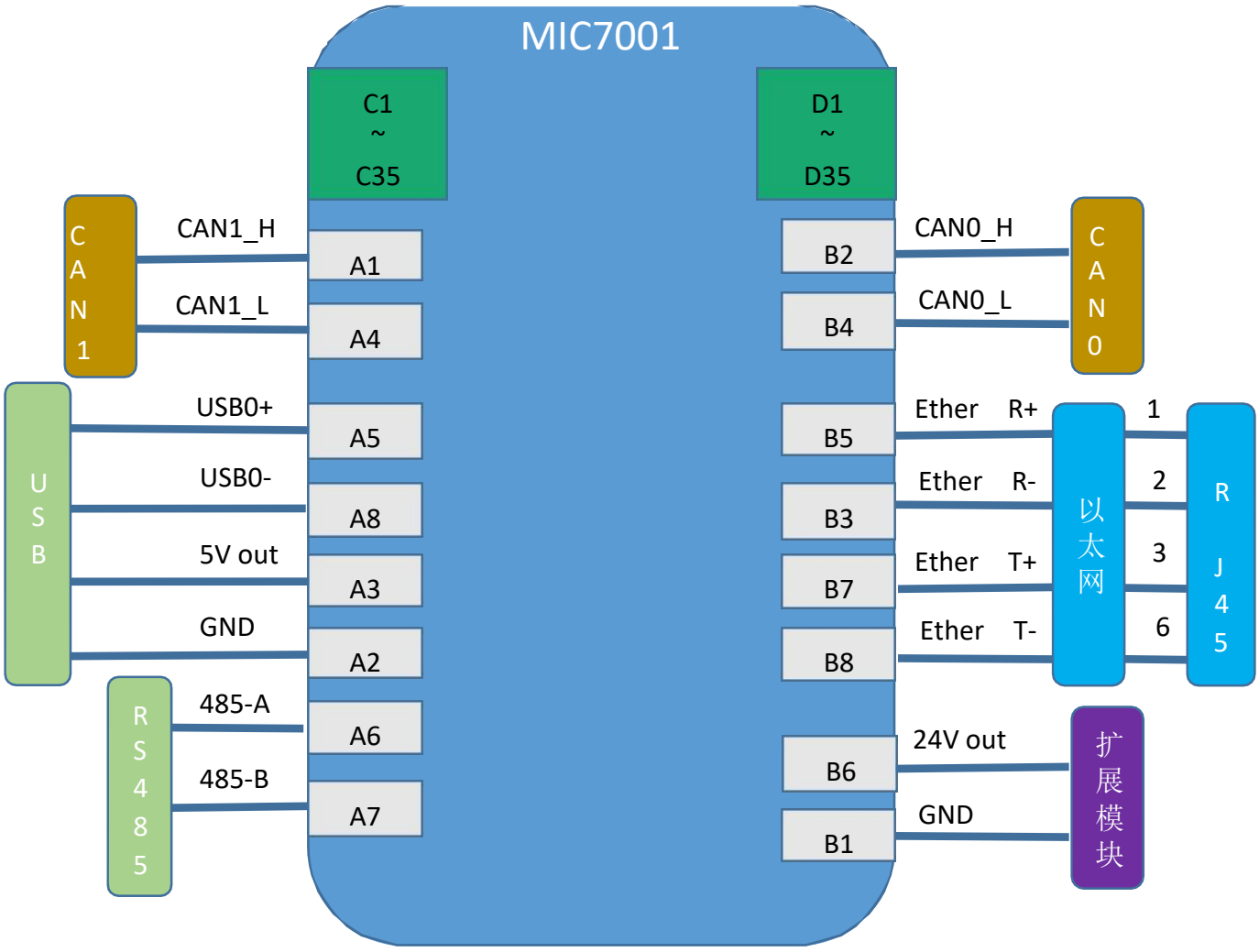
接线：

A6：485-A

A7：485-B

RS485电缆使用双绞线，两端设备的 RS485-A 和RS485-B 直连。

图2-3 MIC7001通信接口接线方法



3. 结构尺寸

MIC70xx 系列控制器安装尺寸：

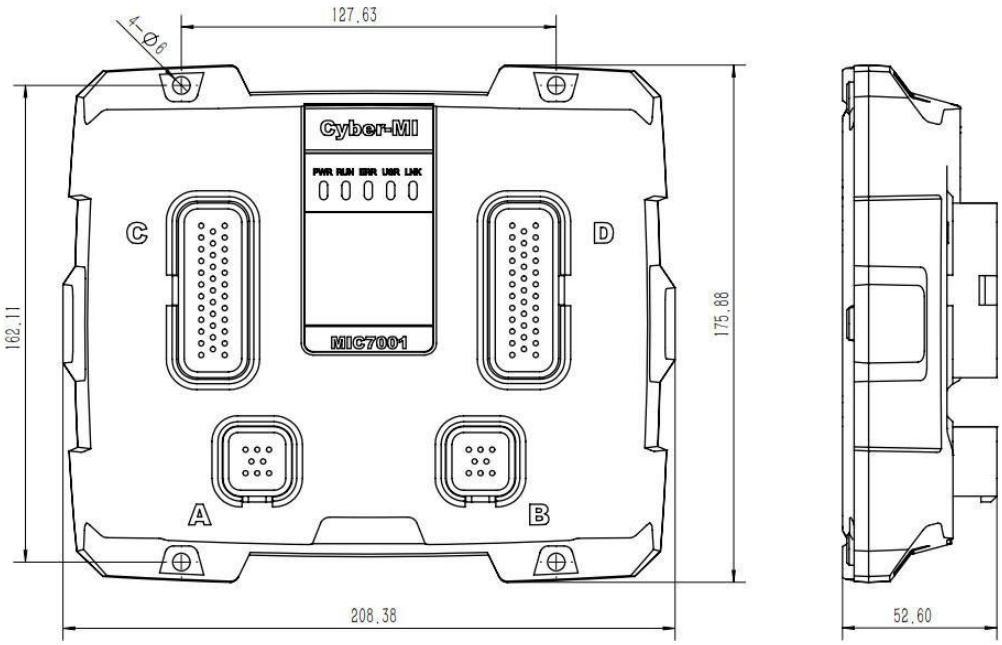


表3-1 需单独购买的配件清单

序号	名称	对接MIC7001插座位置	原厂型号	厂家
1	35针防水插头（黑）	插座C	776164-1	TE
2	35针防水插头（白）	插座D	776164-2	TE
3	8针防水插头（黑）	插座A	776286-1	TE
4	8针防水插头（白）	插座B	776286-2	TE
5	AMP插针		770520-1	TE
6	AMP防水塞		770678-1	TE

4. 系统开发指南

4.1. 开发环境配置

MIC7001支持C语言、python等编程语言直接开发应用程序，也支持通用的IEC61131-3开发环境，例如：IsaGRAF、CoDeSys、MultiProg等。

开发资料的更新地址为： http://github.com/mic7001/foruser

国内同步备份地址为： http://cyber-mi.com/files/controller

可根据网络情况选择任一网址下载。

下面以CoDeSys开发环境的安装和使用为例，其他语言的开发环境安装方法请咨询技术支持人员。

4.1.1. CoDeSys开发软件安装

① 下载地址

http://file.hicodesys.com:5000/sharing/SCOqL6kpe

密码： hicodesys.com

推荐使用版本：3.5.14.40.（文件路径： CODESYS官方Store/CODESYS Development System V3/CODESYS 3.5SP14Patch4，根据自己电脑windows版本选择其中的64位或者32位）

② 安装

执行下载的安装包进行开发环境安装，每一步均可使用默认设置。安装时间较长，请耐心等待。

4.1.2. 库文件安装与更新

① 网络同步库文件

Codesys系统安装完成后内置了很多常用的函数库，另外的一些不常用的函数库需要在使用的时候自动连接官网下载。

② 手动安装库文件

如果由于网络环境等问题无法连接codesys官网下载库文件，可通过我们的共享文件夹中的codesys_lib中的单独的库文件手动安装。

4.1.3. 设备描述文件安装

