目录

1 IMC 函数库使用说明 3
1.1 说明 3
2 设备函数 4
2.1 网络设备 4
2.1.1 IMC_FindNetCard 4
2.1.2 IMC_Open 4
2.1.3 IMC_OpenUsePassword 6
2.1.4 IMC_Close 7
3 基本函数 8
3.1 写参数函数 8
3.1.1 IMC_SetMulParam 8
3.1.2 IMC_SetParam 9
3.2 位操作函数 10
3.2.1 IMC_SetParamBit 10
3.2.2 IMC_ORXORParam 10
3.2.3 IMC_TurnParamBit 11
3.3 读参数函数 11
3.3.1 IMC_GetParam 11
3.3.2 IMC_GetParamBit 13
3.3.3 IMC_GetMulParam 13
3.4 等待函数 14
3.4.1 IMC_WaitTime 14
3.4.2 IMC_WaitParamBit 15
3.4.3 IMC_WaitParam 15
3.4.4 IMC_WaitParamMask 16
3.5 插补函数 16
3.5.1 IMC_AddLineNWithVel 16
3.5.2 IMC_AddLineN 17
3.5.3 IMC_AddArcLineWithVel 17
3.5.4 IMC_AddArcLine 18
3.5.5 IMC_AddArcWithVel 18
3.5.6 IMC_AddArc 18
3.5.7 IMC_AddApiralWithVel 19
3.5.8 IMC_AddSpiral 19
3.6 送入轮廓运动轨迹数据 20
3.7 事件函数 21
3.7.1 IMC_InstallEvent 21
3.8 IMC_ClearFIFO 29
4 功能函数 31

4.1 常用点到点运动函数 31
4.1.1 移动到绝对位置 31
4.1.2 移动相对距离 31
4.1.3 叠加运动 32
4.1.4 移相运动 32
4.2 连续运动模式 33
4.2.1 连续速度运动 33
4.2.2 点动（JOG） 33
4.3 搜寻原点（homeing） 34
4.3.1 仅使用原点开关 34
4.3.2 使用原点开关及编码器索引信号 36
4.3.3 仅使用编码器索引信号 38
4.4 常用电子齿轮运动 39
4.4.1 自由啮合过程的电子齿轮运动 39
4.4.2 啮合过程与位移同步的电子齿轮运动 39
4.4.3 啮合过程与时间同步的电子齿轮运动 40
4.5 电子手轮运动 42
4.5.1 手轮运动 42
4.6 同步运动 43
4.6.1 时间-位移（T-P）同步运动 43
4.6.2 位移-位移（P-P）运动 43
4.6.3 时间-速度运动（T-V） 44
4.6.4 位移-速度运动（P-V） 44
5 iMC 参数地址宏定义 45
5.1 主坐标系（MCS）点到点参数 45
5.2 辅坐标系（PCS）点到点参数 46
5.3 主编码器参数 46
5.4 辅编码器参数 46
5.5 其它轴参数 46
5.6 电子齿轮参数 47
5.7 环形轴参数 47
5.8 非线性同步控制参数 47
5.9 运动状态参数 48
5.10 输入输出参数 48
5.11 轮廓运动参数 48
5.12 预定义用户参数 49
5.13 iMC 基本信息参数 50
6 附录 类型定义 51

1 IMC 函数库使用说明

1.1 说明

用户可以使用函数库开发专用的运动控制应用程序。为方便用户使用，我们共提供两类函数：

(1) 基本函数。基本函数是实现直接读写 iMC 参数的函数。

(2)功能函数。功能函数是面向运动控制功能而封装的函数。这些函数实际上是对若干基本函数的封装。

由于功能函数面向功能封装，易于理解，然而功能函数有限，对于功能函数未涵盖的功能，用户可以使用基本函数实现。

以上函数可以混合使用。

此说明文档对应的函数库版本为 V3.2。

函数库中的函数是线程安全的，可在多线程中调用。不是进程安全的，不能在不同的进程中同时调用。

2 设备函数

设备函数是用于对设备进行操作的函数，包括搜索设备、打开设备、初始化设备以及关闭设备等。

2.1 网络设备

2.1.1 IMC_FindNetCard

【功能】此函数用于查找以太网卡设备。
【定义】

```
IMC_STATUS WINAPI IMC_FindNetCard (NIC_INFO * pinfo, int* num)
```

【参数】
pinfo　指定一个 NIC_INFO 结构变量的地址，用于保存扫描结果。
num　　用于保存搜索到的网卡数量
【返回】成功返回 IMC_OK，失败返回其他错误值。
【说明】NIC_INFO 结构在附录中有定义。当调用此函数后，扫描得到的网卡描述名称保存在中 NIC_INFO 结构的 description 数组成员；使用此数组的索引来连接网卡。

注意：在调用此函数时，不能有设备被打开。

2.1.2 IMC_Open

【功能】此函数用于打开设备。
【定义】

```
IMC_STATUS WINAPI IMC_Open (IMC_HANDLE* Handle, int netcardsel, int
imcid)
```

【参数】
Handle　　指定一个 IMC_HANDLE 类型的指针，用于保存设备句柄；
netcardsel 网卡索引，由搜索网卡函数返回的结果决定；
imcid　　IMC 控制卡的 id，由控制卡上的拨码开关设置决定。
【返回】成功返回 IMC_OK，失败返回其他错误值。
【说明】所有的对设备进行操作的函数在使用前，必须调用一个打开设备函数获得设备句柄，使用此句柄才能与设备进行通信。
【例】

```
IMC_STATUS  Status;
IMC_HANDLE  handle;
NIC_INFO  info;
int num;
```

```
Status= IMC_FindNetCard (&info, &num);
if(Status==IMC_OK)
{
    if(num>0){
        Status=IMC_Open (&handle,0,0);//通过第一块网卡连接 id=0 的控制卡
        if(Status==IMC_OK){
            //打开成功，访问 IMC 卡
        }else{
            //打开失败
        }
    }
}else{
    //其他错误
}
```

2.1.3 IMC_OpenX

【功能】此函数用于打开设备。

【定义】

```
IMC_STATUS WINAPI IMC_OpenX (IMC_HANDLE* Handle, int netcardsel, int
imcid, int timeout, int openMode)
```

【参数】

Handle　　指定一个 IMC_HANDLE 类型的指针，用于保存设备句柄；
netcardsel 网卡索引，由搜索网卡函数返回的结果决定；
imcid　　IMC 控制卡的 id，由控制卡上的拨码开关设置决定；
timeout　　通信超时时间，单位毫秒；
openMode　　打开模式；1：混杂模式，　0：非混杂模式。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】所有的对设备进行操作的函数在使用前，必须调用一个打开设备函数获得设备句柄，使用此句柄才能与设备进行通信。

Timeout 参数最小值为 1，用于设置通信函数等待的超时时间。如果电脑的运算速度慢，建议设置超时时间长些。

openMode 参数，一般情况下使用混杂模式，通信时间会快一些。但混杂模式，对某些无线网卡不支持，只能使用非混杂模式。

当 Timeout=40、openMode=1 时，它和 IMC_Open 函数相同。

【例】

```
IMC_STATUS  Status;
IMC_HANDLE  handle;
NIC_INFO  info;
int num;
Status= IMC_FindNetCard (&info, &num);
if(Status==IMC_OK)
{
```

```
    if(num>0){
        //通过第一块网卡连接 id=0 的控制卡,通信超时 40 毫秒,使用混杂模式
        Status=IMC_OpenX (&handle, 0, 0, 40, 1);
        if(Status==IMC_OK){
            //打开成功，访问 IMC 卡
        }else{
            //打开失败
        }
    }
}else{
    //其他错误
}
```

2.1.4 IMC_OpenUsePassword

【功能】此函数用于打开设备。

【定义】

```
IMC_STATUS  WINAPI  IMC_OpenUsePassword  (IMC_HANDLE*  Handle,  int
netcardsel, int imcid, char *password)
```

【参数】

Handle　　指定一个 IMC_HANDLE 类型的指针，用于保存设备句柄；
netcardsel 网卡索引，由搜索网卡函数返回的结果决定；
imcid　　IMC 控制卡的 id，由控制卡上的拨码开关设置决定；
password　密码字符串。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】此函数与 IMC_Open 类似，只是增加了密码检测功能。

客户可以使用自己设置的密码通过此函数打开设备，即客户编写的程序只适用于自己设置过密码的控制卡，可以使程序与控制卡一一对应。

密码是一个至少 4 个、少于 100 个字符的 ASCII 字符串。出厂的控制卡没有设置密码，要使用密码，请使用 IMCSoft 调试平台中工具菜单下的固件更新功能来设置密码。

如果 password=NULL，则此函数不检测密码是否正确，即与 IMC_Open 相同。

【例】

```
IMC_STATUS  Status;
IMC_HANDLE  handle;
NIC_INFO info;
int num;
char *pw = "12345678";//密码必须先由用户自己通过 IMCSoft 调试平台设置
Status= IMC_FindNetCard (&info, &num);
if(Status==IMC_OK)
{
    if(num>0){
```

```
        Status=IMC_OpenUsePassword(&handle, 0, 0, pw);// 通过第一块网卡
    连接 id=0 的控制卡
        if(Status==IMC_OK){
            //打开成功，访问 IMC 卡
        }else{
            //打开失败
        }
    }
}else{
    //其他错误
}
```

2.1.5 IMC_Close

【功能】此函数用于关闭打开的设备。
【定义】

```
IMC_STATUS  IMC_Close (IMC_HANDLE Handle)
```

【参数】

Handle　　设备句柄。

【返回】成功返回 IMC_OK，失败返回其他错误值。
【说明】当设备关闭后，就无法再与设备进行通信，除非再次打开。
【例】

```
IMC_HANDLE  Handle;
IMC_STATUS  Status;
NIC_INFO info;
int num;
Status= IMC_FindNetCard (&info, &num);
Status = IMC_Open(&Handle, 0, 0);
if (Status == IMC_OK) {
    // 使用句柄访问设备，当结束的时候调用 IMC_Close
    IMC_Close (Handle);
} else {
    // 失败
}
```

3 基本函数

对控制卡的所有操作都是由下面这些基本函数实现的，在后面说到的功能函数其实就是由基本函数组合实现。

iMC 中有若干个先进先出缓存器（FIFO），所有控制指令都发往这些缓存器，iMC 再从这些缓存器中取出指令执行。理解这些缓存器的功能将有助于理解编程原理。

iMC 中按功能分有以下几个 FIFO：

IFIFO（Immediate FIFO）：立即指令 FIFO，写入该 FIFO 的指令会立即执行，一个控制周期至少执行一条指令。一般用于需要立刻执行的指令，例如：写入参数值、载入事件指令。IFIFO 没有等待功能。

QFIFO（Queue FIFO）：指令队列 FIFO，用于写入一连串的指令序列。可执行写参数指令、载入事件指令等。每个控制周期执行 QFIFO 中的一条指令。QFIFO 具有等待功能，若插入等待指令，条件满足后再执行等待指令后面的指令。由于支持等待指令，可使运动控制器按序执行动作。

PFIFO1（Path FIFO1）：插补空间 1 缓存器，用于缓存插补空间 1 的插补运动指令，支持写参数指令、载入事件指令、等待指令、直线段或圆弧段指令。

PFIFO2（Path FIFO2）：插补空间 2 缓存器，用于缓存插补空间 2 的插补运动指令，支持写参数指令、载入事件指令、等待指令、直线段或圆弧段指令。

CFIFO（Contour FIFO）：此 FIFO 为轮廓运动专用，仅在执行轮廓运动时，运动点位的数据发送到此 FIFO。

指令发往哪个FIFO，由函数的最后一个参数决定，即FIFO选择。分别是枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2、SEL_CFIFO。

这几个指令缓存器是互相独立的，主机可以在任何时候往任意一个 FIFO 发送指令。但当函数返回 IMC_FIFO_FULL 时，表明 FIFO 已满，当前发送的指令 IMC 没有接收成功。此时由客户决定是重发还是退出。

