SFC 总线控制器

编程手册

创无限 Innovate for more 赢久远 Win forever

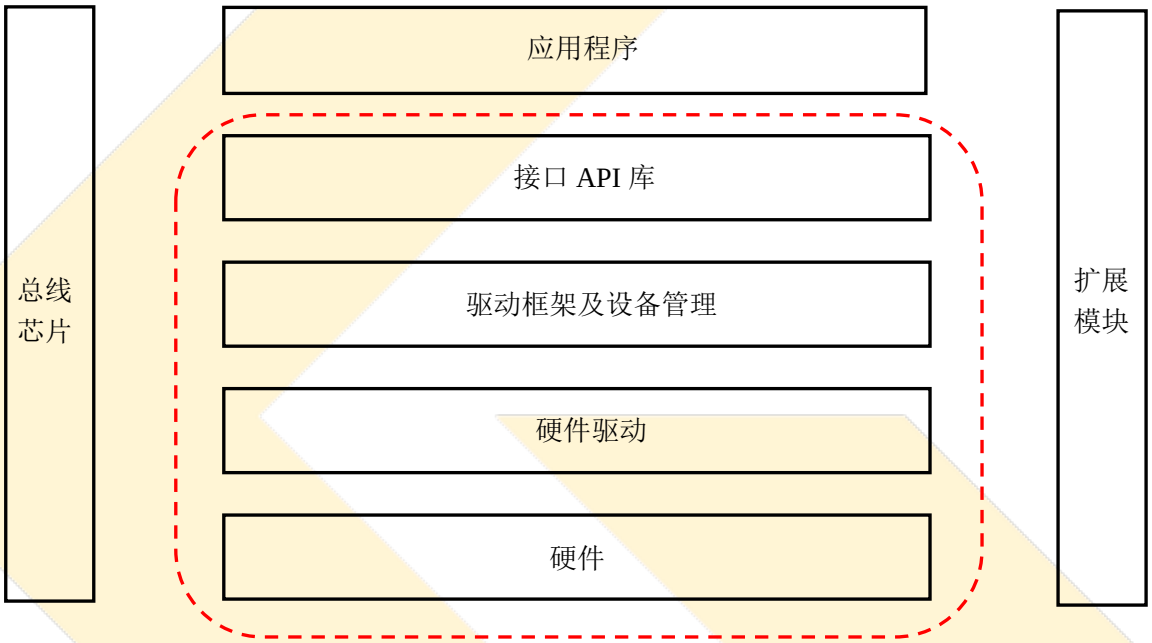
目录

第一章　SFC 简介

SFC 是一款卡片型现场总线控制器。可同时支持 EtherCat 总线、CAN 总线 CANopen 协议、RS485 总线 Modbus Rtu 协议/自由协议。主机本体含 6 点高速输入、6 点晶体管输出、2 路 12 位精度模拟量输入、1 路 12 位精度模拟量输出。最大可扩展 15 个模块。

第二章　应用工程简介

本应用工程使用嵌入式实时多线程操作系统 RT-Thread。在 Keil MDK5 开发环境下构建，是基于 C 语言二次开发的软件工程。以下为软件配图。



备注：红框部分以库的形式提供，应用程序提供二次开发模板

第三章 主要 API 说明

本部分 API 主要给出实际应用中会使用到的部分。

3.1 线程

3.1.1 线程创建函数

```
rt_thread_t rt_thread_create(const char *name,
                             void (*entry)(void *parameter),
                             void       *parameter,
                             rt_uint32_t stack_size,
                             rt_uint8_t  priority,
                             rt_uint32_t tick);
```

调用这个函数时，系统会从动态堆内存中分配一个线程句柄以及按照参数中指定的栈大小从动态堆内存中分配相应的空间。

参数	描述
入口参数	
name	线程的名称；线程名称的最大长度为 8 个字符多余部分会被自动截掉。
entry	线程入口函数。
parameter	线程入口函数参数。

stack_size	分配给线程的堆栈空间，正常 1-2K 即可。
priority	线程优先级，数值越大优先级越低，建议使用 20-30 之间的优先级。
tick	线程运行的时间片,单位毫秒。建议使用 1-2。
出口参数	
rt_thread_t	线程创建成功，返回线程句柄。创建失败，返回 0。

3.1.2 线程删除函数

rt_err_t rt_thread_delete(rt_thread_t thread)

调用该函数后，线程对象将会被移出线程队列并且从内核对象管理器中删除，线程占用的堆栈空间也会被释放，收回的空间将重新用于其他的内存分配。

参数	描述
入口参数	
thread	要删除的线程句柄
出口参数	
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.1.3 启动线程

rt_err_t rt_thread_startup(rt_thread_t thread)

当调用这个函数时，将把线程的状态更改为就绪状态，并放到相应优先级队列中等待调度。

参数	描述
入口参数	
rt_thread_t	线程句柄
出口参数	
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.1.4 获取当前线程

rt_thread_t rt_thread_self(void)

参数	描述
出口参数	
rt_thread_t	线程句柄

3.1.5 线程释放 CPU 控制

rt_err_t rt_thread_yield(void)

调用该函数后，线程让出处理器控制，供相同优先级的其他线程使用 CPU，等待下次调度。如果当前优先级只有这一个线程，则这个线程继续执行，不进行上下文切换动作。

参数	描述
出口参数	
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.1.6 线程休眠

rt_err_t rt_thread_mdelay(rt_int32_t ms)

当前线程释放 CPU 一段时间，在指定的时间到达后重新运行。**若线程运行中，无释放 CPU 的等待操作时，要求在循环的最后插入此函数，释放 CPU 以供其他更低优先级的线程获取 CPU 的控制权。**

参数	描述
入口参数	
ms	释放时间，单位 ms
出口参数	
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.1.7 恢复线程

```
rt_err_t rt_thread_resume (rt_thread_t thread)
```

让休眠的线程重新进入就绪状态。

参数	描述
入口参数	
rt_thread_t	线程句柄
出口参数	
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.2 定时器

3.2.1 定时器创建

```
rt_timer_t rt_timer_create(const char* name,
                    void (*timeout)(void* parameter),
                    void* parameter,
                    rt_tick_t time,
                    rt_uint8_t flag)
```

调用该函数接口后，内核首先从动态内存堆中分配一个定时器控制块，然后对该控制块进行基本的初始化。

参数	描述
入口参数	
name	定时器名称；最大长度为 8 个字符多余部分会被自动截掉
timeout	定时器超时函数
parameter	超时函数函数参数
time	定时器超时时间，单位 ms
flag	定时器创建时的参数，支持的值包括单次定时、周期定时等。
出口参数	
rt_timer_t	创建成功，返回定时器句柄。创建失败，返回 0。

flag 相关宏定义

RT_TIMER_FLAG_ONE_SHOT	单次定时
RT_TIMER_FLAG_PERIODIC	周期定时

3.2.2 定时器删除

```
rt_err_t rt_timer_delete(rt_timer_t timer)
```

调用这个函数接口后，系统会把这个定时器删除，然后释放其占用的系统资源。

参数	描述
入口参数	
rt_timer_t	定时器句柄
出口参数	
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.2.3 启动定时器

```
rt_err_t rt_timer_start(rt_timer_t timer)
```

当定时器被创建后，并不会被立即启动，必须在调用启动定时器函数接口后，才开始工作。

参数		描述
入口参数		
rt_timer_t	定时器句柄	
出口参数		
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败	

3.2.4 停止定时器

```
rt_err_t rt_timer_stop(rt_timer_t timer)
```

调用这个函数接口后停止定时器运行。

参数		描述
入口参数		
rt_timer_t	定时器句柄	
出口参数		
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败	

3.3 信号量

3.3.1 创建信号量

```
rt_sem_t rt_sem_create(const char *name,
                       rt_uint32_t value,
                       rt_uint8_t flag)
```

创建一个信号量，然后对该控制块进行基本的初始化工作。

参数		描述
入口参数		
name	信号量名称；最大长度为 8 个字符多余部分会被自动截掉	
value	信号量初始值	
flag	信号量标志，取如下数值：RT_IPC_FLAG_FIFO 或 RT_IPC_FLAG_PRIO	
出口参数		
rt_sem_t	创建成功，返回信号量句柄。创建失败，返回 0。	

3.3.2 删除信号量

```
rt_err_t rt_sem_delete(rt_sem_t sem)
```

调用这个函数时，系统将删除这个信号量。

参数		描述
入口参数		
rt_sem_t	信号量句柄	
出口参数		
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败	

