AN-8204
AMC 库：速度积分

开发系统

FCM8531 典型开发环境如图 1 所示。应用板可采用 FCM8531 评估板或用户自定义电路板（被称为“目标板”）。FCM8531 评估板可与飞兆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，帮助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。使用飞兆提供的 AMC 库，用户可快速开发无传感器的电机驱动应用。本文档是针对 AMC 库“速度积分”的用户指南。有关 MCDS IDE 和 MCDS 编程套件的详情，请参见飞兆网站：
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。在系统编程 (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。

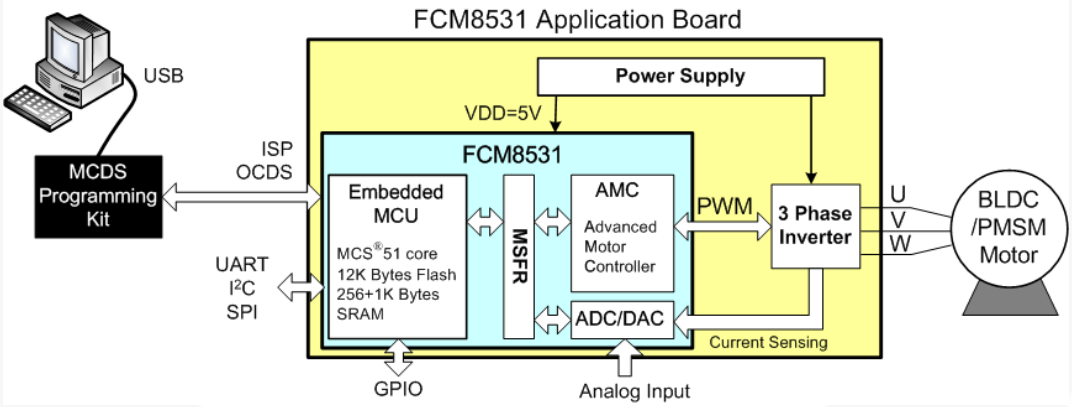


图 1. 典型开发环境

AMC 简介

FCM8531 是一款特定应用控制器，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。

本文件主要介绍速度积分库的原理和使用方法，如图 2 所示。AMC 处理器使用 ADC 获取电机电流，然后通过坐标变换和速度积分算法估算转子角度。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。

速度积分

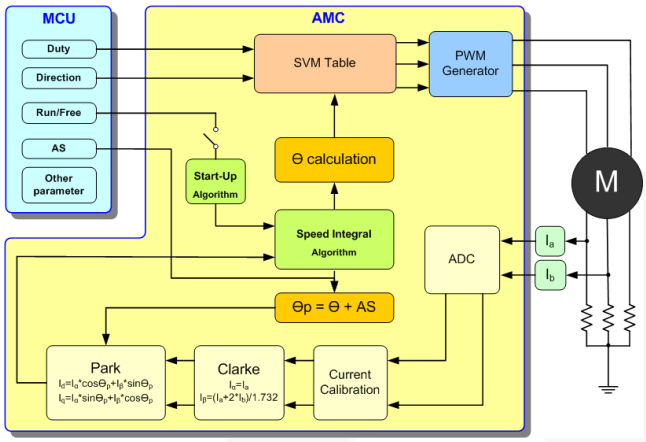


图 2. 速度积分框图

速度积分的原理如下：比例积分 (PI) 控制器利用电流矢量的相位误差产生速度命令，然后对速度命令进行积分运算，产生位置信号，从而实现电机的无传感器控制。速度积分库通过 MCDS IDE 接口的项目设置对话框选择，如图 3 所示。

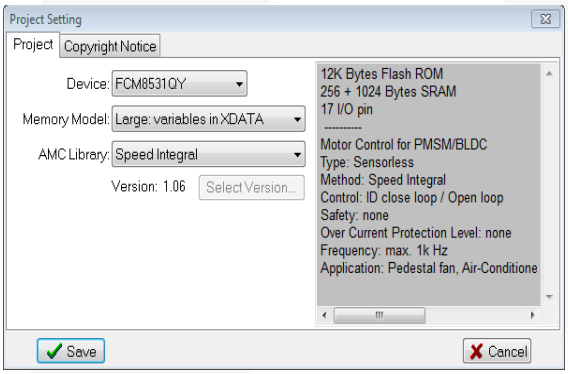


图 3. MCDS IDE - 项目设置

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。d 轴和 q 轴分量通常称为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 4 所示，三相参考系变换为两向参考系时，三相参考系的合成向量 f_S 会投影到两向参考系的坐标轴上。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。

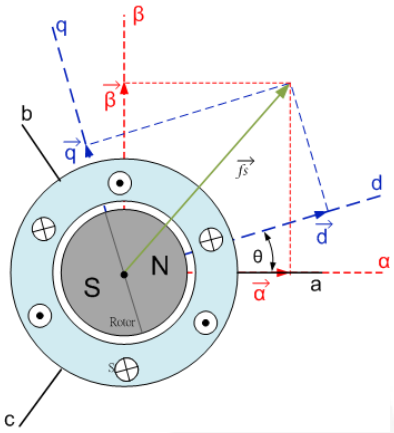


图 4. a-b-c 和 d-q 参考系

下文介绍三个变换模块的等式。

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 5 所示），从而将复杂的三相坐标简化为两相坐标。Clarke 变换的等式如下。

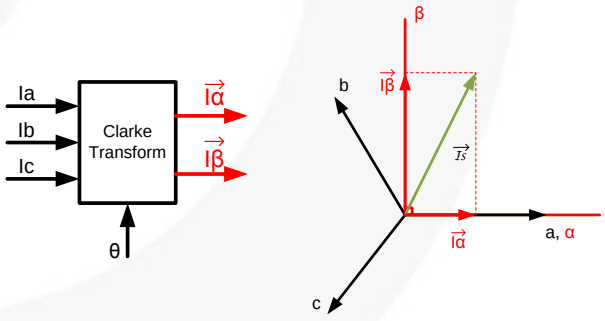


图 5. Clarke 变换

$$i_\alpha = i_a$$

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

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。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 6 所示。Park 变换的等式如下。

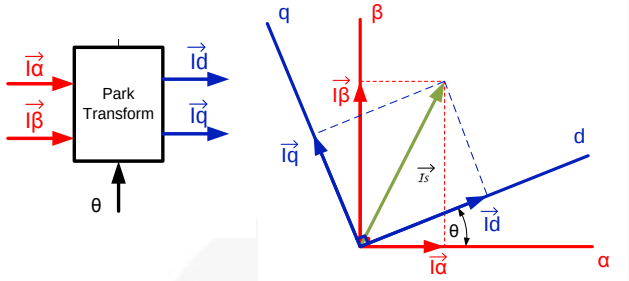


图 6. Park 变换

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

Park^{-1} 变换：

Park^{-1} 变换（也被称为“Park 逆变换”）是 Park 变换的逆变换。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 7 所示。Park^{-1} 变换的等式如下。

