AN5143
应用笔记

如何将电机控制应用程序软件从 SDK v4.3 迁移至 SDK v5.x

引言

STM32电机控制软件开发套件（MC SDK）是STMicroelectronics电机控制生态系统的一部分。根据所应用的软件许可协议，有两个版本，X-CUBE-MCSDK以及X-CUBE-MCSDK-FUL。它包括：

- 用于永磁同步电机（PMSM）磁场定向控制（FOC）的ST MC FOC固件库
- ST MC工作站软件工具，这是一个用于配置MC SDK固件库参数的图形用户界面

本应用笔记有助于将电机控制应用软件从SDK v4.3 迁移至 SDK v5.x 框架。它涵盖了固件方面以及MC软件工具的使用。

注： *SDK v5.x 必须用于新项目，因为它为用户提供了新的API。*

目录

表格索引

图片索引

1 概述

MC SDK用于开发基于Arm®(a) Cortex®-M处理器的STM32 32位微控制器上运行的电机控制应用程序。

*表 1*给出了相关的缩略语定义，帮助您更好地理解本文档。

表1. 缩略语列表

缩略语	说明
API	应用编程接口
GUI	图形用户界面
HAL	硬件抽象层
ICS	隔离电流传感器
IDE	集成开发环境
FOC	磁场定向控制
FW	固件
HFI	高频注入
LL	低电平
MC	电机控制
MC WB	电机控制工作台（STMicroelectronics软件工具）
MP	电机分析仪（STMicroelectronics软件工具）
MTPA	每安培最大转矩
PFC	功率因数校正
PMSM	永磁同步电机
SDK	软件开发套件
SW	软件
SPL	标准外设库

a. Arm是Arm Limited（或其子公司）在美国和/或其他地区的注册商标。

2 相关文档

Arm® 文档

以下文档可从infocenter.arm.com 网页获得：

1. Cortex®-M0技术参考手册
2. Cortex®-M3技术参考手册
3. Cortex®-M4技术参考手册

STMicroelectronics文档

以下文档可从*www.st.com*网页获得：

4. STM32F0系列产品数据表
5. STM32F1系列产品数据表
6. STM32F2系列产品数据表
7. STM32F3系列产品数据表
8. STM32F4系列产品数据表
9. *X-NUCLEO扩展板电机控制 - 选型指南* 在线演示

3 SDK v5.x 与SDK v4.3 比较总结

表 2 是针对主要迁移问题对 SDK v5.x 与 SDK v4.3 进行对比的结果概述。

表2. SDK v5.x 与 SDK v4.3 比较总结

迁移问题	SDK v4.3	SDK v5.x
软件安装	– 电机分析仪 – MC工作站 [4.3] – ST-LINK/V2	– 电机分析仪 – MC工作站 [5.0.0或更高版本] – STM32CubeMX [4.24.0或更高版本] – ST-LINK/V2
支持的微控制器	仅限 STM32F0、STM32F1、STM32F2、STM32F3和STM32F4系列。	– STM32F0、STM32F3和STM32F4系列是5.0版本的一部分。 – STM32F1系列是5.1版本的一部分。 – 其他系列，例如STM32F7、STM32L4和STM32H7计划将在后续版本中发布。
特性	– 磁场定向控制 – 双电机 – 弱磁 – 前馈 – MTPA – PFC – 速度传感器（霍尔/编码器/HFI/无传感器） – 电流传感器（1个分流 / 3个分流 / ICS）	– 磁场定向控制(1) – 双电机(1) – 弱磁(1) – 前馈(1) – MTPA(1) – 速度传感器（霍尔/编码器/无传感器）(1) – 电流传感器（1个分流 / 3个分流 / ICS）(1) – PFC(2) – HFI(3)
处理ST板	直接从MC WB。	直接从MC WB。
处理定制控制板	直接从MC WB，但只有数量有限的MCU。	可通过STMicroelectronics参考板和使用STM32CubeMX （与 SDK v4.3 中的MCU相同）来实现。
架构	三大主要部分： – MCApplication – MCLibrary – UI_Library	三大主要部分，从 SDK v4.3重新排列： – 电机座舱 – 电机控制库 – 用户界面库
工作区	需要两个工作区： – Library – 用户应用程序代码	只需要一个工作空间，用于管理中间件和用户应用程序代码。
API	请参见*第 5节：开发工作流 第 11页*。	SDK v4.3 用作参考的API： - 经过简化； - 每台电机专用。
编码风格	面向对象的C代码。	多维数据集架构C代码。
使用驱动程序	SPL	HAL - LL
外设初始化	– 通过#define在固件主文件中完成 – 代码中存在所有MCU代码	– 由 STM32CubeMX 自动管理 – 仅生成所需代码

表2. SDK v5.x 与 SDK v4.3 比较总结（续）

迁移问题	SDK v4.3	SDK v5.x
可读性	由于以下原因，主要代码难以理解和修改： - 文件尺寸大 - 多个 #define 部分需要由用户来理解和处理	– 由 STM32CubeMX 自动管理 – 仅生成所需代码
调试	不直接： - 数据已隐藏 - 使用虚拟功能	很容易，因为没有数据隐藏，且调用了直接功能。
MIPS	-	单电机控制CPU工作负荷减小量： – STM32F072RB高达13% – STM32F303RE高达3.5% – STM32F446RE高达8.1% 双电机控制CPU工作负荷减小量： – STM32F415ZG高达20.8%
存储器容量	-	单电机控制马达控制库代码减小量： – STM32F072RB高达29.2% – STM32F303RE高达21.9% – STM32F446RE高达17% 双电机控制马达控制库代码减小量： – STM32F415ZG高达14.9%
MC分析仪	-	与SDK v4.3相同。按计划更新所支持的板。(3)
MC工作站	– 仅头文件生成 – 手动项目生成：生成的头文件将复制到用户项目工作区中 – 用户必须确保项目一致性	– 完整的C项目生成（适用于IDE） – 自动完成项目生成（使用STM32CubeMX）包括初始化代码和工具链项目文件 – 支持IAR Embedded Workbench® for Arm® (IAR Systems® AB)或µVision® IDE for Arm® (Keil® MDK)(4)
MC监视器	-	与 SDK v4.3 相同。

1. v5.0版本以上提供。
2. v5.1版本以上提供。
3. 后续版本提供。
4. SDK v5.1或更高版本支持Atollic® TrueSTUDIO® 架构。

