可编程控制器脚本编程手册

文件编号	MGT-MODE-3-003		
文件状态	定稿		
当前版本	V1.02		
拟 制	刘鹏飞	日期	2024-12-12
审 核		日期	
批 准		日期	

麦 格 米 特 控 制 技 术 有 限 公 司

1 引言

1.1 编写目的

本手册阐述了编制 Mc5000,Mc5100 系列脚本程序需要的功能和规定的设定方法。应仔细阅读该手册，在充分理解的基础上编制程序。

实际编程时，关于 CPU 内置功能的指令及扩展单元专用指令，请浏览《MC 系列可编程控制器编程参考手册》、《Mc280 系列用户手册》等各单元的用户手册。

1.2 术语与缩写解释

在该手册的说明中，除部分内容外，请使用以下术语。

缩写、术语	解 释
可编程控制器	是可以通过更改程序来自由控制折本的电子装置，也称之位 Plc(可编程逻辑控制器)
Mc5000,Mc5100	是本公司制造的可编程控制器
MEGreator	支持 MC5000 系列及 MU 系列可编程控制器的软件
梯形图程序	通过 MEGreator 编写的程序

2 概要

2.1 脚本特点

可编程控制器的脚本是为了便于编程用以前的 LD（梯形图）语言难以编程的运算处理和字符串处理等而开发的语言。由于脚本能够夹在 LD 语言中编程，因此使用时不需要中断 LD 程序。

LD 程序适合使用驱动设备的控制和传感器等进行反馈控制的编程。

对于处理数值的复杂的运算处理及处理字符串等的程序，其编程变得复杂，调试和维护也变得困难。脚本是为了弥补这些梯形图程序的主要问题而开发的编程语言。用脚本代替用 LD 程序难以说明的复杂部分，可有效地编制程序。

Mc5x00 系列 Plc 的脚本具有如下特点：

1>.脚本采用标准 C 语言编程方式，支持 C 语言的所有语法，方便灵活,容易入门；

2>.脚本中支持 Plc 本身元件,如位 X, Y, M, S, SM, T, C; 字元件 D, R, SD ,Z,F 等；元件用作原始类型，

如 X 用作位，D 用用作字类型，可以直接使用；其它的类型需要使用使用脚本提供的函数才可以使用；

3>. 支持脚本定义全局变量，预留多达 64K 字节的栈；

4>. 支持 ANSI C 语言支持的所有基础类型，详见 **C 语言关键字**描述；

5>. 支持标准 C 库；

6>. 导出常用的算法及其运动控制库；

2.2 脚本类型

1> 框脚本：

框脚本是能够以 LD 指定何时、以什么顺序执行的脚本程序、带 LD 执行条件的脚本。框脚本只有在以 LD 指定的执行条件成立期间才执行框脚本，因此比较适合于运算式的编程。

图 1

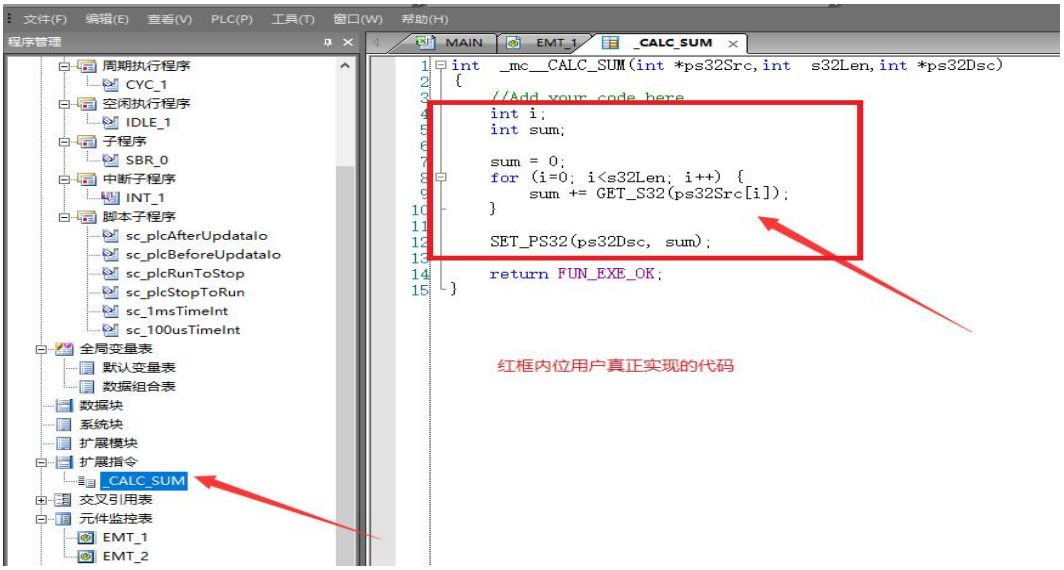


图 2

1> 钩子脚本：

钩子脚本和钩子程序一样，是系统留给用户的"后门"，让用户可以在自己程序中实现特殊的功能。如系统上电钩子,用户可以在系统上电时候执行自己的程序；"stop to run"钩子,用户可以在 plc 由停止到运行时候执行自己的程序；"run to stop"钩子，用户可以在 plc 由运行到停止时候执行自己的程序，等等。用户

只需要关注自己脚本的功能。

不同的型号支持的钩子功能和数目是不一样。

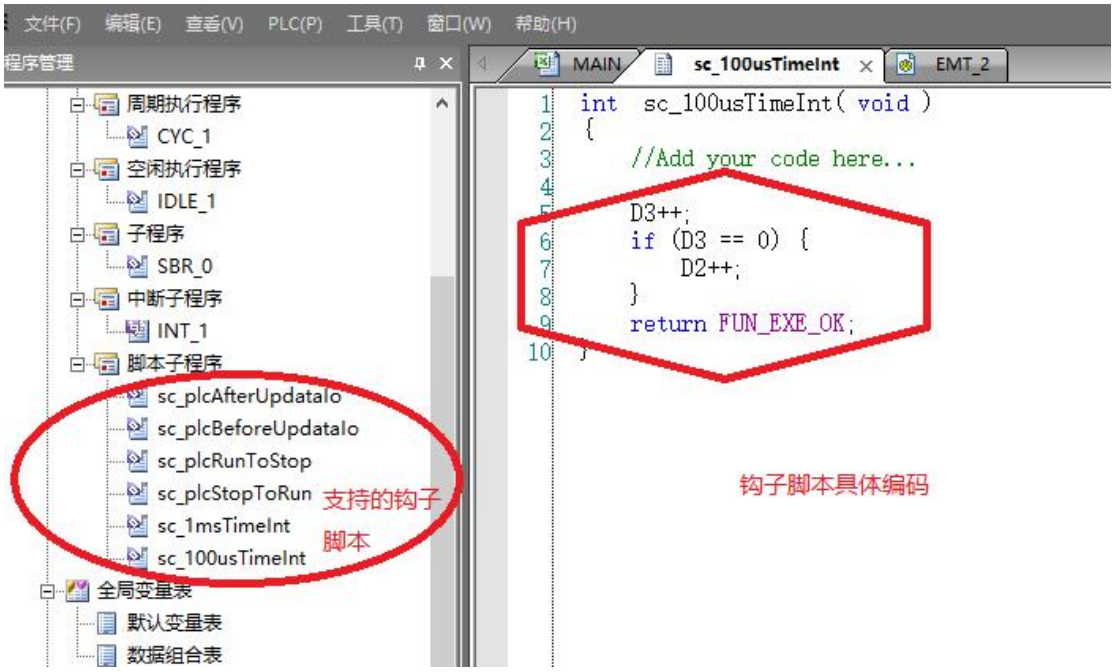


图 3

2.3 脚本功能

使用脚本，可以将 LD 程序编程的一系列控制，用标准 C 语言的编程格式编程。

新建带脚本功能的 plc，都会导出一下三个文件。

文件	功能
export_module.h	Plc 内核导出给脚本用的功能，脚本可以用这些已经实现好的积木来搭房子。
user_common.c	用户写的脚本的公共代码，实现更高的代码服用。
user_common.h	用户写的脚本的公共代码头文件。