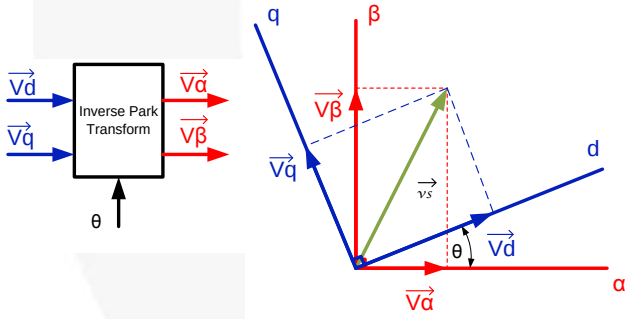


图 7. Park^{-1} 变换

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

速度积分理论

速度积分库的无传感器控制算法被称为速度积分算法。永磁正弦电机 (PMSM) 或无刷直流电机（BLDC 电机）的转子位置可通过控制直接轴电流 I_d 并使其数值为零估算得出。估算转子角度需执行四个步骤；这些步骤已在速度积分库中集成：

- 电流测量
- 坐标变换
- PI 控制
- 积分计算

首先，由电流检测电阻获取电机的三相电流（I_a、I_b 和 I_c）。随后，利用 Clarke 变换和 Park 变换将 I_a、I_b 和 I_c 变换为两相电流（I_d 和 I_q）。接着，控制直接轴电流 I_d 使其为零，从 PI 控制器产生速度命令。也就是说，PI 控制器用于消除参考电流 I_{d_ref} 和实际电流 I_d 之间的电流误差。最后，进行速度命令的积分运算以提供转子角度 (Θr) 信息，从而产生相应的 PWM 信号，实现无传感器电机控制。

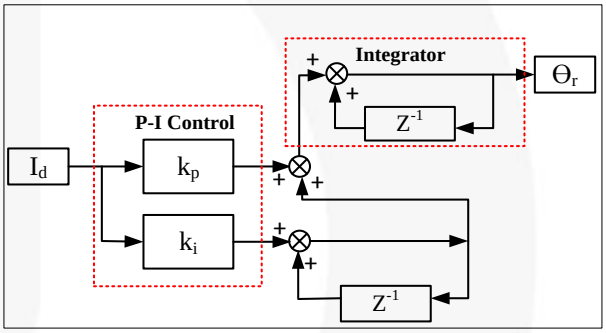


图 8. 速度积分积分器

速度积分和 IEC 60730-1 B 类

为了满足家用电器的 IEC 60730-1 B 类安全标准，速度积分具有 OCP-1 V 和 OCP-1.5 V 版本，由 UL 提供合规认证。开发产品时，这些信息可以从 MCDS IDE 中选择。用户可以直接使用 UL 认证的 AMC 库来缩短电控 PMSM / BLDC 电机的产品开发周期。

速度积分 OCP 1 V 和 OCP 1.5 V 与非 UL 版本相同，仅 IEC 60730-1 B 类有一些改动。用户应仔细阅读附录中的“如何使用 AMC 并符合 IEC 60730-1 B 类标准要求”部分内容，以便开发的产品在硬件和固件上完全合规。