4 支持的STMicroelectronics板

表 3 列出了 SDK v4.3 与 SDK v5.x 共同支持的STMicroelectronics控制板和逆变器板。

表3. SDK v4.3 与 SDK v5.x 共同支持的控制板和逆变器板

MCU	系列	MC WB 参考	MC WB 类型	STM32CubeMX板 参考
STM32F030R8Tx	STM32F0	NUCLEO-F030R8(1)	控制板	NUCLEO-F030R8
STM32F072RB		NUCLEO-F072RB(1)	控制板	NUCLEO-F072RB
STM32F072VBTx		STM32072B-EVAL(1)	控制板	STM32072B-EVAL
STSPIN32F0		STEVAL-SPIN3201(2)	逆变器板	-
STM32F103ZGTx	STM32F1	STM3210E-EVAL(2)	控制板	STM3210E-EVAL
STM32F103RC		STEVAL-IHM034V2(2)	逆变器板	-
STM32F302R8Tx	STM32F3	NUCLEO-F302R8(1)	控制板	NUCLEO-F302R8
STM32F303RETx		NUCLEO-F303RE(1)	控制板	NUCLEO-F303RE
STM32F303VETx		STM32303E-EVAL(1)	控制板	STM32303E-EVAL
STM32F446RETx	STM32F4	NUCLEO-F446RE(1)	控制板	NUCLEO-F446RE
STM32F407IGHx		STM3240G-EVAL(1)	控制板	STM3240G-EVAL
STM32F417IGHx		STM3241G-EVAL(1)	控制板	STM3241G-EVAL
STM32F446ZETx		STM32446E-EVAL(1)	控制板	STM32446E-EVAL
STM32F415ZGT8		STEVAL-IHM039V1(1)	逆变器板	-

1. SDK v5.0以上的版本支持。
2. SDK v5.1以上的版本支持。

表 4 列出了仅SDK v5.1 支持的其他STMicroelectronics控制板和逆变器板。

表4. 仅 SDK v5.1 支持的其他控制板和逆变器板

MCU	系列	MC WB 参考	MC WB 类型	STM32CubeMX板 参考
STSPIN32F0A	STM32F0	STEVAL-SPIN3202	逆变器板	-
STM32F103RBT6	STM32F1	NUCLEO-F103RB	控制板	NUCLEO-F103RB
STM32F303CBT7	STM32F3	STEVAL-ESC001V1	逆变器板	-
STM32F303RETx		X-NUCLEO-IHM16M1、NUCLEO-F303RE	逆变器板	-

表 5 列出了 SDK v4.3 与 SDK v5.0 共同支持的STMicroelectronics功率板。

表5. SDK v4.3 与 SDK v5.0 共同支持的功率板

MC WB 参考	MC WB 类型
STEVAL-IHM023V3	电源板
STEVAL-IHM025V1	电源板
STEVAL-IHM028V2	电源板
STEVAL-IHM045V1	电源板
STEVAL-IPM05F	电源板
STEVAL-IPM07F	电源板
STEVAL-IPM10B	电源板
STEVAL-IPM10F	电源板
STEVAL-IPM15B	电源板
X-NUCLEO-IHM07M1	电源板
X-NUCLEO-IHM08M1	电源板
X-NUCLEO-IHM11M1	电源板

表 6 列出了仅SDK v5.1 支持的其他STMicroelectronics功率板。

表6. 仅 SDK v5.1 支持的其他功率板

MC WB 参考	MC WB 类型
STEVAL-IPM08B	电源板
STEVAL-IPMNG3Q	电源板
STEVAL-IPMNG5Q	电源板
STEVAL-IPMNG8Q	电源板
STEVAL-IPMNM1N	电源板
STEVAL-IPMNM2N	电源板

5 开发工作流

支持以下IDE（请参阅测试版本的版本说明）：

- IAR Embedded Workbench® for Arm® (IAR Systems® AB)
- MDK tools for Arm® (Keil® MDK)
- TrueSTUDIO® for STM32 (Atollic®)

在MCSDK v5.x中，开发工作流与以下例外SDK v4.3类似，可提供更高的效率，改善客户体验：

- 用户无需将MC工作站输出文件复制到其项目工作区
- STM32CubeMX 可用于开发用户软件应用程序

使用SDK v5.xMC工作站工具打开SDK v4.3项目时，将显示一个迁移窗口，其中会显示输出的文件格式从SDK v4.3转换为SDK v5.x。

开发工作流SDK v5.x 请参见 *图 1第 12页* 介绍。对于SDK v4.3，执行开发工作流的以下步骤：

- 电机分析仪可用于识别主要PMSM特性，后者进一步被传输至MC工作站。
- MC工作站是启动新项目的主要入口点。

此外，它提供以下演变特性特征：

- 从MC工作站连接STM32CubeMX ，并从后台调用，以生成所选的IDE项目工作框架。
- 用户可以使用STM32CubeMX，其IDE，或两者来自定义生成的项目。
- 用户可以使用MC工作站监视器功能调整其应用程序，如*图 2第 12页*所示。

