AN-8206
AMC 库：滑动模式

总结

FCM8531 是基于应用的并行核心处理器，用于由先进电机控制器 (AMC) 处理器和兼容 MCS®51 的 MCU 处理器组成的电机控制。AMC 是专门针对电机控制设计的核心处理器，其中集成了一个可配置的处理核心和外围电路，实现无传感器磁场定向控制 (FOC) 电机控制。可通过嵌入式 MCS®51 对系统控制、用户界面、通信接口和输入/输出接口编写程序，实现不同的电机应用。提供滑模观测器来根据输入指令和反馈电流估算电机的反电动势，然后根据 tan-1 函数计算转子角度，从而实现较高的鲁棒性。本文介绍 AMC 处理核心的滑模并对相关的参数和工作原理进行说明。

图 1 给出了 FCM8531 的开发环境配置。应用板可采用 Farichild FCM8531 评估板或用户自定义电路板（称为“目标板”）。FCM8531 评估板可与 Fairchild 提供的电机控制开发系统 (MCDS) 集成式开发环境 (IDE) 和 MCDS 编程套件一起使用，有助于用户开发电机应用产品。

MCDS IDE 可在 Microsoft® Windows 操作系统上运行，包含项目管理、AMC 库选择、寄存器设置和编译器/链接器/调试器链接等功能，有助于用户开发软件。通过 Farichild 提供的 AMC 库，用户可以快速开发无传感器电机驱动，从而缩短电机应用的开发时间。本文档是针对 AMC 库“滑模”的用户指南。有关 MCDS IDE 和 MCDS 编程套件的详情，请参见 Fairchild 网站：

http://www.fairchildsemi.com/applications/motor-control/solutions/bldc-pmsm-controller/

飞兆提供 MCDS IDE 和 MCDS 编程套件，以便用户连接计算机和目标板进行软件的开发。系统内部编程 (ISP) 功能用于将软件载入 FCM8531。最后，提供支持片上调试的片上调试系统 (OCDS) 接口，以缩短软件开发时间。

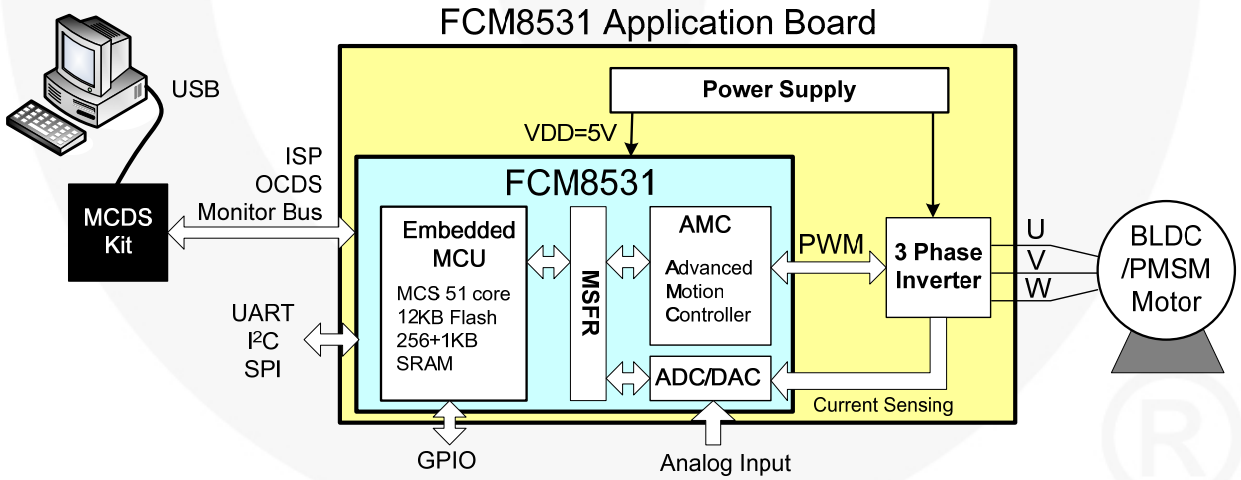


图 1. 典型开发环境

AMC 简介

FCM8531 是一款针对特定应用的控制器，由先进电机控制器 (AMC) 处理器和 MCS®51 兼容型 MCU 处理器组成。AMC 是专为电机控制设计的核心处理器。它用于电机驱动，由可配置处理核、PWM 引擎和角度预测器等数个电机控制模块组成。根据不同应用，处理内核可配置合适的 AMC 库执行磁场定向控制 (FOC) 或无传感器控制等电机控制算法。

本文档介绍滑模库的工作原理和使用方法，如图 2 所示。AMC 处理器利用 ADC 获取电机电流，然后通过坐标变换和滑模算法估算转子角度。SVM 表和 PWM 引擎通过输出对应于角度的 PWM 驱动信号来驱动电机。

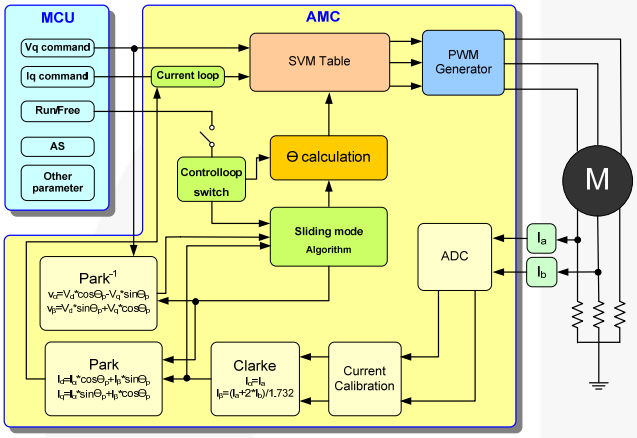


图 2. 滑模框图

图 8 显示带有滑模观测器架构的 AMC，包含电流环路、磁场定向控制 (FOC)、滑模观测器和角度调节功能。所显示的变量通过邮箱寄存器写入 AMC 中。

磁场定向控制 (FOC) 理论

FOC 理论首先由 F. Blaschke 于 1972 年提出。三相交流电机的非线性时变数学模型通过坐标变换可变换为直流电机的线性数学模型，然后直流电机的磁通量和电枢电流受到单独控制，从而简化电机控制算法。

FOC 环路中有三个变换模块：

1. Clarke 变换：三相 a-b-c 参考系变换为两相正交 α-β 参考系。
2. Park 变换：α-β 参考系变换为同步旋转 d-q 参考系；
3. Park^{-1} 变换：同步旋转 d-q 参考系变换为 α-β 参考系。

电机的三相电流从三相 a-b-c 参考系变换为两相 d-q 参考系。其中的 d 轴和 q 轴分量为常用的直接轴和正交轴分量，分别代表扭矩分量和磁场分量。投影概念可用来解释三维坐标系和二维坐标系之间的变换。如图 3 所示，三相参考系变换为两相参考系时，三相参考系的合成向量 f_S 会投影到两相参考系的坐标轴上。反之，两相参考系的合成向量会投影到三相参考系的坐标轴上。

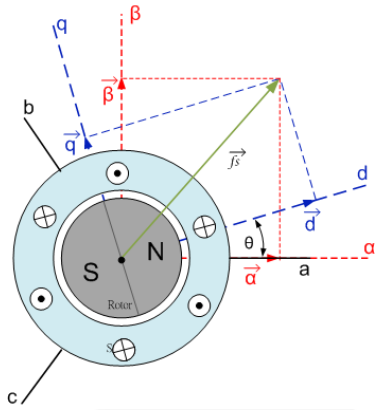


图 3. a-b-c 和 d-q 参考系

下面介绍三组变换的变换方程。

Clarke 变换

利用 Clarke 变换可将三相电流向量映射到两相正交 α-β 平面（如图 4 所示），从而将复杂的三相坐标简化为两相坐标。Clarke 变换的方程如下。

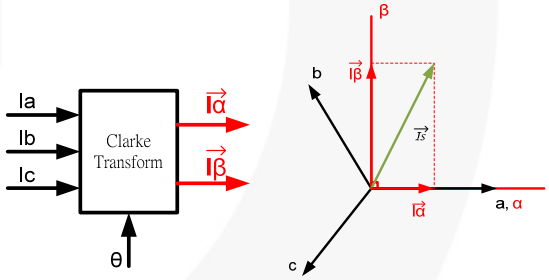


图 4. Clarke 变换

$$i_a + i_b + i_c = 0$$

$$i_\alpha = i_a$$

$$i_\beta = \frac{i_a + 2i_b}{\sqrt{3}}$$

Park 变换

完成 Clarke 变换后，利用 Park 变换将 α-β 参考系变换为同步旋转 d-q 参考系，这是 FOC 最关键的步骤。简单来说，利用 Park 变换可将正交坐标变换为特定角度（即：转子角度）坐标。从物理角度看，d 轴方向是转子磁场分量的方向，而 q 轴方向则是扭矩分量的方向，如图 5 所示。Park 变换的方程如下。

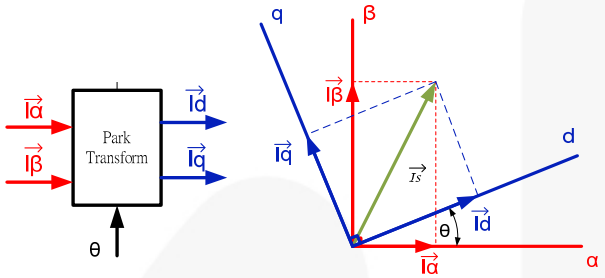


图 5. Park 变换

$$i_d = i_\alpha \cos\theta + i_\beta \sin\theta$$

$$i_q = -i_\alpha \sin\theta + i_\beta \cos\theta$$

Park^{-1} 变换

Park^{-1} 变换也称 Park 逆变换，是 Park 变换的逆向变换。利用 Park^{-1} 变换可将同步旋转 d-q 参考系变换为 α-β 参考系，如图 6 所示。Park^{-1} 变换的方程如下。

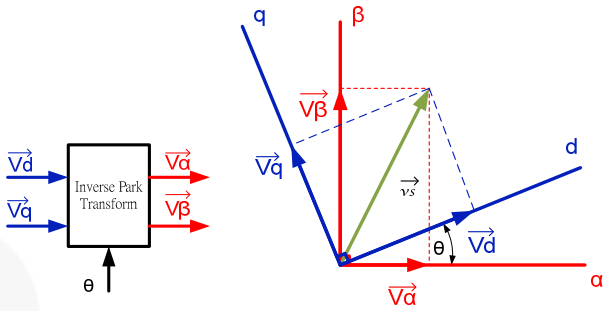


图 6. Park^{-1} 变换

$$v_\alpha = V_d \cos\theta - V_q \sin\theta$$

$$v_\beta = V_d \sin\theta + V_q \cos\theta$$

滑模理论

在滑模观测器 (SMO) 中，通过电机的数学模型获得电流测算值。实际电流与测算电流之间存在一定的电流误差，可用于通过“Bang-Bang”滞环控制器法估算反电动势。最后，通过将估算的反电动势代入 tan^{-1} 函数即可求得转子角度。

滑模观测器的框图如图 7 所示，角度估算步骤分为以下四个部分：

- 电流观测器
- Bang-Bang 控制器
- 数字低通滤波器
- tan^{-1} 函数

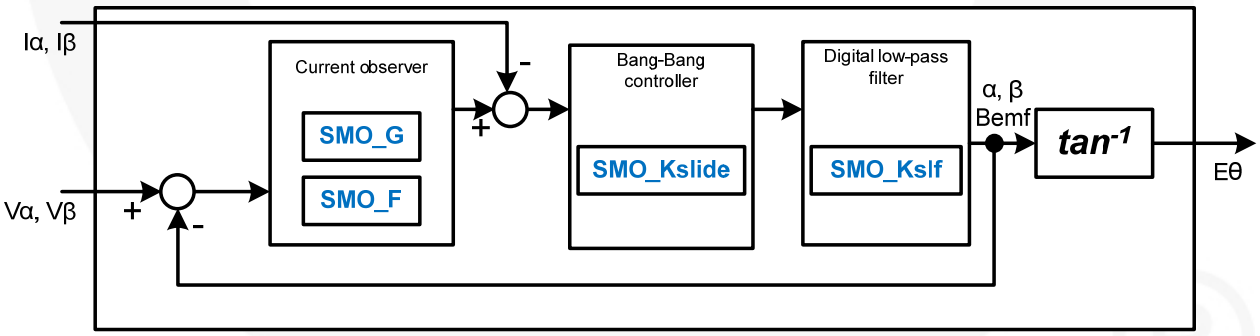


图 7. 滑模观测器框图

电机动态模型

如图 7 所示，可以根据输入电压指令 $\hat{\imath}_\alpha,\hat{\imath}_\beta$ 和 v_α v_β 以及估算的反电动势来估算电流，$e_\alpha$ande$_\beta$。

令 $\hat{\imath}_a = [\hat{\imath}_\alpha \hat{\imath}_\beta]^T$、 $v_a = [v_a v_\beta]^T$、 $\hat{e}_a = [e_a e_\beta]^T$

估算电流 $(\hat{\imath}_a(n+1))$ 的离散方程式为：

$$\hat{\imath}_a(n+1) = \mathrm{F}\hat{\imath}_a(n) + \mathrm{G}\left(v_a^*(n) - \hat{e}_a(n)\right)$$

其中，

$\mathrm{F} = e^{-\frac{R_S}{L_S}t_S}$, $\mathrm{G} = \frac{1}{R_S}\left(1 - e^{-\frac{R_S}{L_S}t_S}\right)$,

t_S：电流采样时间。

参数 F 和 G 可参见参数说明部分的 SMO_F 和 SMO_G。

Bang-Bang 控制器

SMO 的 Bang-Bang 控制器用于根据电流误差使估算的反电动势快速逼近实际反电动势。

Bang-Bang 控制器输出的 $(z(n+1))$ 的离散方程式如下：

$$z(n+1) = Kslide \cdot sign(\hat{\imath}_a(n) - i_a(n))$$

其中，符号函数定义为：

$$sign = \begin{cases} 1 & (\hat{\imath}_a - i_a) > 0 \\ 0 & (\hat{\imath}_a - i_a) = 0 \\ -1 & (\hat{\imath}_a - i_a) < 0 \end{cases}$$

有关参数 Kslide 的详细信息，可参见参数说明部分的 SMO_Kslide。

数字低通滤波器

在根据 Bang-Bang 控制器估算反电动势后，估算的反电动势由数字低通滤波器进行滤波。

估算反电动势输出 $(\hat{e}_a(n+1))$ 的离散方程式为：

$$\hat{e}_a(n+1) = \hat{e}_a(n) + 2\pi f_0(z(n) - \hat{e}_a(n))$$

其中，f_0 是低通滤波器的截止频率。

tan^{-1} 函数

最后，通过已滤波的反电动势 $\hat{e}_a\hat{e}_\beta$ 估算角度，计算式如下：

$$\begin{cases} e_\alpha = -\lambda_f \omega_e \sin\theta_e \\ e_\beta = \lambda_f \omega_e \cos\theta_e \end{cases}$$

因此，估算角度可表达为：

$$\hat{\theta}_e = \tan^{-1}\left(\frac{-\hat{e}_\alpha}{\hat{e}_\beta}\right)$$

有关估算角度的输出和用法，请参见参数说明章节。

控制参数

根据不同的电机和需求，可根据参数调整滑模库，这些参数为图 8 中的蓝色部分并在表 1 中进行了介绍。控制参数通过 8 个特殊功能寄存器 (SFRs) 进行通信：4 个为 MTX0(B0h) – MTX3(B3h)，另外 4 个为 MRX0(B4h) – MRX3(B7h)。前 4 个寄存器用于将信息从 MCU 传输至 AMC，后 4 个寄存器用于接收从 AMC 返回至 MCU 的信息。有关通信方式和应用的详细信息，请参考通信章节。

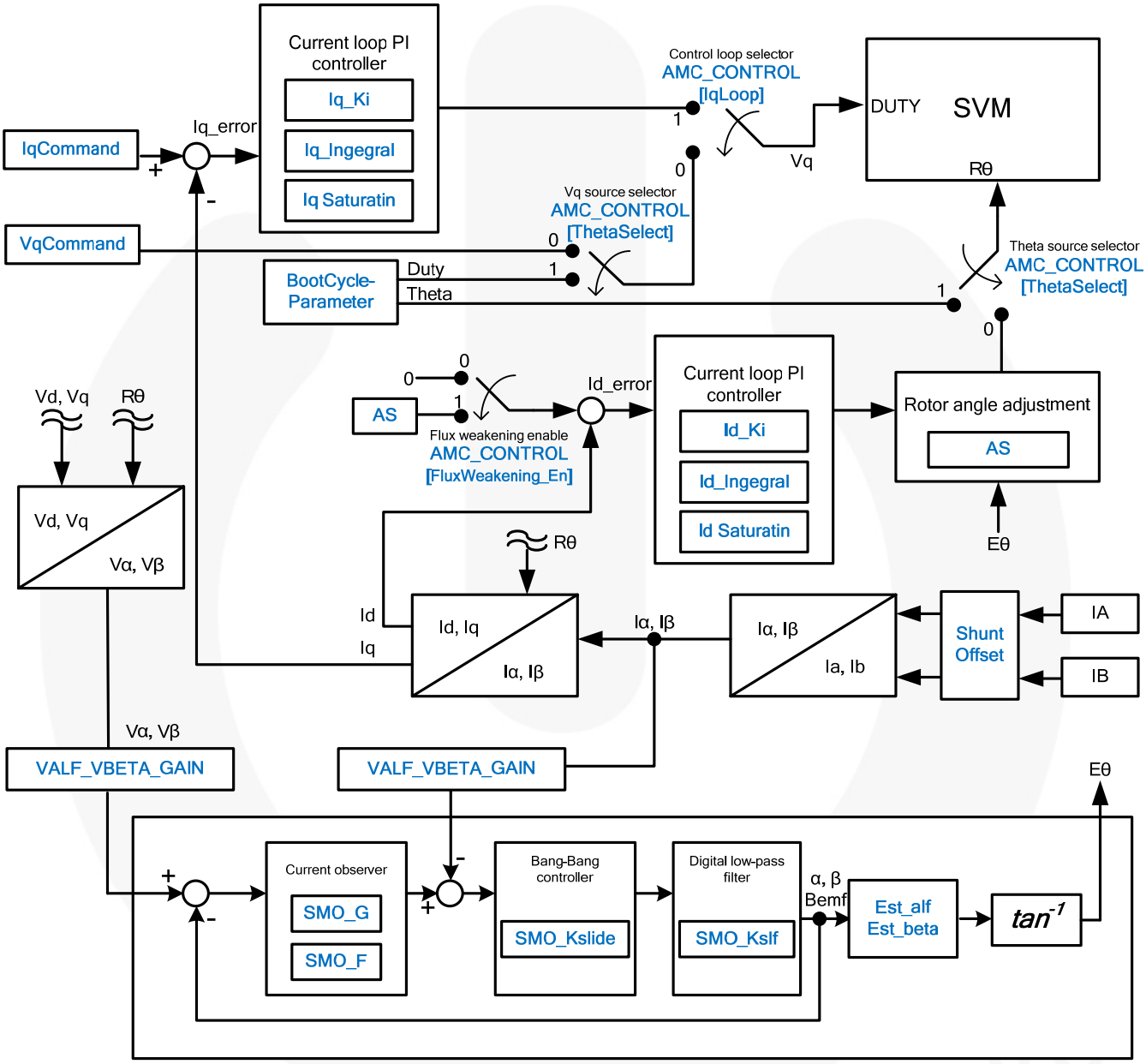
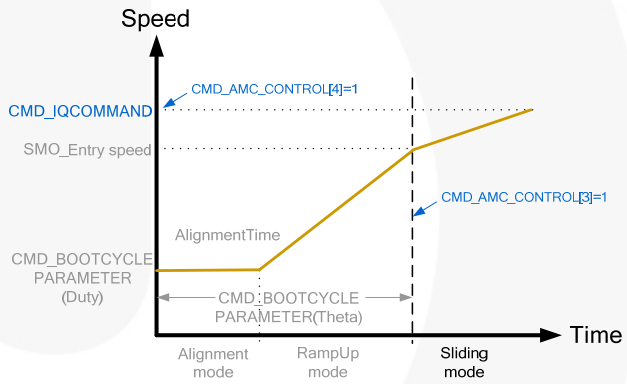
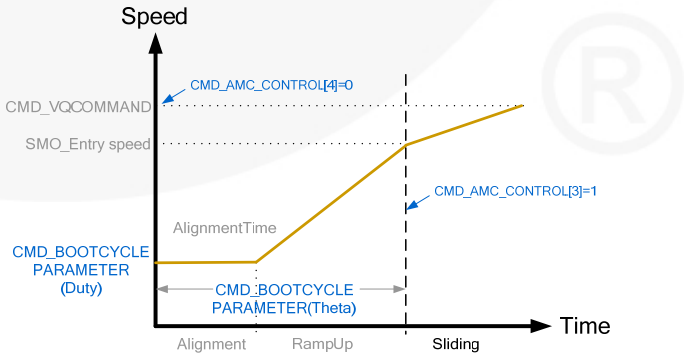
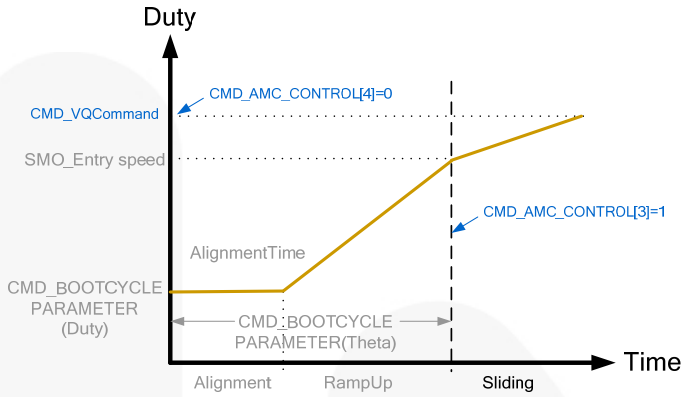
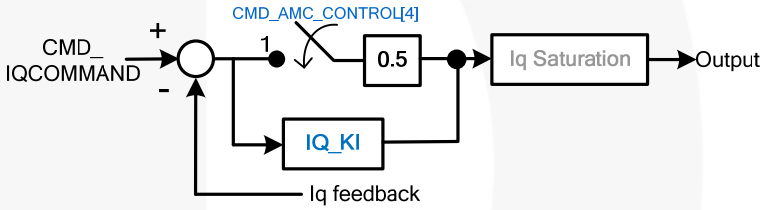
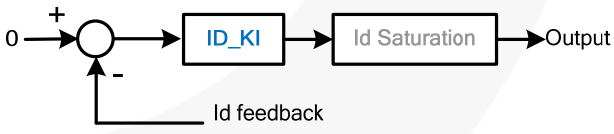


