PCSuite IDE 编程环境使用手册

agito

www.agito-akribis.com

Member of Akribis Systems group

版本记录

版本	描述	作者	日期
1.0	首版发布	HN	2024/2/26
1.1	补充 3.3、3.4、3.9 部分内容	HN	2024/3/1
1.2	补充 3.4、3.4 章节变量、数组定义，以及 2.1 章节注意事项	HN	2024/3/21
1.3	修正 5.3 章节 MotionStat 中 bit 的说明	HN	2024/5/27
1.4	更新 4.5 章节 IO 示例	HN	2025/1/24
1.5	更新 4.5 章节数字输出例程	HN	2025/2/19
1.6	更新 4.5 章节数字输出例程	HN	2025/1/6

目录

1 介绍

1.1 关于手册

感谢您选择 Agito 系列运动控制产品，我们将竭力为您提供追求速度与精度的极致运动控制方案，并提供全方位的技术支持。

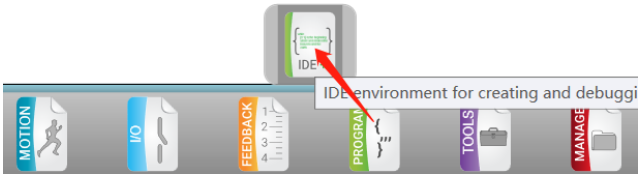
本手册主要介绍在 PCSuite IDE 编程环境的使用方法以及注意事项，帮助用户快速掌握在 IDE+开发用户程序。

1.2 PCSuite IDE 编程环境介绍

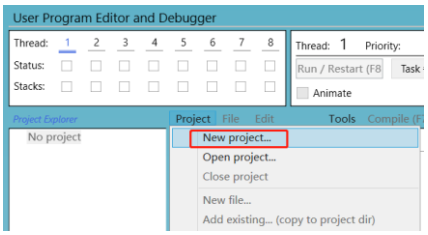
PCSuite 拥有强大和全面的集成开发环境 IDE+，用于控制程序的开发和控制单元的独立运行，我们的用户程序多线程任务同时运行（AGD 系列最大支持 8 线程，AGM800 最大支持 12 线程），直观的脚本语言便于 if/for/while/swith 等语句的编写，用户也可以自定义变量和表达式，设置中断事件函数，甚至可以使用类似于 C 语言的指针读取变量。根据具体的产品和应用，我们的控制器在 1us 内可以执行高达 500 个底层指令，同时 IDE+可支持创建多个任务文件夹，用户可以把常用的函数分组放到独立的文件夹，便于重复使用，IDE+的调试器也支持单行指令执行、断点和监控窗口，便于用户调试。

2 认识 PCSuite IDE 编程环境

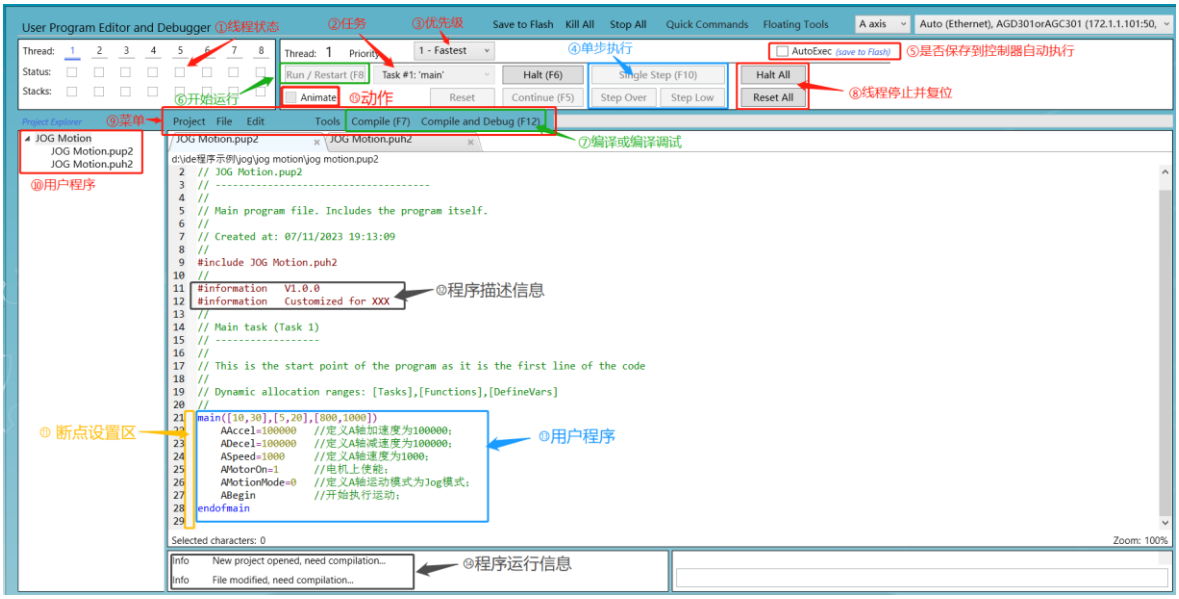
- 切换菜单栏 PROGRAM→IDE+，进入 IDE User-Program 编程环境界面；



- 点击 Project→New project 新建 User-Program 文件，将会在用户选择的目录中自动创建 1 个以项目名称命名的文件夹（详细可参阅本文 3.1 章节）；



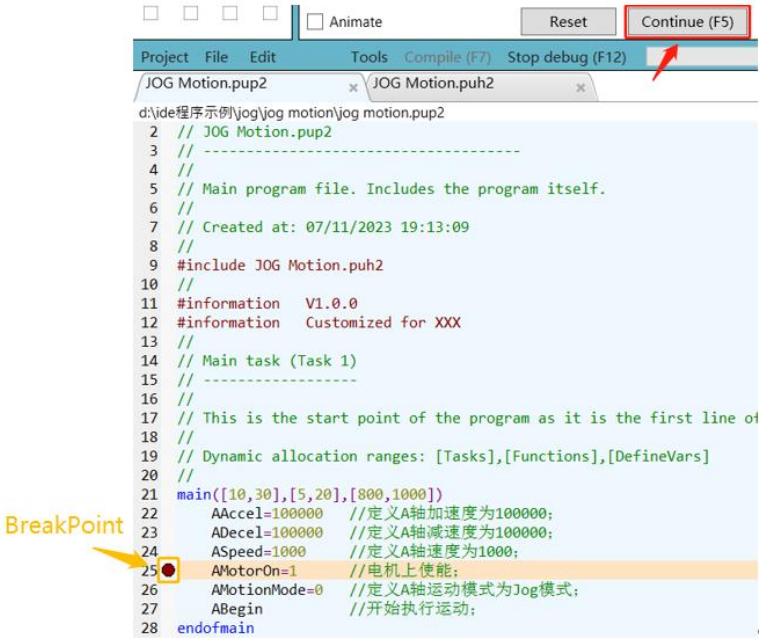
2.1 IDE 编程环境界面



- ①线程状态：控制器支持多线程多任务同时运行，线程的状态将在该处显示，该线程不在运行时为白色底色，程线程正在运行时显示绿色，线程报错时显示红色。当线程出现报错时，将鼠标放置于对应线程下方 Status “□”中将会显示报错信息，用户可根据报错信息排查报错原因；
- ②任务：此处将显示 User-Program 中定义的任务（编译后），用户可选择对应的任务后再点击“Run/Restart”后运行对应的任务；
- ③优先级：此处可定义主线程任务优先级，其他线程或任务优先级设置可参阅本文 3.9 章节内容；
- ④单步执行：点击“Single Step”可按顺序单步执行 User-Program 命令行；
- ⑤自动运行（AutoExec）：勾选后程序将会在上电时自动运行；

- ⑥开始运行：编译成功后（Compile），点击“Run/Restart”开始运行程序；
- ⑦编译/编译调试：用户程序编写完成后，编译程序生成低层程序，当有语法错误时会在编译时提示，值得注意的是，在编译运行时如电机为上使能状态，PCSuite 将会弹出电机下使能提示；
- ⑧线程停止/复位：“Halt All”将会停止全部线程，User-Program 命令为“AprogHalt-All”“Reset All”将会复位所有线程，User-Program 命令为“AProgResetAll”；
- ⑨菜单：菜单栏中包括项目文件新建、打开、关闭等操作，也包括一些工具，例如编译、下载、清除 BreakPoint 等操作；
- ⑩用户程序：用户项目将在该处显示，包括*.pup2 和*.puh2 文件，用户程序包含于*pup2 中，用户自定义变量、指针等包含于*puh2 中；

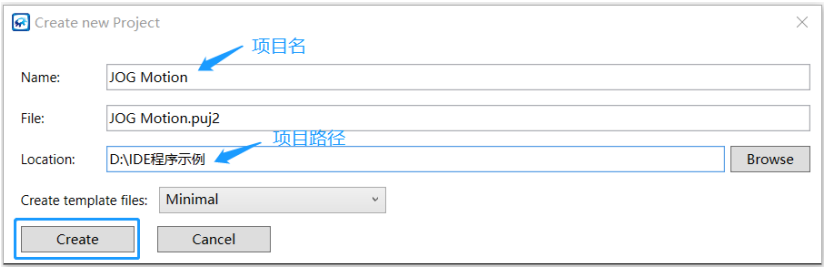
  ⑪BreakPoint：程序编译成功后，鼠标左键单击图示断点设置区对应用于所需的打断行的位置，将会在该行左侧出现一个圆点，即把该行设置为断点位置，程序将会执行到该断点上一行指令暂停，点击“Continue”或 F5 快捷键，程序将继续向后执行，支持设置多个断点；



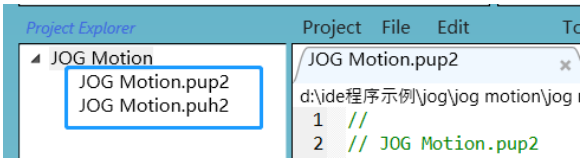
- ⑫程序描述信息：用户可以写入自定义信息，比如 User-Program 版本、功能等，信息将会以字符串的形式保存到控制器中，在 info 中可查询；
- ⑬用户程序：用户主程序在该区域编辑，包括函数、任务等；
- ⑭程序运行信息：用户程序在编译或运行过程中出现的一些调试信息将会在此处显示，包括报错信息等；
- ⑮动作（Animate）：勾选该选项后，程序将会在运行过程中实时显示程序当前运行的代码行，便于用户实时观测到程序流；
- **值得注意的是：重新下载固件后会擦除 flash 中的 User-Program 程序，需要重新编译保存**

2.2 程序运行步骤

- 第 1 步：点击“Project→New Project”新建项目，界面⑨，当然也可以点击“Project→Open Project”打开本地程序；

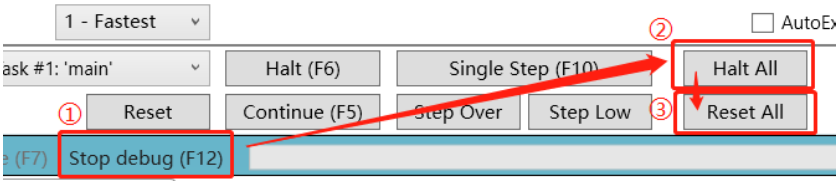


- 第 2 步：双击左侧 Project explorer 中*pup2 文件打开主程序编辑区编写程序，或双击 *puh2 文件的变量编辑区定义变量，界面⑩；

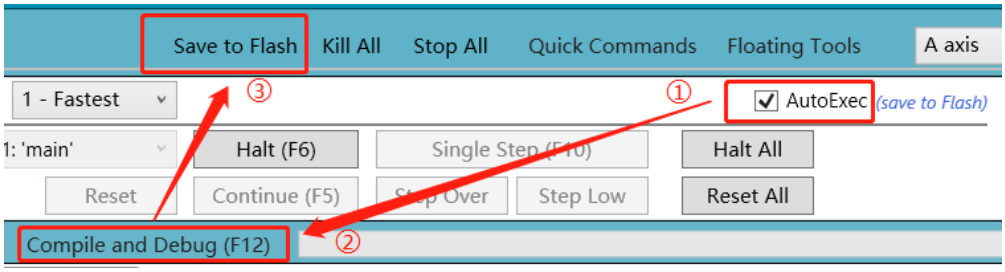


- 第 3 步：点击⑦中的“Compile and Debug”编译调试程序，当语法没有错误编译成功时，程序会变为淡蓝底色；
- 第 4 步：点击⑥中的“Run/Restart”，程序将开始执行；