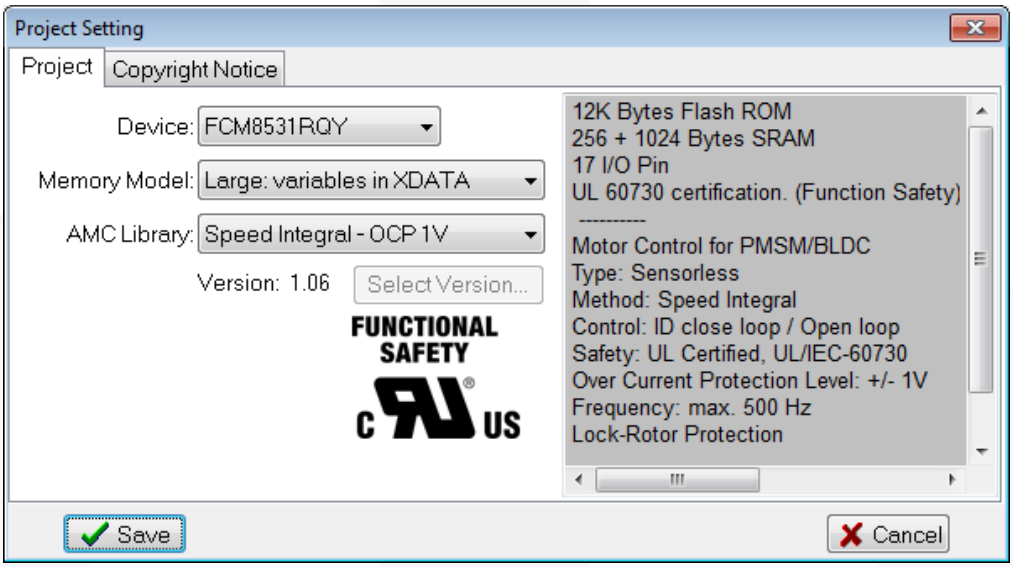


图 9. 速度积分 OCP 1 V 版本，符合 IEC 60730-1 B 类标准

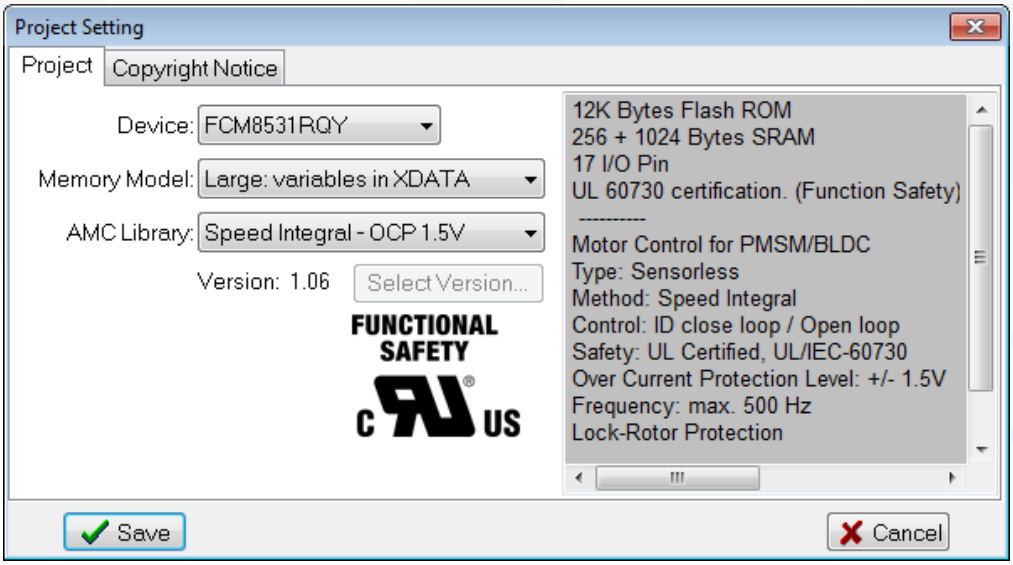


图 10. 速度积分 OCP 1.5 V 版本，符合 IEC 60730-1 B 类标准

控制参数

本节中详细介绍了速度积分库中的各种参数，用户因而能了解如何调试参数值以驱动不同特性的电机。参数包括电机速度命令、电机启动参数、速度积分算法参数和角度位移参数。

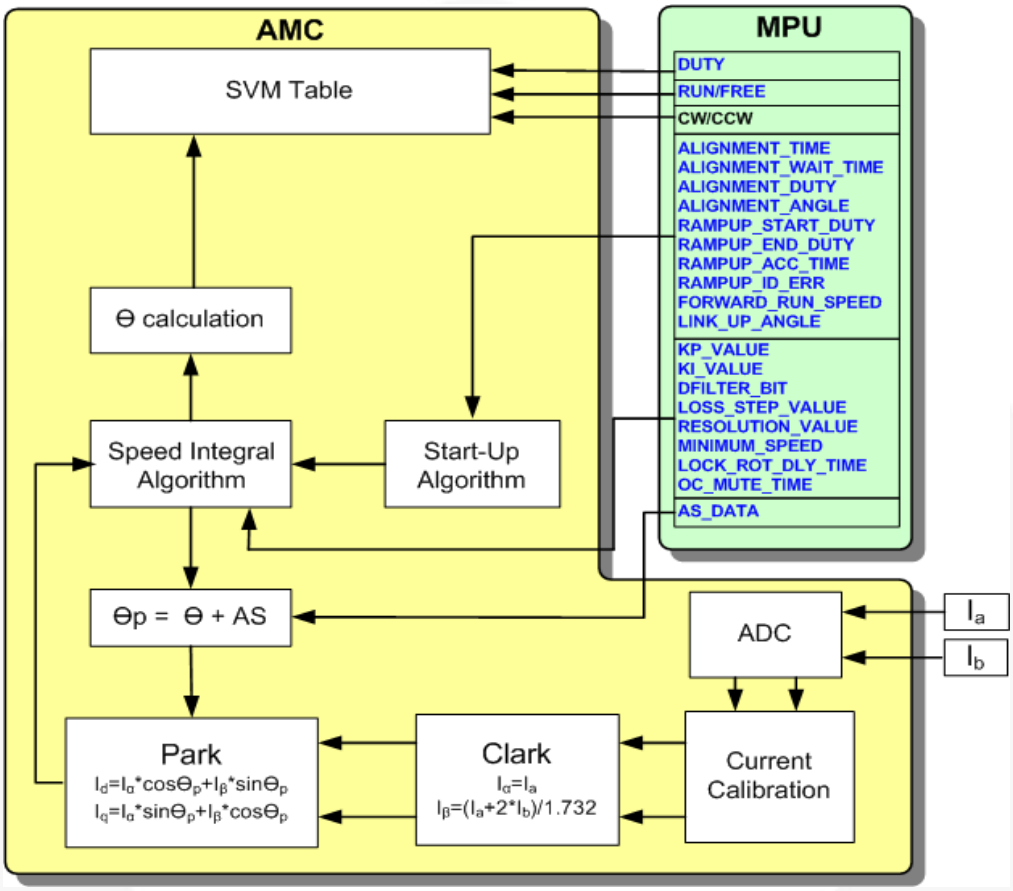
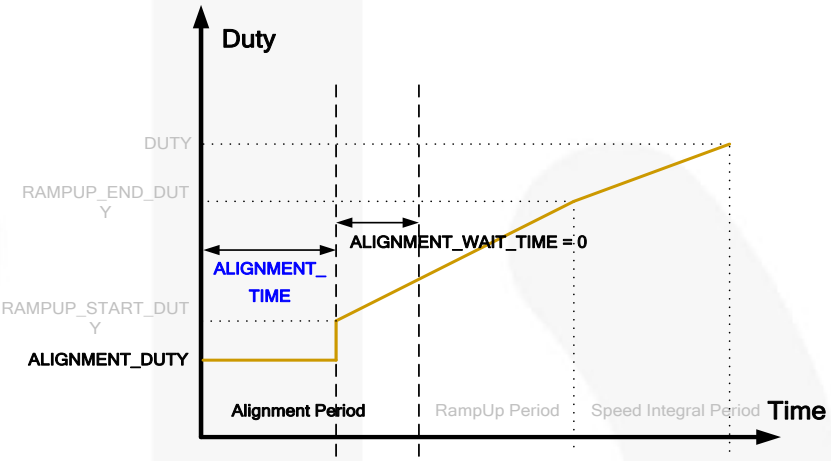
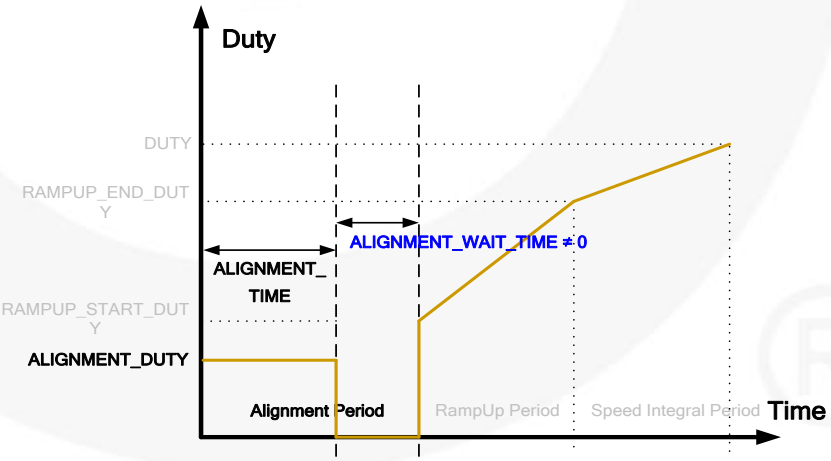
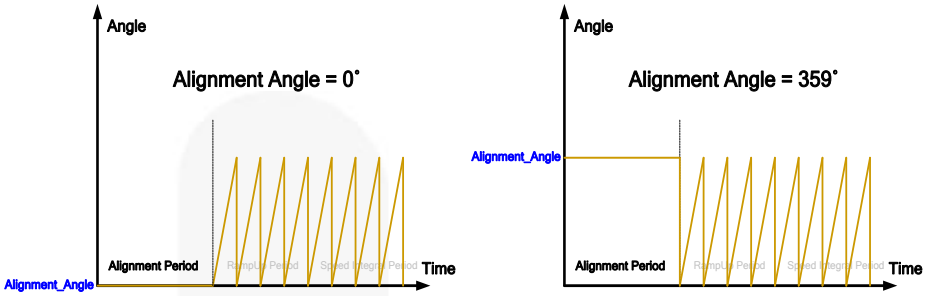
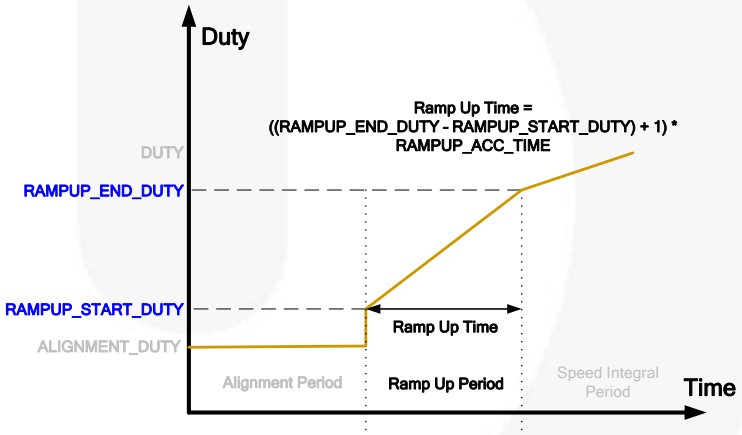
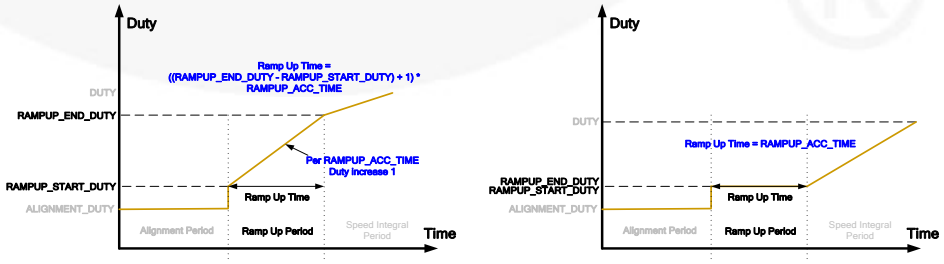