3 如何编写脚本

在 xBuild 软件中可以直接使用 C 语言来进行编程，即编写框脚本或钩子脚本。下面用一个简单的例子来说明如何使用框脚本。

目标：将 D1000 开始的 20000 个字元件求和放到 D998(长整型)中，求平均值放到 D996(浮点型)中。

3.1 添加用户扩展指令

在“工程管理”中,右键点击“扩展指令”，弹出右键菜单，选择“添加扩展指令”，进入第二部。

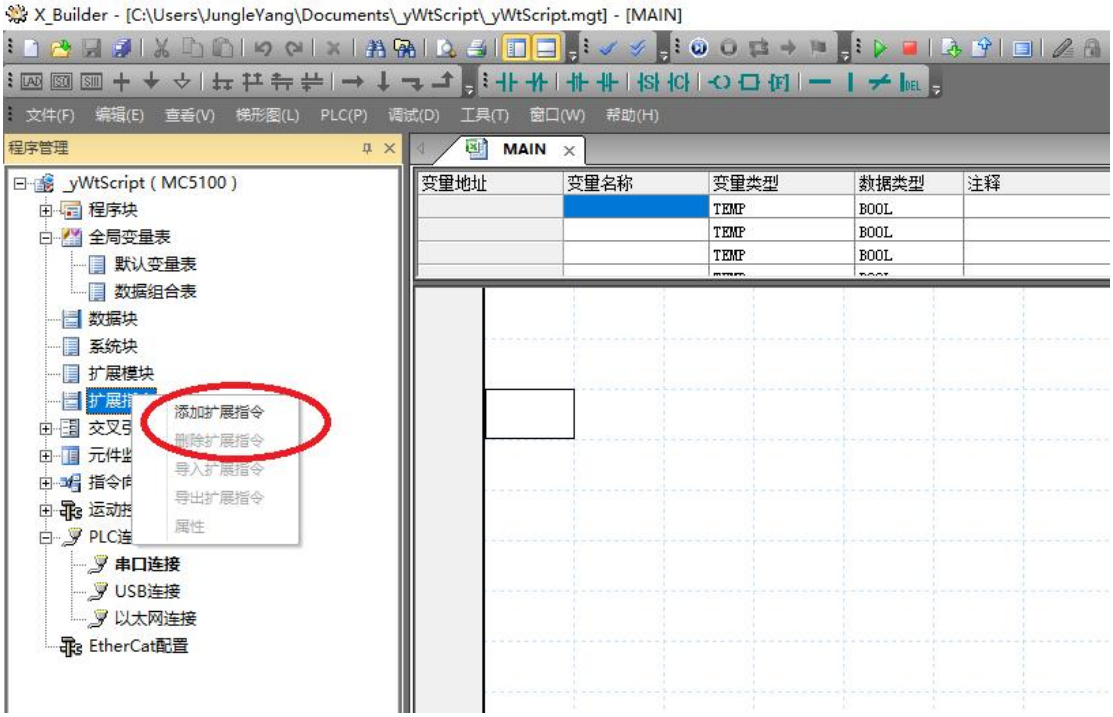


图 4

3.2 填充扩展指令参数

第一步完成后，将弹出如下对话框：

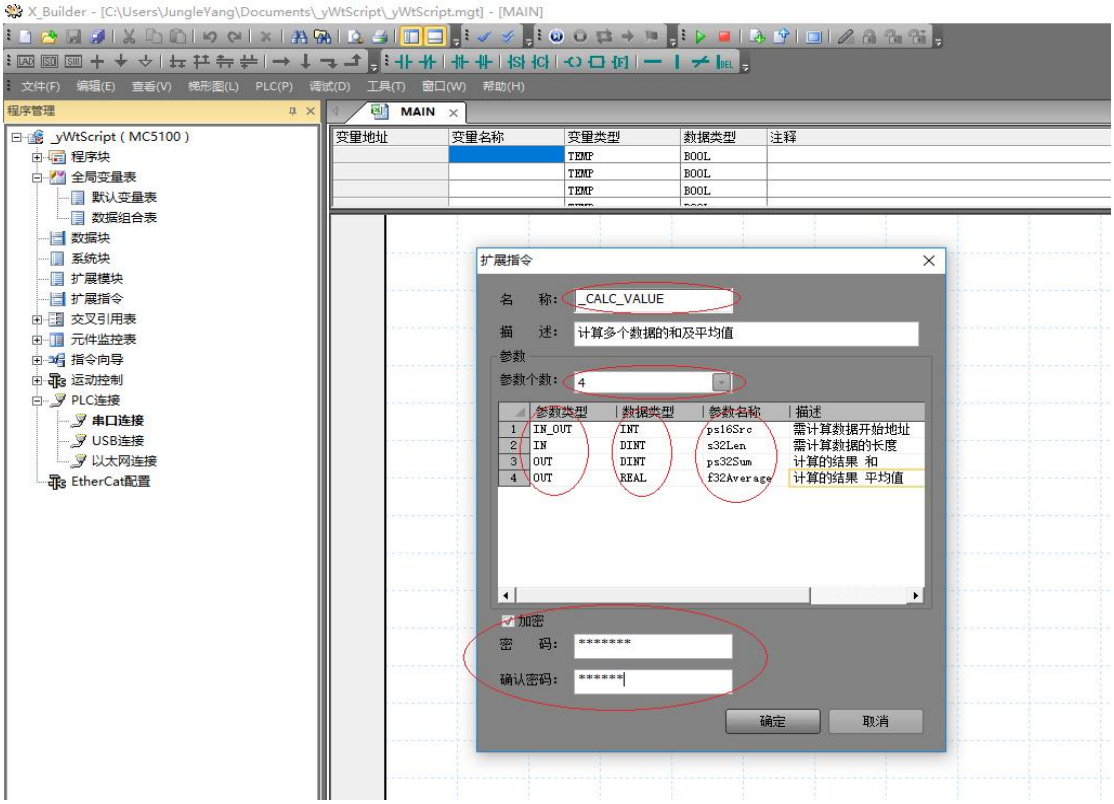


图 5

1> 名称：类似于梯形图指令中的 MOV, ADD,SUB 等，这里取得是_CALC_VALUE，名称第一个字符必行是'_',字母必须大写，必须符合标准 C 函数名称。

2> 描述：扩展指令功能说明性文字。

3> 参数个数：指令的输入,输出，输入输出参数的个数,如 MOV 为 2 个参数，这里为 4 个参数。

4> 参数设置：**参数类型**分为 IN 表示输入参数,是值传递；
参数类型分为 IN_OUT 表示输入输出参数，是指针传递；
参数类型分为 OUT 表示输出参数，是指针传递；
数据类型为 BOOL 表示"bool"行，位参数；
数据类型为 WORD 表示"unsigned short"，16 位无符号整型；
数据类型为 INT 表示"signed short"，16 位有符号整型；
数据类型为 DWORD 表示"unsigned int"，32 位无符号整型；
数据类型为 DINT 表示"signed short"，32 位有符号整型；
数据类型为 REAL 表示"float"，单精度实数；
参数名称就是最后出现函数的参数名字；
描述就是对参数含义说明，方便用户写程序；

5> 加密：用户对自己所写的程序加密，起到保护用户程序的目的。
全部选择完成后，点击确定。

3.3 编辑自定义的代码

第二步完成后，可以看到扩展指令下面已经有了”_CALC_VALUE”指令，双击指令，就可以查看和编辑代码。如果在第二步设置了密码，则需要正确的输入第二步设定密码，才可以查看和编辑代码。