停止复位线程步骤：Stop→Halt All→Reset All



2.3 保存 User-Program 到控制器 flash



- 第 1 步：依次点击“Project”--->“Open project”-->双击文件，打开本地 IDE User-Program；
- 第 2 步：勾选 2.1 IDE 界面中的⑤AutoExec(或者在 Terminal 中执行 AAutoExec=1 命令)；
- 第 3 步：点击⑦“Compile and Debug”编译运行；
- 第 4 步：点击“Save to flash”（或 Alt+S 快捷键）保存到控制器 flash，保存成功时，PCSuite 界面左侧会闪现“Save to flash”字样。

当用户将 User-Program 保存到控制器重新上电后，程序将会在上电后自动执行，

值得注意的是：保存到控制器中的为低层指令，而在 IDE 编程环境中的为高层程序，不支持将保存到控制器 flash 中的程序上传到 PCSuite，因此用户在使用时注意备份 User-Program 源程序到本地及其版本管控。

3 User-Program 基本语法

在使用 IDE 进行编写 User-Program 时，请遵循以下基本规则：

- 关键字：所有关键字前需要加轴号（即使该关键字为轴无关参数），关键字不区分大小写；
- 结束标志：语句以回车、换行或回车+换行结束，每行只能包含一条语句（可以附加行尾注释），且语句必须包含在一行中（没有连续行）；
- 大小写：程序语言区分大小写；
- 注释：支持单行注释（//...）和段注释（/*...*/）；
- 值范围：所有常量或变量及其运算结果都只支持带符号 Int32 类型数据，不支持浮点数，当运算结果未超限且不为整数时，将会对结果舍余取整（和 INT 取整有区别），当运算结果出现值超限或不符合数学基本准则（如除数不能为 0 等）程序运行时将会出现报错。

3.1 User-Program 文件

用户从 User Program Editor 中创建的新项目会在用户选择的目录中自动创建 1 个以项目名称命名的文件夹，该文件夹包含*pup2、*puh2、*puj2 三个文件。

- *.pup2

  用户程序包含于*pup2 中，目前 1 和 User_Program 只能包含 1 个*pup2 文件和多个*puh2 文件，且*pup2 文件名必须和项目名称相同，只有项目的.pup2 文件才会在编译过程中实际使用；

  pup2 文件可以包含理论上无限数量的注释，以增强程序的可读性，注释在编译过程中会被去除，并且.cup2 文件不包含注释（或用户程序执行不需要的任何数据）；

- *.puh2

  *.puh2 中包含用户自定义的常量、变量等；

- *.puj2

  *.puj2 为一个可扩展标记文件（类 XML）；

当用户编译（Compile）程序后，将会在项目文件夹中自动生成 3 个文件，分别为*.cup2、*.cupb2、*_DefineVars.h，

- *.cup2

  编译过程中输出文件保存在*.cup2 扩展名文件中，这是下载到控制器的文件，其包含转换为控制器 CPU 可执行的语句，以及控制器用于优化（大小和速度）执行用户程序的附加信息。

- *.cupb2

  由编译器内部生成，以启用编译和调试过程；

- *_DefineVars.h

  由编译器内部生成，以启用编译和调试过程；

3.2 编译器指令

用户程序包含程序编译器指令以扩展用户程序功能。

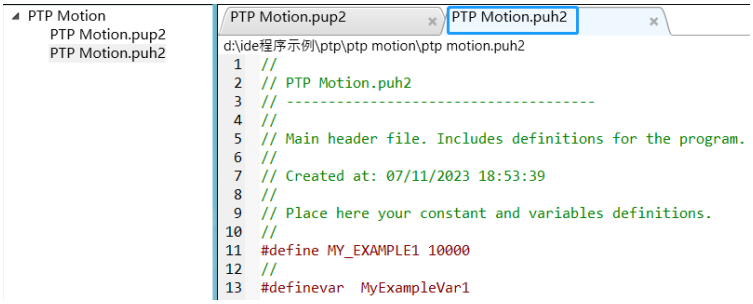
例如：

- 定义常/变量：#define

  语法格式：#define <variable_name> < variable_value>

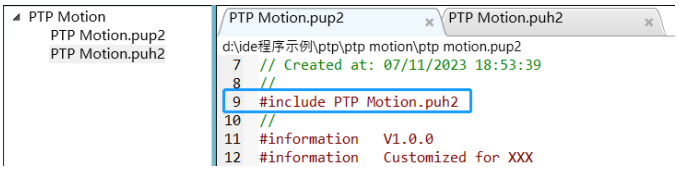
- 定义变量：#definevar

  语法格式：#definevar <variable_name>



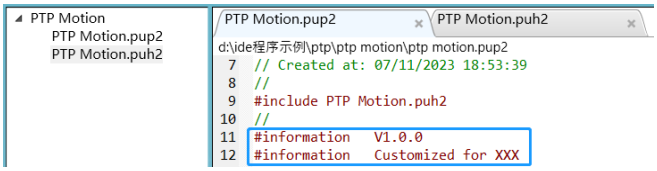
- 调用头文件：#include，同一个 Project 可以包含多个.puh2 文件；

  语法格式：#include <puh2_name.puh2>

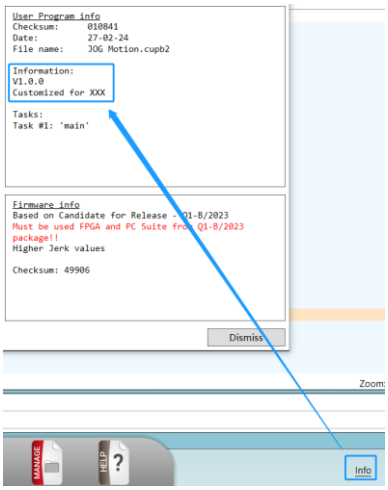


- 载入信息：#information，可下载控制器非易失性存储器中读取，例如可以存入 User-Program 版本信息、程序备注等信息；

  语法格式：#information < information_content>



在控制器中读取 information 内容：



3.3 常量和变量

User-Program 中使用的常量或变量都仅支持整型数据（带符号 32 位 int 数，即值在-2^{31}~2^{31}-1），因此用户在进行常量或变量赋值运算时注意可能的运算结果值大小，避免出现运行报错。

变量名允许字母、数字及下划线组合，但必须以字母开头，且长度需要超过 2 个字符。

定义常量：`#define <variable_name> < variable_value>`

定义变量：`#definevar <variable_name>`

以下以一元一次方程作为示例：

```
15  #define slop_k 10   //定义一元一次方程斜率为10
16  #define intercepts_b 50   //定义一元一次方程截距为50
17
18  #definevar X1   //定义自变量为X1
19  #definevar Y1   //定义因变量为Y1
```

```
23      while (1)
24          X1=AUserParam[5]    //此处为便于展示，将UserParam[5]作为自变量输入
25          Y1 = slop_k * X1 + intercepts_b
26          AUserParam[6]=Y1    //此处为便于展示，将因变量存入UserParam[6]
27      end
```

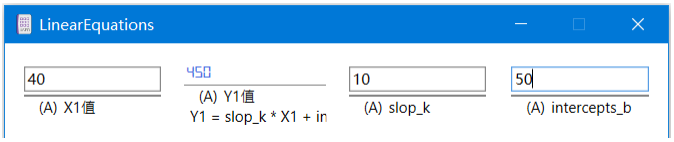
运行以上程序后：将会根据用户给定的参数值计算结果；

当然用户也可以将常量的值和 UserParam 或者 GenData 元素关联，此时可以实时改变对应 UserParam 或者 GenData 元素的值来改变该常量的值：

例如上例中：

```
15  #define slop_k AUserParam[7]   //将slop_k和AUserParam[7]关联
16  #define intercepts_b AUserParam[8]   //将intercepts_b和AUserParam[8]关联
```

运行结果：



（可参阅 PCSuite UserPanels 功能）

当然，User-Program 中也支持 16 进制数的赋值及运算，其结果以 10 进制表示：0x<HEX_value>，例如：AGenData[502] = 0xA+0xB，其计算结果为 AGenData[502]=21；

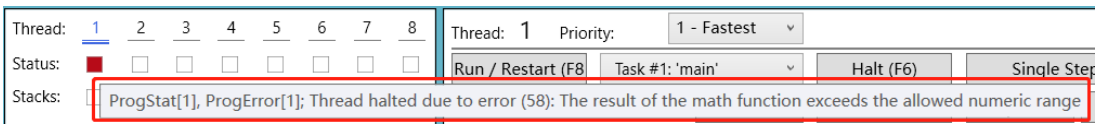
如以下示例

例如，以下程序将会出现报错：

- 数据超出范围（运行报错）：

```
21  main([10,30],[5,20],[800,1000])
22  //
23      variable_a=power(2,31)      //将2^31的值赋给变量variable_a
24
25  endofmain
```

报错信息



- 不支持的数据类型（编译报错）：

```
21  main([10,30],[5,20],[800,1000])
22  //
23      variable_b=0.5
24
25  endofmain
```

3.4 数组（Array）

Agito 控制器内存中分配了一些通用 1 维数组，包括 GenData[*]和 UserParam[*]等数组，由于控制器型号和固件版本不同数组的大小可能存在差异，用户可以在 User-Program 中直接对这些数组进行赋值、运算等操作，值得注意的是 GenData[*]为通用数组，使用某些功能（如 ECAM）会占用这些数组的内容。

在 User-Program 中也允许自定义数组，支持 1 维、2 维、3 维数组：

1 维数组：`#definevar <Array_name>[<Array_size>]`

2 维数组：`#definevar <Array_name>[<X_Array_size>][<Y_Array_size>]`

3 维数组：`#definevar <Array_name>[<X_Array_size>][<Y_Array_size>][<Z_Array_size>]`

值得注意的是，无论是 1 维、2 维或 3 维数组，其总的数组长度不能超过最大限制，如 AGD-301 控制器不超过 192 个，意味着(X_Array_size* Y_Array_size* Z_Array_size) ≤ 192；

变量名允许字母、数字及下划线组合，但必须以字母开头，且长度需要超过 2 个字符；

如以下示例中，将自定义 1 个大小为 10 数组名为 Array_a 的数组 Array_a[10]，并将其值赋给 AGenData[501-510]：

在*.puh2 中定义数组：

```
14
15  #definevar Array_a[10]   //定义数组
16  #definevar iCounter
```

在*.pup2 中定义数组：

```
24      iCounter=1
25      for ( iCounter=1;iCounter<=10 ;iCounter++ )
26          Array_a[iCounter]=power(iCounter,2)      //从iCounter=1开始，将iCounter^2的值赋给数组的每个元素
27          AGenData[500+iCounter]=Array_a[iCounter]//将数组Array_a[1-10]的值赋给AGenData[501-510]
28      end
```

运行结果：

```
----------------------------------------------------------------
AGenData[501-510]
501: 1>; 502: 4>; 503: 9>; 504: 16>; 505: 25>; 506: 36>; 507: 49>; 508: 64>; 509: 81>; 510: 100>
----------------------------------------------------------------
```

用户也可以将变量关联 GenData[*]或 UserParam[*]某个元素：

```
19  #define MyVal_1 AGenData[500]    //将MyVal_1变量和GenData[500]关联
20
21  #definevar MyVal_2{502}    //将MyVal_2变量和GenData[502]关联
```