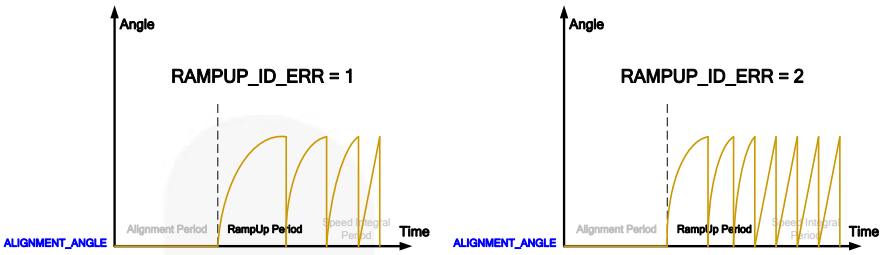
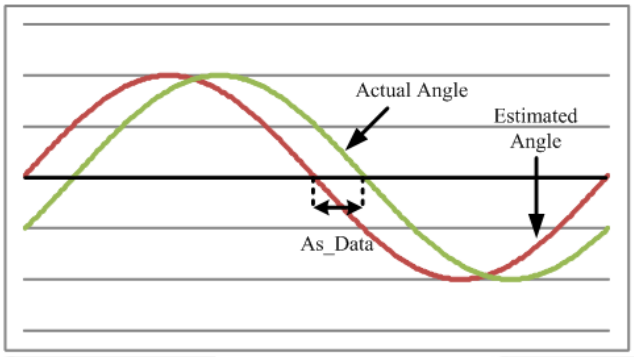
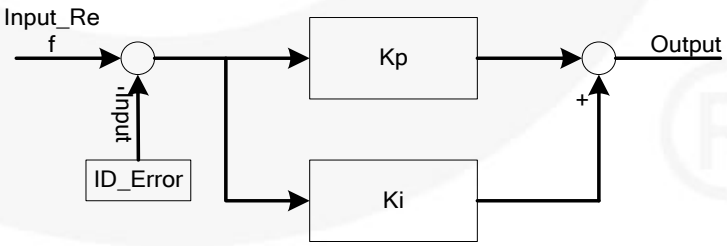
图 11. 速度积分参数

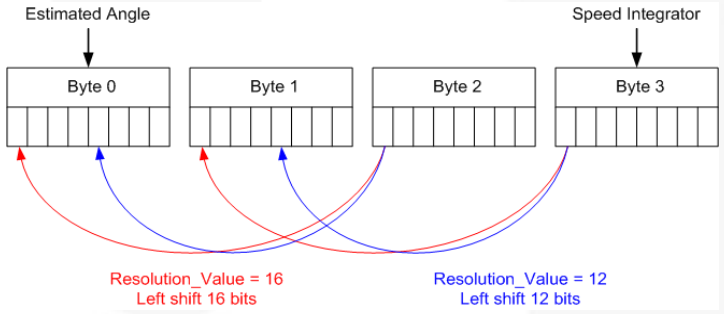
参数说明

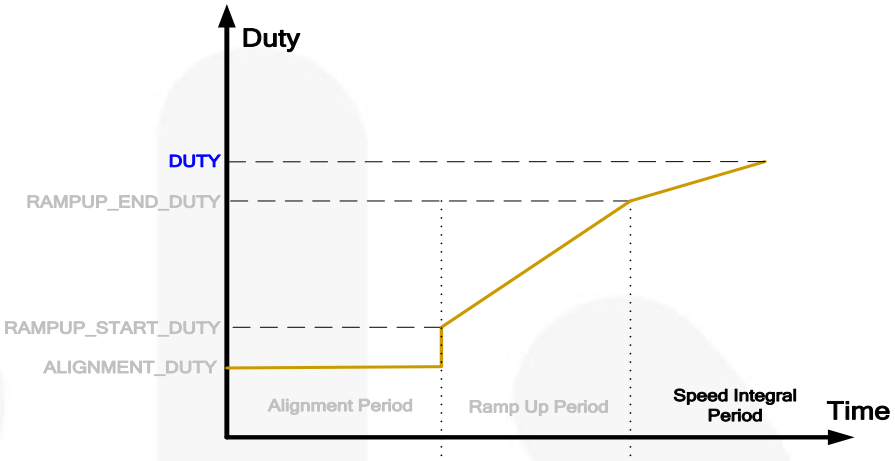
表 1. 参数列表

参数名称	说明
ALIGNMENT_TIME	ALIGNMENT_TIME 是开始斜升前将转子移动到特定位置所需的时间。 斜升前，由于无法确认转子位置，因此需将转子与特定位置对齐。对齐转子的时间通过设置 ALIGNMENT_TIME 可调。 ALIGNMENT_WAIT TIME 和 ALIGNMENT_TIME 之和不可超过 5 s。该参数占用 8 位寄存器，单位：20 ms。  **图 12. 对齐时间**
ALIGNMENT_WAIT_TIME	ALIGNMENT_WAIT_TIME 是电机在 ALIGNMENT_TIME 时间后与特定位置稳定对齐前的时间。 加入一个等待时间，允许转子从之前的对齐位置转到某一特定位置后振动一段时间。该周期中占空比自动设为零。设置 ALIGNMENT_WAIT_TIME = 0，禁用等待时间。 ALIGNMENT_WAIT TIME 和 ALIGNMENT_TIME 之和不可超过 5 s。该参数占用 8 位寄存器，单位为 20 ms。  **图 13. 对齐等待时间**
ALIGNMENT_DUTY	ALIGNMENT_DUTY 是斜升前占空比将转子与特定位置对齐的占空比。对齐时应考虑 ALIGNMENT_DUTY、ALIGNMENT_TIME 和 ALIGNMENT_ANGLE（在下个模块中叙述）。 该参数占用 9 位寄存器，最大输出为 0x1FF。

