

ST MC SDK 5.x 实际使用案例

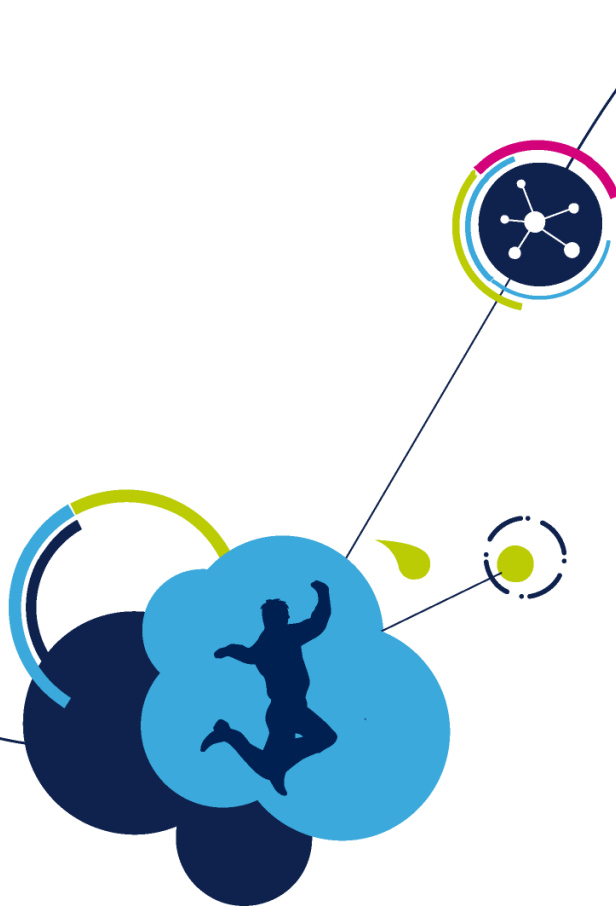
STM32电动机控制应用系列讲座之五



内容

- API函数的使用
- 电机库底层调用与修改
- 电机状态切换以及异常处理
- 增加其他ADC采样以及控制引脚
- 有传感器使用配置

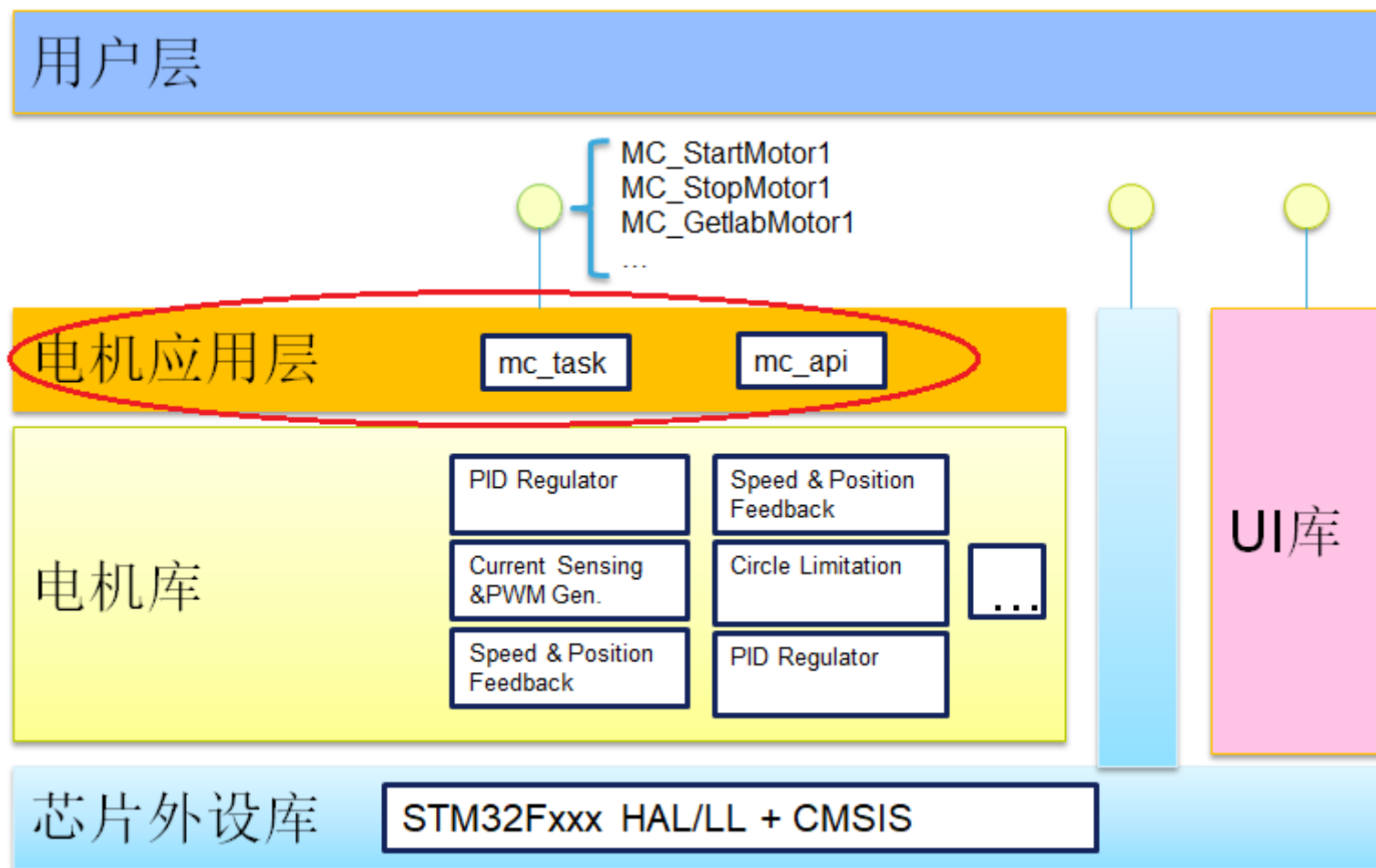
API函数的使用



- ☐ API函数的使用
- ☐ 电机库底层调用与修改
- ☐ 电机状态切换以及异常处理
- ☐ 增加其他ADC采样以及控制引脚
- ☐ 有传感器使用配置

API函数在MC SDK5.x中的位置

- 一般的电机操作调用**API**就足够控制基本的电机运行
- 更像是行为描述操作
- 基于电机库，用于用户调用



API函数列表 (1/2)

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	设定速度以及时间
MC_ProgramTorqueRampMotor1	torque,time	void	设定力矩以及时间
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
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是否可信
MC_GetPhaseCurrentAmplitudeMotor1	void	Is	返回电流值

API函数列表 (2/2)

函数名称	函数参量	函数返回	函数功能
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 α , I β	返回clark变换后的I α , I β
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 α , V β	返回变换电压量V α , V β
MC_GetElAngledppMotor1	void	Angle dpp	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

API函数应用举例（1/4）

■ 案例一：

电机运行在速度模式，设定闭环运行速度为3000RPM，执行时间为1000ms，运行电机

```
MC_ProgramSpeedRampMotor1(3000/6,1000);  
MC_StartMotor1();
```

注意：速度指令参数是以0.1Hz作为单位的，因此
 $3000\text{RPM} = 3000/6 \text{ (0.1HZ)}$

```
main.c  
124  
125  /* USER CODE BEGIN Init */  
126  
127  /* USER CODE END Init */  
128  
129  /* Configure the system clock */  
130  SystemClock_Config();  
131  
132  /* USER CODE BEGIN SysInit */  
133  
134  /* USER CODE END SysInit */  
135  
136  /* Initialize all configured peripherals */  
137  MX_GPIO_Init();  
138  MX_DMA_Init();  
139  MX_ADC1_Init();  
140  MX_DAC_Init();  
141  MX_TIM1_Init();  
142  MX_USART2_UART_Init();  
143  MX_MotorControl_Init();  
144  
145  /* Initialize interrupts */  
146  MX_NVIC_Init();  
147  /* USER CODE BEGIN 2 */  
148  MC_ProgramSpeedRampMotor1(3000/6,1000);  
149  MC_StartMotor1();  
150  /* USER CODE END 2 */  
151  
152  /* Infinite loop */  
153  /* USER CODE BEGIN WHILE */  
154  while (1)
```

API函数应用举例（2/4）

■ 案例二：

电机运行在速度模式，设定闭环运行速度为反向3000RPM，执行时间为1000ms，运行电机

```
MC_ProgramSpeedRampMotor1(-3000/6,1000);  
MC_StartMotor1();
```

```
main.c  
123  
124  /* USER CODE END Init */  
125  
126  /* Configure the system clock */  
127  SystemClock_Config();  
128  
129  /* USER CODE BEGIN SysInit */  
130  
131  /* USER CODE END SysInit */  
132  
133  /* Initialize all configured peripherals */  
134  MX_GPIO_Init();  
135  MX_ADC1_Init();  
136  MX_DAC_Init();  
137  MX_TIM1_Init();  
138  MX_USART2_UART_Init();  
139  MX_MotorControl_Init();  
140  
141  /* Initialize interrupts */  
142  MX_NVIC_Init();  
143  /* USER CODE BEGIN 2 */  
144  
145  MC_ProgramSpeedRampMotor1(-3000/6,1000);  
146  MC_StartMotor1();  
147  
148  /* USER CODE END 2 */
```