在.puh2 中定义：

`#define <MyVal_name> AGenData[<index>]` 或 `#definevar <MyVal_name>{<index>}`

当然也可以 User-Program 中直接将变量赋给 GenData[*]：

```
15  #define variable_c AGenData[501]
```

```
21  main([10,30],[5,20],[800,1000])
22  //
23      AGenData[501]=200
24      AGenData[502]=0
25      AGenData[502]=variable_c
26
27  endofmain
```

运行结果：

```
----------------------------------------------------------------
Agendata[502]
200>
----------------------------------------------------------------
```

3.5 指针（Pointer）

IDE User-Program 中支持类似于 C 中的指针，用户可直接调用变量对应的指针地址。

定义指针格式：

```
#defineptr <Pointer_name> <Pointer_Keyword>
```

获取指针地址：

```
& <Pointer_Keyword>
```

操作对应地址下的值：

```
* <Pointer_Keyword>
```

示例 1：获取指针地址及赋值操作

```
26      AGenData[500] = &AEventTable[1]      //获取AEventTable[1]对应的指针地址，将其存到AGenData[500]中
27      AGenData[501] = *AGenData[500]       //获取AGenData[500]中对应数值地址下的存储的值
```

示例 2：定义一个指向 EventTable 的指针，并对其赋值：

在*.puh2 中定义指针：

```
16  #defineptr EventTable_ptr AEventTable[1]
17  #definevar counter
```

在*pup2 中调用指针：

```
21  main([10,30],[5,20],[800,1000])
22  //
23  // Place here the main code of your program
24  //
25      for(counter=1;counter<=5;counter++)
26          SetEventTable(counter,counter*10)
27      end
28  endofmain
29
30
31  function:SetEventTable(index,val)
32      *(EventTable_ptr+(index-1)*65536)=val    //&AEventTable[1]
33  endoffunc
```

运行结果：

Position Events Table (width in [us])

	Value [user-units]	Select	Corrected Value
[1]	10	1	0
[2]	20	1	0
[3]	30	1	0
[4]	40	1	0
[5]	50	1	0
[6]	0	1	0
[7]	0	1	0
[8]	0	1	0

3.6 判断语句

IDE User-Program 中支持 if、switch 及多层 if、swith 嵌套语句；

- if 语句

  语法格式：

```
if (boolean_expression)
   /* 如果 boolean_expression 为 true 将执行的语句 */
end
```

（实例可参阅本文 4.3 章节）

- if...else 语句

  语法格式：

```
if (boolean_expression)
   /* 如果 boolean_expression 为 true 将执行的语句 */
else
   /* 如果 boolean_expression 为 false 将执行的语句 */
end
```

当然，1 个 if 之后也可以跟从多个 else if 条件：

```
if (boolean_expression 1)
   /* 当 boolean_expression 1 为 true 时执行 */
else if (boolean_expression 2)
   /* 当 boolean_expression 2 为 true 时执行 */
else if (boolean_expression 3)
   /* 当 boolean_expression 3 为 true 时执行 */
else
   /* 当以上条件都不为 true 时执行 */
end
```

一旦某个 else if 满足条件，其他的 else if 或 else 将不会被执行；

（实例可参阅本文 4.5 章节）

- switch 语句

  语法格式：

```
switch(expression)
    case constant-expression
       statement(s)
       break
    case constant-expression
       statement(s)
       break
    /* 可以插入任意数量的 case 语句 */
    default /* 可选的 */
       statement(s)
       break
end
```

（实例可参阅本文 4.8 章节内容）

3.7 循环语句

IDE User-Program 中支持多层循环嵌套语句

- while 循环

  语法格式：

```
while (condition)
   statement(s)
end
```

- for 循环

  语法格式：

```
for (init; condition; increment)
   statement(s)
end
```

（实例可参阅本文 4.6 章节）

3.8 运算符

- 算术运算：

运算符	描述	规则
+	把两个操作数相加	A+B，注意其和勿超出 Int32 数据范围
-	从第一个操作数中减去第二个操作数	A-B，注意其差勿超出 Int32 数据范围
*	把两个操作数相乘	A*B，注意其积勿超出 Int32 数据范围
/	分子除以分母	A/B，分母不允许为 0，得到的商将舍余取整
()	括号运算	先进行括号内的运算
%	取模运算符，整除后的余数	A%B
++	自增运算符，整数值增加 1	A++
--	自减运算符，整数值减少 1	A--

- 基本函数

运算符	描述	规则
power	返回给定数值的乘幂	=power(<value>,<power>)，number 为底数，power 为幂，如 power(2,3)=8
abs	取绝对值	=abs(<value>)，如 abs(-2)=2
aqrt	开平方	=sqrt(<value>)，如 sqrt(9)=3

- 位运算

  位运算通常用于数字信号或系统状态的判断或操作；

运算符	描述	规则
&	按位与	A & B，可参阅本文 4.5 实例
|	按位或	A | B，可参阅本文 4.5 实例
^	异或	A ^ B，如果存在于其中一个操作数中但不同时存在于两个操作数中，二进制异或运算符复制一位到结果中
~	按位取反（包括符号位）	~ B，得到一个有符号二进制数的补码形式
<<	向左移位	A << B，A 的值向左移动 B 位
>>	向右移位	A >> B，A 的值向右移动 B 位

- 关系运算

运算符	描述	规则
==	相等	左右值相等为 true
!=	不等	左右值不等为 true
>	大于	左值大于右值为 true
<	小于	左值小于右值为 true
>=	大于等于	左值大于或等于右值为 true
<=	小于等于	左值小于或等于右值为 true

- 逻辑运算

运算符	描述	规则
&&	逻辑与	如果两个操作数都非零，则条件为 true
||	逻辑或	如果两个操作数中有任意一个非零，则条件为 true
!	逻辑非	用来逆转操作数的逻辑状态，如果条件为 true 则逻辑非运算符将使其为 false

- 赋值运算

运算符	描述	规则
=	简单赋值	C = A + B 将把 A + B 的值赋给 C
+=	加且赋值	C += A 相当于 C = C + A
-=	减且赋值	C -= A 相当于 C = C - A
*=	乘且赋值	C *= A 相当于 C = C * A
/=	除且赋值	C /= A 相当于 C = C / A

3.9 函数（function）

每个 User-Program 包含一个 main 主函数，主函数将会被 CPU 分配到 Task 1 中，不允许被占用，

```
main([10,30],[5,20],[800,1000])
//
// Place here the main code of your program
//
endofmain
```

用户可以在 User-Program 中自定义多个函数，在线程或任务中去调用这些函数；

不带形参:

```
function:<function_name>
   /*function_body*/
endoffunc
```

带形参无返回值:

```
function:<function_name>( param_1, param_2, param_3,...)
   /*function_body*/
endoffunc
```

调用：<function_name>(argum_1, argum_2, argum_3,...)

带形参且有返回值：

```
function:return_1,return_2,... = <function_name>( param_1, param_2, param_3,...)
   ...
   return_1=statement(s)
   return_2=statement(s)
   ...
endoffunc
```

调用：return_1,return_2,... = <function_name>(argum_1, argum_2, argum_3,...)

示例①：三元求和函数（带形参无返回值）

```
main([10,30],[5,20],[800,1000])
//
    Sum(30,50,80)
//
endofmain

function: Sum(X1,X2,X3)
    AGenData[500]=X1+X2+X3
endoffunc
```

运行结果为：AGenData[500]=180；

示例②：返回三元之和与积的函数（带形参且有返回值）

```
main([10,30],[5,20],[800,1000])
    while (1)
        a1=AGenData[501]    //仅用于赋值演示
        b1=AGenData[502]    //仅用于赋值演示
        c1=AGenData[503]    //仅用于赋值演示
        sum1,product1=Sum(a1,b1,c1) //调用function
        AGenData[504]=sum1          //仅用于演示运算结果
        AGenData[505]=product1      //仅用于演示运算结果
    end
endofmain

function: sum,product=Sum(X1,X2,X3)     //定义function
    sum=X1+X2+X3        //第1个返回值
    product=X1*X2*X3    //第2个返回值
    return  //ruturn可省
endoffunc
```

3.10 任务和线程（Task&Thread）

User-Program 支持多任务和多线程，可自定义任务名和分配线程，以及设置任务优先级。

- 定义 Task

```
task:Task_name{<Task#>}
   /*Task_body*/
endoftask
```

- 绑定线程

```
AProgRun[<Thread#>],<Task#>
```

示例：

```
21  main([10,30],[5,20],[800,1000])
22  //
23      AGenData[497]=0
24      AGenData[498]=0
25      AGenData[499]=0
26      AGenData[500]=0
27      AGenData[501]=0
28      AProgRun[5], 2      //将2#_Task绑定到5#_Thread
29      AProgRun[6], 3      //将3#_Task绑定到6#_Thread
30      while (1)
31          X1=AGenData[497]    //将AGenData[497]值赋给变量X1
32          X2=AGenData[498]    //将AGenData[498]值赋给变量X2
33          X3=AGenData[499]    //将AGenData[499]值赋给变量X2
34      end
35
36  //
37  endofmain
38  /*计算X1、X2、X3之和的函数*/
39  function: Sum(X1,X2,X3)
40      AGenData[500]=X1+X2+X3
41  endoffunc
42  /*计算X1、X2、X3之积的函数*/
43  function: Product(X1,X2,X3)
44      AGenData[501]=X1*X2*X3
45  endoffunc
46
47  task: SumX1_X2_X3{2}    //将SumX1_X2_X3任务绑定到2#_Task
48      while (1)
49      Sum(X1,X2,X3)
50      end
51
52  endoftask
53
54  task: ProductX1_X2_X3{3}    //将ProductX1_X2_X3任务绑定到3#_Task
55      while (1)
56      Product(X1,X2,X3)
57      end
58  endoftask
```

以上程序将根据 X1、X2、X3 的值实时计算其和与积并输出到 AGenData[500]和 AGenData-[501]中；

- 优先级

IDE User-program 支持用户定义线程和任务的优先级，共有 10 个优先级可供选择，按顺序从 1~10 优先级分别由高到低，默认所有线程优先级都是“1-Fastest”，当用户需要定义优先级时可以按照以下格式定义和调用：

定义线程优先级：AProgPriority[<Thread#>] = <Priority#>

定义和调用任务：runtask(<Name>, <Thread#>, <Priority#>)

- 重启复位线程

当 User-Program 运行过程中出现报错等异常时，可以重启复位线程；

①停止所有线程：

```
23      AProgHaltAll
```

复位需要分别给每个线程发送 AProgReset[<Thread#>]；

②重启用户指定线程

```
44  AProgRun[2], 2      //Run thread_2
45  AProgRun[3], 3      //Run thread_3
46  AProgRun[4], 4      //Run thread_4
47
48  while (1)
49      if  ((AProgError[2]!=0) && (AGenData[808]==2))   //AGenData[808]=2, reset thread_2#
50              AProgHalt[2]
51              AProgReset[2]
52              AWaitTime, 1000
53              AProgRun[2],2
54              AGenData[808]=0
55          end
56      if ((AProgError[3]!=0) && (AGenData[808]==3))   //AGenData[808]=3, reset thread_3#
57              AProgHalt[3]
58              AProgReset[3]
59              AWaitTime, 1000
60              AProgRun[3],3
61              AGenData[808]=0
62          end
63      if ((AProgError[4]!=0) && (AGenData[808]==4))   //AGenData[808]=4, reset thread_4#
64              AProgHalt[4]
65              AProgReset[4]
66              AWaitTime, 1000
67              AProgRun[4],4
68              AGenData[808]=0
69          end
70  end
```

4 User-Program 范例

以下通过一系列常用的 IDE User-Program 实例帮助用户快速掌握 PCSuite IDE 编程环境的使用，使用户可以根据项目需求来编写复杂的用户程序。