图1. 开发工作流

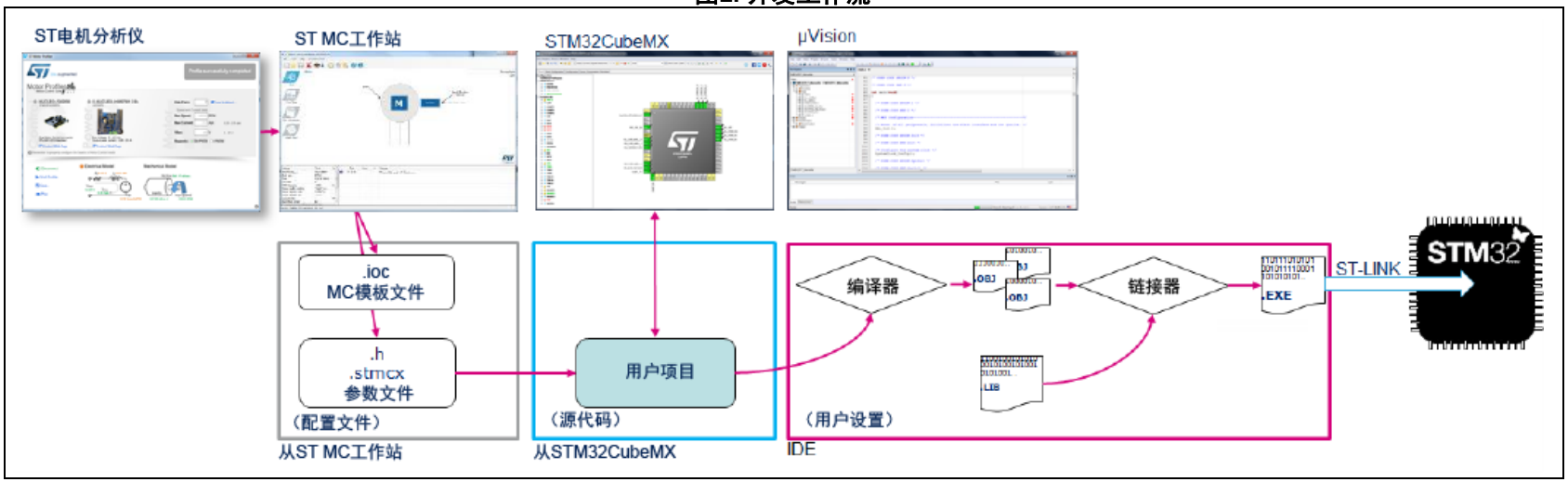
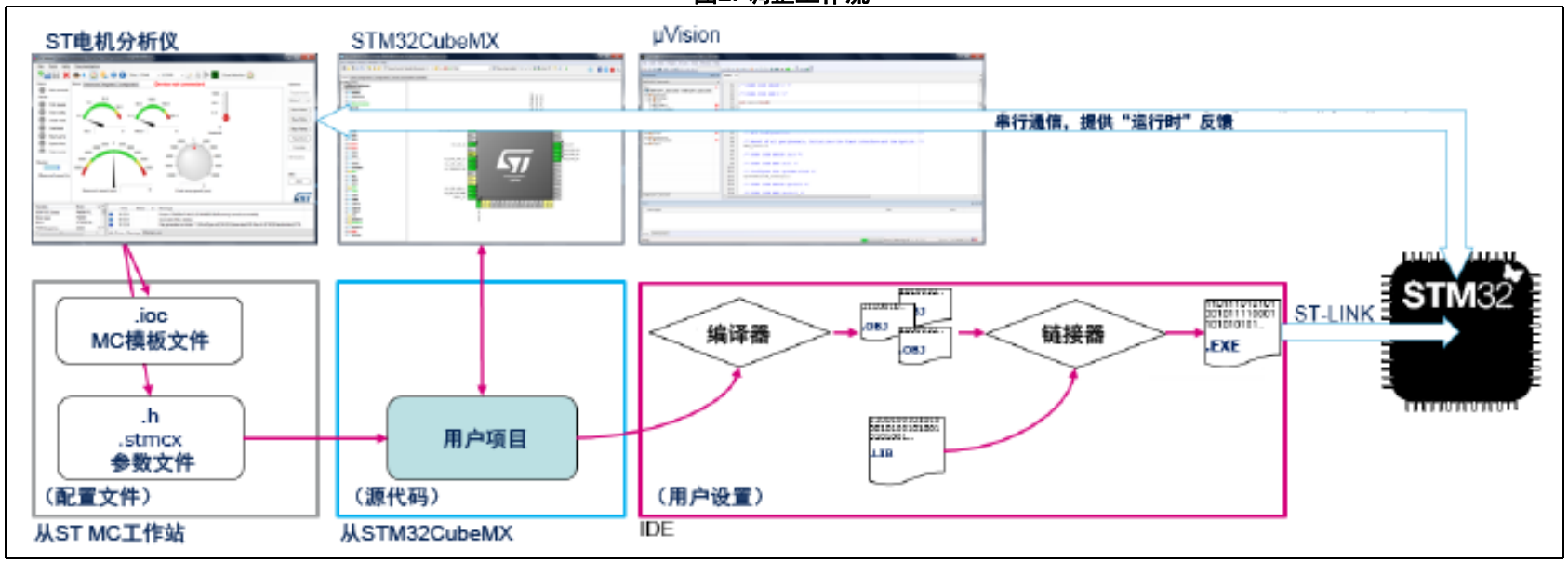


图2. 调整工作流



6 简化

6.1 将对象转换为指针

所有电机控制对象都转换为数据结构指针，以简化其使用、参考、可读性和潜在的调试。这也减少了待管理的头文件数量。*表 7*.描述了浪涌电流限制器经历的此类转换过程。

表7. 对象至指针转换示例

SDK v4.3	SDK v5.x
InrushCurrentLimiterClass.h	***InrushCurrentLimiter.h***

typedef enum
{
ICL_IDLE, ICL_ACTIVATION, ICL_ACTIVE,
ICL_DEACTIVATION, ICL_INACTIVE
} ICLState_t;
 	
typedef struct CICL_t *CICL;
 	
typedef struct {
{
uint16_t hICLFrequencyHz;
uint16_t hDurationms;
 	
} InrushCurrentLimiterParams_t,
*pInrushCurrentLimiterParams_t;
 	
CICL ICL_NewObject(pInrushCurrentLimiterParams_t
pInrushCurrentLimiterParams);
 	
void ICL_Init(CICL this, CVBS oVBS, CDOUT oDOUT);
	typedef enum
	{
	ICL_IDLE,
	ICL_ACTIVATION,
	ICL_ACTIVE,
	ICL_DEACTIVATION,
	ICL_INACTIVE
	} ICL_State_t;
	
	typedef struct
	{
	BusVoltageSensor_Handle_t *pVBS;
	DOUT_handle_t *pDOUT;
	
	ICL_State_t ICLstate;
	uint16_t hICLTicksCounter;
	uint16_t hICLTotalTicks;
	uint16_t hICLFrequencyHz;
	uint16_t hICLDurationms;
	} ICL_Handle_t;
	
	void ICL_Init(ICL_Handle_t *pHandle,
	BusVoltageSensor_Handle_t *pVBS, DOUT_handle_t
	*pDOUT);

表7. 对象至指针转换示例（续）

SDK v4.3	SDK v5.x
InrushCurrentLimiterPrivate.h	

typedef struct
{
  CVBS oVBS;
  CDOUT oDOUT;
 	
  ICLState_t ICLState;
  uint16_t hRemainingTicks;
  uint16_t hTotalTicks;
} Vars_t,*pVars_t;
 	
typedef InrushCurrentLimiterParams_t Params_t,
*pParams_t;
 	
typedef struct
{
  Vars_t Vars_str;
  pParams_t pParams_str;
} _CICL_t, *_CICL;
	-

