

## ST MC SDK5.x 电机库软件框架说明

转载：AI 电堂作者 an（ST 原厂安超）文章

### 前言

在使用 ST MC SDK5.x 库过程中，用户对软件整体全面了解后，开发设计才会得心应手、事半功倍。本文将从系统到软件架构，并对重要的三个任务环路进行细致说明，希望对需要了解 ST 电机库的用户有所帮助。本文默认使用 STM32F30x 产生的工程做文件说明。

### 内容概括：

- 1、总体软件架构
- 2、软件主要环路
- 3、整体软件框图
- 4、三大任务流程
- 5、附录
- 6、总结

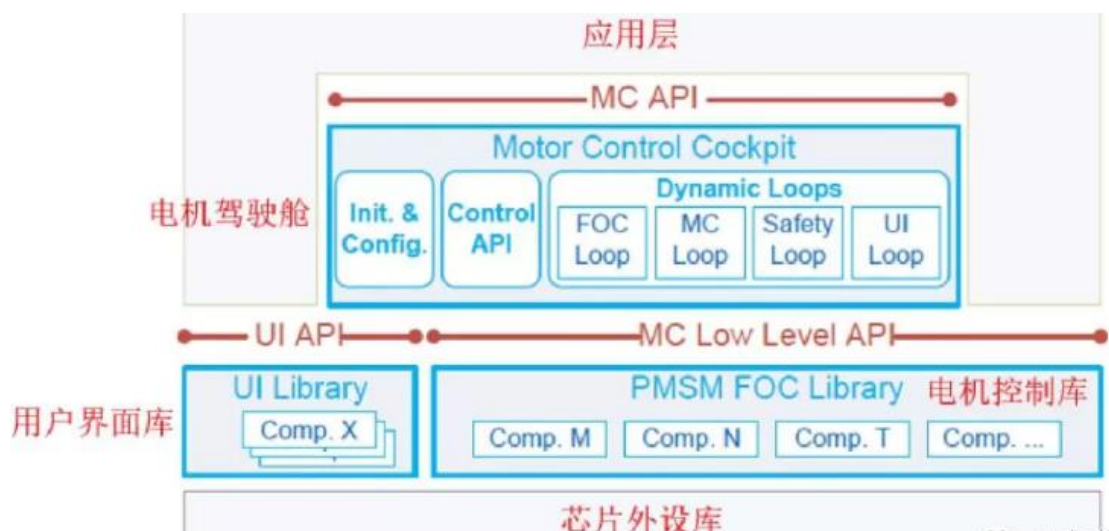
#### 1、总体软件架构

MC SDK5.x 包含有芯片外设库，电机库和电机应用层三个主要部分，其中：

芯片外设库使用 ST HAL/LL 库，可被各个层级调用；

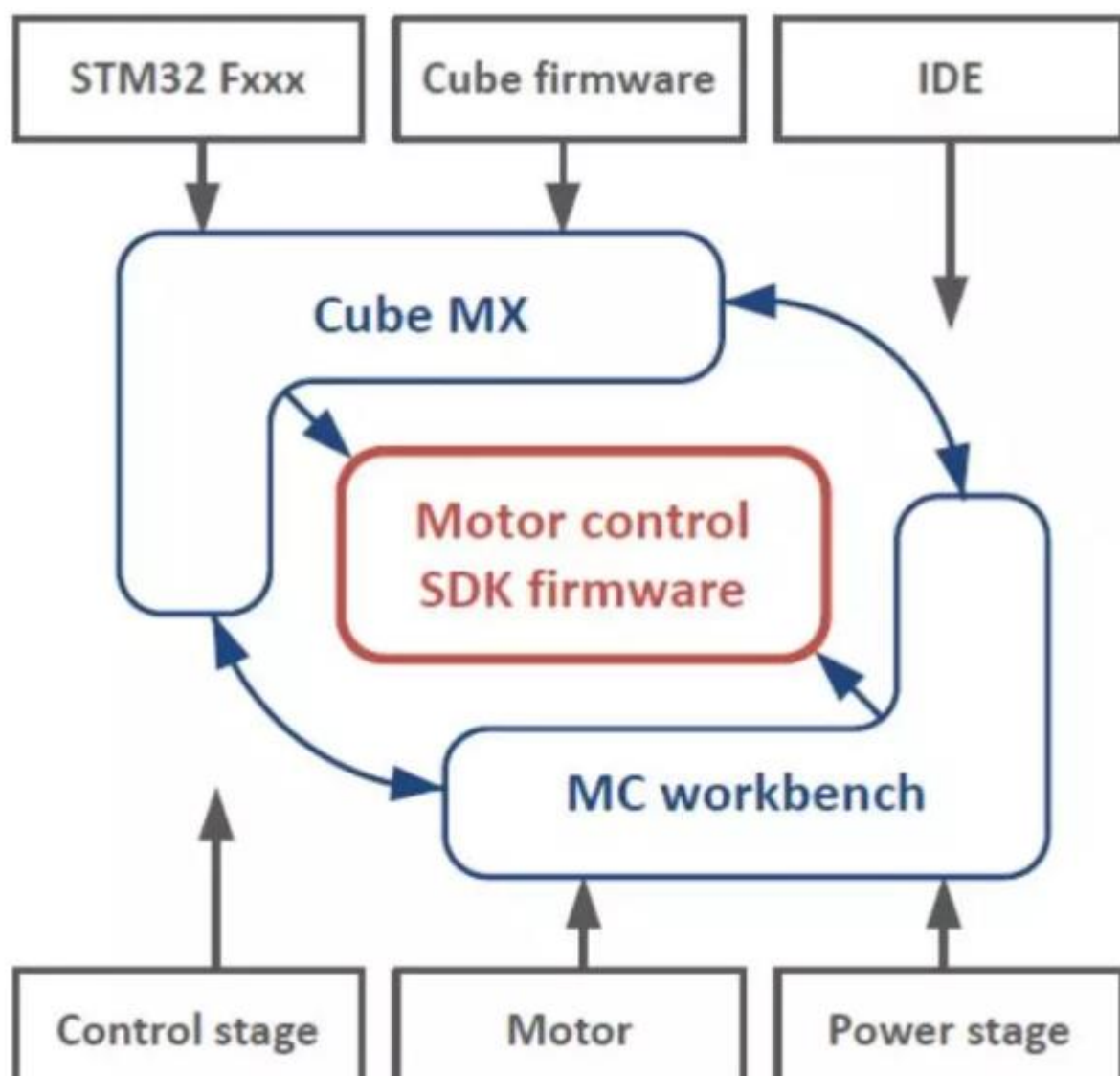
电机库则是主要的电机 FOC 控制层；最上层为电机应用层，供客户直接使用电机库，而不去关心底层如何实现的，加快用户程序开发；另外 MC SDK5.x 还提供 UI 库，用于界面调试通讯使用，比如和 Workbench 之间的交互就是通过 UI 库实现；