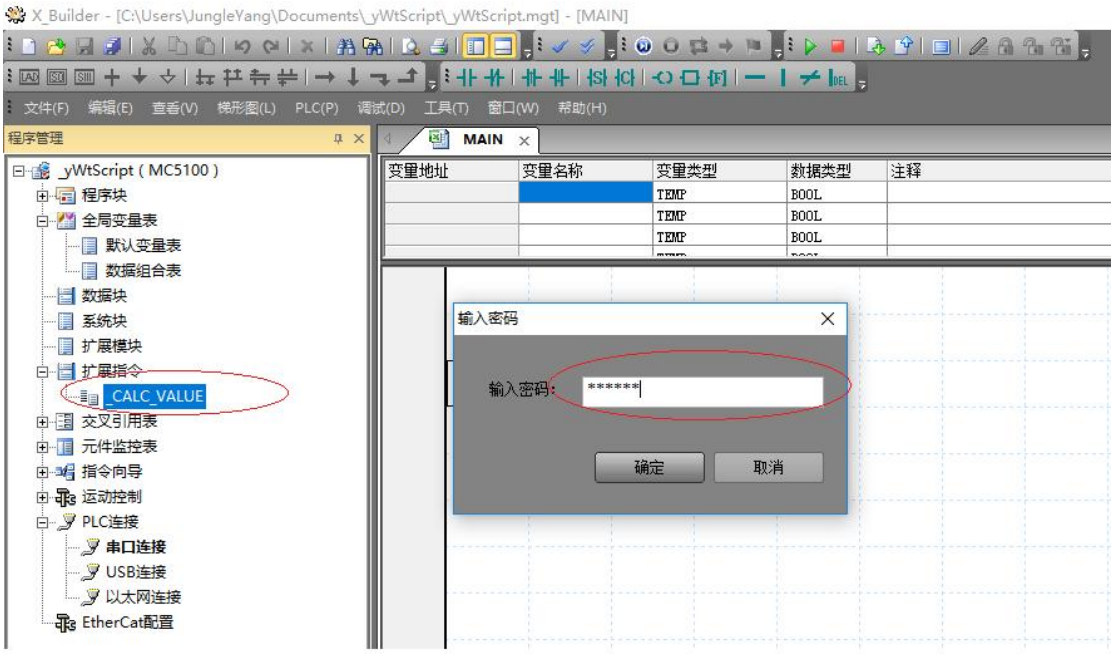


图 6

编辑代码结束后，见下图：

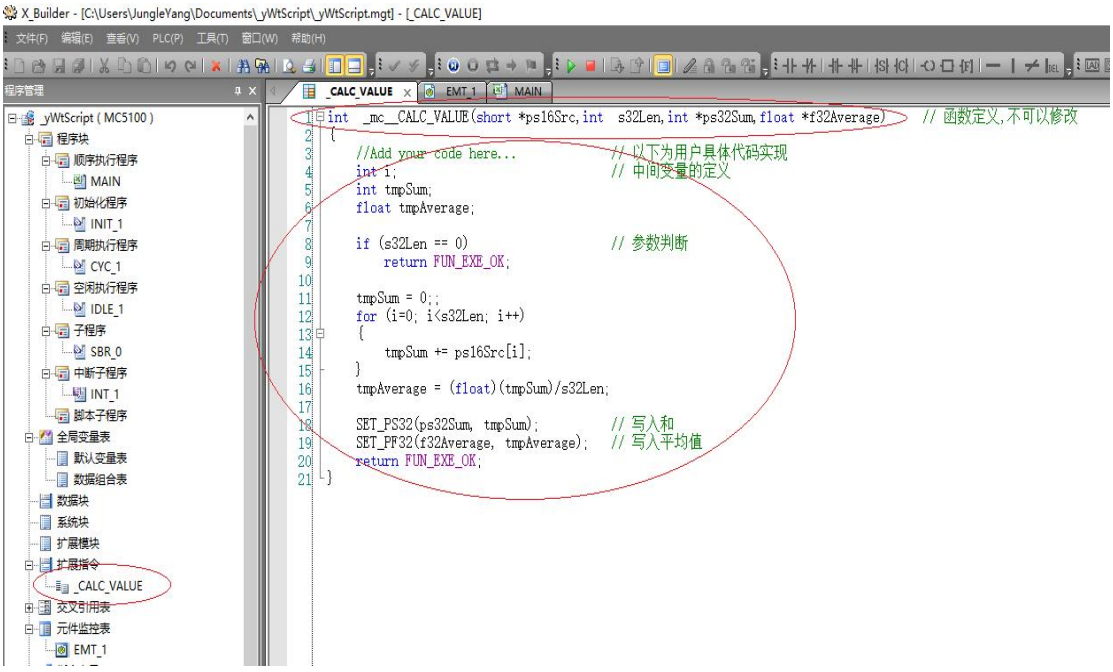


图 7

注意绿色的字体对含义的说明，下面大的椭圆圈内的内容为用户写得”C 语言”代码。

3.4 使用扩展指令

现在就可以像使用其他梯形图指令一样使用这条指令了。建立如下梯形图。

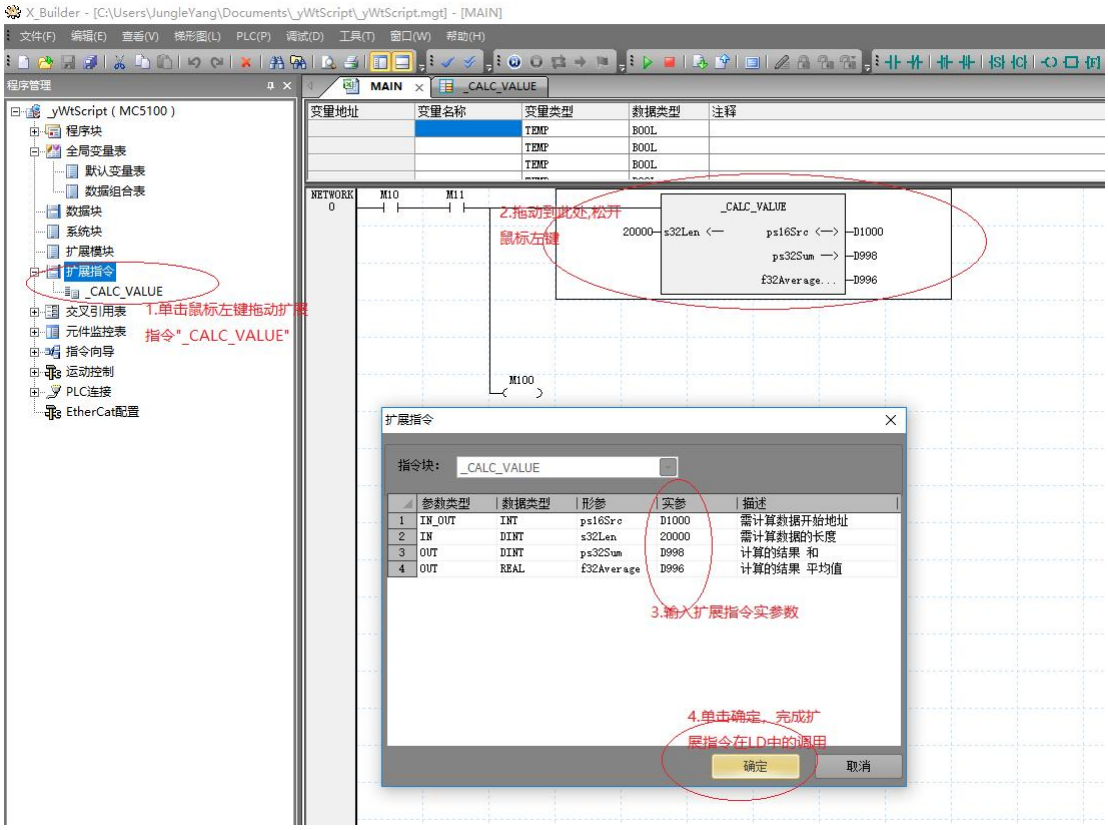


图 8

当 M10 和 M11 同时为 On 时，就执行扩展指令"_CALC_VALUE"， _CALC_VALUE 指令就将 D1000 开始的 20000 个字元件求和放到 D998(长整型)中，求平均值放到 D996(浮点型)中。如下图所示：

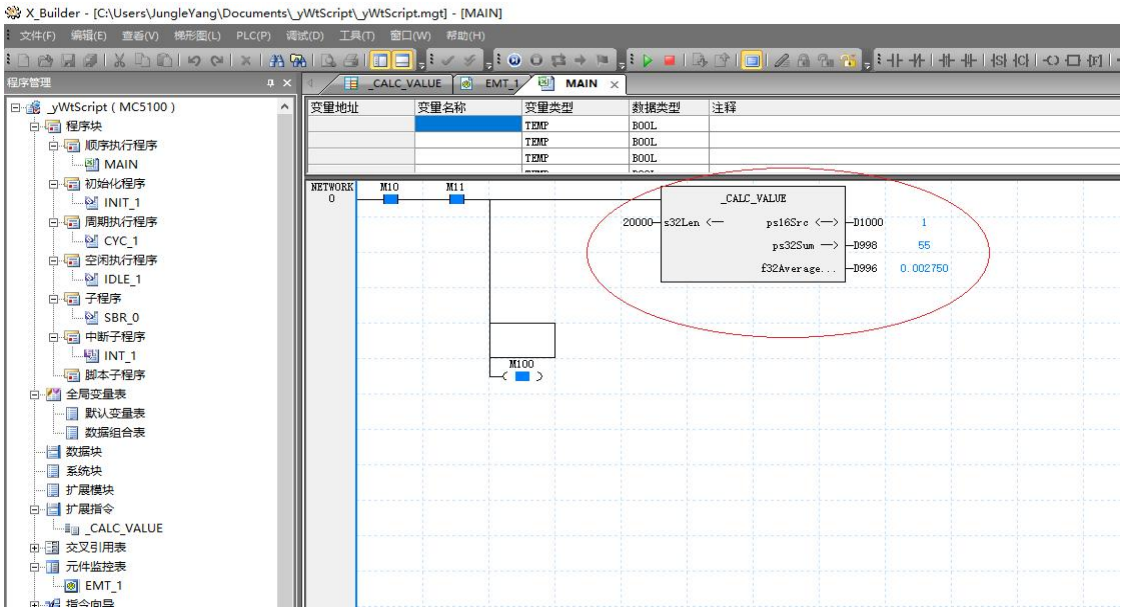
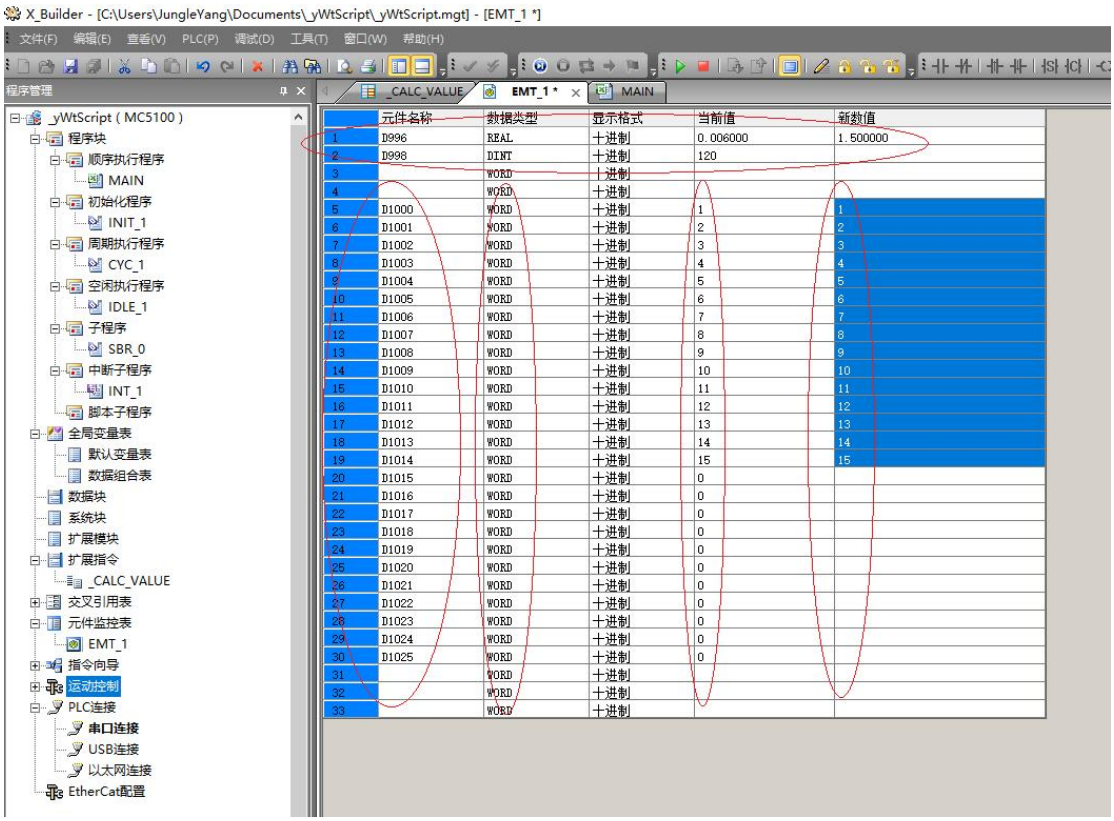