6.2 电机控制API转换

本节表格给出了从SDK v4.3至SDK v5.x的API对应关系：

表8. 从*MCInterfaceClass.h* 转换为*mc_api.h* 文件

SDK v4.3	SDK v5.x
void MCI_ExecSpeedRamp(CMCI this, int16_t hFinalSpeed, uint16_t hDurationms);	void MC_ProgramSpeedRampMotor1(int16_t hFinalSpeed, uint16_t hDurationms); void MC_ProgramSpeedRampMotor2(int16_t hFinalSpeed, uint16_t hDurationms);
void MCI_ExecTorqueRamp(CMCI this, int16_t hFinalTorque, uint16_t hDurationms);	void MC_ProgramTorqueRampMotor1(int16_t hFinalTorque, uint16_t hDurationms); void MC_ProgramTorqueRampMotor2(int16_t hFinalTorque, uint16_t hDurationms);
void MCI_SetCurrentReferences(CMCI this, Curr_Components Iqdref);	void MC_SetCurrentReferenceMotor1 (Curr_Components Iqdref); void MC_SetCurrentReferenceMotor2 (Curr_Components Iqdref);
bool MCI_StartMotor(CMCI this);	bool MC_StartMotor1(void); bool MC_StartMotor2(void);

表8. 从*MCInterfaceClass.h* 转换为*mc_api.h* 文件（续）

SDK v4.3	SDK v5.x
bool MCI_StopMotor(CMCI this);	bool MC_StopMotor1(void); bool MC_StopMotor2(void);
bool MCI_FaultAcknowledged(CMCI this);	bool MC_AcknowledgeFaultMotor1(void); bool MC_AcknowledgeFaultMotor2(void);
bool MCI_EncoderAlign(CMCI this);	*删除了：*
CommandState_t MCI_IsCommandAcknowledged(CMCI this);	MCI_CommandState_t MC_GetCommandStateMotor1(void); MCI_CommandState_t MC_GetCommandStateMotor2(void);
State_t MCI_GetSTMState(CMCI this);	State_t MC_GetSTMStateMotor1(void); State_t MC_GetSTMStateMotor2(void);
int16_t MCI_GetOccurredFaults(CMCI this);	uint16_t MC_GetOccurredFaultsMotor1(void); uint16_t MC_GetOccurredFaultsMotor2(void);
uint16_t MCI_GetCurrentFaults(CMCI this);	uint16_t MC_GetCurrentFaultsMotor1(void); uint16_t MC_GetCurrentFaultsMotor2(void);
int16_t MCI_GetMecSpeedRef01Hz(CMCI this);	int16_t MC_GetMecSpeedReferenceMotor1 (void); int16_t MC_GetMecSpeedReferenceMotor2 (void);
int16_t MCI_GetAvrgMecSpeed01Hz(CMCI this);	int16_t MC_GetMecSpeedAverageMotor1 (void); int16_t MC_GetMecSpeedAverageMotor2 (void);
int16_t MCI_GetTorque(CMCI this);	Not needed as never Implemented in SDK v4.3
STC_Modality_t MCI_GetControlMode(CMCI this);	STC_Modality_t MC_GetControlModeMotor1(void); STC_Modality_t MC_GetControlModeMotor2(void);
int16_t MCI_GetImposedMotorDirection(CMCI this);	int16_t MC_GetImposedDirectionMotor1 (void); int16_t MC_GetImposedDirectionMotor2 (void);
int16_t MCI_GetLastRampFinalSpeed(CMCI this);	int16_t MC_GetLastRampFinalSpeedMotor1 (void); int16_t MC_GetLastRampFinalSpeedMotor2 (void);
bool MCI_RampCompleted(CMCI this);	bool MC_HasRampCompletedMotor1(void); bool MC_HasRampCompletedMotor2(void);
bool MCI_StopSpeedRamp(CMCI this);	bool MC_StopSpeedRampMotor1(void); bool MC_StopSpeedRampMotor2(void);
bool MCI_GetSpdSensorReliability(CMCI this);	bool MC_GetSpeedSensorReliabilityMotor1(void); bool MC_GetSpeedSensorReliabilityMotor2(void);
Curr_Components MCI_GetIab(CMCI this);	Curr_Components MC_GetIabMotor1(void); Curr_Components MC_GetIabMotor2(void);
Curr_Components MCI_GetIalphabeta(CMCI this);	Curr_Components MC_GetIalphabetaMotor1(void); Curr_Components MC_GetIalphabetaMotor2(void);
Curr_Components MCI_GetIqd(CMCI this);	Curr_Components MC_GetIqdMotor1(void); Curr_Components MC_GetIqdMotor2(void);
Curr_Components MCI_GetIqdHF(CMCI this);	Curr_Components MC_GetIqdHFMotor1(void); Curr_Components MC_GetIqdHFMotor2(void)

表8. 从*MCInterfaceClass.h* 转换为*mc_api.h* 文件（续）

SDK v4.3	SDK v5.x
Curr_Components MCI_GetIqdref(CMCI this);	Curr_Components MC_GetIqdrefMotor1(void); Curr_Components MC_GetIqdrefMotor2(void);
Volt_Components MCI_GetVqd(CMCI this);	Volt_Components MC_GetVqdMotor1(void); Volt_Components MC_GetVqdMotor2(void);
Volt_Components MCI_GetValphabeta(CMCI this);	Volt_Components MC_GetValphabetaMotor1(void); Volt_Components MC_GetValphabetaMotor2(void);
int16_t MCI_GetElAngledpp(CMCI this);	int16_t MC_GetElAngledppMotor1 (void); int16_t MC_GetElAngledppMotor2 (void);
int16_t MCI_GetTeref(CMCI this);	int16_t MC_GetTerefMotor1 (void); int16_t MC_GetTerefMotor2 (void);
int16_t MCI_GetPhaseCurrentAmplitude(CMCI this);	int16_t MC_GetPhaseCurrentAmplitudeMotor1 (void); int16_t MC_GetPhaseCurrentAmplitudeMotor2 (void);
int16_t MCI_GetPhaseVoltageAmplitude(CMCI this);	int16_t MC_GetPhaseVoltageAmplitudeMotor1 (void); int16_t MC_GetPhaseVoltageAmplitudeMotor2 (void);
void MCI_SetIdref(CMCI this, int16_t hNewIdref);	(void); MC_SetIdrefMotor1(int16_t hNewIdref); (void); MC_SetIdrefMotor2(int16_t hNewIdref);
void MCI_Clear_Iqdref(CMCI this);	void MC_Clear_IqdrefMotor1(void); void MC_Clear_IqdrefMotor2(void);

表9. 从*MC.h* 转换为 *mc_api.h* 文件

SDK v4.3	SDK v5.x
void MC_RequestRegularConv(uint8_t bChannel, uint8_t bSamplTime);	void MC_ProgramRegularConversion(uint8_t bChannel, uint8_t bSamplTime);
uint16_t MC_GetRegularConv(void);	uint16_t MC_GetRegularConversionValue(void);
UDRC_State_t MC_RegularConvState(void);	UDRC_State_t MC_GetRegularConversionState(void);