这里强调的一点是电机库是综合体，包含 FOC 算法，单片机外配置，中断机制等各个环节，简单控制可能只需要关心电机应用层即可，如果复杂控制将涉及到整体操作。



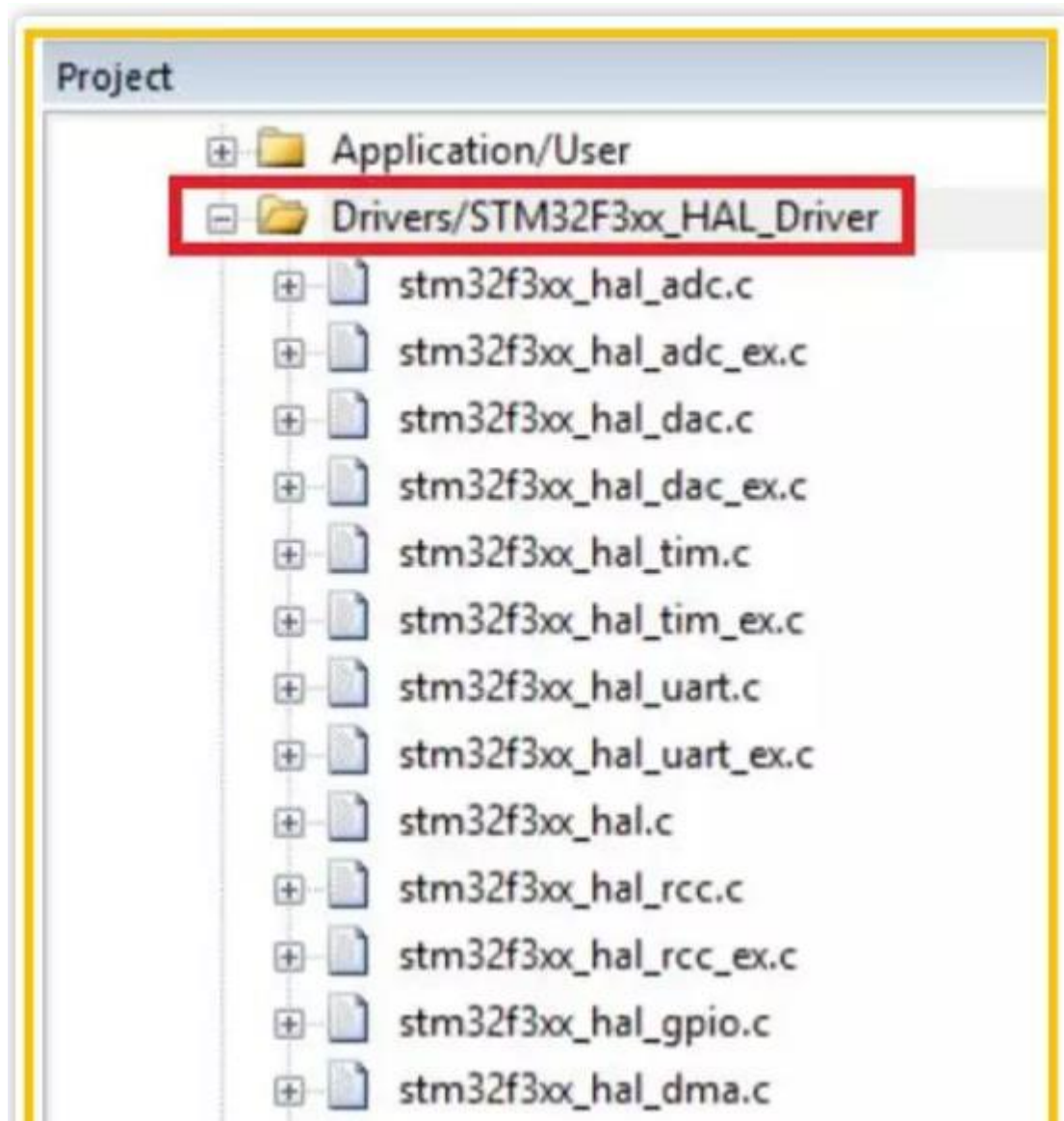
在 SDK 使用过程中，电机本体、电机控制硬件板、控制管脚、控制策略在 MC workbench 中配置完成，顺序为 MC Workbench -> CubeMx 工程 -> 电机库代码（芯片外设库 + 电机控制库 + 电机驾驶舱 + 用户界面库 + 系统初始化），该生成代码加入简单 API 后（比如 MC\_StartMotor1）可以直接运行对应电机，

当需要细化控制或者复杂控制时才有可能涉及到修改电机驾驶舱或者电机控制库中的代码。



### 1.1 芯片外设库

针对芯片的每种外设都有对应的库函数提供，需要结合 Cube 使用说明调用这些库函数；电机库必须掌握的外设有 TIMER，ADC，GPIO。



## 1.2 电机控制库文件及其说明

一般来说，普通应用这个部分不会涉及到，如果是在 API 层已经无法满足电机控制的需求时才会考虑修改这个部分，需要对电机运行框架非常熟悉的前提下去进行修改。

源文件	描述
bus_voltage_sensor.c	母线电压
circle_limitation.c	电压极限限制
enc_align_ctrl.c	编码器初始定位控制
encoder_speed_pos_fdbk.c	编码器传感器相关
fast_div.c	快速软件除法
hall_speed_pos_fdbk.c	Hall 传感器相关
inrush_current_limiter.c	浪涌电流限制
mc_math.c	数学计算
mc_interface.c	马达控制底层接口
motor_power_measurement.c	平均功率计算
ntc_temperature_sensor.c	NTC 温度传感
open_loop.c	开环控制
pid_regulator.c	PID 环路控制
pqd_motor_power_measurement.c	功率计算
pwm_common.c	TIMER 同步使能
pwm_curr_fdbk.c	SVPWM, ADC 设定相关接口
r_divider_bus_voltage_sensor.c	实际母线电压采集
virtual_bus_voltage_sensor.c	虚拟母线电压

ramp_ext_mgr.c	无传感开环转闭环控制
speed_pos_fdbk.c	速度传感接口
speed_torq_ctrl.c	速度力矩控制
state_machine.c	电机状态相关
virtual_speed_sensor.c	无传感开环运行相关
以下部分在工程中只会存在其中一个文件	
ics_f30x_pwm_curr_fdbk.c	STM32F3 的 ICS 采样
r1_f30x_pwm_curr_fdbk.c	STM32F3 的单电阻采样
r3_1_f30x_pwm_curr_fdbk.c	STM32F3 的三电阻采样 (1 各个 ADC)
r3_2_f30x_pwm_curr_fdbk.c	STM32F3 的三电阻采样 (2 各个 ADC)



### 1.3 电机驾驶舱

这个部分就是为客户直接使用电机库而准备的，各种 API 函数供用户调用；可以说简单应用或直接使用这些 API 函数足够达成电机控制应用，客户不需要关心底层如何操作，只需要关注自身需要实现哪些必要的功能，使得项目开发更加快速有效。

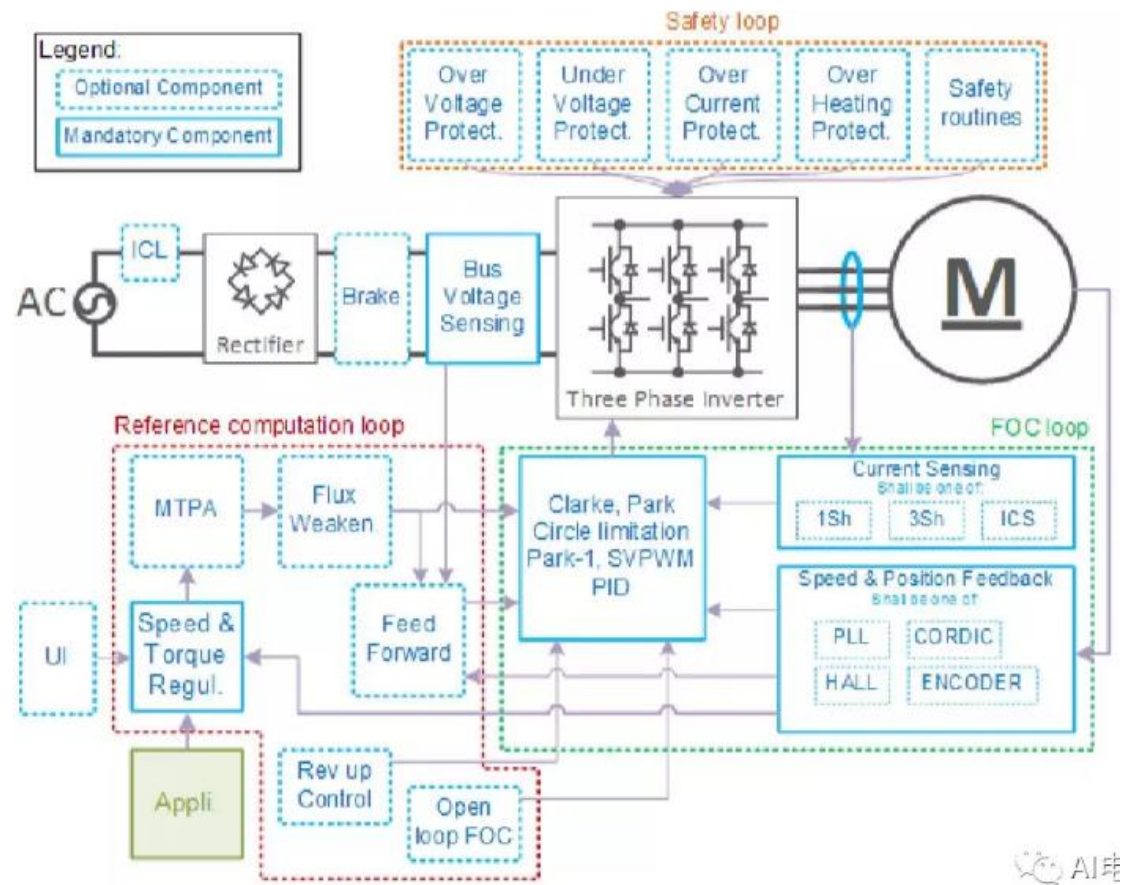
这里以一个马达控制函数为例：如果双马达控制的第二个马达则将后缀变为 Motor2，比如启动第二个电机的函数名为 MC\_StartMotor2()，此处所涉及的除非特别说明，速度即为机械速度。