若使用CoDeSys作为开发环境，则需要安装控制器对应的设备描述文件，文件下载地址：http://www.cyber-mi.com/files/controller/MIC7001/配置文件/，其中包括：

（1）MIC7001设备描述文件： MIC7001-linux.devdesc.xml

（2）Io描述文件：IoDrvMIC7001.19.8.1.devdesc.xml

（3）扩展模块描述文件，参见：4.3.12. 扩展IO模块配置与应用

（4）第三方设备描述文件，参见：4.3.13. 第三方模块连接

具体安装方法，参见《MIC7001控制器快速入门指南》

4.2. 软件项目开发流程

4.2.1. 端口规划与配置

输入、输出端口分配， 参见《MIC7001控制器快速入门指南》

变量定义及绑定，参见《MIC7001控制器快速入门指南》

4.2.2. 程序编写与任务设置

软件功能设计：

建议根据实际功能设计应用程序的总体结构，比如发动机控制、液压系统控制、车辆操作控制等，每

种功能设置一个程序，作为控制器任务调度的基本单元。

每个程序可以调用其它程序、功能块和函数，具体方法可参考IEC61131-3编程规范。

软件任务设置：

每个程序是一个静态的代码模块，如果想要某个程序加入控制器的调度队列，需要创建一个任务，并将一个或多个程序加入该任务。

每个任务中的多个程序共享相同的优先级和运行周期。如果两个程序需要的优先级或运行周期不同，可创建不同的任务。

任务优先级设置：

MIC7001控制器的任务可以设置32个优先级（0~31，数字越小优先级越高），可根据实际情况为每个任务单独设置，一般跟安全性相关的任务需要设置较高的优先级，人机界面、远程通信等任务可以设置较低的优先级。

软件功能分配

4.2.3. 软件调试

软件编译及故障排除

软件开发完成后，可通过菜单

软件下载

软件调试

4.3. MIC7001特色功能

MIC7001多数特色功能需要赛搏控制器基本库的支持，请先安装 MIC7001_baselib_v0.75或以上版本。

4.3.1. 输入端口功能设置

六种输入模式说明

输入功能设置方法

4.3.2. 脉冲输入电压门槛值设置

依赖库：MIC7001_base v0.76或以上

使用方法：

```
piv:UINT := 8000;
Set_piv01(piv);   //将MIC7001的PI-00 和PI-01的输入门槛值设置为8V（8000mV）

piv:UINT := 12000;
Set_piv23(piv);   //将MIC7001的PI-02 和PI-03的输入门槛值设置为12V（12000mV）
```