3.1 写参数函数

3.1.1 IMC_SetMulParam

【功能】同时设置控制卡芯片中多个参数的值。

【定义】

```
IMC_STATUS IMC_SetMulParam (    IMC_HANDLE Handle, pWR_MUL_DES pdes,
                        int arrNum, int fifoSel);
```

【参数】

Handle	设备句柄；
pdes	WR_MUL_DES 结构体数组变量的首地址；
arrNum	pdes 数组中的元素个数；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、

SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。
【返回】指令发送成功返回 IMC_OK，失败返回其他错误值
【说明】

3.1.2 IMC_SetParam

【功能】设置控制卡芯片中的参数值。
【定义】

```
IMC_STATUS IMC_SetParam16 ( IMC_HANDLE Handle, IMC16 paramloc,
                            IMC16 thedata, IMC32 axis, int fifoSel);
IMC_STATUS IMC_SetParam32 ( IMC_HANDLE Handle, IMC16 paramloc,
                            IMC32 thedata, IMC32 axis, int fifoSel);
IMC_STATUS IMC_SetParam48 ( IMC_HANDLE Handle, IMC16 paramloc,
                            __int64 thedata, IMC32 axis, int fifoSel);
```

【参数】

Handle	设备句柄；
axis	指定控制卡有效的轴号；
paramloc	控制卡芯片中的参数地址；
thedata	写入控制卡中参数地址的参数值；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值
【说明】

调用这 3 个函数中的哪个函数，由参数类型决定，如果参数是 16 位的，则调用 IMC_SetParam16，如果是 32 位的则调用 IMC_SetParam32，是 48 位的就调用 IMC_SetParam48。参数类型请参考 IMC 参数地址宏定义。

注意：此 3 个函数都不会自动检测参数类型。比如一个 16 位的参数使用 IMC_SetParam32，IMC_SetParam32 会设置成功，但会改变与此 16 位参数地址相近的其他参数的值，即会产生不可预料的后果。如一个 32 位的参数使用 IMC_SetParam16，则此 32 位参数只有低 16 位被改变，高 16 位无变化，即不会产生预期的效果。

当函数返回值为 IMC_FIFO_FULL 时，表示 IMC 的指令缓存区已满，IMC 对此指令没有接收成功。此时由客户决定是否要重发此指令。

【例子】
下面的例子中使用的变量

```
IMC_HANDLE Handle; //此句柄由函数 IMC_Open 获取
IMC_STATUS Status;
```

A、通过 IFIFO 设置轴 0 的 mcstgpos 参数为 100：
查看参数类型可知，mcstgpos 参数是 32 位的参数，所以：

```
IMC16 addr=mcstgposLoc; //mcstgpos 参数的地址，在 iMC 参数地址宏定义中
```

```
    IMC32 data=100;
    Status= IMC_SetParam32 (Handle, addr, data, 0, SEL_IFIFO);
    if( Status==IMC_OK ){
        //设置参数值成功
    }else{
        //设置参数值失败
    }
```

B、通过 QFIFO 设置轴 1 的 gohome 参数为-1：

查看参数类型可知，gohome 参数是 16 位的参数，所以：

```
    IMC16  addr=gohomeLoc; //gohome 参数的地址，在 iMC 参数地址宏定义中
    IMC16 data=-1;
    Status= IMC_SetParam16 (Handle, addr, data, 1, SEL_QFIFO);
    if(Status==IMC_OK){
        //设置参数值成功
    }else{
        //设置参数值失败
    }
```

3.2 位操作函数

3.2.1 IMC_SetParamBit

【功能】设置控制卡中位宽为 16 的寄存器的某个位。

【定义】

```
IMC_STATUS IMC_SetParamBit (IMC_HANDLE Handle, IMC16 paramloc, IMC16 bit,
IMC16 val, IMC32 axis,int fifoSel);
```

【参数】

Handle	设备句柄；
axis	指定 IMC 有效的轴号；
paramloc	IMC 的参数地址；
bit	寄存器的第几位，范围 0 - 15；
val	需要设置的值；其值为 0 或 1；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值

【说明】

此函数可将 IMC 中的寄存器的某一位置 1 或置 0，适用于 IO 控制。

3.2.2 IMC_ORXORParam

【功能】设置控制卡中位宽为 16 的寄存器的多个位的值。

【定义】

```
IMC_STATUS IMC_ORXORParam (IMC_HANDLE Handle, IMC16 paramloc, IMC16
```

```
ORdata, IMC16 XORdata, IMC32 axis,int fifoSel);
```

【参数】

Handle	设备句柄；
axis	指定 IMC 有效的轴号；
paramloc	IMC 的参数地址；
ORdata	参与或运算的数值；
XORdata	参与异或运算的数值；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值

【说明】

此函数实现原理是将某个寄存器的原值与 ORdata 数值进行或运算，其结果再与 XORdata 数值进行异或运算，将最后的结果替换此参数的原值。

使用此函数可置位或清零目标寄存器的某些位。常用于对 IO 进行操作。

其中 IMC_SetParamBit 函数是此函数的一种特例。

3.2.3 IMC_TurnParamBit

【功能】将控制卡中位宽为 16 的寄存器的某个位的值翻转。

【定义】

```
IMC_STATUS IMC_TurnParamBit (IMC_HANDLE Handle, IMC16 paramloc, IMC16 bit, IMC32 axis,int fifoSel);
```

【参数】

Handle	设备句柄；
axis	指定 IMC 有效的轴号；
paramloc	IMC 的参数地址；
bit	指定 16 位寄存器的某个位，其值为 0-15；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值

【说明】

3.3 读参数函数

3.3.1 IMC_GetParam

【功能】获取控制卡芯片中参数的值。

【定义】

```
IMC_STATUS IMC_GetParam16 ( IMC_HANDLE Handle, IMC16 paramloc,
                            PVOID pdata, IMC32 axis);
IMC_STATUS IMC_GetParam32 ( IMC_HANDLE Handle, IMC16 paramloc,
```

```
                        PVOID pdata, IMC32 axis);
IMC_STATUS IMC_GetParam48 ( IMC_HANDLE Handle, IMC16 paramloc,
                        PVOID pdata, IMC32 axis);
```

【参数】

Handle	设备句柄；
axis	指定控制板有效的轴号；
paramloc	控制卡芯片中的参数地址；
pdata	数据的缓存区指针，其缓存区用于存放获取到的参数值；

【返回】获取成功返回 IMC_OK，失败返回其他错误值

【说明】

此函数获取到的参数值都是整型类型的。

调用这 3 个函数中的哪个函数，由参数类型决定，如果参数是 16 位的，则调用 IMC_GetParam16，如果是 32 位的则调用 IMC_GetParam32，是 48 位的就调用 IMC_GetParam48。参数类型请参考 IMC 参数地址宏定义。

注意：此 3 个函数都不会自动检测参数类型。比如一个 16 位的参数使用 IMC_GetParam32，IMC_GetParam32 会获取成功，但获取的值只有第 16 位为正确的值，高 16 为错误值。如一个 32 位的参数使用 IMC_SetParam16，则此 32 位参数只有低 16 位被读取。

【例子】

下面例子中使用的变量

```
IMC_HANDLE Handle;//此句柄由函数 IMC_Open 获取
IMC_STATUS Status;
```

A、获取轴 1 的参数 mcspos 的值：

查看参数类型可知，mcspos 参数是 32 位的参数，所以：

```
    IMC16 addr = mcsposLoc; //mcspos 参数的地址，在 iMC 参数地址宏定义中
    IMC32 data;
    Status =IMC_GetParam32 (Handle, addr, (PVIOD)&data, 1)
    if(Status==IMC_OK){
        //获取参数值成功
    }else{
        //获取参数值失败
    }
```

B、获取轴 0 的参数 gohome 的值：

查看参数类型可知，gohome 参数是 16 位的参数，所以：

```
    IMC16 addr = gohomeLoc; //gohome 参数的地址，在 iMC 参数地址宏定义中
    IMC16 data;
    Status =IMC_GetParam16 (Handle, addr, (PVIOD)&data, 0)
    if(Status == IMC_OK){
        //获取参数值成功
    }else{
        //获取参数值失败
    }
```

3.3.2 IMC_GetParamBit

【功能】获取位宽为 16 的 IMC 参数的某一位的值。
【定义】

```
IMC_STATUS IMC_GetParamBit (    IMC_HANDLE Handle, IMC16 paramloc,
                        IMC16* pdata, IMC16 bit, IMC32 axis);
```

【参数】

Handle	设备句柄；
axis	指定控制板有效的轴号；
paramloc	控制卡芯片中的参数地址；
pdata	数据的缓存区指针，其缓存区用于存放获取到的值；
bit	IMC 参数的某一位，范围 0 - 15。

【返回】获取成功返回 IMC_OK，失败返回其他错误值
【说明】
获取 IMC 参数指定的 bit 位的值。注，此 IMC 参数必须是 16 位的。
【例】
获取轴 0 的轴 IO 的第 0 位的值

```
IMC16 val;
IMC_STATUS Status;
Status = IMC_GetPatamBit (Handle, aioLoc, &val, 0, 0);
if(Status != IMC_OK) return;
```

3.3.3 IMC_GetMulParam

【功能】同时获取多个 IMC 参数的值。
【定义】

```
IMC_STATUS IMC_GetMulParam (IMC_HANDLE Handle, pWR_MUL_DES pdes,  WORD
NumofArr);
```

【参数】

Handle	设备句柄；
pdes	WR_MUL_DES 结构数组的首地址；
NumofArr	结构数组的有效元素的个数。范围（1 - 32）

【返回】获取成功返回 IMC_OK，失败返回其他错误值
【说明】
此函数同时获取多个参数的值，常用于读取各轴的状态信息。注意，此函数一次最多可读取 32 个参数的值。
【例】
A、获取 6 个轴的错误寄存器的值

```
WR_MUL_DES des[6];
WORD error[6];
IMC_STATUS Status;
int axis;
for(axis=0; axis<6; axis++)
```

```
    {
        des[axis].addr = errorLoc;
        des[axis].axis = axis;
        des[axis].len = 1;
    }
    Status = IMC_GetMulParam (Handle, des, 6);
    if(Status == IMC_OK){
        for(axis=0; axis<6; axis++)
        {
            error[axis] = des[axis].data[0];
        }
    }
```

B、获取 6 个电机的当前位置

```
    WR_MUL_DES des[6];
    int *ptr, pos[6];
    IMC_STATUS Status;
    int axis;
    for(axis=0; axis<6; axis++)
    {
        des[axis].addr = encpLoc;
        des[axis].axis = axis;
        des[axis].len = 2;
    }
    Status = IMC_GetMulParam (Handle, des, 6);
    if(Status == IMC_OK){
        for(axis=0; axis<6; axis++)
        {
            ptr = (int*)des[axis].data;
            pos[axis] = *ptr;
        }
    }
```

3.4 等待函数

3.4.1 IMC_WaitTime

【功能】发送阻塞指令给控制卡。

【定义】

```
IMC_STATUS IMC_WaitTime (IMC_HANDLE Handle, IMC32 time, int fifoSel);
```

【参数】

Handle　　设备句柄；

time　　阻塞时间，单位为毫秒；其值为 0 或-1 时表示一直阻塞；

fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值。

【说明】

此函数用于阻塞指定的 FIFO 一定的时间，超过此时间后才能继续执行后面的指令。

当函数返回值为IMC_FIFO_FULL时，表示IMC的指令缓存区已满，此指令IMC没有接收成功。此时由客户决定是否要重发此指令。

3.4.2 IMC_WaitParamBit

【功能】发送阻塞指令给控制卡。

【定义】

```
IMC_STATUS IMC_WaitParamBit (IMC_HANDLE Handle, IMC16 paramloc, IMC16 bit, IMC16 val, IMC_32 timeout, IMC32 axis, int fifoSel);
```