4.1 PTP 运动

- 绝对目标位置运动

```
main([10,30],[5,20],[800,1000])
    AAccel=100000    //定义A轴加速度为100000;
    ADecel=100000    //定义A轴减速度为100000;
    ASpeed=1000      //定义A轴速度为1000;
    ARelTrgt=0       //定义A轴相对目标位置为0;
    AAbsTrgt=2000    //定义A轴绝对目标位置为2000;
    AMotorOn=1       //电机上使能;
    AMotionMode=1    //定义A轴运动模式为PTP模式;
    ABegin           //开始执行运动;

    while (AInTargetStat!=4)     //判断运动到位;
    end

endofmain
```

- 相对目标位置运动

```
main([10,30],[5,20],[800,1000])
    AAccel=100000    //定义A轴加速度为100000;
    ADecel=100000    //定义A轴减速度为100000;
    ASpeed=1000      //定义A轴速度为1000;
    ARelTrgt=3000       //定义A轴相对目标位置为3000;
    AAbsTrgt=0    //定义A轴绝对目标位置为0;
    AMotorOn=1       //电机上使能;
    AMotionMode=1    //定义A轴运动模式为PTP模式;
    ABegin           //开始执行运动;

    while (AInTargetStat!=4)     //判断运动到位;
    end

endofmain
```

4.2 Jog 运动

```
main([10,30],[5,20],[800,1000])
    AAccel=100000    //定义A轴加速度为100000;
    ADecel=100000    //定义A轴减速度为100000;
    ASpeed=1000      //定义A轴速度为1000;
    AMotorOn=1       //电机上使能;
    AMotionMode=0    //定义A轴运动模式为Jog模式;
    ABegin           //开始执行运动;
endofmain
```

4.3 发送命令换相

```
main([10,30],[5,20],[800,1000])
//
    //如果上电开始执行换相，请在换相前延时5s左右等待控制器初始化完后再执行换相
    AWaitTime, 5000

    if (AComtStatus[1]!=100)    //判断A轴是否已换相完成，如未弯沉则执行换相
        AComtMode[5]=1282       //执行换相指令
        AWaitTime, 2000         //延时等待
        while (AComtStatus[1]==1)   //换相过程中驻留
        end
        AWaitTime, 500
        if (AComtStatus[1]==-3)     //如换相失败则重新再换相一次
            AComtMode[5]=1282
            while (AComtStatus[1]!=100)
            end
        end
    end
    AWaitTime, 1000

    if (BComtStatus[1]!=100)    //判断B轴是否已换相完成，如未弯沉则执行换相
        BComtMode[5]=1282       //执行换相指令
        AWaitTime, 2000         //延时等待
        while (BComtStatus[1]==1)   //换相过程中驻留
        end
        AWaitTime, 500
        if (BComtStatus[1]==-3)     //如换相失败则重新再换相一次
            BComtMode[5]=1282
            while (BComtStatus[1]!=100)
            end
        end
    end
    AWaitTime, 1000

    while ((AComtStatus[1]+BComtStatus[1])!=200)    //判断AB轴换相完成
    end
//
endofmain

```

4.4 回零

```
main([10,30],[5,20],[800,1000])

    /*调用A轴回零参数定义函数，当然也可以不使用函数，直接将所有参数放置于此*/
    AHomingParaDef()

    AHomingOn=1     //给A轴发送回零命令
    while (AHomingStat!=100)    //回零过程驻留
    end

endofmain

function: AHomingParaDef()      //定义A轴回零参数
    AHomingDef[1]=8
    AHomingDef[2]=1
    AHomingDef[3]=16384
```

```
36      AHomingDef[4]=0
37      AHomingDef[5]=0
38      AHomingDef[6]=0
39      AHomingDef[7]=0
40      AHomingDef[8]=0
41      AHomingDef[9]=0
42      AHomingDef[10]=0
43      AHomingDef[11]=14
44      AHomingDef[12]=1
45      AHomingDef[13]=31
46      AHomingDef[14]=3277
47      AHomingDef[15]=0
48      AHomingDef[16]=0
49      AHomingDef[17]=0
50      AHomingDef[18]=0
51      AHomingDef[19]=0
52      AHomingDef[20]=0
53      AHomingDef[21]=15
54      AHomingDef[22]=200000
55      AHomingDef[23]=2000000
56      AHomingDef[24]=2000000
57      AHomingDef[25]=983040
58      AHomingDef[26]=0
59      AHomingDef[27]=0
60      AHomingDef[28]=0
61      AHomingDef[29]=0
62      AHomingDef[30]=0
63      AHomingDef[31]=16
64      AHomingDef[32]=200000
65      AHomingDef[33]=2000000
66      AHomingDef[34]=2000000
67      AHomingDef[35]=81920
68      AHomingDef[36]=0
69      AHomingDef[37]=0
70      AHomingDef[38]=0
71      AHomingDef[39]=0
72      AHomingDef[40]=0
73      AHomingDef[41]=6
74      AHomingDef[42]=0
75      AHomingDef[43]=16384
76      AHomingDef[44]=0
77      AHomingDef[45]=0
78      AHomingDef[46]=0
79      AHomingDef[47]=0
80      AHomingDef[48]=0
81      AHomingDef[49]=0
82      AHomingDef[50]=0
83      AHomingDef[51]=0
84      AHomingDef[52]=0
85      AHomingDef[53]=0
86      AHomingDef[54]=0
87      AHomingDef[55]=0
88      AHomingDef[56]=0
89      AHomingDef[57]=0
90      AHomingDef[58]=0
91      AHomingDef[59]=0
92      AHomingDef[60]=0
93  endoffunc
```

4.5 数字 I/O 处理

Agito 控制器的数字信号是从 bit_0 开始递增，即 Digital_input_1/ Digital_output_1 对应 bit_0.

```
//////数字输出操作-方法1////////////////////////////////////////////////////////////////////////////

    ADOutPort = ADOutPort | (1<< DOutPort_bit )      //将数字输出某位[DOutPort_bit]置ON
    ADOutPort = ADOutPort&(~(1<< DOutPort_bit ))     //将数字输出某位[DOutPort_bit]置OFF

//////数字输出操作-方法2////////////////////////////////////////////////////////////////////////////

    ADOutPortSBit[DOutPort_num]                      //将数字输出某位[DOutPort_num]置ON，num从1开始
    ADOutPortCBit[DOutPort_num]                      //将数字输出某位[DOutPort_num]置OFF，num从1开始

////////////////////////////////////////////////////////////////////////////////////////////////////

    if (((ADInPort>> DInPort_bit )&1)==1)            //判断数字输入某位[DInPort_bit]状态是否为ON
        /* 当数字输入某位[DInPort_bit]为1时执行 */
    else if (((ADInPort>> DInPort_bit )&1)==0)       //判断数字输入某位[DInPort_bit]状态是否为OFF
        /* 当数字输入某位[DInPort_bit]为0时执行 */
    end

//////数字输出状态翻转操作-方法1////////////////////////////////////////////////////////////////////

    ADOutPort = ADOutPort ^ (1 << DOutPort_bit)      //将数字输出某位[DOutPort_bit]输出状态翻转

//////数字输出状态翻转操作-方法2////////////////////////////////////////////////////////////////////

    ADOutPortTBit[DOutPort_num]                      //将数字输出某位[DOutPort_num]输出状态翻转

////////////////////////////////////////////////////////////////////////////////////////////////////

    ADInLog = ADInLog | (1<< ADInLog_bit )           //将数字输入某位[ADInLog_bit]逻辑取反

////////////////////////////////////////////////////////////////////////////////////////////////////

    ADOutLog = ADOutLog | (1<< ADOutLog_bit )        //将数字输出某位[ADOutLog_bit]逻辑取反

////////////////////////////////////////////////////////////////////////////////////////////////////
```

4.6 误差补偿激光程序

LaserCalibration.pup2 主程序 LaserCalibration.puh2 变量

d:\ide程序示例\lasercalibration\lasercalibration.pup2

```
16  #definevar Rept_Times
17  #definevar Motion_Times
```

```
20  //
21  main([10,30],[5,20],[800,1000])
22      AGenData[500]=0
23      AHomingOn=1
24      while (AHomingStat!=100)
25      end
26      AMotorOn=1
27      ARelTrgt=0
28      AAbsTrgt=0
29      ASpeed=10000    //激光标定时运动速度;
30      AAccel=100000   //激光标定时加速度;
31      ADecel=100000   //激光标定时减速度;
32      ABegin
33      while (AGenData[500]==0)    //驻留指令，当给AGenData[500]赋值1时，程序继续执行;
34      end
35      for ( Rept_Times=1; Rept_Times<=3;Rept_Times++ )    //总的运动次数;
36          for ( Motion_Times=1; Motion_Times<=12; Motion_Times++)     //正向运动点数;
37              ARelTrgt=10000      //正向运动距离间隔;
38              ABegin      //开始运动;
39              AWaitStatus[7],0    //等待运动结束;
40              AWaitTime, 3000     //运动结束等待3s;
41          end
42          AWaitTime, 3000     //正向全行程运动结束等待3s;
43          for ( Motion_Times=1; Motion_Times<=12; Motion_Times++ )    //反向运动点数;
44              ARelTrgt=-10000     //反向运动距离间隔;
45              ABegin      //开始运动;
46              AWaitStatus[7],0    //等待运动结束;
47              AWaitTime, 3000     //运动结束等待3s;
48          end
49          AWaitTime, 3000     //反向全行程运动结束等待3s;
50      end
51      AGenData[500]=0
52      //
53  endofmain
```

4.7 多线程调用

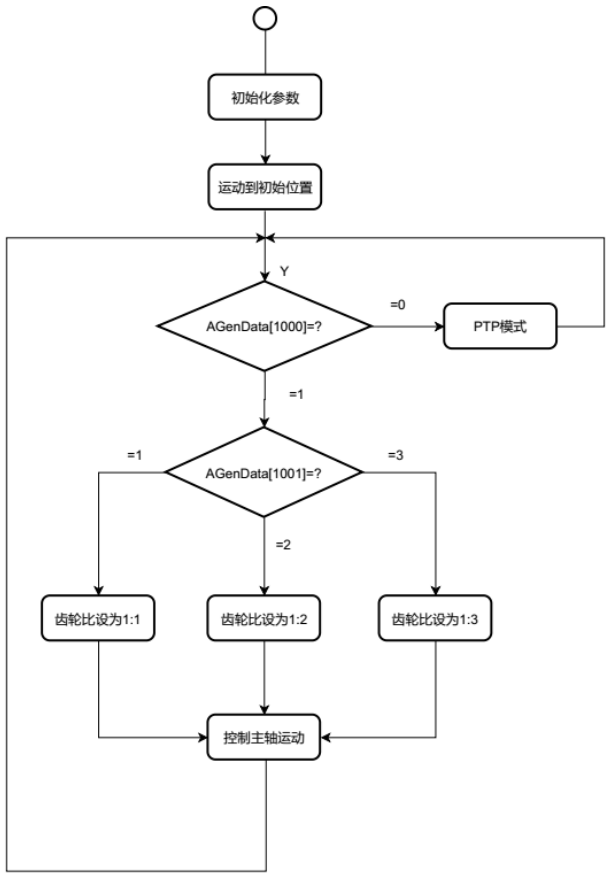
```
21  main([10,30],[5,20],[800,1000])
22
23      AProgRun[5],2   //将2#_task绑定thread_5
24          AWaitTime, 500
25      AProgRun[6],3   //将3#_task绑定thread_6
26          AWaitTime, 500
27      AProgRun[7],4   //将4#_task绑定thread_7
28          AWaitTime, 500
29
30  endofmain
31
32  task: A_AxisMotion{2}   //将第1个任务绑定到2#_task，注意1#_task为主线程保留task，不允许自定义使用
33      AMotorOn=1
34  endoftask
35
36  task: B_AxisMotion{3}   //将第2个任务绑定3#_task
37      BMotorOn=1
38  endoftask
39
40  task: AGenData_Def{4}   //将第3个任务绑定4#_task
41      AGenData[1000]=100
42  endoftask
```