API函数应用举例 (3/4)

■ 案例三:

电机运行在力矩模式，设定闭环运行力矩为0.5 Amp
执行时间为1000ms，运行电机

```
MC_ProgramTorqueRampMotor1(DI,1000);  
MC_StartMotor1();
```

$$DI = \frac{I * 65536 * R_{shunt} * A_{op}}{V_{adc}}, \text{ 本例子中 } I = 0.5Amp$$

$$\text{电流数字量} = \frac{\text{电流值} * 65536 * \text{采样电阻阻值} * \text{运放放大倍数}}{\text{ADC参考电压}}$$

```
main.c  
140  MX_NVIC_Init();  
141  /* USER CODE BEGIN 2 */  
142  
143  uint32_t DI;  
144  /*  
145   DI = Iamp*65536*Rshunt*Aop/Vadc  
146   Here we set Iamp = 0.5Amp  
147   For P-Nulcelo-IHM002: Rshunt = 0.330hm, Aop = 1.53, Vadc = 3.3V  
148   DI = 0.5*65536*0.33*1.53/3.3 = 5013  
149  */  
150  DI = 5013;  
151  MC_ProgramTorqueRampMotor1(DI,1000);  
152  MC_StartMotor1();  
153  
154  /* USER CODE END 2 */  
155  
156  /* Infinite loop */  
157  /* USER CODE BEGIN WHILE */  
158  while (1)  
159  {  
160      /* USER CODE END WHILE */
```

API函数应用举例（4/4）

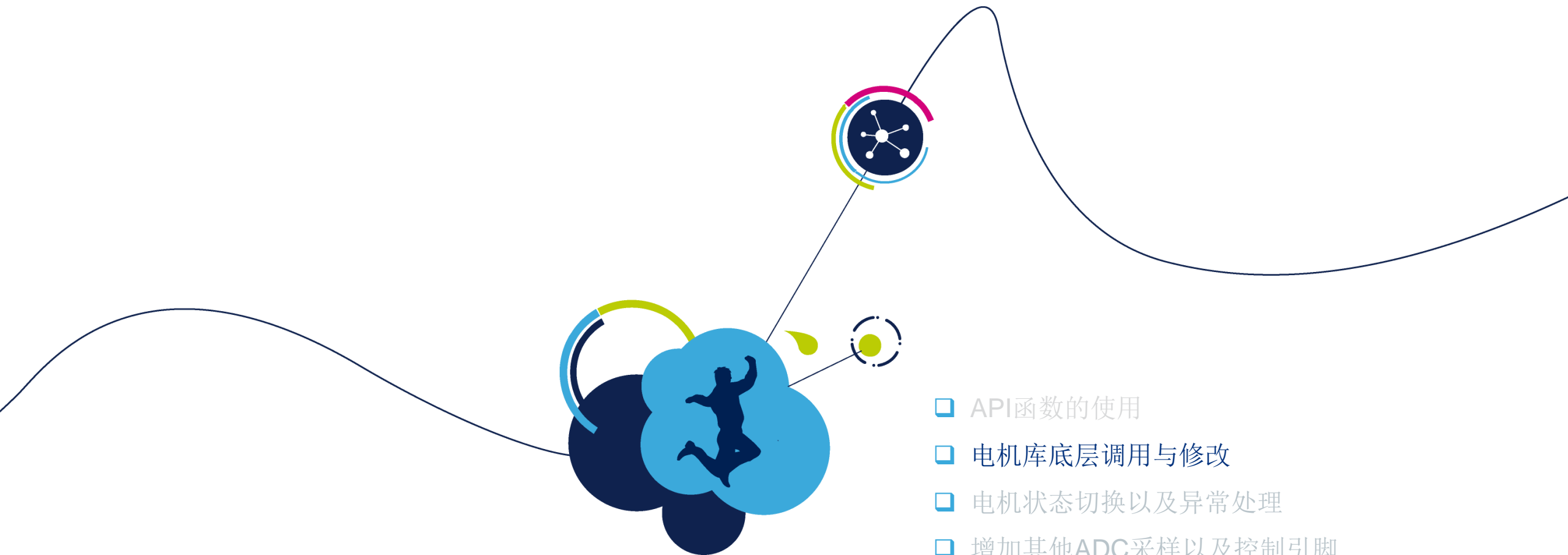
■ 案例四：

得到当前电机运行速度，得到当前Iq，Id参考数据

```
MC_GetMecSpeedAverageMotor1();  
MC_GetIqdrefMotor1();
```

注意：速度返回是以0.1Hz作为单位的，转换为RPM
需要 $0.1\text{Hz} \times 6$

```
141  /* USER CODE BEGIN 2 */  
142  
143  static volatile int16_t Speed_RPM;  
144  static volatile Curr_Components Iqd_Ref_Value;  
145  static volatile Curr_Components Iqd_Real_Value;  
146  
147  MC_ProgramSpeedRampMotor1(3000/6,1000);  
148  MC_StartMotor1();  
149  
150  /* USER CODE END 2 */  
151  
152  /* Infinite loop */  
153  /* USER CODE BEGIN WHILE */  
154  while (1)  
155  {  
156      Speed_RPM = MC_GetMecSpeedAverageMotor1()*6;  
157      Iqd_Ref_Value = MC_GetIqdrefMotor1();  
158      Iqd_Real_Value = MC_GetIqdMotor1();  
159      /* USER CODE END WHILE */  
160  
161      /* USER CODE BEGIN 3 */  
162      }  
163      /* USER CODE END 3 */  
164  }  
165
```



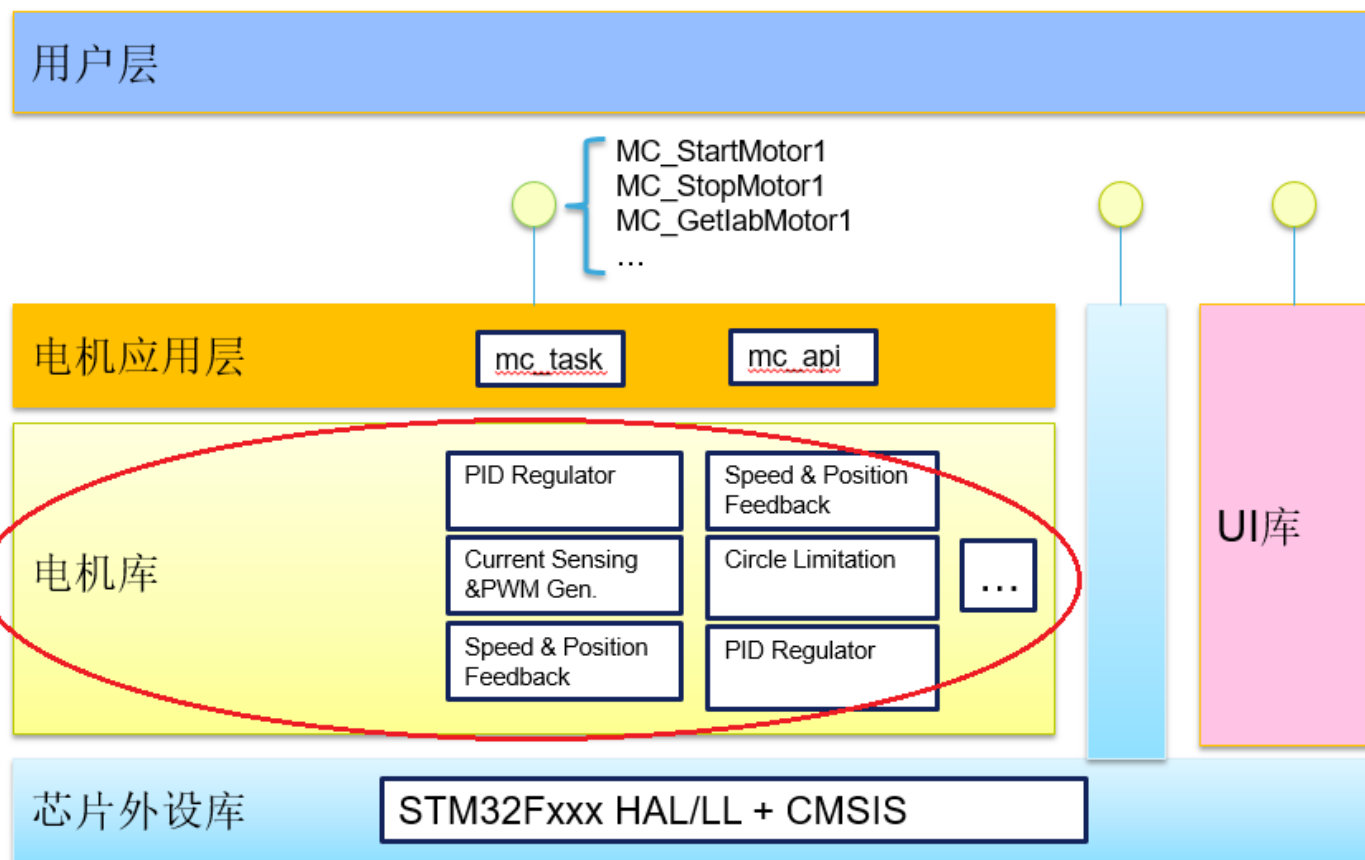
- ☐ API函数的使用
- ☐ 电机库底层调用与修改
- ☐ 电机状态切换以及异常处理
- ☐ 增加其他ADC采样以及控制引脚

电机库底层调用与修改

电机库在MC SDK5.x中的位置

给电机控制开发者
更大发挥空间!!!