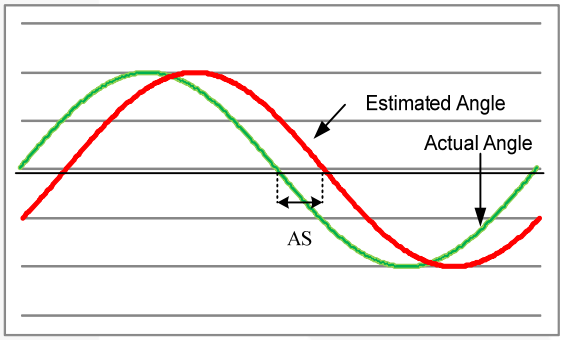
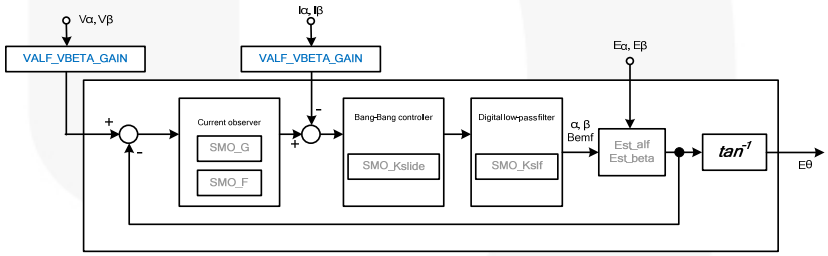
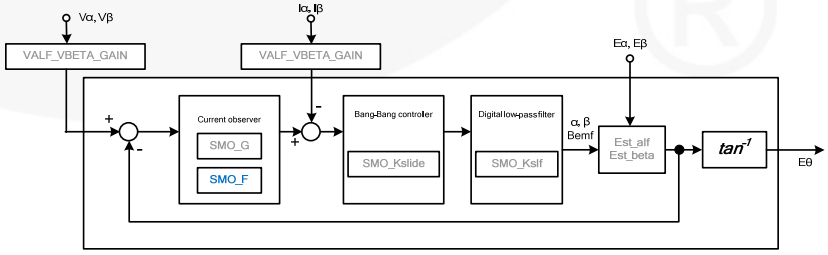
图 8. 滑模架构

参数说明

表 1. 参数列表

地址	参数名称	R/W	描述
0x10	CMD_AMC_CONTROL	W	位 0：电机运转/ 空闲　0：空闲；　1：运行 位 1：保留 位 2：LossStepCheck　0：禁用检查；　1：使能检查 位 3：ThetaSelect　0：MCU 发来的 Theta 值　1：滑模 位 4：IqLoop　0：禁用 Iq 环路　1：使能 Iq 环路 位 5：FluxWeakening_En　0：禁用　1：使能 位 6：IqKp_En　0：禁用　1：Enable（启用） 位 7：保留
0x12	CMD_IQ_INTEGRALSET	W	CMD_IQ_INTEGRALSET 用于设置 Iq 积分值。 该参数是一个 24 位的寄存器，取值范围 0~2^{24}。 MTX1 是最高位字节，MTX3 是最低位字节。
0x14	CMD_IQCOMMAND	W	CMD_IQCOMMAND_A 用于携带 Iq 指令的值。Iq 指令的赋值范围 0 至 511。 Iq 指令输入是通过令 CMD_AMC_CONTROL[4] = 1 来选择的。 CMD_IQCOMMAND = Iq (A) × 256 × Rshunt(Ω) × ADC Preamp MTX1：Iq 指令的高位字节数据 MTX2：Iq 指令的低位字节数据  **图 9. Parameter（参数）：IqCommand**
0x18	CMD_BOOTCYCLEPARAMETER	W	若 CMD_AMC_CONTROL [3] = 0，则占空比和 Theta 都通过该参数馈入 SVM。 占空比范围是 0 至 255，对应零负荷和满负荷。 Theta 是角度步进值（SVM 只有 192 个地址，需映射 360°，故 360/192 = 1.875°/步进）。 MTX1: Duty MTX2: Theta  **图 10. BootCycleParameter**

地址	参数名称	R/W	描述
0x1C	CMD_VQCOMMAND	W	CMD_VqCommand 用于设置 SVM 的占空比输入，它是一个取值范围为 0 至 255 的 8 位寄存器，对应零负荷至满负荷。Vq 指令输入是通过设定 CMD_AMC_CONTROL [4] = 0 来选择的。 MTX1：Vq 指令数据  **图 11. VqCommand**
0x1E	CMD_IQ_KI	W	CMD_IQ_KI 是 q 轴电流闭环中的 PI 控制器的积分常数。 该参数是一个 8 位寄存器，取值范围 0 至 255。默认值是 4Fh。 MTX2: Iq data  **图 12. Iq_Ki**
0x20	CMD_ID_KI	W	CMD_ID_KI 是 d- 轴电流闭环中 I 控制器的积分常数。 该参数是一个 8 位寄存器，取值范围 0 至 255。默认值是 4Fh。 MTX2：数据  **图 13. Id_Ki**

地址	参数名称	R/W	描述
0x24	CMD_AS	W	CMD_AS 指令包括 AS 数据和弱磁数据。 AS 数据是滑模的补角。 在滑模算法中，估算角度和实际角度之间会产生一个特定误差，如下所示。由于定子磁场和转子磁场不是正交的，必须根据 AS 纠正具体误差以实现更佳的电机效率。 该参数是一个 8 位寄存器，取值范围 -128 至 +127，对应 -180 度至 +180 度。 弱磁数据，即 Id 指令，是通过令 CMD_AMC_CONTROL[5] = 1 来选择的。该参数是一个 8 位寄存器，取值范围 0 至 255。 弱磁数据 = Id (A) × 256 × Rshunt(Ω) × ADC Preamp MTX1：AS 数据 MTX2：弱磁数据  **图 14. AS**
0x26	CMD_VALF_VBETA_GAIN	W	CMD_VALF_VBETA_GAIN 是电压指令和电流反馈的增益常数。 该参数是一个 8 位寄存器，赋值范围 0 至 15，默认值是 8。  **图 15. 滑模观测器**
0x28	CMD_SMO_F	W	SMO_F= $e^{(\frac{-Ra}{La}\times t_s)} \times 65536$ 其中， R_a：电机的相位电阻 (Ω) L_a：电机的相位电感 (H) t_s：125×10^{-6}（秒） 假设 R_a=0.3 Ω 且 L_a=47 mH， SMO_F= $e^{(\frac{-0.3}{0.047}\times 125*10^{-6})} \times 65536 = 65483$ 该参数是一个 16 位寄存器，取值范围 0 至 FFFFh。  **图 16. 滑模观测器**

地址	参数名称	R/W	描述
0x2A	CMD_SMO_G	W	SMO_G = G * Scale_Ratio 其中， G= $\frac{1}{R_a}(1-e^{\left(\frac{-Ra}{La}\times t_s\right)})\times 65536$ Scale_Ratio = DC_Bus(V) × Rshunt(Ω) × ADC Preamp /2 R_a：电机的相位电阻 (Ω) L_a：电机的相位电感 (H) t_s：125×10^{-6}（秒） ADC 前置放大器：1、2 和 4 假设 R_a = 0.3 Ω、L_a = 47 mH、DC 总线 = 300 V、Rshunt = 1 Ω、且 preamp = 1。 G= $\frac{1}{0.3}(1-e^{\left(\frac{-0.3}{0.047}\times125\times10^{-6}\right)})\times65536=174$ Scale_Ratio=300*1*1/2=150 SMO_G= 174*150=26100
0x2C	CMD_SMO_KSLIDE	W	SMO_Kslide 是 Bang-Bang 控制器的增益，值越大，反应越快。 该参数是一个 16 位寄存器，赋值范围 100 h 至 FFFFh。
0x2E	CMD_SMO_KSLF	W	SMO_Kslf 是数字滤波器的一个参数。值越小，截止频率越低，估计角度滞后越大。 该参数是一个 16 位寄存器，取值范围 100 h 至 FFFFh。
0x30	CMD_BANG_LIMIT	W	Bang_limit 是 Bang-Bang 控制器的输出钳位。值越小，反应越快。 该参数是一个 16 位寄存器，取值范围 400 h 至 2000 h。
0x38	CMD_SHUNT_OFFSET	W	当电机停转时，ADC 相位零电流校正数据。 赋值范围 – 128 至 127，对应 -512 至 511 ADC 数据。 MTX1：相位 A 校正数据 MTX2：相位 B 校正数据
0x42	CMD_OFFSETADJ_LOSESTEP	W	CMD_OFFSETADJ_LOSESTEP 包含三个参数：Valf_Adjust、LoseStep_deltaTheta 和 LoseStep_CheckTime。 Valf_Adjust 供用户微调因分量变化而偏置的相位电流的偏移。 LoseStep_deltaTheta 是电机损耗步进的阀值。当 Δ Theta 降低至小于 LoseStep_deltaTheta 时，内部计数器开始运行。 如果计数器达到 LoseStep_CheckTime，AMC 中 MRX0 的位 4 将设置为高并且电机将停止运行。 Valf_Adjust = MTX1 的低位半字节； LoseStep_deltaTheta = MTX1 的高位半字节，单位 3.125 H。 LoseStep_CheckTime = MTX2，单位 1.25 毫秒。

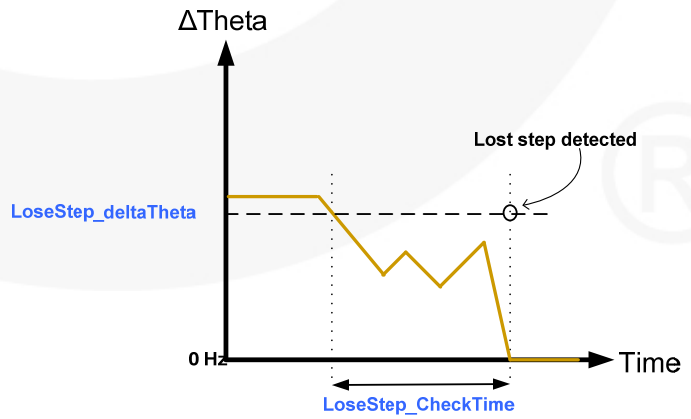
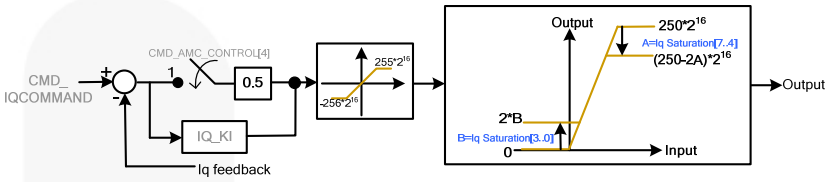
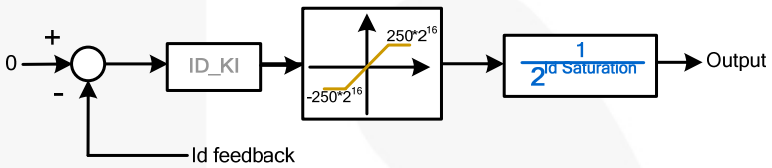
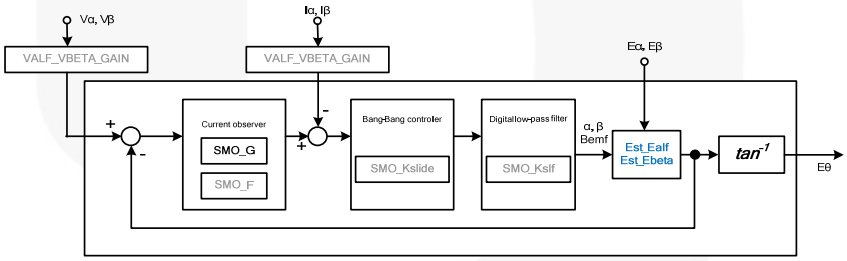


图 17. 损耗步进

地址	参数名称	R/W	描述
0x44	CMD_IQIDSATURATION	W	q- 轴电流控制器的积分饱和度，赋值范围 (0+2B)*2^{16} 至 (250-2A)*2^{16}。 A= Iq 饱和的高位半字节 B = Iq 饱和的低位半字节 d- 轴电流控制器的积分饱和度，赋值范围 -250*2^{16} 至 250*2^{16}，标度 $b_{\frac{1}{2^{Id\ Saturation}}}y$。 MTX1：Iq 饱和 MTX2：Id 饱和  **图 18. IQ 饱和**  **图 19. ID 饱和**
0x46	CMD_EALF_EBETA_SET	W	当电机停滞时，可根据 Est_Ealf 和 Est_Ebeta 设定两相静态参考系中估算反电动势的初始值。 Est_Ealf 和 Est_Ebeta 为 16 位寄存器，赋值范围 -32768 至 32512。 Est_Ealf =$MTX1 * 2^{(8-MTX3)}$ Est_Ebeta = $MTX2 * 2^{(8-MTX3)}$  **图 20. Ealf Ebeta 设置**
0x48	IDINTEGRALSET	W	CMD_ID_INTEGRALSET 用于设置 Id 积分值。 该参数是 24 位寄存器，赋值范围 0 至 2^{24}。 MTX1 是最高位字节，MTX3 是最低位字节。
0xFA	CMD_ENABLE_WATCHDOG	W	Watchdog 默认状态为禁用。 若要使能 Watchdog 功能，用户需要在 MCU 中使能 Watchdog 功能，并发送 “Command (CMD_ENABLE_WATCHDOG)” 至 AMC。

估算角度读取

MSFR 地址 0x07 用于直接读取估算角度。

通信

通信接口

滑模库参数均从 MCU 传输至 AMC。AMC 接收传输过来的参数并控制电机。参数传输通过 8 个邮箱寄存器实现，其中 4 个为 MTX0(B0h) - MTX3(B3)，另外 4 个为 MRX0(B4h) - MRX3(B7h)。前 4 个寄存器用于将信息从 MCU 传输至 AMC，后 4 个寄存器用于接收从 AMC 返回至 MCU 的信息，如图 21 所示。

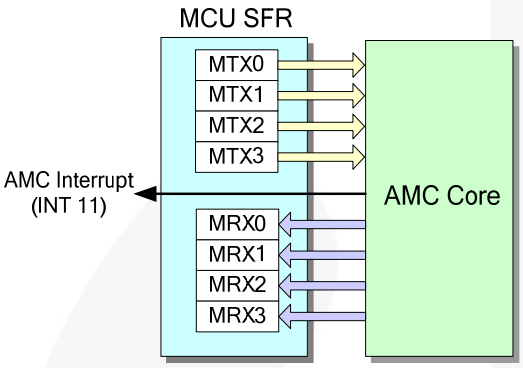


图 21. 通信接口

邮箱寄存器定义

要在 MCU 和 AMC 之间实现通信，用户必须遵守通信协议。本节介绍通信中所用各寄存器的定义。

MCU 数据传输 (MTX0 - MTX3)

MTX0:

b0（触发位）：

传输至 AMC 的指令和数据存储在相关寄存器中，然后将该位从 1 改为 0，从而令 AMC 开始接收指令和数据。

b1–b7（指令）：

存储传输至 AMC 的指令。

MTX1:

b0–b7（数据高位字节）：

存储传输至 AMC 的参数的高位字节。

MTX2:

b0–b7（数据低位字节）：

存储传输至 AMC 的参数的低位字节。

MTX3:

b0–b7（扩展数据）：

存储传输至 AMC 的参数的扩展字节。

MCU 数据读取 (MRX0 - MRX3)

MRX0:

b0: 保留

b1 (AMC_Cmd):

AMC_Cmd=1：AMC 正在处理指令，无法接收其他指令。

AMC_Cmd=0：AMC 可以接收指令。

b2 (AMC_Cal):

AMC_Cal=1：AMC 繁忙，无法接收指令。
AMC_Cal=0：AMC 不忙，可接受指令。

b3 (AMC_Fault):

AMC_Fault=1：AMC 运行异常。当出现 AMC_Fault 时，MCU 可重试或停止。

AMC_Fault=0：AMC 运行正常。

b4 (AMC_Lock):

AMC_Lock=1：检测到电机损耗步进。

AMC_Lock=0：电机工作正常。

b5-b7：保留

MRX1:

b0–b7（数据高位字节）：

接收来自 AMC 的参数的高位字节。

MRX2:

b0–b7（数据低位字节）：

接收来自 AMC 的参数的低位字节。

MRX3:

b0–b7（保留）：

表 2. 邮箱寄存器定义

字节名称（地址）	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MTX0 (B0h)	指令							触发
MTX1 (B1h)	数据高位字节							
MTX2 (B2h)	数据低位字节							
MTX3 (B3h)	保留							
MRX0 (B4h)	保留			AMC_Lock	AMC_Fault	AMC_Cal	AMC_Cmd	保留
MRX1 (B5h)	数据高位字节							
MRX2 (B6h)	数据低位字节							
MRX3 (B7h)	保留							