3.3.3 获取信号量

```
rt_err_t rt_sem_take (rt_sem_t sem, rt_int32_t time)
```

线程获取信号量资源实例，当信号量值大于零时，线程将获得信号量，并且相应的信号量值会减 1，当为零时则需要进行等待。

参数		描述
入口参数		
	sem	信号量句柄
	time	等待时间，单位 ms
出口参数		
rt_err_t		RT_EOK 成功获取 -RT_ETIMEOUT 超时

3.3.4 无等待获取信号量

```
rt_err_t rt_sem_trytake(rt_sem_t sem)
```

无等待方式获取信号量。

参数		描述
入口参数		
	sem	信号量句柄
出口参数		
rt_err_t		RT_EOK 成功获取 -RT_ETIMEOUT 超时

3.3.5 释放信号时

```
rt_err_t rt_sem_release(rt_sem_t sem)
```

释放信号量可以唤醒挂起在该信号量上的线程。当信号量不为 0 时，信号量值会加 1。

参数		描述
入口参数		
	sem	信号量句柄
出口参数		
rt_err_t		返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.4 互斥量

3.4.1 创建互斥量

```
rt_mutex_t rt_mutex_create (const char* name, rt_uint8_t flag)
```

创建一个互斥量控制块，然后完成对该控制块的初始化工作。

参数		描述
入口参数		
	name	互斥量名称；最大长度为 8 个字符多余部分会被自动截掉
	flag	互斥量标志，取如下数值：RT_IPC_FLAG_FIFO 或 RT_IPC_FLAG_PRIO
出口参数		
rt_mutex_t		创建成功，返回互斥量句柄。创建失败，返回 0。

3.4.2 删除互斥量

```
rt_err_t rt_mutex_delete (rt_mutex_t mutex)
```

删除互斥量以释放系统资源。

参数		描述
入口参数		
	mutex	互斥量句柄
出口参数		
	rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败

3.4.3 获取互斥量

```
rt_err_t rt_mutex_take (rt_mutex_t mutex, rt_int32_t time)
```

线程获取了互斥量，那么线程就有了对该互斥量的所有权，即某一个时刻一个互斥量只能被一个线程持有。

参数		描述
入口参数		
	mutex	互斥量句柄
	time	等待时间，单位 ms
出口参数		
	rt_err_t	RT_EOK 成功获取 -RT_ETIMEOUT 超时

3.4.4 释放互斥量

```
rt_err_t rt_mutex_release(rt_mutex_t mutex)
```

当线程完成互斥资源的访问后，应尽快释放它占据的互斥量，使得其他线程能及时获取该互斥量。

参数		描述
入口参数		
	mutex	互斥量句柄
出口参数		
	rt_err_t	RT_EOK 成功获取 -RT_ETIMEOUT 超时

3.5 事件集

3.5.1 创建事件集

```
rt_event_t rt_event_create(const char* name, rt_uint8_t flag)
```

创建一个事件集控制块，然后对该事件集控制块进行基本的初始化。

参数		描述
入口参数		
	name	事件集名称；最大长度为 8 个字符多余部分会被自动截掉
	flag	事件集标志，取如下数值：RT_IPC_FLAG_FIFO 或 RT_IPC_FLAG_PRIO
出口参数		
	rt_event_t	创建成功，返回互斥量句柄。创建失败，返回 0。

3.5.2 删除事件集

```
rt_err_t rt_event_delete(rt_event_t event)
```

删除事件集对象控制块来释放系统资源。

参数		描述
入口参数		
	event	事件集句柄

出口参数

rt_err_t	RT_EOK 成功获取	-RT_ETIMEOUT 超时

3.5.3 接收事件

```
rt_err_t rt_event_recv(rt_event_t event,
                       rt_uint32_t set,
                       rt_uint8_t option,
                       rt_int32_t timeout,
                       rt_uint32_t* recved)
```

使用 32 位的无符号整数来标识事件集，每一位代表一个事件，因此一个事件集对象可同时等待接收 32 个事件，通过指定选择参数“逻辑与” 或“逻辑或” 来选择如何激活线程，使用“逻辑与” 参数表示只有当所有等待的事件都发生时才激活线程，而使用“逻辑或” 参数则表示只要有一个等待的事件发生就激活线程。

参数	描述
入口参数	
event	事件集句柄
set	接收线程感兴趣事件
option	接收选项
timeout	指定超时时间
recved	接收事件
出口参数	
rt_err_t	RT_EOK 成功获取 -RT_ETIMEOUT 超时 -RT_ERROR 错误

optipn 宏义

RT_EVENT_FLAG_AND	事件集逻辑与
RT_EVENT_FLAG_OR	事件集逻辑或
RT_EVENT_FLAG_CLEAR	清除事件集

3.5.4 发送事件

```
rt_err_t rt_event_send(rt_event_t event, rt_uint32_t set)
```

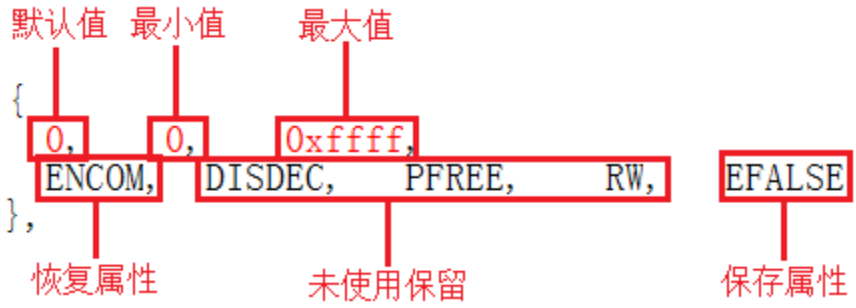
发送事件集中的一个或多个事件。

参数	描述
入口参数	
event	事件集句柄
set	发送一个或多个事件标志
出口参数	
rt_err_t	RT_EOK 成功获取 -RT_ERROR 错误

3.6 参数管理

基本参数就像是高楼大厦的基础，如果没有基础，就不会有高楼。一段程序，一个项目，一个工程都是如此，都要有一个基础，一个框架，而后通过不断地修改，不断的革新以达成自己想要的结果。为了方便用户使用，这里提供了两种参数管理方式：

1.每个参数拥有一个属性表，表包含参数最大值、最小值、默认值、恢复属性、保存属性等（其中位参数组，16 个位共享一个属性表）。并对所有参数分为以下几组：位参数组 2 组 PB、PKB，字参数组 2 组 PW、PKW，系统参数组 1 组 SYS(内核使用)。



以上为属性结构说明，具体详见 ParamTable.h 文件。

2.不配置参数属性表，只对参数的保存属性进行划分组，分别定义如下：位参数组 PB(不保存)，位参数组 PKB(掉电保存)，字参数组 PW(不保存)，字参数组 PKW(修改保存)，字参数组 PDW(掉电保存)，系统参数组 SYS(内核使用)。

这些参数组又对应到 Modbus-Rtu/Modbus-Tcp 通信地址上。如下表：

序号	参数组	通信类型	起始地址	结束地址
1	PB	0x	0	PBNUM*16-1
2	PKB	0x	PBNUM*16	PBNUM*16+PKBNUM*16-1
3	PW	4x	0	PWNUM-1
4	PKW	4x	PWNUM	PWNUM+PKWNUM-1
5	PDW	4x	PWNUM+PKWNUM	PWNUM+PKWNUM+PDWNUM-1
6	SYS	4x	0x8000	0x8000+SYSNUM-1

3.6.1 获取字参数组参数

UINT16 GetParamWord(UINT32 Gp,UINT32 Id)

通过参数组及组成员地址获取指定的字参数。

参数		描述
入口参数		
	Gp	参数组
	Id	组成员地址
出口参数		
	UINT16	返回单寄存器参数值
Gp 宏定义		
	PB	位参数组
	PKB	位参数组
	PW	字参数组
	PKW	字参数组
	SYS	系统参数组

UINT32 GetParamDWord(UINT32 Gp,UINT32 Id)

通过参数组及组成员地址获取指定的双字参数。即组成员地址(低 16 位)及下一地址(高 16 位)组成。

参数		描述
入口参数		
	Gp	参数组
	Id	组成员地址
出口参数		
	UINT32	返回双寄存器参数值

UINT16 GetWord(UINT16 Addr)

按通信地址的方式获取指定的字参数。

参数		描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
出口参数		
	UINT16	返回单寄存器参数值

UINT32 GetDWord(UINT16 Addr)

按通信地址的方式获取指定的双字参数。即指定地址(低 16 位)及下一地址(高 16 位)组成。

参数		描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
出口参数		
	UINT32	返回双寄存器参数值

INT16 GetInt16(UINT16 Addr)

按通信地址的方式获取指定的有符号字参数。

参数		描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
出口参数		
	INT16	返回单寄存器参数值

INT32 GetInt32(UINT16 Addr)