4.8 Gear 模式齿轮比切换

```
#definevar GearTemp                    //在.puh2 中定义一个变量，用于存放当前齿轮比系数
```

```
//////////////////////////////////////////////////
    //初始化变量
    AGenData[1000]=0
    AGenData[1001]=1
    AGenData[1002]=0
    GearTemp=0

    //运动到起始位置
    AMotionMode=1
    BMotionMode=1
    ARelTrgt=0
    AAbsTrgt=1000
    ASpeed=5000
    AAccel=50000
    ADecel=50000
    BRelTrgt=0
    BAbsTrgt=2000
    BSpeed=5000
    BAccel=50000
    BDecel=50000
    AMotorOn=1
    BMotorOn=1
    ABegin
    BBegin

    //判断A、B轴运动到位
    while ((AInTargetStat!=4) || (BInTargetStat!=4))
    end

    while ((AConFlt==0) && (BConFlt==0))
        if (AGenData[1000]==0)
            if (BMotionMode==5)
                BStop
                while (BMotionStat!=0)
                end
                GearTemp=0
                BMotionMode=1
            end
        else if (AGenData[1000]==1)
            if (GearTemp!=AGenData[1001])
                switch (AGenData[1001])
                    case 1
                        BStop
                        while (BMotionStat!=0)
                        end
                        BMotionMode=5
                        BMasterFact=65536
                        GearTemp=AGenData[1001]
                        BBegin
                        break

                    case 2
                        BStop
                        while (BMotionStat!=0)
                        end
                        BMotionMode=5
                        BMasterFact=131072
                        GearTemp=AGenData[1001]
                        BBegin
                        break
                    case 3
                        BStop
                        while (BMotionStat!=0)
                        end
                        BMotionMode=5
                        BMasterFact=196608
                        GearTemp=AGenData[1001]
                        BBegin
                        break
                    end
                end
            end
    end
//////////////////////////////////////////////////////////
```

程序逻辑框图：



4.9 Event 事件函数

在某些情况下，需要在特定的用户程序函数上执行控制器内部或外部时间（数字信号变化、内部状态变化、传感器值等）。

用户可以将 User-Program 和用于预定义的事件函数进行链接，此时当事件触发后，用户程序执行将跳转（类似于中断）到此事件函数，并在该事件函数结束时返回到事件触发之前的位置。

Event 函数只能在 User-Program 主线程（即线程 1），因此用户在使用 Event 函数时，需要保证主线程一直在运行（如示例中第 43~44 行的 while(1)无限循环）。一旦 User-Program 跳转到 Event 函数，所有其他线程将继续像之前一样运行，但是用户可以在事件函数中包含停止其他线程或修改其优先级的命令。

User-Program 最大支持 5 个 Event 函数（由于产品型号和固件版本可能存在差异），其中 Event #1 为快速事件，#2~#5 为普通事件，用户可完全自定义每个 Event 的特征，包括使用哪个控制器参数（ProgEventPar）、掩码（ProgEvent-Mask）、触发类型（ProgEventType）以及触发值（ProgEventVal）。

以下实例实现在无限运动（ModRev）模式下实现连续开启 PEG 功能，其使用 Event 函数使其每次越过重新计数位置（零位置）后重新打开 PEG 功能，实现在开启 ModRev 模式下实现 PEG 信号的连续输出。

```
21  main([10,30],[5,20],[800,1000])
22
23      AProgEventGEn = 0
24      AProgEventOn = 0            /*to be sure it is disabled while configuring.This is the master on/off of the
25                                  evenets detection. If it is 0, teh controller do nothing for events*/
26
27      AProgEventMask[1] = -1      /*AND mask before checking the trigger. -1 means 0xFFFFFFFF, so like no mask.
28                                  Mask is used for looking at specific bits, like given input bit. */
29
30      AProgEventType[1] = 5       /*Type of trigger, just like in data recording,5 is rising edge*/
31
32      AProgEventVal[1] = 100      //The value of the trigger. Must be rising edge througth value of 100
33
34      AProgEventPar[1] = &APos    /*This is a 'pointer' to any variable (parameter/keyword) of the controller.
35                                     Just any of them. Here it points to APos (position of A axis)*/
36      AProgEventEn[1] = 1         // Enable event #1 (specific enable).
37
38      AProgEventGEn = 1           /*This is teh main events of the interrupts (maine nabling of looking for
39                                   events flags at the user program engine). Here: enabled.*/
40      AProgEventOn = 1            /*Enable the feature. The controller will start looking fr events (in the interrupt it will
41                                   look fr event occournces, and also when running user program, it will look for event flags)*/
42
43      while (1)                   //  A loop of doing almost nothing, and waiting for events
44      end
45
46  endofmain
```

```
47
48
49  function: MyEventFunc1() [Event#1]
50      if (AEventOn==0)
51          AEventOn=1
52      end
53
54  endoffunc
```

4.10 正弦运动

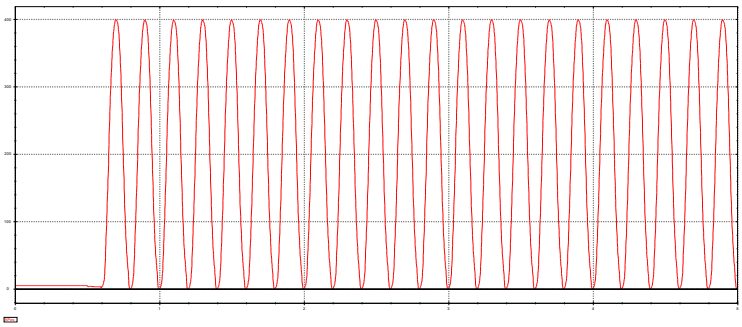
Sine Wave.pup2 主程序 | Sine Wave.puh2 | Sine Wave.puh2 变量

d:\ide程序示例\sine wave\sine wave\sine wave.pup2

```
main([10,30],[5,20],[800,1000])
//
    ARptWait = 0                                // Don't wait between point to point motion
    AJerk = 0
    AmpIn = 200
    FreqIn = 5                                  // No "smoothing" of profile needed
//
    while(1)
        if(SineEnable==1)                        //If UpdateVal=1, enable sine wave
            AStop
            AWaitTime, 500                      // Allow time to stop, wait 500[ms]

            AAccel = 32*AmpIn*FreqIn*FreqIn     // Calculated needed acceleration for quasi-sinusoidal motion
            ADecel = AAccel                     // Calculated needed deceleration for quasi-sinusoidal motion
            ASpeed = 8*AmpIn*FreqIn             // Calculated needed speed for quasi-sinusoidal motion
            AMotionMode = 1                     // Non-repetitive point to point motion
            AAbsTrgt = 0                        // Go to predefined 0
            AMotorOn = 1
            ABegin
            while (AInTargetStat != 4)          // Wait until motor has reached its target
                AWaitTime, 100
            end
            AMotionMode = 2                     // Repetitive point to point motion
            AAbsTrgt = 2*AmpIn                  // Full range is twice the required amplitude
            ABegin
            SineEnable = 0
        else if (SineEnable==2)                  //If UpdateVal=2, disable sine wave
            AStop
            SineEnable = 0
        end

    end
//
endofmain
```

d:\ide程序示例\sine wave\sine wave\sine wave.puh2

```
//
// Sine Wave.puh2
// -------------------------------------
//
// Main header file. Includes definitions for the program.
//
// Created at: 07/29/2020 09:54:50
//
// Place here your constant and variables definitions.
//
#define MY_EXAMPLE1 10000
//

#definevar  SineEnable{797}
#definevar  FreqIn{798}
#definevar  AmpIn{799}
```



4.11 典型力控

- 闭环力控

```
CGoToPosMode //切换到位置模式，防止悬空状态下力控模式使能造成撞击
while (COperationMode==4) //确保切换到位置模式
    CGoToPosMode
    CWaitTime, 500
end
```

1.确保在使能前切换为位置模式

```
    CMotionMode = 1 // 切换到PTP运动模式
    CRelTrgt = 0
    CAbsTrgt = 3800//设置一个一定会碰到受压物体的绝对目标位置
    CSpeed= 20000//设置高速接近时的速度
    CAccel = 2000000//设置高速接近时的加速度
    CDecel = 2000000//设置高速接近时的减速度
    CJerk = 0

    CMotorOn =1//电机上使能

    CSpeedChgNew=500  //设置切换为慢速后的速度
    CSpeedChgPos=3400 //设置切换到慢速的位置
    CSpeedChgDir=0
    CSpeedChgOn=1     //打开切换慢速功能的开关
    CBegin            //开始PTP运动
```

2.设置位置模式下高速与慢速接近的相关参数

```
    CCurrPosTh=3500   //设置转换力控的位置条件阈值
    CCurrPosThDir=1
    CForceAInTh = 5   //设置转换力控的力反馈阈值

    CForceCmdSrc=1    //设置力控指令模式为"Scheduled Force Command Values"模式

    CForceGain=500000  //设置闭环力控的PID参数，下同
    CForceKi=5
    CForceKd=0
    CForceFFW=0
    CForceVelFFW=0
```

3.设置切换闭环力控的条件，以及力闭环的PID参数

```
    CForceCmdVal[1]=150       //设置第一段力控指令为150force-unit
    CForceCmdHTime[1]= 500    //设置第一段力控指令持续时间为500毫秒
    CForceCmdSlope[1] = 1000  //设置第一段力控指令的斜率为1000force-unit/s
    CForceCmdVal[2]=170       //设置第二段力控指令为170force-unit
    CForceCmdHTime[2]= 500    //设置第二段力控指令持续时间为500毫秒
    CForceCmdSlope[2] = 1000  //设置第二段力控指令的斜率为1000force-unit/s
    CForceCmdVal[3]=120       //设置第三段力控指令为120force-unit
    CForceCmdHTime[3]= 500    //设置第三段力控指令持续时间为500毫秒
    CForceCmdSlope[3] = 1000  //设置第三段力控指令的斜率为1000force-unit/s
    CForceCmdHTime[4]=0       //设置第四段力控指令持续时间为0，即只执行三段力控指令
```

4.设置闭环力控的指令参数

```
    CBeginOnToPos=1           //打开从力控模式切换回位置模式后的回退功能
    CRetractTarget=0          //设置回退的目标位置点
    CRetractSpeed=20000       //设置回退的速度
```

5.设置力控指令结束后的回退运动

- 开环力控

```
CGoToPosMode //切换到位置模式，防止悬空状态下力控模式使能造成撞击
while (COperationMode==1) //确保切换到位置模式
    CGoToPosMode
    CWaitTime, 500
end
```

1.确保在使能前切换为位置模式

```
    CMotionMode = 1 // 切换到PTP运动模式
    CRelTrgt = 0
    CAbsTrgt = 3800//设置一个一定会碰到受压物体的绝对目标位置
    CSpeed= 20000//设置高速接近时的速度
    CAccel = 2000000//设置高速接近时的加速度
    CDecel = 2000000//设置高速接近时的减速度
    CJerk = 0

    CMotorOn =1//电机上使能

    CSpeedChgNew=500  //设置切换为慢速后的速度
    CSpeedChgPos=3400 //设置切换到慢速的位置
    CSpeedChgDir=0
    CSpeedChgOn=1     //打开切换慢速功能的开关
    CBegin            //开始PTP运动
```

2.设置快速和慢速接近受压物体的运动参数

```
    CCurrPosTh=3500   //设置转换力控的位置条件阈值
    CCurrPosThDir=1
    CCurrCurrTh = 120   //设置转换力控的电流指令阈值
    CCurrCurrThDir=0

    CCurrCmdSrc=1     //设置力控指令模式为"Scheduled Current Command Values"模式

    CCurrGain=770  //设置开环力控的PID参数，下同
    CCurrKi=358
```