通信协议

必须遵守 MCU 和 AMC 之间的正确通信协议和流程，以免发生传输错误。通信协议和流程的正确步骤在下面流程图中介绍。

MCU 将指令写入 AMC

第 1 步：检查 AMC 是否繁忙。

若结果为“是”，则重复执行步骤 1，直到超时。

若结果为“否”，则执行步骤 2。

第 2 步：将指令和数据分别存储在对应寄存器中，然后将 MTX0 的 b0 从 1 变为 0，以开始传输。

步骤 3：等待 20 µs。

第 4 步：确认 AMC 是否已完成指令处理，即：

AMC_Cmd=1

若结果为“是”，则重复执行步骤 4，直到超时。

若结果为“否”，传输完成。

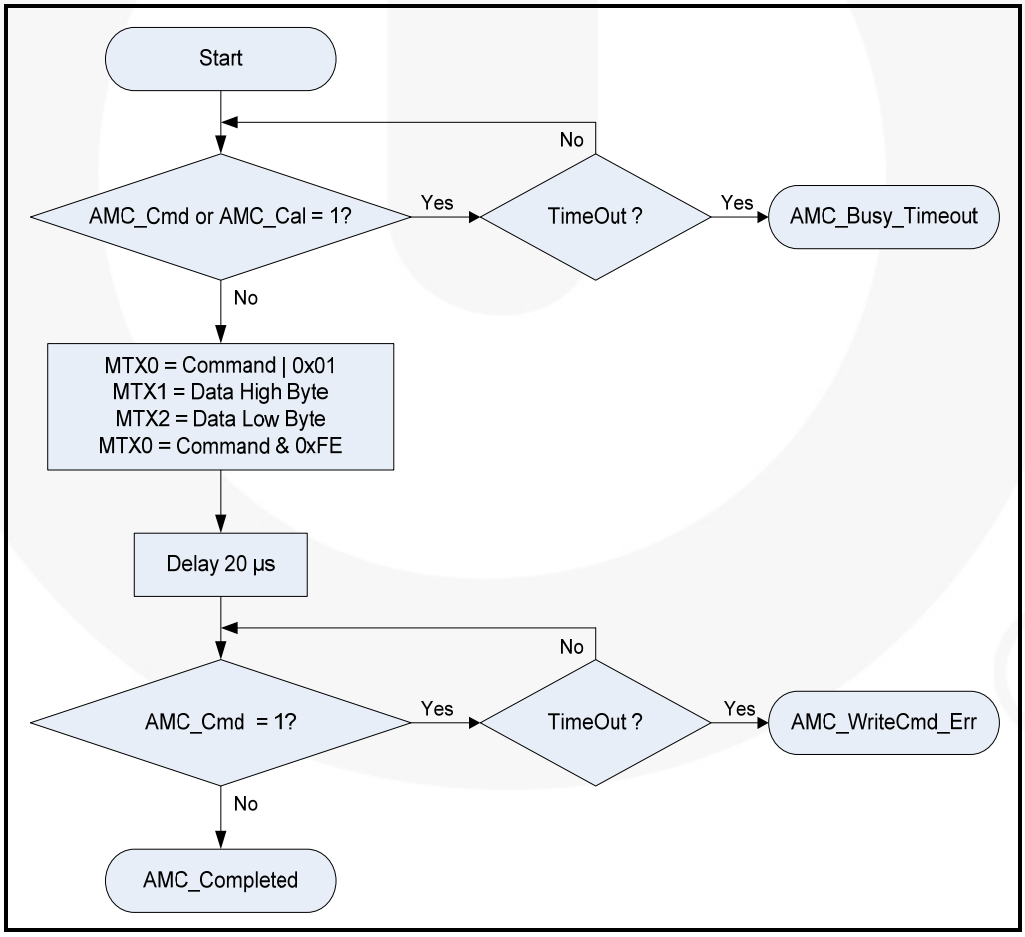


图 22. 写入指令协议

MCU 从 AMC 读取数据

第 1 步：检查 AMC 是否繁忙。

若结果为“是”，则重复执行步骤 1，直到超时。

若结果为“否”，则执行步骤 2。

第 2 步：将指令和数据分别存储在对应寄存器中，然后将 MTX0 的 b0 从 1 变为 0，以开始传输。

步骤 3：等待 20 µs。

第 4 步：确认 AMC 是否已完成指令处理，即：

AMC_Cmd=1

若结果为“是”，则重复执行步骤 4，直到超时。

若结果为“否”，则执行步骤 5。

步骤 5：检查 AMC 是否繁忙，即 (AMC_Cal=1)。

若结果为“是”，则重复执行步骤 5，直到超时。

若结果为“否”，读取完成并且数据自动存储到 MRX1 和 MRX2 中。

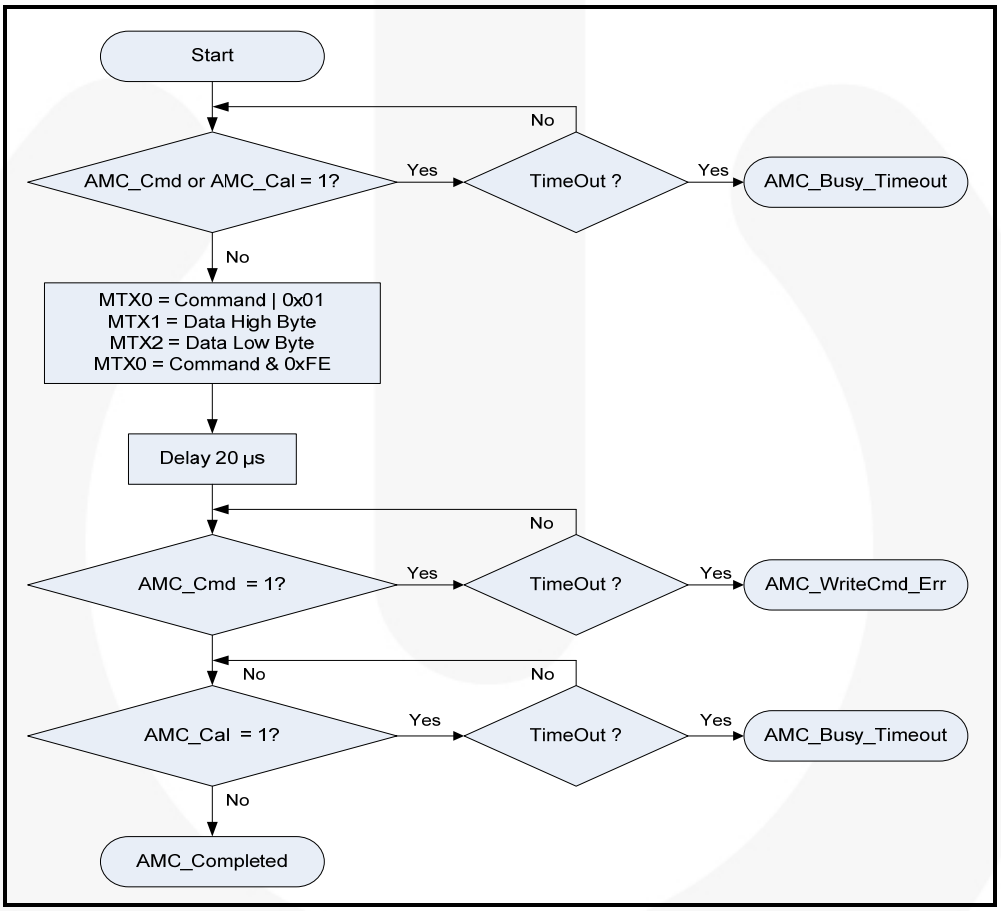


图 23. 读取数据协议

内置函数

方便起见，当在 MCDS IDE 软件中创建一个新项目时会生成两个函数。这些函数用于传输和读取 MCU 和 AMC 之间的数据以减少编码时间。下面对其进行介绍。

MCU 写数据至 AMC

U8 bWriteCmdToAMC(U8 bCommand, U8 bData_HB, U8 bData_LB)

bCommand：与已传输数据对应的指令

bData_HB：已传输数据的高位字节

bData_LB：已传输数据的低位字节

示例：

若要将 0x40 写入 Vq 指令，应该写为：

btWriteCmdToAMC(CMD_VQCommand, 0, 0x40)

执行后，如果函数返回值为 0，则代表数据发送成功。

滑模文件

在 MCDS IDE 中创建一个新项目文件时，会生成三个滑模相关文件。表 3 给出文件名和文件中的函数。

表 3. 滑模库的相关文件

文件名	说明	可修改
AMC_SlidingMode.c	提供了一些常见函数，如 1. Delay1 ms 2. Delay10 µs 3. bWriteCmdToAMC 4. Reset_AMC 5. btInitial_AMCToSlidingMode 6. btTransmit_ParameterToAMC	否
AMC_SlidingMode.h	所有指令和索引都在 AMC_SlidingMode.h 中声明。	否
Parameter_SlidingMode.h	滑模所需的所有参数都存储在 Parameter_SlidingMode.h 中。某些参数可以在 MCDS IDE 的电机调谐功能中进行测试和微调，如果按下“保存并生成文件”会覆盖这些参数。	是

上述文件的详细说明如下。

AMC_SlidingMode.c

介绍几个表 4 中所述的常见功能。

表 4. AMC_SlidingMode.c 通用函数

函数名称	说明
Delay1ms	等待 AMC，延迟时间 1 ms
Delay10us	等待 AMC，延迟时间 10 µs
bWriteCmdToAMC	将数据传输至 AMC
Reset_AMC	复位 AMC
btInitialize_AMCToSlidingMode	Initializing AMC
btTransmit_ParameterToAMC	将所有参数传输至 AMC

下面列出 AMC_SlidingMode.c 的内容。

第 19 行至第 40 行是滑模库的参数；

第 41 行至第 60 行是延迟函数；

第 61 行至第 90 行是写指令函数；

第 91 行至第 100 行是重置 AMC 函数；

第 101 行至第 112 行是初始化 AMC 函数；且

第 113 行至第 195 行是用于将默认数据传输至 AMC 的函数。

```
1   /*-------------------------------------------------------------------------
2    * Copyright 2013 Fairchild Semiconductor
3    * http://www.fairchildsemi.com
4    *-------------------------------------------------------------------------
5    */
6
7   //-------------------------------------------------------------------------
8   //  Includes
9   //-------------------------------------------------------------------------
```

```
10  #include "compiler-define.h"
11  #include "FCM8531.h"
12  #include "MSFR-define.h"
13  #include "AMC_SlidingMode.h"
14  #include "MCS51.h"
15  #include "Program.h"
16  #include "MotorCtrl.h"
17  #include "Parameter_SlidingMode.h"
18
19  //---------------------------------------------------------------------
20  //  Parameters
21  //---------------------------------------------------------------------
22  U16  SEG_XDATA _wSMO_F;
23  U16  SEG_IDATA _wSMO_Kslide;
24  U16  SEG_IDATA _wSMO_Kslf;
25  U16  SEG_XDATA _wSMO_Bang_limit;
26  U8   SEG_XDATA _bValf_Adjust;
27  S8   SEG_XDATA _cAS;
28  S8   SEG_XDATA _cAS_SMO_out;
29  U8   SEG_XDATA _bValf_Vbeta_Gain;
30  U8   SEG_XDATA _bLoseStep_deltaTheta;
31  U8   SEG_XDATA _bLoseStep_CheckTime;
32  U8   SEG_XDATA _bIq_Ki_init;
33  U8   SEG_XDATA _bId_Ki;
34  U8   SEG_XDATA _bAlignment_Theta;
35  U8   SEG_XDATA _bAlignment_Vq;
36  U8   SEG_IDATA _bOVERLOAD_DETECTING_TIME;
37  U8   SEG_IDATA _bLockRotorDelayTime;
38  UU16 SEG_IDATA _stTemp16;
39  U8   SEG_IDATA _bBootVqSet, SEG_IDATA _bBootThetaSet;
40
41  //---------------------------------------------------------------------
42  void Delay10 µs (U16 Counter) //Delay 10 µs
43  {
44      U16 i, k;
45      for ( i = 0; i < Counter; i++)
46          for(k = 0; k < 16; k++);
47  }
48
49  //---------------------------------------------------------------------
50  void Delay1 ms(U16 Counter)
51  {
52      // delay time : count * 1 mS
53      unsigned int i;
54
55      for ( i = 0; i < Counter; i++)
56      {
57          Delay10 µs(110);
58      }
59  }
60
61  //---------------------------------------------------------------------
62  U8 bWriteCmdToAMC(U8 bCommand, U8 bData_HB, U8 bData_LB)
63  {
64      U16 wWaitTime;
65      SEG_BIT btAMCStandby_Flag;
66
67      btAMCStandby_Flag = 0;
68      for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)                    // Check AMC calculating or Processing Command
69          if((MRX0 & (AMC_PROCESSING_CMD | AMC_CALCULATING)) == 0x0)
70          {
71              btAMCStandby_Flag = 1;
72              break;
73          }
74      if(!btAMCStandby_Flag)                                              // Check Time-Out
75          return(AMC_BUSY_TIMEOUT);
76
77      MTX0 = bCommand | MCU_MAILBOX_INTR_STOP;
78      MTX1 = bData_HB;
79      MTX2 = bData_LB;
80      MTX0 = bCommand & MCU_MAILBOX_INTR_START;
81
82      Delay10us(2);
83
84      for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
```

```
85          if((MRX0 & AMC_PROCESSING_CMD) == 0)
86              return(AMC_COMPLETED);
87
88      return(AMC_WRITE_CMD_ERROR);
89  }
90
91  //---------------------------------------------------------------------
92  void Reset_AMC()
93  {
94      U8 bDataByte;
95
96      READ_MSFR(MSFR_MCNTL, bDataByte);
97      WRITE_MSFR(MSFR_MCNTL, (bDataByte | 0x40));
98      WRITE_MSFR(MSFR_MCNTL, (bDataByte & 0xBF));
99
100 }
101 //---------------------------------------------------------------------
102 SEG_BIT btInitialize_AMCToSlidingMode()
103 {
104     SEG_BIT btError_code;
105
106     Reset_AMC();
107     Delay1 ms(100);
108
109     btError_code = btTransmit_ParameterToAMC();
110     Delay1 ms(1000);
111     return btError_code;
112 }
113 //---------------------------------------------------------------------
114 SEG_BIT btTransmit_ParameterToAMC(void)
115 {
116     U8 bWriteCMD_Status;
117
118
119     _stTemp16.U16 = _wSMO_F;
120     bWriteCMD_Status = bWriteCmdToAMC(CMD_SMO_F, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
121     if(bWriteCMD_Status)
122         return 1;
123
124     _stTemp16.U16 = SMO_G_DEF;
125     bWriteCMD_Status = bWriteCmdToAMC(CMD_SMO_G, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
126     if(bWriteCMD_Status)
127         return 1;
128
129
130     _stTemp16.U16 = _wSMO_Kslide;
131     bWriteCMD_Status = bWriteCmdToAMC(CMD_SMO_KSLIDE, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
132     if(bWriteCMD_Status)
133         return 1;
134
135     _stTemp16.U16 = _wSMO_Kslf;
136     bWriteCMD_Status = bWriteCmdToAMC(CMD_SMO_KSLF, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
137     if(bWriteCMD_Status)
138         return 1;
139
140     _stTemp16.U16 = _wSMO_Bang_limit;
141     bWriteCMD_Status = bWriteCmdToAMC(CMD_BANG_LIMIT, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
142     if(bWriteCMD_Status)
143         return 1;
144
145
146     _stTemp16.U16 = _bValf_Vbeta_Gain;
147     bWriteCMD_Status = bWriteCmdToAMC(CMD_VALF_VBETA_GAIN, 0, _stTemp16.U8[LSB]);
148     if(bWriteCMD_Status)
149         return 1;
150
151
152     _stTemp16.U8[MSB] = _bIq_Ki_init;
153     _stTemp16.U8[LSB] = _bId_Ki;
154     bWriteCMD_Status = bWriteCmdToAMC(CMD_ID_IQ_KI, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
155     if(bWriteCMD_Status)
156         return 1;
157
158
159     _stTemp16.U8[MSB] = 0xA5;
```

```
160     _stTemp16.U8[LSB] = 0x03;
161     bWriteCMD_Status = bWriteCmdToAMC(CMD_IQIDSATURATION, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
162     if(bWriteCMD_Status)
163         return 1;
164
165     _stTemp16.U16 = _bLockRotorDelayTime;
166     bWriteCMD_Status = bWriteCmdToAMC(CMD_LOCKROTORDELAYTIME, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
167     if(bWriteCMD_Status)
168         return 1;
169
170     _stTemp16.U8[MSB] = (_bLoseStep_deltaTheta<<4) | (_bValf_Adjust & 0x0f);
171     _stTemp16.U8[LSB] = _bLoseStep_CheckTime;
172     bWriteCMD_Status = bWriteCmdToAMC(CMD_OFFSETADJ_LOSESTEP, _stTemp16.U8[MSB], _bLoseStep_CheckTime);
173     if(bWriteCMD_Status)
174         return 1;
175
176     _stTemp16.U16 = _bOVERLOAD_DETECTING_TIME;
177     bWriteCMD_Status = bWriteCmdToAMC(CMD_OVERLOAD_DETECTING_TIME, _stTemp16.U8[MSB], _stTemp16.U8[LSB]);
178     if(bWriteCMD_Status)
179         return 1;
180
181     _bBootVqSet = _bAlignment_Vq;
182     _bBootThetaSet = _bAlignment_Theta;
183     bWriteCMD_Status = bWriteCmdToAMC(CMD_BOOTCYCLEPARAMETER, _bBootVqSet, _bBootThetaSet);
184     if(bWriteCMD_Status)
185         return 1;
186
187     _stTemp16.S8[LSB] = _cAS;
188     _stTemp16.S8[MSB] = _cAS_SMO_out;
189     bWriteCMD_Status = bWriteCmdToAMC(CMD_AS, _stTemp16.S8[MSB], _stTemp16.S8[LSB]);
190     if(bWriteCMD_Status)
191         return 1;
192
193     return 0;  //initial pass
194 }
195 //-----------------------------------------------------------------------
```

AMC_SlidingMode.h

对读取 AMC 数据的指令、参数和索引，以及恒量进行了声明。下面列出 AMC_SlidingMode.h 的代码。

第 10 行至第 37 行是 AMC ID；

第 38 行至第 56 行是 AMC_CONTROL 位定义；

第 57 行至第 62 行是 AMC 状态索引；

第 63 行至第 90 行是用 extern 定义的函数。

```
1   //-----------------------------------------------------------------------
2   // Copyright 2013 Fairchild Semiconductor
3   // http://www.fairchildsemi.com
4   //-----------------------------------------------------------------------
5   #ifndef  _AMC_SlidingMode_H_
6   #define  _AMC_SlidingMode_H_
7   //-----------------------------------------------------------------------
8   //          Parameter ID define for user (Sliding Mode)
9   //-----------------------------------------------------------------------
10  #define CMD_AMC_CONTROL                 0x10
11  #define CMD_IQINTEGRALSET               0x12
12  #define CMD_IQCOMMAND                   0x14
13  #define CMD_BOOTCYCLEPARAMETER          0x18
14  #define CMD_VQCOMMAND                   0x1C
15  #define CMD_ID_IQ_KI                    0x20
16  #define CMD_AS                          0x24
17  #define CMD_VALF_VBETA_GAIN             0x26
18  #define CMD_SMO_F                       0x28
19  #define CMD_SMO_G                       0x2A
20  #define CMD_SMO_KSLIDE                  0x2C
21  #define CMD_SMO_KSLF                    0x2E
22  #define CMD_BANG_LIMIT                  0x30
23  #define CMD_LOCKROTORDELAYTIME          0x34
```

```
#define CMD_OVERLOAD_DETECTING_TIME     0x36
#define CMD_SHUNT_OFFSET                0x38
#define CMD_OFFSETADJ_LOSESTEP          0x42
#define CMD_IQIDSATURATION              0x44
#define CMD_EALF_EBETA_SET              0x46
#define CMD_IDINTEGRALSET               0x48
#define CMD_ENABLE_WATCHDOG             0xFA
#define CMD_READ_PARAMETER              0xFE
//-----------------------------------------------------------------------
#define MCU_MAILBOX_INTR_START          0xFE
#define MCU_MAILBOX_INTR_STOP           0x01
#define AMC_PROCESSING_CMD              0x02
#define AMC_CALCULATING                 0x04
#define AMC_FAULT                       0x08
//-----------------------------------------------------------------------
//                          AMC Control
//-----------------------------------------------------------------------
#define MOTOR_RUN_BIT                   0x01
#define LOSESTEP_BIT                    0x04        // Lose step check enable. 0 : disable, 1 : enable
#define THETA_SELECT_BIT                0x08        // theta enable. 0 : MCU, 1: Sliding mode calculate
#define IQLOOP_BIT                      0x10        // Iq_Vq_SelectBit,  Iq or Vq command select. 0 : Vq command, 1 : Iq command
#define FLUXWEAKENING_BIT               0x20
#define IQ_KP_EN_BIT                    0x40


#define MOTOR_RUN_BIT_CLR               0xFE
#define LOSESTEP_BIT_CLR                0xFB        // Lose step check enable. 0 : disable, 1 : enable
#define THETA_SELECT_BIT_CLR            0xF7        // theta enable. 0 : MCU, 1: Sliding mode calculate
#define IQLOOP_BIT_CLR                  0xEF        // Iq or Vq select
#define FLUXWEAKENING_BIT_CLR           0xDF
#define IQ_KP_EN_BIT_CLR                0xBF

//-----------------------------------------------------------------------
//  AMC status code
//-----------------------------------------------------------------------
#define AMC_COMPLETED                   0x00
#define AMC_BUSY_TIMEOUT                0x01
#define AMC_WRITE_CMD_ERROR             0x02
//-----------------------------------------------------------------------
extern U16  SEG_XDATA _wSMO_F;
extern U16  SEG_IDATA _wSMO_Kslide;
extern U16  SEG_IDATA _wSMO_Kslf;
extern U16  SEG_XDATA _wSMO_Bang_limit;
extern U8   SEG_XDATA _bValf_Adjust;
extern S8   SEG_XDATA _cAS;
extern S8   SEG_XDATA _cAS_SMO_out;
extern U8   SEG_XDATA _bValf_Vbeta_Gain;
extern U8   SEG_XDATA _bLoseStep_deltaTheta;
extern U8   SEG_XDATA _bLoseStep_CheckTime;
extern U8   SEG_XDATA _bIq_Ki_init;
extern U8   SEG_XDATA _bId_Ki;
extern U8   SEG_XDATA _bAlignment_Theta;
extern U8   SEG_XDATA _bAlignment_Vq;
extern U8   SEG_IDATA _bOVERLOAD_DETECTING_TIME;
extern U8   SEG_IDATA _bLockRotorDelayTime;
extern UU16 SEG_IDATA _stTemp16;
extern U8   SEG_IDATA _bBootVqSet, SEG_IDATA _bBootThetaSet;

extern void Delay10 μs(U16 count);
extern void Delay1 ms(U16 count);
extern U8 bWriteCmdToAMC(U8 bCommand, U8 bData_HB, U8 bData_LB);
extern void Reset_AMC(void);
extern SEG_BIT btTransmit_ParameterToAMC(void);
extern SEG_BIT btInitialize_AMCToSlidingMode();
extern void Fault_Led_Flash();

#endif
```