函数名称	函数参量	函数返回	函数功能
MC_StartMotor1	void	bool	启动电机
MC_StopMotor1	void	bool	停止电机
MC_ProgramSpeedRampMotor1	speed,time	void	设定目标速度以及所需时间
MC_ProgramTorqueRampMotor1	torque,time	void	设定目标力矩以及所需时间
MC_SetCurrentReferenceMotor1	Iqref, Idref	void	设定 Iq, Id 参考
MC_GetCommandStateMotor1	void	MCI_CommandState_t	返回指令执行状态
MC_StopSpeedRampMotor1	void	bool	停止速度指令执行,速度指令保存为执行停止前速度指令
MC_HasRampCompletedMotor1	void	bool	指令是否执行完成
MC_GetMecSpeedReferenceMotor1	void	int16	返回机械参考速度
MC_GetMecSpeedAverageMotor1	void	int16	返回平均机械速度
MC_GetLastRampFinalSpeedMotor1	void	int16	返回上次指令速度
MC_GetControlModeMotor1	void	STC_Modality_t	返回控制模式
MC_GetImposedDirectionMotor1	void	int16	返回电机转动方向
MC_GetSpeedSensorReliabilityMotor1	void	bool	返回当前速度传感器可信度
MC_GetPhaseCurrentAmplitudeMotor1	void	Is	返回电流值
MC_GetPhaseVoltageAmplitudeMotor1	void	Vs	返回电压值
MC_GetIabMotor1	void	Ia, Ib	返回 a, b 相电流
MC_GetIalphabetaMotor1	void	I $\alpha$ , I $\beta$	返回 clark 变换后的 I $\alpha$ , I $\beta$
MC_GetIqdMotor1	void	Id, Iq	返回 park 变换后的 Id, Iq
MC_GetIqdrefMotor1	void	Idref, Iqref	返回 Id, Iq 参考
MC_GetVqdMotor1	void	Vd, Vq	返回变换电压量 Vd, Vq
MC_GetValphabetamotor1	void	V $\alpha$ , V $\beta$	返回变换电压量 V $\alpha$ , V $\beta$
MC_GetElAngledppMotor1	void	Angledpp	返回电角度 DPP 数据

MC_GetTerefMotor1	void	Iqref	返回电流参考
MC_SetidrefMotor1	Idref	void	设定电流 Id 参考
MC_Clear_IqdrefMotor1	void	void	Iq, Id 数据回到默认值
MC_AcknowledgeFaultMotor1	void	Bool	清除异常状态
MC_GetOccurredFaultsMotor1	void	Fault	得到发生了的 Fault 状态
MC_GetCurrentFaultsMotor1	void	Fault	得到当前的 Fault 状态
MC_GetSTMStateMotor1	void	State	得到电机状态

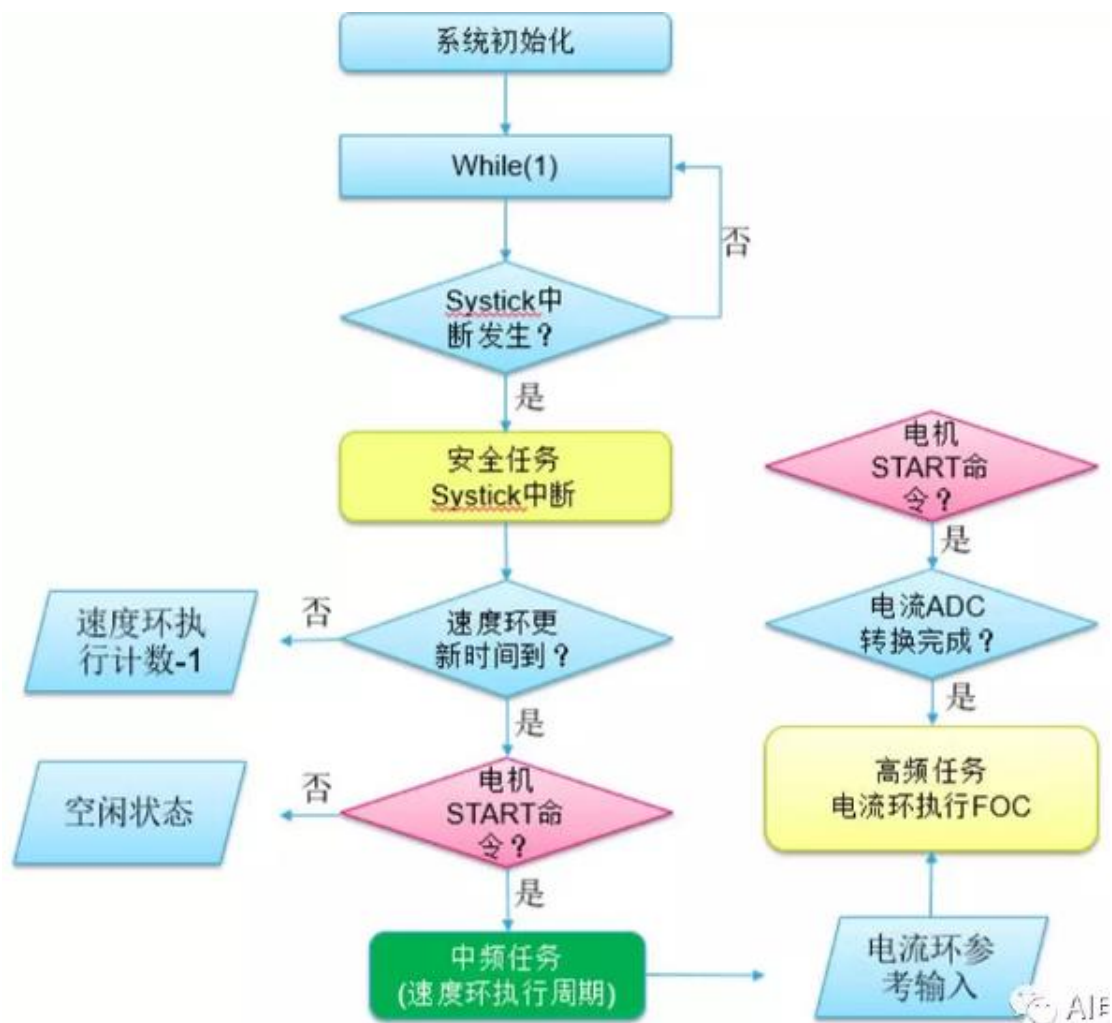
## 2、软件主要环路

包含有三个主要环路：FOC 环路，安全环路，电机控制环路；虚线方框部分为可选择组件，实现部分为主要组件，可以看到整个控制围绕了电机控制系统的方方面面。



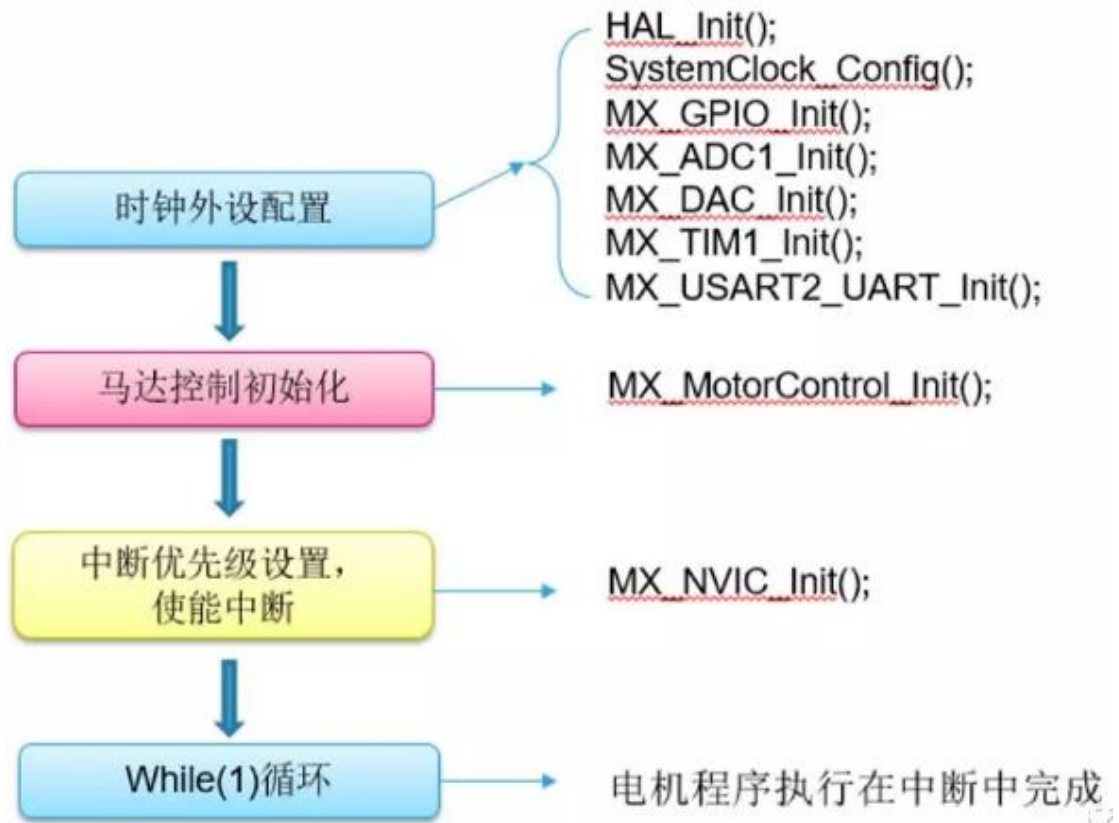
## 3、整体软件框图

可以看到上面的电机控制框图三个环路构成了底层驱动部分，对于整体程序流程上，电机库控制过程都发生在中断中，区别于普通顺序控制流程，也无任务调度；这样做可以将电机控制做到实时控制，因此整个 STM32 产品都可以用于 FOC 控制。