参数名称	说明
ALIGNMENT_ANGLE	ALIGNMENT_ANGLE 表示对齐模式下 PWM 占空比的电气角度。它是一个 8 位寄存器，范围从 0 到 255。 Angle = 360° * ALIGNMENT_ANGLE / 256.  **图 14. 对齐角度**
RAMPUP_START_DUTY **RAMPUP_END_DUTY**	RAMPUP_START_DUTY 和 RAMPUP_END_DUTY 定义斜升周期开始和结束时的斜升周期。RAMPUP_END_DUTY 必须大于等于 RAMPUP_START_DUTY 才能保证正常工作。如果 RAMPUP_START_DUTY 定义的驱动电能过低，那么电机可能无法转动；如果驱动电能过高，则电机可能停止转动。斜升时，考虑 RAMPUP_START_DUTY、RAMPUP_END_DUTY、RAMPUP_ACC_TIME 和 RAMPUP_ID_ERR（在下个模块中描述），确保平滑启动并过渡到最终速度积分阶段。这两个参数占用 8 位寄存器，最大输出为 0xFF。  **图 15. 斜升起始占空比和斜升结束占空比**
RAMPUP_ACC_TIME	RAMPUP_ACC_TIME 是斜升模式下的斜升占空比斜率。占空比强制始于 RAMPUP_START_DUTY，并且每次 RAMPUP_ACC_TIME 递增 1，直到到达 RAMPUP_END_DUTY。之后，关闭控制环路，开始执行速度积分。 斜升时间等于 RAMPUP_ACC_TIME（如果 RAMPUP_START_DUTY 与 RAMPUP_END_DUTY 设置为两者相等的话）。 该参数占用 16 位寄存器，单位：1 ms。  **图 16. 斜升加速时间**

参数名称	说明
RAMPUP_ID_ERR	RAMPUP_ID_ERR 是斜升加速度，占用 8 位寄存器。加速度随参数值变大而变大。尤其建议将该参数设为 1 和 2 之间的数值。若该参数过大，启动步骤可能会失败。  **图 17. 斜升 ID 错误**
AS_DATA	AS_DATA 是速度积分的补偿角度。 估计角度与实际角度之间的计算值为角度误差，如图 18 所示。为了使定子磁场与转子象限电流正交对齐以便优化电机效率，必须以 AS_DATA 校正误差。Fairchild 提供的 MCDS IDE 软件包含电机调试工具，可在不同速度范围内找到适当的补偿角度。 该参数占用 8 位寄存器，范围为 0 至 127，对应于 0 到 180 度。  **图 18. 角度位移**
KP_VALUE **KI_VALUE**	KP_VALUE 和 KI_VALUE 是速度积分算法中 PI 控制器的可控比例变量。 较大的 KP_VALUE 具有较少的稳态误差和较高的环路灵敏度，而环路稳定性相对较低。KI_VALUE 与 KP_VALUE 一同使用时，可消除稳态误差。为了获得平滑的瞬态响应并消除稳态误差，应同时考虑这两个变量。 KP_VALUE 参数占用 16 位寄存器，可设置范围为 0 至 65,535。 KI_VALUE 参数占用 16 位寄存器，可在 0 到 65,535 范围内设置。实际值在速度积分算法中除以 256。  **图 19. 速度积分 PI 控制**

参数名称	说明
MINIMUM_SPEED	MINIMUM_SPEED 是允许的最低电机速度，由速度积分控制。 一旦速度积分输出的速度值低于 MINIMUM_SPEED，输出速度就会箝位在 MINIMUM_SPEED。该参数占用 8 位寄存器。
LPSS_STEP_VALUE	LOSS_STEP_VALUE 可用来确定电机是否失去同步控制。 PI 控制器的积分输出大于该参数时，可判断电机损失步长为真。算法让电机停止工作，并在发生电机损失步长时设置 AMC_Fault。 该参数可根据电机的最大速度进行设置。 LOSS_STEP_VALUE = $(K \times P \times t_S \times (2^{(40\text{-RESOLUTION\_VALUE})}) \times (Rpm\_Max/120)) / 2^{16}$ 其中， K: 1.5 至 2 P: 电机极数 t_S: 125 µs（速度积分算法的采样时间） Rpm_Max: 电机的最大旋转速度 RESOLUTION_VALUE: 角度计算的分辨率
RESOLUTION_VALUE	RESOLUTION_VALUE 是角度计算的分辨率值。 速度积分器是一个 32 位寄存器。最低字节（字节 3）用于对速度 (ω) 进行积分，角度最终估算值位于最高字节（字节 0）。RESOLUTION_VALUE 确定速度积分寄存器从右向左移的位数。较小的 RESOLUTION_VALUE 值可获得较大的角度计算值，反之亦然。如果需要新的（更高 /更低）分辨率来确保电机正常工作，那么当 RESOLUTION_VALUE 改变时（减少/增大）时，必须调节 KP_VALUE 和 KI_VALUE（增大/减少），反之亦然。 该参数的可设置范围为 1 至 16；建议设为 16。  **图 20. 分辨率值**

参数名称	说明
DUTY	DUTY 用于控制 PWM 输出信号的占空比，范围为 0x00 至 0x1FF。 占空比随速度增大或减少。如果速度变化幅度过大，则很容易损失电机步长。在设置 STR_RDY 的斜升模式结束以前不要设置占空比数值，因为占空比由 AMC 在斜升模式下设置。  **图 21. 控制占空比**
DFILTER_BIT	DFILTER_BIT 是数字滤波器的阶数，范围为 0 至 10。 数字滤波器通过过滤角度变化来稳定角度输出，其阶数由此参数确定。高阶值过大会降低系统响应时间，因此建议将该参数的数值范围设为 1 至 2。
LOCK_ROT_DLY_TIME	电机锁定后，在 LOCK_ROT_DLY_TIME 设置的时间周期内，AMC 关闭 PWM 输出，并忽略来自 MCU 的一切命令。 LOCK_ROT_DLY_TIME 的输入范围为 0~255，对应于 0~255 s。
OC_MUTE_TIME	若 VIA、VIB 或 VIC 在 OC_MUTE_TIME 设置的时间内连续超过 IEC 60730-1 B 类标准规定的过流保护水平，则 AMC 关断 PWM 输出，同时触发故障。 OC_MUTE_TIME 的输入范围为 0~106，对应于 0~53 s。
RUN	RUN 用于启动或停止 AMC 的速度积分算法。 若 Run=1，则启动速度积分算法； 若 Run=0，则停止速度积分算法。 若要开始执行速度积分，MSFR (MCNTL) 的位 0 必须设置为 RUN =1。