【参数】

Handle　　设备句柄；

axis　　指定控制板有效的轴号；

paramloc　　控制卡芯片中的参数地址。

bit　　IMC 参数的某一位，其值范围是 0 - 15

val　　其值为 0 或 1；

time　　阻塞超时时间，单位为毫秒；其值为 0 或-1 时表示一直阻塞；

fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值。

【说明】

此函数阻塞指定的 FIFO，直到 Param 参数的第 bit 位的值为 val，或者时间超过 time 才继续执行后面的指令。

当函数返回值为IMC_FIFO_FULL时，表示IMC的指令缓存区已满，此指令IMC没有接收成功。此时由客户决定是否要重发此指令。

3.4.3 IMC_WaitParam

【功能】发送阻塞指令给控制卡。

【定义】

```
IMC_STATUS IMC_WaitParam (IMC_HANDLE Handle, IMC16 paramloc, IMC16 data, IMC_32 timeout, IMC32 axis, int fifoSel);
```

【参数】

Handle　　设备句柄；

axis　　指定控制板有效的轴号；

paramloc　　控制卡芯片中的参数地址。

data　　参数值；

time　　阻塞超时时间，单位为毫秒；其值为 0 或-1 时表示一直阻塞；

fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的

SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值。

【说明】

此函数阻塞指定的 FIFO，直到指定的参数的值为 data，或者时间超过 time 为止。

当函数返回值为IMC_FIFO_FULL时，表示IMC的指令缓存区已满，此指令IMC没有接收成功。此时由客户决定是否要重发此指令。

3.4.4 IMC_WaitParamMask

【功能】发送阻塞指令给控制卡。

【定义】

```
IMC_STATUS IMC_WaitParamMask (IMC_HANDLE Handle, IMC16 paramloc, IMC16 mask, IMC16 data, IMC_32 timeout, IMC32 axis, int fifoSel);
```

【参数】

Handle	设备句柄；
axis	指定控制板有效的轴号；
paramloc	控制卡芯片中的参数地址；
mask	指定 IMC 参数的屏蔽位；
data	指定的参数值；
time	阻塞超时时间，单位为毫秒；其值为 0 或-1 时表示一直阻塞；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】指令发送成功返回 IMC_OK，失败返回其他错误值。

【说明】

此函数阻塞指定的 FIFO，直到 Param 参数值与 mask 相与后得到的值与 data 相等，或者时间超过 time 为止。

当函数返回值为IMC_FIFO_FULL时，表示IMC的指令缓存区已满，此指令IMC没有接收成功。此时由客户决定是否要重发此指令。

3.5 插补函数

3.5.1 IMC_AddLineNWithVel

```
IMC_STATUS IMC_AddLineNWithVel (IMC_HANDLE Handle, PFIFOSegInfo* pdata, int fifoSel)
```

【功能】增加直线段到 PFIFO 中。

【参数】

Handle:	设备句柄；
pdata	PFIFOSegInfo 类型的数据缓存区地址；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。

【说明】

IMC 在执行完此段直线段之前，PFIFO 中的其他指令是被阻塞的，即参与运动的各轴运动到此段直线的末端之后，IMC 才会执行 PFIFO 中的其他指令。

在使用此函数前，必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。即将轴数 N 写入到 IMC 参数 pathaxisnum1（pathaxisnum2）(N 小于等于控制卡的有效轴数)；将参与运动的轴号写入到 IMC 参数 segmap_x、segmap_y、…等中；例如，使用 PFIFO1，参与运动的轴数为 2，轴号为 0、1，设轴 1 为 X 轴，轴 0 为 Y 轴，则将 2 写入 pathaxisnum1，数值 1 写入到 segmap_x1 参数中，将数值 0 写入到 segmap_y1 参数中。

3.5.2 IMC_AddLineN

IMC_STATUS IMC_AddLineN (IMC_HANDLE Handle, PFIFOSegData* pdata, int fifoSel)

【功能】增加直线段数据到 PFIFO 中。

【参数】

Handle：	设备句柄；
pdata	PFIFOSegData 类型的数据缓存区地址；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。

【说明】

如果增加的直线段的速度不变,可调用此函数直接将段数据增加到PFIFO中。

在使用此函数前，必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。

注意，IMC 在执行完此段直线段之前，PFIFO 中的其他指令是被阻塞的，即参与运动的各轴运动到此段直线的末端之后,IMC 才会执行 PFIFO 中的其他指令。

3.5.3 IMC_AddArcLineWithVel

IMC_STATUS IMC_AddArcLineWithVel (IMC_HANDLE Handle, PFIFOSegInfo* pdata, int fifoSel)

【功能】增加圆弧直线段到 PFIFO 中。

【参数】

Handle：	设备句柄；
pdata	PFIFOSegInfo 类型的数据缓存区地址；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。

【说明】

IMC 在执行完此段直线段之前，PFIFO 中的其他指令是被阻塞的，即参与运动的各轴运动到此段直线的末端之后，IMC 才会执行 PFIFO 中的其他指令。

在使用此函数前，必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。

3.5.4 IMC_AddArcLine

IMC_STATUS IMC_AddArcLine (IMC_HANDLE Handle, PFIFOSegData* pdata, int fifoSel)

【功能】增加圆弧直线段数据到 PFIFO 中。

【参数】

Handle:	设备句柄；
pdata	PFIFOSegData 类型的数据缓存区地址；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。

【说明】

如果增加的直线段的速度不变，可调用此函数直接将段数据增加到 PFIFO 中。

注意，IMC 在执行完此段直线段之前，PFIFO 中的其他指令是被阻塞的，即参与运动的各轴运动到此段直线的末端之后，IMC 才会执行 PFIFO 中的其他指令。

在使用此函数前，必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。

3.5.5 IMC_AddArcWithVel

IMC_STATUS IMC_AddArcWithVel (IMC_HANDLE Handle, IMC32 TgVel, IMC32 EndVel, IMC32 Endx, IMC32 Endy, IMC32 centerX, IMC32 centerY, IMC32 dir, int fifoSel)

【功能】增加圆弧段到 PFIFO 中。

【参数】

Handle	设备句柄；
TgVel	运行速度；
EndVel	终点速度；
Endx	圆弧段终点 x 轴位置；
Endy	圆弧段终点 y 轴位置；
CenterX	圆弧中心 x 坐标；
CenterY	圆弧中心 y 坐标；
dir	圆弧运动方向； 0：逆时针方向； 1：顺时针方向
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。

【说明】

IMC 在执行完此段直线段之前，PFIFO 中的其他指令是被阻塞的，即参与运动的各轴运动到此段直线的末端之后，IMC 才会执行 PFIFO 中的其他指令。

在使用此函数前，必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。

3.5.6 IMC_AddArc

IMC_STATUS IMC_AddArc (IMC_HANDLE Handle, IMC32 Endx, IMC32 Endy, IMC32 centerX, IMC32 centerY, IMC32 dir, int fifoSel)

【功能】增加圆弧段数据到 PFIFO 中。
【参数】
Handle:　设备句柄;
Endx　圆弧段终点 x 轴位置;
Endy　圆弧段终点 y 轴位置;
CenterX　圆弧中心 x 坐标;
CenterY　圆弧中心 y 坐标;
dir　圆弧运动方向;　0：逆时针方向;　1：顺时针方向
fifoSel　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。
【说明】
如果增加的直线段的速度不变,可调用此函数直接将段数据增加到PFIFO中。
注意，IMC 在执行完此段直线段之前，PFIFO 中的其他指令是被阻塞的，即参与运动的各轴运动到此段直线的末端之后,IMC 才会执行 PFIFO 中的其他指令。
在使用此函数前，必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。

3.5.7 IMC_AddApiralWithVel

IMC_STATUS IMC_AddApiralWitchVel (IMC_HANDLE Handle, IMC32 TgVel, IMC32 EndVel, IMC32 Endx, IMC32 Endy, IMC32 EndLine, IMC32 centerX, IMC32 centerY, IMC32 dir, int fifoSel)
【功能】增加螺旋段到 PFIFO 中。
【参数】
Handle:　设备句柄;
TgVel　运行速度;
EndVel　终点速度;
Endx　圆弧段终点 x 轴位置;
Endy　圆弧段终点 y 轴位置;
EndLine　z 轴终点位置;
CenterX　圆弧中心 x 坐标;
CenterY　圆弧中心 y 坐标;
dir　圆弧运动方向;　0：逆时针方向;　1：顺时针方向
fifoSel　选择接收此指令的 FIFO。其值为枚举类型 FIFO_SEL 中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。
【说明】
IMC 在执行完此段直线段之前，PFIFO 中的其他指令是被阻塞的，即参与运动的各轴运动到此段直线的末端之后，IMC 才会执行 PFIFO 中的其他指令。
在使用此函数前，必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。

3.5.8 IMC_AddSpiral

IMC_STATUS　IMC_AddSpiral (IMC_HANDLE Handle, IMC32 Endx, IMC32 Endy,

IMC32 EndLine, IMC32 centerX, IMC32 centerY, IMC32 dir, int fifoSel)
【功能】增加螺旋段数据到 PFIFO 中。
【参数】

Handle:	设备句柄;
TgVel	运行速度;
EndVel	终点速度;
Endx	圆弧段终点 x 轴位置;
Endy	圆弧段终点 y 轴位置;
EndLine	z 轴终点位置;
CenterX	圆弧中心 x 坐标;
CenterY	圆弧中心 y 坐标;
dir	圆弧运动方向; 0: 逆时针方向; 1: 顺时针方向
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的 SEL_PFIFO1、SEL_PFIFO2 中的一个。

【说明】
如果增加的直线段的速度不变,可调用此函数直接将段数据增加到 PFIFO 中。
注意, IMC 在执行完此段直线段之前, PFIFO 中的其他指令是被阻塞的, 即参与运动的各轴运动到此段直线的末端之后,IMC 才会执行 PFIFO 中的其他指令。
在使用此函数前, 必须对相应的 PFIFO 映射轴号、设置路径加速度、设置段的坐标数据以绝对值还是相对值表示、设置进给倍率。

3.6 送入轮廓运动轨迹数据

IMC_STATUS IMC_PutContourData (IMC_HANDLE Handle, IMC16 *pdata,
DWORD nWords, int fifoSel)
【功能】将参与轮廓运动的各轴数据发送到 IMC 控制卡的 CFIFO 中。
【参数】

Handle	设备句柄;
pdata	保存各轴的相对位移数据的缓存区地址;
nWords	pdata 所指向的缓存区中的 short 类型的数据个数。
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_CFIFO 。

【返回】成功返回 IMC_OK, 失败返回其他错误值。
【说明】Pdata 缓存区中的数据是按 X、Y、…轴的顺序存储的, 例如, 若参与轮廓运动的轴数为 3, 假设整个运动轨迹中有 10 个运动点, 则 pdata 存储的数据序列是: X1、Y1、Z1、X2、Y2、Z2、…、X10、Y10、Z10。
其中 Xn、Yn、Zn 表示第 n 个点的三个坐标轴的数据, 函数参数 nWords 表示的是 pdata 缓存区中 short 类型数据的个数,如上例中,10 个点共有 30 个数据, 因此 nWord=30。
Pdata 中的数据, 如果是正数, 则表示正方向移动, 负数表示负方向移动。
注意 1: pdata 所指向的缓存区中每个点的数据都必须完整, 即上例中第 n 个点的数据必须完整包含 Xn、Yn、Zn 三个轴的数据。
注意 2: pdata 中的数据使用的单位是 IMC 基本单位。

注意 3：pdata 中的数据是各轴的相对位移，而不是绝对位置。

注意 4：如果是使用 CFIFO，则 pdata 中的数据表示的相对位移 d 是指在用户自己设置的 IMC 参数 groupsmooth 个周期内设定 IMC 发出的脉冲数，即轮廓运动的速度为 d/groupsmooth，因此要注意各轴数据的大小，否则会出现速度或加速度超限的错误。如果是 PFIFO，则速度由 segtgvel1 或 segtgvel2 设置，加速度由 pathacc1 或 pathacc2 设置。

注意 5：nWords 的值必须小于或等于 126。

3.7 事件函数

3.7.1 IMC_InstallEvent

IMC_STATUS IMC_InstallEvent (IMC_HANDLE Handle, PEventInfo pEvent, IMCU16 EventNum, int fifoSel)

【功能】增加事件指令到事件空间中。

【参数】

Handle	设备句柄；
pEvent	EventInfo 类型数组的首地址；
EventNum	本次载入事件的数量。虽然执行空间最多支持 128 条事件指令，但每次载入应少于 32 条；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【说明】