3.设置切换开环力控条件以及
开环力控的PI参数设置

```
    CCurrCmdVal[1]=150        //设置第一段力控指令为150mA
    CCurrCmdHTime[1]= 500     //设置第一段力指令持续时间为500毫秒
    CCurrCmdSlope[1] = 1000   //设置第一段力指令的斜率为1000mA/s
    CCurrCmdVal[2]=170        //设置第二段力控指令为170mA
    CCurrCmdHTime[2]= 500     //设置第二段力指令持续时间为500毫秒
    CCurrCmdSlope[2] = 1000   //设置第二段力指令的斜率为1000mA/s
    CCurrCmdVal[3]=120        //设置第三段力控指令为120mA
    CCurrCmdHTime[3]= 500     //设置第三段力指令持续时间为500毫秒
    CCurrCmdSlope[3] = 1000   //设置第三段力指令的斜率为1000mA
    CCurrCmdHTime[4]=0        //设置第四段力指令持续时间为0，即只执行三段力控指令
```

4.设置开环力控指令参数

```
    CBeginOnToPos=1           //打开从力控模式切换回位置模式后的回退功能
    CRetractTarget=0          //设置回退的目标位置点
    CRetractSpeed=20000       //设置回退的速度
```

5.设置从开环力控回到位置模式后的回退指令

5 常用关键字介绍

以下仅对部分常用关键字作介绍，其他详细关键字使用文档请参阅《Akribis-Agito Controller Keywords Reference》文档。

语法格式中 Axis_index 表示轴号，如 A 轴是 A、B 轴是 B，依次类推。

5.1 换相相关

- **ComtMode**

ComtMode[]是一个用于控制换相过程的数组。

关键字	ComtMode
类型	参数
语法格式	读：Axis_index+ComtMode[Array_index]； 写：Axis_index+ComtMode[Array_index]=Value；
值类型	int
访问	读/写
是否允许运动时写入	否
是否允许使能时写入	否
是否保存到 flash	是
数组范围	1:24
是否轴相关	是
常用指令	ComtMode[5]=1282，重新开始进行换相操作

- **ComtStatus**

ComtStatus[]用于返回换相状态。

关键字	ComtStatus
类型	参数
语法格式	读：Axis_index+ComtStatus[Array_index]；
值类型	int
访问	只读
是否允许运动时写入	是
是否允许使能时写入	是
是否保存到 flash	是
数组范围	1:2
是否轴相关	是
值含义	Axis_index ComtStatus[1]=100，表示换相成功。

5.2 回零相关

回零相关参数主要由以下关键字定义：

- **HomingOn**

上电时或复位后 HomingOn 值将会被置 0，当用户将其置 1 时，控制器将会按照所定义的回零顺序执行回零操作。 在回零结束后（无论成功还是报错失败）， HomingOn 值都将重新被置 0.

关键字	HomingOn
类型	参数
语法格式	读：Axis_index+HomingOn； 写：Axis_index+HomingOn=Value (Value=0、1)
值范围	Int：0~1
访问	读/写
是否允许运动时写入	否
是否允许使能时写入	是
是否保存到 flash	否
是否轴相关	是
值含义	Axis_index HomingOn=1，开始进行回零程序。

- **HomingDef**

HomingDef 用于定义回零过程参数，包括步骤和类型等参数，一般该参数通过 Pcsuite 回零工具导出（详细内容请参阅《Agito 回零使用手册》）。

关键字	HomingDef
类型	参数
语法格式	读：Axis_index+HomingDef[Array_index]； 写：Axis_index+HomingDef[Array_index]=Value
数组范围	[1:150]
访问	读/写
是否允许运动时写入	是
是否允许使能时写入	是
是否保存到 flash	是
是否轴相关	是
值含义	HomingDef[1]定义步骤类型，HomingDef[2-10]定义第 1 步参数； 类似的 HomingDef[21-30]定义第 2 步，依次类推；

- **HomingStat**

HomingStat 用于描述回零状态。

关键字	HomingStat	
类型	参数	
语法格式	读：Axis_index+HomingStat；	
访问	只读	
是否允许运动时写入	是	
是否允许使能时写入	是	
是否保存到 flash	否	
是否轴相关	是	
值含义	值	含义
	0	上电或重启后未执行回零。
	正值 (除 100 以外)	回零进行中，HomingStat 值表示当前回零进行的步骤。
	-1	由于 HomingDef 参数错误导致的回零失

		败，在回零开始前每个步骤都会检查每步相关参数。
	-2	由于超时导致的回零失败，每个步骤都定义了执行延时时间，若该时间内未完成该步骤将会报错。
	-3	由于回零过程中电机意外下使能导致回零失败。
	-4	由于运动错误导致的回零失败，如未定义运动结束方式。
	-5	由于定义了不可识别的步骤类型导致的回零失败，如 HomingDef[1]=50。
	-6	由于检测到在步骤切换时电机运动状态为运动中导致的回零失败。
	-7	有错误的步骤导致的回零失败，意味着回零程序的最后一步不是“0-End Homing”，比如最后一步 HomingDef[61]=1 是错误的。
	-8	由于未知的步骤类型导致的回零失败。
	100	回零完成并成功。

5.3 运动相关

- **Abort**

将速度置 0 立即停止运动（电机保持使能状态）。

关键字	Abort
类型	命令
语法格式	Axis_index+Abort；
访问	只读
是否允许运动时写入	是
是否允许使能时写入	是
是否保存到 flash	否
是否轴相关	是

- **AbsTrgt**

AbsTrgt 用于设置 PTP(点到点运动)和往复运动的绝对目标位置。该目标位置和当前位置无关，AbsTrgt 定义运动停止的位置。对于 PTP 运动，该位置是电机运动停止的位置；对于往复 PTP 运动，AbsTrgt 位置将会是 PTP 运动的其中一目标点，而把当前位置作为往复运动另一目标点。如果 RelTrgt(相对目标位置)不为 0，则忽略 AbsTrgt，并使用 RelTrgt 来实际执行下一个目标动作，AbsTrgt 可以在运动过程中被修改并且立即执行。

关键字	AbsTrgt
类型	变量
语法格式	读：Axis_index+AbsTrgt；

值范围	-2,147,483,648~2,147,483,647
访问	读/写
是否允许运动时写入	是
是否允许使能时写入	是
是否保存到 flash	否
是否轴相关	是

示例:

如当前位置 Pos=1000,AbsTrgt=5000,RelTrgt=0,则下一个 PTP 运动将会停止在 Pos=5000 位置处。

如当前位置 Pos=1000，AbsTrgt=5000,RelTrgt=7000,则下一个 PTP 运动将会停止在 Pos=8000 位置处(目标位置=Pos(当前)+RelTrgt)。

如当前位置 Pos=1000, AbsTrgt = 5000 and RelTrgt = 0,则下一个往复 PTP 运动将会先运动到 Pos=5000 位置处，然后再运动到 Pos=1000 处，重复进行上述运动。

另请参阅：RelTrgt

- **Accel**

Accel 在所有运动中使用 user units/sec^2 来表示加速度，Accel 的值可以在运动过程中被改变，并且会被立即执行，如当前正在做加速运动，加速度会被当前指令改变，如果此时已达目标速度，则新的加速度将会在下一次加速运动中执行。

关键字	Accel
类型	变量
语法格式	读：Axis_index+Accel；
值范围	100~2,000,000,000
访问	读/写
是否允许运动时写入	是
是否允许使能时写入	是
是否保存到 flash	是
是否轴相关	是

另请参阅：Vel, Speed, Decel, MaxAcc.

- **Begin**

Begin 命令用于在设置完所有运动参数（如运动模式、加减速度、速度等）后开始运动，此命令必须在电机使能之后才能生效，否则将会报错。将开始的运动类型由 MotionMode 定义，发送“Begin”命令之后，可以实时更改运动参数（速度、加减速等）并立即生效。

关键字	Begin
类型	命令
语法格式	Axis_index+Begin；
访问	只读
是否允许运动时写入	否
是否允许使能时写入	是
是否保存到 flash	否
是否轴相关	是

- **BeginOnToPos**

BeginOnToPos 命令用于在运动过程中将控制模式更改为位置模式，主要应用于力控。

关键字	BeginOnToPos
类型	参数
语法格式	读：Axis_index+BeginOnToPos； 写：Axis_index+BeginOnToPos=Value，Value=0,1；
值范围	0,1
默认值	0
访问	读/写
是否允许运动时写入	是
是否允许使能时写入	是
是否保存到 flash	否
是否轴相关	是

- **Decel**

Decel 用于定义所有运动的减速度，单位为 user units/sec^2。Decel 可以在运动过程中进行更改，更新后的值将会立即生效。如当前运动在减速过程中，则减速率将会改变。

Stop 指令以 Decel 定义值减速停止，急停使用 EmrgDec 定义减速度停止。

关键字	Decel
类型	参数
语法格式	读：Axis_index+Decel； 写：Axis_index+Decel=Value；
值范围	100~2,000,000,000
默认值	100,000
访问	读/写
是否允许运动时写入	是
是否允许使能时写入	是
是否保存到 flash	否
是否轴相关	是

- **AuxPos**

AuxPos 反馈辅编码器读数。

关键字	AuxPos
类型	参数
语法格式	读：Axis_index+AuxPos； 写：Axis_index+AuxPos=Value；
值（Value）范围	-2,147,483,648~2,147,483,647
访问	读/写
是否允许运动时写入	否
是否允许使能时写入	否
是否保存到 flash	否
是否轴相关	是

- **CurrRef**

CurrRef 返回控制器回路电流（指令电流），单位 mA。

关键字	CurrRef
类型	参数
语法格式	读：Axis_index+CurrRef；
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **IndexPos**

IndexPos 保持检测到的最后一个 index 信号位置（主编码器）。

关键字	IndexPos
类型	参数
语法格式	读：Axis_index+IndexPos；
值范围	-2,147,483,648~2,147,483,647
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **InTargetStat**

InTargetStat 指示运动的进程，直到到达目标为止。当以 user-units 为单位表示的位置误差在到位时间（InTargetTime）内小于最大允许到位差（InTargetTol）时，则 InTargetStat 将会返回到达目标位置。

关键字	InTargetStat	
类型	参数	
语法格式	读：Axis_index+InTargetStat；	
默认值	0	
访问	只读	
是否允许运动时读取	是	
是否允许使能时读取	是	
是否保存到 flash	否	
是否轴相关	是	
值含义	值	含义
	0	下使能
	1	上使能
	2	运动中
	3	等待 InTargetTime 结束
	4	到达目标位置

- **InTargetTime**

InTargetTime 指示电机应在所允许的误差范围内到达目标位置的时间（以 ms 为单位）。如位置误差在 InTargetTime 时间内小于 InTargetTol，则 InTargetStat 返回目标到达。

关键字	InTargetTime
类型	参数
语法格式	读：Axis_index+InTargetTime；
值范围	0~1,000
默认值	3
访问	读/写
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **Jerk**

Jerk 定义了基于轨迹规划的运动中建立所需加减速度所需的时间，

建立加减速度的时间计算如下：

Jerk Time= SAMPLE_TIME *2^{Jerk} ， SAMPLE_TIME 为采样时间，Agito 控制器采样时间为 1/16384=61us.