测试空调室外风扇的控制参数参照表在表 2 中列出。

表 2. 空调室外风扇的参数

参数	寄存器位	建议值
ALIGNMENT_TIME	8-bit	0x64
ALIGNMENT_WAIT_TIME	8-bit	0
ALIGNMENT_DUTY	8-bit	0x14
ALIGNMENT_ANGLE	8-bit	0
RAMPUP_START_DUTY	8-bit	0x20
RAMPUP_END_DUTY	8-bit	0x30
RAMPUP_ACC_TIME	16-bit	0x19
RAMPUP_ID_ERR	8-bit	2
MINIMUM_SPEED	8-bit	0
AS_DATA	8-bit	Adjusted by Experiments
KP_VALUE	16-bit	0xC8
KI_VALUE	16-bit	0x100
DFILTER_BIT	8-bit	1
LOSS_STEP_VALUE	16-bit	Adjusted by Experiments
RESOLUTION_VALUE	8-bit	0x10
LOCK_ROT_DLY_TIME	8-bit	0xA
OC_MUTE_TIME	8-bit	0xA

实验数据

测试空调室外风扇的实验波形在本节中显示。用户可以看到电机在无传感器控制的情况下，运行良好时的正确时序图。

表 3. 实验规格

参数	规格 / 条件
输入电压	176 - 264 V_{AC}, 50/60 Hz
额定输出功率	60 W（220 V_{AC} 时的典型值）
最大输出功率	100 W ±2 W（220 V_{AC} 时）
待机功耗	0.8 W ±0.1 W（220 V_{AC} 时）
速度调节范围	300 - 1100 rpm
速度调节响应时间	13 s （加速度为 0 - 1100 rpm）
V_{SP} 输入电压	0 – 5.0 V，< 0.4 V = 电机不工作 ≥ 0.4 V = 最小速度，150 rpm
DIR 输入电压	切换开关输入电平， 高电平 = 三相电机：U→V→W 低电平 = 三相电机：W→V→U
电机过压保护（OVP）的阈值电压	> 440 V（典型值）
电机 OVP 的释放电压	< 380 V（典型值）
电机短路保护的阈值电流	2.0 A（典型值）
PWM 逐周期电流保护的阈值电流	1 A（典型值）
SPM 过温保护	100° C

其中，

CH1： 显示估算角度

CH2： 显示 U 相驱动电压；

CH3： 显示 A 相霍尔信号（用于验证估算角度的精度）；以及

CH4： 显示 U 相驱动电流。

图 22 显示的是 U 相电压、电流、霍尔信号和估算角度的波形。电流波形为正弦波；电机采用正弦波驱动。峰值 U 相驱动电流与霍尔信号的下降沿对齐，表示估算角度精确。AS 调试可用于校正超前或滞后电流。峰值电流与估算角度的相位关系可通过 FCM8531 的 SVM 表格配置。

U 相电流

测试条件：输入电压 = AC 220 V，速度 = 1,500 rpm。规格：电流波形必须为正弦波，且霍尔信号的下降沿必须对准电流波形正半周的中心点。

CH1: Theta CH2: SU CH3: HA CH4: 电流

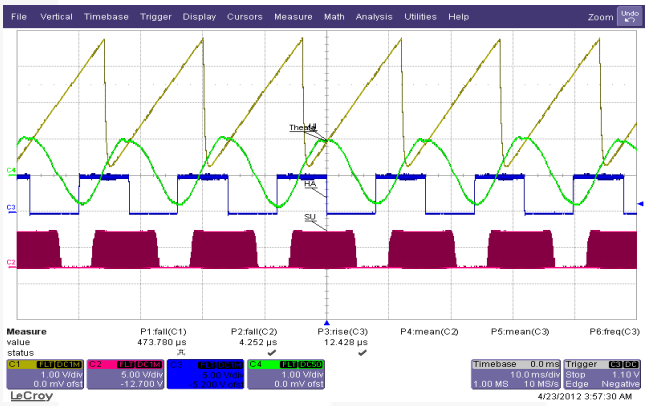


图 22. U 相电流波形

通信

通信接口

MCU 和 AMC 之间的通信通过邮箱寄存器进行传输。AMC 通过 MTX0(B0h) -MTX3(B3) 从 MCU 接收控制命令和数据，控制相应的电机；MCU 通过 MRX0(B4h) - MRX3(B7h) 读取 AMC 传送过来的数据。如图 23 所示。

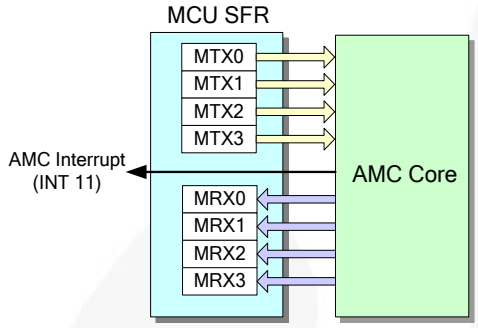


图 23. 通信接口

邮箱寄存器定义

本节介绍了用于通信的每个寄存器定义。

MCU 数据传输 (MTX0 - MTX3)

MTX0:

b0（触发位）：

发送到 AMC 的命令和数据保存在相关寄存器中。b0 位从 1 转换为 0 可启动传输。

b1–b7（命令）：

保存发送到 AMC 的命令。

MTX1:

b0–b7（高位数据字节）：

保存发送到 AMC 的高位字节数据。

MTX2:

b0–b7（低位数据字节）：

保存发送到 AMC 的低位字节数据。

MTX3:

b0 (ForwardFreeRun)：

确定电机正向启动：1：正向，0：非正向。

b1 （反向空闲运行）：

确定电机反向启动：1：反向，0：非反向。

b2–b7（保留）：

设为 0。

MCU 数据读取 (MRX0 - MRX3)

MRX0:

b0: 保留

b1 (AMC_Cmd):

AMC_Cmd=1：AMC 忙于处理命令，不能接收命令。

AMC_Cmd=0：AMC 能接收命令。

b2 (AMC_Cal):

AMC_Cal=1：AMC 忙碌，不能接收命令。

AMC_Cal=0：AMC 可用，并且可接收命令。

b3 (AMC_Fault):

AMC_Fault=1：AMC 运行异常 发生 AMC_Fault 时，MCU 可尝试重试或停止。

AMC_Fault=0：AMC 运行正常。

b4 (STR_Rdy):