Parameter_SlidingMode.h

声明了滑模所需的所有参数。下面列出 Parameter_SlidingMode.h 的代码。

第 18 行至第 31行是滑模算法的参数；

第 33 至第 35 行是电机指令的参数；且

第 37 行至第 44 行指电流控制环路的参数；

第 46 行至第 51 行是电机对准的参数；

第 53 行至第 59 行是电机加速的参数；

第 61 行至第 67 行是速度控制回路的参数；

第 61 行至第 63 行是 IEC 60730 的参数；

第 73 行至第 76 行是电压控制环路的参数；

第 78 行至第 80 行是保护参数；且

第 82 行至第 91 行是故障消息号参数；

```
1   /*-------------------------------------------------------------------------
2    * Copyright 2013 Fairchild Semiconductor
3    * http://www.fairchildsemi.com
4    *
5    * Motor Parameters:
6    *  -1. Inductance between the motor line: 490 mH
7    *  -2. Resistance between the motor line: 75750 mΩ
8    *  -3. Motor rotor pole: 8
9    *  -4. Input DC voltage: 300 V
10   *  -5. Shunt resistor value of drive: 1000 mΩ
11   *
12   *-------------------------------------------------------------------------
13   */
14
15  #ifndef  _Parameter_SlidingMode_h_
16  #define  _Parameter_SlidingMode_h_
17
18  //-----Motor and SMO Parameters
19  #define  SMO_F_DEF                          0xFB19
20  #define  SMO_G_DEF                          0x1367
21  #define  SMO_KSLIDE_DEF                     0x2000
22  #define  SMO_KSLF_DEF                       0x1770
23  #define  SMO_BANG_LIMIT_DEF                 0x1000
24  #define  VALF_ADJUST_DEF                    0x0
25  #define  AS_DEF                             0
26  #define  AS_SMO_DEF                         -64
27  #define  LEAD_ANGLE_DEF                     16
28  #define  VALF_VBETA_GAIN_DEF                0x8
29  #define  MOTORPOLE_DEF                      0x8
30  #define  LOSESTEP_DELTATHETA_DEF            0x2
31  #define  LOSESTEP_CHECKTIME_DEF             0x5
32
33  //-----Motor Command
34  #define  MOTOR_DIRECTION_DEF                0x0    // 1:ccw, 0:cw
35  #define  OPERATION_MODE_DEF                 0x0    // 0:Vq, 1:Iq, 2:Vq+Speed, 3:Iq+Speed
36
37  //-----Current Control Loop
38  #define  IQ_COMMAND_DEF                     0x64
39  #define  IQ_INIT_DEF                        0x64
40  #define  IQ_COMMAND_MAX_DEF                 0xF0
41  #define  IQ_COMMAND_MIN_DEF                 0x1
42  #define  IQ_KI_INIT_DEF                     0xF
43  #define  IQ_KI_RUN_DEF                      0x4F
44  #define  ID_KI_DEF                          0x4F
45
46  //-----Alignment Parameter
47  #define  ALIGNMENT_THETA_DEF                0x0
48  #define  ALIGNMENT_VQ_DEF                   0x1
49  #define  ALIGNMENT_TIME_DEF                 0x3E8
50  #define  ALIGNMENT_SPEED_DEF                0x32
```

```
51  #define  SMO_ENTRY_SPEED_DEF                0x12C
52
53  //-----Rampup Parameter
54  #define  RAMPUP_TIMEOUT_DEF                 0x1194
55  #define  RAMPUP_SLOPE_DEF                   0x2     // 100 rpm/s
56  #define  ASCNTSET_DEF                       0x1
57  #define  SYNCSPEED_DEF                      0x12C
58  #define  SYNCDELYTIME_DEF                   0x0
59  #define  SPEEDLOOPDELAYTIME_DEF             0x0     //mS, 0~3000
60
61  //-----Speed Control Loop
62  #define  SPEED_COMMAND_DEF                  0x258
63  #define  SPEED_OUTPUTLIMIT_MAX_DEF          0xE6
64  #define  SPEED_OUTPUTLIMIT_MIN_DEF          0xA
65  #define  SPEED_KP_DEF                       0x2000
66  #define  SPEED_KI_DEF                       0x20
67  #define  SPEED_SLOPE_DEF                    0x4
68
69  //-----UL60730
70  #define  OVERLOAD_DETECTING_TIME_DEF        0x6A
71  #define  LOCKROTORDELAYTIME_DEF             0xF
72
73  //-----Voltage Control Loop
74  #define  VQ_COMMAND_DEF                     0xF0
75  #define  VQ_COMMAND_MAX_DEF                 0xFA
76  #define  VQ_COMMAND_MIN_DEF                 0x14
77
78  //-----OVP & UVP
79  #define  VBUSHIVOLTAGE_DEF                  0xC2
80  #define  VBUSLOVOLTAGE_DEF                  0x81
81
82  //-----Fault Display
83  #define  AMCFAULTDISP                       1
84  #define  OCSFAULTDISP                       2
85  #define  FOFAULTDISP                        3
86  #define  LOSSSTEPFAULTDISP                  4
87  #define  OVPFAULTDISP                       5
88  #define  UVPFAULTDISP                       6
89  #define  TIMEOUTFAULTDISP                   7
90  #define  UL60730FAULTDISP                   8
91  #define  REVERSEFAULTDISP                   9
92
93  #endif
```

如何使用滑模

下面几节介绍如何使用滑模。

初始化滑模

本文说明如何初始化滑模库。

滑模库是在 MCDS IDE 的项目设置窗口选择的，如图 24 所示。有关 MCDS IDE 的详细信息，请参考 AN-8207 — FCM8531 MCDS IDE 用户指南。

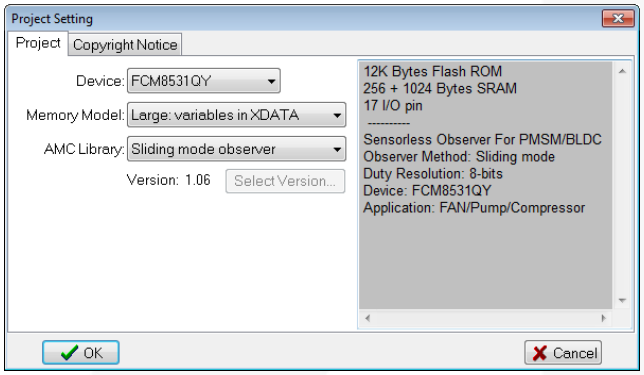


图 24. MCDS IDE - 项目设置

初始化

在重置 AMC 后，PWM 和 ADC 必须由 MCU 设置。图 25 为 MCU 的建议启动流程图，包括 MCU 初始化、AMC 重置、中断使能，以及将默认参数写入 AMC。

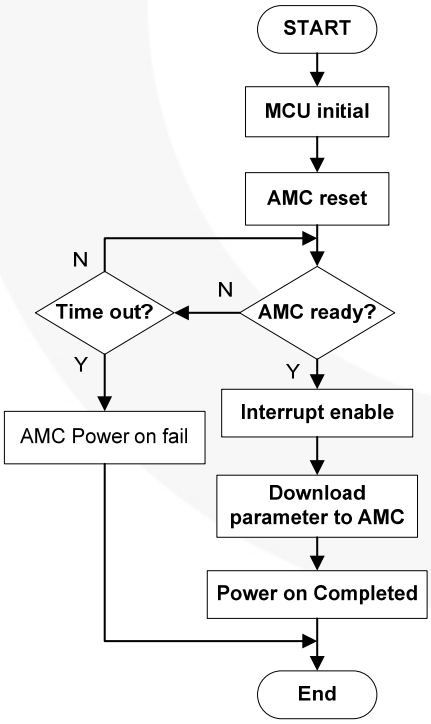


图 25. MCU 初始化流程图

PWM

下面列出 PWM 相关的 MSFR 和对应的地址。有关详细信息，请参考 FCM8531 数据表。

- MCNTL 0x00h
- PWMCFG 0x08h
- SAWCNTL 0x09h
- SPRDL 0x0Ah
- SPRDH 0x0Bh

示例项目中的配置如下。

1. 上下锯切模式；
2. PWM 载频是 14.7 kHz；
3. 自动缩放模式；锯切周期 0x3FF；且
4. 死区时间：2.0 µsec.

示例代码

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */

#define INITIAL_MCNTL       0x20
#define INITIAL_PSMCFG0x00
#define INITIAL_SPRD        0x3FF
#define PwmDefault          0xA0

void Initial_PWM()
{
   WRITE_MSFR(MSFR_PWMCFG, INITIAL_PWMCFG);
   WRITE_MSFR(MSFR_SPRDH, INITIAL_SPRD >> 3);
   WRITE_MSFR(MSFR_SPRDL, INITIAL_SPRD << 5);
   WRITE_MSFR(MSFR_SAWCNTL, INITIAL_SAWCNTL);
   WRITE_MSFR(MSFR_MCNTL, INITIAL_MCNTL);
}
```

ADC

下面列出 ADC 的相关 SFR 以及对应地址。有关详细信息，请参考 FCM8531 数据表。

- ADCFG 0x1Fh
- IAL0x20h
- IBL 0x22h
- ICL 0x24h
- DAC3 0x47h

示例项目中的配置设置如下。

1. 负向输入偏置电流；
2. IA、IB、IC 的前置放大器增益是 1x；
3. 清除 DAC 输出数据；
4. DAC 输出；
5. SAW- 顶部 ADC 触发器；且
6. 采样速率分频器为 1。

示例代码

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
#define INITIAL_ADCFG       0x20
#define INITIAL_VIAL        0x04
#define INITIAL_VIBL        0x04
#define INITIAL_VICL        0x04
#define INITIAL_DAC3        0x00

void Initial_ADC()
{
   WRITE_MSFR(MSFR_ADCFG, INITIAL_ADCFG);
   WRITE_MSFR(MSFR_IAL, INITIAL_VIAL);
   WRITE_MSFR(MSFR_IBL, INITIAL_VIBL);
   WRITE_MSFR(MSFR_ICL, INITIAL_VICL);
   WRITE_MSFR(MSFR_DAC3, INITIAL_DAC3);
}
```

角度输入

下面列出角度输入源的相关 SFRs 和对应地址。有关详细信息，请参考 FCM8531 数据表。

- ANGCTL 0x01h
- AS 0x02h
- ANGDET 0x03h

示例项目中的配置设置如下。

1. 角度输入源：MSFR;
2. 重置位差角；
3. 正弦波由 SIN_EA 位使能；且
4. SIN_EA=1.

示例代码

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
#define INITIAL_ANGCTL      0x00
#define INITIAL_AS          0x00
#define INITIAL_ANGDET      0xC0

void Initial_Angle()
{
   WRITE_MSFR(MSFR_ANGCTL, INITIAL_ANGCTL);
   WRITE_MSFR(MSFR_AS, INITIAL_AS);
   WRITE_MSFR(MSFR_ANGDET, INITIAL_ANGDET);
}
```

Interrupt

下面列出初始值、默认值和对应地址的相关 SFR。有关详细信息，请参考 FCM8531 数据表。

- IP0 0xA9
- IP1 0xB9
- IT0 0x88h bit0
- EX0 0xA8h bit0
- IEN1 0xB8
- IEN2 0x9Ah

示例项目中的配置设置如下：

1. 设置来自 SPM® 的外部过流保护信号、下降沿触发器以及外部中断 0；
2. 禁用中断；
3. 清除所有事件标志；
4. 重新使能中断。

示例代码

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
#define INITIAL_IP0         0x03        //G0:L3,G4:L2,G1:L1, the others:L0
#define INITIAL_IP1         0x11
#define INITIAL_IEN1        0x00
#define INITIAL_IEN2        0x12
void Initial_Interrupt()
{
   // Interrupt Priority
   IP0 = INITIAL_IP0;
   IP1 = INITIAL_IP1;

   // Interrupt
   EX0 = 1;          // External Interrupt 0 Enable
   IT0 = 1;          // External Interrupt 0 Falling edge trigger
   IEN1 = INITIAL_IEN1;
   IEN2 = INITIAL_IEN2;
}
```

马达控制

在启动程序之前，必须从四个 AMC 运行模式中选择一个。

这四个模式为以下四个 Theta 输入：

1. MCU，电压控制
2. MCU，电流控制
3. 滑模观测器，电压控制
4. 滑模观测器，电流控制。

VqCommand、IqCommand 和 CMD_AMC_CONTROL 通过邮箱寄存器的 btWriteCmdToAMC 函数写入 AMC 内核。示例代码在启动示例代码章节中介绍。

选择 SVM 的 Vq 或 Iq 指令

Vq 控制（默认）

```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
&= IqLoopBitClr);
```

Iq control

```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
|= IqLoopBitClr);
```

写指令到 AMC

Vq Control

```
btWriteCmdToAMC (CMD_VqCommand, 0, dataLobyte);
```

Iq Control

```
btWriteCmdToAMC (CMD_IqCommand, dataHibyte,
dataLobyte);
```

选择 SVM 的 theta 输入

来自 MCU 的 theta 输入

```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
&= ThetaSelectBitClr);
```

来自滑模观测器的 theta 值

```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
|= ThetaSelectBitClr);
```

theta 值输入和指令输入的选择配置

需要设置下面两个选择位。

- CMD_AMC_CONTROL[3]=0 来自 MCU 的 theta 输入
- CMD_AMC_CONTROL [3]=1 来自滑模观测器的 theta 输入
- CMD_AMC_CONTROL [4]=0 电压控制
- CMD_AMC_CONTROL [4]=1 电流控制

下面列出四个对应的设置。

1. 来自 MCU 的 theta 输入，电压控制

- btWriteCmdToAMC(CMD_BootCycleParameter, BootVqSet, BootThetaSet);
- BootVqSet = 电压指令
- BootThetaSet = SVM 的 theta 值 (0~191)

2. 来自 MCU 电流控制的 theta 输入

- btWriteCmdToAMC(CMD_BootCycleParameter, BootVqSet, BootThetaSet);
- BootThetaSet = SVM 的 theta 值 (0~191)
- BootVqSet = 不适用
- btWriteCmdToAMC (CMD_IqCommand, dataHibyte, dataLobyte);
- dataHibyte：电流指令高位字节
- dataLobyte：电流指令低位字节

3. 来自电压控制滑模观测器的 theta 输入

- btWriteCmdToAMC (CMD_VqCommand, 0, dataLobyte);
- dataLobyte = 电压指令

4. 来自滑模观测器电流控制的 theta 输入

- btWriteCmdToAMC (CMD_IqCommand, dataHibyte, dataLobyte);
- dataHibyte：电流指令高位字节
- dataLobyte：电流指令低位字节

角度补偿

禁用（默认）

```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
&= ThetaCompensateBitClr);
```

Enable（启用）

```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
|= ThetaCompensateBitClr);
```

电机运行/停止

空载

```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
|= MotorRunBitClr);
```

Stop（停止）

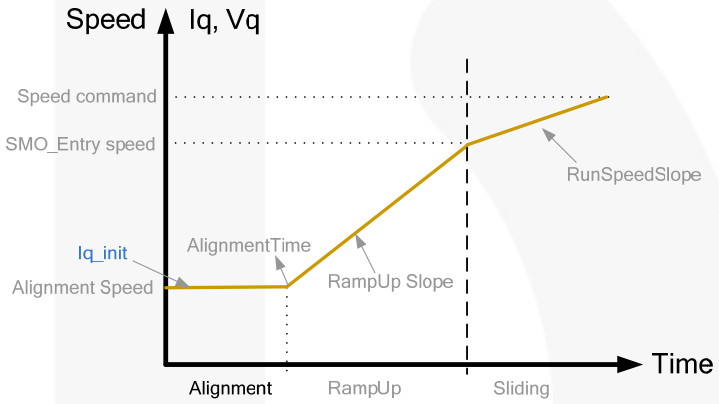
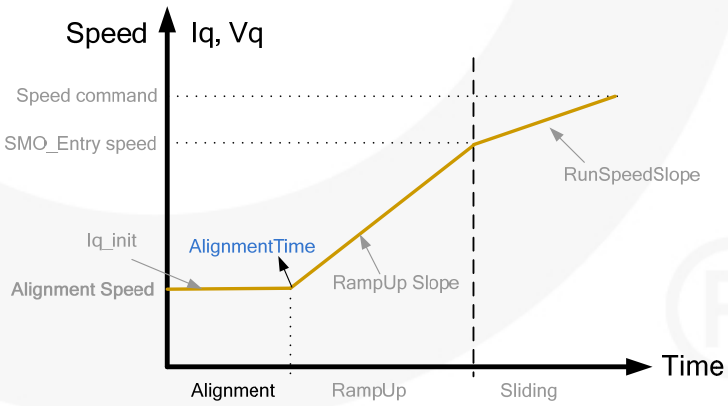
```
btWriteCmdToAMC(CMD_AMC_CONTROL, 0, AMCconfig
&= MotorRunBitClr);
```

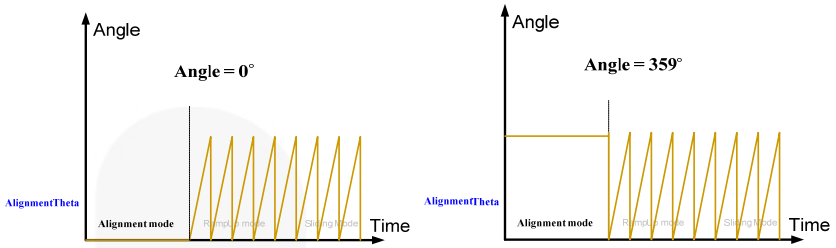
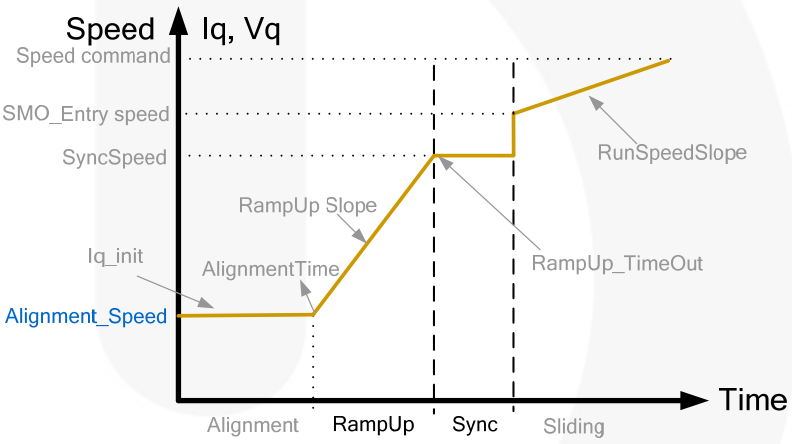
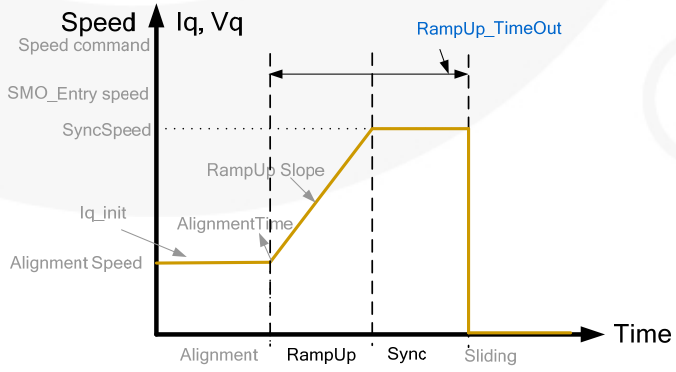
PWM 输出由 MCNTL[0] 控制，在执行 run 指令之前必须设置为“1”。