表10. 从*MCInterfaceClass.h* 转换为*flux_weakening_ctrl.h* 文件

SDK v4.3	SDK v5.x
[*弱磁*] ***FluxWeakeningCtrl*类导出方法**	
void FW_SetVref(CFW this, uint16_t hNewVref);	void FW_SetVref(FW_Handle_t *pHandle, uint16_t hNewVref);
uint16_t FW_GetVref(CFW this);	uint16_t FW_GetVref(FW_Handle_t *pHandle);
int16_t FW_GetAvVAmplitude(CFW this);	int16_t FW_GetAvVAmplitude(FW_Handle_t *pHandle);
uint16_t FW_GetAvVPercentage(CFW this);	uint16_t FW_GetAvVPercentage(FW_Handle_t *pHandle);

表11. 从*MCTuningClass.h* 转换为*feed_forward_ctrl.h*文件

SDK v4.3	SDK v5.x
***[前馈]* *FeedForwardCtrl*类导出方法**	
void FF_SetFFConstants(CFF this, FF_TuningStruct_t sNewConstants);	void FF_SetFFConstants(FF_Handle_t *pHandle, FF_TuningStruct_t sNewConstants);
FF_TuningStruct_t FF_GetFFConstants(CFF this);	FF_TuningStruct_t FF_GetFFConstants(FF_Handle_t *pHandle);
Volt_Components FF_GetVqdff(CFF this);	Volt_Components FF_GetVqdff(FF_Handle_t *pHandle);
Volt_Components FF_GetVqdAvPIout(CFF this);	Volt_Components FF_GetVqdAvPIout(FF_Handle_t *pHandle);

表12. 从*MCTuningClass.h*转换为*open_loop.h*文件

SDK v4.3	SDK v5.x
[开环] ***OLCtrl类导出方法***	
void OL_UpdateVoltage(COL this, int16_t hNewVoltage);	void OL_UpdateVoltage(OpenLoop_Handle_t *pHandle, int16_t hNewVoltage);

表13. 从*MCTuningClass.h*转换为*pid_regulator.h*文件

SDK v4.3	SDK v5.x
[比例积分] ***PI稳压器类导出方法***	
void PI_SetKP(CPI this, int16_t hKpGain);	void PID_SetKP(PID_Handle_t* pHandle, int16_t hKpGain);
void PI_SetKI(CPI this, int16_t hKiGain);	void PID_SetKI(PID_Handle_t* pHandle, int16_t hKiGain);
int16_t PI_GetKP(CPI this);	int16_t PID_GetKP(PID_Handle_t* pHandle);
uint16_t PI_GetKPDivisor(CPI this);	uint16_t PID_GetKPDivisor(PID_Handle_t* pHandle);
int16_t PI_GetKI(CPI this);	int16_t PID_GetKI(PID_Handle_t* pHandle);
uint16_t PI_GetKIDivisor(CPI this);	uint16_t PID_GetKIDivisor(PID_Handle_t* pHandle);
int16_t PI_GetDefaultKP(CPI this);	int16_t PID_GetDefaultKP(PID_Handle_t* pHandle);
int16_t PI_GetDefaultKI(CPI this);	int16_t PID_GetDefaultKI(PID_Handle_t* pHandle);
void PI_SetIntegralTerm(CPI this, int32_t wIntegralTermValue);	void PID_SetIntegralTerm(PID_Handle_t* pHandle, int32_t wIntegralTermValue);
[导出的比例积分] ***PID类导出方法***	
void PID_SetPrevError(CPID_PI this, int32_t wPrevProcessVarError);	void PID_SetPrevError(PID_Handle_t* pHandle, int32_t wPrevProcessVarError);
void PID_SetKD(CPID_PI this, int16_t hKdGain);	void PID_SetKD(PID_Handle_t* pHandle, int16_t hKdGain);
int16_t PID_GetKD(CPID_PI this);	int16_t PID_GetKD(PID_Handle_t* pHandle);

表14. 从*MCTuningClass.h*转换为*pwm_curr_fdbk.h*文件

SDK v4.3	SDK v5.x
[脉冲宽度调制控制] ***PWMnCurrFdbk类导出方法***	
uint16_t PWMC_ExecRegularConv(CPWMC this, uint8_t bChannel);	uint16_t PWMC_ExecRegularConv(PWMC_Handle_t *pHandle, uint8_t bChannel);
void PWMC_ADC_SetSamplingTime(CPWMC this, ADConv_t ADConv_struct);	void PWMC_ADC_SetSamplingTime(PWMC_Handle_t *pHandle, ADConv_t ADConv_struct);

表15. 从*MCTuningClass.h*转换为*revup_ctrl.h*文件

SDK v4.3	SDK v5.x
[加速控制] ***RevupCtrl类导出方法***	
void RUC_SetPhaseDurationms(CRUC this, uint8_t bPhase, uint16_t hDurationms);	void RUC_SetPhaseDurationms(RevUpCtrl_Handle_t *pHandle, uint8_t bPhase, uint16_t hDurationms);
void RUC_SetPhaseFinalMecSpeed01Hz(CRUC this, uint8_t bPhase, int16_t hFinalMecSpeed01Hz);	void RUC_SetPhaseFinalMecSpeed01Hz(RevUpCtrl_Handle_t *pHandle, uint8_t bPhase,
void RUC_SetPhaseFinalTorque(CRUC this, uint8_t bPhase, int16_t hFinalTorque);	void RUC_SetPhaseFinalTorque(RevUpCtrl_Handle_t *pHandle, uint8_t bPhase, int16_t hFinalTorque);
uint16_t RUC_GetPhaseDurationms(CRUC this, uint8_t bPhase);	uint16_t RUC_GetPhaseDurationms(RevUpCtrl_Handle_t *pHandle, uint8_t bPhase);
int16_t RUC_GetPhaseFinalMecSpeed01Hz(CRUC this, uint8_t bPhase);	int16_t RUC_GetPhaseFinalMecSpeed01Hz(RevUpCtrl_Handle_t *pHandle, uint8_t bPhase);
int16_t RUC_GetPhaseFinalTorque(CRUC this, uint8_t bPhase);	int16_t RUC_GetPhaseFinalTorque(RevUpCtrl_Handle_t *pHandle, uint8_t bPhase);
uint8_t RUC_GetNumberOfPhases(CRUC this);	uint8_t RUC_GetNumberOfPhases(RevUpCtrl_Handle_t *pHandle);