- 涉及底层操作
- 可以调用在API中没有涉及的函数
- 修改需要熟悉电机运行框架
- 可以根据实际需求修改对应代码



电机库底层源文件

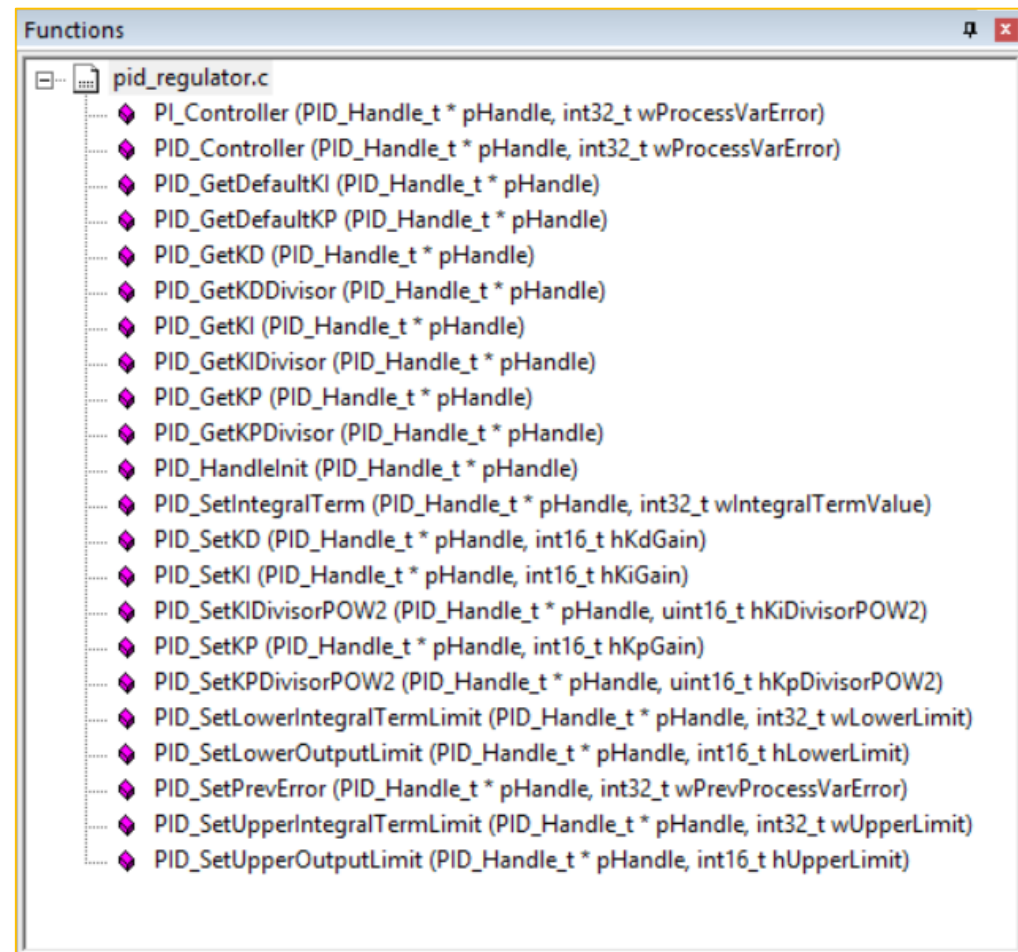
源文件	描述
bus_voltage_sensor.c	母线电压
circle_limitation.c	电压极限限制
enc_align_ctrl.c	编码器初始定位控制
encoder_speed_pos_fdbk.c	编码器传感器相关
hall_speed_pos_fdbk.c	Hall传感器相关
mc_math.c	数学计算
motor_power_measurement.c	平均功率计算
ntc_temperature_sensor.c	NTC温度传感
open_loop.c	开环控制
pid_regulator.c	PID环路控制
pqd_motor_power_measurement.c	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_mngr.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

电机库底层函数使用案例（1/4）

■ 案例：

得到速度PI参数，设定速度PI参数为当前的2倍

- PI相关函数可以参看pid_regulator.c
- 注意这边修改的PI参数不能超过参数范围
- 可以有两种方法进行修改

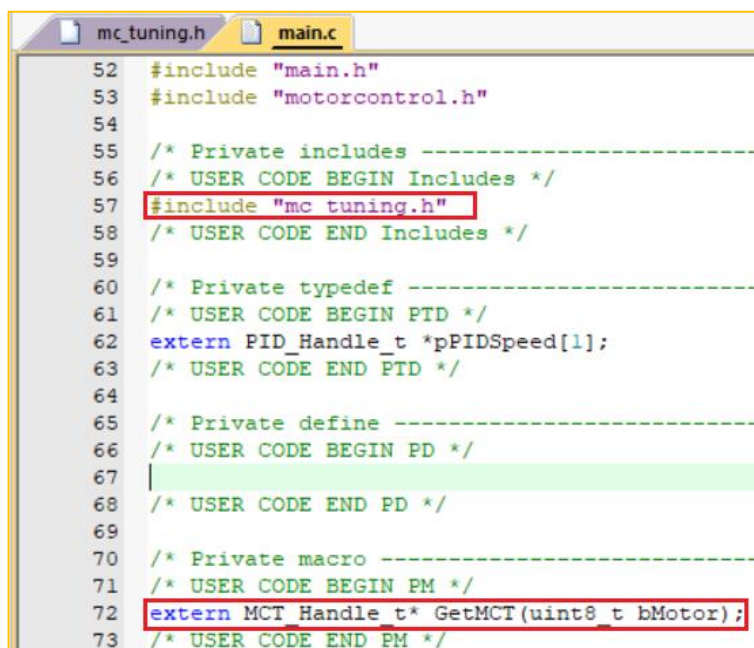


电机库底层函数使用案例（2/4）

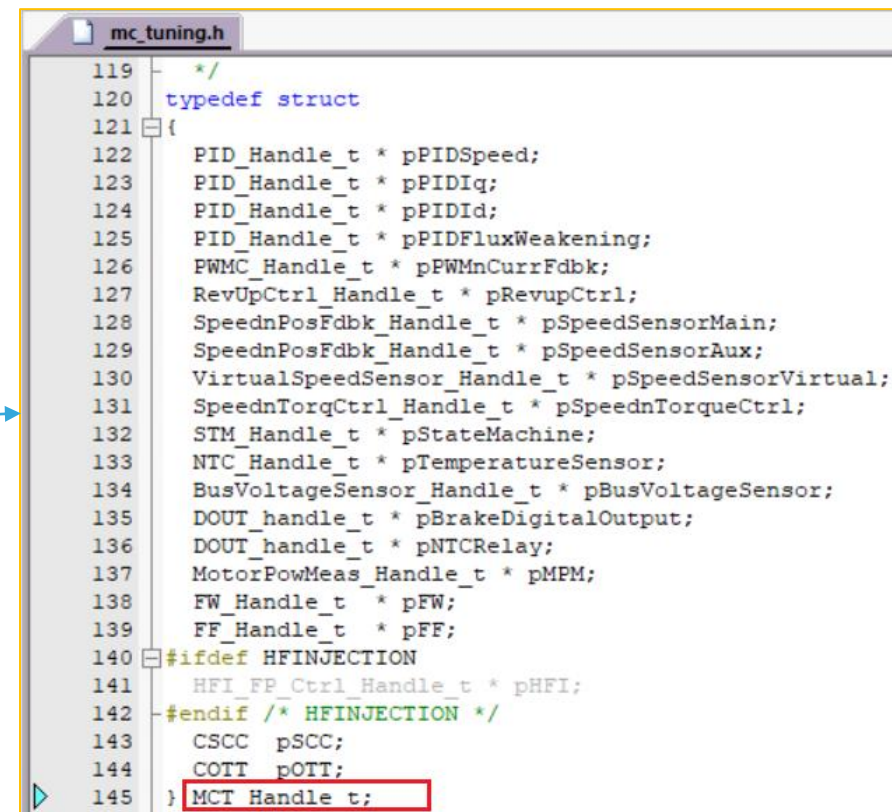
方法一： 采用与MC SDK V4.x版本类似方法，

得到内部修改（Tuning）的变量

- 需要包含mc_tuning.h头文件
- 需要在其他.c文件中重载GetMCT(uint8_t bMotor);



```
52 #include "main.h"
53 #include "motorcontrol.h"
54
55 /* Private includes -----
56 /* USER CODE BEGIN Includes */
57 #include "mc_tuning.h"
58 /* USER CODE END Includes */
59
60 /* Private typedef -----
61 /* USER CODE BEGIN PTD */
62 extern PID_Handle_t *pPIDSpeed[1];
63 /* USER CODE END PTD */
64
65 /* Private define -----
66 /* USER CODE BEGIN PD */
67
68 /* USER CODE END PD */
69
70 /* Private macro -----
71 /* USER CODE BEGIN PM */
72 extern MCT_Handle_t* GetMCT(uint8_t bMotor);
73 /* USER CODE END PM */
```



```
119 */
120 typedef struct
121 {
122     PID_Handle_t * pPIDSpeed;
123     PID_Handle_t * pPIDIq;
124     PID_Handle_t * pPIDId;
125     PID_Handle_t * pPIDFluxWeakening;
126     PWM_Handle_t * pPWMnCurrFdbk;
127     RevUpCtrl_Handle_t * pRevupCtrl;
128     SpeednPosFdbk_Handle_t * pSpeedSensorMain;
129     SpeednPosFdbk_Handle_t * pSpeedSensorAux;
130     VirtualSpeedSensor_Handle_t * pSpeedSensorVirtual;
131     SpeednTorqCtrl_Handle_t * pSpeednTorqueCtrl;
132     STM_Handle_t * pStateMachine;
133     NTC_Handle_t * pTemperatureSensor;
134     BusVoltageSensor_Handle_t * pBusVoltageSensor;
135     DOUT_handle_t * pBrakeDigitalOutput;
136     DOUT_handle_t * pNTCRelay;
137     MotorPowMeas_Handle_t * pMPM;
138     FW_Handle_t * pFW;
139     FF_Handle_t * pFF;
140 #ifdef HFINJECTION
141     HFI_FP_Ctrl_Handle_t * pHFI;
142 #endif /* HFINJECTION */
143     CSCC pSCC;
144     COTT pOTT;
145 } MCT_Handle_t;
```