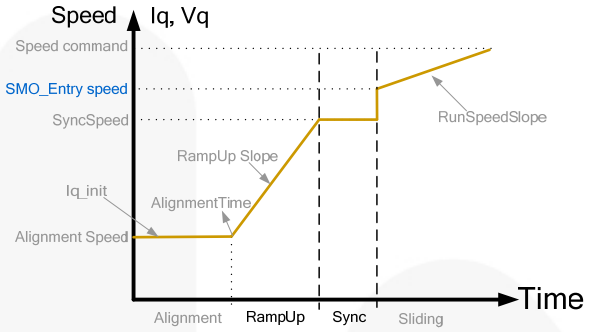
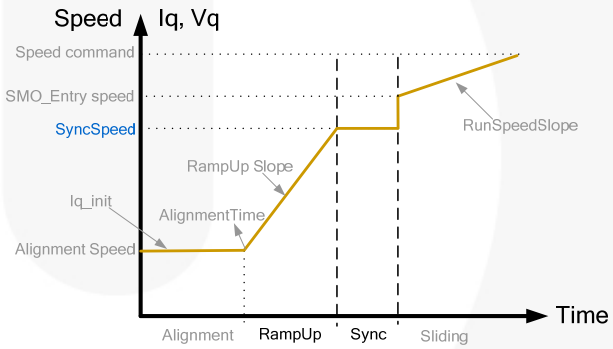
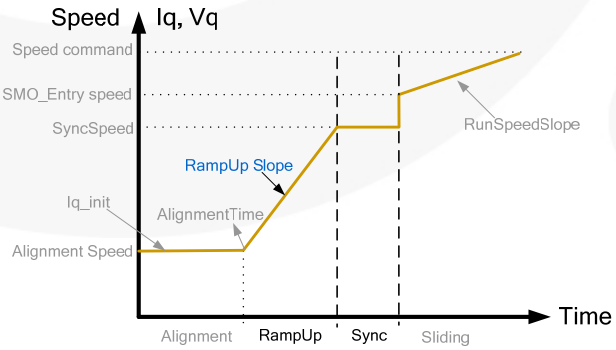
电机启动参数

业界有几种常用的启动方法。本节和下节将介绍一种简单的启动程序。该程序和相关的参数适用于不同的应用。这里先介绍参数，下节介绍示例代码。

表 5. 电机启动参数列表

参数名称	描述
_wIq_Init	_wIq_Init is the Iq command to align the rotor to a specific position before Ramp-up. _wIq_Init = Iq (A) × 256 × Rshunt(Ω) × ADC Preamp  **图 26. _wVq_init**
_wAlignment_Time	_wAlignment_Timeis time waiting until the motor is stably aligned to the specific position and in unit of millisecond.  **图 27. _wAlignment_Time**

参数名称	描述
_bAlignment_Theta	AlignmentTheta_init 是开始斜升前使转子与特定位置对齐的角度。它是一个 8 位参数，赋值范围 0 至 255，对应 0 到 359°。  **图 28. _bAlignment_Theta**
_wAlignment_Speed	在对齐模式期间，可以选择两种子模式。 按过去的做法，转子被推动到特定的位置并固定在那。An advanced method is to run the rotor at an arbitrary low speed to overcome static friction. 传统方法中，_wAlignment_Speed 可以设置为零，在这种先进方法中，它可以设置为任意值。 它是一个 8 位参数，赋值范围 0 至 255，对应 0 至 255 rpm。  **图 29. _wAlignment_Speed**
_wRampUp_TimeOut	_wRampUp_Timeout is the duration of ramp-up。定期检测估算角度和斜坡上升角度之间的角度差，以便在同步模式中汇合。如果角度差在 _wRampUp_TimeOut 时间内无法汇合，则电机停止运行。 它是一个 16 位参数，赋值范围 0 至 65535，对应 0 至 65535 ms。  **图 30. _wRampUp_TimeOut**

参数名称	描述
_wSMO_Entry_Speed	_wSMO_Entry_Speed 是用来在短时间内加速电机的速度指令，能够在从同步模式切换到滑模之前，跳过机械设计中的共振带。 该参数是可选的，并且如果在进入滑模之前机械振动不可见，则可以将该参数设置为与 SyncSpeed 相同的值。 它是一个 16 位参数，赋值范围 0 至 65535，对应 0 至 65535 rpm。
	 **图 31. _wSMO_Entry_Speed**
_wSyncSpeed	_wSyncSpeed 是一个速度指令，它能够将电机上升到滑模观测器可以开始正确估算转子角度的的速度。 它是一个 16 位参数，赋值范围 0 至 65535，对应 0 至 65535 rpm。 注意：同步模式时间：1* 定子的 delta theta（毫秒），delta theta 0~255 对应 0~359。
	 **图 32. _wSyncSpeed**
_bRampUp_Slope	_bRampUp_Slope 是斜坡上升的变化率。它从 _wAlignment_Speed 变为 _wSyncSpeed，单位是 50 rpm/s。
	 **图 33. _bRampUp_Slope**

启动示例代码

It can be found in Unitily_SlidingMode.C of the example project.

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */

//---------------------------------------------------------------------------
//  Includes
//---------------------------------------------------------------------------
#include "compiler-define.h"
#include "FCM8531.h"
#include "MSFR-define.h"
#include "AMC_SlidingMode.h"
#include "MCS51.h"
#include "Program.h"
#include "MotorCtrl.h"
#include "Utility_SlidingMode.h"
#include "Parameter_SlidingMode.h"

UU16 SEG_XDATA _stSMO_G;
S8   SEG_XDATA _cAS_SMO;
S8   SEG_XDATA _cLead_angle;
U8   SEG_XDATA _bMotorPole;
UU16 SEG_IDATA _stIq_Command;
U16  SEG_XDATA _wIq_Init;
U16  SEG_IDATA _wIq_Command_max;
U16  SEG_IDATA _wIq_Command_min;
bit            _btIqEnable;
U8   SEG_XDATA _bIq_Ki_Run;
U16  SEG_XDATA _wAlignment_Time;
U16  SEG_XDATA _wAlignment_Speed;
U16  SEG_XDATA _wSMO_Entry_Speed;
U16  SEG_XDATA _wRampUp_TimeOut;
U8   SEG_XDATA _bRampUp_Slope;
U16  SEG_XDATA _wRampUp_SlopeGain;
U8   SEG_XDATA _bAScntSet;
U16  SEG_XDATA _wSyncSpeed;
U16  SEG_XDATA _wSyncDelyTime;
U16  SEG_IDATA _wSpeed_Command;
bit            _btSpeed_Enable;
U16  SEG_XDATA _wSpeed_OutputLimit_max;
S32  SEG_IDATA _dwSpeed_OutputLimit_maxQ16;
U16  SEG_XDATA _wSpeed_OutputLimit_min;
S32  SEG_IDATA _dwSpeed_OutputLimit_minQ16;
U16  SEG_IDATA _wSpeed_Kp;
U16  SEG_IDATA _wSpeed_Ki;
U8   SEG_IDATA _bSpeed_Slope;
U8   SEG_IDATA _bVq_Command;
U8   SEG_IDATA _bVq_Command_Buf;
U8   SEG_IDATA _bVq_Command_max;
U8   SEG_IDATA _bVq_Command_min;
bit            _btMotor_Direction;
U8   SEG_XDATA _bOCH_init;
U8   SEG_XDATA _bOCL_init;
U8   SEG_XDATA _bSHORT_init;
U8   SEG_XDATA _bOperation_Mode;
UU16 SEG_IDATA _stTemp16_Turning;
U8   SEG_IDATA _bTemp16_Turning1;
UU16 SEG_XDATA _stAlignmentTimeCountSet;
U8   SEG_IDATA _bAMCcontrol;
U8   SEG_IDATA _bEstTheta;
U8   SEG_IDATA _bOld_EstTheta;
UU16 SEG_IDATA _stSpeedKp;//Q16
UU16 SEG_IDATA _stSpeedKi;//Q16
UU16 SEG_IDATA _stSpeedCmd;
UU16 SEG_IDATA _stKslfSlope;
UU16 SEG_IDATA _stMotorSpeedEst;
UU32 SEG_IDATA _stIntegralKp;
UU32 SEG_IDATA _stIntegralKi;
UU32 SEG_IDATA _stPiSum;
UU16 SEG_IDATA _stDetaThetaCount;
```

```
72  UU16 SEG_IDATA _stSpeedEstGain;
73  UU32 SEG_IDATA _stFilterDetaThetaAverage;
74  U8   SEG_IDATA _bMotorRunState;
75  bit  SEG_IDATA _btSpeedLoopUpdate;
76  U8   SEG_IDATA _bTranstoAMCState;
77  UU16 SEG_IDATA _bKslideTable;
78  UU16 SEG_IDATA _bKslfTable;
79  U8   SEG_XDATA _bUL60730Message;
80  U8   SEG_XDATA _bFaultState;
81  U8   SEG_XDATA _bRampUpDetaTheta;
82  U16  SEG_XDATA _wRampUpTheta;
83  U8   SEG_IDATA _bVqLoopState;
84  U8   SEG_IDATA _bTurnning_address;
85  bit            _btUART_Updata;
86  U8   SEG_XDATA _bOCS_CNT;
87  U8   SEG_XDATA _bVBusHiSet;
88  U8   SEG_XDATA _bVBusLoSet;
89  U16  SEG_IDATA _wSpeedLoopDelayTime;
90  U16  SEG_IDATA _wReverseFaultcnt;
91  UU16 SEG_IDATA _wSpeedCmdOut;
92  U16  SEG_IDATA _wRampUpCmdSpeed;
93  U8   SEG_IDATA _bRampUpSmoDetaTheta;
94
95
96
97
98  //-------------------------------------------------------------------------------------------------------------------------------------------
99  void IncrThetatoAMC(void)
100 {
101    _bRampUpDetaTheta = ((unsigned long)_wRampUpCmdSpeed*_wRampUp_SlopeGain)>>15;
102
103    if(_btMotor_Direction == 0) _wRampUpTheta += _bRampUpDetaTheta;
104    else                        _wRampUpTheta -= _bRampUpDetaTheta;
105
106    _bBootThetaSet = _wRampUpTheta>>2;                                    //bDetaTheta; 0~1023 => 0~255
107    bWriteCmdToAMC_EXT(CMD_BOOTCYCLEPARAMETER, _bBootVqSet, _bBootThetaSet,0);
108 }
109 //-------------------------------------------------------------------------------------------------------------------------------------------
110 void OutputThetatoDAC(void)
111 {
112    U8 bDACDataByte;
113
114    READ_MSFR(MSFR_DUTYB, bDACDataByte);                                  // get SMO theta from AMC
115
116
117    if(MRX0 & 0x80)
118    {
119       WRITE_MSFR(MSFR_DAC3, _bUL60730Message);                           // output UL60730 Message to DAC
120    }
121    else
122    {
123       WRITE_MSFR(MSFR_DAC3, bDACDataByte);                               // output SMO theta to DAC
124    }
125
126 }
127 //-------------------------------------------------------------------------------------------------------------------------------------------
128 void Delay_11CYC()                                                       // about 12/30 MHz * 11 cycle = 4.4 µs
129 {
130    _nop_ ();
131    _nop_ ();
132    _nop_ ();
133    _nop_ ();
134    _nop_ ();
135    _nop_ ();
136    _nop_ ();
137 }
138 //-------------------------------------------------------------------------------------------------------------------------------------------
139
140 U8 bWriteCmdToAMC_EXT(U8 bCommand, U8 bData_HB, U8 bData_LB, U8 bData_LB1)
141 {
142    U16 wWaitTime;
143    SEG_BIT btAMCStandby_Flag;
144
145    btAMCStandby_Flag = 0;
146    for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)                      // Check AMC calculating or Processing Command
```

```
147         if((MRX0 & (AMC_PROCESSING_CMD | AMC_CALCULATING)) == 0x0)
148         {
149             btAMCStandby_Flag = 1;
150             break;
151         }
152     if(!btAMCStandby_Flag)                                          // Check Time-Out
153         return(AMC_BUSY_TIMEOUT);
154
155     MTX0 = bCommand | MCU_MAILBOX_INTR_STOP;
156     MTX1 = bData_HB;
157     MTX2 = bData_LB;
158     MTX3 = bData_LB1;
159     MTX0 = bCommand & MCU_MAILBOX_INTR_START;
160
161     Delay10us(1);
162
163     for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
164         if((MRX0 & AMC_PROCESSING_CMD) == 0)
165             return(AMC_COMPLETED);
166
167     return(AMC_WRITE_CMD_ERROR);
168 }
169 //-----------------------------------------------------------------------------------------------------------------------------
170 void SetMotorRun(void)
171 {
172     U8   xdata bMotorRunDataByte;
173     U8   xdata bTempDataByte;
174     U16  xdata wTotalTimeoutCount;
175     static U16  xdata wAlignmentTimeCount;
176     static U16  xdata wSyncDelyTimeCount;
177     static U16  xdata wTimoutCountStartup;
178     static U8   xdata bRampUpcnt;
179     static U8   xdata bAScnt;
180     static U16  xdata _wSpeedLoopDelayTimeCNT;
181
182     _stAlignmentTimeCountSet.U16 = _wAlignment_Time;                 // Alignment time set
183
184     //state 20~24 is stop and init
185     if(_bMotorRunState == 20)                                       // stop motor
186     {
187         READ_MSFR(MSFR_MCNTL, bMotorRunDataByte);
188         WRITE_MSFR(MSFR_MCNTL, (bMotorRunDataByte & 0xFE));          // disable PWM output
189         _btSpeedCloseFlag = 0;
190         _btSpeedLoopUpdate = 0;
191
192         _bAMCcontrol &= IQLOOP_BIT_CLR;                             // disable Iq loop
193         _bAMCcontrol &= LOSESTEP_BIT_CLR;                           // disable lose step chek
194         _bAMCcontrol &= THETA_SELECT_BIT_CLR;                       // select theta from MCU
195         _bAMCcontrol &= MOTOR_RUN_BIT_CLR;                          // stop motor
196         _bAMCcontrol &= FLUXWEAKENING_BIT_CLR;                      // disable fluxweaking
197
198         if(!bWriteCmdToAMC_EXT(CMD_AMC_CONTROL, 0, _bAMCcontrol,0))
199         {
200             P0 = 0;
201             P0_CFG = 0x00;                                          //set pwm mode
202             _bMotorRunState = 21;
203         }
204     }
205     else if(_bMotorRunState == 21)
206     {
207         _bBootVqSet = _bAlignment_Vq;
208         _bBootThetaSet = _bAlignment_Theta;
209         if(!bWriteCmdToAMC_EXT(CMD_BOOTCYCLEPARAMETER, _bBootVqSet, _bBootThetaSet,0))
210         _bMotorRunState = 22;
211     }
212     else if(_bMotorRunState == 22)
213     {
214         _stTemp16.U16 = _wSMO_Kslide;
215         _bKslideTable.U16 = _wSMO_Kslide;
216         if(!bWriteCmdToAMC_EXT(CMD_SMO_KSLIDE, _stTemp16.U8[MSB], _stTemp16.U8[LSB],0))
217         _bMotorRunState = 23;
218     }
219     else if(_bMotorRunState == 23)
220     {
221         _bKslfTable.U16 = _wSMO_Kslf;
```

```
222         _stTemp16.U16 = _wSMO_Kslf;
223         if(!bWriteCmdToAMC_EXT(CMD_SMO_KSLF, _stTemp16.U8[MSB], _stTemp16.U8[LSB],0))
224         _bMotorRunState = 24;
225     }
226     else if(_bMotorRunState == 24)
227     {
228         _bMotorRunState = 25;
229     }
230     else if(_bMotorRunState == 25)
231     {
232         _stTemp16.U16 = _wIq_Init;
233         if(!bWriteCmdToAMC_EXT(CMD_IQCOMMAND,_stTemp16.U8[MSB],_stTemp16.U8[LSB],0))
234         _bMotorRunState = 26;
235
236     }
237     else if(_bMotorRunState == 26)
238     {
239         //if(btWriteCmdToAMC_EXT(CMD_IqIntegralSet,0,0,0))
240             _bMotorRunState = 27;
241     }
242     else if(_bMotorRunState == 27)
243     {
244         //if(btWriteCmdToAMC_EXT(CMD_IdIntegralSet,0,0,0))
245             _bMotorRunState = 28;
246     }
247     else if(_bMotorRunState == 28)
248     {
249         _stTemp16.S8[LSB] = _cAS;
250         _stTemp16.S8[MSB] = _cAS_SMO_out;
251         if(!bWriteCmdToAMC_EXT(CMD_AS, _stTemp16.S8[MSB], _stTemp16.S8[LSB],0))
252         _bMotorRunState = 88;       // stop motor end state
253     }
254
255     //state 1~10 is ramp up
256     else if(_bMotorRunState == 1)
257     {
258         if      (_bOperation_Mode == 0) {_btIqEnable = 0; _btSpeed_Enable = 0;}          //Vq loop
259         else if (_bOperation_Mode == 1) {_btIqEnable = 1; _btSpeed_Enable = 0;}          //Iq loop
260         else if (_bOperation_Mode == 2) {_btIqEnable = 0; _btSpeed_Enable = 1;}           //Vq loop + Speed loop
261         else if (_bOperation_Mode == 3) {_btIqEnable = 1; _btSpeed_Enable = 1;}          //Iq loop + Speed loop
262
263         if(!_btIqEnable) _bVqLoopState = 1;                                              //Vq Loop
264         else             _bVqLoopState = 0;                                              //Iq Loop
265
266
267         P0_CFG = 0x00;                                                                   //set pwm mode
268
269         if(_btMotor_Direction) //ccw
270             _cAS_SMO_out        = _cAS_SMO - _cLead_angle;
271         else                //cw
272             _cAS_SMO_out        = _cAS_SMO + _cLead_angle;
273
274         _bOCS_CNT = 0;
275         _bVq_Command_Buf = 0;
276         _wRampUpTheta = (unsigned int)_bAlignment_Theta<<2;                              //set initial Alignment theta
277         _wReverseFaultcnt = 0;                                                           //clear ReverseFaultcnt
278         _wRampUpCmdSpeed = _wAlignment_Speed;
279         _bKslfTable.U16 = _wSMO_Kslf;
280                                     _bKslideTable.U16 = _wSMO_Kslide;
281         _stIntegralKi.S32 = (long)_wIq_Init<<16;                                         //set initial PI integrator
282
283         wTotalTimeoutCount = _wRampUp_TimeOut + _wAlignment_Time + _wSpeedLoopDelayTime;   // Time out set
284
285         _wSpeedLoopDelayTimeCNT = 0;
286         bAScnt = 0;
287         wTimoutCountStartup = 0;
288         wAlignmentTimeCount = 0;                                                         // clear alignment time counter
289         wSyncDelyTimeCount = 0;                                                          // clear motor sync. time counter
290         _bFaultState = 0;
291
292         bWriteCmdToAMC_EXT(CMD_ID_IQ_KI, _bIq_Ki_init, _bId_Ki,0);
293         READ_MSFR(MSFR_MCNTL, bMotorRunDataByte);
294         if((bMotorRunDataByte & 0x01) == 0)                                              // check motor running?
295         {
296             READ_MSFR(MSFR_OCSTA, bMotorRunDataByte);                                    // clear OC status
```