在 PCSuite 界面以 JerkTime 显示，

注意 Jerk 不允许在运动中更改，显然 Jerk 值越大，运动时间越长，但是运动将会更平稳，通常过冲也会减小，应根据具体应用调整以获得最佳性能。

关键字	Jerk
类型	参数
语法格式	读：Axis_index+Jerk；
值范围	0~9
默认值	0
访问	读/写
是否允许运动时写入	否
是否允许使能时读写	是
是否保存到 flash	否
是否轴相关	是

- **ModRev**

ModRev 用于定义控制器反馈的模量，当该值为 0 时表示关闭该功能，当值设为 M 时表示编码器值将只会在[0，M)Counts 之间范围内变化，如超过 M 时将从 0Counts 处重新计数。

这允许旋转轴在同一方向无限运动（[0，M]在[RevPLim，FwdPLim]范围内）。

注意，不要将 ModeRev 和 Shaping 功能同时使用。

如果使用 SetPosition 手动将位置设置为超出 ModRev 范围外的值，则可能意外发生。

无限运动（ModRev 不为 0）可以和平滑（Jerk 不为 0）同时使用，但用户必须要正确设置 ModRev 和 Jerk 值，即在最大速度下，移动 ModRev 距离的时间应该大于 JerkTime，

例如：如果最大速度是 10,000,000 [counts/sec]，Jerk = 9 (即2^9 = 512 samples)，控制器的采样率为 16,384 samples/sec，则 ModRev 必须大于 10,000,000/16,384*512=312,500 [counts]。

与 Jerk($2^{Jerk[samples]}$)和 ModRev ([counts])采样数的乘积必须小于（2^{31}-1），

例如 Jerk=9，则 ModRev 必须小于$\frac{2^{31}-1}{2^9}$≈4,194,304 [counts]

关键字	ModRev
类型	参数
语法格式	读：Axis_index+ModRev； 写：Axis_index+ModRev=Value；
值范围	0~2,000,000,000
默认值	0
访问	读/写
是否允许运动时写入	否
是否允许使能时写入	否
是否保存到 flash	是
是否轴相关	是

- **MotionMode**

MotionMode 用于定义下一次“Begin”命令之后将执行的运动类型，注意在当前运动结束前，无法更改 MotionMode.

关键字	MotionMode	
类型	参数	
语法格式	读：Axis_index+MotionMode； 写：Axis_index+MotionMode=Value；	
值范围	0~18	
默认值	0	
访问	读/写	
是否允许运动时写入	否	
是否允许使能时写入	是	
是否保存到 flash	是	
是否轴相关	是	
值含义	值	含义
	0	Jog（点动）
	1	PTP（点到点）
	2	Point to point repetitive（往复 PTP）
	3	Pulse Direction direct mode（直接脉冲方向）
	4	Pulse Direction indirect mode（非直接脉冲方向）
	5	Gear direct motion（直接齿轮模式）
	6	Gear indirect motion（间接齿轮模式）

	7	ECAM direct motion
	8	ECAM indirect motion
	9	FIFO motion
	10	Slave position reference
	11	CNC group A motion
	12	Joystick direct motion, input analog in signal representing position reference
	13	Joystick indirect motion, input analog in signal representing position reference, and this profiler can be setting by user
	14	Joystick direct motion, input analog in signal representing velocity reference
	15	Joystick indirect motion, input analog in signal representing velocity reference ,and this profiler can be setting by user
	16	Vector motion
	17	CNC group B motion
	18	Spline buffer motion

- **MotionReason**

MotionReason 返回值用于指示导致最后一次运动结束的原因；

“Begin”指令会将 MotionReason 值重置为 0.

关键字	MotionReason	
类型	参数	
语法格式	读：Axis_index+MotionReason；	
值范围	0~11	
默认值	0	
访问	只读	
是否允许运动时读取	是	
是否允许使能时读取	是	
是否保存到 flash	否	
是否轴相关	是	
值含义	值	含义
	0	当前运动还未结束或运动正常
	1	Stop 命令停止
	2	Abort 命令停止
	3	StopRep 命令停止
	4	检测到负限位开关停止
	5	检测到正限位开关停止
	6	负软限位停止

	7	正软限位停止
	8	电机下使能停止
	9	StopECAM 命令停止
	10	StopFIFO 命令停止
	11	检测到 index 停止（仅 Jogging 模式）

- **MotionStat**

MotionStat 返回当前运动状态，MotionStat 中的每个位表示一种运动状态，在某些情况下，可能会处于多个运动状态，MotionStat=0（即每位都为 0）为运动停止。MotionStat 值为十进制，可以转换成二进制数或者按位与（&）来根据每一个 bit 指示当前运动状态。

12	11	10	9	8	7	6	5	4	3	2	1	0	Bit

关键字	MotionStat	
类型	参数	
语法格式	读：Axis_index+MotionStat；	
默认值	0	
访问	只读	
是否允许运动时读取	是	
是否允许使能时读取	是	
是否保存到 flash	否	
是否轴相关	是	
值含义	Bit	含义
	0	运动中
	1	往复运动等待中
	2	往复运动停止（可参阅 StopRep 命令）
	3	收到 Stop 命令
	4	加速运动中
	5	减速运动中
	6	等待平滑结束
	7	正在停止 ECAM 运动
	8	正在停止 FIFO
	9	等待输入（运动暂停直到用户定义输入的上升沿）
	10	CNC group A 关联轴
	11	CNC group A 运动中
	12	CNC 运动停止中

- **MotorOn**

MotorOn 指令用于控制电机上下使能，电机下使能时，电源不会加到电机上，也无法进行运动控制。因为故障（请参阅 ConFlt），驱动器可能会在内部下使能，重新上使能时，ConFlt 会被清除。如果导致下使能的故障未清除，电机将仍保持下使能状态。

某些命令只能在下使能时使用。

关键字	MotorOn	
类型	参数	
语法格式	读：Axis_index+MotorOn； 写：Axis_index+MotorOn=Value；	
值范围	0、1	
默认值	0	
访问	读/写	
是否允许运动时读取	是	
是否允许使能时读取	是	
是否保存到 flash	否	
是否轴相关	是	
值含义	值	含义
	0	下使能
	1	上使能

- **Pos**

Pos 以 UsrUnits 为返回编码器读数。如 UsrUnits=65536，则 Pos 以 Counts 为单位。

关键字	Pos
类型	参数
语法格式	读：Axis_index+Pos； 写：Axis_index+Pos=Value；
值范围	-2,147,483,648~2,147,483,647
默认值	0
访问	只读
是否允许运动时读取	否
是否允许使能时读取	否
是否保存到 flash	否
是否轴相关	是

- **PosRef**

PosRef 以 UsrUnits 表示用户位置指令。

关键字	PosRef
类型	参数
语法格式	读：Axis_index+PosRef； 写：Axis_index+PosRef=Value；
值范围	-2,147,483,648~2,147,483,647
默认值	0
访问	只读

是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **PosErr**

PosErr 以 UsrUnits 为单位返回位置误差值，是位置指令和实际位置之间的差值。如 PosErr 值超过 MaxPosErr 所定义值，则电机将会下使能。

关键字	PosErr
类型	参数
语法格式	读：Axis_index+PosErr； 写：Axis_index+PosErr=Value；
值范围	-2,147,483,648~2,147,483,647
默认值	0
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **RelTrgt**

RelTrgt 定义了当前位置和目标位置的距离，用于 PTP 运动和往复 PTP 运动。如 RelTrgt 不为 0，则忽略 AbsTrgt，使用 RelTrgt 来确定下一次运动的实际目标。RelTrgt 可以在运动中改变并立即生效。

关键字	RelTrgt
类型	参数
语法格式	读：Axis_index+RelTrgt； 写：Axis_index+RelTrgt=Value；
值范围	-2,147,483,648~2,147,483,647
默认值	0
访问	读/写
是否允许运动时读写	是
是否允许使能时读写	是
是否保存到 flash	否
是否轴相关	是

另请参阅：AbsTrgt

- **Speed**

Speed 用于定义目标速度，单位为 UsrUnits/Sec。在运动中可以实时改变速度。如当前运动处于加速和匀速阶段，将改变速度为更新后的值。

关键字	Speed
类型	参数
语法格式	读：Axis_index+Speed； 写：Axis_index+Speed=Value；

值范围	-60,000,000~60,000,000
默认值	0
访问	读/写
是否允许运动时读写	是
是否允许使能时读写	是
是否保存到 flash	是
是否轴相关	是

- **Vel**

Vel[]数组以三种不同格式返回编码器速度反馈，单位为 UsrUnits / Sec。

Vel[1]是经过滤波器之后的速度值，滤波器参数根据 VelFilt 定义；

Vel[2]为滤波前原始速度值；

Vel[3]为 16 个采样周期的平均值，约 1ms.

关键字	Vel
类型	参数
语法格式	读：Axis_index+Vel[Array_index]；
值范围	-2,147,483,648~2,147,483,647
默认值	0
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **SpeedChgNew**

SpeedChgNew 定义了速度变量，当满足触发条件时，将会把 SpeedChgNew 值赋给 Speed.

关键字	SpeedChgNew
类型	参数
语法格式	读：Axis_index+SpeedChgNew； 写：Axis_index+SpeedChgNew=Value；
值范围	-60,000,000~60,000,000
默认值	0
访问	读/写
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	是
是否轴相关	是

- **Stop**

Stop 命令将按照设定的减速度停止当前运动，如果在没有在运动中，则没有任何效果。

关键字	Stop
类型	命令
语法格式	Axis_index+Stop；

访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

另请参阅：Abort

- **StopCNCA**

StopCNCA 命令将停止当前 A 组 CNC 运动。

关键字	StopCNCA
类型	命令
语法格式	Axis_index+ StopCNCA;
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **StopRep**

StopRep 命令将停止往复运动。如在其他运动过程中发送该指令，则指令将会被忽略。在往复运动过程中发送 StopPep 命令，运动不会立即减速停止，而是在当前运动结束后在下一次运动的起点处停止。如果需要立即停止运动，请使用“Stop”命令。

关键字	StopRep
类型	命令
语法格式	Axis_index+ StopRep;
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

5.4 电机参数相关

- **EncRes**

EncRes 用于定义编码器分辨率。

关键字	EncRes
类型	参数
语法格式	Axis_index+ EncRes;
访问	读/写
是否允许运动时写入	否
是否允许使能时写入	否
是否保存到 flash	是
是否轴相关	是

- **Ia**

Ia 用于返回 A 相电流值，单位 mA。

关键字	Ia
类型	参数
语法格式	Axis_index+ Ia;
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **Ib**

Ib 用于返回 B 相电流值，单位 mA。

关键字	Ib
类型	参数
语法格式	Axis_index+ Ib;
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **MotorCurr**

MotorCurr 是电机的总电流（所有相的总和），单位 mA。

关键字	Ia
类型	参数
语法格式	Axis_index+ Ia;
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **Va**

Va 用于返回施加到 A 相电压千分比

关键字	Va
类型	参数
语法格式	Axis_index+ Va;
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **Vb**

Vb 用于返回施加到 B 相电压千分比。

关键字	Vb
类型	参数
语法格式	Axis_index+ Vb；
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **Vc**

Vc 用于返回施加到 C 相电压千分比。

关键字	Vb
类型	参数
语法格式	Axis_index+ Vc；
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

5.5 I/O 相关

- **AInPort**

AInPort 用于返回各模拟输入通道模拟量值，单位与 AInMode 相关，不同控制器模拟输入通道数量（Port_index）有差异。

关键字	AInPort
类型	参数
语法格式	Axis_index+ AInPort[Port_index]；
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **AOutPort**

AOutPort 用于返回各模拟输出通道模拟量值，与 AOutMode 相关，不同控制器模拟输出通道数量（Port_index）有差异。

关键字	AOutPort
类型	参数
语法格式	读：Axis_index+ AOutPort[Port_index]； 写：Axis_index+ AOutPort[Port_index]=Value；

值范围	-12,000~12,000
访问	读/写
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	是
是否轴相关	是