### 3.1 Main 函数

在 main.c 中主要是系统的初始化函数，main 函数更是条理清晰。

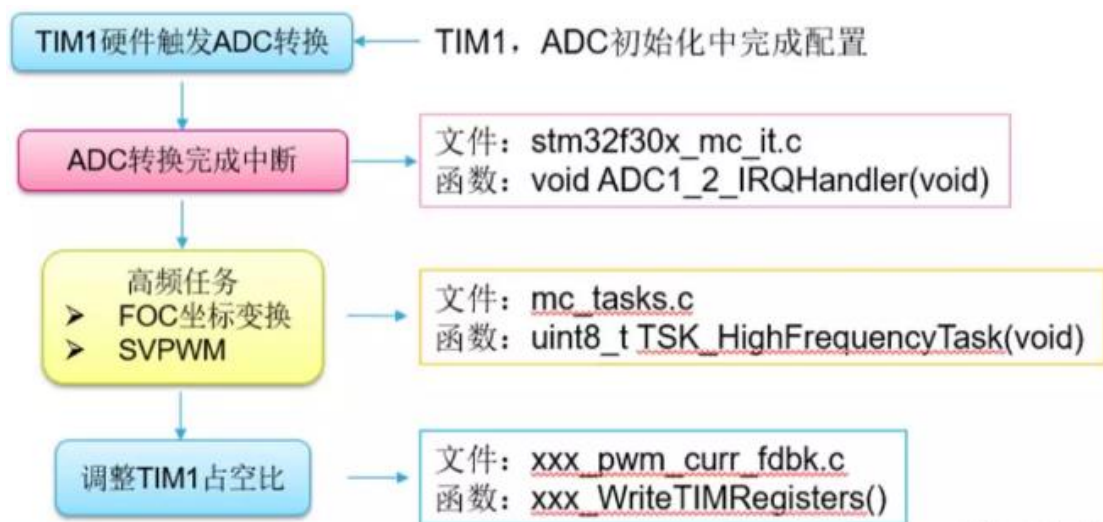


### 3.2 两个重要的中断

#### 3.2.1 ADC 转换完成中断

#### 3.2.1 ADC 转换完成中断

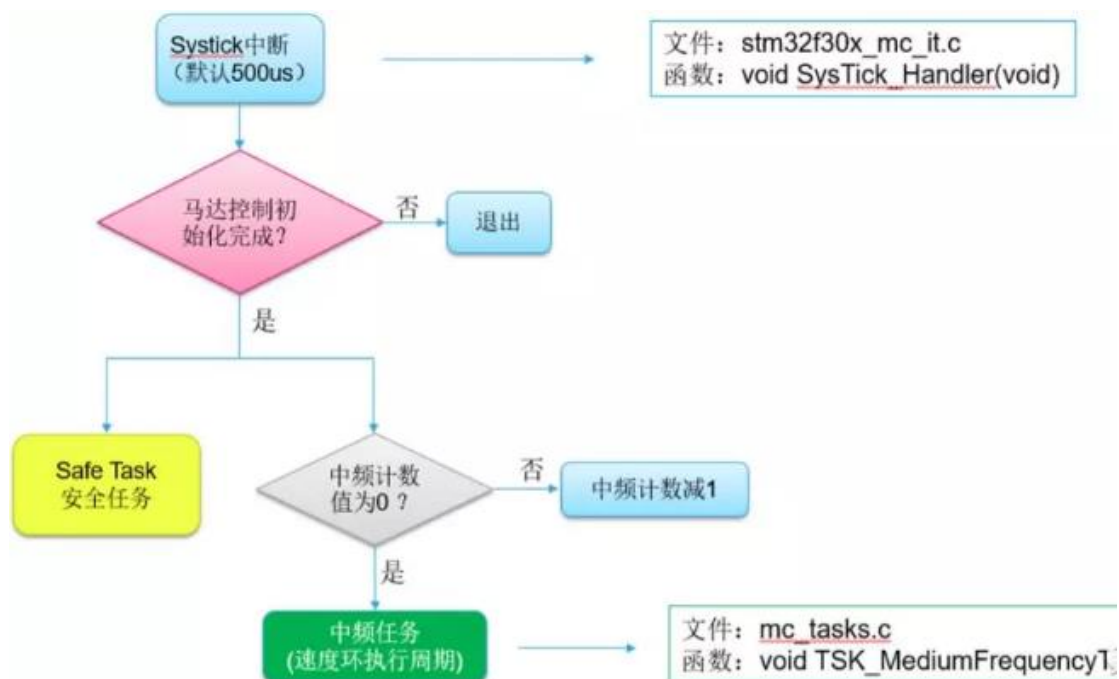
高频任务执行于 ADC 采样转换完成中断，ADC 采样开始由 TIM1 硬件触发，转换完成后进入中断，在这个中断中执行 FOC 坐标变换以及 SVPWM 的执行，最终控制 TIM1 的 PWM 占空比输出。



#### 3.2.2 SysTick 中断

这个中断默认为 500us 的定时中断，安全任务执行在这个中断中，并且以 500us 为基础，中频任务也执行在这个中断中，比如我们设定的速度环执行为 2ms，实际是 500us\*4，也就是四次 systick 中断后执行一次中频任务。