```
297
298             _stTemp16.U16 = _wIq_Init;
299             if(!bWriteCmdToAMC_EXT(CMD_IQCOMMAND,_stTemp16.U8[MSB],_stTemp16.U8[LSB],0))
300                 _bMotorRunState = 2;
301
302         }
303     }
304     else if(_bMotorRunState == 2)
305     {
306         _bAMCcontrol &= IQLOOP_BIT_CLR;                                     // disable Iq loop
307         _bAMCcontrol &= LOSESTEP_BIT_CLR;                                   // disable lose step chek
308         _bAMCcontrol &= THETA_SELECT_BIT_CLR;                               // select theta from MCU
309         _bAMCcontrol &= MOTOR_RUN_BIT_CLR;                                  // stop motor
310         _bAMCcontrol &= FLUXWEAKENING_BIT_CLR;                              // disable fluxweaking
311
312
313         if(!bWriteCmdToAMC_EXT(CMD_AMC_CONTROL, 0, _bAMCcontrol,0))
314         {
315             READ_MSFR(MSFR_MCNTL, bMotorRunDataByte);
316             WRITE_MSFR(MSFR_MCNTL, (bMotorRunDataByte | 0x01));             // enable PWM output
317             _bMotorRunState = 3;
318         }
319     }
320     else if(_bMotorRunState == 3)
321     {
322   bWriteCmdToAMC_EXT(CMD_IQINTEGRALSET,0,0,0);
323         bWriteCmdToAMC_EXT(CMD_IDINTEGRALSET,0,0,0);
324         // Iq alignment start
325     _bAMCcontrol |= MOTOR_RUN_BIT;                          // AMC motor run command
326         _bAMCcontrol |= IQ_KP_EN_BIT;
327         _bAMCcontrol |= IQLOOP_BIT;                                         // enable Iq loop
328         if(!bWriteCmdToAMC_EXT(CMD_AMC_CONTROL, 0, _bAMCcontrol,0))
329         {
330             _bMotorRunState = 4;
331         }
332     }
333     //Alignment state
334     else if(_bMotorRunState == 4)
335     {
336         READ_MSFR(MSFR_ECH, bMotorRunDataByte);
337         if ( (bMotorRunDataByte & 0x01) == 0x01 )
338         {
339             IncrThetatoAMC();
340             wAlignmentTimeCount++;                                  // Alignment time counter
341             if ( wAlignmentTimeCount > _stAlignmentTimeCountSet.U16 )       // Alignment mode end? (AlignmentTimeCount * 1mS)
342             {
343                 _bMotorRunState = 5;
344             }
345         }
346     }
347     //ramp up state
348     else if(_bMotorRunState == 5)
349     {
350         _bFaultState = 0;                                                   // clear bFaultState
351         IncrThetatoAMC();
352         if(_wRampUpCmdSpeed >= _wSyncSpeed)
353         {
354             _bMotorRunState = 6;
355         }
356         else
357         {
358             if(++bRampUpcnt >= 20)                                          // add (x)rpm per 20 msec
359             {
360                 _wRampUpCmdSpeed += _bRampUp_Slope;
361                 bRampUpcnt = 0;
362             }
363         }
364     }
365     //SyncDelay state
366     else if ( _bMotorRunState == 6 )
367     {
368         wSyncDelyTimeCount++;
369         if ( wSyncDelyTimeCount > _wSyncDelyTime )
370         {
371             _bAMCcontrol |= LOSESTEP_BIT;                                   // enable lose step check
```

```
372         if ( !bWriteCmdToAMC_EXT(CMD_AMC_CONTROL, 0, _bAMCcontrol,0) )
373         {
374             _bMotorRunState = 7;
375         }
376     }
377     IncrThetatoAMC();
378 }
379 //sync. start and switch to smo mode state
380 else if ( _bMotorRunState == 7 )
381 {
382     IncrThetatoAMC();
383
384     READ_MSFR(MSFR_DUTYB, bMotorRunDataByte);                                   // get SMO theta
385
386     _bRampUpSmoDetaTheta = 0;
387     if ( (_bBootThetaSet <= bMotorRunDataByte) && (_btMotor_Direction == 0))        //cw ,_bRampUpSmoDetaTheta>0
388     {
389         _bRampUpSmoDetaTheta = bMotorRunDataByte - _bBootThetaSet;
390                                     }
391     else if ( (_bBootThetaSet >= bMotorRunDataByte) && (_btMotor_Direction == 1))   //ccw ,_bRampUpSmoDetaTheta<0
392     {
393         _bRampUpSmoDetaTheta = _bBootThetaSet - bMotorRunDataByte;
394                                     }
395
396     //-DetaTheta;
397     if( _bRampUpSmoDetaTheta <= 64 && _bRampUpSmoDetaTheta >= 1  )
398     {
399         if(_btMotor_Direction == 0)
400             _stTemp16.S8[LSB] = -_bRampUpSmoDetaTheta ;
401         else if(_btMotor_Direction == 1) _stTemp16.S8[LSB] = _bRampUpSmoDetaTheta ;
402
403         _stTemp16.S8[MSB] = _cAS_SMO_out;
404         bWriteCmdToAMC_EXT(CMD_AS, _stTemp16.S8[MSB], _stTemp16.S8[LSB],0);
405
406         _bAMCcontrol |= THETA_SELECT_BIT;                       //theta slect:AMC
407         if(!bWriteCmdToAMC_EXT(CMD_AMC_CONTROL, 0, _bAMCcontrol,0))
408         _bMotorRunState = 8;
409     }
410 }
411 //sync. AS reduce to zero state
412 else if(_bMotorRunState == 8)
413 {
414     if(bAScnt++ >= _bAScntSet)
415     {
416         bAScnt = 0;
417         if(_bRampUpSmoDetaTheta > 0) _bRampUpSmoDetaTheta--;
418
419         if(_btMotor_Direction == 0)
420             _stTemp16.S8[LSB] = -_bRampUpSmoDetaTheta ;
421         else if(_btMotor_Direction == 1) _stTemp16.S8[LSB] = _bRampUpSmoDetaTheta ;
422
423
424         _stTemp16.S8[MSB] = _cAS_SMO_out;
425         if(!bWriteCmdToAMC_EXT(CMD_AS, _stTemp16.S8[MSB], _stTemp16.S8[LSB],0))
426         {
427             if(_bRampUpSmoDetaTheta == 0) _bMotorRunState = 9;
428         }
429     }
430 }
431 //sync. end and ramp up end state
432 else if ( _bMotorRunState == 9 )
433 {
434     if(_wSpeedLoopDelayTimeCNT++ >= _wSpeedLoopDelayTime)
435     {
436         if(_btSpeed_Enable)
437         {
438             _btSpeedCloseFlag = 1;                          // enable Speed loop
439             _wSpeedCmdOut.U16 = _wSMO_Entry_Speed;
440         }
441         _bMotorRunState = 10;
442
443         bWriteCmdToAMC_EXT(CMD_ID_IQ_KI, _bIq_Ki_Run, _bId_Ki,0);
444
445         _bAMCcontrol |= FLUXWEAKENING_BIT;
446         bWriteCmdToAMC_EXT(CMD_AMC_CONTROL, 0, _bAMCcontrol,0);
```

```
447
448         }
449     }
450
451
452     if(_bMotorRunState <= 9) wTimoutCountStartup++;                                  // Ramp up timeoutcnt++
453
454     if ( wTimoutCountStartup > wTotalTimeoutCount && _bMotorRunState <= 9 && _bMotorRunState >= 1)//Ramp up time out?
455     {
456         READ_MSFR(MSFR_MCNTL, bTempDataByte);
457         WRITE_MSFR(MSFR_MCNTL, bTempDataByte & 0xFE);                                // stop motor
458         if(_bFaultState == 0)   _bFaultState = TIMEOUTFAULTDISP;
459     }
460
461 }
462 //-------------------------------------------------------------------------------------------------------------------------------------------
463 void MotorSpeedEstimate(void)
464 {
465     static U8   idata bThetaVariance;
466
467     READ_MSFR(MSFR_DUTYB, _bEstTheta);                                               // get SMO theta
468
469     if(_btMotor_Direction == 0)
470     {
471         if ( _bEstTheta >= _bOld_EstTheta )                                          // cw, get delta theta
472             _stDetaThetaCount.U16  = _bEstTheta - _bOld_EstTheta;
473         else _stDetaThetaCount.U16 = (256-_bOld_EstTheta) + _bEstTheta;
474
475         if(_stDetaThetaCount.U16 >= 185) _stDetaThetaCount.U16 = bThetaVariance;    // threshold 360 Hz
476         bThetaVariance = _stDetaThetaCount.U16;
477     }
478     else if(_btMotor_Direction == 1)
479     {
480         if ( _bOld_EstTheta >= _bEstTheta )                                          // ccw, get delta theta
481             _stDetaThetaCount.U16  = _bOld_EstTheta - _bEstTheta;
482         else _stDetaThetaCount.U16 = (256-_bEstTheta) + _bOld_EstTheta;
483
484         if(_stDetaThetaCount.U16 >= 185) _stDetaThetaCount.U16 = bThetaVariance;    // threshold 360 Hz
485         bThetaVariance = _stDetaThetaCount.U16;
486     }
487     _bOld_EstTheta = _bEstTheta;
488
489     //_wMotorSpeedEst.U16 = ((unsigned long)_wDetaThetaCount.U16 * _wSpeedEstGain.U16)>>8; //need 12 µs
490     //MDU 12 µs=> 2 µs
491     MD0 = _stSpeedEstGain.U8[LSB];
492     MD4 = _stDetaThetaCount.U8[LSB];
493     MD1 = _stSpeedEstGain.U8[MSB];
494     MD5 = _stDetaThetaCount.U8[MSB];
495     Delay_11CYC();
496
497     _stMotorSpeedEst.U8[LSB] = MD1;
498     _stMotorSpeedEst.U8[MSB] = MD2;
499
500
501
502     MD0 = _stFilterDetaThetaAverage.U8[b0];
503     MD4 = 15;
504     MD1 = _stFilterDetaThetaAverage.U8[b1];
505     MD5 = 0;
506     Delay_11CYC();
507
508     _stFilterDetaThetaAverage.U8[b0] = MD0;
509     _stFilterDetaThetaAverage.U8[b1] = MD1;
510     _stFilterDetaThetaAverage.U8[b2] = MD2;
511     _stFilterDetaThetaAverage.U8[b3] = MD3;
512
513
514     _stFilterDetaThetaAverage.U32 += _stMotorSpeedEst.U16;
515
516     //_lFilterDetaThetaAverage.U32 >>= 4;                                            // Average times are 16
517     //MDU 5.6us=> 2.4 µs
518     MD0 = _stFilterDetaThetaAverage.U8[b0];
519     MD1 = _stFilterDetaThetaAverage.U8[b1];
520     MD2 = _stFilterDetaThetaAverage.U8[b2];
521     MD3 = _stFilterDetaThetaAverage.U8[b3];
```

```
522     ARCON = 0x24;
523     _nop_ ();
524     _nop_ ();
525     _nop_ ();
526     _nop_ ();
527     _stFilterDetaThetaAverage.U8[b0] = MD0;
528     _stFilterDetaThetaAverage.U8[b1] = MD1;
529     //_lFilterDetaThetaAverage.U8[b2] = MD2;
530     //_lFilterDetaThetaAverage.U8[b3] = MD3;
531
532     _stMotorSpeedEst.U16 = _stFilterDetaThetaAverage.U32;                    //get Wr
533
534 }
535 //-----------------------------------------------------------------------------------------------------------------------
536 void Speed_PI_Calculate(void)
537 {
538     UU16 idata wSpeedErr;
539     static U8 bSpeedCnt = 0;
540
541     if(_stSpeedCmd.U16 < 300)                    _stSpeedCmd.U16 = 300;
542     else if(_stSpeedCmd.U16 > 1500)  _stSpeedCmd.U16 = 1500;
543
544     if(++bSpeedCnt >= 10)//20 ms
545     {
546         bSpeedCnt = 0;
547         if                          ( _wSpeedCmdOut.S16 > _stSpeedCmd.S16 )  _wSpeedCmdOut.S16 -= _bSpeed_Slope;
548         else if ( _wSpeedCmdOut.S16 < _stSpeedCmd.S16 )  _wSpeedCmdOut.S16 += _bSpeed_Slope;
549     }
550
551     SmoParaTable();                                                          //get kslid & kslf
552
553     wSpeedErr.S16 = _wSpeedCmdOut.S16 - _stMotorSpeedEst.U16;
554     // speed close loop for Iq command or Vq command
555     _stIntegralKi.S32 += (long)wSpeedErr.S16 * _wSpeed_Ki;
556
557     if(_stIntegralKi.S32 >= _dwSpeed_OutputLimit_maxQ16)        _stIntegralKi.S32 = _dwSpeed_OutputLimit_maxQ16;
558     else if(_stIntegralKi.S32 <= _dwSpeed_OutputLimit_minQ16)   _stIntegralKi.S32 = _dwSpeed_OutputLimit_minQ16;
559
560     _stIntegralKp.S32 = (long)wSpeedErr.S16 * _wSpeed_Kp;
561
562     _stPiSum.S32 = _stIntegralKp.S32 + _stIntegralKi.S32;
563
564     if(_stPiSum.S32 >= _dwSpeed_OutputLimit_maxQ16)             _stPiSum.S32 = _dwSpeed_OutputLimit_maxQ16;
565     else if(_stPiSum.S32 <= _dwSpeed_OutputLimit_minQ16)        _stPiSum.S32 = _dwSpeed_OutputLimit_minQ16;
566
567     _btSpeedLoopUpdate = 1;                     // set AMC update flag
568 }
569 //-----------------------------------------------------------------------------------------------------------------------
570 void SmoParaTable(void)
571 {
572
573     //if(_wSpeedCmdOut.S16 >= 350) _bKslfTable.U16 = (5*_wSpeedCmdOut.S16 + 0);
574     //else _bKslfTable.U16 = 500;
575 }
576 //-----------------------------------------------------------------------------------------------------------------------
577 void UL60730_FaultMessage(void)
578 {
579     if((MRX0 & 0x80) && !(MRX0 & 0x10))
580     {
581         READ_MSFR(MSFR_ECL, _bUL60730Message);                               // get ECL register
582         if(_bFaultState == 0)   _bFaultState = UL60730FAULTDISP;
583     }
584     else _bUL60730Message = 0;
585 }
586 //-----------------------------------------------------------------------------------------------------------------------
587 void ReverseFault(void)
588 {
589     U8 bEstTheta_REV;
590     U8 bData;
591     static U8 bOldEstTheta_REV;
592
593     READ_MSFR(MSFR_DUTYB, bEstTheta_REV);                                    // get SMO theta
594
595     if(_btMotor_Direction == 0)                                              //CW, bEstTheta_REV > bOldEstTheta_REV
596     {
```

```
597         if(bEstTheta_REV < bOldEstTheta_REV)                    //err state
598         {
599             _wReverseFaultcnt++;
600         }
601         else if(bEstTheta_REV > bOldEstTheta_REV)               //normal state
602         {
603             if(_wReverseFaultcnt > 1) _wReverseFaultcnt--;
604         }
605     }
606     else if(_btMotor_Direction == 1)                             //CW, bEstTheta_REV < bOldEstTheta_REV
607     {
608         if(bEstTheta_REV > bOldEstTheta_REV)                    //err state
609         {
610             _wReverseFaultcnt++;
611         }
612         else if(bEstTheta_REV < bOldEstTheta_REV)               //normal state
613         {
614             if(_wReverseFaultcnt > 1) _wReverseFaultcnt--;
615         }
616     }
617
618     bOldEstTheta_REV = bEstTheta_REV;
619
620     if(_wReverseFaultcnt > 400)
621     {
622         _wReverseFaultcnt = 500;
623
624         READ_MSFR(MSFR_MCNTL, bData);
625         bData &= 0xFE;
626         WRITE_MSFR(MSFR_MCNTL, bData);
627
628         if(_bFaultState == 0)   _bFaultState = REVERSEFAULTDISP;
629
630     }
631 }
632 //-------------------------------------------------------------------------------------------------------------------------------
633 void Fault_Led_Flash()
634 {
635     while(1)
636     {
637         EA = 0;
638         P2_6 = 0;
639         Delay1 ms(500);
640         P2_6 = 1;
641         Delay1 ms(500);
642     }
643 }
644 //-------------------------------------------------------------------------------------------------------------------------------
```

固件示例

在 FCM8531 中，AMC 和 MCU 并联运行。一旦 MCU 完成 AMC 的初始化进程，MCU 就开始发送指令，启动电机并控制转速。

示例程序可在 MCDS IDE 安装文件夹下面找到，通过邮箱将参数如 VqCommand、IqCommand、AMC_Control 写入 AMC，以实现电机启动和速度循环。示例代码包括主程序、Timer0 中断程序、和 AMC 同步中断程序，下面进行详细介绍。

Main

图 34 是主程序的流程图。

主程序包括两部分：初始化和无限控制循环（在 (1) 期间 {}）。

所有变量如 PWM 频率、保护点和中断相关参数都被初始化。AMC 启动参数在初始化流程中被清除、重置和传输。

Watchdog 函数和输出 theta 至 DAC 函数无限控制循环执行。

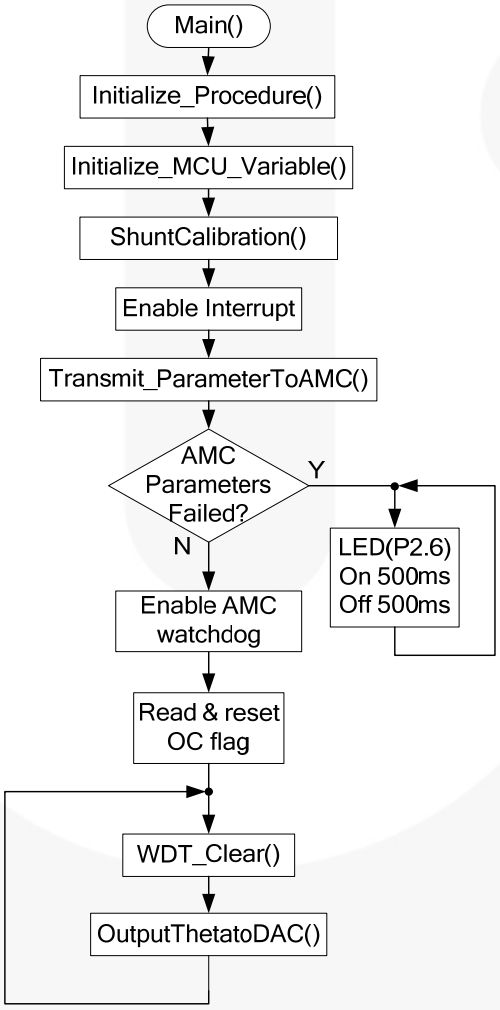


图 34. 主程序流程图

Timer0

在 FCM8531 评估板的示例代码中，硬件开关和电路板保护功能的状态由从 Timer0 开始的 25 ms 计时器询问。Timer0 中断的流程图如图 35 所示。

扳钮开关 1，即 RUN/FREE 和扳钮开关 2，即 CW/CCW 的状态记录为变量，被其它程序引用。

同时还询问各种保护机制，如 OVP 和 UVP。PWM 输出在 Check_OVP() 中被禁用并且 FREE 指令被传输至 AMC，用于终止滑模运行。故障状态，包括来自邮箱的 AMC 故障，由 FaultDisp() 处理和显示。红色 LED 灯以各种频率闪烁，表示不同的故障状态。

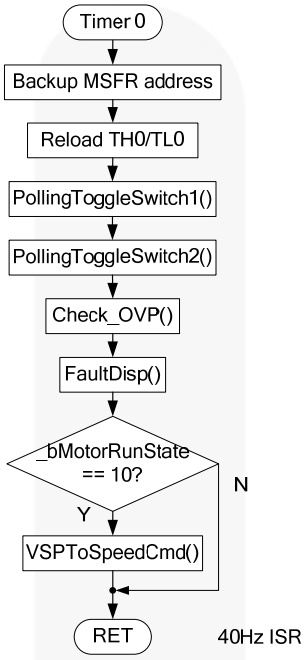


图 35. Timer0 中断流程图

Timer 0 示例程序代码

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
#define INITIAL_T0_INTERVAL0x0BDC
                                    //16bit mode => 40 Hz
INTERRUPT(ISR_T0, VECTOR_ET0)
{
   U8 bBackupADR;
   UU16 V;
   //User variable start here.(39)
   //User variable end here.(39)
   bBackupADR = MSFRADR;
   //Initial Timer0
   V.U16 = INITIAL_T0_INTERVAL;
   TH0 = V.U8[MSB];
   TL0 = V.U8[LSB];
   //User program start here.(13)
   PollingToggleSwitch1();                  //PollingToggleSwitch1
   PollingToggleSwitch2();                  //PollingToggleSwitch2
   Check_OVP();                        //check vbus
   FaultDisp();                        //check MCU, AMC Fault
   if(_bMotorRunState == 10)           //sliding mode
   {
      VSPToSpeedCmd();
    }
   //User program end here.(13)
   MSFRADR = bBackupADR;
}
```