按通信地址的方式获取指定的有符号双字参数。通信地址(低 16 位)，下一地址(高 16 位)。

参数		描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
出口参数		
	INT32	返回双寄存器参数值

float GetFloat(UINT16 Addr)

按通信地址的方式获取指定的浮点双字参数。通信地址(低 16 位)，下一地址(高 16 位)。

参数		描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
出口参数		
	INT32	返回双寄存器参数值

3.6.2 设置字参数组参数

UINT8 SetParamWord(UINT32 Gp,UINT32 Id,UINT16 Val)

通过参数组及组成员地址，来设置字参数。

参数		描述
入口参数		
	Gp	参数组
	Id	组成员地址

	Val	单寄存器参数值
出口参数		
	UINT8	1 设置成功 0 设置失败

UINT8 SetParamDWord(UINT32 Gp,UINT32 Id,UINT32 Val)

通过参数组及组成员地址，来设置双字参数。即组成员地址(低 16 位)及下一地址(高 16 位)。

	参数	描述
入口参数		
	Gp	参数组
	Id	组成员地址
	Val	双寄存器参数值
出口参数		
	UINT8	1 设置成功 0 设置失败

UINT8 SetWord(UINT16 Addr,UINT16 Val)

按通信地址的方式，设置字参数。

	参数	描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
	Val	单寄存器参数值
出口参数		
	UINT8	1 设置成功 0 设置失败

UINT8 SetDWord(UINT16 Addr,UINT32 Val)

按通信地址的方式，设置双字参数。即通信地址(低 16 位)，下一地址(高 16 位)。

	参数	描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
	Val	双寄存器参数值
出口参数		
	UINT8	1 设置成功 0 设置失败

UINT8 SetInt16(UINT16 Addr,INT16 Val)

按通信地址的方式设置有符号字参数。

	参数	描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
	Val	单寄存器参数值
出口参数		
	UINT8	1 设置成功 0 设置失败

UINT8 SetInt32(UINT16 Addr,INT32 Val)

按通信地址的方式设置有符号双字参数。即通信地址(低 16 位)，下一地址(高 16 位)。

	参数	描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
	Val	双寄存器参数值
出口参数		

	UINT8	1 设置成功 0 设置失败

UINT8 SetFloat(UINT16 Addr,float Val)

按通信地址的方式设置浮点双字参数。即通信地址(低 16 位)，下一地址(高 16 位)。

参数		描述
入口参数		
	Addr	地址以 PW、PKW 组成的连续寄存器组
	Val	双寄存器参数值
出口参数		
	UINT8	1 设置成功 0 设置失败

3.6.3 获取位参数组参数

UINT16 GetParamBit(UINT32 Gp,UINT32 Id)

按参数组及成员地址获取位参数。

参数		描述
入口参数		
	Gp	参数组
	Id	组成员地址
出口参数		
	UINT16	返回位参数值

UINT16 GetBit(UINT16 Addr)

按通信地址的方式获取位参数。

参数		描述
入口参数		
	Addr	地址以 PB、PKB 组成的连续寄存器组
出口参数		
	UINT16	返回位参数值

3.6.4 设置位参数组参数

UINT8 SetParamBit(UINT32 Gp,UINT32 Id,UINT16 Val)

按参数组和成员地址设置位参数。

参数		描述
入口参数		
	Gp	参数组
	Id	组成员地址
	Val	位参数值
出口参数		
	UINT8	1 设置成功 0 设置失败

UINT8 SetBit(UINT16 Addr,UINT16 Val)

按通信地址的方式，设置位参数。

参数		描述
入口参数		
	Addr	地址以 PB、PKB 组成的连续寄存器组
	Val	位参数值

出口参数

UINT8　　1 设置成功　0 设置失败

3.6.5 系统 SYS 组参数说明

成员地址	说明
0	密码值（供应用使用）
1	软件版本（供应用使用）
2	机型（供应用使用）
3	软件日期（供应用使用）
4	Modbus-Rtu 从机串口波特率
5	Modbus-Rtu 从机站号
6	Modbus-Tcp IP 地址
7	Modbus-Tcp IP 地址
8	Modbus-Tcp 端口号

成员地址	说明
9	AI1 线性修正
10	AI2 线性修正
11	AO1 线性修正
12	AI1 偏移修正
13	AI2 偏移修正
14	AO1 偏移修正
15	AI1 滤波系数
16	AI2 滤波系数

3.6.6 掉电数据备份保持区

掉电数据备份保持区，是与 RTC 共用内部超级电容。在芯片掉电后，通过超级电容提供给备份区保持数据。备份区是一个 4KB 大小的空间。优点是数据可无限次数读写，并在掉电后通过超级电容进行数据保持，缺点则是因是超级电容保持，当超级电容没电时，掉电后数据则将丢失。因此备份区的使用，请使用者综合考虑后使用。

```
void WriteToBKRam(UINT16 Addr,UINT8 *Ptr,UINT16 Num)
```

将数据缓冲区的数据搬入备份保持区内。

参数　　描述

入口参数

Addr　　备份区偏移地址

Ptr　　数据缓冲区

Num　　搬入的字节数

```
void ReadFromBKRam(UINT16 Addr,UINT8 *Ptr,UINT16 Num)
```

将备份操持区内的数据搬入数据缓冲区。

参数　　描述

入口参数

Addr　　备份区偏移地址

Ptr　　数据缓冲区

Num　　搬出的字节数

3.7 开关量操作函数

3.7.1 获取输入开关量

```
UINT8 GetIoStu(UINT8 Cnl)
```

参数　　描述

入口参数

Cnl　　输入开关量通道

出口参数

UINT8　　0 断开　非 0 闭合

Cnl 宏定义

IN_X1	主机 DI1
.....	
IN_X6	主机 DI6
IN_X8	扩展模块 X0
.....	
IN_X128	扩展模块 Xn

3.7.2 设置输入开关量扩展模块滤波

```
void SetDiFlt(UINT8 SmpNo,UINT8 Flt)
```

参数	描述
入口参数	
SmpNo	输入开关量扩展模块编号
Flt	输入开关量滤波设置
Flt 宏定义	
EM_DI_SMP_08	滤波时间 0.8ms
EM_DI_SMP_16	滤波时间 1.6ms
EM_DI_SMP_32	滤波时间 3.2ms
EM_DI_SMP_64	滤波时间 6.4ms
EM_DI_SMP_128	滤波时间 12.8ms
EM_DI_SMP_256	滤波时间 25.6ms
EM_DI_SMP_512	滤波时间 51.2ms

3.7.3 设置输出开关量

```
void SetIoStu(UINT8 Cnl,UINT8 Val)
```

参数	描述
入口参数	
Cnl	输出开关量通道
Val	0 断开 非 0 闭合
Cnl 宏定义	
OUT_Y1	主机 DO1
.....	
OUT_Y6	主机 DO6
OUT_Y8	扩展模块 Y0
.....	
OUT_Y128	扩展模块 Yn

3.7.4 获取输出开关量

```
UINT8 GetOutStu(UINT8 Cnl)
```

参数	描述
入口参数	
Cnl	输出开关量通道
出口参数	
Val	0 断开 非 0 闭合
Cnl 宏定义	
OUT_Y1	主机 DO1

.....

OUT_Y6	主机 DO6
OUT_Y8	扩展模块 Y0

.....

OUT_Y128	扩展模块 Yn

3.7.5 获取高速输入脉冲值

UINT32 GetXinPul(UINT8 Cnl)

参数		描述
入口参数		
	Cnl	高速输入通道 X1-X6
出口参数		
	UINT32	获取脉冲输入值
Cnl 宏定义		
	IN_X1_PUL	主机 DI1 脉冲输入

.....