本功能需要在函数运行后再次上电生效。

4.3.3. 参数永久保存

依赖库：MIC7001_base v0.76或以上

使用方法：

（1）变量定义：

① myRetain:FB_RETAIN;

② inited:BOOL := FALSE;

③ g_retain:ARRAY[0..Constants.RETAIN_NUM_MAX] OF DINT;//retain 变量, 建议放在全局变量列表 gvl中

（2）代码实例：

```
//步骤一：初始化默认值,如果读取文件失败，则使用默认值作为当前参数
```

```
IF NOT inited THEN  //上电执行一次，设置retain变量的缺省值
   inited := TRUE;

   //当前版本支持256个retain变量，数组序号0~255
   myRetain.default_val[0] := 123;
   myRetain.default_val[1] := 456;
   myRetain.default_val[2] := 789;
   // .....
   myRetain.default_val[87] := 321;
   myRetain.default_val[88] := 654;

   //下面的文件名最好跟本主机的机型相关，避免与其他工程重名
   myRetain(value := g_retain, filename :=  '/userdata/XXXX_retain_dint.txt');

   END_IF

//步骤二：执行功能块，处理参数的保存和读取

myRetain(value := g_retain);
```

//步骤三：在其它任务中直接使用 g_retain[] 作为retain变量,对于该数组中的数据的改动，上面的功能块会自动保存。

4.3.4. 数据录波及文件读写

依赖库：MIC7001_base v0.76或以上

使用方法：

（1）变量定义：

① data:ARRAY[0..MIC7001_base.Constants.FILE_DATA_MAX] OF REAL;

② data2file :MIC7001_base.FB_DATA2FILE;

（2）程序说明：

① 根据实际需要，把需要保存的数据依次填入数组：data[0] ~~~data[255],当前版本最多支持256个浮点数，如果需要更多，可分为两部分记录，或者联系赛搏公司修改库文件来提高上限。

② 调用data2file功能块，写入数据。参数如下：

1) xEnable: 输入参数，为True则开始记录，为False则停止记录。每次开始记录都会在指定的目录中自动创建一个新的数据文件，文件名根据当前时间生成。如果启动记录的时间是2020年3月5日18点23分45秒，生成的文件名为：2020.3.5.18.23.45.csv。该文件可拷贝到电脑，通过excel软件直接查看和编辑。

2) Data: 输入参数，长度为256的浮点数数组，需要随时写入需记录的数据。

3) Length: 输入参数，需要记录的数据个数，如果设定为26， 则记录Data[0]~data[25]。

4) Interval：输入参数，记录数据的间隔（毫秒），如果设定值小于本功能块所在的任务周期，则以任务周期为准。

5) Filedir: 输入参数，制定保存数据文件的路径，如果不设置，默认为：/userdata （用户可用的总空间约 2500 Mbytes）

6) Maxsize_m: 单个文件的最大体积（单位：Mbytes），如果单个文件的体积超过该设定值，本功能块会停止记录并报错，可将xEnable设置为false后再置为true，创建下一个文件并自动开始记录。

7) xBusy: 输出参数， 为true则说明数据正在记录。

8) xError：输出参数，为true则说明发生错误。

9) sErrorInfo: 输出参数，如果xError为true，则本参数显示错误原因

10) UdiFileSize： 输出参数，当前正在记录文件的大小（单位： 字节）

（3）程序实例：

```
//变量定义部分：
PROGRAM prg_data2file
VAR
	data:ARRAY[0..MIC7001_base.Constants.FILE_DATA_MAX] OF REAL;
	data2file :MIC7001_base.FB_DATA2FILE;

	start_save:BOOL := FALSE;
	i:UINT := 0;
	cntr :real := 1.23;
END_VAR

//程序部分：
data2file.xEnable := start_save;  //start_save 变量用于控制是否开始记录数据

//下面制造一些变化的模拟数据，真实项目中用实际的变量填充下面的data数组即可
cntr := cntr + 1;
data[0] := 123.456;
FOR i := 1 TO 255 DO
	data[i] := data[0]*(cntr+6);
END_FOR

data2file(
	data := data,  //实际数据
	length := 88, //数组中实际的数据个数，最多255
	interval := T#300MS, //数据保存的间隔，每隔300毫秒将上面data数组中的前88个数据保存一次，每次一行
	filedir := '/userdata/',  //保存数据文件的绝对路径
	maxsize_m := 1   //单个文件的最大体积，单位; Mbytes
 );
```

4.3.5. RS485通信软件开发

依赖库：

4.3.6. CAN总线通信软件开发

依赖库：CANBusAPI.package 需手动安装

CAN接口内置终端电阻设置：

```
Disable_120ohm_can0();  //禁用CAN0接口内置的终端电阻
Disable_120ohm_can1();  //禁用CAN1接口内置的终端电阻
Enable_120ohm_can0();  //启用CAN0接口内置的终端电阻
Enable_120ohm_can1();  //启用CAN0接口内置的终端电阻
```