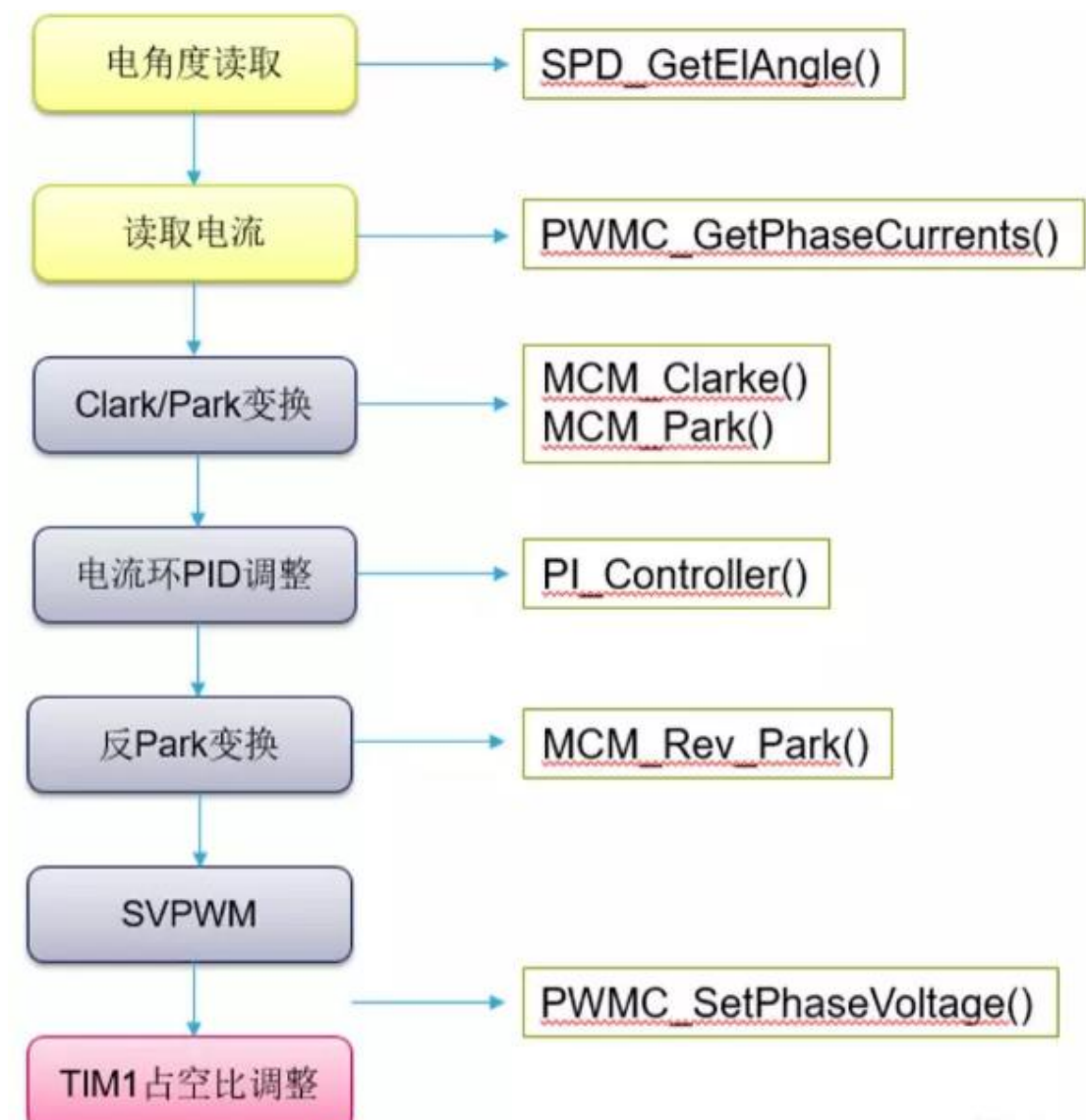




#### 4、三大任务流程

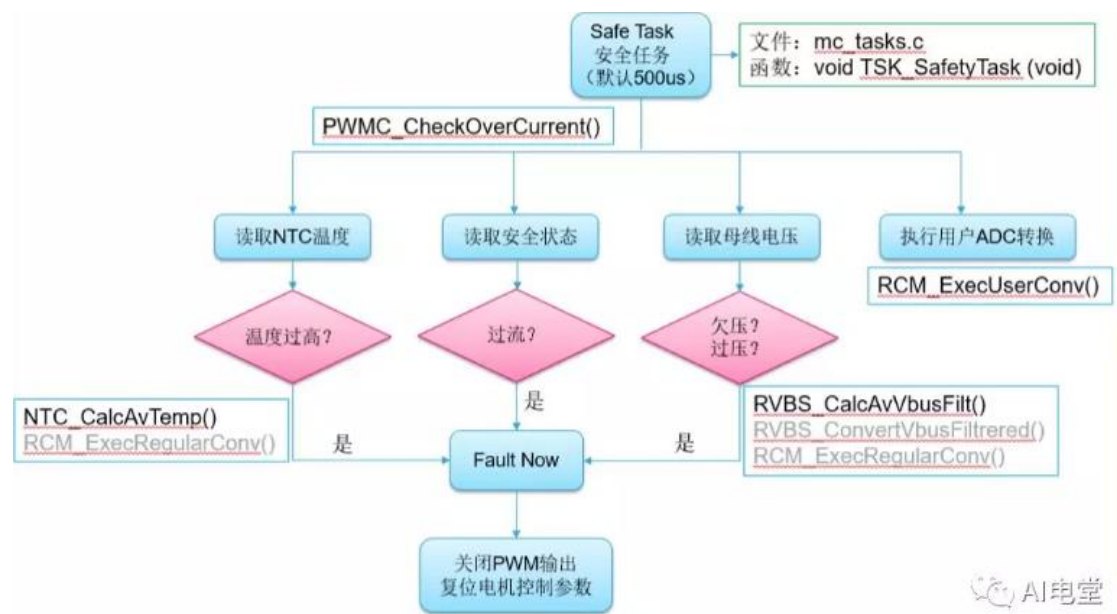
##### 4.1 高频任务

高频任务函数位于 `mc_task.c` 的 `TSK_HighFrequencyTask()` 函数中，执行的是核心 FOC 算法。



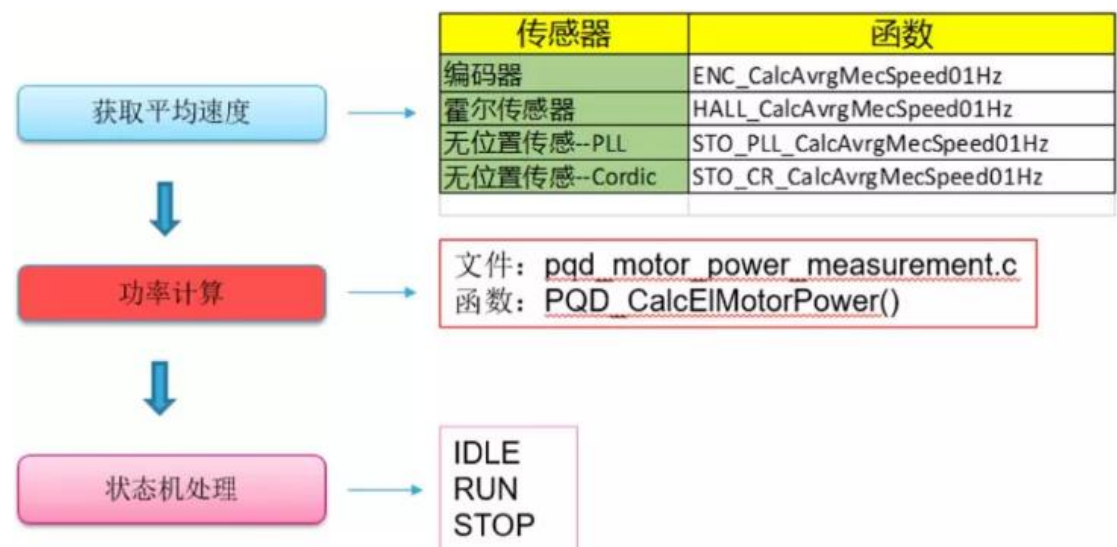
#### 4.2 安全任务

所说的安全任务，字面意义已经很明确。这个部分对于过温、过流、欠压、过压保护进行判断，如果触发了上述保护，则会关闭 PWM 输出，复位电机控制参数，同时这个部分也是温度 ADC 采样，母线电压采样，用户 ADC 采样的执行地方。



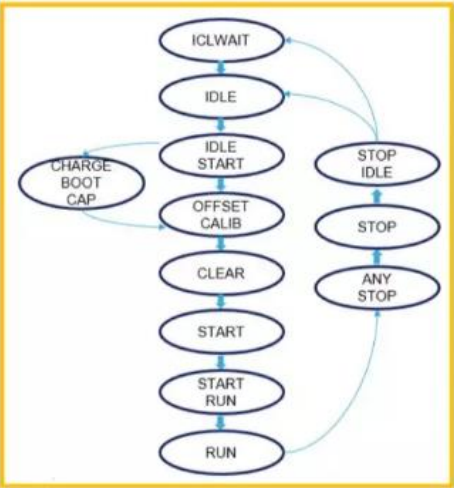
#### 4.3 中频任务

中频任务执行于 `systick` 中断中，实际为速度环以及状态机执行地方，同时包含有功率计算，如需要增加个人状态机，这个地方为添加点，同时注意根据传感器不同，平均速度获取函数也不同。

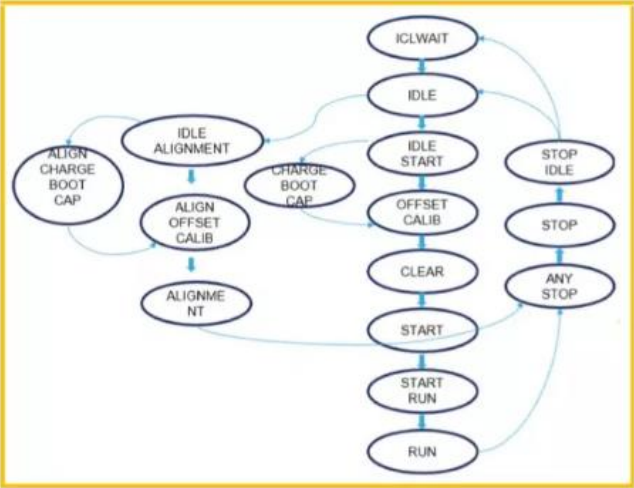


状态机

状态返回值	枚举值	状态描述
IDLE	0	空闲状态
IDLE_ALIGNMENT	1	Encoder定位后的过度状态
ALIGNMENT	2	Encoder定位状态
IDLE_START	3	执行启动电机后的过渡状态
START	4	无传感开环状态
START_RUN	5	开环转闭环过渡
RUN	6	闭环运行状态
ANY_STOP	7	电机停转前状态
STOP	8	电机停转状态
STOP_IDLE	9	停止到空闲状态过渡
FAULT_NOW	10	发生fault状态
FAULT_OVER	11	Fault可转入STOP_IDLE前状态
ICLWAIT	12	浪涌电流限制等待
ALIGN_CHARGE_BOOT_CAP	13	Encoder自举电容充电状态
ALIGN_OFFSET_CALIB	14	Encoder电流偏移量读取状态
ALIGN_CLEAR	15	Encoder进入startup过渡状态
CHARGE_BOOT_CAP	16	自举电容充电状态
OFFSET_CALIB	17	电流偏移量读取状态
CLEAR	18	进入startup过渡状态