图 9

3.5 调试脚本

上面四部已经完成了编写一个脚本程序，且在梯形图中应用。接下来就可以像调试梯形图程序一样调试脚本。



	元件名称	数据类型	显示格式	当前值	新数值
1	D996	REAL	十进制	0.006000	1.500000
2	D998	DINT	十进制	120	
3		WORD	十进制		
4		WORD	十进制		
5	D1000	WORD	十进制	1	1
6	D1001	WORD	十进制	2	2
7	D1002	WORD	十进制	3	3
8	D1003	WORD	十进制	4	4
9	D1004	WORD	十进制	5	5
10	D1005	WORD	十进制	6	6
11	D1006	WORD	十进制	7	7
12	D1007	WORD	十进制	8	8
13	D1008	WORD	十进制	9	9
14	D1009	WORD	十进制	10	10
15	D1010	WORD	十进制	11	11
16	D1011	WORD	十进制	12	12
17	D1012	WORD	十进制	13	13
18	D1013	WORD	十进制	14	14
19	D1014	WORD	十进制	15	15
20	D1015	WORD	十进制	0	
21	D1016	WORD	十进制	0	
22	D1017	WORD	十进制	0	
23	D1018	WORD	十进制	0	
24	D1019	WORD	十进制	0	
25	D1020	WORD	十进制	0	
26	D1021	WORD	十进制	0	
27	D1022	WORD	十进制	0	
28	D1023	WORD	十进制	0	
29	D1024	WORD	十进制	0	
30	D1025	WORD	十进制	0	
31		WORD	十进制		
32		WORD	十进制		
33		WORD	十进制		

图 10

4 脚本导出的系统变量函数

4.1 系统变量

4.1.1 X 位变量

变量名称	X
功能描述	PLC 输入点
范围	X0 ~ X7777
类型	Bit
注意事项	8 进制定义，字母’X’大写

应用示例：

```
int  _mc__BitOp()
{
    //Add your code here...
    if ( X4 )
    {
        Y10 = 1;
```

8进制编号

注意：支持 X[10]输入，编译不会出错，但实际编号错误，所以请勿使用 X[XX]输入模式。

4.1.2 Y 位变量

变量名称	Y
功能描述	PLC 输出点
范围	Y0～Y7777
类型	Bit
注意事项	8 进制定义，字母’Y’大写

应用示例：

```
int  _mc__BitOp()
{
    //Add your code here...
    if ( X4 )
    {
        Y10 = 1;
```

Y位元件输入，编号位8进制

注意：支持 Y[15]输入，编译不会出错，但实际编号错误，所以请勿使用 Y[XX]输入模式。

4.1.3 SM 位变量

变量名称	SM
功能描述	PLC SM 元件
范围	SM0～SM4095
类型	Bit
注意事项	10 进制定义，字母’SM’大写

应用示例：

```
int  _mc__BitOp()
{
    //Add your code here...
    if ( X4 )
    {
        SM[40]=1;
        SM41=1;
```

SM位元件支持SMxx及SM[xx]

注意：SM 元件编号按十进制输入。

4.1.4 S 位变量

变量名称	S
功能描述	PLC S 元件
范围	S0 ~ S4095
类型	Bit
注意事项	10 进制定义，字母’S’大写

应用案例：

```
int _mc__BitOp()
{
    //Add your code here...
    if ( X4 )
    {
        S[100]=1;
        S101 = 1;
```

S位元件支持Sxx及S[xx]输入，编号为十进制

4.1.5 T 位变量

变量名称	T
功能描述	PLC T 元件
范围	T0 ~ T4095
类型	Bit
注意事项	10 进制定义，字母’T’大写

```
int _mc__BitOp()
{
    //Add your code here...
    if (T10)
    {
        D[102] =20;
    }
```

T元件位变量，支持Txx及T[xx]输入，十进制编号

4.1.6 C 位变量

变量名称	C
功能描述	PLC C 元件
范围	C0 ~ C4095
类型	Bit
注意事项	10 进制定义，字母’C’大写

```
int _mc__BitOp()
{
    //Add your code here...
    if ( C[10])
    {
        D100 = 10;
    }
```

C位元件输入格式支持Cxx及C[xx],十进制编号

4.1.7 M 位变量

变量名称	C
功能描述	PLC M 元件
范围	M0 ~ M65535
类型	Bit
注意事项	10 进制定义，字母'M'大写

```
int _mc__BitOp()
{
    //Add your code here...

    if ( X4 )
    {
        M0 = 1;
        M[1] = 1;
```

M位元件支持Mxx及M[xx]输入，十进制编号

4.1.8 SD 字变量

变量名称	SD
功能描述	PLC SD 元件
范围	SD0 ~ SD4095
类型	字，signed short 定义
注意事项	10 进制定义，字母'SD'大写

应用示例：

```
int _mc__BitOp()
{
    //Add your code here...

    if ( X4 )
    {
        D110 = SD101;
        D[111] = SD [102];
```

SD支持SDxx及SD[xx]

注意：32 位的 SD 元件值，只能通过运算处理，单独读写 16 位的 SD 元件来获取正确的 32 位数据！

4.1.9 Z 字变量

变量名称	Z
功能描述	PLC Z 元件
范围	Z0 ~ Z4095
类型	字，signed short 定义
注意事项	10 进制定义，字母'Z'大写

应用案例：

```
int _mc__BitOp()
{
    //Add your code here...

    if ( X4 )
    {
        Z110 = 10;
        Z[4095] = 100;
```

Z字元件支持Zxx及Z[xx]读写操作

4.1.10 D 字变量

变量名称	D
功能描述	PLC　D 元件
范围	D0～D65535
类型	字，signed short 定义
注意事项	10 进制定义，字母’D’大写

应用案例：

```
int  _mc__BitOp()
{
    //Add your code here...
    if ( X4 )
    {
      D110 = SD101;
      D[111] = SD [102];
```

D字单元直接读写支持Dxx及D[xx]

4.1.11 R 字变量

变量名称	R
功能描述	PLC　R 元件
范围	R0～R65535
类型	字，signed short 定义
注意事项	10 进制定义，字母’R’大写

应用案例：

```
int  _mc__BitOp()
{
    //Add your code here...
    if ( X4 )
    {
      R110 = 10;
      R[4095] = 100;
```

R字变量支持Rxx及R[xx]直接读写操作

4.2 系统函数

MC5000 系列 PLC 数据结构为大端数据，所以在对长整数或浮点数等双字（32 位）的数据进行读写操作时，需要进行大小端数据调整，需用到系统函数进行处理！

4.2.1　读取 D 元件的单个长整数值

函数名称	int GET_DD(unsigned short stNum)
功能描述	读取内部字寄存器“D”的双字值
输入参数	stNum:　D 元件的开始地址
输出参数	无
返回值	int，得到有符号长整数的值

注意事项	无

应用案例：

```
int _mc__BitOp()
{
    //Add your code here...

    if ( X4 )
    {
        long tmp;
        tmp = GET_DD(1000);
```

读取D1000长整型数据到tmp

4.2.2 写入单个长整数值到 D 元件内

函数名称	void SET_DD(unsigned short stNum, int val)
功能描述	写入单个长整数值到内部字寄存器“D”元件内
输入参数	stNum: D 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

```
int _mc__BitOp()
{
    //Add your code here...

    if ( X4 )
    {
        long tmp;
        tmp = GET_DD(1000);
        SET_DD(150,tmp);
```