电机库底层函数使用案例（3/4）

方法一：通过MCT接口对底层参数进行修改

```
/* Get recent Speed PI value */
Speed_Kp = PID_GetKP(pMctHdl->pPIDSpeed);
Speed_Ki = PID_GetKI(pMctHdl->pPIDSpeed);

/* Set new PI value */
PID_SetKP(pMctHdl->pPIDSpeed, Speed_Kp*2);
PID_SetKI(pMctHdl->pPIDSpeed, Speed_Ki*2);
```

```
main.c
133 MX_ADC1_Init();
134 MX_DAC_Init();
135 MX_TIM1_Init();
136 MX_USART2_UART_Init();
137 MX_MotorControl_Init();
138
139 /* Initialize interrupts */
140 MX_NVIC_Init();
141 /* USER CODE BEGIN 2 */
142
143 static int16_t Speed_Kp, Speed_Ki;
144 MCT_Handle_t *pMctHdl;
145
146 /* Get MCT handler */
147 pMctHdl = GetMCT(M1);
148
149 /* Get recent Speed PI value */
150 Speed_Kp = PID_GetKP(pMctHdl->pPIDSpeed);
151 Speed_Ki = PID_GetKI(pMctHdl->pPIDSpeed);
152
153 /* Set new PI value */
154 PID_SetKP(pMctHdl->pPIDSpeed, Speed_Kp*2);
155 PID_SetKI(pMctHdl->pPIDSpeed, Speed_Ki*2);
156
157 /* USER CODE END 2 */
```


电机库底层函数使用案例（4/4）

方法二：通过已定义变量对底层参数进行修改

- 实际上在mc_task.c中已经有PI变量定义，可以直接使用到其他*.c中

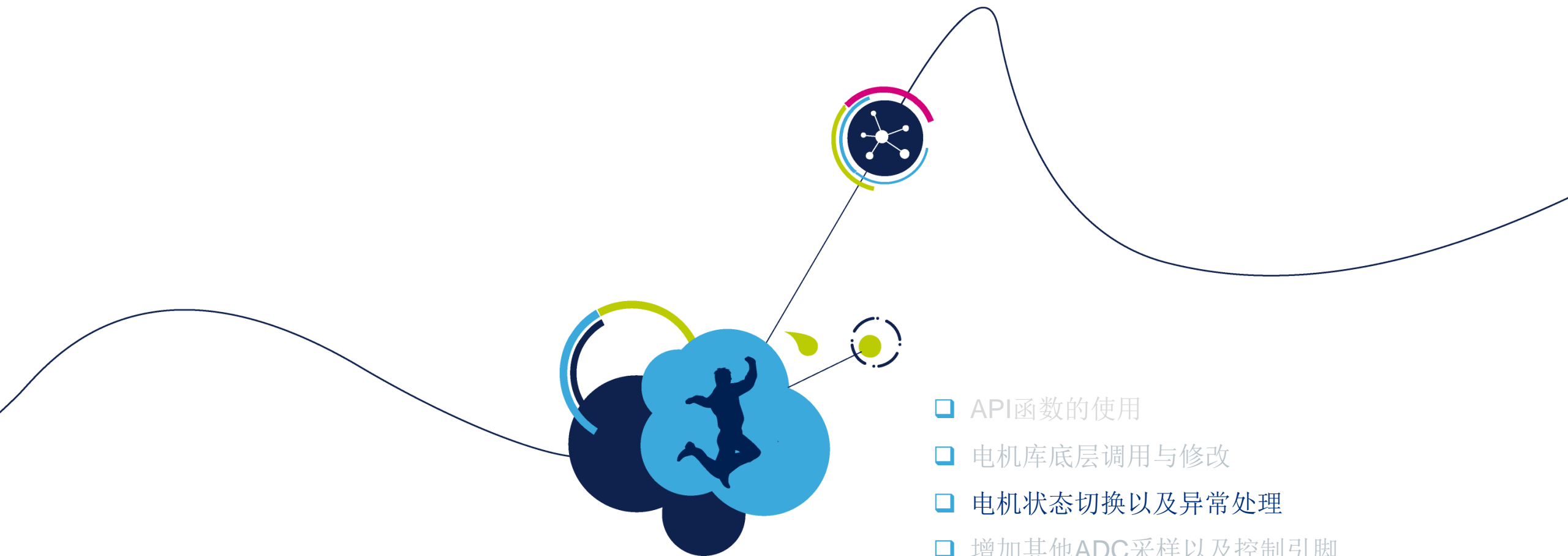
```
extern PID_Handle_t *pPIDSpeed[1];
```

```
Speed_Kp = PID_GetKP(pPIDSpeed[0]);  
Speed_Ki = PID_GetKI(pPIDSpeed[0]);
```

```
PID_SetKP(pPIDSpeed[0], Speed_Kp*2);  
PID_SetKI(pPIDSpeed[0], Speed_Ki*2);
```

```
main.c  
136 MX_USART2_UART_Init();  
137 MX_MotorControl_Init();  
138  
139 /* Initialize interrupts */  
140 MX_NVIC_Init();  
141 /* USER CODE BEGIN 2 */  
142  
143 extern PID_Handle_t *pPIDSpeed[1];  
144 static int16_t Speed_Kp, Speed_Ki;  
145  
146 /* Get recent Speed PI value */  
147 Speed_Kp = PID_GetKP(pPIDSpeed[0]);  
148 Speed_Ki = PID_GetKI(pPIDSpeed[0]);  
149  
150 /* Set new PI value */  
151 PID_SetKP(pPIDSpeed[0], Speed_Kp*2);  
152 PID_SetKI(pPIDSpeed[0], Speed_Ki*2);  
153  
154 MC_ProgramSpeedRampMotor1(3000/6, 1000);  
155 MC_StartMotor1();  
156  
157 /* USER CODE END 2 */  
158  
159 /* Infinite loop */  
160 /* USER CODE BEGIN WHILE */  
161 while (1)  
162 {
```

```
main.c mc_tasks.c  
86 /* #define MC.SMOOTH BRAKING ACTION ON OVERVOLTAGE */  
87  
88 /* USER CODE END Private define */  
89  
90 /* Private variables-----  
91 FOCVars_t FOCVars[NBR_OF_MOTORS];  
92 MCI_Handle_t Mci[NBR_OF_MOTORS];  
93 MCI_Handle_t * oMCInterface[NBR_OF_MOTORS];  
94 MCT_Handle_t MCT[NBR_OF_MOTORS];  
95 STM_Handle_t STM[NBR_OF_MOTORS];  
96 SpeednTorqCtrl_Handle_t *pSTC[NBR_OF_MOTORS];  
97 PID_Handle_t *pPIDSpeed[NBR_OF_MOTORS];  
98 PID_Handle_t *pPIDIq[NBR_OF_MOTORS];  
99 PID_Handle_t *pPIDId[NBR_OF_MOTORS];  
100 RDivider_Handle_t *pBusSensorM1;  
101 NTC_Handle_t *pTemperatureSensor[NBR_OF_MOTORS];  
102 PWMCHandle_t *pwmCHandle[NBR_OF_MOTORS];  
103 DOUT_handle_t *pR_Brake[NBR_OF_MOTORS];  
104 DOUT_handle_t *pOCPDisabling[NBR_OF_MOTORS];  
105 PQD_MotorPowMeas_Handle_t *pMPM[NBR_OF_MOTORS];  
106 CircleLimitation_Handle_t *pCLM[NBR_OF_MOTORS];  
107 RampExtMngr_Handle_t *pREMNG[NBR_OF_MOTORS];  
108
```