表16. 从*MCTuningClass.h*转换为*ntc_temperature_sensor.h*文件

SDK v4.3	SDK v5.x
[温度传感器] ***温度传感器类导出方法***	
int16_t TSNS_GetAvTemp_C(CTSNS this);	int16_t NTC_GetAvTemp_C(NTC_Handle_t *pHandle);
uint16_t TSNS_CheckTemp(CTSNS this);	uint16_t NTC_CheckTemp(NTC_Handle_t *pHandle);

表17. 从*MCTuningClass.h*转换为*digital_output.h*文件

SDK v4.3	SDK v5.x
[数字输出] ***DigitalOutput类导出方法***	
DOutputState_t DOUT_GetOutputState(CDOUT this);	DOutputState_tDOUT_GetOutputState(DOUT_handle_t *pHandle);

表18. 从*MCTuningClass.h*转换为*motor_power_measurement.h*文件

SDK v4.3	SDK v5.x
[***电机电源测量***] ***MotorPowerMeasurement类导出方法***	
int16_t MPM_GetElMotorPowerW(CMPM this);	int16_t MPM_GetElMotorPowerW(MotorPowMeas_Handle_t *pHandle);
int16_t MPM_GetAvrgElMotorPowerW(CMPM this);	int16_t MPM_GetAvrgElMotorPowerW(MotorPowMeas_Handle_t *pHandle);

表19. 从*MCTuningClass.h*转换为*speed_pos_fdbk.h*文件

SDK v4.3	SDK v5.x
[速度和位置] ***SpeednPosFdbk类导出方法***	
int16_t SPD_GetElAngle(CSPD this);	int16_t SPD_GetElAngle(SpeednPosFdbk_Handle_t *pHandle);
int16_t SPD_GetMecAngle(CSPD this);	int16_t SPD_GetMecAngle(SpeednPosFdbk_Handle_t *pHandle);
int16_t SPD_GetAvrgMecSpeed01Hz(CSPD this);	int16_t SPD_GetAvrgMecSpeed01Hz(SpeednPosFdbk_Handle_t *pHandle);
int16_t SPD_GetElSpeedDpp(CSPD this);	int16_t SPD_GetElSpeedDpp(SpeednPosFdbk_Handle_t *pHandle);
bool SPD_Check(CSPD this);	bool SPD_Check(SpeednPosFdbk_Handle_t *pHandle);

表19. 从*MCTuningClass.h*转换为*speed_pos_fdbk.h*文件（续）

SDK v4.3	SDK v5.x
int16_t SPD_GetS16Speed(CSPD this);	bool SPD_IsMecSpeedReliable(SpeednPosFdbk_Handle_t *pHandle, int16_t *pMecSpeed01Hz);
-	int16_t SPD_GetS16Speed(SpeednPosFdbk_Handle_t *pHandle);
uint8_t SPD_GetElToMecRatio(CSPD this);	uint8_t SPD_GetElToMecRatio(SpeednPosFdbk_Handle_t *pHandle);
void SPD_SetElToMecRatio(CSPD this, uint8_t bPP);	void SPD_SetElToMecRatio(SpeednPosFdbk_Handle_t *pHandle, uint8_t bPP);

表20. 从*MCTuningClass.h*转换为*virtual_speed_sensor.h*文件

SDK v4.3	SDK v5.x
[虚拟传感器速度] ***VSS类导出方法***	
void VSPD_SetMecAcceleration(CSPD this, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms);	void VSS_SetMecAcceleration(VirtualSpeedSensor_Handle_t *pHandle, int16_t hFinalMecSpeed01Hz, uint16_t hDurationms);
int16_t VSPD_GetLastRampFinalSpeed(CSPD this);	int16_t VSS_GetLastRampFinalSpeed(VirtualSpeedSensor_Handle_t *pHandle);

表21. 从*MCTuningClass.h*转换为*sto_speed_pos_fdbk.h*文件

SDK v4.3	SDK v5.x
[状态观察器] ***STO类导出方法***	
Volt_Components STO_GetEstimatedBemf(CSTO_SPD this);	Volt_Components STO_GetEstimatedBemf(STO_Handle_t *pHandle);
Curr_Components STO_GetEstimatedCurrent(CSTO_SPD this);	Curr_Components STO_GetEstimatedCurrent(STO_Handle_t *pHandle);
void STO_GetObserverGains(CSTO_SPD this, int16_t *pC2, int16_t *pC4);	void STO_GetObserverGains(STO_Handle_t *pHandle, int16_t *pC2, int16_t *pC4);
void STO_SetObserverGains(CSTO_SPD this, int16_t hC1, int16_t hC2);	void STO_SetObserverGains(STO_Handle_t *pHandle, int16_t hC1, int16_t hC2);
void STO_GetPLLGains(CSTO_SPD this, int16_t *pPgain, int16_t *pIgain);	void STO_GetPLLGains(STO_Handle_t *pHandle, int16_t *pPgain, int16_t *pIgain);
void STO_SetPLLGains(CSTO_SPD this, int16_t hPgain, int16_t hIgain);	void STO_SetPLLGains(STO_Handle_t *pHandle, int16_t hPgain, int16_t hIgain);
void STO_ResetPLL(CSTO_SPD this);	void STO_ResetPLL(STO_Handle_t *pHandle);

表21. 从*MCTuningClass.h*转换为*sto_speed_pos_fdbk.h*文件（续）

SDK v4.3	SDK v5.x
int32_t STO_GetEstimatedBemfLevel(CSTO_SPD this);	int32_t STO_GetEstimatedBemfLevel(STO_Handle_t *pHandle);
int32_t STO_GetObservedBemfLevel(CSTO_SPD this);	int32_t STO_GetObservedBemfLevel(STO_Handle_t *pHandle);
void STO_BemfConsistencyCheckSwitch(CSTO_SPD this, bool bSel);	void STO_BemfConsistencyCheckSwitch(STO_Handle_t *pHandle, bool bSel);
bool STO_IsBemfConsistent(CSTO_SPD this);	bool STO_IsBemfConsistent(STO_Handle_t *pHandle);