IN_X6_PUL	主机 DI6 脉冲输入

3.7.6 设置高速输入脉冲值

void SetXinPul(UINT8 Cnl,UINT32 Val)

参数		描述
入口参数		
	Cnl	高速输入通道 X1-X6
	Val	设置脉冲输入值

3.7.7 获取正交编码输入脉冲值

UINT32 GetEncPul(UINT8 Cnl)

参数		描述
入口参数		
	Cnl	高速输入通道 X1-X2；X4-X5 组成的 AB 正交通道
出口参数		
	UINT32	获取脉冲输入值 4 倍频
Cnl 宏定义		
	IN_ENC1_PUL	主机 X1-X2 组成的 AB 正交通道
	IN_ENC2_PUL	主机 X4-X5 组成的 AB 正交通道

3.7.8 设置正交编码输入脉冲值

void SetEncPul(UINT8 Cnl,UINT32 Val)

参数		描述
入口参数		
	Cnl	高速输入通道 X1-X2；X4-X5 组成的 AB 正交通道
	Val	设置脉冲输入值

3.7.9 设置 LED 显示模式

void SetLedMod(UINT8 Cnl,UINT8 Mode)

参数	描述
入口参数	
Cnl	主机 LED 通道
Mode	显示模式
Cnl 宏定义	
LED_RUN	运行灯
LED_COM	通信灯
LED_ERR	故障灯
Mode 宏定义	
LED_MODE_OFF	常灭
LED_MODE_ON	常亮
LED_MODE_SLOW	慢速闪烁
LED_MODE_QUICK	快速闪烁

3.7.10 获取输入开关量上升沿

UINT32 GetIoUp(UINT8 Cnl)

参数	描述
入口参数	
Cnl	输入开关量通道
出口参数	
UINT32	0 无上升沿 1 上升沿

注：本函数只能应用于 void SysActIdle(void)函数内，并且只在变化当次循环有效。

3.7.11 获取输入开关量下降沿

UINT32 GetIoDw(UINT8 Cnl)

参数	描述
入口参数	
Cnl	输入开关量通道
出口参数	
UINT32	0 无下降沿 1 下降沿

注：本函数只能应用于 void SysActIdle(void)函数内，并且只在变化当次循环有效。

3.7.12 获取输出开关量上升沿

UINT32 GetIoOutUp(UINT8 Cnl)

参数	描述
入口参数	
Cnl	输出开关量通道
出口参数	
UINT32	0 无上升沿 1 上升沿

注：本函数只能应用于 void SysActIdle(void)函数内，并且只在变化当次循环有效。

3.7.13 获取输出开关量下降沿

UINT32 GetIoOutDw(UINT8 Cnl)

参数		描述
入口参数		
	Cnl	输出开关量通道
出口参数		
	UINT32	0 无下降沿 1 下降沿

注：本函数只能应用于 void SysActIdle(void)函数内，并且只在变化当次循环有效。

3.8 模拟量操作函数

3.8.1 获取模拟量输入值

UINT16 GetAi0To10V(UINT8 Channel)

获取模拟量输入类型为 0-10V 的通道电压值。

参数		描述
入口参数		
	Cnl	模拟量输入通道
出口参数		
	UINT16	获取处理后的模拟量值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

AD_CHANNEL1	主机 AI1 模拟输入
AD_CHANNEL2	主机 AI2 模拟输入
AD_CHANNEL3	扩展模块 AI0
.....	
AD_CHANNEL32	扩展模块 AIn

UINT16 GetAi0To5V(UINT8 Channel)

获取模拟量输入类型为 0-5V 的通道电压值。

参数		描述
入口参数		
	Cnl	模拟量输入通道
出口参数		
	UINT16	获取处理后的模拟量值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

AD_CHANNEL1	主机 AI1 模拟输入
AD_CHANNEL2	主机 AI2 模拟输入
AD_CHANNEL3	扩展模块 AI0
.....	
AD_CHANNEL32	扩展模块 AIn

UINT16 GetAi1To5V(UINT8 Channel)

获取模拟量输入类型为 1-5V 的通道电压值。

参数		描述
入口参数		
	Cnl	模拟量输入通道
出口参数		

UINT16	获取处理后的模拟量值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

AD_CHANNEL1	主机 AI1 模拟输入
AD_CHANNEL2	主机 AI2 模拟输入
AD_CHANNEL3	扩展模块 AI0
.....	
AD_CHANNEL32	扩展模块 AIn

UINT16 GetAi0To20mA(UINT8 Channel)

获取模拟量输入类型为 0-20mA 的通道电流值。

参数	描述
入口参数	
Cnl	模拟量输入通道
出口参数	
UINT16	获取处理后的模拟量值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

AD_CHANNEL1	主机 AI1 模拟输入
AD_CHANNEL2	主机 AI2 模拟输入
AD_CHANNEL3	扩展模块 AI0
.....	
AD_CHANNEL32	扩展模块 AIn

UINT16 GetAi4To20mA(UINT8 Channel)

获取模拟量输入类型为 4-20mA 的通道电流值。

参数	描述
入口参数	
Cnl	模拟量输入通道
出口参数	
UINT16	获取处理后的模拟量值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

AD_CHANNEL1	主机 AI1 模拟输入
AD_CHANNEL2	主机 AI2 模拟输入
AD_CHANNEL3	扩展模块 AI0
.....	
AD_CHANNEL32	扩展模块 AIn

3.8.2 设置模拟量输入扩展模块滤波

void SetAiSmp(UINT8 Channel,UINT8 Smp)

参数	描述
入口参数	
Channel	扩展模块模拟量输入通道
Smp	模拟量输入滤波设置

Flt 宏定义

EM_AI_SMP_2	采样 2 次
EM_AI_SMP_4	采样 4 次
EM_AI_SMP_8	采样 8 次
EM_AI_SMP_16	采样 16 次

EM_AI_SMP_32	采样 32 次
EM_AI_SMP_64	采样 64 次
EM_AI_SMP_128	采样 128 次
EM_AI_SMP_256	采样 256 次

3.8.3 设置模拟量输入扩展模块类型

```
void SetAiMode(UINT8 Channel,UINT8 Mode)
```

参数	描述
入口参数	
Channel	模拟量输入扩展模块通道
Mode	模拟量输入类型

Channel 宏定义

AD_CHANNEL3	扩展模块模拟量输入 AI0
.....	
AD_CHANNEL32	扩展模块模拟量输入 AIn

Mode 宏定义

EM_AI_MODE_4_20_MA	输入类型 DC4-20mA
EM_AI_MODE_0_20_MA	输入类型 DC0-20mA
EM_AI_MODE_1_5_V	输入类型 DC1-5V
EM_AI_MODE_0_5_V	输入类型 DC0-5V
EM_AI_MODE_0_10_V	输入类型 DC0-10V

3.8.4 设置模拟量输出值

```
void SetAo0To10V(UINT8 Channel,UINT16 Vol)
```

设置模拟量输出类型为 0-10V 的通道电压值。

参数	描述
入口参数	
Cnl	模拟量输出通道
Vol	设置模拟量输出值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

DA_CHANNEL1	主机 AO1 模拟输出
.....	
DA_CHANNEL2	扩展模块 AO0
DA_CHANNEL32	扩展模块 AOn

```
void SetAo0To5V(UINT8 Channel,UINT16 Vol)
```

设置模拟量输出类型为 0-5V 的通道电压值。

参数	描述
入口参数	
Cnl	模拟量输出通道
Vol	设置模拟量输出值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

DA_CHANNEL1	主机 AO1 模拟输出
.....	
DA_CHANNEL2	扩展模块 AO0
DA_CHANNEL32	扩展模块 AOn

void SetAo1To5V(UINT8 Channel,UINT16 Vol)

设置模拟量输出类型为 1-5V 的通道电压值。

参数		描述
入口参数		
	Cnl	模拟量输出通道
	Vol	设置模拟量输出值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

DA_CHANNEL1	主机 AO1 模拟输出
.....	
DA_CHANNEL2	扩展模块 AO0
DA_CHANNEL32	扩展模块 AOn

void SetAo0To20mA(UINT8 Channel,UINT16 Cir)

设置模拟量输出类型为 0-20mA 的通道电流值。

参数		描述
入口参数		
	Cnl	模拟量输出通道
	Cir	设置模拟量输出值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

DA_CHANNEL1	主机 AO1 模拟输出
.....	
DA_CHANNEL2	扩展模块 AO0
DA_CHANNEL32	扩展模块 AOn

void SetAo4To20mA(UINT8 Channel,UINT16 Cir)

设置模拟量输出类型为 4-20mA 的通道电流值。

参数		描述
入口参数		
	Cnl	模拟量输出通道
	Cir	设置模拟量输出值，2 位缩放，即 100 表示 1.00

Cnl 宏定义

DA_CHANNEL1	主机 AO1 模拟输出
.....	
DA_CHANNEL2	扩展模块 AO0
DA_CHANNEL32	扩展模块 AOn

3.8.5 设置模拟量输出扩展模块类型

void SetAoMode(UINT8 Channel,UINT8 Mode)

参数		描述
入口参数		
	Channel	模拟量输出扩展模块通道
	Mode	模拟量输出类型

Channel 宏定义

DA_CHANNEL2	扩展模块模拟量输出 AO0
.....	
DA_CHANNEL32	扩展模块模拟量输出 AOn

Mode 宏定义

EM_AO_MODE_4_20_MA	输入类型 DC4-20mA
EM_AO_MODE_0_20_MA	输入类型 DC0-20mA
EM_AO_MODE_1_5_V	输入类型 DC1-5V
EM_AO_MODE_0_5_V	输入类型 DC0-5V
EM_AO_MODE_0_10_V	输入类型 DC0-10V