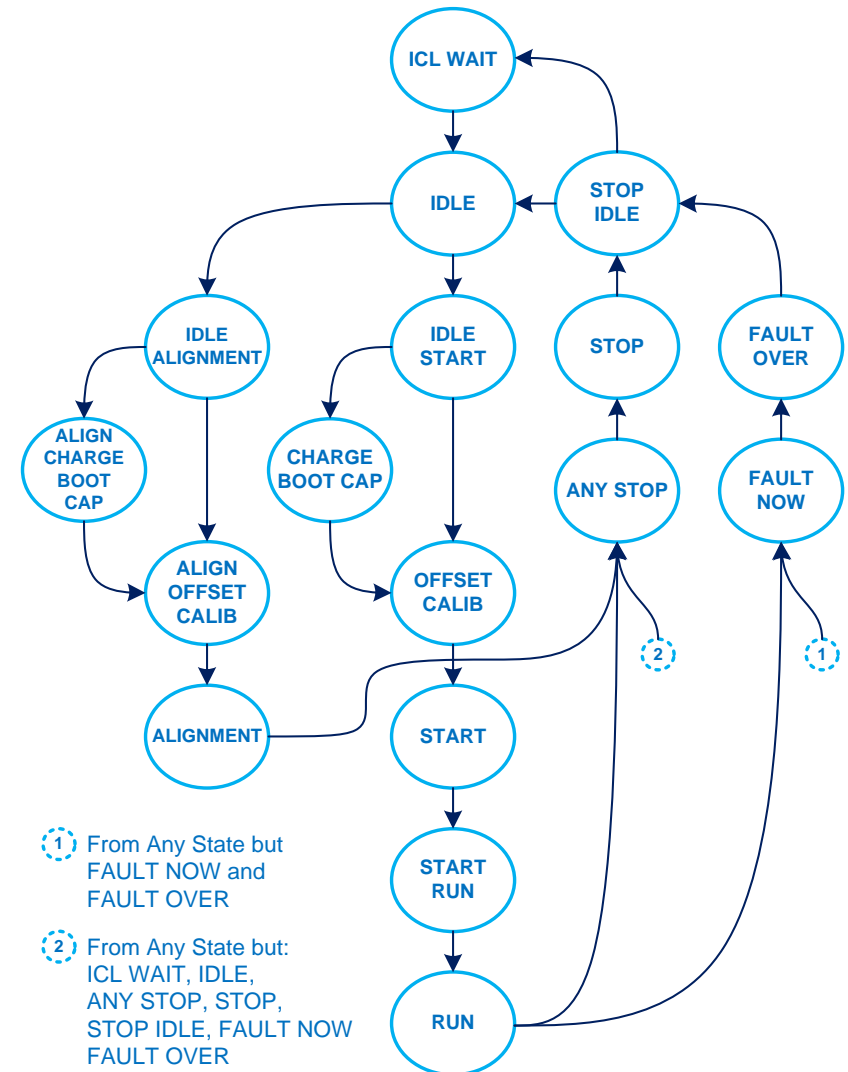


- ☐ API函数的使用
- ☐ 电机库底层调用与修改
- ☐ 电机状态切换以及异常处理
- ☐ 增加其他ADC采样以及控制引脚
- ☐ 有传感器使用配置

电机状态切换以及异常处理

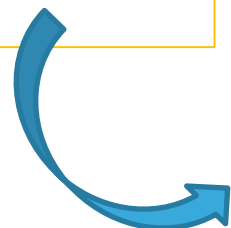
电机控制状态机

- 程序中都有相应的状态机来控制电机状态。
- 用户不可以直接操作状态机，而应该通过 SDK 提供的 API 来改变电机状态。
- 如果添加客户自定义状态机，需要符合电机控制库规则。



状态返回值

- 调用函数MC_GetSTMStateMotor1
将返回当前状态



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

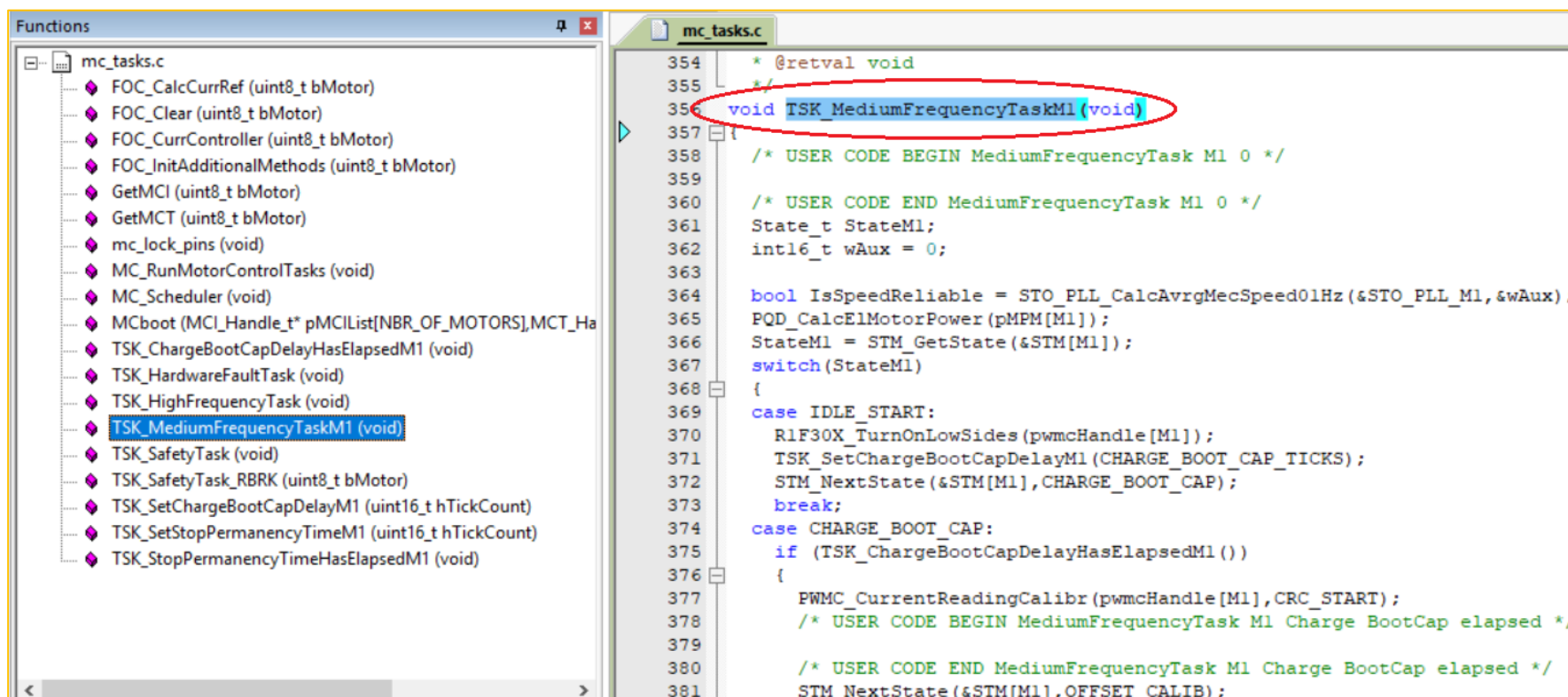
故障状态返回值

- 调用函数MC_GetOccurredFaultsMotor1将返回刚刚发生了的Fault状态
- 调用函数MC_GetCurrentFaultsMotor1将返回发生了的Fault状态，返回值可能是几种情况的组合

状态返回值	枚举值	状态描述
MC_NO_FAULTS	0x0000	无故障
MC_FOC_DURATION	0x0001	FOC算法溢出
MC_OVER_VOLT	0x0002	过电压故障
MC_UNDER_VOLT	0x0004	欠电压故障
MC_OVER_TEMP	0x0008	过温度故障
MC_START_UP	0x0010	启动失败
MC_SPEED_FDBK	0x0020	速度失控
MC_BREAK_IN	0x0040	过电流故障
MC_SW_ERROR	0x0080	软件故障