将tmp值写入到长整数D150中

4.2.3 读取 D 元件的多个长整数值

函数名称	int GET_MutiDD(int stNum, int len, int *ps32Dsc)
功能描述	获取内部字寄存器“D”的双字值
输入参数	stNum: 读取 D 元件的开始地址
输入参数	Len : 读取双字元件的个数
输出参数	ps32Dsc: 存储读取到的长整数的地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	输出参数为指针型

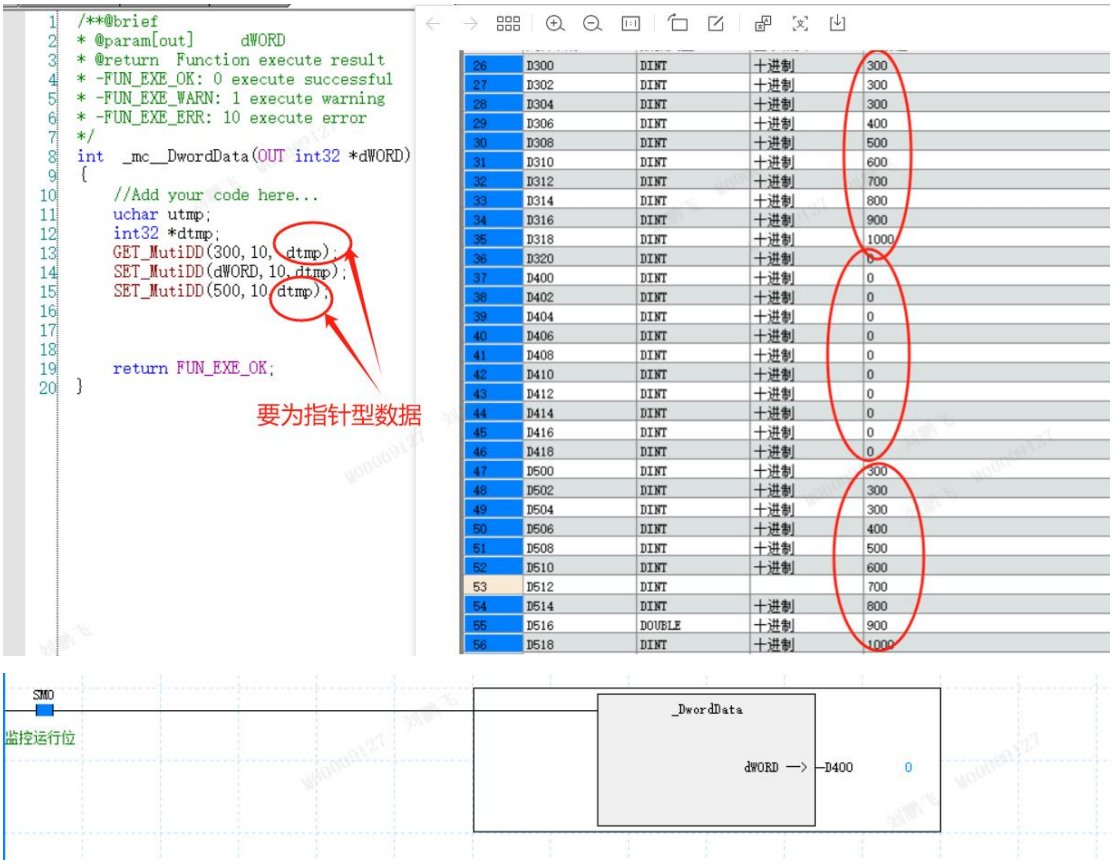
应用案例：参见 4.2.4

4.2.4 写入多个长整数值到 D 元件内

函数名称	int SET_MutiDD(int stNum, int len, int *ps32Src)
功能描述	写入单个长整数值到内部字寄存器“D”元件内
输入参数	stNum: 写入 D 元件的开始地址
输入参数	Len : 写入双字元件的个数

输出参数	ps32Dsc： 存储要写入到的 D 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	输出参数为指针型

应用案例：



26	D300	DINT	十进制	300
27	D302	DINT	十进制	300
28	D304	DINT	十进制	300
29	D306	DINT	十进制	400
30	D308	DINT	十进制	500
31	D310	DINT	十进制	600
32	D312	DINT	十进制	700
33	D314	DINT	十进制	800
34	D316	DINT	十进制	900
35	D318	DINT	十进制	1000
36	D320	DINT	十进制	0
37	D400	DINT	十进制	0
38	D402	DINT	十进制	0
39	D404	DINT	十进制	0
40	D406	DINT	十进制	0
41	D408	DINT	十进制	0
42	D410	DINT	十进制	0
43	D412	DINT	十进制	0
44	D414	DINT	十进制	0
45	D416	DINT	十进制	0
46	D418	DINT	十进制	0
47	D500	DINT	十进制	300
48	D502	DINT	十进制	300
49	D504	DINT	十进制	300
50	D506	DINT	十进制	400
51	D508	DINT	十进制	500
52	D510	DINT	十进制	600
53	D512	DINT		700
54	D514	DINT	十进制	800
55	D516	DOUBLE	十进制	900
56	D518	DINT	十进制	1000

说明：

1、示例中可以看出 GET_MutiDD(int stNum, int len, int *ps32Dsc)中的读取数据或 SET_MutiDD(int stNum, int len, int *ps32Src)写入数据的 D 元件地址，是立即数或具体数据的整型变量，不能是指针，dWORD 指向 D400，结果数据写不到 D400 开始的 10 长整数单元中。

2、参数中的 int *ps32Dsc 和 int *ps32Src 定义为长整型指针型变量。

4.2.5 读取 D 元件的单个单精度实数值

函数名称	float GET_FD(unsigned short stNum)
功能描述	读取内部字寄存器“D”的浮点数值
输入参数	stNum： D 元件的开始地址
输出参数	无
返回值	float, 得到浮点数的值
注意事项	无

应用案例请参考 4.2.6 中案例。

4.2.6 写入单个单精度实数值到 D 元件内

函数名称	void SET_FD(unsigned short stNum, float val)
功能描述	写入单个浮点数值到内部字寄存器“D”元件内
输入参数	stNum: D 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

应用案例：

```
int _mc__DwordData(OUT int32 *dWORD)
{
    //Add your code here...
    uchar utmp;
    int32 *dtmp;
    float fTmp , *pfTmp;
    fTmp = GET_FD(600);
    SET_FD(700,fTmp);
```

读取d600内浮点数存入变量fTmp中

将浮点数fTmp的值写入D700中

4.2.7 读取 D 元件的多个单精度实数值

函数名称	int GET_MutiFD(int stNum, int len, float *pf32Dsc)
功能描述	获取内部字寄存器“D”的浮点数值
输入参数	stNum: 读取 D 元件的开始地址
输入参数	Len : 读取双字元件的个数
输出参数	ps32Dsc: 存储读取到的浮点数值的地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

应用案例请参考 4.2.8 中案例。

4.2.8 写入多个单精度实数值到 D 元件内

函数名称	int SET_MutiFD(int stNum, int len, float *pf32Src)
功能描述	写入单个长整数值到内部字寄存器“D”元件内
输入参数	stNum: 写入 D 元件的开始地址
输入参数	Len : 写入双字元件的个数
输出参数	ps32Dsc: 存储要写入到的 D 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

应用案例：

```
int  _mc__DwordData(OUT int32 *dWORD)
{
    //Add your code here...
    uchar utmp;
    int32 *dtmp;
    float fTmp , *pfTmp;
    int i = 610;
    GET_MutiFD(i,5,pfTmp);
    SET_MutiFD(i+20,5,pfTmp);
```

读取从D610开始的5个浮点数，存入指针变量pfTmp中

将指针变量pfTmp中连续5个浮点数存入D630开始地址中

注意变量 i 的类型要是 16 位整型数据！不然指令执行不能正确寻址！

4.2.9 读取 R 元件的单个长整数值

注意：读取 R 元件的操作指令用法和对应的 D 元件一致，只是指令名称有变化而已，每个指令用法具体可以参考对应的 D 元件指令案例，不再另行做例程。

函数名称	int GET_DR(unsigned short stNum)
功能描述	读取内部字寄存器“R”的双字值
输入参数	stNum: R 元件的开始地址
输出参数	无
返回值	int, 得到有符号长整数的值
注意事项	无

4.2.10 写入单个长整数值到 R 元件内

函数名称	void SET_DR(unsigned short stNum, int val)
功能描述	写入单个长整数值到内部字寄存器“R”元件内
输入参数	stNum: R 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

4.2.11 读取 R 元件的多个长整数值

函数名称	int GET_MutiDR(int stNum, int len, int *ps32Dsc)
功能描述	获取内部字寄存器“R”的双字值
输入参数	stNum: 读取 R 元件的开始地址
输入参数	Len : 读取双字元件的个数
输出参数	ps32Dsc: 存储读取到的长整数的地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

4.2.12 写入多个长整数值到 R 元件内

函数名称	int SET_MutiDR(int stNum, int len, int *ps32Src)
功能描述	写入单个长整数值到内部字寄存器“R”元件内
输入参数	stNum: 写入 R 元件的开始地址
输入参数	Len : 写入双字元件的个数
输出参数	ps32Dsc: 存储要写入到的 R 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