3.8.6 获取热电阻输入值

```
INT16 GetRCVal(UINT8 Channel)
```

参数	描述
入口参数	
Cnl	热电阻输入通道
出口参数	
UINT16	获取处理后的热电阻输入值,1 位缩放，即 10 表示 1.0，有正负值

Cnl 宏定义

RC_CHANNEL1	扩展模块 RC0
.....	
RC_CHANNEL32	扩展模块 RCn

3.8.7 设置热电阻输入扩展模块滤波

```
void SetRcSmp(UINT8 Channel,UINT8 Smp)
```

参数	描述
入口参数	
Channel	扩展模块模拟量输入通道
Smp	模拟量输入滤波设置

Flt 宏定义

EM_AI_SMP_2	采样 2 次
EM_AI_SMP_4	采样 4 次
EM_AI_SMP_8	采样 8 次
EM_AI_SMP_16	采样 16 次
EM_AI_SMP_32	采样 32 次
EM_AI_SMP_64	采样 64 次
EM_AI_SMP_128	采样 128 次
EM_AI_SMP_256	采样 256 次

3.8.8 设置热电阻输入类型

```
void SetRCMode(UINT8 Channel,UINT8 Mode)
```

参数	描述
入口参数	
Channel	模拟量输入扩展模块通道
Mode	模拟量输入类型

Channel 宏定义

RC_CHANNEL1	扩展模块模拟量输入 RC0
.....	
RC_CHANNEL32	扩展模块模拟量输入 RCn

Mode 宏定义

EM_RC_MODE_PT100	传感器类型 PT100
EM_RC_MODE_PT1000	传感器类型 PT1000
EM_RC_MODE_CU50	传感器类型 CU50
EM_RC_MODE_CU100	传感器类型 CU100

3.8.9 获取热电偶输入值

```
UINT16 GetTCVal(UINT8 Channel)
```

参数	描述
入口参数	
Cnl	热电偶输入通道
出口参数	
UINT16	获取处理后的热电偶输入值，1 位缩放，即 10 表示 1.0，有正负值

Cnl 宏定义

TC_CHANNEL1	扩展模块 TC0
.....	
TC_CHANNEL32	扩展模块 TCn

3.8.10 设置热电偶输入扩展模块滤波

```
void SetTCSmp(UINT8 Channel,UINT8 Smp)
```

参数	描述
入口参数	
Channel	扩展模块模拟量输入通道
Flt	模拟量输入滤波设置

Flt 宏定义

EM_AI_SMP_2	采样 2 次
EM_AI_SMP_4	采样 4 次
EM_AI_SMP_8	采样 8 次
EM_AI_SMP_16	采样 16 次
EM_AI_SMP_32	采样 32 次
EM_AI_SMP_64	采样 64 次
EM_AI_SMP_128	采样 128 次
EM_AI_SMP_256	采样 256 次

3.8.11 设置热电偶输入扩展模块类型

```
void SetTCMode(UINT8 Channel,UINT8 Mode)
```

参数	描述
入口参数	
Channel	模拟量输入扩展模块通道
Mode	模拟量输入类型

Channel 宏定义

TC_CHANNEL1	扩展模块模拟量输出 TC0
.....	
TC_CHANNEL32	扩展模块模拟量输出 TCn

Mode 宏定义

EM_TC_MODE_S	传感器类型 S
EM_TC_MODE_K	传感器类型 K
EM_TC_MODE_T	传感器类型 T
EM_TC_MODE_E	传感器类型 E
EM_TC_MODE_J	传感器类型 J
EM_TC_MODE_B	传感器类型 B
EM_TC_MODE_N	传感器类型 N
EM_TC_MODE_R	传感器类型 R
EM_TC_MODE_WRE3_25	传感器类型 Wre3/25
EM_TC_MODE_WRE5_26	传感器类型 Wre5/26
EM_TC_MODE_0_20_MV	传感器类型 0-20mV
EM_TC_MODE_0_50_MV	传感器类型 0-50mV
EM_TC_MODE_0_100_MV	传感器类型 0-100mV

3.8.12 获取温湿度模块温度值

```
UINT16 GetDTTVal(UINT8 Channel)
```

参数	描述
入口参数	
Cnl	温度输入通道
出口参数	
UINT16	获取处理后的温度输入值

Cnl 宏定义

DT_CHANNEL1	扩展模块 DT0
.....	
DT_CHANNEL32	扩展模块 DTn

3.8.13 获取温湿度模块湿度值

```
UINT16 GetDTHVal(UINT8 Channel)
```

参数	描述
入口参数	
Cnl	湿度输入通道
出口参数	
UINT16	获取处理后的湿度输入值

Cnl 宏定义

DT_CHANNEL1	扩展模块 DT0
.....	
DT_CHANNEL32	扩展模块 DTn

3.8.14 设置温湿度输入扩展模块类型

```
void SetDtMode(UINT8 Channel,UINT8 Mode)
```

参数	描述
入口参数	
Channel	模拟量输入扩展模块通道
Mode	模拟量输入类型

Channel 宏定义

DT_CHANNEL1 扩展模块模拟量输出 DT0
.....
DT_CHANNEL32 扩展模块模拟量输出 DTn
Mode 宏定义
EM_DT_MODE_DS 传感器类型 DS
EM_DT_MODE_SHT 传感器类型 SHT

3.8.15 获取称重模块实时重量值

UINT16 GetWDVal(UINT8 Channel)

参数		描述
入口参数		
Cnl	重量输入通道	
出口参数		
UINT16	获取处理后的重量输入值	

Cnl 宏定义
WG_CHANNEL1 扩展模块通道 WG0
.....
WG_CHANNEL32 扩展模块通道 WGn

3.8.16 设置称重输入扩展模块滤波

void SetWgSmp(UINT8 Channel,UINT8 Smp)

参数		描述
入口参数		
Channel	扩展模块模拟量输入通道	
Flt	模拟量输入滤波设置	

Flt 宏定义

EM_WG_SMP_625	转换速率 6.25Hz
EM_WG_SMP_1250	转换速率 12.5Hz
EM_WG_SMP_2500	转换速率 25Hz
EM_WG_SMP_5000	转换速率 50Hz
EM_WG_SMP_10000	转换速率 100Hz
EM_WG_SMP_20000	转换速率 200Hz
EM_WG_SMP_50000	转换速率 500Hz
EM_WG_SMP_100000	转换速率 1000Hz

3.9 总线操作函数

3.9.1 总线扫描

uint8_t FB_CONFIG(uint8_t type)

参数		描述
入口参数		
type	总线类型 0-ECAT 1-CAN	
出口参数		
uint8_t	0-扫描成功 1-扫描失败 2-类型错误	

3.9.2 获取节点数

```
uint8_t FB_NODE_COUNT(uint8_t type)
```

参数		描述
入口参数		
type	总线类型	0-ECAT 1-CAN
出口参数		
uint8_t	总线节点数	

3.9.3 启动总线通信

```
uint8_t FB_START(uint8_t type)
```

参数		描述
入口参数		
type	总线类型	0-ECAT 1-CAN
出口参数		
uint8_t	0-成功	1-失败

3.9.4 设置同步周期

```
uint8_t FB_SYNC_CYCLE(uint32_t cycle)
```

参数		描述
入口参数		
cycle	周期时间	单位 ns
出口参数		
uint8_t	0-成功	1-失败

3.9.5 获取从站厂商 ID

```
uint8_t FB_GET_ECAT_VID(uint8_t slv, uint32_t *vid)
```

参数		描述
入口参数		
slv	从站序号	
*vid	ID 指针	
出口参数		
uint8_t	0-成功	1-失败

3.9.6 获取从站产品 ID

```
uint8_t FB_GET_ECAT_PID(uint8_t slv, uint32_t *pid)
```

参数		描述
入口参数		
slv	从站序号	
*pid	ID 指针	
出口参数		
uint8_t	0-成功	1-失败

3.9.7 获取 EtherCat 状态

```
uint8_t FB_ECAT_STATUS(void)
```

参数		描述
出口参数		
	uint8_t	b1-b0 ESM 状态 b4 从站掉线 b5 从站报警 b6 PDO 错误

3.9.8 SDO 读对象字典

```
uint32_t FB_SDO_READ(uint8_t slave, uint16_t idx, uint8_t subidx, uint8_t *len, uint8_t *dat)
```

参数		描述
入口参数		
	slave	从站序号
	idx	对象字典索引
	subidx	对象字典子索引
	*len	长度指针
	*dat	数据指针
出口参数		
	uint32_t	0-成功 1-失败 其他-中止代码