AMC 同步中断

AMC 中的滑模运行频率为 8 kHz，电流控制环路也是如此。AMC 以相同的频率中断 MCU，并自动提供估算转子角度（SFR 地址 0x07）。

AMC 同步中断程序将中断频率降低为 1 Khz 以满足电机启动的需要，降低为 500 Hz 以满足速度闭环控制环路的需要。

图 36 是 AMC 同步中断流程图。

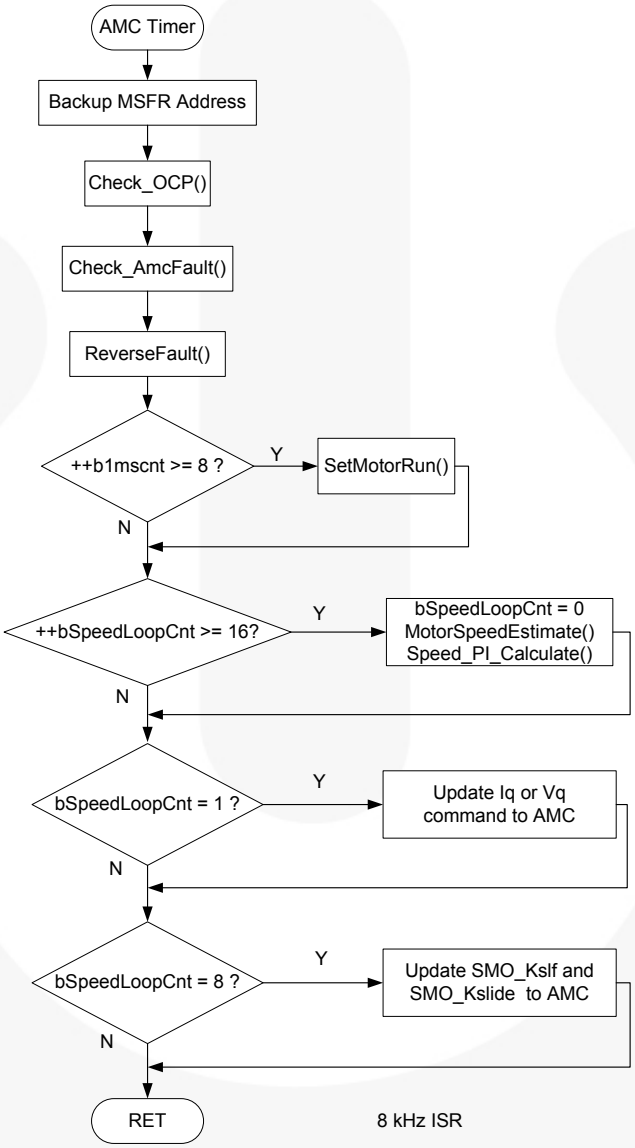


图 36. AMC 同步中断流程图

下面列出 AMC 同步中断的示例程序代码。

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
INTERRUPT(ISR_AMC, VECTOR_EX11)
{
   U8 bBackupADR;

   //User variable start here.(45)
                                 static U8 bSpeedLoopCnt = 0;
   static U8 b1mscnt = 0;
   //User variable end here.(45)
   bBackupADR = MSFRADR;

   //User program start here.(46)

   Check_OCP();                          // check OC status
   Check_AmcFault();                     // check AMC error status

   if(_bMotorRunState == 10)                // sliding mode
      ReverseFault();                    // check Reverse Fault

   if(++b1mscnt >= 8)                    // 1 kHz loop
   {
      b1mscnt = 0;
      SetMotorRun();
   }

   //speed loop
   if(++bSpeedLoopCnt >= 16)                     //499.1 Hz loop
   {
      bSpeedLoopCnt = 0;
      MotorSpeedEstimate();
      if(_btSpeedCloseFlag)  Speed_PI_Calculate();
   }

   //AMC parameter update, bSpeedLoopCnt = 0~15
   if(bSpeedLoopCnt == 1)
   {
      if(_btSpeedLoopUpdate == 1)              //speed loop cmd update flag
      {
         if(_bVqLoopState == 200)              //Vq loop
         {
            bWriteCmdToAMC_EXT(CMD_VQCOMMAND,_stPiSum.U8[b2],_stSMO_G.U8[MSB],_stSMO_G.U8[LSB]);

         }
         else                                  //Iq loop
         {
            if(_stPiSum.U8[b3] == 0)
            {
               bWriteCmdToAMC_EXT(CMD_IQCOMMAND_L,_stPiSum.U8[b2],_stSMO_G.U8[MSB],_stSMO_G.U8[LSB]);
            }
            else
            {
               bWriteCmdToAMC_EXT(CMD_IQCOMMAND_H,_stPiSum.U8[b2],_stSMO_G.U8[MSB],_stSMO_G.U8[LSB]);
            }
         }
         _btSpeedLoopUpdate = 0;
      }

      // CMD selection (VSP or Tuning UI)
      if( _btSpeed_Enable == 0 && _bMotorRunState == 10)
      {
         if(_bVqLoopState == 0)                //Iq loop
         {
            if(_stIq_Command.U16 >= _wIq_Command_max)      _stIq_Command.U16 = _wIq_Command_max;
            else if(_stIq_Command.U16 <= _wIq_Command_min) _stIq_Command.U16 = _wIq_Command_min;

            if(_stIq_Command.U16 <= 255)
            {
               bWriteCmdToAMC_EXT(CMD_IQCOMMAND_L,_stIq_Command.U8[LSB],_stSMO_G.U8[MSB],_stSMO_G.U8[LSB]);
            }
```

```
            else
            {
               bWriteCmdToAMC_EXT(CMD_IQCOMMAND_H,_stIq_Command.U8[LSB],_stSMO_G.U8[MSB],_stSMO_G.U8[LSB]);
            }
         }
         else if(_bVqLoopState == 200)            //Vq loop
         {
            if(_bVq_Command_Buf >= _bVq_Command_max)     _bVq_Command_Buf = _bVq_Command_max;
            else if(_bVq_Command_Buf <= _bVq_Command_min) _bVq_Command_Buf = _bVq_Command_min;

            if                              ( _bVq_Command > _bVq_Command_Buf )  _bVq_Command -= 1;
            else if ( _bVq_Command < _bVq_Command_Buf )  _bVq_Command += 1;

            bWriteCmdToAMC_EXT(CMD_VQCOMMAND,_bVq_Command,_stSMO_G.U8[MSB],_stSMO_G.U8[LSB]);
         }
      }
   }
   else if(bSpeedLoopCnt == 8)
   {
      if(_bMotorRunState == 10)                  //sliding mode
      {
         if(_bTranstoAMCState == 0)
         {
            bWriteCmdToAMC_EXT(CMD_SMO_KSLF, _bKslfTable.U8[MSB], _bKslfTable.U8[LSB],0);
            _bTranstoAMCState = 1;
         }
         else if(_bTranstoAMCState == 1)
         {
            bWriteCmdToAMC_EXT(CMD_SMO_KSLIDE, _bKslideTable.U8[MSB], _bKslideTable.U8[LSB],0);
            _bTranstoAMCState = 0;
         }

         if(_bVqLoopState >= 1 && _bVqLoopState < 100) _bVqLoopState++;
         if(_bVqLoopState == 100)
         {
            _bVqLoopState = 200;
            READ_MSFR(MSFR_DUTYA, _bVq_Command);            // get DUTYA from AMC
            _bVq_Command_Buf = _bVq_Command;
            bWriteCmdToAMC_EXT(CMD_VQCOMMAND,_bVq_Command,_stSMO_G.U8[MSB],_stSMO_G.U8[LSB]);
            _bAMCcontrol &= IQLOOP_BIT_CLR;                 // disable Iq loop
            bWriteCmdToAMC_EXT(CMD_AMC_CONTROL, 0, _bAMCcontrol,0);
            _stIntegralKi.S32 = _bVq_Command<<16;
         }
      }
   }
   //User program end here.(46)

   MSFRADR = bBackupADR;
}
```

速度闭环控制环路

来自 AMC 的估算角度可用于计算转速。

执行周期为 2 ms 或者频率为 500 Hz。360 电角度表示为 0 至 255 估算角度（SFR 地址 0x07）。Rpm 是根据以下方程式得出的估算角度差：

$$\text{Rpm} = [\theta(n) - \theta(n-1)] * Gain/2^8$$

$$Gain = \frac{1}{256}\frac{120}{\text{Motor poles}}f_s * 2^8 \qquad (1)$$

$$f_s = 500\ \text{Hz}$$

θ = SFR 0x07

示例代码中的变量：

$Gain$ = _stSpeedEstGain.U16

$[\theta(n) - \theta(n-1)]$ = _wDetaThetaCount.U16

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
void MotorSpeedEstimate(void)
{
   static U8   idata bThetaVariance;

   READ_MSFR(MSFR_DUTYB, _bEstTheta);                              // get SMO theta

   if(_btMotor_Direction == 0)
   {
      if ( _bEstTheta >= _bOld_EstTheta )                          // cw, get delta theta
         _stDetaThetaCount.U16  = _bEstTheta - _bOld_EstTheta;
      else _stDetaThetaCount.U16 = (256-_bOld_EstTheta) + _bEstTheta;

      if(_stDetaThetaCount.U16 >= 185) _stDetaThetaCount.U16 = bThetaVariance; // threshold 360 Hz
      bThetaVariance = _stDetaThetaCount.U16;
   }
   else if(_btMotor_Direction == 1)
   {
      if ( _bOld_EstTheta >= _bEstTheta )                          // ccw, get delta theta
         _stDetaThetaCount.U16  = _bOld_EstTheta - _bEstTheta;
      else _stDetaThetaCount.U16 = (256-_bEstTheta) + _bOld_EstTheta;

      if(_stDetaThetaCount.U16 >= 185) _stDetaThetaCount.U16 = bThetaVariance; // threshold 360 Hz
      bThetaVariance = _stDetaThetaCount.U16;
   }
   _bOld_EstTheta = _bEstTheta;

   //_wMotorSpeedEst.U16 = ((unsigned long)_wDetaThetaCount.U16 * _wSpeedEstGain.U16)>>8;
   MD0 = _stSpeedEstGain.U8[LSB];
   MD4 = _stDetaThetaCount.U8[LSB];
   MD1 = _stSpeedEstGain.U8[MSB];
   MD5 = _stDetaThetaCount.U8[MSB];

   Delay_11CYC();

   _stMotorSpeedEst.U8[LSB] = MD1;
   _stMotorSpeedEst.U8[MSB] = MD2;



   MD0 = _stFilterDetaThetaAverage.U8[b0];
   MD4 = 15;
   MD1 = _stFilterDetaThetaAverage.U8[b1];
   MD5 = 0;

   Delay_11CYC();

   _stFilterDetaThetaAverage.U8[b0] = MD0;
   _stFilterDetaThetaAverage.U8[b1] = MD1;
```

```
    _stFilterDetaThetaAverage.U8[b2] = MD2;
    _stFilterDetaThetaAverage.U8[b3] = MD3;


    _stFilterDetaThetaAverage.U32 += _stMotorSpeedEst.U16;

    //_lFilterDetaThetaAverage.U32 >>= 4;                                                    // Average times are 16
    MD0 = _stFilterDetaThetaAverage.U8[b0];
    MD1 = _stFilterDetaThetaAverage.U8[b1];
    MD2 = _stFilterDetaThetaAverage.U8[b2];
    MD3 = _stFilterDetaThetaAverage.U8[b3];
    ARCON = 0x24;
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _nop_ ();
    _stFilterDetaThetaAverage.U8[b0] = MD0;
    _stFilterDetaThetaAverage.U8[b1] = MD1;

    _stMotorSpeedEst.U16 = _stFilterDetaThetaAverage.U32;                    //get Wr

}
//-----------------------------------------------------------
void Speed_PI_Calculate(void)
{
    UU16 idata wSpeedErr;
    static U8 bSpeedCnt = 0;

    if(_stSpeedCmd.U16 < 1500)    _stSpeedCmd.U16 = 1500;
    else if(_stSpeedCmd.U16 > 4500)  _stSpeedCmd.U16 = 4500;

    if(++bSpeedCnt >= 10)//20 ms
    {
        bSpeedCnt = 0;
        if ( _wSpeedCmdOut.S16 > _stSpeedCmd.S16 )  _wSpeedCmdOut.S16 -= _bSpeed_Slope;
        else if ( _wSpeedCmdOut.S16 < _stSpeedCmd.S16 )  _wSpeedCmdOut.S16 += _bSpeed_Slope;
    }

    wSpeedErr.S16 = _wSpeedCmdOut.S16 - _stMotorSpeedEst.U16;
    // speed close loop for Iq command or Vq command
    _stIntegralKi.S32 += (long)wSpeedErr.S16 * _wSpeed_Ki;

    if(_stIntegralKi.S32 >= _dwSpeed_OutputLimit_maxQ16)        _stIntegralKi.S32 = _dwSpeed_OutputLimit_maxQ16;
    else if(_stIntegralKi.S32 <= _dwSpeed_OutputLimit_minQ16)    _stIntegralKi.S32 = _dwSpeed_OutputLimit_minQ16;

    _stIntegralKp.S32 = (long)wSpeedErr.S16 * _wSpeed_Kp;

    _stPiSum.S32 = _stIntegralKp.S32 + _stIntegralKi.S32;

    if(_stPiSum.S32 >= _dwSpeed_OutputLimit_maxQ16)            _stPiSum.S32 = _dwSpeed_OutputLimit_maxQ16;
    else if(_stPiSum.S32 <= _dwSpeed_OutputLimit_minQ16)        _stPiSum.S32 = _dwSpeed_OutputLimit_minQ16;

    _btSpeedLoopUpdate = 1;                    // set AMC update flag
}
```

Check_MotorRunningProcess

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
void Check_RunMotorProcess(void)
{

   if(RunMotorProcessFlag)
   {
      RunMotorProcessFlag = 0;                  // clear RunMotorProcessFlag
      if(MotorRun)                              // run motor startup process
      {
         P2_6 = 1;                                 // turn off status LED
         IEN2 &= 0xFD;                             // disable FCM8531 fault interrupt
         if(SetMotorRun(1))                        // run motor startup & confirm status
         {
            MotorSpeedEstimateInitial();           // speed link
            btWriteCmdToAMC(CMD_AS, 0, 0x50);              // AS setup
            SpeedSyncFlag = 1;
         }
         else StartUpFailFlag = 1;
         IEN2 |= 0x02;                          // enable FCM8531 fault interrupt
      }
      else
      {
         IEN2 &= 0xFD;                          // disable FCM8531 fault
                                                interrupt
         SetMotorRun(0);                        // run stop motor process
         Delay1 ms(3000);                        // delay 3000 mS
         IEN2 |= 0x02;                          // enable FCM8531 fault
                                                interrupt
         btWriteCmdToAMC(CMD_AS, 0, As_init);           // AS setup
         VDD_UV_Flag = 0;
         Short_Flag = 0;
         OC_Flag = 0;
         AMC_Fault_Flag = 0;
         VbusOVP_Flag = 0;
         StartUpFailFlag = 0;
         old_EstTheta = 0;
         P2_6 = 0;                              // turn on status LED

      }
   }
}
```

Speed_PI_Calculate

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
void Speed_PI_Calculate(void)
{
   UU16 idata wSpeedErr;
   static U8 bSpeedCnt = 0;

   #ifndef Tuning_UI_Enable
   if(_stSpeedCmd.U16 < 1500)                    _stSpeedCmd.U16 = 1500;
   else if(_stSpeedCmd.U16 > 4500)  _stSpeedCmd.U16 = 4500;
   #endif

   if(++bSpeedCnt >= 10)//20 ms
   {
      bSpeedCnt = 0;
      if                                    ( _wSpeedCmdOut.S16 > _stSpeedCmd.S16 )  _wSpeedCmdOut.S16 -= _bSpeed_Slope;
      else if ( _wSpeedCmdOut.S16 < _stSpeedCmd.S16 )  _wSpeedCmdOut.S16 += _bSpeed_Slope;
   }

   SmoParaTable(); //get kslid & kslf

   wSpeedErr.S16 = _wSpeedCmdOut.S16 - _stMotorSpeedEst.U16;
   // speed close loop for Iq command or Vq command
   _stIntegralKi.S32 += (long)wSpeedErr.S16 * _wSpeed_Ki;

   if(_stIntegralKi.S32 >= _dwSpeed_OutputLimit_maxQ16)        _stIntegralKi.S32 = _dwSpeed_OutputLimit_maxQ16;
   else if(_stIntegralKi.S32 <= _dwSpeed_OutputLimit_minQ16)   _stIntegralKi.S32 = _dwSpeed_OutputLimit_minQ16;

   _stIntegralKp.S32 = (long)wSpeedErr.S16 * _wSpeed_Kp;

   _stPiSum.S32 = _stIntegralKp.S32 + _stIntegralKi.S32;

   if(_stPiSum.S32 >= _dwSpeed_OutputLimit_maxQ16)             _stPiSum.S32 = _dwSpeed_OutputLimit_maxQ16;
   else if(_stPiSum.S32 <= _dwSpeed_OutputLimit_minQ16)        _stPiSum.S32 = _dwSpeed_OutputLimit_minQ16;

   _btSpeedLoopUpdate = 1;                     // set AMC update flag
}
```

Check_OCP

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
void Check_OCP(void)
{
U8 bTempDataByte;
   READ_MSFR(MSFR_OCSTA, bTempDataByte);                        // get OC status
   bTempDataByte &= 0x3F;

   if ( bTempDataByte != 0 )                                    // check OCAH, OCBH, OCAL, OCBL
   {
      //READ_MSFR(MSFR_MCNTL, bTempDataByte)
      //WRITE_MSFR(MSFR_MCNTL, bTempDataByte & 0xFE)    // stop motor
   }
}
```

Check_OVP

```
/*---------------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
 */
void Check_OVP(void)
```

```
{
   //U8 bGetHiVoltage;
   U8 bTempDataByte;

   READ_MSFR(MSFR_MCNTL, bTempDataByte);                    // get motor control register configuration
   if ( (bTempDataByte & 0x01) == 0x01 )                    // check motor running status
   {                                                        // motor running
      if ( _bVBusLoSet != 0 )
      {
         if ( bGetHiVoltage < _bVBusLoSet )
         {
            if ( !_btVbusUnder )
            {
               WRITE_MSFR(MSFR_MCNTL, bTempDataByte & 0xFE);// stop motor
               _btVbusUnder = 1;                            // set UVP flag
               if(_bFaultState == 0)   _bFaultState = UVPFAULTDISP;
            }
         }
      }
      if ( _bVBusHiSet != 0xFF )
      {
         if ( bGetHiVoltage > _bVBusHiSet )
         {
            if ( !_btVbusOver )
            {
               WRITE_MSFR(MSFR_MCNTL, bTempDataByte & 0xFE); // stop motor
               _btVbusOver = 1;                             // set OVP flag
               if(_bFaultState == 0)   _bFaultState = OVPFAULTDISP;
            }
         }
      }
   }
}
```

Check_AmcFault

```
/*---------------------------------------------------------------------
 * Copyright 2013 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------
 */
void Check_AmcFault(void)
{
   U8 bTempDataByte;

   if(MRX0 & 0x10)                          //check lose Step
   {
      if(_bFaultState == 0)   _bFaultState = LOSSSTEPFAULTDISP;
      READ_MSFR(MSFR_MCNTL, bTempDataByte);
      WRITE_MSFR(MSFR_MCNTL, bTempDataByte & 0xFE);     // stop motor