4.2.13 读取 R 元件的单个单精度实数值

函数名称	float GET_FR(unsigned short stNum)
功能描述	读取内部字寄存器“R”的浮点数值
输入参数	stNum: R 元件的开始地址
输出参数	无
返回值	float, 得到浮点数的值
注意事项	无

4.2.14 写入单个单精度实数值到 R 元件内

函数名称	void SET_FR(unsigned short stNum, float val)
功能描述	写入单个浮点数值到内部字寄存器“R”元件内
输入参数	stNum: R 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

4.2.15 读取 R 元件的多个单精度实数值

函数名称	int GET_MutiFR(int stNum, int len, float *pf32Dsc)
功能描述	获取内部字寄存器“R”的浮点数值
输入参数	stNum: 读取 R 元件的开始地址
输入参数	Len : 读取双字元件的个数
输出参数	ps32Dsc: 存储读取到的浮点数值的地址

返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

4.2.16 写入多个单精度实数值到 R 元件内

函数名称	int SET_MutiFR(int stNum, int len, float *pf32Src)
功能描述	写入单个长整数值到内部字寄存器“R”元件内
输入参数	stNum: 写入 R 元件的开始地址
输入参数	Len : 写入双字元件的个数
输出参数	ps32Dsc: 存储要写入到的 R 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

4.2.17 读取 F 元件的单个长整数值（包括 F0～F9）

函数名称	int GET_DF(int stNum)
功能描述	读取内部字寄存器“F”的双字值
输入参数	stNum: F 元件的开始地址
输出参数	无
返回值	int, 得到有符号长整数的值
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.18 写入单个长整数值到 F 元件内（包括 F0～F9）

函数名称	void SET_DF(int stNum, int val)
功能描述	写入单个长整数值到内部字寄存器“F”元件内
输入参数	stNum: F 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.19 读取 F 元件的多个长整数值（包括 F0～F9）

函数名称	int GET_MutiDF(int stNum, int len, int *ps32Dsc)
功能描述	获取内部字寄存器“F”的双字值

输入参数	stNum： 读取 F 元件的开始地址
输入参数	Len ： 读取双字元件的个数
输出参数	ps32Dsc： 存储读取到的长整数的地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.20 写入多个长整数值到 F 元件内（包括 F0～F9）

函数名称	int SET_MutiDF(int stNum, int len, int *ps32Src)
功能描述	写入单个长整数值到内部字寄存器“F”元件内
输入参数	stNum： 写入 F 元件的开始地址
输入参数	Len ： 写入双字元件的个数
输出参数	ps32Dsc： 存储要写入到的 F 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.21 读取 F 元件的单个单精度实数值（包括 F0～F9）

函数名称	float GET_FF(unsigned short stNum)
功能描述	读取内部字寄存器“F”的浮点数值
输入参数	stNum: F 元件的开始地址
输出参数	无
返回值	float, 得到浮点数的值
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.22 写入单个单精度实数值到 F 元件内（包括 F0～F9）

函数名称	void SET_FF(unsigned short stNum, float val)
功能描述	写入单个浮点数值到内部字寄存器“F”元件内
输入参数	stNum: F 元件的开始地址
输入参数	val ： 要写入的值
输出参数	无
返回值	无
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.23 读取 F 元件的多个单精度实数值（包括 F0～F9）

函数名称	int GET_MutiFF(int stNum, int len, float *pf32Dsc)
功能描述	获取内部字寄存器“F”的浮点数值
输入参数	stNum: 读取 F 元件的开始地址
输入参数	Len : 读取双字元件的个数
输出参数	ps32Dsc: 存储读取到的浮点数值的地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.24 写入多个单精度实数值到 F 元件内（包括 F0～F9）

函数名称	int SET_MutiFF(int stNum, int len, float *pf32Src)
功能描述	写入单个长整数值到内部字寄存器“F”元件内
输入参数	stNum: 写入 F 元件的开始地址
输入参数	Len : 写入双字元件的个数
输出参数	ps32Dsc: 存储要写入到的 F 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.25 读取 Fx 元件的单个长整数值（F0 ... F9）

函数名称	int GET_DF0(int stNum)
	int GET_DF1(int stNum)
	int GET_DF2(int stNum)
	int GET_DF3(int stNum)
	int GET_DF4(int stNum)
	int GET_DF5(int stNum)
	int GET_DF6(int stNum)
	int GET_DF7(int stNum)
	int GET_DF8(int stNum)
	int GET_DF9(int stNum)
功能描述	读取内部字寄存器“Fx”的双字值
输入参数	stNum: Fx 元件的开始地址
输出参数	无
返回值	int, 得到有符号长整数的值
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.26 写入单个长整数值到 Fx 元件内（F0 ... F9）

函数名称	void SET_DF0(int stNum, int val)
	void SET_DF1(int stNum, int val)
	void SET_DF2(int stNum, int val)
	void SET_DF3(int stNum, int val)
	void SET_DF4(int stNum, int val)
	void SET_DF5(int stNum, int val)
	void SET_DF6(int stNum, int val)
	void SET_DF7(int stNum, int val)
	void SET_DF8(int stNum, int val)
	void SET_DF9(int stNum, int val)
功能描述	写入单个长整数值到内部字寄存器“Fx”元件内
输入参数	stNum: Fx 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.27 读取 Fx 元件的多个长整数值（F0 ... F9）

函数名称	int GET_MutiDF0(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF1(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF2(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF3(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF4(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF5(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF6(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF7(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF8(int stNum, int len, int *ps32Dsc)
	int GET_MutiDF9(int stNum, int len, int *ps32Dsc)
功能描述	获取内部字寄存器“Fx”的双字值
输入参数	stNum: 读取 Fx 元件的开始地址
输入参数	Len : 读取双字元件的个数
输出参数	ps32Dsc: 存储读取到的长整数的地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.28 写入多个长整数值到 Fx 元件内（F0 ... F9）

函数名称	int SET_MutiDF0(int stNum, int len, int *ps32Src)
	int SET_MutiDF1(int stNum, int len, int *ps32Src)
	int SET_MutiDF2(int stNum, int len, int *ps32Src)
	int SET_MutiDF3(int stNum, int len, int *ps32Src)
	int SET_MutiDF4(int stNum, int len, int *ps32Src)
	int SET_MutiDF5(int stNum, int len, int *ps32Src)
	int SET_MutiDF6(int stNum, int len, int *ps32Src)
	int SET_MutiDF7(int stNum, int len, int *ps32Src)
	int SET_MutiDF8(int stNum, int len, int *ps32Src)
	int SET_MutiDF9(int stNum, int len, int *ps32Src)
功能描述	写入单个长整数值到内部字寄存器“F”元件内
输入参数	stNum: 写入 F 元件的开始地址
输入参数	Len : 写入双字元件的个数
输出参数	ps32Dsc: 存储要写入到的 F 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.29 读取 Fx 元件的单个单精度实数值（F0 ... F9）

函数名称	float GET_FF0(unsigned short stNum)
	float GET_FF1(unsigned short stNum)
	float GET_FF2(unsigned short stNum)
	float GET_FF3(unsigned short stNum)
	float GET_FF4(unsigned short stNum)
	float GET_FF5(unsigned short stNum)
	float GET_FF6(unsigned short stNum)
	float GET_FF7(unsigned short stNum)
	float GET_FF8(unsigned short stNum)
	float GET_FF9(unsigned short stNum)
功能描述	读取内部字寄存器“Fx”的浮点数值
输入参数	stNum: Fx 元件的开始地址
输出参数	无
返回值	float, 得到浮点数的值
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.30 写入单个单精度实数值到 Fx 元件内（F0 … F9）

函数名称	void SET_FF0(unsigned short stNum, float val)
	void SET_FF1(unsigned short stNum, float val)
	void SET_FF2(unsigned short stNum, float val)
	void SET_FF3(unsigned short stNum, float val)
	void SET_FF4(unsigned short stNum, float val)
	void SET_FF5(unsigned short stNum, float val)
	void SET_FF6(unsigned short stNum, float val)
	void SET_FF7(unsigned short stNum, float val)
	void SET_FF8(unsigned short stNum, float val)
	void SET_FF9(unsigned short stNum, float val)
功能描述	写入单个浮点数值到内部字寄存器“Fx”元件内
输入参数	stNum: Fx 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.31 读取 Fx 元件的多个单精度实数值（F0 … F9）

函数名称	int GET_MutiFF0(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF1(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF2(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF3(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF4(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF5(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF6(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF7(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF8(int stNum, int len, float *pf32Dsc)
	int GET_MutiFF9(int stNum, int len, float *pf32Dsc)
功能描述	获取内部字寄存器“Fx”的浮点数值
输入参数	stNum: 读取 Fx 元件的开始地址
输入参数	Len : 读取双字元件的个数
输出参数	ps32Dsc: 存储读取到的浮点数值的地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.32 写入多个单精度实数值到 Fx 元件内（F0 ... F9）