3.9.9 SDO 写对象字典

```
uint32_t FB_SDO_WRITE(uint8_t slave, uint16_t idx, uint8_t subidx, uint8_t len, uint8_t *dat)
```

参数		描述
入口参数		
	slave	从站序号
	idx	对象字典索引
	subidx	对象字典子索引
	*len	长度指针
	*dat	数据指针
出口参数		
	uint32_t	0-成功 1-失败 其他-中止代码

3.9.10 主站 RS485 初始化

```
uint8_t FB_RS485_INIT(uint8_t par, uint8_t stopbits, uint32_t baudrate)
```

参数		描述
入口参数		
	par	校验 0-无 1-奇 2-偶
	stopbits	停止位 0-1bit 1-2bit
	baudrate	波特率
出口参数		
	uint8_t	0-成功 1-失败

3.9.11 Modbus-Rtu 位读操作

```
uint8_t FB_MB_COIL_READ(uint8_t slv, uint16_t addr, uint8_t num, uint8_t *pdat,
                        uint16_t timeout)
```

参数	描述
入口参数	
slv	从站地址
addr	位地址
num	操作个数
*pdat	数据指针
timeout	超时设定
出口参数	
uint8_t	0-成功 1-失败

3.9.12 Modbus-Rtu 位写操作

```
uint8_t FB_MB_COIL_WRITE(uint8_t slv, uint16_t addr, uint8_t num,uint8_t *pdat, uint16_t timeout)
```

参数	描述
入口参数	
slv	从站地址
addr	位地址
num	操作个数
*pdat	数据指针
timeout	超时设定
出口参数	
uint8_t	0-成功 1-失败

3.9.13 Modbus-Rtu 寄存器读操作

```
uint8_t FB_MB_REG_READ(uint8_t slv, uint16_t addr, uint8_t num, uint8_t *pdat, uint16_t timeout)
```

参数	描述
入口参数	
slv	从站地址
addr	寄存器地址
num	操作个数
*pdat	数据指针
timeout	超时设定
出口参数	
uint8_t	0-成功 1-失败

3.9.14 Modbus-Rtu 寄存器写操作

```
uint8_t FB_MB_REG_WRITE(uint8_t slv, uint16_t addr, uint8_t num, uint8_t *pdat, uint16_t timeout)
```

参数	描述
入口参数	
slv	从站地址
addr	寄存器地址
num	操作个数

*pdat	数据指针
timeout	超时设定

出口参数

uint8_t	0-成功 1-失败

3.10 运动控制

3.10.1 设置轴类型

```
uint8_t MC_AXIS_TYPE(uint8_t axis, int32_t src, uint8_t fb_no)
```

参数	描述

入口参数

axis	轴号
src	0-虚拟位置轴 1-ECAT 位置控制
fb_no	总线序号

出口参数

uint8_t	0-成功 1-失败 2-轴号超出 3-总线错误

3.10.2 获取轴类型

```
uint8_t MC_GET_AXIS_TYPE(uint8_t axis, uint8_t *fbno)
```

参数	描述

入口参数

axis	轴号
fb_no	总线序号指针

出口参数

uint8_t	0-虚拟位置轴 1-ECAT 位置控制 0xFF-未定义 0xFE-轴号超出

3.10.3 设置驱动 PDO 文件

```
uint8_t MC_PDO_CONFIG(uint8_t axis, int32_t src)
```

参数	描述

入口参数

axis	轴号
src	PDO 文件编号

出口参数

uint8_t	0-成功 2-轴号超出 0xFF-文件不存在

3.10.4 查询轴空闲状态

```
uint8_t MC_IDLE(uint8_t AxisSel)
```

参数	描述

入口参数

AxisSel	轴号

出口参数

uint8_t	0-忙 1-空闲 2-轴号超出

3.10.5 等待轴空闲

```
void MC_WAIT_IDLE(uint8_t AxisSel)
```

参数		描述
入口参数		
AxisSel	轴号	

3.10.6 设定轴使能

```
uint8_t MC_ENABLE(uint8_t axis, uint8_t on_off)
```

参数		描述
入口参数		
axis	轴号	
on_off	1-使能 0-关使能	
出口参数		
uint8_t	0-成功 2-轴号超出	

3.10.7 设置轴单位量

```
uint8_t MC_UNITS(uint8_t axis, double UNITS)
```

参数		描述
入口参数		
axis	轴号	
UINTS	范围 0.00001～100000	
出口参数		
uint8_t	0-成功 1-单位超出 2-轴号超出	

3.10.8 取消轴运动

```
uint8_t MC_CANCEL(uint8_t axis, int32_t mode)
```

参数		描述
入口参数		
axis	轴号	
mode	0-以停车减速停车 1-立即停车 2-以当前减速停车	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.9 增量式移动轴

```
uint8_t MC_MOVE(uint8_t axis, int32_t Length)
```

参数		描述
入口参数		
axis	轴号	
Length	移动距离 单位 UINTS	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.10 绝对式移动轴

```
uint8_t MC_MOVEABS(uint8_t axis, int32_t Length)
```

参数		描述
入口参数		
axis	轴号	
Length	移动距离 单位 UINTS	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.11 连续移动

```
uint8_t MC_MOVE_CO(uint8_t axis, int8_t dir)
```

参数		描述
入口参数		
axis	轴号	
dir	移动方向 1-正向 -1-反向 其他-停止	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.12 设置轴加速度

```
uint8_t MC_ACCEL(uint8_t axis, uint32_t Accel)
```

参数		描述
入口参数		
axis	轴号	
Accel	加速度 单位 UINTS/s/s	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.13 设置轴减速度

```
uint8_t MC_DECEL(uint8_t axis, uint32_t Decel)
```

参数		描述
入口参数		
axis	轴号	
Decel	加速度 单位 UINTS/s/s	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.14 取消轴减速度

```
uint8_t MC_CANCEL_DECEL(uint8_t axis, uint32_t Decel)
```

参数		描述
入口参数		
axis	轴号	
Decel	加速度 单位 UINTS/s/s	
出口参数		

uint8_t 0-成功 1-失败 2-轴号超出

3.10.15 设置轴速度

uint8_t MC_SPEED(uint8_t axis, int32_t Speed)

参数		描述
入口参数		
axis	轴号	
Speed	速度 单位 UINTS/s	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.16 定义轴位置

uint8_t MC_DEF_POS(uint8_t axis, int32_t pos)

参数		描述
入口参数		
axis	轴号	
pos	速度 单位 UINTS	
出口参数		
uint8_t	0-成功 1-失败 2-轴号超出	

3.10.17 获取轴位置

int64_t MC_GET_CMD_POS(uint8_t axis)

参数		描述
入口参数		
axis	轴号	
出口参数		
int64_t	位置 单位 UINTS	

3.10.18 获取轴反馈位置

int64_t MC_GET_FBK_POS(uint8_t axis)

参数		描述
入口参数		
axis	轴号	
出口参数		
int64_t	位置 单位 UINTS	

3.10.19 获取轴电机实际反馈位置

int32_t MC_GET_FBK_POS_MOTOR(uint8_t axis)

参数		描述
入口参数		
axis	轴号	
出口参数		
int32_t	位置 单位 UINTS	

3.10.20 获取轴单位量

```
double MC_GET_UNITS(uint8_t axis)
```

参数		描述
入口参数		
axis	轴号	
出口参数		
double	单位量	

3.10.21 获取轴设置速度

```
int32_t MC_GET_SPEED_SETVAL(uint8_t axis)
```

参数		描述
入口参数		
axis	轴号	
出口参数		
int32_t	速度 单位 UINTS/s	

3.10.22 获取轴指令速度

```
int32_t MC_GET_SPEED(uint8_t axis)
```

参数		描述
入口参数		
axis	轴号	
出口参数		
int32_t	速度 单位 UINTS/s	

3.10.23 获取轴反馈速度

```
int32_t MC_GET_SPEED_MOTOR(uint8_t axis)
```

参数		描述
入口参数		
axis	轴号	
出口参数		
int32_t	速度 单位 UINTS/s	

3.10.24 获取轴反馈转矩

```
int16_t MC_MTORQUE(uint8_t axis)
```

参数		描述
入口参数		
axis	轴号	
出口参数		
int16_t	转矩 单位 0.1%	

3.10.25 获取轴加速度

```
int32_t MC_GET_ACCEL(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	int32_t	速度 单位 UINTS/s/s

3.10.26 获取轴减速度

```
int32_t MC_GET_DECEL(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	int32_t	速度 单位 UINTS/s/s

3.10.27 获取轴的取消速度

```
int32_t MC_GET_CANCEL_DECEL(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	int32_t	速度 单位 UINTS/s/s

3.10.28 获取轴报警状态

```
uint8_t MC_STATUS_ALARM(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	uint8_t	0-无报警 1-报警 2-轴号超出