上述功能需要在函数运行后再次上电生效。

通信功能开发：

该CAN接口主要分为四个部分，分别为驱动、接收器、处理器和发送器。每个CAN总线物理接口只需要创建一个驱动，但需要为每一个要接收的CAN id或id组单独创建一个接收器和一个处理器。发送器是一个独立的函数，可以随时发送任何id的CAN数据帧。

① 驱动 MIC7001有两个CAN接口，标号分别为0和1，可根据需要为每个CAN接口创建一个驱动。

1) 配置信息定义：

```
        g_busConfig0: CAN.DRIVER_CONFIG := (usiNetwork:= 0, uiBaudrate:= 250,
ctMessages:= 100);
        g_busConfig1: CAN.DRIVER_CONFIG := (usiNetwork:= 1, uiBaudrate:= 250,
ctMessages:= 100);
```

2) 在程序中调用：

```
        driver0(DriverConfig:= g_busConfig0, eError=> );
        driver1(DriverConfig:= g_busConfig1, eError=> );
```

3) 后面的接收器、处理器和发送器可以直接使用 driver0 和driver1 操作两个CAN接口。

② 先定义处理器

1) 创建处理器功能块：

```
    //定义部分， 以创建一个接收 16#0CF00301 的扩展帧数据为例
    FUNCTION_BLOCK MsgProcessor_F003 IMPLEMENTS CAN.IMessageProcessor
    VAR_IN_OUT
      CANdriver    : CAN.CANBus_29bit;
    END_VAR
    VAR_OUTPUT
    END_VAR
    VAR
      _pCANdriver : POINTER TO CAN.CANBus_29bit;
    END_VAR

    //程序实体：
    _pCANdriver := ADR(CANdriver);

    //为功能块创建一个名为ProcessMessage的Method
    //定义部分：
    METHOD ProcessMessage
    VAR_IN_OUT
      (* The messages received by the receiver *)
      Message      : CAN.RxMessage;
    END_VAR
    VAR
    END_VAR
```

//程序实体，这里是应用开发工程师主要写的部分，如果上面创建的接收器收到符合要求的CAN数据帧，会自动调用本 Method，收到的数据存在结构体 Message中，开发工程师可以根据需要访问数据并处理。

```
    IF _pCANdriver <> 0 THEN
      //在这里处理接收到的数据
      Recv_id := Message.udiCanId;
      Recv_data0 := Message.abyData[0];
      Recv_data7 := Message.abyData[7];
    END_IF
```

③ 再创建接收器并绑定接收器

1) 定义接收参数：

```
  (* 为接收ID设置参数 *)
  maskConfig0: CAN.RECEIVER_MASK := (
        dwIdValue:= 16#0CF00301 ,     //要接收的ID
        dwIdMask:= 16#FFFFFFFF,      //要接收的ID过滤器，为1的位必须一致才接收
```

```
        xRTRValue:= FALSE,    //是否远程帧
        xRTRMask:= FALSE,     //是否启用远程帧过滤器
        x29BitIdValue:= TRUE,  //是否扩展帧
        x29BitIdMask:= TRUE,   //是否启用扩展帧过滤器
        xTransmitValue:= FALSE, //是否用本驱动发送
        xTransmitMask:= FALSE,  //是否启用发送过滤器
        xAlwaysNewest:= TRUE    //为True则只保留最新一帧， 为False则启用接收缓存
    );
(* 定义接收器句柄，用于关联处理器 *)
hMaskReceiver0       : CAN.CAA.HANDLE := CAN.CAA.gc_hINVALID;
msgProcessorF003 : MsgProcessor_F003;  //生成前面接收器功能块的一个实例
```

2) 程序实体：

```
IF hMaskReceiver0 = CAN.CAA.gc_hINVALID THEN
      hMaskReceiver0 := driver0.GetMaskReceiver(Mask:= maskConfig0, eError=> );
END_IF

//接收数据帧，如果CAN0接口收到ID为16#0CF00301的扩展帧，会将内容传递给msgProcessorF003 .
driver0.ReceiveMessage(
    hReceiver:= hMaskReceiver0,  //关联接收器
    itfMsgProcessor:= msgProcessorF003,  //关联处理器
    tTimeLimit:= g_tTimeLimit,  //超时时间
    eError=>
);

/// 运行处理器
msgProcessorF003(CANdriver:= driver0);
```

④ 发送器

```
//以发送一个ID为16#3A的扩展帧为例
//变量定义：
message0 :CAN.MESSAGE;     //定义一个CAN数据帧的实体

//程序实体：
//填充数据帧
message0.udiCanID := 16#3A;  //填充ID
message0.usiDataLength := 8; //填充数据长度
message0.xIs29BitMessage := TRUE; //指定为扩展帧
message0.abyData[0] := 12;  //填充数据0
message0.abyData[1] := 21;  //填充数据1
//。。。。。。
message0.abyData[7] := 34;  //填充数据7

//发送数据帧

driver0.SendMessage(message0, eError=>);
```