根据所需要的事件选择事件指令及相应的数据填充 pEvent 结构体数组。

其中 EventInfo 结构体的定义为：

```
typedef struct _EVENT_INFO_ {
 IMC16 EventCMD;  //操作码，即枚举类型 IMC_EVENT_CMD 中的值
 IMC16 EventType; //执行类型，即枚举类型 IMC_EventType 中的值
 IMC16 Src1_loc;  //指向操作数 1 的参数地址
 IMC16 Src1_axis; //操作数 1 所属的轴号
 union{           //用一个联合体来组织操作数 2
    struct {
       IMC16 Src2_loc;  //指向操作数 2 的参数地址
       IMC16 Src2_axis; //操作数 2 所属的轴号
       IMC32 reserve;   //保留
    }param;
    struct {
       IMC16 data;      //16 位宽的常数
       IMC16 reserve1;  //保留
       IMC32 reserve2;  //保留
    }data16;
    struct {
```

```
            IMC32 data;         //32 位宽的常数
            IMC32 reserve;      //保留
        }data32;
        __int64 data48;         //48 位宽的常数
    }Src2;
    IMC16 dest_loc;             //指向存储目标的参数地址
    IMC16 dest_axis;            //目标参数所属的轴号
    IMC32 reserve;              //保留
}EventInfo, *PEventInfo;
```

每个事件不同，所需要填充的结构体成员也是不同的。需要填充哪些成员由 IMC_EVENT_CMD 枚举类型决定。

而 IMC_EVENT_CMD 枚举类型的定义为:

```
enum IMC_EVENT_CMD{
    CMD_ADD32,      //两个 32bit 参数相加
    CMD_ADD32i,     //32bit 参数 加 32bit 立即数
    CMD_ADD48,      //两个 48bit 参数相加
    CMD_ADD48i,     //48bit 参数 加 48bit 立即数
    CMD_CMP32,      //两个 32bit 参数相比较
    CMD_CMP32i,     //32bit 参数 与 32bit 立即数 相比较
    CMD_CMP48,      //两个 48bit 参数相比较
    CMD_CMP48i,     //48bit 参数 与 48bit 立即数 相比较
    CMD_SCA32,      //32bit 参数缩放，倍率(48bit)为另一参数
    CMD_SCA32i,     //32bit 参数缩放，倍率(48bit)为立即数
    CMD_SCA48,      //48bit 参数缩放，倍率(48bit)为另一参数
    CMD_SCA48i,     //48bit 参数缩放，倍率(48bit)为立即数
    CMD_MUL32L,     //32bit 参数 乘以 32bit 参数 其结果取低 32bit
    CMD_MUL32iL,    //32bit 参数 乘以 立即数 其结果取低 32bit
    CMD_MUL32A,     //32bit 参数 乘以 32bit 参数 其结果取低 48bit
    CMD_MUL32iA,    //32bit 参数 乘以 立即数 其结果取低 48bit
    CMD_COP16,      //拷贝 16bit 参数
    CMD_COP32,      //拷贝 32bit 参数
    CMD_COP48,      //拷贝 48bit 参数
    CMD_SET16,      //设置 16bit 参数
    CMD_SET32,      //设置 32bit 参数
    CMD_SET48,      //设置 48bit 参数
    CMD_OR16,       //参数 OR 参数
    CMD_OR16i,      //参数 OR 立即数
    CMD_OR16B,      //参数 OR 参数  其结果转换为 BOOL 类型
    CMD_OR16iB,     //参数 OR 立即数  其结果转换为 BOOL 类型
    CMD_AND16,      //参数 AND 参数
    CMD_AND16i,     //参数 AND 立即数
    CMD_AND16B,     //参数 AND 参数  其结果转换为 BOOL 类型
    CMD_AND16iB,    //参数 AND 立即数  其结果转换为 BOOL 类型
```

```
    CMD_XOR16,      //参数 OR 参数
    CMD_XOR16i,     //参数 OR 立即数
    CMD_XOR16B,     //参数 OR 参数 其结果转换为 BOOL 类型
    CMD_XOR16iB     //参数 OR 立即数 其结果转换为 BOOL 类型
};
```

IMC_EventType 枚举类型的定义为：

```
enum IMC_EventType{
    IMC_Allways,            //“无条件执行”
    IMC_Edge_Zero,          //“边沿型条件执行”——变为 0 时
    IMC_Edge_NotZero,       //“边沿型条件执行”——变为非 0 时
    IMC_Edge_Great,         //“边沿型条件执行”——变为大于时
    IMC_Edge_GreatEqu,      //“边沿型条件执行”——变为大于等于时
    IMC_Edge_Little,        //“边沿型条件执行”——变为小于时
    IMC_Edge_Carry,         //“边沿型条件执行”——变为溢出时
    IMC_Edge_NotCarry,      //“边沿型条件执行”——变为无溢出时
    IMC_IF_Zero,            //“电平型条件执行”——若为 0
    IMC_IF_NotZero,         //“电平型条件执行”——若为非 0
    IMC_IF_Great,           //“电平型条件执行”——若大于
    IMC_IF_GreatEqu,        //“电平型条件执行”——若大于等于
    IMC_IF_Little,          //“电平型条件执行”——若小于
    IMC_IF_Carry,           //“电平型条件执行”——若溢出
    IMC_IF_NotCarry         //“电平型条件执行”——若无溢出
};
```

以下为 IMC 事件根据 IMC_EVENT_CMD 类型而填充 EventInfo 结构变量 Ev 的例子：

```
EventInfo Ev;
```

3.7.1.1 加运算事件

A) ADD32

```
Ev.EventCMD = CMD_ADD32;        //32 位参数与参数相加运算
Ev.EventType = IMC_Allways;     //触发事件类型
Ev.Src1_loc = curposLoc;        //参数 1 地址
Ev.Src1_axis = 0;               //参数 1 轴号
Ev.Src2.param.Src2_loc = curposLoc; //参数 2 地址
Ev.Src2.param.Src2_axis = 1;    //参数 2 轴号
Ev.dest_loc = user32b0Loc;      //不保存结果则使其为 0
Ev.dest_axis = 0;               //轴号
```

B) ADD48

```
Ev.EventCMD = CMD_ADD48;        //48 位参数与参数相加运算
Ev.EventType = IMC_Allways;     //触发事件类型
Ev.Src1_loc = user48b0Loc;      //参数 1 地址
```

```
    Ev.Src1_axis = 0;              //参数 1 轴号
    Ev.Src2.param.Src2_loc = user48b1Loc;   //参数 2 地址
    Ev.Src2.param.Src2_axis = 1; //参数 2 轴号
    Ev.dest_loc = user48b2Loc;    //不保存结果则使其为 0
    Ev.dest_axis = 0;              //轴号
C) ADD32i
    Ev.EventCMD = CMD_ADD32i;     //32 位参数与常数相加运算
    Ev.EventType = IMC_Allways;  //触发事件类型
    Ev.Src1_loc = curposLoc;      //参数 1 地址
    Ev.Src1_axis = 0;              //参数 1 轴号
    Ev.Src2.data32.data = 10000; //常数
    Ev.dest_loc = user32b0Loc;    //不保存结果则使其为 0
    Ev.dest_axis = 0;              //轴号
D) ADD48i
    Ev.EventCMD = CMD_ADD48i;     //48 位参数与常数相加运算
    Ev.EventType = IMC_Allways;  //触发事件类型
    Ev.Src1_loc = user48b1Loc;    //参数 1 地址
    Ev.Src1_axis = 0;              //参数 1 轴号
    Ev.Src2.data48 = 100000000;  //常数
    Ev.dest_loc = user48b2Loc;    //不保存结果则使其为 0
    Ev.dest_axis = 0;              //轴号
```

3.7.1.2 比较运算事件

```
A) CMP32、CMP48
    Ev.EventCMD = CMD_CMP48;      //48 位参数与参数比较运算
    Ev.EventType = IMC_Allways;  //触发事件类型
    Ev.Src1_loc = user48b1Loc;    //参数 1 地址
    Ev.Src1_axis = 0;              //参数 1 轴号
    Ev.Src2.param.Src2_loc = user48b2Loc;   //参数 2 地址
    Ev.Src2.param.Src2_axis = 1; //参数 2 轴号
    Ev.dest_loc = user48b3Loc;    //不保存结果则使其为 0
    Ev.dest_axis = 0;              //轴号
B) CMP32i
    Ev.EventCMD = CMD_CMP32i;     //48 位参数与常数比较运算
    Ev.EventType = IMC_Allways;  //触发事件类型
    Ev.Src1_loc = curposLoc;      //参数 1 地址
    Ev.Src1_axis = 0;              //参数 1 轴号
    Ev.Src2.  data32.data = 10000;//常数
    Ev.dest_loc = user32b1Loc;    //不保存结果则使其为 0
    Ev.dest_axis = 0;              //轴号
```

C) CMP48i

```
Ev.EventCMD = CMD_CMP48i;     //48 位参数与常数比较运算
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user48b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.data48 = 1000000;     //常数
Ev.dest_loc = user48b2Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;
```

3.7.1.3 放大/缩小

A) SCA32

```
Ev.EventCMD = CMD_SCA32;      //32 位参数放大/缩小
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user32b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.param.Src2_loc = user32b2Loc;   //参数 2，其值为倍率
Ev.Src2.param.Src2_axis = 1; //参数 2 轴号
Ev.dest_loc = user32b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

B) SCA48

```
Ev.EventCMD = CMD_SCA48;      //48 位参数放大/缩小
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user48b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.param.Src2_loc = user48b2Loc;   //参数 2 地址，其值为倍率
Ev.Src2.param.Src2_axis = 1; //参数 2 轴号
Ev.dest_loc = user48b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

C) SCA32i

```
Ev.EventCMD = CMD_SCA32i;     //32 位参数与倍率
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user32b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.data32.data = 65536; //倍率
Ev.dest_loc = user32b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

D) SCA48i

```
Ev.EventCMD = CMD_SCA48i;     //48 位参数与倍率
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user48b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
```

```
Ev.Src2.data48 = 655360;      //倍率
Ev.dest_loc = user48b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

3.7.1.4 乘运算事件

A) MUL32L

```
Ev.EventCMD = CMD_MUL32L;     //32 位参数与参数相乘运算
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user32b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.param.Src2_loc = user32b2Loc;   //参数 2 地址
Ev.Src2.param.Src2_axis = 1; //参数 2 轴号
Ev.dest_loc = user32b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

B) MUL32A

```
Ev.EventCMD = CMD_MUL32A;     //32 位参数与参数相乘运算
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user32b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.param.Src2_loc = user32b2Loc;   //参数 2 地址，其值为倍率
Ev.Src2.param.Src2_axis = 1; //参数 2 轴号
Ev.dest_loc = user48b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

C) MUL32iL

```
Ev.EventCMD = CMD_MUL32iL;    //32 位参数与常数相乘运算
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user32b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.data32.data = 22222; //常数
Ev.dest_loc = user32b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

D) MUL32iA

```
Ev.EventCMD = CMD_MUL32iL;    //32 位参数与常数相乘运算
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user32b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.data32.data = 22222; //常数
Ev.dest_loc = user48b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

3.7.1.5 拷贝事件

A) COP16

```
Ev.EventCMD = CMD_COP16;      //拷贝 16 位参数
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src2.param.Src2_loc = user16b1Loc;   //源参数地址
Ev.Src2.param.Src2_axis = 0;            //源参数轴号
Ev.dest_loc = user16b0Loc;   //目标参数地址
Ev.dest_axis = 0;            //目标参数轴号
```

B) COP32

```
Ev.EventCMD = CMD_COP32;      //拷贝 32 位参数
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src2.param.Src2_loc = user32b1Loc;   //源参数地址
Ev.Src2.param.Src2_axis = 0;            //源参数轴号
Ev.dest_loc = user32b0Loc;   //目标参数地址
Ev.dest_axis = 0;            //目标参数轴号
```

C) COP48

```
Ev.EventCMD = COP48;          //拷贝 48 位参数
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src2.param.Src2_loc = user48b1Loc;   //源参数地址
Ev.Src2.param.Src2_axis = 0;            //源参数轴号
Ev.dest_loc = user48b0Loc;   //目标参数地址
Ev.dest_axis = 0;            //目标参数轴号
```