无传感模式状态机



编码器模式状态机

5、附录

下面是就用户比较关心的电流，电压问题做必要说明。

5.1 三相电流获取

电流的采样在电机培训中提及，为硬件触发机制，TIM1 或者辅助 TIMER 触发 ADC 转换，转换结束后进入到 ADC 中断中，关于采样点选择，触发点请参看电机培训文档，这边



## 技术交流 QQ 群：124545085

不做过多描述；如果用户需要使用得到三相电流值，则可以通过几种方式获取，这边使用了基尔霍夫定理  $I_a + I_b + I_c = 0$ ，因此取得  $I_a$ ， $I_b$  电流即可。

### 5.1.1 API 函数读取电流值

使用 API 函数 `MC_GetIabMotor1()`，返回值为 `Curr_Components` 这个结构体。

```
Curr_Components MC_GetIabMotor1(void)
{
    return MCI_GetIab( pMCI[M1] );
}
```

```
typedef struct
{
    int16_t qI_Component1;
    int16_t qI_Component2;
} Curr_Components;
```

如果定义一个变量，则  $I_a$ ， $I_b$  分别是结构体变量的两个 `int16` 数据，如下所示：

```
Curr_Components Iab_Value;
Iab_Value = MC_GetIabMotor1() ;
Ia = Iab_Value. qI_Component1 ;
Ib = Iab_Value. qI_Component2 ;
```

### 5.1.2 利用已有的全局变量获取

我们看到在 `mc_task.c` 中有全局变量 `FOCVars` 这个变量，而这个结构体变量的其中一员即我们需要得到的  $I_a$ ， $I_b$  数据。

```

mc_tasks.c
55  #define STOPPERMANENCY_TICKS2  (uint16_t)((SY
56
57  /* Un-Comment this macro define in order to a
58  braking action on over voltage */
59  /* #define MC.SMOOTH_BRAKING_ACTION_ON_OVERV
60
61  /* USER CODE END Private define */
62  #define VBUS_TEMP_ERR_MASK ~(0 | 0 | MC_OVER_
63
64  /* Private variables-----
65  FOCVars_t FOCVars[NBR_OF_MOTORS];
66  MCI_Handle_t Mci[NBR_OF_MOTORS];
67  MCI_Handle_t * oMCInterface[NBR_OF_MOTORS];
68  MCT_Handle_t MCT[NBR_OF_MOTORS];
69  STM_Handle_t STM[NBR_OF_MOTORS];
70  SpeednTorqCtrl_Handle_t *pSTC[NBR_OF_MOTORS];

```

```

mc_tasks.c  mc_type.h
176  * @brief FOC variables structure
177  */
178  typedef struct
179  {
180      Curr_Components Iab;          /**< @brief Stator current on stator reference frame
181      Curr_Components Ialphabeta;  /**< @brief Stator current on stator reference frame
182      Curr_Components IqdHF;       /**< @brief Stator current on stator reference frame
183      Curr_Components Iqd;         /**< @brief Stator current on rotor reference frame
184      Curr_Components Iqdref;      /**< @brief Stator current on rotor reference frame
185      int16_t UserIdref;           /**< @brief User value for the Idref stator current
186      Volt_Components Vqd;        /**< @brief Phase voltage on rotor reference frame q
187      Volt_Components Valphabeta; /**< @brief Phase voltage on stator reference frame
188      int16_t hTeref;             /**< @brief Reference torque */
189      int16_t hElAngle;           /**< @brief Electrical angle used for reference fram
190      uint16_t hCodeError;        /**< @brief error message */
191      CurrRefSource_t bDriveInput; /**< @brief It specifies whether the current referen
192                                   *          #INTERNAL or #EXTERNAL*/
193  } FOCVars_t, *pFOCVars_t;

```

可以有下面的读取方式：

```

Curr_Components Iab_Value;
Iab_Value = FOCVars[0].Iab;
Ia = Iab_Value. qI_Component1 ;
Ib = Iab_Value. qI_Component2 ;

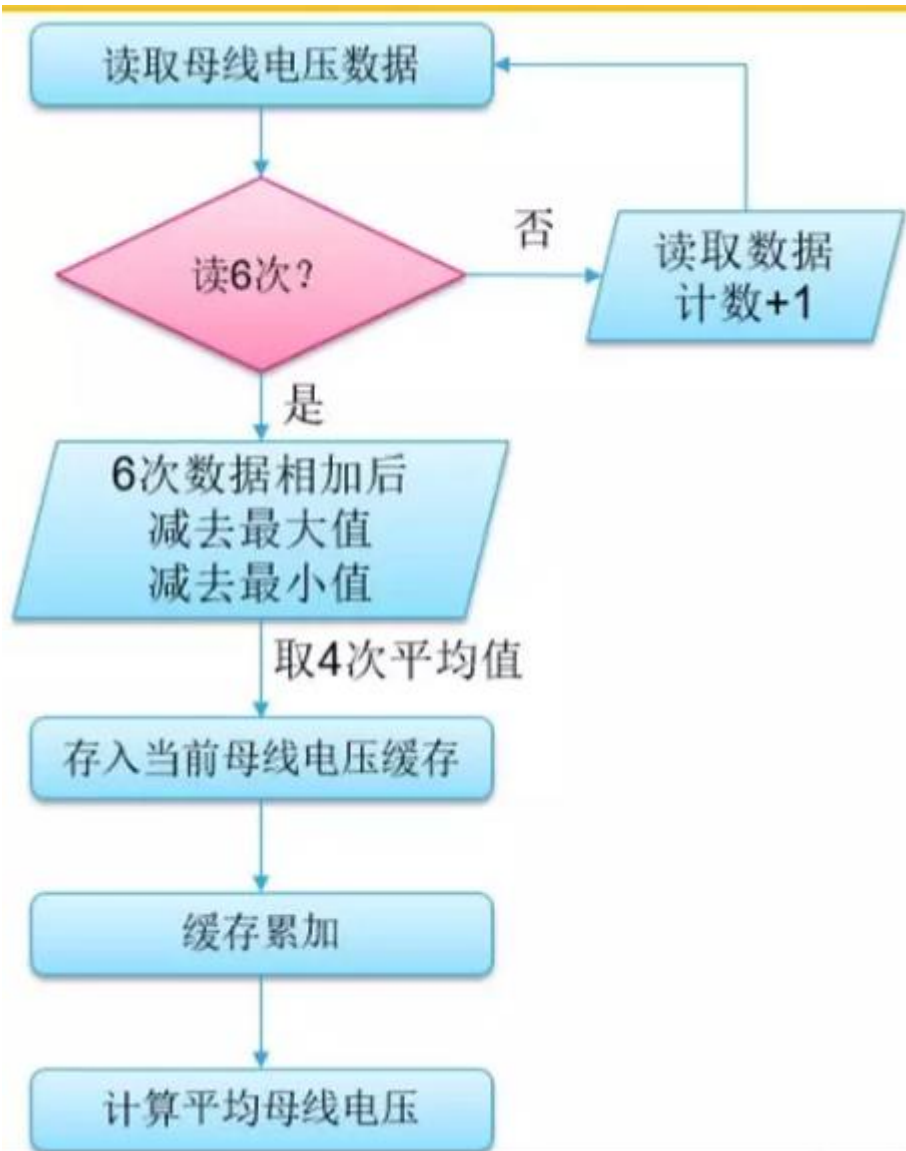
```

值得说明的是有些关注于底层 ADC 采样的用户很可能想到的是直接从 ADC 外设获取，很遗憾，在 ST 的电机库中 Ia, Ib 的数据得到非固定方式存在，即使使用了 Inject 注

入通道进行采样，因为算法的灵活性 JDR1, JDR2 寄存器中存入的数据的意义是根据控制在变动的，比如 JDR1 可能 Ia，也可能是 Ib，因此该方式无法直接获取电流数据。

## 5.2 母线电压的获取

在 API 函数中这个部分没有涉及获取函数，因此有必要在此做些说明。母线电压的读取部分在 TSK\_SafetyTask()函数中执行，这边会得到一个平均值电压数据，这边简单介绍下获取过程，如果对这个部分不关心，可以忽略。



### 5.2.1 函数获取方式

在 bus\_voltage\_sensor.c 中给出了两种电压格式函数，VBS\_GetAvBusVoltage\_d()返回的是母线电压的数字量，VBS\_GetAvBusVoltage\_V()返回的是以伏特（V）为单位的母线电压。

```
bus_voltage_sensor.c
64  * @param pHandle related Handle of BusVoltageSensor_Handle
65  * @retval uint16_t Latest averaged Vbus measurement in digi
66  */
67  uint16_t VBS_GetAvBusVoltage_d( BusVoltageSensor_Handle_t * p
68  {
69      return ( pHandle->AvBusVoltage_d );
70  }
71
72  /**
73   * @brief It return latest averaged Vbus measurement expres
74   * @param pHandle related Handle of BusVoltageSensor_Handle
75   * @retval uint16_t Latest averaged Vbus measurement in Volt
76   */
77  uint16_t VBS_GetAvBusVoltage_V( BusVoltageSensor_Handle_t * p
78  {
79      uint32_t temp;
80
81      temp = ( uint32_t ) ( pHandle->AvBusVoltage_d );
82      temp *= pHandle->ConversionFactor;
83      temp /= 65536u;
84
85      return ( ( uint16_t ) temp );
86  }
```