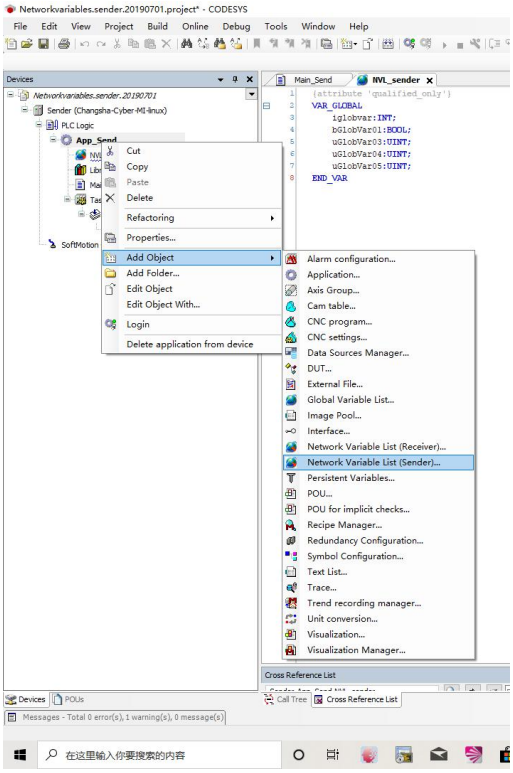
4.3.7. 主机设备间以太网通信

MIC7001控制器内置一个100M工业以太网接口（B口），可以基于UDP或TCP方式与其它厂家的以太网设备自由通信。如果对方设备支持，可以使用更方便的网络变量方式通信。

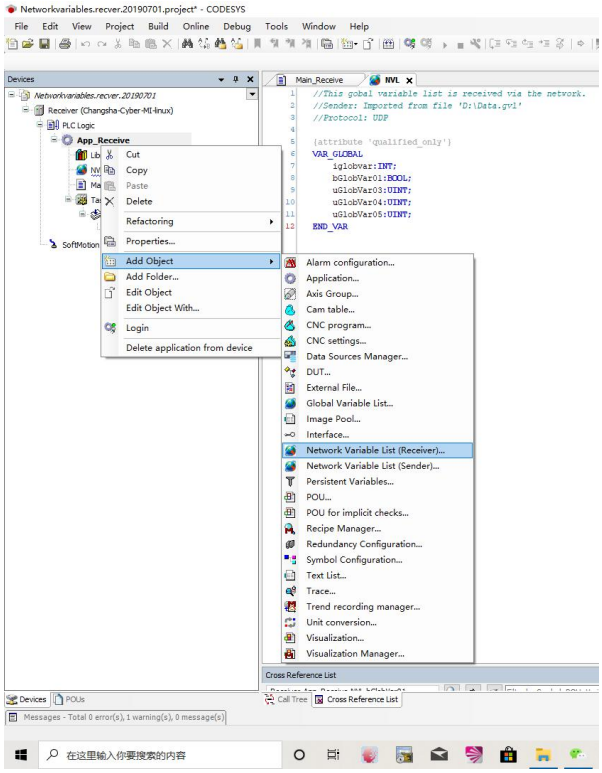
（1）自定义协议通信（TCP或UDP）

用户可通过MIC7001控制器提供的函数库自定义通信协议，与其它以太网设备自由通信。

（2）网络变量通信



发动端定义和设置



接收端定义和设置

在发送端和接收端分别定义完变量后，系统会自动对相同变量名的变量进行同步，开发人员无需关注具体的通信细节。

4.3.8. 控制器信息获取

依赖库：MIC7001_base v0.76或以上

① 获取控制器序列号

```
mysn:string;
ret:bool := false;

ret := read_sn(adr(mysn));
if ret then
  //can use sn
end_if
```

② 获取控制器sim卡号（全球唯一，可作为控制器身份标识）

```
myiccid:string;
ret:bool := false;

ret := read_iccid(adr(myiccid));
if ret then
  //can use myiccid
end_if
```

③ 获取控制器当前的基站代码

```
mylac:string;  //控制器当前所连基站的小区号
myci:string;   //基站号
ret:bool := false;

ret := read_location(adr(mylac), adr(myci));
if ret then
```

```
        // can use mylac & myci
    end_if
```

④ 获取控制器角度

定义一个real类型变量my_x，绑定IO表中的 AI58_G-x,即可直接读取控制器的X轴角度；
定义一个real类型变量my_y，绑定IO表中的 AI59_G-y,即可直接读取控制器的Y轴角度；
定义一个real类型变量my_z，绑定IO表中的 AI60_G-z,即可直接读取控制器的Z轴角度；

⑤ 获取控制器内部温度

定义一个real类型变量my_temp，绑定IO表中的 AI57_CPU-T,即可直接读取控制器的内部温度；

⑥ 获取控制器当前时间

依赖库： CAA DTUtil Extern

变量定义：

```
get_dt:dtu.GetDateAndTime; //获取系统时间
g_uiYear:UINT := 1970;
g_uiMonth:UINT := 1;
g_uiDate:UINT := 1;
g_uiHour:UINT := 0;
g_uiMinute:UINT := 0;
g_uiSecond:UINT := 0;
```