3.7.1.6 Set 事件

A) SET16

```
Ev.EventCMD = CMD_SET16;      //设置 16 位参数值
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src2.data16.data = 12345; //常数
Ev.dest_loc = user16b0Loc;   //目标参数地址
Ev.dest_axis = 0;            //目标参数轴号
```

B) SET32

```
Ev.EventCMD = CMD_SET32;      //设置 32 位参数值
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src2.data32.data = 12345678;//常数
Ev.dest_loc = user32b0Loc;   //目标参数地址
Ev.dest_axis = 0;            //目标参数轴号
```

C) SET48

```
Ev.EventCMD = CMD_SET48;      //设置 48 位参数值
```

```
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src2.data32.data = 12345678000;//常数
Ev.dest_loc = user48b0Loc;   //目标参数地址
Ev.dest_axis = 0;            //目标参数轴号
```

3.7.1.7 AND 事件

A) AND16、AND16B

```
Ev.EventCMD = CMD_AND16;     //16 位参数与上 16 位参数
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src1_loc = user16b1Loc;   //参数 1 地址
Ev.Src1_axis = 0;            //参数 1 轴号
Ev.Src2.param.Src2_loc = user16b2Loc;   //参数 2 地址
Ev.Src2.param.Src2_axis = 0; //参数 2 轴号
Ev.dest_loc = user16b0Loc;   //不保存结果则使其为 0
Ev.dest_axis = 0;            //轴号
```

B) AND16i、AND16iB

```
Ev.EventCMD = CMD_AND16iB;   //16 位参数与上常数
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src1_loc = user16b1Loc;   //参数 1 地址
Ev.Src1_axis = 0;            //参数 1 轴号
Ev.Src2.data16.data = 10000; //常数
Ev.dest_loc = user16b0Loc;   //不保存结果则使其为 0
Ev.dest_axis = 0;            //轴号
```

3.7.1.8 XOR 事件

A) XOR16、XOR16B

```
Ev.EventCMD = CMD_XOR16;         //16 位参数异或 16 位参数
Ev.EventType = IMC_Allways;  //触发事件类型
Ev.Src1_loc = user16b1Loc;   //参数 1 地址
Ev.Src1_axis = 0;            //参数 1 轴号
Ev.Src2.param.Src2_loc = user16b2Loc;   //参数 2 地址
Ev.Src2.param.Src2_axis = 0; //参数 2 轴号
Ev.dest_loc = user16b0Loc;   //不保存结果则使其为 0
Ev.dest_axis = 0;            //轴号
```

B) XOR16i、XOR16iB

```
Ev.EventCMD = CMD_XOR16iB;       //16 位参数异或常数
Ev.EventType = IMC_Allways;  //触发事件类型
```

```
Ev.Src1_loc = user16b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.data16.data = 4;      //常数
Ev.dest_loc = user16b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

3.7.1.9 OR 事件、

A) OR16、OR16B

```
Ev.EventCMD = CMD_OR16;       //16 位参数或 16 位参数
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user16b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.param.Src2_loc = user16b2Loc;   //参数 2 地址
Ev.Src2.param.Src2_axis = 0; //参数 2 轴号
Ev.dest_loc = user16b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

B) OR16i、OR16iB

```
Ev.EventCMD = CMD_OR16iB;     //16 位参数或常数
Ev.EventType = IMC_Allways;   //触发事件类型
Ev.Src1_loc = user16b1Loc;    //参数 1 地址
Ev.Src1_axis = 0;             //参数 1 轴号
Ev.Src2.data16.data = 4;      //常数
Ev.dest_loc = user16b0Loc;    //不保存结果则使其为 0
Ev.dest_axis = 0;             //轴号
```

3.8 IMC_ClearFIFO

【功能】清除 FIFO 中的所有指令
【定义】

```
IMC_STATUS IMC_ClearFIFO (IMC_HANDLE Handle, int fifoSel);
```

【参数】

Handle　　设备句柄；
fifoSel　　指定需要清除的FIFO。其值为枚举类型FIFO_SEL中的 SEL_IFIFO 、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 、SEL_CFIFO中的一个。

【返回】清除成功返回 IMC_OK，失败返回其他错误值。
【说明】 清除选定的 FIFO 中的所有指令和数据。

3.9 读取探针捕获函数

【功能】从探针捕获缓冲区（CAPFIFO）中读取位置。
【定义】
IMC_STATUS IMC_GetCapData (IMC_HANDLE Handle, IMC32 rdnum, IMC32 *pdata, IMC32 *dataNum, IMC32 *lastNum, IMC32 axis);
【参数】
Handle　　设备句柄；
rdnum　　读取的数据个数
pdata　　用于保存数据的数组；
dataNum　　返回成功读取到的数据个数；
lastNum　　返回缓冲区中剩余的数据个数；
axis　　指定读取的轴号。
【返回】成功返回 IMC_OK，失败返回其他错误值。
【说明】
　　使用探针捕获功能，需将 encpctr 寄存器的 bit10 置 1。
　　参数 rdnum 最大值只能到 120，即此函数一次读取回来的数据个数最多只能有 120 个。
【例】

```
IMC_HANDLE Handle;//此句柄由函数 IMC_Open 获取
IMC_STATUS Status;
IMC32 rdnum, dataNum, lastNum, axis;
IMC32 data[120];
…
rdnum = 100;
Axis = 0;
do{
    Status=IMC_GetCapData (Handle, rdnum,data,&dataNum,&lastNum, axis);
    if(Status==IMC_OK){ //获取捕获位置成功
        if(dataNum > 0){
            //处理读回的数据
        }
        if(lastNum == 0)//如果缓冲区中没有数据，则等待 50 毫秒
            Sleep(50);
    }else{
        //获取失败
    }
}while(1);
```

4 功能函数

功能函数是面向运动控制功能而封装的函数。这些函数实际上是对若干基本函数的封装，因此使用方便，易于理解，然而功能函数有限，对于功能函数未涵盖的功能，用户可以使用基本函数实现。

4.1 常用点到点运动函数

以下为常用的点到点函数，若无特别注明，均使用主坐标系。

4.1.1移动到绝对位置

```
IMC_STATUS IMC_MoveAbsolute (IMC_HANDLE Handle, IMC32 pos,
                             IMC32 axis, int fifoSel)
```

【功能】移动到指定的绝对目标位置

【参数】

Handle	设备句柄；
Pos	目标位置；
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该函数一般用于在点到点的普通模式下移动到指定的目标位置。

如果此指令发送到 PFIFO，则在插补指令之前必须增加等待指令，等待运动结束。

【例子】通过 QFIFO 发送指令使轴 1 从当前指令位置移动到 6250 位置

```
IMC_STATUS  Status;
IMC_HANDLE  handle; //此句柄值由 IMC_Open 获取
Status = IMC_MoveAbsolute （Handle， 6250，1，SEL_QFIFO）;
if( Status == IMC_OK) {
 //发送成功
}
```

4.1.2移动相对距离

```
IMC_STATUS IMC_MoveRelative (IMC_HANDLE Handle,IMC32 dist,
                             IMC32 axis, int fifoSel)
```

【功能】从当前指令位置移动指定长度的距离。

【参数】

Handle	设备句柄；
dist	需移动的相对于当前指令位置的距离；
axis	轴号；

fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该函数一般用于在点到点的普通模式下移动指定的相对距离。

如果此指令发送到 PFIFO，则在插补指令之前必须增加等待指令，等待运动结束。

4.1.3叠加运动

IMC_STATUS IMC_MoveSuperimposed (IMC_HANDLE Handle, IMC32 dist, IMC32 axis, int fifoSel)

【功能】以叠加的方式运动一段距离，该轴当前可能正在执行主坐标系下的点到点运动，连续运动，或跟随其它轴作电子齿轮运动等。该函数实际上是执行辅坐标系下相对距离的移动。

【参数】

Handle　　设备句柄；

dist　　需叠加移动的距离；

axis　　轴号；

fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该函数一般用于从动轴跟随主动轴作电子齿轮运动时，改变从动轴相对于主动轴的相位。

如果此指令发送到 PFIFO，则在插补指令之前必须增加等待指令，等待运动结束。

4.1.4移相运动

IMC_STATUS IMC_Phasing (IMC_HANDLE Handle, IMC32 dist, IMC32 axis, int fifoSel)

【功能】在齿轮同步运动中，对主动轴执行移相运动可使主动轴相对于从动轴的相位移动一段距离，该函数执行的实际上是在主动轴的辅坐标系中执行相对运动。

【参数】

Handle　　设备句柄；

dist　　相对于从动轴移动的距离；

axis　　轴号；

fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】从动轴的跟随源必须指向主动轴的 MCSvel 参数。

如果此指令发送到 PFIFO，则在插补指令之前必须增加等待指令，等待运动结束。

4.2 连续运动模式

连续运动模式是指给定运行的目标速度，电机加速或减速到达目标速度后，以目标速度持续运行。连续运动模式可以长期连续运行，若位置溢出 iMC 的位置范围[-2^{31}, $2^{31}-1$]，自动重新由原点位置开始继续运行，不影响正在运行的状态。

4.2.1 连续速度运动

IMC_STATUS IMC_MoveVelocity (IMC_HANDLE Handle, IMC32 vel, IMC32 axis, int fifoSel)

【功能】轴以设定的目标速度值 vel 连续运动。

【参数】

Handle	设备句柄；
Vel	目标移动速度；
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】

Vel 为正数则往正方向运动，为负数则往负方向运动。

4.2.2 点动（JOG）

该功能可由主机调用 IMC_MoveVelocity(IMC_HANDLE Handle, IMCFLOAT vel, IMC32 axis, int fifoSel)实现；将速度 vel 设置为非零值，则电机以指定速度 vel 连续移动，将速度设为 0，则电机停止移动。

【例子】启动轴 0 的电机以 100 脉冲每毫秒的速度移动

```
IMC_STATUS  Status;
IMC_HANDLE  handle; //此句柄值由 IMC_Open 获取
Status = IMC_MoveVelocity（handle， 100*65536，0，SEL_IFIFO）；
if( Status == IMC_OK) {
 //发送成功，将会以 100 的速度连续运动
}
```

停止轴 0 的电机移动

```
IMC_STATUS  Status;
IMC_HANDLE  handle; //此句柄值由 IMC_Open 获取
Status = IMC_MoveVelocity（handle， 0，0，SEL_IFIFO）；
if( Status == IMC_OK) {
 //发送成功，电机停止
}
```

4.3 搜寻原点（homeing）

iMC 提供七种搜寻机械原点的方案，可归为三类，分别是：（1）仅使用原点开关作为原点信号；（2）使用原点开关及编码器索引信号（Z 相脉冲）；（3）仅使用编码器的索引信号。搜寻原点过程中使用两个速度参数：highvel 和 lowvel，因此，在调用搜寻原点的函数前，应先设置这两个速度参数值。

4.3.1仅使用原点开关

（1）IMC_STATUS IMC_HomeSwitch1 (IMC_HANDLE Handle, IMC16 direction, IMC16 RiseEdge, IMC16 stop, IMC32 axis, int fifoSel)

【功能】向指定方向搜寻机械原点，不使用编码器索引信号，仅搜寻原点开关（下面的 ORG 即代表该开关信号）。采用这种方式搜寻原点时，电机轴应朝原点开关所在的方向移动，直到控制器检测到原点开关 ORG 的有效边沿信号（可设置为上升沿或下降沿）时执行设置原点的操作，并停止移动，完成整个过程。

【参数】

Handle	设备句柄；
direction	指定启动的方向，0 为正方向，1 为负方向；
RiseEdge	指定检测原点开关的边沿。非零：上升沿有效； 零：下降沿有效；
stop	非零：搜寻到原点后移动到指定位置； 零：搜寻到原点后减速停止
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该过程使用高速度值（预设的 highvel 参数值）。