函数名称	int SET_MutiFF0(int stNum, int len, float *pf32Src)
	int SET_MutiFF1(int stNum, int len, float *pf32Src)
	int SET_MutiFF2(int stNum, int len, float *pf32Src)
	int SET_MutiFF3(int stNum, int len, float *pf32Src)
	int SET_MutiFF4(int stNum, int len, float *pf32Src)
	int SET_MutiFF5(int stNum, int len, float *pf32Src)
	int SET_MutiFF6(int stNum, int len, float *pf32Src)
	int SET_MutiFF7(int stNum, int len, float *pf32Src)
	int SET_MutiFF8(int stNum, int len, float *pf32Src)
	int SET_MutiFF9(int stNum, int len, float *pf32Src)
功能描述	写入单个长整数值到内部字寄存器“F”元件内
输入参数	stNum: 写入 F 元件的开始地址
输入参数	Len : 写入双字元件的个数
输出参数	ps32Dsc: 存储要写入到的 F 元件数据地址
返回值	等于 0 函数执行正确,其他值函数执行失败
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.33 读取 Fx 元件的单个短整数值（F0 ... F9）

函数名称	signed short GET_F0(int stNum)
	signed short GET_F1(int stNum)
	signed short GET_F2(int stNum)
	signed short GET_F3(int stNum)
	signed short GET_F4(int stNum)
	signed short GET_F5(int stNum)
	signed short GET_F6(int stNum)
	signed short GET_F7(int stNum)
	signed short GET_F8(int stNum)
	signed short GET_F9(int stNum)
功能描述	读取内部字寄存器“Fx”的单字值
输入参数	stNum: Fx 元件的开始地址
输出参数	无
返回值	signed short, 得到有符号短整数的值
注意事项	无

案例详见 4.2.34 下 F0~F9 数据存贮器应用案例。

4.2.34 写入单个短整数值到 Fx 元件内（F0 ... F9）

函数名称	void SET_F0(int stNum, signed short val)
	void SET_F1(int stNum, signed short val)
	void SET_F2(int stNum, signed short val)
	void SET_F3(int stNum, signed short val)
	void SET_F4(int stNum, signed short val)
	void SET_F5(int stNum, signed short val)
	void SET_F6(int stNum, signed short val)
	void SET_F7(int stNum, signed short val)
	void SET_F8(int stNum, signed short val)
	void SET_F9(int stNum, signed short val)
功能描述	写入单个短整数值到内部字寄存器“Fx”元件内
输入参数	stNum: Fx 元件的开始地址
输入参数	val : 要写入的值
输出参数	无
返回值	无
注意事项	无

F0~F9 数据存贮器应用案例：

```
int  _mc__FmemeryOp(IN_OUT uint16 *FInPra,OUT uint16 *FOut)
{
    //Add your code here...
    int iTemp,iTemp1;
    int iAdr= 400,iAdr1=300;
    float fData= 90.12;
    int *piData;
    float *pfData;

    iTemp = GET_DF(300);//F1正常32位读和写
    SET_DF(400,iTemp);
    iTemp1 = GET_DF1(300);//F1如果用GET_DF(地址)写是16位的数据，不支持32位的
    SET_DF1(400,iTemp1);
    iTemp1 = GET_DF9(iAdr);//F9地址数据写不进去，自己变动

    SET_FF4(iAdr,40.12);
    SET_FF5(iAdr,50.12);
    SET_FF6(iAdr,60.12);
    SET_FF7(iAdr,70.12);
    SET_FF8(iAdr,fData*10);
    SET_FF8(iAdr+10,fData);

    GET_MutiDF0(iAdr1,5,piData);
    SET_MutiDF4(iAdr1,5,piData);

    GET_MutiFF2(iAdr1,5,pfData);
    SET_MutiFF8(iAdr1,5,pfData);
```

长整数多个读写参数值为指针变量

多个单精度数据度和写的参数使用指针变量

4.2.35 设置 int 型指针的值(大小端调整)

函数名称	void SET_PS32(int *ps32Dsc, int s32Src)
功能描述	把 s32Src 的值写到 ps32Dsc 所指向的地址
输入参数	s32Src: 要写入的数值
输出参数	ps32Dsc: 存储要写入值的地址
返回值	无
注意事项	无

应用案例：

```
#define DD *(int32 *)&D
#define FD *(float *)&D
#define DR *(int32 *)&R
int  _mc__pointerOP(IN int32  InPra,IN_OUT uint16 *IOPra,OUT uint16 *OPra)
{
    //Add your code here...

    int *Adr1,*Adr2,Adr3 = 620;
    long dInt1,*dInt2;

    float fData1,*fData2;
    float fTemp = 568.12;
    long DTemp = 654321;
    long temp;

    Adr1 = &D600;
    Adr2 = &R550;

    SET_PS32(Adr1,DTemp);
```

写入目标位指针指向的地址

被写数据为立即数

//Adr1为指针，将长整数DTemp的值写入Adr1指向的地址（大小端调整）

备注：写入目标地址指针，可以自行定义，如图所示地址指针 Adr1 指向 D600，ADr2 指向 R550。

4.2.36 获取 int 型指针的值(大小端调整)

函数名称	int GET_PS32(int *ps32Src)
功能描述	获取 int 型指针的值
输入参数	ps32Src:　要读取值的地址
输出参数	无
返回值	获取到的值
注意事项	无

应用案例：

```
#define DD *(int32 *)&D
#define FD *(float *)&D
#define DR *(int32 *)&R
int  _mc__pointerOP(IN int32  InPra,IN_OUT uint16 *IOPra,OUT uint16 *OPra)
{
    //Add your code here...

    int *Adr1,*Adr2,Adr3 = 620;
    long dInt1,*dInt2;

    float fData1,*fData2;
    float fTemp = 568.12;
    long DTemp = 654321;
    long temp;

    Adr1 = &D600;
    Adr2 = &R550;

    SET_PS32(Adr1,DTemp);   //Adr1为指针，将长整数DTemp的值写入Adr1指向的地址（大小端调整）
    dInt1 = GET_PS32(IOPra);  //将指针IOPra指向的地址的值读出来，赋给dInt1
    SET_PS32(Adr1+1,dInt1);   //将dInt1的值经大小端调整后赋给指针（Adr1+1）指向的地址
```

操作数为指向地址的指针

备注：写入目标地址指针，可以自行定义，如图所示地址指针 IOPra 由函数实参给定。

4.2.37 获取 int 型数据的值(大小端调整)

函数名称	int GET_S32(int s32Src)
功能描述	获取 int 型数据的值，即 32 位长整数变量或立即数。
输入参数	s32Src:　要转化的值
输出参数	无

返回值	转换后的值，即经过大小端调整的数据
注意事项	经指令大小端调整后的数据，可以直接赋给 PLC 的数据寄存器；

应用案例：参考下图*1

```
#define DD *(int32 *)&D
#define FD *(float *)&D
#define DR *(int32 *)&R
int  _mc__pointerOP(IN int32  InPra,IN_OUT uint16 *IOPra,OUT uint16 *OPra)
{
    //Add your code here...

    int *Adr1,*Adr2,Adr3 = 620;
    long dInt1,*dInt2;

    float fData1,*fData2;
    float fTemp = 568.12;
    long DTemp = 654321;
    long temp;

    Adr1 = &D600;
    Adr2 = &R550;

    SET_PS32(Adr1,DTemp);        //Adr1为指针，将长整数DTemp的值写入Adr1指向的地址（大小端调整）
    dInt1 = GET_PS32(IOPra);    //将指针IOPra指向的地址的值读出来，赋给dInt1
    SET_PS32(Adr1+1,dInt1);     //将dInt1的值经大小端调整后赋给指针（Adr1+1）指向的地址
    dInt1 =-654321;
    temp = GET_S32(dInt1);  *1  //将长整数dInt的值经大小端调整后赋给变量Temp
    DD[650]= temp;              //将经过大小端调整的长整数结果Temp的值直接赋给D650
    dInt1 *= -2;
    SET_DR(Adr3,dInt1*10);      //将dInt1*10后的值经大小端调整后赋给指针Adr3指向的区R地址
    dInt1= GET_U32(650*100);*2  //将长整数（650*100)的值经大小端调整后赋给变量dInt1
    DR[652]=dInt1;              // 将经过大小端调整的长整数结果dInt1的值直接赋给R652
```

4.2.38 设置 unsigned int 型指针的值(大小端调整)

函数名称	void SET_PU32(unsigned int *pu32Dsc, unsigned int u32Src)
功能描述	把 u32Src 的值写到 pu32Dsc 所指向的地址
输入参数	u32Src:　要写入的数值
输出参数	pu32Dsc:　存储要写入值的地址
返回值	无
注意事项	无

应用案例：参考 4.4.23 案例

4.2.39 获取 unsigned int 型指针的值(大小端调整)

函数名称	unsigned int GET_PU32(unsigned int *pu32Src)
功能描述	获取 int 型指针的值
输入参数	pu32Src:　要读取值的地址
输出参数	无
返回值	获取到的值
注意事项	无

应用案例：参考 4.4.23 案例

4.2.40 获取 unsigned int 型数据的值(大小端调整)