- **DInPort**

DInPort 返回一个十进制值用于表示数字输入端口状态，不同控制器数字输出通道数量有差异；

当输入端口 Inputs 状态为“On”时，DInPort=$\sum_0^i 2^i$, i=bit，逻辑为“Off”时不参与运算；

DInPort 返回值可以转换成二进制以查看每个单独输入状态；

Bit_0 (LBS)对应 Digital_Input_1，依次类推；

值得注意的是 DInLog 会控制输入的逻辑，

例如：

如果 DInLog=4 (仅将 Digital_Input_3 逻辑取反)，并且输入状态如下：

Digital_Input_1=On

Digital_Input_2=Off

Digital_Input_3=Off

其他输入端口都为 Off

此时 DInPort 在 bit_0 返回“1”，bit_1 返回“0”，bit_2 返回“1”（因为 Digital_Input_3 虽实际为 Off，但因为逻辑取反，所以返回状态“1”），DInPort=$\sum_0^2 2^i$=2^0+2^2=5；

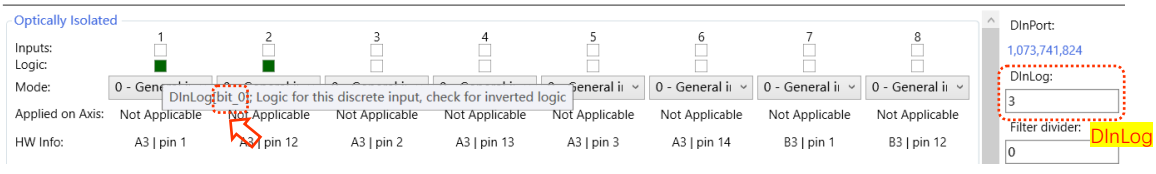
关键字	AOutPort
类型	参数
语法格式	读：Axis_index+ DInPort；
值范围	-12,000~12,000
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **DInLog**

DInLog 用于定义各数字输入通道逻辑。如 DInLog=0，则当输入端口输入为 1 时，DInPort 返回为 1；DInLog=1 时，则当输入端口输入为 1 时，DInPort 返回为 0；

当输入端口 Logic 为“On”时，DInLog=$\sum_0^i 2^i$, i=bit，逻辑为“Off”时不参与运算；

例如仅开启 Digital_Input_1 和 Digital_Input_2，DInLog=2^0+2^1=3.



bit

关键字	AInPort
类型	参数
语法格式	Axis_index+ AInPort[Port_index]；
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **DInPortHigh**

DInPortHigh 返回一个十进制值用于表示高速数字输入端口状态，不同控制器数字输出通道数量有差异；

当输入端口 Inputs 状态为“On”时，DInPortHigh=$\sum_0^i 2^i$, i=bit；逻辑为“Off”时不参与运算；

DInPortHigh 返回值可以转换成二进制以查看每个单独输入状态；

Bit_0 (LBS)对应 DInPortHigh Input_1，依次类推；

值得注意的是 DInLogHigh 会控制输入的逻辑，

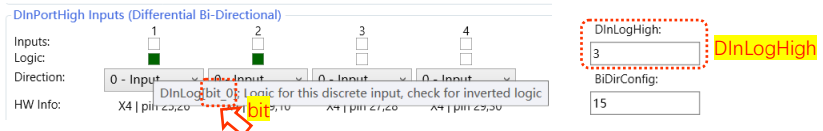
关键字	DInPortHigh
类型	参数
语法格式	读：Axis_index+ DInPortHigh[DInPortHigh_index]；
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **DInLogHigh**

DInLogHigh 用于定义各数字输入通道逻辑。如 DInLogHigh=0，则当输入端口输入为 1 时，DInPort 返回为 1；DInLogHigh=1 时，则当输入端口输入为 1 时，DInPort 返回为 0；

当输入端口 Logic 为“On”时，DInLogHigh=$\sum_0^i 2^i$, i=bit，逻辑为“Off”时不参与运算；

例如仅开启 DInPortHigh Input_1 和 DInPortHigh Input_2，DInLogHigh=2^0+2^1=3.



关键字	AInPort
类型	参数
语法格式	Axis_index+ AInPort[Port_index]；
访问	只读
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	是

- **DOutPort**

DOutPort 返回一个十进制值用于表示数字输出端口状态，不同控制器数字输出通道数量有差异；

当输出端口 Outputs 状态为“On”时，DOutPort=$\sum_0^i 2^i$，逻辑为“Off”时不参与运算；

DOutPort 返回值可以转换成二进制以查看每个单独输入状态；

实际的输出状态还取决于其他相关参数：

DOutPort 的 bit_i 与 DOutLog 的 bit_i 进行异或（XOR）运算，i=0,1,2...；

若 DOutMode[Port_index]≠0，则与输出相关的输出状态将不受 DOutMode 对应位的值影响，输出将仅反映特殊功能。

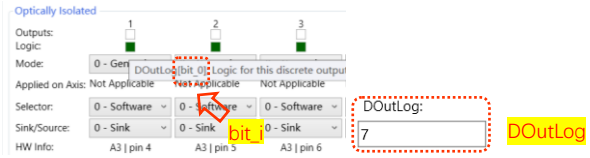
关键字	AOutPort
类型	参数
语法格式	读：Axis_index+ DOutPort； 写：Axis_index+ DOutPort=Value；
访问	读/写
是否允许运动时读写	是
是否允许使能时读写	是
是否保存到 flash	是
是否轴相关	是

- **DOutLog**

DOutLog 用于定义数字输出通道逻辑。如 DOutLog=0，则当输入端口输出为 1 时，DOutPort 返回为 1；DOutLog=1 时，则当输入端口输入为 1 时，DInPort 返回为 0；

当 DOutLog≠0 (即开启)时，DOutLog=$\sum_0^i 2^i$, i=bit，

例如仅将 Digital_Output_1、Digital_Output_2、Digital_Output_3 的 Logic 置 1，其他均置 0 时，DOutLog=$2^0+2^1+2^2$=7.



关键字	AInPort
类型	参数
语法格式	读：Axis_index+ DOutLog； 写：Axis_index+ DOutLog=Value
访问	读/写
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	是
是否轴相关	是

5.6 系统相关

- **WaitStatus**

WaitStatus 是 UserProgram 低层关键字，与“Compare”相关的语法通常由 PCSuite 在编译过程中自动生成。等待直到满足所定义的状态，用户程序才会继续执行。

值得注意的是该关键字仅允许在 IDE 程序中使用。

<table>
<tr><td>关键字</td><td colspan="2">WaitStatus</td></tr>
<tr><td>类型</td><td colspan="2">命令</td></tr>
<tr><td>语法格式</td><td colspan="2">Axis_index+ WaitStatus[Status type], status value；</td></tr>
<tr><td>访问</td><td colspan="2">读/写</td></tr>
<tr><td>是否允许运动时读取</td><td colspan="2">是</td></tr>
<tr><td>是否允许使能时读取</td><td colspan="2">是</td></tr>
<tr><td>是否保存到 flash</td><td colspan="2">否</td></tr>
<tr><td>是否轴相关</td><td colspan="2">是</td></tr>
<tr><td>值含义</td><td colspan="2"><table>
<tr><th>Status type</th><th>含义</th></tr>
<tr><td>1</td><td>COUNTERDOWN1: Wait until CounterDown[1] reaches the value</td></tr>
<tr><td>2</td><td>COUNTERDOWN2: Wait until CounterDown[2] reaches the value</td></tr>
<tr><td>3</td><td>COUNTERDOWN3: Wait until CounterDown[3] reaches the value</td></tr>
<tr><td>4</td><td>COUNTERDOWN4: Wait until CounterDown[4] reaches the value</td></tr>
<tr><td>5</td><td>COUNTERUP1: Wait until CounterUp[1] reaches the value</td></tr>
<tr><td>6</td><td>COUNTERUP2: Wait until CounterUp[2] reaches the value</td></tr>
<tr><td>7</td><td>IN_MOTION: Wait until MotionStat in motion bit reaches the value</td></tr>
<tr><td>8</td><td>IN_RPT_WAIT: Wait until MotionStat in repeat wait bit reaches the value</td></tr>
<tr><td>9</td><td>IN_RPT_STOP: Wait until MotionStat stop bit reaches the value</td></tr>
<tr><td>10</td><td>IN_STOP_REQUEST: Wait until MotionStat stop request bit reaches the value</td></tr>
<tr><td>11</td><td>IN_ACCELERATION: Wait until MotionStat in acceleration bit reaches the value</td></tr>
<tr><td>12</td><td>IN_DECELERATION: Wait until MotionStat in deceleration bit reaches the value</td></tr>
<tr><td>13</td><td>IN_WAIT_END_SMOOTH: Wait until MotionStat smoothing ended bit reaches the value</td></tr>
<tr><td>14</td><td>IN_ECAM_STOP: Wait until MotionStat in ECAM stop bit reaches the value</td></tr>
<tr><td>15</td><td>IN_FIFO_STOP: Wait until MotionStat in FIFO stop bit reaches the value</td></tr>
<tr><td>16</td><td>COMMUTATION: Wait until StatReg commutation bit reaches the value</td></tr>
</table></td></tr>
</table>

17	IN_TARGET: Wait until StatReg in target bit reaches the value
18	REC_TRIG_DETECTED_STATUS: Wait until RecStat trigger detected reaches the value
19	REC_COMPLETE_STATUS: Wait until RecStat recording complete reaches the value
20	DIN1: Wait until digital input 1 reaches the value
21	DIN2: Wait until digital input 2 reaches the value
22	DIN3: Wait until digital input 3 reaches the value
23	DIN4: Wait until digital input 4 reaches the value
24	DIN5: Wait until digital input 5 reaches the value
25	DIN6: Wait until digital input 6 reaches the value
26	DIN7: Wait until digital input 7 reaches the value
27	DIN8: Wait until digital input 8 reaches the value
28	DIN9: Wait until digital input 9 reaches the value
29	DIN10: Wait until digital input 10 reaches the value
30	DIN11: Wait until digital input 11 reaches the value
31	DIN12: Wait until digital input 12 reaches the value
32	DIN13: Wait until digital input 13 reaches the value
33	DIN14: Wait until digital input 14 reaches the value
34	MOTOR_ON_INPUT_STATUS: Wait until the input designated as motor on reaches the value
35	VEL_GAIN_CHANGE_STATUS: Wait until the input designated as gain change reaches the value
36	CLEAR_ABS_ENC_STATUS: Wait until the input designated as clear absolute encoder on reaches the value
37	CLEAR_IN_PULSES_STATUS: Wait until the input designated clear input pulses on reaches the value
38	REV_LIMIT_STATUS: Wait until the input designated as reverse limit reaches the value
39	FWD_LIMIT_STATUS: Wait until the input designated as forward limit reaches the value
40	TORQUE_LIMIT_ON_STATUS: Wait until the input designated as torque limit reaches the value
41	ALARM_RESET_STATUS: Wait until the input designated as alarm reset reaches the value
42	ABORT_INPUT_STATUS: Wait until the input designated as abort reaches the value
43	MODE_SWITCH_VEL_POS_STATUS: Wait until the input designated as vel / pos switch reaches the value
44	MODE_SWITCH_VEL_CUR_STATUS: Wait until the input designated as vel/cur switch reaches the value

	45	MODE_SWITCH_POS_CUR_STATUS: Wait until the input designated as pos/cur switch reaches the value
	46	ADD_FILTER_STATUS: Wait until the input designated as add filter reaches the value

- **GenData**

GenData 是分配给用户的通用数组，在这个数组中，用户可以存储整型数据，通常用于存储 ECAM 等表格。

关键字	GenData
类型	参数
语法格式	读：Axis_index+ GenData[Array_index]； 写：Axis_index+ GenData[Array_index]=Value
访问	读/写
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	是
是否轴相关	否

- **WaitTime**

WaitTime 用于定义继续执行下一行语句前的等待时间，以 ms 为单位。

值得注意的是该关键字仅允许在 IDE 编程环境中使用。

关键字	WaitTime
类型	参数
语法格式	写：Axis_index+ WaitTime，Value；
访问	读/写
是否允许运动时读取	是
是否允许使能时读取	是
是否保存到 flash	否
是否轴相关	否