STR_Rdy=1：AMC 已完成启动步骤。MCU 可进行随后的通用电机控制。

STR_Rdy=0：AMC 未完成启动步骤。

b5 (Al_Rdy):

Al_Rdy=1：AMC 已完成对齐步骤；可启动随后的斜升控制。

Al_Rdy=0：AMC 未完成对齐步骤。

b6 (RST_Rdy):

RST_Rdy=1：AMC 已完成复位操作；MCU 可以开始传输和读取数据。

RST_Rdy=0：AMC 未完成复位操作。

b7: (SAFETY_Warning):

SAFETY_Warning=1：与安全有关的故障 —— 包括 IEC60730-1 B 类故障 —— 可通过 AMC 检测。

SAFETY_Warning=0：无故障。

故障对应的错误信息参见附录中的“如何使用 AMC 并符合 IEC 60730-1 B 类标准要求”部分内容。

MRX1:

b0–b7（高位数据字节）：

存储从 AMC 传输的数据的高位字节。

MRX2:

b0–b7（低位数据字节）：

存储从 AMC 传输的数据的低位字节。

表 4. 邮箱寄存器定义

字节名称（地址）	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
MTX0 (B0h)	命令							触发
MTX1 (B1h)	高位数据字节							
MTX2 (B2h)	低位数据字节							
MTX3 (B3h)	保留							正向
MRX0 (B4h)	SAFETY_Warning	RST_Rdy	AI_Rdy	STR_Rdy	AMC_Fault	AMC_Cal	AMC_Cmd	保留
MRX1 (B5h)	高位数据字节							
MRX2 (B6h)	低位数据字节							
MRX3 (B7h)	保留							

通信协议

必须完全执行正确的通信协议和 MCU 与 AMC 之间的流向，以避免传输错误。通信协议和流向的正确步骤说明见下文流程图。

MCU 将命令写入 AMC

步骤 1：检查 AMC 是否忙碌。

若结果为“是”，则重复执行步骤 1，直到超时。

若结果为“非”，则执行步骤 2。

步骤 2：在相应寄存器中存储命令和数据，然后将 MTX0 的 b0 从 1 转换为 0，开始传输。

步骤 3：等待 20 µs。

步骤 4：检查 AMC 是否已完成处理命令，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“非”，则传输完成。

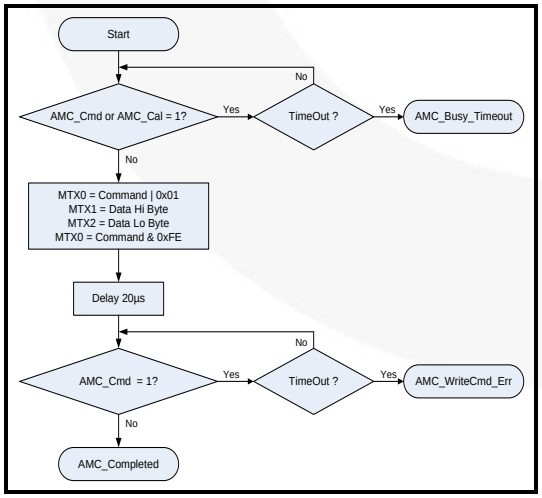


图 24. 写入命令协议

MCU 从 AMC 读取数据

步骤 1：检查 AMC 是否忙碌。

若结果为“是”，则重复执行步骤 1，直到超时。

若结果为“非”，则执行步骤 2。

步骤 2：将命令和数据保存在相应的寄存器中，然后将 MTX0 的 b0 从 1 转换为 0，以开始传送。

步骤 3：等待 20 µs。

步骤 4：检查 AMC 是否已完成处理命令，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“非”，执行步骤 5。

第5步：确认 AMC 是否已完成算法的执行 (AMC_Cal=1)。

若结果为“是”，则重复执行步骤 5，直到超时。

若结果为“非”，则读操作已完成，数据自动存储在 MRX1 和 MRX2 中。

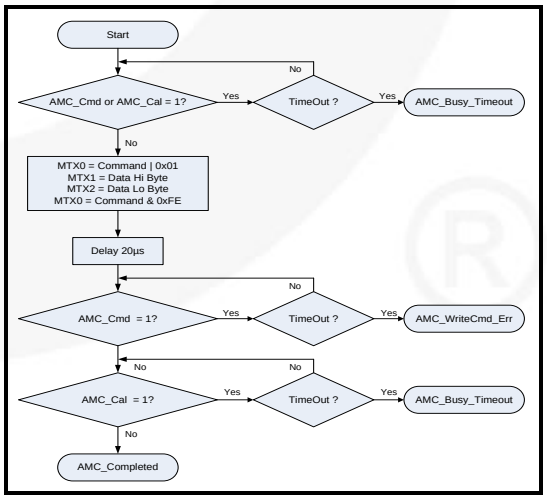


图 25. 读取数据协议

内置函数

为简便起见，当采用 MCDS IDE 软件建立新项目时，将产生两个函数（如下文所述）。这些函数用来在 MCU 和 AMC 之间传输并读取数据，缩短编码时间。

MCU 将命令写入 AMC

bWriteCmdToAMC(U8 Command, U8 Data_High_Byte, U8 Data_Low_Byte)

其中，

命令：与已传输数据对应的命令。

Data_High_Byte：已传输数据的高位字节。

Data_Low_Byte：已传输数据的低位字节。

示例：

2,000 (0x64) ms 对齐时间可设置如下：

bWriteCmdToAMC(CMD_ALIGNMENT_TIME, 0x0, 0x64)

若函数返回值为 0，则数据成功传输。

MCU 从 AMC 读取数据

bReadDataFromAMC (U8 Command, 0, U8 ReadData_Index)

其中，

命令：与已传输数据对应的命令。

ReadData_Index：请求数据的索引值。

示例：

AMC 中的当前 ID_ERROR 值可回读如下：

bReadDataFromAMC (CMD_MCU_READ_AMC_DATA, AMC_DATA_ID_ERR)