电机状态机位置

- 电机控制状态机在mc_task.c文件的
TSK_MediumFrequencyTaskM1函数中



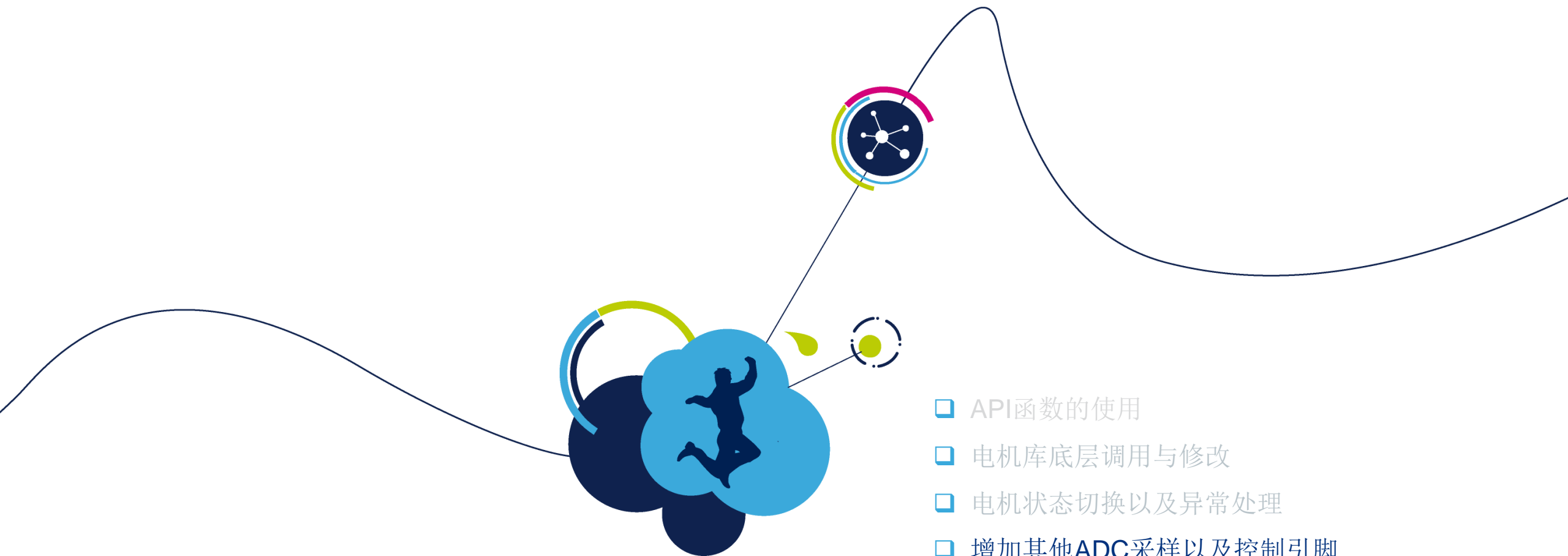
```
354  * @retval void
355  */
356  void TSK_MediumFrequencyTaskM1(void)
357  {
358      /* USER CODE BEGIN MediumFrequencyTask M1 0 */
359
360      /* USER CODE END MediumFrequencyTask M1 0 */
361      State_t StateM1;
362      int16_t wAux = 0;
363
364      bool IsSpeedReliable = STO_PLL_CalcAvrgMecSpeed01Hz (&STO_PLL_M1, &wAux);
365      PQD_CalcElMotorPower (pMPM[M1]);
366      StateM1 = STM_GetState (&STM[M1]);
367      switch (StateM1)
368      {
369          case IDLE_START:
370              R1F30X_TurnOnLowSides (pwmCHandle[M1]);
371              TSK_SetChargeBootCapDelayM1 (CHARGE_BOOT_CAP_TICKS);
372              STM_NextState (&STM[M1], CHARGE_BOOT_CAP);
373              break;
374          case CHARGE_BOOT_CAP:
375              if (TSK_ChargeBootCapDelayHasElapsedM1 ())
376              {
377                  PPMC_CurrentReadingCalibr (pwmCHandle[M1], CRC_START);
378                  /* USER CODE BEGIN MediumFrequencyTask M1 Charge BootCap elapsed */
379
380                  /* USER CODE END MediumFrequencyTask M1 Charge BootCap elapsed */
381                  STM_NextState (&STM[M1], OFFSET_CALIB);
```

电机状态函数使用案例

■ 案例：

得到电机状态，得到电机故障状态，如果是失速故障自动清除

```
static uint8_t State_Mark;  
static uint8_t Fault_Mark;  
  
/* Get motor control state*/  
State_Mark = MC_GetSTMStateMotor1();  
  
/* Get Fault state*/  
Fault_Mark = MC_GetOccurredFaultsMotor1();  
  
if(Fault_Mark == MC_SPEED_FDBK)  
{  
    /* Clear fault state*/  
    MC_AcknowledgeFaultMotor1();  
}  
if((State_Mark == IDLE) && (Fault_Mark == MC_NO_FAULTS))  
{  
    MC_StartMotor1();  
}
```

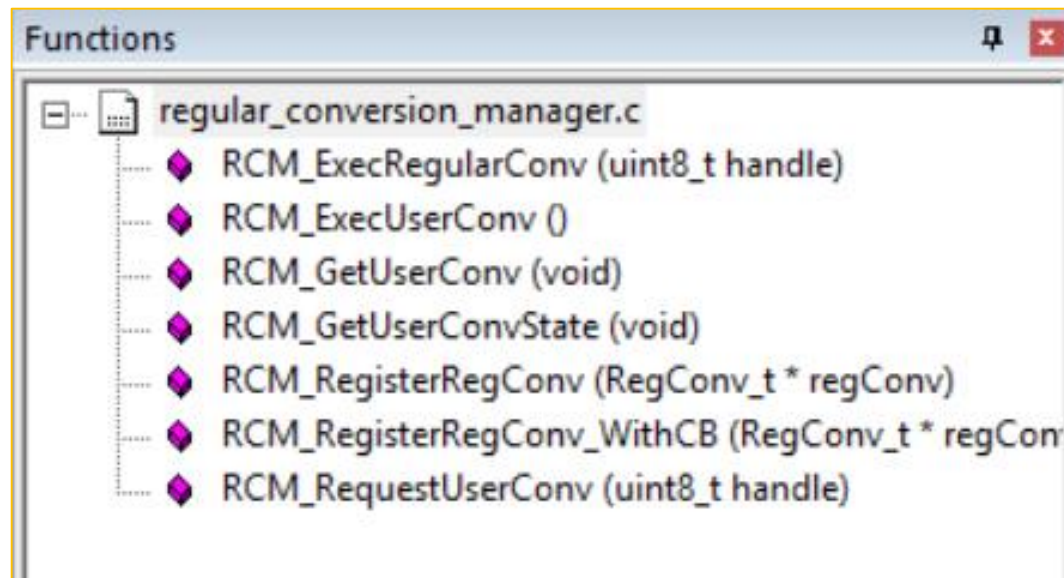


- ☐ API函数的使用
- ☐ 电机库底层调用与修改
- ☐ 电机状态切换以及异常处理
- ☐ 增加其他**ADC**采样以及控制引脚
- ☐ 有传感器使用配置

增加其他ADC采样以及控制引脚

增加其他ADC采样

- 方便客户增加自行的ADC采样，同时又不会影响电机控制的ADC采样
- 电机库特意为客户增加了相关可调用函数
- 文件名称：regular_conversion_manager.c

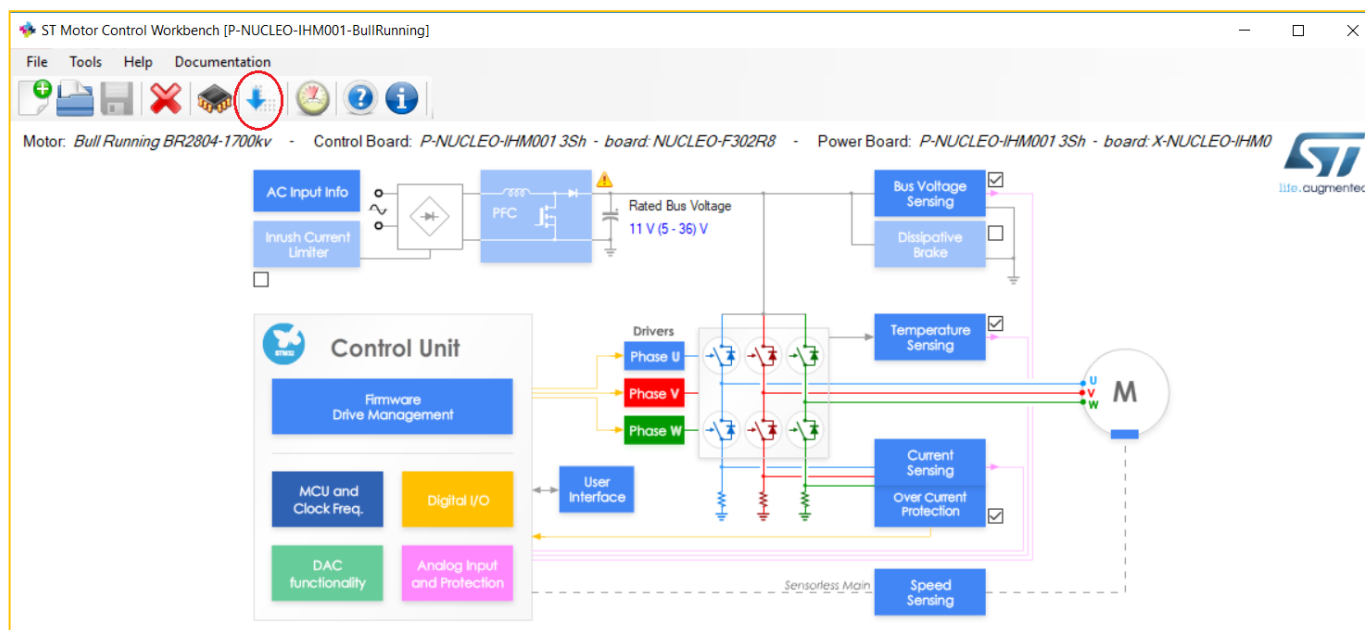


ADC采样以及控制引脚案例（1/6）

■ 案例：

增加PB1（ADC1_CH12）作为旋钮电压输入,PB0作为控制引脚输出

步骤一：使用Workbench生成程序工程，本例子中使用P-NUCLEO-IHM001-BullRunning工程作为案例



ADC采样以及控制引脚案例（2/6）

步骤二：打开CubeMx工程P-NUCLEO-IHM001-BullRunning.ioc
添加PB1为ADC_IN12，PB0为GPIO_Output

ADC1 Mode and Configuration

Mode	
IN4	Disable
IN5	Disable
IN6	IN6 Single-ended
IN7	IN7 Single-ended
IN8	IN8 Single-ended
IN9	Disable
IN10	Disable
IN11	Disable
IN12	IN12 Single-ended
IN13	Disable
IN14	Disable