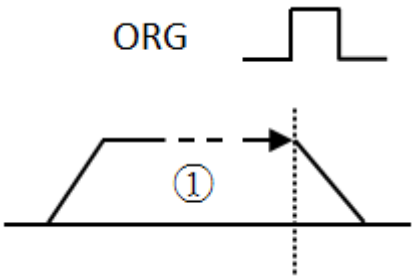

图 3.5-1

（2）IMC_STATUS IMC_HomeSwitch2 (IMC_HANDLE Handle, IMC16 direction, IMC16 RiseEdge, IMC16 stop, IMC32 axis, int fifoSel)

【功能】向指定方向搜寻机械原点，不使用编码器索引信号，仅搜寻原点开关，触发原点开关后先停止，然后返回并在到达 ORG 第一次触发的“同一边沿位置”时执行设置原点的操作，然后停止移动，完成搜寻过程。

【参数】

Handle	设备句柄；
direction	指定开始搜寻的方向，0 为正方向，1 为负方向；

RiseEdge　指定检测原点开关的边沿。非零：上升沿有效；　零：下降沿有效；
stop　非零：搜寻到原点后移动到指定位置；　零：搜寻到原点后减速停止
axis：　轴号；
fifoSel　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该封装函数规定，第一阶段（从起动到第一次停止的过程）的搜寻速度使用预设的 highvel，第二阶段使用预设的 lowvel。所谓“同一边沿位置”是指开关信号发生翻转的同一个位置点，如图 3.5-2 中的虚线所处的位置，第一次触发时是上升沿，返回时是下降沿。

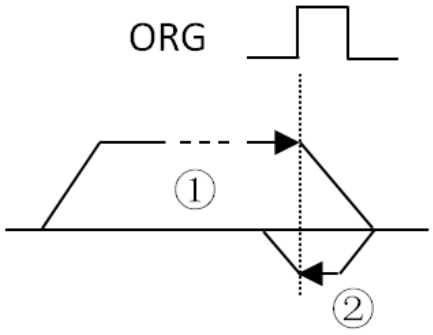

图 3.5-2

(3) IMC_STATUS IMC_HomeSwitch3 (IMC_HANDLE Handle, IMC16 direction, IMC16 RiseEdge, IMC16 stop, IMC32 axis, int fifoSel)

【功能】向指定方向搜寻机械原点，仅搜索原点开关。触发原点开关后先停止，然后返回并在到达 ORG 第一次触发的“同一边沿位置”时并不执行原点设置操作，而是先停止，然后再次返回，直到又一次检测到原点开关的同一边沿位置才执行设置原点的操作并停止，完成搜寻过程。

【参数】

Handle　设备句柄；
direction　指定开始搜寻的方向，0 为正方向，1 为负方向；
RiseEdge　指定检测原点开关的边沿。非零：上升沿有效；　零：下降沿有效；
stop　非零：搜寻到原点后移动到指定位置；　零：搜寻到原点后减速停止
axis：　轴号；
fifoSel　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该封装函数规定，第一阶段的搜寻速度使用 highvel，其余阶段使用 lowvel。

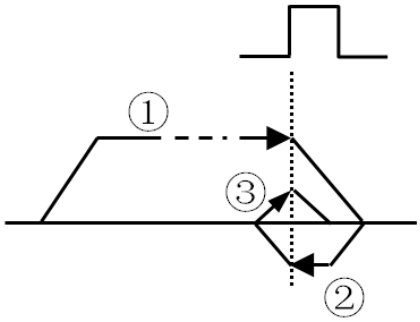

图 3.5-3

4.3.2使用原点开关及编码器索引信号

（1）IMC_STATUS IMC_HomeSwitchIndex1 (IMC_HANDLE Handle, IMC16 direction, IMC16 RiseEdge, IMC16 stop, IMC32 axis, int fifoSel)

【功能】向指定方向搜寻机械原点，搜索到原点开关后再检测编码器的索引信号（Index）。检测到原点开关 ORG 的有效边沿后，iMC 内部开始对索引信号（index）进行计数，在计数值等于 indexNum 之时执行设置原点的操作，并停止移动，完成原点搜寻过程。

【参数】

Handle	设备句柄；
direction	指定开始搜寻原点的移动方向，0 为正方向，1 为负方向；
RiseEdge	指定检测原点开关的边沿。非零：上升沿有效； 零：下降沿有效；
stop	非零：搜寻到原点后移动到指定位置； 零：搜寻到原点后减速停止
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该封装函数规定，搜寻速度使用 highvel。

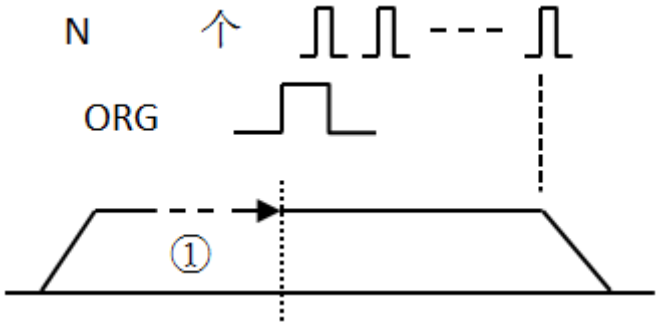

图 3.5-4

（2）IMC_STATUS IMC_HomeSwitchIndex2 (IMC_HANDLE Handle, IMC16 direction, IMC16 RiseEdge, IMC16 stop, IMC32 axis, int fifoSel)

【功能】向指定方向搜寻机械原点，搜索到原点开关后，先停止，然后返回，并

在检测到原点开关的同一边沿位置时开始对编码器的 index 信号进行计数，在计数值等于 indexNum 之时执行设置原点的操作，并停止移动，完成原点搜寻过程。

【参数】

Handle　　设备句柄；

direction　指定开始搜寻的方向，0 为正方向，1 为负方向；

RiseEdge　指定检测原点开关的边沿。非零：上升沿有效；　零：下降沿有效；

stop　　非零：搜寻到原点后移动到指定位置；　零：搜寻到原点后减速停止

axis　　轴号；

fifoSel　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该封装函数规定，第一阶段的搜寻速度使用 highvel，第二阶段使用 lowvel。

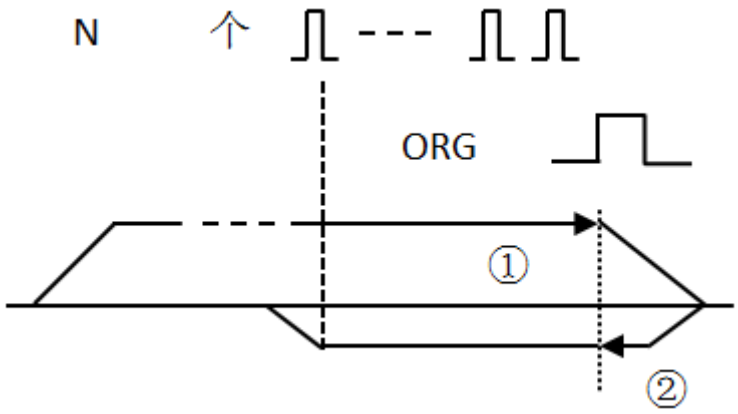

图 3.5-5

(3)IMC_STATUS IMC_HomeSwitchIndex3 (IMC_HANDLE Handle, IMC16 direction, IMC16 RiseEdge, IMC16 stop, IMC32 axis, int fifoSel)

【功能】向指定方向搜寻机械原点，搜索到原点开关后，先停止，然后反向移动，并在检测到原点开关的同一边沿位置时停止，然后再次返回，即与起始方向同向，在原点开关的同一边沿位置开始对 index 信号进行计数，在计数值等于 indexNum 之时执行设置原点的操作，并停止移动，完成原点搜寻过程。

【参数】

Handle　　设备句柄；

direction　指定开始搜寻的方向，0 为正方向，1 为负方向；

RiseEdge　指定检测原点开关的边沿。非零：上升沿有效；　零：下降沿有效；

stop　　非零：搜寻到原点后移动到指定位置；　零：搜寻到原点后减速停止

axis　　轴号；

fifoSel　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该封装函数规定，第一阶段的搜寻速度使用 highvel，其余阶段使用 lowvel。

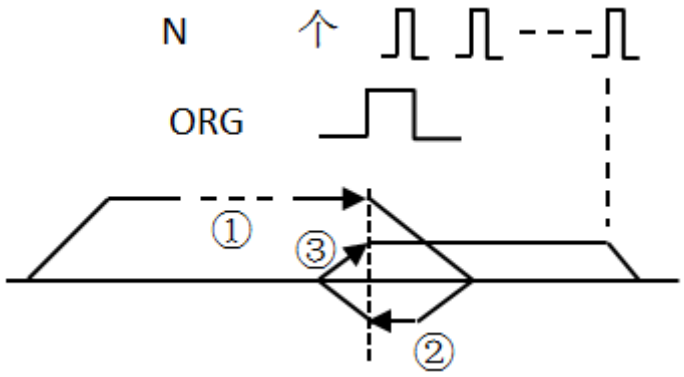

图 3.5-6

4.3.3仅使用编码器索引信号

```
IMC_STATUS IMC_HomeIndex (IMC_HANDLE Handle,IMC16 direction,
                          IMC16 stop,IMC32 axis, int fifoSel)
```

【功能】向指定方向搜寻机械原点，仅检测编码器的索引信号（Index），在计数值等于 indexNum 之时执行设置原点的操作，并停止移动，完成原点搜寻过程。

【参数】

Handle	设备句柄；
direction	指定搜寻的方向，0 为正方向，1 为负方向；
stop	非零：搜寻到原点后移动到指定位置； 零：搜寻到原点后减速停止
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】该封装函数规定，使用 highvel 作为运行速度。

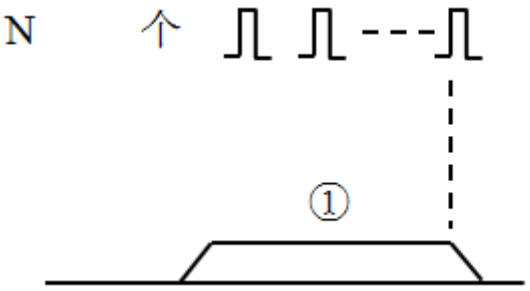

图 3.5-7

4.4 常用电子齿轮运动

为了使用方便，封装了部分常用的电子齿轮运动。

4.4.1 自由啮合过程的电子齿轮运动

```
IMC_STATUS IMC_GearFree (IMC_HANDLE Handle, IMCFLOAT ratio, IMC32 master,
                         IMC32 sourcetype, IMC32 axis, int fifoSel)
```

【功能】跟随主动轴以设定的倍率运动，可以在主动轴运动的过程中任何时候执行该函数。从动轴由其当前速度加速/减速达到设定的传动比，即存在一个斜升过程，该过程使用该轴的加速度和减速度。在跟随过程中可以随时改变传动倍率。

【参数】

Handle	设备句柄；
ratio	传动倍率，取值范围-32768.0~32767.9999999998；
master	主动轴轴号，值范围 0~AXISNUM-1；
sourcetype	用于指定源参数的类型；
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】注意：axis 参数是从动轴的轴号。sourcetype 用于指定源参数的类型，所谓源参数是指主动轴中提供位置变量为从动轴跟随的参数，可选的有：

0——指令位置增量，即指令速度值，该值包含了主坐标系的运动增量和辅坐标系的运动增量，并经平滑处理后的量 curvel；

1——编码器增量，即编码器速度（实际速度 encpvel）；

2——主坐标系的指令位置增量 MCSvel；

3——辅坐标系的指令位置增量 PCSvel；

4——辅助编码器增量 encsvel。

在一般的应用场合下，若需跟随的是主动轴的指令位置，可选择经平滑处理后的指令位置增量，即 sourcetype=0，若需跟随的是主动轴的实际位置，可选择编码器增量，即 sourcetype=1。

要使从动轴脱离齿轮运动，需设置engear寄存器的值为0值。

4.4.2 啮合过程与位移同步的电子齿轮运动

```
IMC_STATUS IMC_GearInDist (IMC_HANDLE Handle,IMCFLOAT ratio,
                           IMC32 master, IMC32 sourcetype, IMC32 synaxis,
                           IMC32 synsourcetype,IMC32 syndist,
                           IMC32 axis, int fifoSel)
```