表22. 从*MCTuningClass.h*转换为*sto_cordic_speed_pos_fdbk.h* 文件

SDK v4.3	SDK v5.x
[*使用CORDIC算法的状态观察器*] ***STO_CORDIC*类导出方法**	
Volt_Components STO_CR_GetEstimatedBemf(CSTO_CR_SPD this);	Volt_Components STO_CR_GetEstimatedBemf(STO_CR_Handle_t* pHandle);
Curr_Components STO_CR_GetEstimatedCurrent(CSTO_CR_SPD this);	Curr_Components STO_CR_GetEstimatedCurrent(STO_CR_Handle_t* pHandle);
void STO_CR_GetObserverGains(CSTO_CR_SPD this, int16_t *pC2, int16_t *pC4);	void STO_CR_GetObserverGains(STO_CR_Handle_t* pHandle, int16_t *pC2, int16_t *pC4);
void STO_CR_SetObserverGains(CSTO_CR_SPD this, int16_t hC1, int16_t hC2);	void STO_CR_SetObserverGains(STO_CR_Handle_t* pHandle, int16_t hC1, int16_t hC2);
int32_t STO_CR_GetEstimatedBemfLevel(CSTO_CR_SPD this);	int32_t STO_CR_GetEstimatedBemfLevel(STO_CR_Handle_t* pHandle);
int32_t STO_CR_GetObservedBemfLevel(CSTO_CR_SPD this);	int32_t STO_CR_GetObservedBemfLevel(STO_CR_Handle_t* pHandle);
void STO_CR_BemfConsistencyCheckSwitch(CSTO_CR_SPD this, bool bSel);	void STO_CR_BemfConsistencyCheckSwitch(STO_CR_Handle_t* pHandle, bool bSel);
bool STO_CR_IsBemfConsistent(CSTO_CR_SPD this);	bool STO_CR_IsBemfConsistent(STO_CR_Handle_t* pHandle);

表23. 从*MCTuningClass.h*转换为*speed_torq_ctrl.h*文件

SDK v4.3	SDK v5.x
[速度和扭矩控制] ***SpeednTorqCtrl类导出方法***	
int16_t STC_GetMecSpeedRef01Hz(CSTC this);	int16_t STC_GetMecSpeedRef01Hz(SpeednTorqCtrl_Handle_t *pHandle);
int16_t STC_GetTorqueRef(CSTC this);	int16_t STC_GetTorqueRef(SpeednTorqCtrl_Handle_t *pHandle);
STC_Modality_t STC_GetControlMode(CSTC this);	STC_Modality_t STC_GetControlMode(SpeednTorqCtrl_Handle_t *pHandle);
uint16_t STC_GetMaxAppPositiveMecSpeed01Hz(CSTC this);	uint16_t STC_GetMaxAppPositiveMecSpeed01Hz(SpeednTorqCtrl_Handle_t *pHandle);
int16_t STC_GetMinAppNegativeMecSpeed01Hz(CSTC this);	int16_t STC_GetMinAppNegativeMecSpeed01Hz(SpeednTorqCtrl_Handle_t *pHandle);
Curr_Components STC_GetDefaultIqdref(CSTC this);	Curr_Components STC_GetDefaultIqdref(SpeednTorqCtrl_Handle_t *pHandle);
void STC_SetNominalCurrent(CSTC this, uint16_t hNominalCurrent);	void STC_SetNominalCurrent(SpeednTorqCtrl_Handle_t *pHandle, uint16_t hNominalCurrent);

表24. 从*MCTuningClass.h*转换为*state_machine.h*文件

SDK v4.3	SDK v5.x
[状态机] ***StateMachine类导出方法***	
State_t STM_GetState(CSTM this);	State_t STM_GetState(STM_Handle_t *pHandle);
uint32_t STM_GetFaultState(CSTM this);	uint32_t STM_GetFaultState(STM_Handle_t *pHandle);

表25. 从*MCTuningClass.h*转换为*bus_voltage_sensor.h*文件

SDK v4.3	SDK v5.x
[总线电压传感器] ***BusVoltageSensor类导出方法***	
uint16_t VBS_GetAvBusVoltage_V(CVBS this);	uint16_t VBS_GetAvBusVoltage_V(BusVoltageSensor_Handle_t *pHandle);
uint16_t VBS_CheckVbus(CVBS this);	uint16_t VBS_CheckVbus(BusVoltageSensor_Handle_t *pHandle);

表 26 列出在SDK v5.0中未向用户公开的*MCTuningClass.h*文件的API。这些API仅涉及电机分析仪工具，为支持其使用，还提供了特定的二进制固件。

表26. SDK v5.x 中未公开的*MCTuningClass.h* API

SDK v4.3	SDK v5.x
[自调试] ***Selfcommissioning类导出方法***	
uint8_t SCC_GetState(CSCC this);	不适用。
uint8_t SCC_GetSteps(CSCC this);	
uint32_t SCC_GetRs(CSCC this);	
uint32_t SCC_GetLs(CSCC this);	
uint32_t SCC_GetKe(CSCC this);	
uint32_t SCC_GetVbus(CSCC this);	
uint32_t SCC_GetEstNominalSpeed(CSCC this);	
void SCC_ForceProfile(CSCC this);	
void SCC_StopProfile(CSCC this);	
void SCC_SetPolesPairs(CSCC this, uint8_t bPP);	
void SCC_SetNominalCurrent(CSCC this, float fCurrent);	
float SCC_GetNominalCurrent(CSCC this);	
void SCC_SetLdLqRatio(CSCC this, float fLdLqRatio);	
float SCC_GetLdLqRatio(CSCC this);	
void SCC_SetNominalSpeed(CSCC this, int32_t wNominalSpeed);	
int32_t SCC_GetNominalSpeed(CSCC this);	
int32_t SCC_GetEstMaxOLSpeed(CSCC this);	
int32_t SCC_GetEstMaxAcceleration(CSCC this);	
int16_t SCC_GetStartupCurrentS16(CSCC this);	
float SCC_GetStartupCurrentAmp(CSCC this);	
void SCC_SetCurrentBandwidth(CSCC this, float fCurrentBW);	
float SCC_GetCurrentBandwidth(CSCC this);	
uint16_t SCC_GetPWMFrequencyHz(CSCC this);	
uint8_t SCC_GetFOCRepRate(CSCC this);	

表26. SDK v5.x（续）中未公开的*MCTuningClass.h* API

SDK v4.3	SDK v5.x
[单触式校正] ***单触式校正类导出方法***	
void OTT_ForceTuning(COTT this);	不适用。
uint32_t OTT_GetNominalSpeed(COTT this);	
uint8_t OTT_GetSteps(COTT this);	
uint8_t OTT_GetState(COTT this);	
bool OTT_IsSpeedPITuned(COTT this);	
float OTT_fGetNominalSpeedRPM(COTT this);	
void OTT_SetPolesPairs(COTT this, uint8_t bPP);	
void OTT_SetNominalCurrent(COTT this, uint16_t hNominalCurrent);	
void OTT_SetSpeedRegulatorBandwidth(COTT this, float fBW);	
float OTT_GetSpeedRegulatorBandwidth(COTT this);	
float OTT_GetJ(COTT this);	
float OTT_GetF(COTT this);	
bool OTT_IsMotorAlreadyProfiled(COTT this);	