程序示例：

```
get_dt(xExecute := TRUE);
IF get_dt.xDone THEN
    tempdt := DT_TO_DWORD(get_dt.dtDateandtime);
    tempdt := tempdt + 3600*8;  //增加8个小时，转换成东八区的北京时间
    dtu.DTSplit(DWORD_TO_DT(tempdt),    //分解成 年月日时分秒
          ADR(g_uiYear),
          ADR(g_uiMonth),
          ADR(g_uiDate),
          ADR(g_uiHour),
          ADR(g_uiMinute),
          ADR(g_uiSecond));
End_if
```

4.3.9. 数据上传云端

① 通过UDP客户端

依赖库：

② 通过TCP客户端

③ 通过MQTT客户端

4.3.10. 基于U盘的数据导入导出

U盘升级

U盘导入数据

U盘导出数据

4.3.11. 远程升级与调试

环境准备

（1）控制器端：把控制器序列号通报赛搏技术支持人员，远程启用该控制器的vpn连接。

（2）调试电脑端：安装openvpn软件，通过赛搏提供的配置文件，启动openvpn软件。

远程下载与调试

与本地以太网或WIFI连接同样的方法下载与调试，由于GPRS网络的信号问题，如果速度较慢，需耐心等待。

4.3.12. 扩展IO模块配置与应用

设备描述文件安装

文件位置：http://www.cyber-mi.com/files/controller/MIR3502/MIR3502.xml

在CoDeSys开发环境中安装该设备描述文件。

网络连接

通过四对双绞线连接MIC7001控制器的B口（8针白口）与MIR3502的A口（8针黑口），按照相同的针号直连。其中pin-1和pin-6为一对， pin-2和pin-4为一对， pin-3和pin-5为一对， pin-7和pin-8为一对。如果需要安装多个MIR3502模块，可使用同样的电缆连接第一个MIR3502的B口和第二个MIR3502的A口，以此类推，可通过第二个MIR3502的B口接下一个MIR3502模块或者第三方EtherCAT设备。

设备扫描

变量绑定

4.3.13. 第三方模块连接

扩展IO模块

伺服驱动器

4.3.14. 运动控制系统开发简介

结构建模

运动控制软件开发

附录1 库函数的使用说明书

（1）Disable_120ohm_can0（）

功能描述：

关闭CAN0的终端电阻，当使用CAN0端口连接到CAN总线上，且CAN总线上除了控制器CAN0已经有两个节点有终端电阻的情况下，需要关闭控制器的终端电阻，以免影响CAN的通信功能。

接口定义：

Disable_120ohm_can0 (FUN)

FUNCTION Disable_120ohm_can0 : BOOL

InOut:

Scope	Name	Type
Return	Disable_120ohm_can0	BOOL

函数框图：

ST调用语句示例：

```
disable_120ohm_can0();
```

调用说明：

(1)该函数使用时，不用定义变量；
(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（2）Disable_120ohm_can1（）

功能描述：

关闭CAN1的终端电阻，当使用CAN1端口连接到CAN总线上，且CAN总线上除了控制器CAN1已经有两个节点有终端电阻的情况下，需要关闭控制器的终端电阻，以免影响CAN的通信功能。

接口定义：

Disable_120ohm_can1 (FUN)

FUNCTION Disable_120ohm_can1 : BOOL

InOut:

Scope	Name	Type
Return	Disable_120ohm_can1	BOOL

函数框图：

ST调用语句示例：

```
disable_120ohm_can1();
```

调用说明：

(1)该函数使用时，不用定义变量；
(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（3）Enable_120ohm_can0（）

功能描述：

使能CAN0的终端电阻，当使用CAN0端口连接到CAN总线上，且CAN总线上除了控制器CAN0，有且只有一个节点有终端电阻的情况下，需要打开控制器的终端电阻，才能确保CAN正常通信。

接口定义：

Enable_120ohm_can0 (FUN)

FUNCTION Enable_120ohm_can0 : BOOL

InOut:

Scope	Name	Type
Return	Enable_120ohm_can0	BOOL

函数框图：

ST调用语句示例：

Enable_120ohm_can0();

调用说明：

(1)该函数使用时，不用定义变量；
(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（4）Enable_120ohm_can1（）

功能描述：

使能CAN1的终端电阻，当使用CAN1端口连接到CAN总线上，且CAN总线上除了控制器CAN1，有且只有一个节点有终端电阻的情况下，需要打开控制器的终端电阻，才能确保CAN正常通信。

接口定义：

Enable_120ohm_can1 (FUN)

FUNCTION Enable_120ohm_can1 : BOOL

InOut:

Scope	Name	Type
Return	Enable_120ohm_can1	BOOL

函数框图：

ST调用语句示例：

Enable_120ohm_can1();

调用说明：

(1)该函数使用时，不用定义变量；
(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（5）read_iccid（）

功能描述：

读GPRS通信的SIM卡号（20位长度），SIM卡号具有不可更改的特性，可以用为设备的唯一身份识别码。

接口定义：

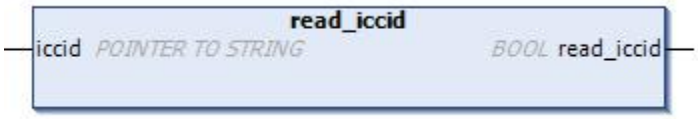
read_iccid (FUN)