这边需要使用的是 MCT 这个结构体组件，定义在 mc\_task.c 中。

```
mc_tasks.c
60
61  /* USER CODE END Private define */
62  #define VBUS_TEMP_ERR_MASK ~(0 | 0 | MC_OVE
63
64  /* Private variables-----
65  FOCVars_t FOCVars[NBR_OF_MOTORS];
66  MCI_Handle_t Mci[NBR_OF_MOTORS];
67  MCI_Handle_t * oMCInterface[NBR_OF_MOTORS];
68  MCT_Handle_t MCT[NBR_OF_MOTORS];
69  STM_Handle_t STM[NBR_OF_MOTORS];
70  SpeednTorqCtrl_Handle_t *pSTC[NBR_OF_MOTORS
71  PID_Handle_t *pPIDSpeed[NBR_OF_MOTORS];
72  PID_Handle_t *pPIDIq[NBR_OF_MOTORS];
73  PID_Handle_t *pPIDId[NBR_OF_MOTORS];
74  RDivider_Handle_t *pBusSensorMl;
75  NTC_Handle_t *pTemperatureSensor[NBR_OF_MOT
```

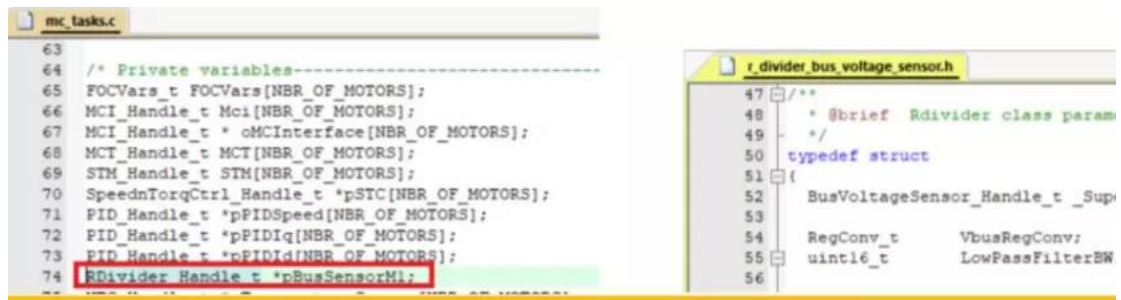
具体使用如下操作：

## 5.2.2 全局变量取得母线电压



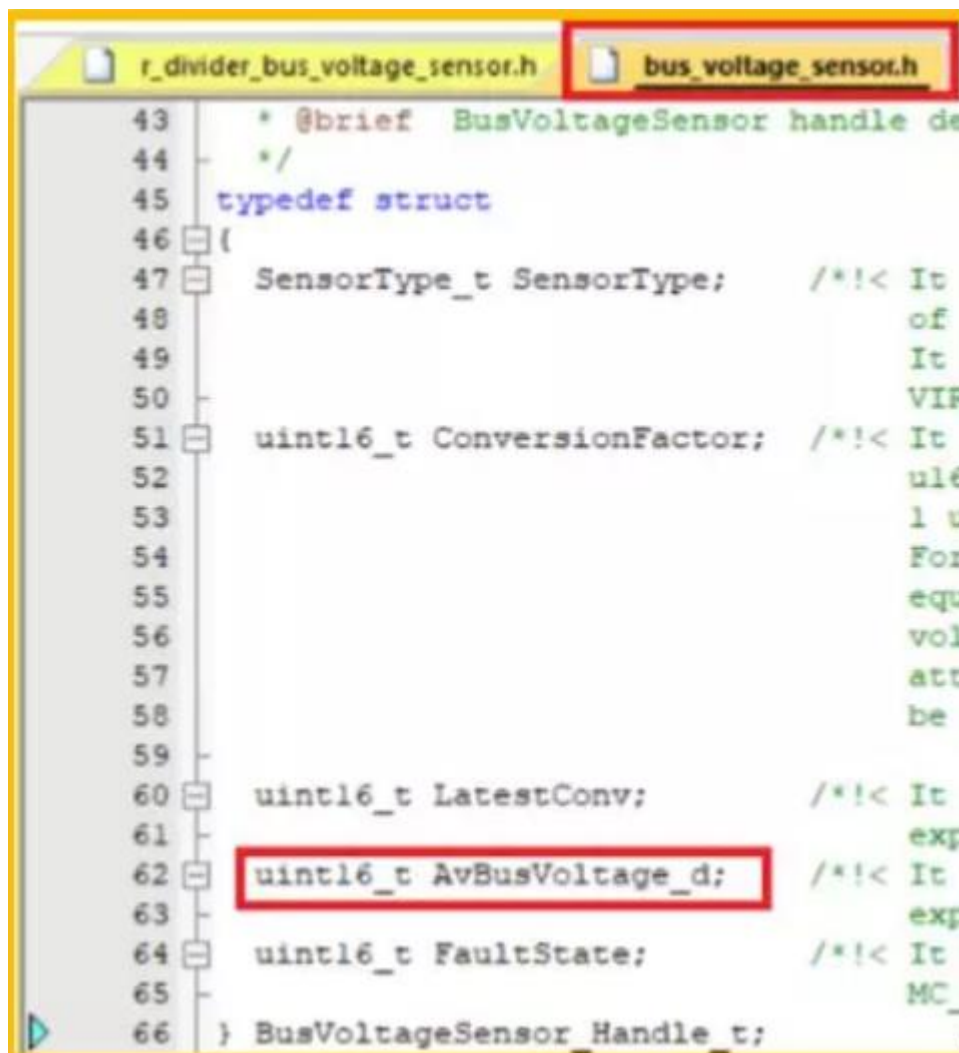
## 技术交流 QQ 群：124545085

注意到在母线电压结构体定义过程，有直接的全局变量 `RDivider_Handle_t`，该结构体成员之一即为 `BusVoltageSensor_Handle_t_Super`；这个我们所需要的结构体。



```
mc_tasks.c
63
64 /* Private variables-----
65 FOCVars_t FOCVars[NBR_OF_MOTORS];
66 MCI_Handle_t Mci[NBR_OF_MOTORS];
67 MCI_Handle_t * cMCInterface[NBR_OF_MOTORS];
68 MCT_Handle_t MCT[NBR_OF_MOTORS];
69 STM_Handle_t STM[NBR_OF_MOTORS];
70 SpeednTorqCtrl_Handle_t *pSTC[NBR_OF_MOTORS];
71 PID_Handle_t *pPIDSpeed[NBR_OF_MOTORS];
72 PID_Handle_t *pPIDiq[NBR_OF_MOTORS];
73 PID_Handle_t *pPIDid[NBR_OF_MOTORS];
74 RDivider_Handle_t *pBusSensorM1;

r_divider_bus_voltage_sensor.h
47 /**
48 * @brief Rdivider class param
49 */
50 typedef struct
51 {
52     BusVoltageSensor_Handle_t_Super
53     RegConv_t          VbusRegConv;
54     uint16_t           LowPassFilterBW
56
```