3.10.29 获取轴使能状态

```
uint8_t MC_STATUS_SRVON(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	uint8_t	0-未使能 1-使能 2-轴号超出

3.10.30 获取轴定位完成状态

```
uint8_t MC_STATUS_INPOS(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
uint8_t		0-未完成 1-完成 2-轴号超出

3.10.31 获取轴位置超限状态

```
uint8_t MC_STATUS_LMT(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
uint8_t		0-未超限 1-超限 2-轴号超出

3.10.32 获取轴跟随误差超出状态

```
uint8_t MC_STATUS_OF(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
uint8_t		0-未超出 1-超出 2-轴号超出

3.10.33 轴报警复位

```
uint8_t MC_ALARM_RESET(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
uint8_t		0-成功 2-轴号超出

3.10.34 轴控制字设定

```
uint8_t MC_CONTROLWORD(uint8_t axis, uint16_t ctw)
```

参数		描述
入口参数		
	axis	轴号
	ctw	控制字
出口参数		
uint8_t		0-成功 2-轴号超出

3.10.35 获取轴控制字

```
uint16_t MC_GET_CONTROLWORD(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	uint16_t	控制字

3.10.36 轴状态字读取

```
uint16_t MC_STATUSWORD(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	uint16_t	控制字

3.10.37 轴操作模式设定

```
uint8_t MC_OPERATION_MODE(uint8_t axis, int8_t mod)
```

参数		描述
入口参数		
	axis	轴号
	mod	模式
出口参数		
	uint8_t	0-成功 2-轴号超出

3.10.38 获取轴操作模式

```
int8_t MC_OPERATION_MODE_DISPLAY(uint8_t axis)
```

参数		描述
入口参数		
	axis	轴号
出口参数		
	uint8_t	操作模式

3.10.39 SDO 对象字典读取

```
uint32_t MC_SDO_READ(uint8_t axis, uint16_t idx, uint8_t subidx, uint8_t *len, uint8_t *dat)
```

参数		描述
入口参数		
	axis	轴号
	idx	对象字典索引
	subidx	对象字典子索引
	*len	长度指针
	*dat	数据指针
出口参数		
	uint32_t	0-成功 1-失败 其他-中止代码

3.10.40 SDO 对象字典写

```
uint32_t MC_SDO_WRITE(uint8_t axis, uint16_t idx, uint8_t subidx,
                      uint8_t len, uint8_t *dat)
```

参数	描述
入口参数	
axis	轴号
idx	对象字典索引
subidx	对象字典子索引
*len	长度指针
*dat	数据指针
出口参数	
uint32_t	0-成功 1-失败 其他-中止代码

3.11 其他操作函数

3.11.1 设置 Modbus-Tcp IP

```
void SetEthernetIpByNum(UINT32 Ip)
```

通过无符号 32 位整形进行 IP 设置。

参数	描述
入口参数	
Ip	IP 地址，如 192.168.1.30 分别为 b31-b24.b23-b16.b15-b8.b7-b0

```
void SetEthernetIpByString(INT8 *Ptr)
```

通过字符串进行 IP 设置。

参数	描述
入口参数	
Ptr	IP 地址，字符串如”192.168.1.30”

3.11.2 设置 Modbus-Tcp 子网掩码

```
void SetEthernetMaskByNum(UINT32 Mask)
```

通过无符号 32 位整形进行子网掩码设置。

参数	描述
入口参数	
Mask	子网掩码，如 255.255.255.0 分别为 b31-b24.b23-b16.b15-b8.b7-b0

```
void SetEthernetMaskByString(INT8 *Ptr)
```

通过字符串进行子网掩码设置。

参数	描述
入口参数	
Ptr	子网掩码，字符串如”255.255.255.0”

3.11.3 设置 Modbus-Tcp 网关

```
void SetEthernetGwByNum(UINT32 Gw)
```

通过无符号 32 位整形进行网关设置。

参数		描述
入口参数		
	Gw	网关，如 192.168.1.1 分别为 b31-b24.b23-b16.b15-b8.b7-b0

```
void SetEthernetGwByString(INT8 *Ptr)
```

通过字符串进行网关设置。

参数		描述
入口参数		
	Ptr	网关，字符串如”192.168.1.1”

3.11.4 设置从站 Modbus-Rtu 波特率

```
void SetModbusBtr(UINT32 Ptr)
```

参数		描述
入口参数		
	Ptr	波特率值

从站的通信格式固定为：数据位-8bits 停止位-1bit 无校验

3.11.5 设置从站 Modbus-Rtu 站号

```
void SetModbusAddr(UINT8 Addr)
```

参数		描述
入口参数		
	Addr	从站地址

3.11.6 设置代码读保护

```
void SetFlashReadprotection(void)
```

芯片级代码保护，调用此函数后，芯片将禁止通过仿真器进行读写代码、仿真等操作。可通过专门的厂家软件进行芯片保护消除，但消除的同时，代码也随之擦除。

3.11.7 应用层密钥安装

```
void SetupAppKeyPrt(UINT32 UserKey)
```

参数		描述
入口参数		
	UserKey	用户密钥

3.11.8 应用层密钥判断

```
UINT8 DetAppKeyPrt(UINT32 UserKey)
```

参数		描述
入口参数		
	UserKey	用户密钥
出口参数		
	UINT8	0 密钥错误 非 0 密钥正确

3.11.9 周期定时函数

```
UINT32 GetIdleTB(UINT32 Flg)
```

参数		描述
入口参数		
Flg	定时时基	
出口参数		
UINT32	0 未到达 非 0 到达	

Flg 宏定义

TB_2MS	2 毫秒
TB_4MS	4 毫秒
TB_8MS	8 毫秒
TB_16MS	16 毫秒
TB_32MS	32 毫秒
TB_64MS	64 毫秒
TB_128MS	128 毫秒
TB_256MS	256 毫秒
TB_512MS	512 毫秒
TB_1S	1 秒
TB_2S	2 秒
TB_4S	4 秒
TB_8S	8 秒

注：本函数只能应用于 void SysActIdle(void)函数内，并且只在变化当次循环有效。

3.11.10 获取实时时间戳

```
time_t time(time_t *t)
```

参数		描述
入口参数		
time_t	实时时间戳指针	
出口参数		
time_t	实时时间戳	

3.11.11 获取实时时间

```
struct tm* localtime(const time_t* t)
```

参数		描述
入口参数		
time_t	实时时间戳指针	
出口参数		
struct tm	实时时间结构体指针	

```
struct tm
{
  int tm_sec;           /* Seconds. [0-60] (1 leap second) */
  int tm_min;           /* Minutes. [0-59] */
  int tm_hour;          /* Hours.   [0-23] */
  int tm_mday;          /* Day.     [1-31] */
  int tm_mon;           /* Month.   [0-11] */
```

```
  int tm_year;          /* Year - 1900. */
  int tm_wday;          /* Day of week. [0-6] */
  int tm_yday;          /* Days in year.[0-365] */
  int tm_isdst;         /* DST.      [-1/0/1]*/

  long int tm_gmtoff;       /* Seconds east of UTC.  */
  const char *tm_zone;      /* Timezone abbreviation.  */
};
```

注：获取的 Year 需要加上 1900 才是当前年份，Month 需要加上 1 才是当前月份。

3.11.12 设置实时时间年月日

```
rt_err_t set_date(rt_uint32_t year, rt_uint32_t month, rt_uint32_t day)
```

参数		描述
入口参数		
year	实时时间-年	
month	实时时间-月	
day	实时时间-日	
出口参数		
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败	

3.11.13 设置实时时间时分秒

```
rt_err_t set_time(rt_uint32_t hour, rt_uint32_t minute, rt_uint32_t second)
```

参数		描述
入口参数		
hour	实时时间-时	
minute	实时时间-分	
second	实时时间-秒	
出口参数		
rt_err_t	返回 RT_EOK 操作成功，返回-RT_ERROR 操作失败	

3.11.14 启动用户定时器

```
void StartUserTimer(UINT16 Tick)
```

系统提供的定时器时基为 1ms，当用户需要应用小于 1ms 的定时器，可调用此函数启动指定时间的定时器。超时处理在 void UserTimerDeal(void)此函数中进行处理。

参数		描述
入口参数		
Tick	定时时间，单位 1us	

3.11.15 停止用户定时器