      #ifdef Tuning_UI_Enable
      if(_bErrorCode == 0) _bErrorCode = 4;
      #endif
   }

   if(MRX0 & 0x08)                          //check Amc Fault
   {
      if(_bFaultState == 0)   _bFaultState = AMCFAULTDISP;
      READ_MSFR(MSFR_MCNTL, bTempDataByte);
      WRITE_MSFR(MSFR_MCNTL, bTempDataByte & 0xFE);     // stop motor

      #ifdef Tuning_UI_Enable
      if(_bErrorCode == 0) _bErrorCode = 1;
      #endif

   }

}
```

附录 A

短路保护 (SCP)

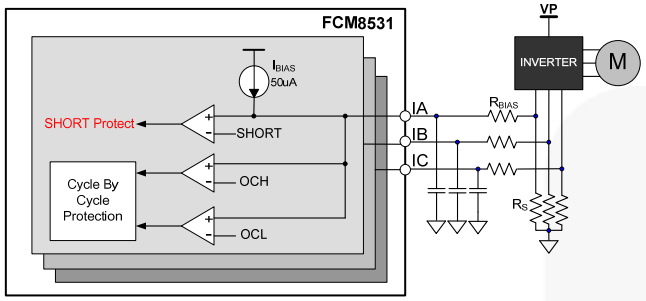


图 37. 电流保护框图

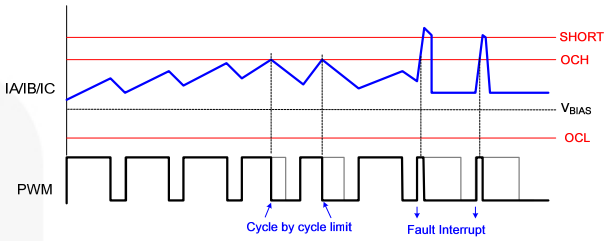


图 38. 电流保护原理示意图

表 6. 短路保护的相关 MSFR

字节名称（地址）	位								Rst
	7	6	5	4	3	2	1	0	
MSTAT (3Fh)	VDD_UV	H_SLOW	SHORT_A	SHORT_B	SHORT_C	H_ERR	S_ACT	DIR	00h
DAC2 (46h)	DAC_SHORT								FFh
IEN2 (9Ah)	保留		EX12	EX11	EX10	EX9	EX8	保留	00h

短路通常是电机工作时最严重的故障，在下列条件下可能发生：

1. 电机内部绕组线圈老化导致铜导线的绝缘层剥落；
2. 电机三相连接线短路；
3. 三相连接线绝缘塑料层老化导致导线裸露，造成短路。

尽管这些情况不常发生，但是只要发生其中任意一种，巨大的短路电流瞬间就会损坏功率驱动组件。FCM8531 短路保护的电流检测点为 IA、IB、IC，如图 37 所示。电流检测点的电压电平与 MSFR SHORT DAC (0x46) 设置的电压作比较。如果电压电平超过设定电压，则会生成故障信号以触发中断 8，这样故障状态在 MSFR MSTAT 中就能读取，功率驱动组件就会得到保护。此外，当电流超过 OCH 或 OCL（参考图 38）时，通过逐周期限流可即时保护 PWM，确保过载电流不会损坏三相功率元件。另外，电流还与 MSFR SHORT 寄存器进行比较，并可编程设置。有关短路保护的详细信息，请参见数据手册。

短路保护示例程序代码如下。

```
/*-------------------------------------------------------------------------
 * Copyright 2011 Fairchild Semiconductor
 * http://www.fairchildsemi.com
*-------------------------------------------------------------------------
 */
#define Initial_Short           0xFF
//-------------------------------------------------------------------------
//  Initial Protection Register
//-------------------------------------------------------------------------
void Initial_Protect()
{
    WRITE_MSFR(MSFR_OCCNTL, INITIAL_OCCNTL);
    WRITE_MSFR(MSFR_OCH, INITIAL_OCH);
    WRITE_MSFR(MSFR_OCL, INITIAL_OCL);
    WRITE_MSFR(MSFR_SHORT, INITIAL_SHORT);
}
//-------------------------------------------------------------------------
```

```
//  Fault Interrupt
//-----------------------------------------------------------------------
INTERRUPT(ISR_Fault, VECTOR_EX8)
{
    U8 bBackupADR;
    U8 bV;
    //User variable start here.(24)

    //User variable end here.(24)

    bBackupADR = MSFRADR;
    READ_MSFR(MSFR_MSTAT, bV);
    if(bV)
    {
        //User program start here.(1B)

        //User program end here.(1B)
        if(bV & 0x20)
            Evt_ShortA();
        if(bV & 0x10)
            Evt_ShortB();
        if(bV & 0x08)
            Evt_ShortC();
        if(bV & 0x40)
            Evt_HSLOW();
        if(bV & 0x04)
            Evt_HERR();
        //User program start here.(18)

        //User program end here.(18)
    }
    MSFRADR = bBackupADR;
}
//-----------------------------------------------------------------------
//  Short Protect Event Function
//-----------------------------------------------------------------------
void Evt_ShortA()
{
    //User program start here.(13)
    if(_bADCCounterForSC == _bADCCounterForSC_Temp + 1)
    {
        _bADCCounterForSC_Temp = _bADCCounterForSC;
        _bShortCircuit_Cnt++;
    }
    else
    {
        _bADCCounterForSC_Temp = _bADCCounterForSC;
        _bShortCircuit_Cnt = 0;
    }

    if(_bShortCircuit_Cnt >= 2)
    {
        WRITE_MSFR(MSFR_MCNTL, USER_DEFINE_SVM_TABLE | MOTOR_FREE);
        _btShortCircuit_Protect = 1;
        P2_6 = FAULT_LED_TURNON;
    }
    //User program end here.(13)
}

void Evt_ShortB()
{
    //User program start here.(14)
    if(_bADCCounterForSC == _bADCCounterForSC_Temp + 1)
    {
```

```
        _bADCCounterForSC_Temp = _bADCCounterForSC;
        _bShortCircuit_Cnt++;
    }
    else
    {
        _bADCCounterForSC_Temp = _bADCCounterForSC;
        _bShortCircuit_Cnt = 0;
    }


if(_bShortCircuit_Cnt >= 2)
    {

        WRITE_MSFR(MSFR_MCNTL, USER_DEFINE_SVM_TABLE | MOTOR_FREE);

_btShortCircuit_Protect = 1;
P2_6 = FAULT_LED_TURNON;
    }
    //User program end here.(14)
}

void Evt_ShortC()
{
    //User program start here.(15)
    if(_bADCCounterForSC == _bADCCounterForSC_Temp + 1)
    {
        _bADCCounterForSC_Temp = _bADCCounterForSC;
        _bShortCircuit_Cnt++;
    }
    else
    {
        _bADCCounterForSC_Temp = _bADCCounterForSC;
        _bShortCircuit_Cnt = 0;
    }

    if(_bShortCircuit_Cnt >= 2)
    {
        WRITE_MSFR(MSFR_MCNTL, USER_DEFINE_SVM_TABLE | MOTOR_FREE);
        _btShortCircuit_Protect = 1;
        P2_6 = FAULT_LED_TURNON;
    }
    //User program end here.(15)
}
```

过流保护 (OCP)

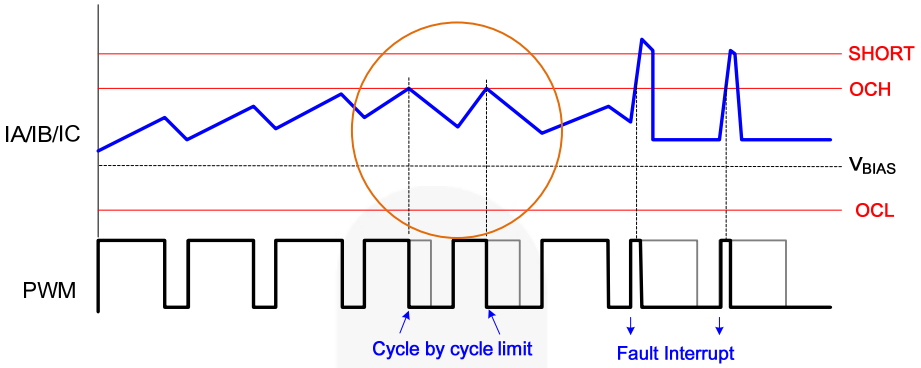


图 39. 过流保护

表 7. 过流保护寄存器

字节名称（地址）	位								Rst
	7	6	5	4	3	2	1	0	
OCCNTL (26h)	OC_DEB		OCCH_EA	OCBH_EA	OCAH_EA	OCCL_EA	OCBL_EA	OCAL_EA	00h
OCSTA (27h)	保留		OCCH	OCBH	OCAH	OCCL	OCBL	OCAL	00h
DAC0 (44h)	DAC_OCH								FFh
DAC1 (45h)	DAC_OCL								10h

表 8. OCCNTL 寄存器

地址：26h						复位：00h	
7	6	5	4	3	2	1	0
OC_DEB		OCCH_EA	OCBH_EA	OCAH_EA	OCCL_EA	OCBL_EA	OCAL_EA

OC_DEB: OC 保护去抖选择
00 → 300 ns+300-600 ns（模拟去抖）
01 → 600 ns+300-600 ns（模拟去抖）
10 → 900 ns+300-600 ns（模拟去抖）
11 → 1200 ns+300-600 ns（模拟去抖）
OCCH_EA: 相位 C OCH 保护使能
OCBH_EA: 相位 B OCH 保护使能
OCAH_EA: 相位 A OCH（电流相位）保护使能
OCCL_EA: OCC 保护使能
OCBL_EA: OCB 保护使能
OCAL_EA: OCA（电流相位）保护使能

表 9. OCSTA 寄存器

地址：27h						复位：00h	
7	6	5	4	3	2	1	0
保留		OCCH	OCBH	OCAH	OCCL	OCBL	OCAL

OCCH: 相位 C OCH 标识，读取后自动复位
OCBH: 相位 B OCH 标识，读取后自动复位
OCAH: 相位 A OCH 标识，读取后自动复位
OCCL: 相位 C OCL 标识，读取后自动复位
OCBL: 相位 B OCL 标识，读取后自动复位
OCAL: 相位 A OCL 标识，读取后自动复位

表 10. DAC0 寄存器

地址：44h						复位：FF h	
7	6	5	4	3	2	1	0
DAC_OCH							
DAC_OCH: 高电平 OC 保护 (0-4 V)							

表 11. DAC1 寄存器

地址：45h						复位：10 h	
7	6	5	4	3	2	1	0
DAC_OCL							
DAC_OCL: 低电平负 OC 保护 (0-4 V)							

因老化的电机轴承突然承受重载或电机转子被机械锁定导致瞬时大电流，电机和功率器件可能被损坏。若要保护电机和功率器件，FCM8531 可以在检测到过流后立即关断 PWM 信号。

按照下面顺序，设置过流保护参数：

i. 禁用过流保护；

ii. 设置去抖时间；

iii. 设置保护电平；

iv. 使能过流保护；然后

v. 从 OCSTA 寄存器读取过流状态。

硬件会自动根据设置提供保护。此外，相关状态可从 OCSTA(0x27) 读取。

过流保护示例程序代码如下。（如果与该步骤有关的 SFR、#define、变量名或程序代码已在先前的示例程序中声明，则为了简便起见，此处省略重复性声明）

```
/*---------------------------------------------------------------------
 * Copyright 2011 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------
*/
#define INITIAL_OCCNTL     0x38
// OCH A phase, OCH B phase, OCH C phase protect enable
// OCL Disable
// OC debounce: 300 ns
#define INITIAL_OCH        0xBF
#define INITIAL_OCL        0x00
//---------------------------------------------------------------------
//  Initial Protection Register
//---------------------------------------------------------------------
void Initial_Protect()
{
    WRITE_MSFR(MSFR_OCCNTL, INITIAL_OCCNTL);
    WRITE_MSFR(MSFR_OCH, INITIAL_OCH);
    WRITE_MSFR(MSFR_OCL, INITIAL_OCL);
    WRITE_MSFR(MSFR_SHORT, INITIAL_SHORT);
}
```

过压保护 (OVP)

FCM8531 中采用多个 ADC 输入，用于检测外部信号，如 IA、IB、IC、VA、VB、VC。在该示例中，采用一个 ADC 输入来检测直流总线电压，并通过示例代码实现过压保护。

突然的电机降速或制动可能因为反电动势引起直流总线上出现过大电压。过大电压可能损坏驱动功率器件和电容。需要一个很好的保护方案。

图 40 给出一个推荐电路。ADC3 由软件检测，用于监控直流总线电压。一旦软件检测到过压，PWM 信号输出就被关断。过压保护一直持续到直流总线电压返回到正常电平。

为了避免保护电平附近出现 OVP 抖动，在程序中定义了一个电压滞环范围。

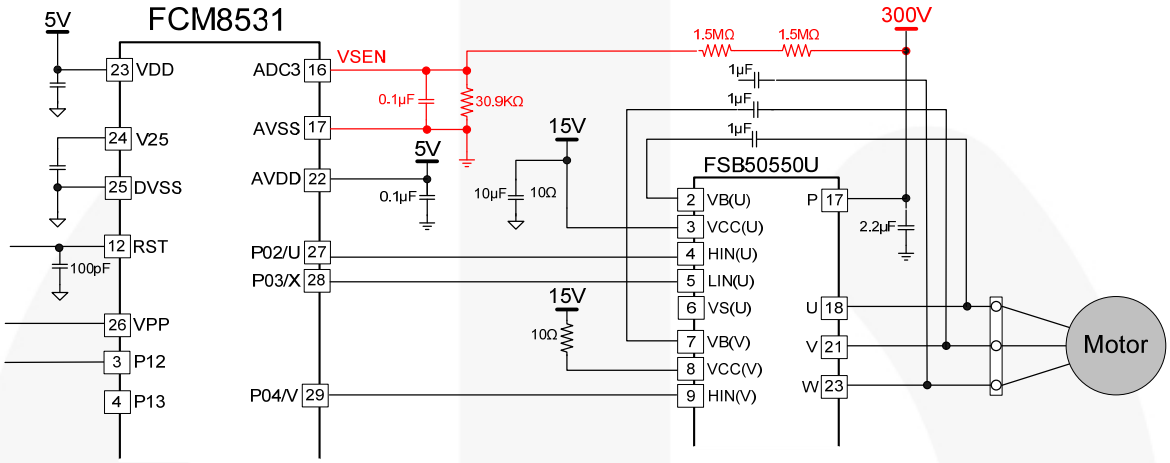


图 40. 过压保护电路

过流保护示例程序代码如下。（如果与该步骤有关的 SFR、#define、变量名或程序代码已在先前的示例程序中声明，则为了简便起见，此处省略重复性声明）

```
/*-------------------------------------------------------------------------
 * Copyright 2012 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *-------------------------------------------------------------------------
*/
#define HVDD_OVP_LATCH              0xED          // DC 440V OVP Latch
#define HVDD_OVP_RELEASE            0xD2          // DC 380V OVP Release
//-------------------------------------------------------------------------
//  ADC Interrupt
//-------------------------------------------------------------------------
// ADC ISR
INTERRUPT(ISR_ADC, VECTOR_EX9)
{
    U8 bBackupADR;
    U8 bV;
    //User variable start here.(23)

    //User variable end here.(23)
    bBackupADR = MSFRADR;
    //User program start here.(1A)
    READ_MSFR(MSFR_ADC0H, _wVsp_Input.U8[0]);
    READ_MSFR(MSFR_ADC0L, _wVsp_Input.U8[1]);
    READ_MSFR(MSFR_VAH, _bVsen_Input);
    READ_MSFR(MSFR_VBH, _bSPMOT_Input);

    //User program end here.(1A)

     MSFRADR = bBackupADR;
}
```

```
//---------------------------------------------------------------------
//  Motor Over Voltage Protection Check
//---------------------------------------------------------------------
void Check_OVP_OTP()
{
    if(_bVsen_Input > HVDD_OVP_LATCH)
    {
        _btOverVoltage_Protect = 1;
        P2_6 = FAULT_LED_TURNON;
    }
    if(_bSPMOT_Input > SPM_OTP_LATCH)
    {
        _btOverTemperature_Protect = 1;
        P2_6 = FAULT_LED_TURNON;
    }

    if(_btOverVoltage_Protect | _btOverTemperature_Protect)
    {
        if(_btOverVoltage_Protect && (_bVsen_Input < HVDD_OVP_RELEASE))
        {
            _btOverVoltage_Protect = 0;
            if(!_btShortCircuit_Protect)
                P2_6 = FAULT_LED_TURNOFF;
        }
        if(_btOverTemperature_Protect && (_bSPMOT_Input < SPM_OTP_RELEASE))
        {
            _btOverTemperature_Protect = 0;
            if(!_btShortCircuit_Protect)
                P2_6 = FAULT_LED_TURNOFF;
        }
    }
}
```

过温保护 (OTP)

对 OTP 采用一个与 OVP 类似的方案，除了输入源不同。NTC 电阻和正常电阻级联，形成分压器。由软件读取和处理检测到的电压，并与滞环窗比较，当出现 OTP 时关断 PWM 信号，当 OTP 解除时，再次导通 PWM 信号。

图 41 中显示的推荐电路是评估板中的真实设计。

过温保护示例程序代码如下。（如果与该步骤有关的 SFR、#define、变量名或程序代码已在先前的示例程序中声明，则为了简明起见，此处省略重复说明。）

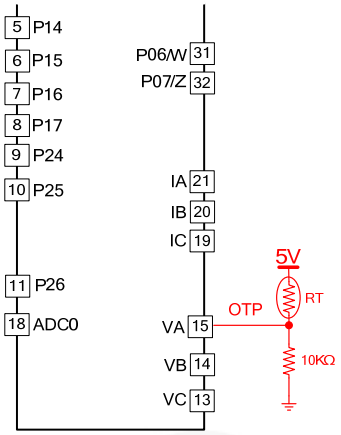


图 41. 过温保护电路

```
/*---------------------------------------------------------------------------
 * Copyright 2011 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
*/
#define SPM_OTP_LATCH        0xDF                  // 100 degree
#define SPM_OTP_RELEASE      0x95                  // 60 degree
//---------------------------------------------------------------------------
//  ADC Interrupt
//---------------------------------------------------------------------------
// ADC ISR
INTERRUPT(ISR_ADC, VECTOR_EX9)
{
    U8 bBackupADR;
    U8 bV;
    //User variable start here.(23)

    //User variable end here.(23)
    bBackupADR = MSFRADR;
    //User program start here.(1A)
    READ_MSFR(MSFR_ADC0H, _wVsp_Input.U8[0]);
    READ_MSFR(MSFR_ADC0L, _wVsp_Input.U8[1]);
    READ_MSFR(MSFR_VAH, _bVsen_Input);
    READ_MSFR(MSFR_VBH, _bSPMOT_Input);

//User program end here.(1A)
    MSFRADR = bBackupADR;
}

//---------------------------------------------------------------------------
//  Motor Over Voltage Protection Check
//---------------------------------------------------------------------------
void Check_OVP_OTP()
{
    if(_bVsen_Input > HVDD_OVP_LATCH)
    {
        _btOverVoltage_Protect = 1;
        P2_6 = FAULT_LED_TURNON;
    }
    if(_bSPMOT_Input > SPM_OTP_LATCH)
    {
        _btOverTemperature_Protect = 1;
```

```
        P2_6 = FAULT_LED_TURNON;
    }

    if(_btOverVoltage_Protect | _btOverTemperature_Protect)
    {
        if(_btOverVoltage_Protect && (_bVsen_Input < HVDD_OVP_RELEASE))
        {
            _btOverVoltage_Protect = 0;
            if(!_btShortCircuit_Protect)
                P2_6 = FAULT_LED_TURNOFF;
        }
        if(_btOverTemperature_Protect && (_bSPMOT_Input < SPM_OTP_RELEASE))
        {
            _btOverTemperature_Protect = 0;
            if(!_btShortCircuit_Protect)
                P2_6 = FAULT_LED_TURNOFF;
        }
    }
}
```

相关数据手册

FCM8531 — 嵌入式 MCU 和可配置三相 PMSM / BLDC 电机控制器

AN-8202 — FCM8531 用户手册（硬件说明）

AN-8203 — FCM8531 用户手册（指令集）

AN-8204 — FCM8531 AMC 库：速度积分

AN-8205 — FCM8531 AMC 库：霍尔接口

AN-8207 — 飞兆电机控制开发系统 (MCDS) 集成式开发环境 (IDE)