函数名称	unsigned int GET_U32(unsigned int u32Src)
功能描述	获取 Unsigned int 型数据的值，即 32 位无符号长整数变量或立即数。
输入参数	u32Src: 要转化的值
输出参数	无
返回值	转换后的值，即经过大小端调整的数据
注意事项	经指令大小端调整后的数据，可以直接赋给 PLC 的数据寄存器；

应用案例：参考下图*2

```
#define DD *(int32 *)&D
#define FD *(float *)&D
#define DR *(int32 *)&R
int  _mc__pointerOP(IN int32  InPra,IN_OUT uint16 *IOPra,OUT uint16 *OPra)
{
    //Add your code here...

    int *Adr1,*Adr2,Adr3 = 620;
    long dInt1,*dInt2;

    float fData1,*fData2;
    float fTemp = 568.12;
    long DTemp = 654321;
    long temp;

    Adr1 = &D600;
    Adr2 = &R550;

    SET_PS32(Adr1,DTemp);        //Adr1为指针，将长整数DTemp的值写入Adr1指向的地址（大小端调整）
    dInt1 = GET_PS32(IOPra);    //将指针IOPra指向的地址的值读出来，赋给dInt1
    SET_PS32(Adr1+1,dInt1);     //将dInt1的值经大小端调整后赋给指针（Adr1+1）指向的地址
    dInt1 =-654321;
    temp = GET_S32(dInt1);  *1  //将长整数dInt的值经大小端调整后赋给变量Temp
    DD[650]= temp;              //将经过大小端调整的长整数结果Temp的值直接赋给D650
    dInt1 *= -2;
    SET_DR(Adr3,dInt1*10);      //将dInt1*10后的值经大小端调整后赋给指针Adr3指向的区R地址
    dInt1= GET_U32(650*100);*2  //将长整数（650*100)的值经大小端调整后赋给变量dInt1
    DR[652]=dInt1;              // 将经过大小端调整的长整数结果dInt1的值直接赋给R652
```

4.2.41 设置 float 型指针的值(大小端调整)

函数名称	void SET_PF32(float *pf32Dsc, float f32Src)
功能描述	把 f32Src 的值写到 pf32Dsc 所指向的地址
输入参数	f32Src: 要写入的数值
输出参数	pf32Dsc: 存储要写入值的地址
返回值	无
注意事项	无

应用案例：参考 4.4.23 案例

4.2.42 获取 float 型指针的值(大小端调整)

函数名称	float GET_PF32(float *pf32Src)
功能描述	获取 int 型指针的值
输入参数	pf32Src: 要读取值的地址

输出参数	无
返回值	获取到的值
注意事项	无

应用案例：参考 4.4.23 案例

4.2.43 获取 float 型数据的值(大小端调整)

函数名称	float GET_F32(float f32Src)
功能描述	获取 int 型数据的值
输入参数	f32Src:　要转化的值
输出参数	无
返回值	转换后的值
注意事项	无

应用案例：

```
#define DD *(int32 *)&D
#define FD *(float *)&D
#define DR *(int32 *)&R
int  _mc__pointerOP(IN int32  InPra,IN_OUT uint16 *IOPra,OUT uint16 *OPra)
{
    //Add your code here...

    int *Adr1,*Adr2,Adr3 = 620;
    long dInt1,*dInt2;

    float fData1,*fData2;
    float fTemp = 568.12;
    long DTemp = 654321,dTmp;
    long temp;
    short sTmp =123;

    Adr1 = &D600;
    Adr2 = &R550;

    SET_PU32(Adr2,sTmp*200);    //将长整数sTmp*200=24600大小端调整赋给指针ADr2指向的地址R550
    dTmp = GET_PU32(Adr2);      //将指针针ADr2指向的地址R550的长整数读出来大小端调整后，赋给dTmp
    SET_PU32(Adr1+20,dTmp);    //将长整数dTmp大小端调整赋给指针(ADr1+20)指向的地址D640

    fData1 = GET_F32(fTemp );   //将浮点数fTmp经大小端调整后赋给变量fData1
    FD[40] =fData1 ;            //已经大小端调整的浮点数据赋给PLC元件D40
    SET_PF32(Adr2+10,100.125);  //将浮点数100.125大小端调整赋给指针(ADr2+10)指向的地址R570
    fTemp=GET_PF32(Adr2+10)*10;  //指针(ADr2+10)指向的地址R570的浮点数经大小端调整乘以10后赋给变量fTemp
    SET_PF32(Adr2+12,fTemp);    //将浮点数fTemp大小端调整赋给指针(ADr2+12)指向的地址R572
```

指针案例程序

说明：

1、注意指针指向的地址+1 对应的 PLC 元件地址增加 2，如上图所示

2、GET_F32（float Src）指令执行后的数据是经过大小端调整的，不能再参与函数内运算，可以直接赋给 PLC 数据寄存器。如上例子的两句

```
dTmp = GET_PU32(Adr2);
SET_PU32(Adr1+20,dTmp);
```

等同于 FD[40]=GET_F32(fTemp);

5 脚本导出的算法函数

5.1.1 计算效验和 modbus crc

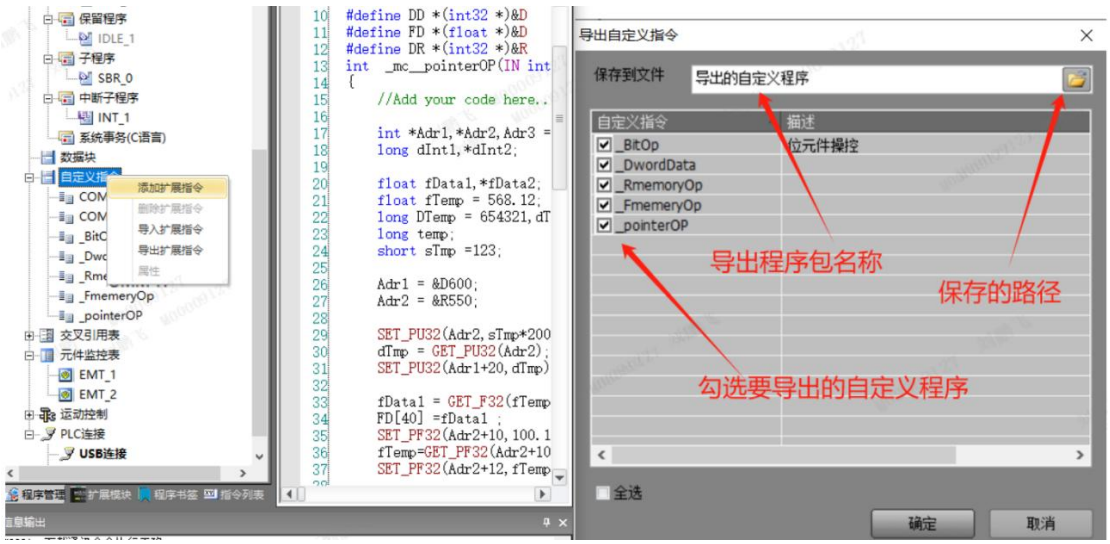
函数名称	unsigned short _ycrcModbus(unsigned char *data,unsigned int length)
功能描述	计算数据的 Modbus crc
输入参数	data: 要计算 Crc 数据的开始地址
输入参数	Length: 要计算 Crc 数据的长度
输出参数	无
返回值	计算得到的 Crc
注意事项	无

5.1.2 计算效验和 ccit crc

函数名称	unsigned short _ycrcCcitt(unsigned char *ptr, unsigned int len)
功能描述	计算数据的 ccitt 效验
输入参数	data: 要计算 Crc 数据的开始地址
输入参数	Length: 要计算 Crc 数据的长度
输出参数	无
返回值	计算得到的 Crc
注意事项	无

6 脚本导出的运动控制库函数

用户自定义指令支持导出、导入:右击自定义指令，弹出菜单选择导入或导出自定义指令。


保留程序
IDLE_1
子程序
SBR_0
中断子程序
INT_1
系统事务(C语言)
数据块
自定义指令
添加扩展指令
删除扩展指令
导入扩展指令
导出扩展指令
属性
_FmemeryOp
_pointerOP
交叉引用表
元件监控表
EMT_1
EMT_2
运动控制
PLC连接
USB连接
程序管理
扩展模块
程序书签
指令列表
#define DD *(int32 *)&D
#define FD *(float *)&D
#define DR *(int32 *)&R
int _mc__pointerOP(IN int
{
//Add your code here..
int *Adr1,*Adr2,Adr3 =
long dInt1,*dInt2;
float fData1,*fData2;
float fTemp = 568.12;
long DTemp = 654321,dT
long temp;
short sTmp =123;
Adr1 = &D600;
Adr2 = &R550;
SET_PU32(Adr2,sTmp*200
dTmp = GET_PU32(Adr2);
SET_PU32(Adr1+20,dTmp)
fData1 = GET_F32(fTemp
FD[40] =fData1 ;
SET_PF32(Adr2+10,100.1
fTemp=GET_PF32(Adr2+10
SET_PF32(Adr2+12,fTemp
导出自定义指令
保存到文件
导出的自定义程序
自定义指令
描述
_BitOp
位元件操控
_DwordData
_RmemoryOp
_FmemeryOp
_pointerOP
导出程序包名称
保存的路径
勾选要导出的自定义程序
全选
确定
取消