Pinout diagram of the STM32F302R8Tx LQFP64 package. The diagram shows the following connections:

- PC5: VSS..
- PC6: VDD..
- PC7: VSS..
- PC8: VDD..
- PC9: PA0
- PC10: PA1
- PC11: PA2
- PC12: PA3
- PC13: VSS
- PC14: VDD
- PC15: PA4
- PC16: PA5
- PC17: PA6
- PC18: PA7
- PC19: PC4
- PC20: PC5
- PC21: PB0
- PC22: PB1
- PC23: PB2
- PC24: PB10
- PC25: PB11
- PC26: VSS
- PC27: VDD
- PC28: PC6
- PC29: PB15
- PC30: PB14
- PC31: PB13
- PC32: PB12

Labels for specific pins and functions:

- UART_RX: PA3
- DBG_DAC_CH1: PA4
- GPIO_Output: PB0
- ADC1_IN12: PB1

ADC采样以及控制引脚案例（3/6）

步骤三：修改PB0的引脚输出配置，本案修改为Pull-up 的 Push-Pull输出
使用CubeMx重新生成工程

Configuration

☐ Group By Peripherals

☒ GPIO ☒ ADC1 ☒ DAC ☒ RCC ☒ TIM1 ☒ USART2 ☒ NVIC

Pin Name	Signal on Pin	GPIO output ...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PB0	n/a	Low	Output Push...	Pull up	Medium	n/a		✓
PC10	n/a	Low	Output Push...	Pull down	High	n/a	M1_PWM_E...	✓
PC11	n/a	Low	Output Push...	Pull down	High	n/a	M1_PWM_E...	✓
PC12	n/a	Low	Output Push...	Pull down	High	n/a	M1_PWM_E...	✓
PC13	n/a	n/a	External Inte...	No pull up p...	n/a	n/a	Start/Stop [P...	✓

PB0 Configuration :

GPIO output level: Low

GPIO mode: Output Push Pull

GPIO Pull-up/Pull-down: Pull up

Maximum output speed: Medium

User Label:



10th Anniversary

f y t

GENERATE CODE

Tools

ST

ADC采样以及控制引脚案例（4/6）

步骤四：包含头文件以及定义ADC采样相关变量

```
/* Private includes -----  
/* USER CODE BEGIN Includes */  
#include "regular_conversion_manager.h"  
/* USER CODE END Includes */  
  
/* Private typedef -----  
/* USER CODE BEGIN PTD */  
  
/* USER CODE END PTD */  
  
/* Private define -----  
/* USER CODE BEGIN PD */  
RegConv_t  ADC_UserConv;  
uint8_t ADC_UserHandle;  
uint16_t ADC_UserValue;  
/* USER CODE END PD */
```

ADC采样以及控制引脚案例（5/6）

步骤五：添加ADC端口配置初始化程序以及控制引脚电平

```
/* Initialize interrupts */
MX_NVIC_Init();
/* USER CODE BEGIN 2 */
ADC_UserConv.regADC = ADC1 ;
ADC_UserConv.channel = ADC_CHANNEL_12;
ADC_UserConv.samplingTime = ADC_SAMPLETIME_61CYCLES_5;
ADC_UserHandle = RCM_RegisterRegConv (&ADC_UserConv);

HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0,GPIO_PIN_SET);

/* USER CODE END 2 */
```