【功能】跟随主动轴以设定的倍率运动，可以在主动轴运动的过程中任何时候执行该函数。接合过程与位移同步，开始同步后（置 startsyn 非零），从动轴经由

特定的算法控制逐渐到达设定的传动比，即完全接合，该过程是参照 synaxis 轴的某个参数的变化同步的，该参数值变化 syndist 的长度后，达到完全接合。该参数称为同步源参数。

【参数】

Handle　　设备句柄；

ratio　　传动倍率，取值范围-32768.0~32767.999999999767

master　　主动轴轴号，值范围 0~AXISNUM-1

sourcetype　　用于指定源参数的类型，可选的有：

0——指令位置增量，即指令速度值，该值包含了主坐标系的运动增量和辅坐标系的运动增量，并经平滑处理后的量 curvel；

1——编码器增量，即编码器速度（实际速度 encpvel）；

2——主坐标系的指令位置增量 MCSvel；

3——辅坐标系的指令位置增量 PCSvel；

4——辅助编码器增量 encsvel。

synaxis　　同步源参数所在的轴号。注：synaxis 可以不同于 master。

synsourcetype 同步源参数类型，可选的有：

0——指令位置，即指令位置值，该值包含了主坐标系的运动及辅坐标系的运动合成，并经平滑处理后的量，即 curpos；

1——编码器位置，即实际位置 encp；

2——主坐标系的指令位置 MCSpos；

3——辅坐标系的指令位置 PCSpos；

4——辅助编码器位置 encs，

syndist　　同步啮合过程中同步源参数历经的长度。

axis　　从动轴的轴号。

fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值

【说明】注：该函数执行后并非立即开始接合，而是等待 startsyn 置为非零值才开始。可以通过事件置 startsyn 非零。startsyn 置非零且同步源参数历经 syndist 长度的位移后，从动轴达到完全接合。

要使从动轴脱离齿轮运动，需设置engear寄存器的值为0值。

4.4.3啮合过程与时间同步的电子齿轮运动

```
IMC_STATUS IMC_GearInTime (IMC_HANDLE Handle, IMCFLOAT ratio,
                           IMC32 master, IMC32 sourcetype,
                           IMC32 syntime , IMC32 axis, int fifoSel)
```

【功能】跟随主动轴以设定的倍率运动，可以在主动轴运动的过程中任何时候执行该函数。开始同步接合后（置 startsyn 非零），从动轴经由特定的算法控制逐渐到达设定的传动比，即达到完全接合，该过程是与时间同步，历经 syntime 的时间后，达到完全接合。

【参数】

Handle　　设备句柄；
ratio　　传动倍率，取值范围-32768.0~32767.999999999767；
master　　主动轴轴号，值范围 0~AXISNUM-1
sourcetype 用于指定源参数的类型，可选的有：
　0——指令位置增量，即指令速度值，该值包含了主坐标系的运动增量和辅坐标系的运动增量，并经平滑处理后的量 curvel；
　1——编码器增量，即编码器速度（实际速度 encpvel）；
　2——主坐标系的指令位置增量 MCSvel；
　3——辅坐标系的指令位置增量 PCSvel；
　4——辅助编码器增量 encsvel。
syntime　　同步啮合过程中经历时间的长度。
axis　　从动轴的轴号。
fifoSel　　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。
【返回】成功返回 IMC_OK，失败返回其他错误值。
【说明】注：该函数执行后并非立即开始接合，而是等待 startsyn 置为非零值才开始。可以通过事件置 startsyn 非零。startsyn 置非零且经历 syntime 的时间后，从动轴达到完全接合。
　要使从动轴脱离齿轮运动，需设置engear寄存器的值为0值。

4.5 电子手轮运动

手轮运动也是由电子齿轮衍生而实现的，手轮编码器连接辅助编码器，所连接的辅助编码器所在的轴作为主动轴，执行手轮运动的轴作为从动轴。

4.5.1手轮运动

```
IMC_STATUS IMC_HandWheel (IMC_HANDLE Handle, IMCFLOAT ratio,
                          IMC32 encsaxis, IMC32 axis, int fifoSel)
```

【功能】指定的轴跟随手轮移动，实际上是跟随连接了手轮的辅助编码器作电子齿轮运动。

【参数】

Handle　　设备句柄

ratio　　手轮倍率，即该轴跟随辅助编码器的倍率，取值范围：-32768.0~32767.9999999998；

encsaxis　所跟随的辅助编码器所在的轴号，即手轮接入的轴的轴号；

axis　　执行手轮运动的轴号。

fifoSel　选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值

【说明】注意：通过该函数实现手轮运动后，若需退出，可调用函数IMC_SetParam16对相应轴的engear寄存器清零。

4.6 同步运动

iMC 提供了一些不同于电子齿轮的同步运动，如时间-位移同步运动（T-P）、位移-位移同步运动（P-P）、时间-速度（T-V）、位移-速度（P-V）。

4.6.1时间-位移（T-P）同步运动

```
IMC_STATUS IMC_MoveInTime (IMC_HANDLE Handle, IMC32 time,
                           IMC32 movedist, IMC32 axis, int fifoSel)
```

【功能】在指定的时间 time 内，指定的轴移动 movedist 的距离；注意：执行该函数后没有立即移动，仅在 startsyn 置非零后才开始移动，并在指定时间内移动指定的距离，并停止。

【参数】

Handle	设备句柄；
time	指定的时间；
movedist	在指定的时间内移动的距离；
axis	执行移动的轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】时间长度是从 startsyn 有效（非零）开始计时。

4.6.2位移-位移（P-P）运动

```
IMC_STATUS IMC_MoveInDist (IMC_HANDLE Handle, IMC32 refdist,
                           IMC32 refaxis, IMC32 movedist,
                           IMC32 axis, int fifoSel)
```

【功能】轴号为 axis 的轴，在参照轴(refaxis)的指定位移(refdist)内移动指定的距离(movedist)。

【参数】

Handle	设备句柄；
refdist	参照轴移动的距离；
refaxis	参照轴的轴号；
movedist	轴号为 axis 的轴移动的距离；
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】函数执行后，必须设置 enasyn 和 startsyn 为非零后才开始移动，并在参照轴移动了指定的长度 refdist 后停止，此过程中该轴恰好移动了 movedist 的距离。

4.6.3时间-速度运动（T-V）

```
IMC_STATUS IMC_TgvelInTime (IMC_HANDLE Handle, IMC32 time,
                            IMC32 tgvel, IMC32 axis, int fifoSel)
```

【功能】指定的时间内达到目标速度，即在指定的时间 time 内，指定轴的速度从当前速度加速或减速到 tgvel 指定的目标速度，并以此速度连续运动。

【参数】

Handle	设备句柄；
time	时间长度；
tgvel	设定的目标速度；
axis	轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】时间长度是从 startsyn 有效（非零）开始计时。

4.6.4位移-速度运动（P-V）

```
IMC_STATUS IMC_TgvelInDist (IMC_HANDLE Handle, IMC32 refdist,
                            IMC32 refaxis, IMC32 tgvel,
                            IMC32 axis, int fifoSel)
```

【功能】参照轴 refaxis 移动指定的距离 refdist 内，axis 轴达到指定的目标速度，并以此速度连续运动。

【参数】

Handle	设备句柄；
refdist	参照轴移动的距离；
refaxis	参照轴的轴号；
tgvel	执行轴的目标速度；
axis	执行轴的轴号；
fifoSel	选择接收此指令的FIFO。其值为枚举类型FIFO_SEL中的SEL_IFIFO、SEL_QFIFO、SEL_PFIFO1、SEL_PFIFO2 中的一个。

【返回】成功返回 IMC_OK，失败返回其他错误值。

【说明】执行该函数后，必须设置 enasyn 和 startsyn 为非零后才开始该运动。

5 iMC 参数地址宏定义

本章列出的是 iMC 参数地址的宏定义，其格式为：

#define 参数名 Loc　　地址　　//数据类型

其中“参数名”是指在 iMC 中的参数名，在其后加 Loc 表示该参数的地址。

“//数据类型”决定了最终使用的 IMC_SetParam 函数或 IMC_GetParam 函数，共有以下几种数据格式的类型：

S32：32 位带符号位的整数，参数值的范围：[-2147483648,2147483647]

U32：32 位无符号位的整数，参数值的范围：[0, 4294967296]

S16：16 位带符号位的整数，参数值的范围：[-32768,32767]

U16：16 位无符号位的整数，参数值的范围：[0, 65535]

F16：16 位标志，仅取两个值：0 或 FFFFh

R16：16 位寄存器，各位域具有具体的意义，部分位域需设置

S16.32：48 位带符号位，高 16 位为整数部分，低 32 位为小数部分，参数值的范围：[-32768.0,32767.999999999767]

U16.32：48 位无符号位，高 16 位为整数部分，低 32 位为小数部分，参数值的范围：[0.0,65535.999999999767]

S16.16：32 位带符号位，高 16 位为整数部分，低 16 位为小数部分，参数值的范围：[-32768.0,32767.999984741211]

U16.16：32 位无符号位，高 16 位为整数部分，低 32 位为小数部分，参数值的范围：[0.0,65535.999984741211]

除了在此处列有参数地址外，同时也保存在头文件 paramdef.h 中。这里列出的宏定义只能做为参考，请以文件 paramdef.h 中的为准，因为文件中实时保持最后更新。

5.1 主坐标系（MCS）点到点参数

```
#define mcsvelLoc        4      //S16.16
#define mcstgvelLoc      6      //S16.16
#define mcsslopeLoc      8      //F16
#define mcsposLoc        10     //S32
#define mcstgposLoc      12     //S32
#define mcsdistLoc       14     //S32
#define mcsmaxvelLoc     16     //S16.16
#define mcsaccelLoc      18     //S16.16
#define mcsdecelLoc      20     //S16.16
#define mcstrackLoc      22     //F16
#define mcsgoLoc         23     //F16
#define mcsmovingLoc     25     //F16
```

5.2 辅坐标系（PCS）点到点参数

```
#define pcsvelLoc        27     //S16.16
#define pcstgvelLoc      29     //S16.16
#define pcsslopeLoc      31     //F16
#define pcsposLoc        33     //S32
#define pcstgposLoc      35     //S32
#define pcsdistLoc       37     //S32
#define pcsmaxvelLoc     39     //S16.16
#define pcsaccelLoc      41     //S16.16
#define pcsdecelLoc      43     //S16.16
#define pcstrackLoc      45     //F16
#define pcsgoLoc         46     //F16
#define pcsmovingLoc     48     //F16
```

5.3 主编码器参数

```
#define encpfactorLoc    73     //U16.16
#define encpvelLoc       75     //S16.16
#define encpLoc          78     //S32
```

5.4 辅编码器参数

```
#define encsfactorLoc    81     //U16.16
#define encsvelLoc       83     //S16.16
#define encsLoc          86     //S32
```

5.5 其它轴参数

```
#define runLoc           128    //F16
#define statusLoc        129    //R16
#define errorLoc         130    //R16
#define poserrlimLoc     131    //U16
#define smoothLoc        132    //U16
#define trackwinLoc      133    //U16
#define settlewinLoc     134    //U16
#define settlenLoc       135    //U16
#define stopfiltLoc      136    //R16
#define stopmodeLoc      137    //R16
#define exitfiltLoc      138    //R16
```

```
#define psoftlimLoc      139    //S32
#define nsoftlimLoc      141    //S32
#define breakpLoc        143    //S32
#define homeposLoc       145    //S32
#define lowvelLoc        147    //S16.16
#define highvelLoc       149    //S16.16
#define homeseqLoc       151    //R16
#define gohomeLoc        152    //F16
#define sethomeLoc       153    //F16
#define captureLoc       155    //F16
#define clearLoc         157    //F16
#define vellimLoc        158    //S16.16
#define accellimLoc      160    //S16.16
#define fixvelLoc        162    //S16.16
```

5.6 电子齿轮参数

```
#define masterLoc        169    //U16
#define gearsrcLoc       170    //U16
#define engearLoc        171    //F16
#define gearoutmodLoc    172    //F16
#define shiftmasterLoc   174    //F16
#define gearratioLoc     175    //S16.32
#define gearoutvelLoc    178    //S16.16
```