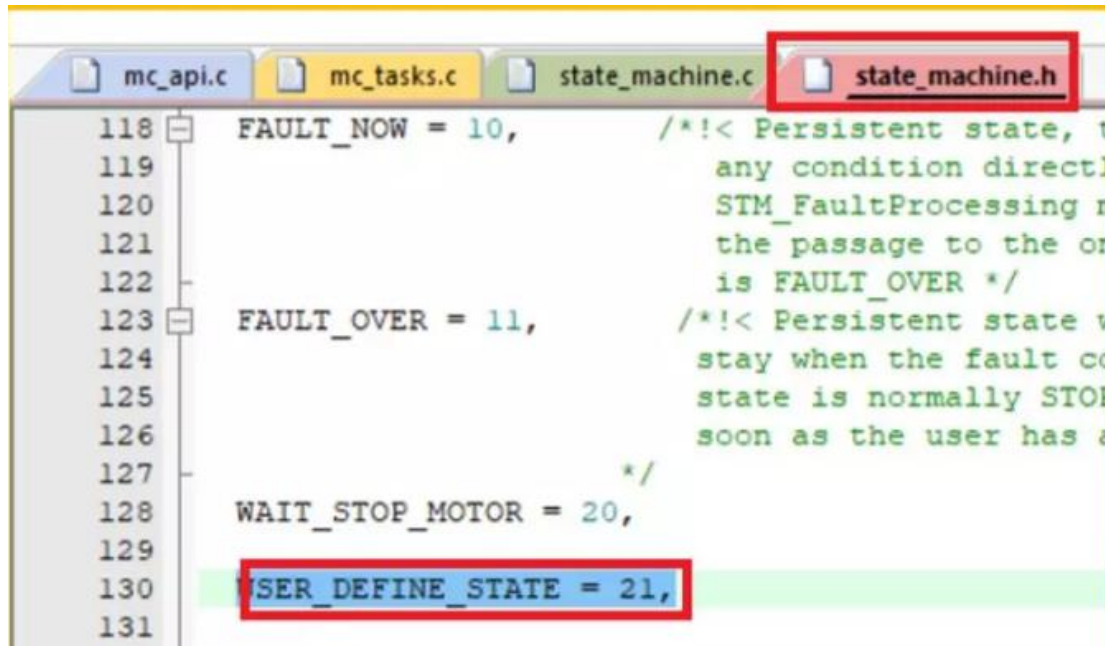
```
r_divider_bus_voltage_sensor.h
bus_voltage_sensor.h
43 * @brief BusVoltageSensor handle de
44 */
45 typedef struct
46 {
47     SensorType_t SensorType; /*!< It
48                                of
49                                It
50                                VIR
51     uint16_t ConversionFactor; /*!< It
52                                ul6
53                                l u
54                                For
55                                equ
56                                vol
57                                att
58                                be
59
60     uint16_t LatestConv; /*!< It
61                                exp
62     uint16_t AvBusVoltage_d; /*!< It
63                                exp
64     uint16_t FaultState; /*!< It
65                                MC_
66 } BusVoltageSensor_Handle_t;
```

因此使用全局变量得到母线电压的操作如下，这边得到的是数字量的母线电压，如果需要转换为伏特则根据参数进行计算得到即可：

```
uint16_t Bus_Voltage_D;
Bus_Voltage_D = pBusSensorM1->_Super.AvBusVoltage_d;
```

### 5.3 状态机操作

一般操作中不需要另外增加状态，如果需要在中频任务中增加自己的状态，假定添加的状态需要位于 START RUN 和 RUN 中间则需要有以下操作：

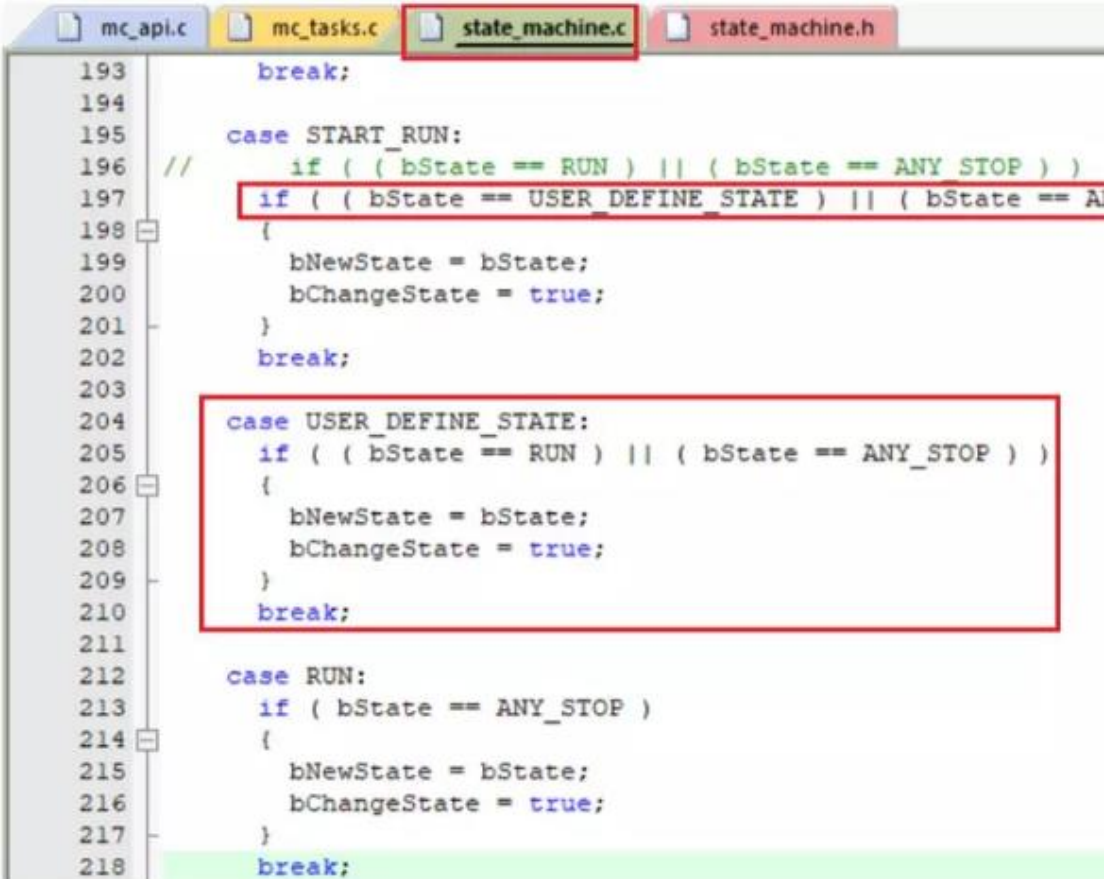


```
118 FAULT_NOW = 10,          /*!< Persistent state, t
119                          any condition directl
120                          STM_FaultProcessing n
121                          the passage to the or
122                          is FAULT_OVER */
123 FAULT_OVER = 11,         /*!< Persistent state w
124                          stay when the fault co
125                          state is normally STOP
126                          soon as the user has a
127                          */
128 WAIT_STOP_MOTOR = 20,
129
130 USER_DEFINE_STATE = 21,
131
```

在 state\_machine.h 的 State\_t 结构体变量中增加状态，比如 USER\_DEFINE\_STATE = 21。修改 mc\_task.c 中频任务函数 TSK\_MediumFrequencyTaskM1()中的状态跳转，START RUN -> USER\_DEFINE\_STATE -> RUN 同时增加一个新的状态 USER\_DEFINE\_STATE 分支语句。

```
468 break;
469
470 case START_RUN:
471 /* only for sensor-less control */
472 STC_SetSpeedSensor(pSTC[M1], &STO_PLL_M1._Super);
473 {
474 /* USER CODE BEGIN MediumFrequencyTask M1 1 */
475
476 /* USER CODE END MediumFrequencyTask M1 1 */
477 FOC_InitAdditionalMethods(M1);
478 FOC_CalcCurrRef( M1 );
479 //STM NextState( &STM[M1], RUN );
480 STM NextState( &STM[M1], USER DEFINE STATE );
481 }
482 STC_ForceSpeedReferenceToCurrentSpeed( pSTC[M1] );
483 MCI_ExecBufferedCommands( oMCInterface[M1] ); /* E
484
485 break;
486
487 case USER_DEFINE_STATE:
488 STM_NextState( &STM[M1], RUN);
489 break;
490
491 case RUN:
492 /* USER CODE BEGIN MediumFrequencyTask M1 2 */
493
```

因为涉及到状态跳转是否正确问题，还需要在 state\_machine.c 中修改状态跳转的判断，做到承上启下的状态跳转，比如下面的判断修改：



```
193     break;
194
195     case START_RUN:
196         // if ( ( bState == RUN ) || ( bState == ANY_STOP ) )
197         if ( ( bState == USER_DEFINE_STATE ) || ( bState == A
198     {
199         bNewState = bState;
200         bChangeState = true;
201     }
202     break;
203
204     case USER_DEFINE_STATE:
205         if ( ( bState == RUN ) || ( bState == ANY_STOP ) )
206     {
207         bNewState = bState;
208         bChangeState = true;
209     }
210     break;
211
212     case RUN:
213         if ( bState == ANY_STOP )
214     {
215         bNewState = bState;
216         bChangeState = true;
217     }
218     break;
```

上述为添加状态操作，删除操作或者是改变状态转移同样需要注意上下状态的切换，可以使用 STM\_NextState()函数以及修改 state\_machine.c 中修改状态跳转的判断灵活操作。

## 6、总结

- ▼简单概括 MC SDK 整体框图，大家可以很清晰的看到三重架构，从下往上分别是外设层，电机库层，电机应用层；一般应用用户只需要熟练掌握电机应用层的 API 即可使用，用户可以从 ST 电机控制培训文档进行详细了解；
- ▼软件框架角度，因为主体控制位于中断服务程序中，需要关注 ADC 采样完成中断以及 SysTick 中断；
- ▼再深入研究，三大任务需要注意，高频任务，安全任务，中频任务；
- ▼高级应用则需要深入到电机底层库，详细掌握各个组件，进行相应的修改或者调用。