ADC通道以及
采样时间配置

PB0输出高电平

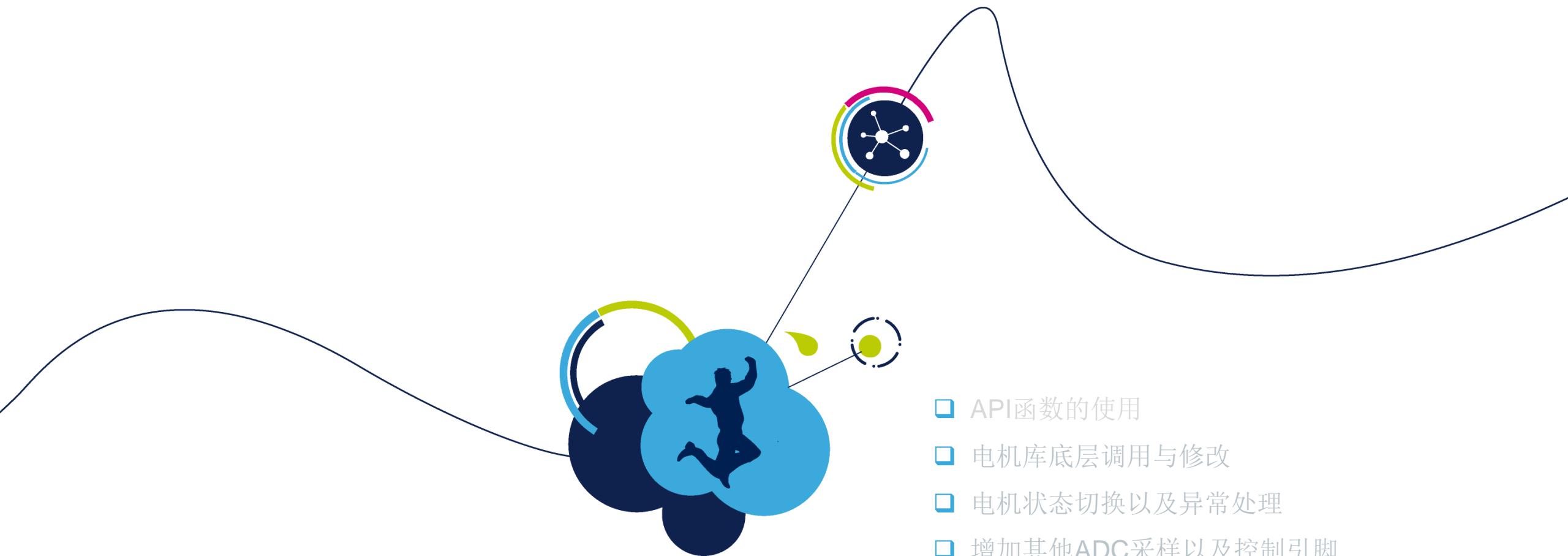
ADC采样以及控制引脚案例（6/6）

步骤六：调用用户ADC转换函数进行ADC采样，并得到采样结果

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* Check regular conversion readiness */
    if (RCM_GetUserConvState() == RCM_USERCONV_IDLE) 判断是否可以开始转换用户ADC
    {
        /* if Idle, then program a new conversion request */
        RCM_RequestUserConv(ADC_UserHandle);
    }
    else if (RCM_GetUserConvState() == RCM_USERCONV_EOC) 转换结束
    {
        /* if Done, then read the captured value */
        ADC_UserValue = RCM_GetUserConv();
    }
}
/* USER CODE END WHILE */
```



Watch 1		
Name	Value	Type
ADC_UserValue	0x08B0	unsigned short
<Enter expression>		



- ☐ API函数的使用
- ☐ 电机库底层调用与修改
- ☐ 电机状态切换以及异常处理
- ☐ 增加其他ADC采样以及控制引脚
- ☐ 有传感器使用配置

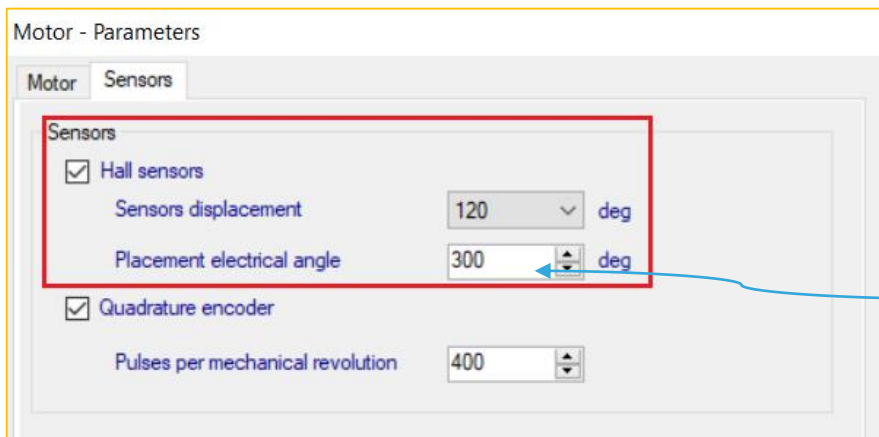
有传感器使用配置

Hall传感器使用案例（1/3）

■ 案例： 使用Hall传感器模式使用案例

配置一：在Motor Sensors勾选Hall Sensors

- 选择Hall安装方式120度或者60度
- 输入Hall相关的同步电角度数据，默认为300度
- 精确的同步电角度测试说明见<https://www.stmcu.com.cn/>上的文档说明



首页 / 设计资源 / 文档下载



电机控制同步电角度测试说明

版本: 1

下载次数: 469次 更新日期: 2017-03-21

立即下载

☆ 收藏



文件说明

在使用ST FOC电机库时，当使用Hall信号作为位置信号时，需要输入同步电角度数据，这个数据根据当前使用电机的特性进行输入，会在每次Hall信号变化时同步电角度，如果角度偏差较大时会影响控制效果，可能带来效率或者电机的震荡，初始测试还是有必要的，本文详细说明测试注意事项以及测试方法。

Hall传感器使用案例（2/3）

配置二：在Driver Management的速度传感选择中选择Hall Sensors

Drive Management - Speed Position Feedback Management

Main sensor | Auxiliary sensor

Sensor selection: Hall sensors

Max measurement errors number before fault: 3

Hall Sensors

Average speed FIFO depth: 6

Input Capture filter duration: 1.3 usec

平均速度计算缓冲深度

输入信号滤波时间

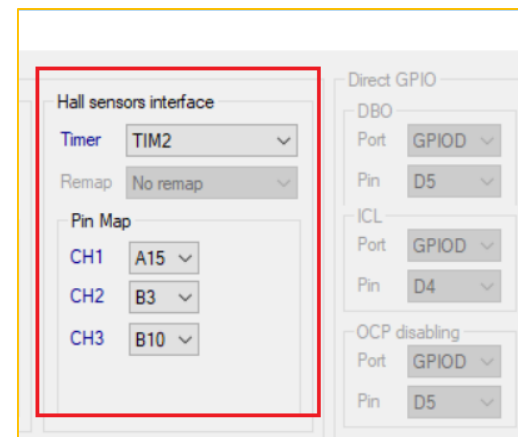
Hall传感器使用案例（3/3）

配置三：Control Stage中配置正确的Hall信号引脚接口

注意:电机的Hall信号（H1,H2,H3）要与三相电机输出控制线OUT1，2，3对应起来，
否则电机堵转或者过流

- 对于电机Shinano LA052-080E3NL1，对应关系如下：

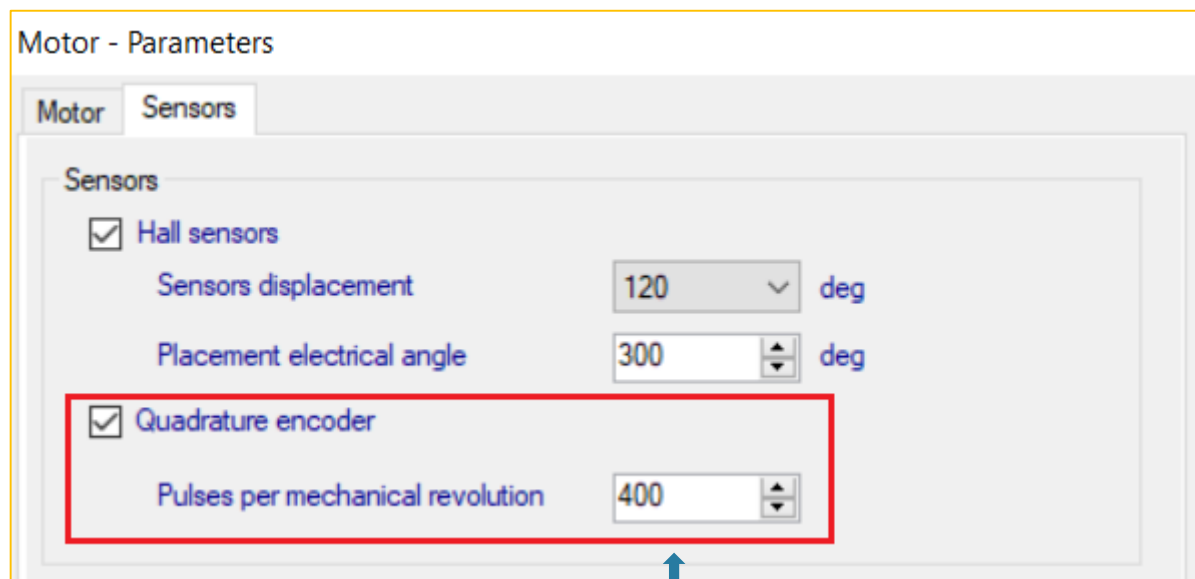
X-Nucleo-IHM07M1	Motor: Shinano LA052-080E3NL1	Note
OUT1	BLACK	W1
OUT2	WHITE	W2
OUT3	RED	W3
A.	ORANGE	HALL1
B.	GREEN	HALL2
Z.	YELLOW	HALL3
5V	BROWN	5V
GND	BLUE	GND



Encoder使用案例（1/4）

配置一：在Motor Sensors勾选Quadrature encoder

- 填入正确的Encoder线数参数
- 这边是旋转一圈编码器的脉冲个数
- 对于Shinano LA052-080E3NL1，该参数为400



旋转一圈的编码器单向脉冲个数

Encoder使用案例（2/4）

配置二：在Driver Management的速度传感选择中选择Quadrature encoder

Drive Management - Speed Position Feedback Management

Main sensor | Auxiliary sensor

Sensor selection: Quadrature encoder

Max measurement errors number before fault: 3

Quadrature Encoder

Average speed FIFO depth: 16

Input Capture filter duration: 0.7 usec

Reverse counting direction: ☐

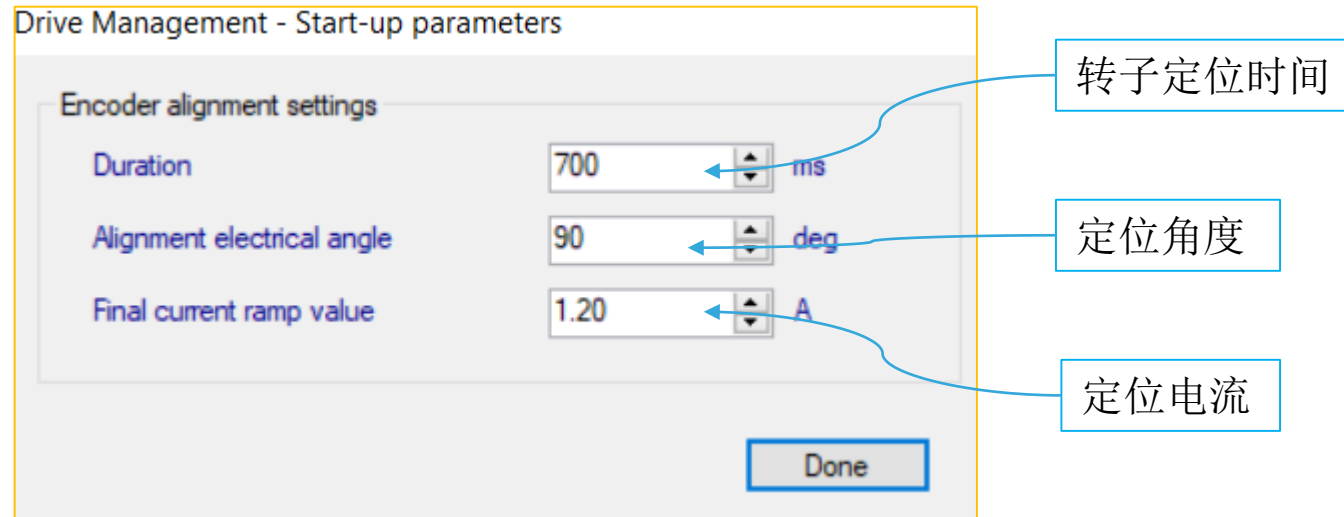
平均速度计算缓冲深度

输入信号滤波时间

Encoder使用案例（3/4）

配置三：在Driver Management的Start-up参数上根据电机调整定位参数

- 因为是增量编码器，需要将转子定位到知道角度的位置才可以正常启动运行
- 定位时间以及定位的电流可根据具体电机进行调整



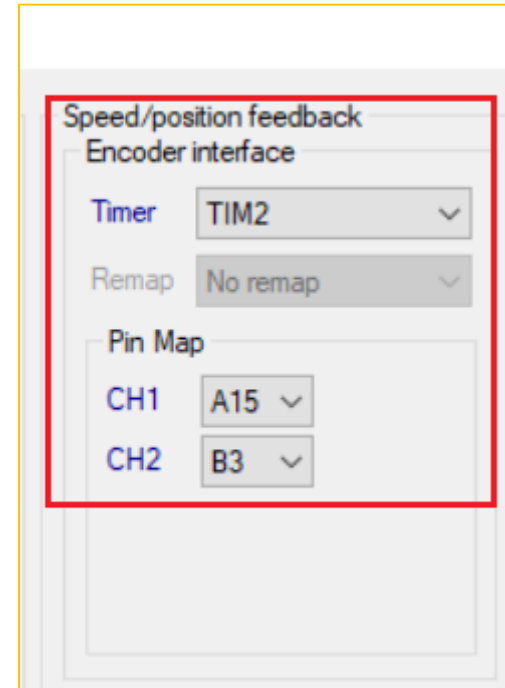
Encoder使用案例（4/4）

配置四：Control Stage中配置正确的Encoder信号引脚接口

注意:电机的Encoder信号（EA，EB）要与三相电机输出控制线OUT1，2，3对应起来，
否则电机堵转或者过流

- 对于电机Shinano LA052-080E3NL1，对应关系如下：

X-Nucleo-IHM07M1	Motor: Shinano LA052-080E3NL1	Note
OUT1	BLACK	W1
OUT2	WHITE	W2
OUT3	RED	W3
A.	BLUE	EA
B.	YELLOW	EB
Z.		
5V	RED	5V
GND	BLACK	GND



Releasing your creativity



- Thank you -

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。

版权声明

本文档为意法半导体公司及其子公司（“ST”）版权所有，未经ST允许不得复制、修改、转发或应用于商业目的。