5.7 环形轴参数

```
define cirposLoc         184    //S32
#define ciraxisLoc       186    //F16
```

5.8 非线性同步控制参数

```
#define enasynLoc        188    //F16
#define startsynLoc      189    //F16
#define syntypeLoc       190    //F16
#define synsrcLoc        191    //U16
#define synaxisLoc       192    //U16
#define synsrcvarLoc     193    //S32
#define slavedistLoc     197    //S32
#define slaveabsLoc      199    //F16
```

5.9 运动状态参数

```
#define profilingLoc     215    //F16
#define smoothingLoc     216    //F16
#define contouringLoc    217    //F16
#define movingLoc        218    //F16
#define motionLoc        219    //F16
#define outsettleLoc     220    //F16
#define outtrackLoc      221    //F16
#define poserrLoc        223    //S16
#define curposLoc        225    //S32
#define curvelLoc        227    //S16.16
#define capposLoc        229    //S32
```

5.10 输入输出参数

```
#define encsctrLoc          531    //R16
#define clrencsLoc          532    //F16
#define encpctrLoc          539    //R16
#define enaLoc              550    //F16
#define aioLoc              562    //R16
#define aiodLoc             563    //R16
#define stepmodLoc          615    //R16
#define dirtimeLoc          618    //U16
#define steptimeLoc         619    //R16
#define aioctrLoc           680    //R16
#define aiolatLoc           682    //R16
#define clraiolatLoc        682    //R16
```

5.11 轮廓运动参数

```
#define startgroup          256    //F16
#define groupnumLoc         257    //U16
#define group_xLoc          258    //U16
#define group_yLoc          259    //U16
#define group_zLoc          260    //U16
#define group_aLoc          261    //U16
#define group_bLoc          262    //U16
#define group_cLoc          263    //U16
#define group_dLoc          264    //U16
#define group_eLoc          265    //U16
```

```
#define group_fLoc          266    //U16
#define group_gLoc          267    //U16
#define group_hLoc          268    //U16
#define group_iLoc          269    //U16
#define group_jLoc          270    //U16
#define group_kLoc          271    //U16
#define group_lLoc          272    //U16
#define group_mLoc          273    //U16
#define groupsmoothLoc      274    //U16
```

5.12 预定义用户参数

```
#define user16b0Loc         307
#define user16b1Loc         308
#define user16b2Loc         309
#define user16b3Loc         310
#define user16b4Loc         311
#define user16b5Loc         312
#define user16b6Loc         313
#define user16b7Loc         314
#define user16b8Loc         315
#define user16b9Loc         316
#define user32b0Loc         317
#define user32b1Loc         319
#define user32b2Loc         321
#define user32b3Loc         323
#define user32b4Loc         325
#define user32b5Loc         327
#define user32b6Loc         329
#define user32b7Loc         331
#define user32b8Loc         333
#define user32b9Loc         335
#define user48b0Loc         337
#define user48b1Loc         340
#define user48b2Loc         343
#define user48b3Loc         346
#define user48b4Loc         349
```

```
#define waitfiltLoc          479    //R16
#define waitequLoc           480    //R16
#define timerLoc             481    //U16
#define eventsLoc            489    //U16
#define emstopLoc            500    //F16
#define hpauseLoc            501    //F16
#define ticksLoc             502    //U32
#define clrififoLoc          513    //F16
#define cfifocntLoc          519    //R16
#define clrqfifoLoc          521    //F16
#define gioLoc               560    //R16
#define giodLoc              561    //R16
#define eventidLoc           549    //U16
#define eventbaseLoc         568    //U16
#define out1Loc              624    //F16
#define out2Loc              625    //F16
#define delayusLoc           704    //U32
#define delayoutLoc          704    //U32
```

5.13 iMC 基本信息参数

```
#define clkdivLoc        509    //U16
#define fwversionLoc     511    //U16
#define sysclkLoc        628    //U32
#define naxisLoc         634    //U16
#define hwversionLoc     635    //U16
```

6 附录 类型定义

iMC 有 4 个先进先出(FIFO)缓冲器,主机通过这 4 个先进先出缓冲器(FIFO)与 iMC 运动控制芯片进行信息交互；分别为 IFIFO 和 QFIFO，PFIFO1 和 PFIFO2

IFIFO 和 QFIFO 的功能是一样的,都可以按序逐个写入指令和主机请求序列。可以理解为主机与芯片通信的两条通道，它们的区别主要体现于使用上：

IFIFO：写入该 FIFO 的指令（或请求）会立即执行，一般用于需要立刻执行的指令，因此该 FIFO 的深度较浅

QFIFO：一般用于写入一连串的指令序列，包括阻塞指令。通常深度较 IFIFO 深，是运动控制指令的主要通道。

PFIFO1 和 PFIFO2 的功能也是一样的，这 2 个 FIFO 主要用于插补运动。控制卡上电时，这 2 个 FIFO 的功能是禁止的，需要在程序中启用才能有作用。当启用这 2 个 FIFO 后,它们除了可以用于插补运动,同时还有 IFIFO、QFIFO 的功能。

```
typedef PVOID IMC_HANDLE;

enum IMC_STATUS {
    IMC_OK=0,                   //成功
    IMC_SEND_FAIL,              //数据发送失败
    IMC_READ_FAIL,              //数据接收失败
    IMC_TIME_OUT,               //数据发送接收超时
    IMC_DEVICE_NOT_OPEN,        //设备没有打开
    IMC_DEVICE_NOT_FOUND,       //设备没有找到,用于打开设备时
    IMC_INVALID_HANDLE,         //无效的设备句柄
    IMC_INVALID_PARAM,          //无效的参数
    IMC_INVALID_AXIS,           //无效的轴号
    IMC_INVALID_FIFO,           //无效的FIFO
    IMC_FIFO_FULL,              //FIFO满
    IMC_FIFO_NULL,              //FIFO空
    IMC_PARAM_READ_ONLY,        //只读参数
    IMC_OUT_OF_RANGE,           //传递进来的函数参数值超出范围
    IMC_CHECK_ERROR,            //校ê验错误
    IMC_VERSION_ERROR,          //函数库版本与硬件版本不匹配
    IMC_OTHER_ERROR,            //其他错误值
    IMC_PASSWORD_ERROR,         //密码错误
    IMC_RBFIFO_EMPTY,           //RBFIFO空
};

//网卡信息
typedef struct {
    char description[16][256];  //网卡描述
```

```
}NIC_INFO,*PNIC_INFO;

typedef struct _WR_MUL_DES_{
    WORD addr;//参数地址
    WORD axis;//轴号
    WORD len;  //参数长度（单位：字（16 位宽）， 其值为 1、2、3；  1:表示
一个字（short） 2:表示 2 个字（long） 3:表示 3 个字（））
    WORD data[4];//返回的数据
}WR_MUL_DES, *pWR_MUL_DES;


//事件类型
enum IMC_EventType{
    IMC_Allways,       //“无条件执行”

    IMC_Edge_Zero,     //“边沿型条件执行”——变为0时
    IMC_Edge_NotZero, //“边沿型条件执行”——变为非0时
    IMC_Edge_Great,    //“边沿型条件执行”——变为大于时
    IMC_Edge_GreatEqu, //“边沿型条件执行”——变为大于等于时
    IMC_Edge_Little,  //“边沿型条件执行”——变为小于时
    IMC_Edge_Carry,    //“边沿型条件执行”——变为溢出时
    IMC_Edge_NotCarry, //“边沿型条件执行”——变为无溢出时

    IMC_IF_Zero,       //“电平型条件执行”——若为0
    IMC_IF_NotZero,    //“电平型条件执行”——若为非0
    IMC_IF_Great,      //“电平型条件执行”——若大于
    IMC_IF_GreatEqu,  //“电平型条件执行”——若大于等于
    IMC_IF_Little,     //“电平型条件执行”——若小于
    IMC_IF_Carry,      //“电平型条件执行”——若溢出
    IMC_IF_NotCarry   //“电平型条件执行”——若无溢出
};

//事件指令
enum IMC_EVENT_CMD{
    CMD_ADD32,     //两个bit参数相加
    CMD_ADD32i,    //32bit参数加32bit立即数
    CMD_ADD48,     //两个bit参数相加
    CMD_ADD48i,    //48bit参数加48bit立即数
    CMD_CMP32,     //两个32bit参数相比较
    CMD_CMP32i,    //32bit参数与32bit立即数相比较
    CMD_CMP48,     //两个48bit参数相比较
    CMD_CMP48i,    //48bit参数与48bit立即数相比较
    CMD_SCA32,     //32bit参数缩放，倍率(48bit)为另一参数
```

```
    CMD_SCA32i,    //32bit参数缩放，倍率(48bit)为立即数
    CMD_SCA48,     //48bit参数缩放，倍率(48bit)为另一参数
    CMD_SCA48i,    //48bit参数缩放，倍率(48bit)为立即数
    CMD_MUL32L,    //32bit参数乘以32bit参数其结果取低32bit
    CMD_MUL32iL,   //32bit参数乘以立即数其结果取低32bit
    CMD_MUL32A,    //32bit参数乘以32bit参数其结果取低48bit
    CMD_MUL32iA,   //32bit参数乘以立即数其结果取低48bit
    CMD_COP16,     //拷贝16bit参数
    CMD_COP32,     //拷贝32bit参数
    CMD_COP48,     //拷贝48bit参数
    CMD_SET16,     //设置16bit参数
    CMD_SET32,     //设置32bit参数
    CMD_SET48,     //设置48bit参数
    CMD_OR16,      //参数OR 参数
    CMD_OR16i,     //参数OR 立即数
    CMD_OR16B,     //参数OR 参数 其结果转换为BOOL类型
    CMD_OR16iB,    //参数OR 立即数 其结果转换为BOOL类型
    CMD_AND16,     //参数AND 参数
    CMD_AND16i,    //参数AND 立即数
    CMD_AND16B,    //参数AND 参数 其结果转换为BOOL类型
    CMD_AND16iB,   //参数AND 立即数 其结果转换为BOOL类型
    CMD_XOR16,     //参数OR 参数
    CMD_XOR16i,    //参数OR 立即数
    CMD_XOR16B,    //参数OR 参数其结果转换为BOOL类型
    CMD_XOR16iB    //参数OR 立即数其结果转换为BOOL类型
};

typedef struct _EVENT_INFO_ {
    IMC16 EventCMD;   //事件指令，即枚举类型 IMC_PATH_EVENT_CMD 中的值
    IMC16 EventType;  //事件格式，即枚举类型 IMC_EventType 中的值
    IMC16 Src1_loc;              //源参数 1
    IMC16 Src1_axis;          //源参数 1 所属的轴号
    union{
        struct {
            IMC16 Src2_loc;       //源参数 2
            IMC16 Src2_axis;  //源参数 2 所属的轴号
            IMC32 reserve;    //保留
        }param;
        struct {
            IMC16 data;           //16 位宽的数据
            IMC16 reserve1;       //保留
            IMC32 reserve2;       //保留
        }data16;
```

```
        struct {
            IMC32 data;          //32 位宽的数据
            IMC32 reserve;       //保留
        }data32;
        __int64 data48;          //48 位宽的数据
    }Src2;
    IMC16 dest_loc;                 //目标参数
    IMC16 dest_axis;             //目标参数所属的轴号
    IMC32 reserve;               //保留
}EventInfo, *PEventInfo;



typedef struct _PFIFO_SEG_DATA_{
    IMC32 datanum;       //SegEndData 数组中有效数据个数
                         //即参与路径运动的轴数
    IMC32 SegEndData[16];
    //以下3个成员为圆弧插补使用
    IMC32 CenterX;       //圆弧路径圆心的横坐标
    IMC32 CenterY;       //圆弧路径圆心的纵坐标
    IMC32 dir;           //0: 顺时针方向,   -1：逆时针方向

}PFIFOSegData,*PPFIFOSegData;

typedef struct _PFIFO_SEG_INFO_{
    IMC32 SegTgVel;   //目标速度
    IMC32 SegEndVel;  //终点速度
    PFIFOSegData data;   //段数据
}PFIFOSegInfo, *PPFIFOSegInfo;
```