FUNCTION read_iccid : BOOL

InOut:

Scope	Name	Type
Return	read_iccid	BOOL
Input	iccid	POINTER TO STRING

函数框图：



ST调用语句示例：

```
iccid:string(24) ;                          //首先定义一个24位的字符串变量
Booted:bool := false;

IF  not booted  THEN      //在初始化时调用一次
     Booted := true;
     read_iccid(ADR（iccid));
END_IF
```

调用说明：

(1)该函数使用时，必须先定义一个长度大于20个字符，且为8的倍数的字符串变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（6）read_ip（）

功能描述：

读控制器内部的IP地址。用户在使用网线进行程序下装时，或者在使用WIFI无线网络访问控制器的WEB界面时，均需要知道控制器的IP才能正常使用。一般情况下，控制器的出厂默认IP地址为192.168.1.253。在使用多个控制器的控制系统中，每个控制器的IP地址不能重复，因此客户会给不同的控制器分配不同的IP地址，在这种情况小，如果客户不能确定一台控制器的IP地址是多少，可以使用该功能块进行查询。

接口定义：

read_ip (FUN)

FUNCTION read_ip : BOOL

InOut:

Scope	Name	Type
Return	read_ip	BOOL
Input	ip	POINTER TO STRING

函数框图：

ST调用语句示例：

```
ip:string(24);                          //首先定义一个24位的字符串变量
Booted:bool := false;

IF  not booted  THEN      //在初始化时调用一次
     Booted := true;
     read_ip(ADR(ip));
END_IF
```

调用说明：

(1)该函数使用时，必须先定义一个长度大于等于16个字符，且为8的倍数的字符串变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（7）set_ip（）

功能描述：

写控制器内部的IP地址。控制器的出厂默认IP地址为192.168.1.253。在使用多个控制器的控制系统中，每个控制器的IP地址不能重复，此时客户可以使用该函数给不同的控制器分配不同的IP地址。

接口定义：

set_ip (FUN)

FUNCTION set_ip : BOOL

InOut:

Scope	Name	Type
Return	set_ip	BOOL
Input	ip	POINTER TO STRING

函数框图：

ST调用语句示例：

```
ip:string(24);                          //首先定义一个24位的字符串变量
Booted:bool := false;

IF  not booted  THEN      //在初始化时调用一次
     Booted := true;
     Set_ip(ADR(ip));
END_IF
```

调用说明：

(1)该函数使用时，必须先定义一个长度大于等于16个字符，且为8的倍数的字符串变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（8）read_location（）

功能描述：

读控制器的基站定位地址。控制器没有GPS，但是可以通过查询GPRS反馈回来的基站位置信息来确定设备所在的地理位置。

接口定义：

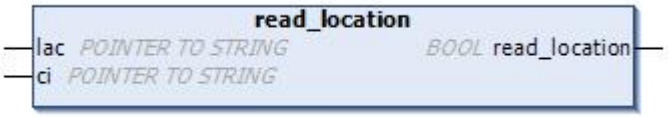
read_location (FUN)

FUNCTION read_location : BOOL

InOut:

Scope	Name	Type
Return	read_location	BOOL
Input	lac	POINTER TO STRING
	ci	POINTER TO STRING

函数框图：



ST调用语句示例：

```
GPRS_lac:string(8) ;    //首先定义一个8位的字符串变量
GPRS_ci:string(8) ;     //首先定义一个8位的字符串变量
Booted:bool := false;

IF  not booted  THEN      //在初始化时调用一次
    Booted := true;
    read_location(ADR(GPRS_lac),ADR(GPRS_ci));
END_IF
```

调用说明：

(1)该函数使用时，先定义一个长度大于等于8的字符串变量；

(2)调用语句不能放在主循环程序中，因为否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；由于定位是基于基站，所以不支持实时位置更新，建议只上电获取一次地址。

（9）read_sn（）

功能描述：

读控制器的产品序列号。产品序列号被用作产品质量追溯的身份码，云平台上进行远程监控和远程升级等操作都是基于产品序列号开展数据解析和升级操作的。一般的心跳信息都要求带产品序列号上传。

接口定义：

read_sn (FUN)

FUNCTION read_sn : BOOL

InOut:

Scope	Name	Type
Return	read_sn	BOOL
Input	sn	POINTER TO STRING

函数框图：

ST调用语句示例：

```
Sn:string(16) ;                    //首先定义一个16位的字符串变量
Booted:bool := false;

IF  not booted  THEN      //在初始化时调用一次
     Booted := true;
     read_sn(ADR(sn));
END_IF
```

调用说明：

(1)该函数使用时，必须先定义一个长度大于等于15个字符，且为8的倍数的字符串变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（10）read_wifi_passwd（）

功能描述：

读控制器的WIFI密码。用户在使用WIFI无线网络访问控制器的WEB界面时，需要知道控制器的WIFI密码才能正常使用。

接口定义：

read_wifi_passwd (FUN)

FUNCTION read_wifi_passwd : BOOL

InOut:

Scope	Name	Type
Return	read_wifi_passwd	BOOL
Input	passwd	POINTER TO STRING

函数框图：

ST调用语句示例：

```
wifi_passwd:string(16) ;                    //首先定义一个16位的字符串变量
Booted:bool := false;

IF  not booted  THEN      //在初始化时调用一次
     Booted := true;
     read_wifi_passwd(ADR(wifi_passwd));
END_IF
```

调用说明：

(1)该函数使用时，必须先定义一个长度大于等于16个字符的字符串变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（11）set_wifi_passwd（）

功能描述：

设置控制器的WIFI密码。由于使用默认的WIFI密码，存在被非预期的用户访问到WEB界面，甚至更改控制器程序的风险，建议用户在设备出厂前给WIFI设置专用的WIFI密码，作为管理控制器的WEB界面访问和程序更新权限的一种手段。

接口定义：

set_wifi_passwd (FUN)

FUNCTION set_wifi_passwd : BOOL

InOut:

Scope	Name	Type
Return	set_wifi_passwd	BOOL
Input	passwd	POINTER TO STRING
	len	DINT

函数框图：

ST调用语句示例：

```
wifi_passwd:string(16) := 'yours_password';          //首先定义一个16位的字符串变量
len:DINT;
Booted:bool := false;

IF  not booted  THEN      //在初始化时调用一次
     Booted := true;
     len := len(wifi_passwd);
     set_wifi_passwd(ADR(wifi_passwd)，len);
END_IF
```

调用说明：

(1)该函数使用时，必须先定义一个长度大于等于16个字符的字符串变量和一个双整型变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（12）read_work_mins（）

功能描述：

读控制器的工作小时计。工作小时计信息经常被用于质保期的确定，该信息由控制器底层程序生成和记录，一般1分钟记录一次，记录的单位为分钟，如果需要在界面上显示小时计，则需要将读取回来的分钟数转换成小时数进行显示。

接口定义：

read_work_mins (FUN)

FUNCTION read_work_mins : DINT

InOut:

Scope	Name	Type
Return	read_work_mins	DINT

函数框图：

ST调用语句示例：

```
work_mins:DINT;      //首先定义一个双整型变量（最大范围2147483647）

work_mins:=read_work_mins();      //调用函数
```

调用说明：

(1)该函数使用时，必须先定义一个双整型变量，与函数中返回值的变量类型一致；
(2)调用语句可以放在主循环程序中，实时更新；如果小时计数值是1分钟更新一次，为了节省资源，减少读取文件的次数，也可以1min读一次；

（13）Set_piv01（）

功能描述：

设置控制器低频PI端口（PIO/PI3）的脉冲幅度门限值。控制器的出厂默认PI门限值为12V，可以对幅值为13V至32V范围内的方波正脉冲进行准确计数，不支持含0V以下负脉冲的波形计数。如果客户需要对其他幅值（尤其是幅值低于13V）的脉冲进行计数，可以使用该函数进行门限值的设置，门限值设置的原则是门限值必须低于被测波形真实幅值1V以上。设置值为mV值，推荐设置范围为：1000～12000（对应1V至12V），极限范围可以设置为16.5V，但不推荐使用。

接口定义：

set_piv01 (FUN)

FUNCTION set_piv01 : BOOL

InOut:

Scope	Name	Type	Initial	Comment
Return	set_piv01	BOOL		
Input	piv_mv	DINT	10000	default: 10V

函数框图：

ST调用语句示例：

```
Booted:bool := false;

IF not booted THEN      //在初始化时调用一次
    Booted := true;
    Set_piv01(1000);    //设置 控制器左侧 PI-00 和 PI-01 的门槛值为1000mV（1V）
END_IF
```

调用说明：

(1)该函数使用时，可以不定义变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；

（14）Set_piv23（）

功能描述：

设置控制器高频PI端口（PI1/PI2）的脉冲幅度门限值。控制器的出厂默认PI门限值为12V，可以对幅值为13V至32V范围内的方波正脉冲进行准确计数，不支持含0V以下负脉冲的波形计数。如果客户需要对其他幅值（尤其是幅值低于13V）的脉冲进行计数，可以使用该函数进行门限值的设置，门限值设置的原则是门限值必须低于被测波形真实幅值1V以上。设置值为mV值，推荐设置范围为：1000～12000（对应1V至12V），极限范围可以设置为16.5V，但不推荐使用。

接口定义：

set_piv23 (FUN)

FUNCTION set_piv23 : BOOL

InOut:

Scope	Name	Type	Initial	Comment
Return	set_piv23	BOOL		
Input	piv_mv	DINT	10000	default: 10V

函数框图：

ST调用语句示例：

```
Booted:bool := false;

IF not booted THEN      //在初始化时调用一次
    Booted := true;
    Set_piv23(7600);   //设置 控制器右侧 PI-02 和 PI-03 的门槛值为7600mV（7.6V）
END_IF
```

调用说明：

(1)该函数使用时，可以不定义变量；

(2)调用语句不能放在主循环程序中，否则可能因多次频繁的文件操作造成存储介质损坏。应放在初始化程序代码中，每次上电仅需要执行一次；