```
void StopUserTimer(void)
```

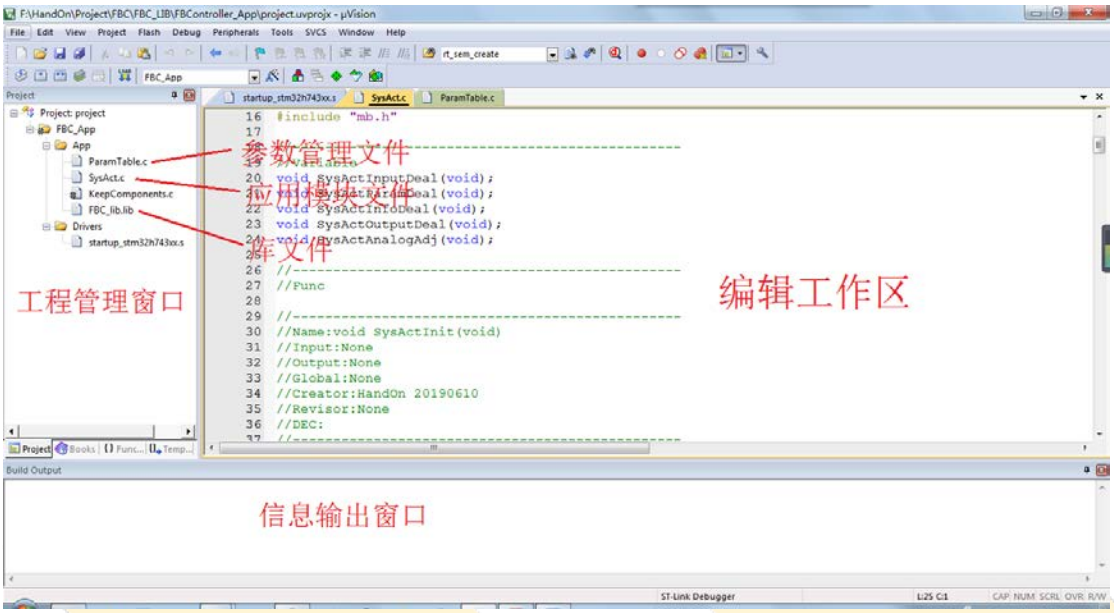
停止用户定时器，超时处理函数也相应的停止处理。

第四章　应用软件

4.1 开发环境

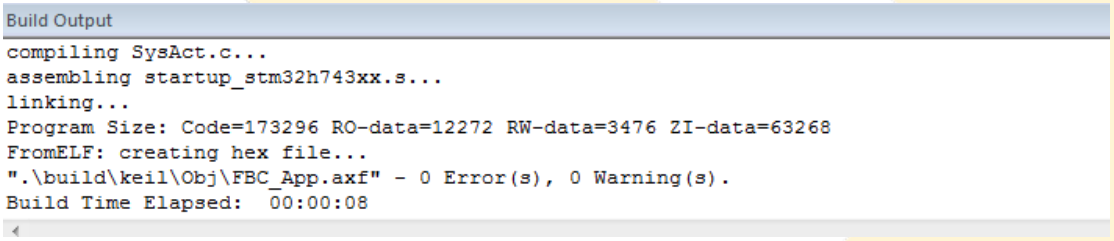
本应用工程使用的开发环境为 Keil5，请使用者自行购买安装使用。并对开发环境进行熟悉。

4.2 工程界面说明



4.3 工程编译

点击工程界面  按钮进行工程编译。成功编译后信息输出窗口出现如下信息：



4.4 工程下载

将仿真器与 SFC 控制器正确连接并供电后，点击工程界面  按钮将工程下载到 SFC。断开 SFC 控制器电源，再重新上电 SFC 即运行新下载的工程程序。

第五章　应用示例

```
//IN
#define IN_X1_ADDR       PB,0
#define IN_X2_ADDR       PB,1
#define IN_X3_ADDR       PB,2
#define IN_X4_ADDR       PB,3
#define IN_X5_ADDR       PB,4
#define IN_X6_ADDR       PB,5

//OUT
#define OUT_Y1_ADDR     PB,8
#define OUT_Y2_ADDR     PB,9
#define OUT_Y3_ADDR     PB,10
#define OUT_Y4_ADDR     PB,11
#define OUT_Y5_ADDR     PB,12
#define OUT_Y6_ADDR     PB,13

//Ain、Aout
#define AI1_VIN_ADDR     PW,4
#define AI2_IIN_ADDR      PW,5
#define AO1_IOUT_ADDR   PW,6

//脉冲
#define X1_PUL            PW,16
#define X2_PUL            PW,18
#define X3_PUL            PW,20
#define X4_PUL            PW,22
#define X5_PUL            PW,24
#define X6_PUL            PW,26
#define ENC1_PUL         PW,28
#define ENC2_PUL         PW,30

void SysActInputDeal(void);
void SysActParamDeal(void);
void SysActInfoDeal(void);
void SysActOutputDeal(void);

//-----------------------------------------------
//Name:void SysActInit(void)
//Input:None
//Output:None
//Global:None
//Creator:HandOn 20190610
//Revisor:None
//DEC:应用模块初始化
//-----------------------------------------------
void SysActInit(void)
```

```
{
    KeepComponentsInit();              //相关调用不可去除
    SetFlashReadprotection();           //代码保护

    //Modbus-Rtu 初化
    eMBInit(MB_RTU,GetParamWord(SYSSLV_ADDR),6,
           GetParamWord(SYSBTR_ADDR),MB_PAR_NONE);
    eMBEnable();

    //Modbus-Tcp 初始化
    SetEthernetIpByNum(GetParamDWord(SYSIP12_ADDR));
    eMBTCPInit(GetParamWord(SYSPORT_ADDR));
    eMBTcpEnable();

    //指示灯闪烁 慢闪
    SetLedMod(LED_RUN,LED_MODE_SLOW);
    SetLedMod(LED_COM,LED_MODE_SLOW);
    SetLedMod(LED_ERR,LED_MODE_SLOW);
}

//-----------------------------------------------
//Name:void SysActIdle(void)
//Input:None
//Output:None
//Global:None
//Creator:HandOn 20190610
//Revisor:None
//DEC:应用模块循环处理
//-----------------------------------------------
void SysActIdle(void)
{
    //输入处理
    SysActInputDeal();

    //4 毫秒周期定时处理
    if(GetIdleTB(TB_4MS))
    {
        SysActInfoDeal();
        SysActParamDeal();
    }

    //输出处理
    SysActOutputDeal();
}

//-----------------------------------------------
//Name:void SysActTimer(void)
//Input:None
```

```
//Output:None
//Global:None
//Creator:HandOn 20190610
//Revisor:None
//DEC:应用模块 1 毫秒定时器
//-----------------------------------------------
void SysActTimer(void)
{

}

//-----------------------------------------------
//Name:void SysActInputDeal(void)
//Input:None
//Output:None
//Global:None
//Creator:HandOn 20190610
//Revisor:None
//DEC:应用模块输入处理
//-----------------------------------------------
void SysActInputDeal(void)
{
    int i;

    //输入 将开关量输入信息存储到 PB 位参数组
    for(i=0;i<IN_RVD1;i++)
        SetParamBit(IN_X1_ADDR+i,GetIoStu(i));

    //Ai1 将模拟输入存储到 PW 寄存器参数组
    SetParamWord(AI1_VIN_ADDR,GetInVol(GetAiVal(AD_CHANNEL1)));

    //Ai2
    SetParamWord(AI2_IIN_ADDR,GetInCir(GetAiVal(AD_CHANNEL2)));

    //脉冲计数
    for(i=0;i<IN_MAX_PUL;i++)
        SetParamDWord(X1_PUL+2*i,GetXinPul(i));

    //AB 相脉冲计数
    for(i=0;i<IN_ENC_MAX;i++)
        SetParamDWord(ENC1_PUL+2*i,GetEncPul(i));
}

//-----------------------------------------------
//Name:void SysActParamDeal(void)
//Input:None
//Output:None
//Global:None
```

```
//Creator:HandOn 20190610
//Revisor:None
//DEC:应用模块处理
//-----------------------------------------------
void SysActParamDeal(void)
{

}

//-----------------------------------------------
//Name:void SysActInfoDeal(void)
//Input:None
//Output:None
//Global:None
//Creator:HandOn 20190610
//Revisor:None
//DEC:应用模块处理
//-----------------------------------------------
void SysActInfoDeal(void)
{

}

//-----------------------------------------------
//Name:void SysActOutputDeal(void)
//Input:None
//Output:None
//Global:None
//Creator:HandOn 20190610
//Revisor:None
//DEC:应用模块输出处理
//-----------------------------------------------
void SysActOutputDeal(void)
{
    int i;

    //输出 将 PB 位参数信息控制输出开关量
    for(i=0;i<OUT_RVD1;i++)
        SetIoStu(i,GetParamBit(OUT_Y1_ADDR+i) ? 1 : 0);

    //Ao1 将 PW 寄存器参数信息输出到输出模拟量
    SetAoVal(DA_CHANNEL1,GetOutVol(GetParamWord(AO1_IOUT_ADDR)));
}
```