表 27 列出在SDK v5.0中未向用户公开的*MCTuningClass.h*文件的API。这些API在 SDK v5.0 中未提及。在 SDK v4.3 中，它们用于提供对象可视性（亦称为类导出）。

表27. SDK v5.x 中不存在的*MCTuningClass.h* API

SDK v4.3	SDK v5.x
[电机控制调整] ***MCTuning类导出方法***	
CFW MCT_GetFluxWeakeningCtrl(CMCT this);	不适用。
CFF MCT_GetFeedForwardCtrl(CMCT this);	
CHFI_FP MCT_GetHFICtrl(CMCT this);	
CPI MCT_GetSpeedLoopPID(CMCT this);	
CPI MCT_GetIqLoopPID(CMCT this);	
CPI MCT_GetIdLoopPID(CMCT this);	
CPI MCT_GetFluxWeakeningLoopPID(CMCT this);	
CPWMC MCT_GetPWMnCurrFdbk(CMCT this);	
CRUC MCT_GetRevupCtrl(CMCT this);	
CSPD MCT_GetSpeednPosSensorMain(CMCT this);	
CSPD MCT_GetSpeednPosSensorAuxiliary(CMCT this);	
CSPD MCT_GetSpeednPosSensorVirtual(CMCT this);	
CSTC MCT_GetSpeednTorqueController(CMCT this);	
CSTM MCT_GetStateMachine(CMCT this);	
CTSNS MCT_GetTemperatureSensor(CMCT this);	
CVBS MCT_GetBusVoltageSensor(CMCT this);	
CDOUT MCT_GetBrakeResistor(CMCT this);	
CDOUT MCT_GetNTCRelay(CMCT this);	
CMPM MCT_GetMotorPowerMeasurement(CMCT this);	
CSCC MCT_GetSelfCommissioning(CMCT this);	
COTT MCT_GetOneTouchTuning(CMCT this);	

表28. 从*MCTuningClass.h*转换为*hifreqinj_fpu_ctrl.h*文件

SDK v4.3	SDK v5.x(1)
[高频注入] ***HFICtrl类导出方法***	
int16_t HFI_FP_GetRotorAngleLock(CHFI_FP this);	int16_t HFI_FP_GetRotorAngleLock(HFI_FP_Ctrl_Handle_t *pHandle);
int16_t HFI_FP_GetSaturationDifference(CHFI_FP this);	int16_t HFI_FP_GetSaturationDifference(HFI_FP_Ctrl_Handle_t *pHandle);
int16_t HFI_FP_GetCurrent(CHFI_FP this);	int16_t HFI_FP_GetCurrent(HFI_FP_Ctrl_Handle_t *pHandle);
CPI HFI_FP_GetPITrack(CHFI_FP this);	PID_Handle_t* HFI_FP_GetPITrack(HFI_FP_Ctrl_Handle_t *pHandle);
void HFI_FP_SetMinSaturationDifference(CHFI_FP this, int16_t hMinSaturationDifference);	void HFI_FP_SetMinSaturationDifference(HFI_FP_Ctrl_Handle_t *pHandle, int16_t MinSaturationDifference);

1. 此特性在 SDK v5.0 中未实施，但在最新版本中提供。

表29. 从*MCTasks.h*转换为*mc_tasks.h*文件

SDK v4.3	SDK v5.x
void MCboot(CMCI oMCIList[],CMCT oMCTList[]);	void MCboot(MCI_Handle_t* pMCIList[], MCT_Handle_t* pMCTList[]);
void MC_Scheduler(void);	void MC_Scheduler(void);
void TSK_SafetyTask(void)	void TSK_SafetyTask(void)
uint8_t TSK_HighFrequencyTask(void)	uint8_t TSK_HighFrequencyTask(void)
void TSK_DualDriveFIFOUpdate(void *oDrive);	void TSK_DualDriveFIFOUpdate(void *oDrive);
void TSK_HardwareFaultTask(void)	void TSK_HardwareFaultTask(void)

表30. 从*MC.h* 转换为*mc_extended_api.h*文件

SDK v4.3	SDK v5.x
CMCI GetMCI(uint8_t bMotor);	MCI_Handle_t * GetMCI(uint8_t bMotor);
CMCT GetMCT(uint8_t bMotor);	MCT_Handle_t* GetMCT(uint8_t bMotor);
void MC_SetDAC(DAC_Channel_t bChannel, MC_Protocol_REG_t bVariable);	void MC_SetDAC(DAC_Channel_t bChannel, MC_Protocol_REG_t bVariable);
void MC_SetUserDAC(DAC_UserChannel_t bUserChNumber, int16_t hValue);	void MC_SetUserDAC(DAC_UserChannel_t bUserChNumber, int16_t hValue);

7 Cube方式转化（Cubification）

Cube方式转化是将电机控制的应用程序从面向对象的C源代码迁移到标准C源代码，以便符合STM32CubeMX的使用。

7.1 库的使用从SPL转换为HAL或LL

STMicroelectronics的SPL2LL-转换器工具将SPL转换为LL。可从*www.st.com*下载，以便在此转换阶段为用户提供支持。上述网站还提供演示文稿和应用笔记，以帮助用户使用该工具。

7.2 使用 STM32CubeMX 工具

STM32CubeMX 是STMicroelectronics提供的STM32Cube初始化代码生成器。它旨在支持用户自行开发应用程序。

STM32CubeMX 可从 *www.st.com*下载。*用于STM32配置和初始化C代码生成的STM32CubeMX*用户手册（UM1718）也可用于指导用户使用该工具。

8 版本历史

表31. 文档版本历史

日期	版本	变更
2018 年 3 月 5 日	1	初始版本。
2018年4月23日	2	更新了*表2：SDK v5.0与SDK v4.3比较总结*。
2018年7月10日	3	将文档范围扩展至SDK v5.x： – 更新了文档标题 – 更新了*表 2*的*支持的微控制器、特性、MIPS和存储器容量* – 更新了*表 3* – 增加了*表 4*，*表 5*，以及*表 6*。 – 更新了 *表 8* 的双电机API

表32. 中文文档版本历史

日期	版本	变更
2018 年 10月 15日	1	中文初始版本。

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。本文档的中文版本为英文版本的翻译件，仅供参考之用；若中文版本与英文版本有任何冲突或不一致，则以英文版本为准。