若函数返回值为 0，则数据读取成功。请求的数据自动存储在 MRX1（高位字节）和 MRX2（低位字节）中。

命令索引

命令和相关参数如表 5 所示。

表 5. 速度积分命令和参数

命令	索引	参数
CMD_RUN	0x10	1: RUN 0: FREE
CMD_DUTY	0x12	DUTY
CMD_ANGLE_SHIFT	0x14	AS_TABLE[Index]
CMD_ALIGNMENT_TIME	0x16	ALIGNMENT_TIME
CMD_ALIGNMENT_WAIT_TIME	0x18	ALIGNMENT_WAIT_TIME
CMD_ALIGNMENT_DUTY	0x1A	ALIGNMENT_DUTY
CMD_ALIGNMENT_ANGLE	0x1C	ALIGNMENT_ANGLE
CMD_RAMPUP_START_DUTY	0x1E	RAMPUP_START_DUTY
CMD_RAMPUP_END_DUTY	0x20	RAMPUP_END_DUTY
CMD_RAMPUP_ACC_TIME	0x22	RAMPUP_ACC_TIME
CMD_FORWARD_RUN_SPEED	0x24	FORWARDRUN_SPEED
CMD_KP	0x26	KP_VALUE
CMD_KI	0x28	KI_VALUE
CMD_DIGITAL_FILTER	0x2A	DFILTER_BIT
CMD_LOSS_STEP	0x2C	LOSS_STEP_VALUE
CMD_RESOLUTION	0x2E	RESOLUTION_VALUE
CMD_MINIMUM_SPEED	0x30	MINIMUM_SPEED
CMD_RAMPUP_ID_ERR	0x32	RAMPUP_ID_ERR
CMD_LOCK_ROTOR_DELAY_TIME	0x34	LOCK_ROT_DLY_TIME
CMD_OC_MUTE_TIME	0x36	OC_MUTE_TIME
CMD_SHUNT_OFFSET	0x38	IA_SHUNT, IB_SHUNT
CMD_LINK_UP_ANGLE	0x3A	LINK_UP_ANGLE
CMD_ENABLE_WATCHDOG	0xFA	0
CMD_MCU_READ_AMC_DATA	0xFC	ReadData_Index
CMD_TRANS_COMPLETE	0xFE	0

读取命令 (CMD_MCU_READ_AMC_DATA) 及其相关数据索引如表 6 所示。

表 6. 读取 AMC 数据索引

参数	索引
AMC_DATA_ID_ERR	0x05
AMC_DATA_FW_SPEED	0x06

速度积分文件

在 MCDS IDE 软件中建立新项目时，将生成 5 个附件。表 7 显示的是这 5 个附件的文件名和函数。

表 7. 速度积分附件

文件名	说明	可编辑
AMC_SpeedIntegral.c	AMC_SpeedIntegral.c 提供通用函数，如 bWriteCmdToAMC(),bReadDataFromAMC(),btInitial_AMCToSpeedIntegral()。用户不可修改此文件。MCDS IDE 的生成代码函数一旦使用，该文件即被重写。	否
AMC_SpeedIntegral.h	AMC_SpeedIntegral.h 提供所有命令的定义以及读取数据时索引的定义。用户不可修改此文件。MCDS IDE 的生成代码函数一旦使用，该文件即被重写。	否
Parameter_SpeedIntegral.h	SI_StartUp.h 提供 AMC 启动电机时所需参数的值。这些值可利用电机调试工具优化。	是
AsTable_SpeedIntegral.h	SI_AS_Table.h 提供 AMC 控制电机时根据速度变化进行调整的角度位移值。这些值可利用电机调试工具优化。	是

上述附件的详细说明如下。

AMC_SpeedIntegral.c

为便于使用，AMC_SpeedIntegral.c 包含数个通用函数，如表 8 所示。

表 8. 通用函数在 AMC_SpeedIntegral.c 中声明

函数名称	说明
Reset_AMC( )	复位 AMC
btInitial_AMCToSpeedIntegral( )	速度积分模式对 AMC 进行初始化，并将所有参数传输到要设置的 AMC
bWriteCmdToAMC( )	将数据传输至 AMC
bReadDataFromAMC( )	从 AMC 读取数据
Delay10 μs( )	在 10 μs 延迟时间内等待 AMC
Delay1 ms( )	在 1 ms 延迟时间内等待 AMC

AMC_SpeedIntegral.c 的内容如下。

```
1   /*-------------------------------------------------------------------------
2   * Copyright 2012 Fairchild Semiconductor
3   * http://www.fairchildsemi.com
4   *-------------------------------------------------------------------------
5   */
6   #include "compiler-define.h"
7   #include "FCM8531.h"
8   #include "MSFR-define.h"
9   #include "Program.h"
10  #include "MCS51.h"
11  #include "Parameter_SpeedIntegral.h"
12  #include "AsTable_SpeedIntegral.h"
13  #include "AMC_SpeedIntegral.h"
14  //-------------------------------------------------------------------------
15  U8 _bIA_Shunt, _bIB_Shunt;
16
17  //-------------------------------------------------------------------------
18  //  Delay time Routine
19  //-------------------------------------------------------------------------
20  void Delay10 us(U16 Counter)  //Delay 10 μs
21  {
22  U16 i, k;
```

```
23
24      for(i = 0; i < Counter; i++)
25      for(k = 0; k < 16; k++);
26  }
27
28      void Delay1ms(U16 Counter)  //Delay 1 ms
29  {
30      U16 i;
31
32      for(i = 0; i < Counter; i++)
33      Delay10 μs(110);
34  }
35  //-------------------------------------------------------------------------
36  //  AMC Reset Function
37  //-------------------------------------------------------------------------
38  void Reset_AMC()
39  {
40      MD0 = AS_Table[0];
41      WRITE_MSFR(MSFR_MCNTL, 0x60);
42      WRITE_MSFR(MSFR_MCNTL, 0x20);
43  }
44  //-------------------------------------------------------------------------
45  //  Transmit Parameter that Speed Integral used
46  //-------------------------------------------------------------------------
47  SEG_BIT btTransmit_ParameterToAMC()
48  {
49      U8 bWriteCMD_Status;
50
51      bWriteCMD_Status = bWriteCmdToAMC(CMD_ALIGNMENT_TIME,
(U16)ALIGNMENT_TIME >> 8, ALIGNMENT_TIME);
52      if(bWriteCMD_Status)
53      return 1;
54
55      bWriteCMD_Status = bWriteCmdToAMC(CMD_ALIGNMENT_DUTY,
(U16)ALIGNMENT_DUTY>> 8, ALIGNMENT_DUTY);
56      if(bWriteCMD_Status)
57      return 1;
58
59      bWriteCMD_Status = bWriteCmdToAMC(CMD_ALIGNMENT_WAIT_TIME,
(U16)ALIGNMENT_WAIT_TIME >> 8, ALIGNMENT_WAIT_TIME);
60      if(bWriteCMD_Status)
61      return 1;
62
63      bWriteCMD_Status = bWriteCmdToAMC(CMD_ALIGNMENT_ANGLE, 0, ALIGNMENT_ANGLE);
64      if(bWriteCMD_Status)
65      return 1;
66
67      bWriteCMD_Status = bWriteCmdToAMC(CMD_RAMPUP_START_DUTY, 0, RAMPUP_START_DUTY);
68      if(bWriteCMD_Status)
69      return 1;
70
71      bWriteCMD_Status = bWriteCmdToAMC(CMD_RAMPUP_END_DUTY, 0, RAMPUP_END_DUTY);
72      if(bWriteCMD_Status)
73      return 1;
74
75      bWriteCMD_Status = bWriteCmdToAMC(CMD_RAMPUP_ACC_TIME,
(U16)RAMPUP_ACC_TIME >> 8, RAMPUP_ACC_TIME);
76      if(bWriteCMD_Status)
77      return 1;
78
79      bWriteCMD_Status = bWriteCmdToAMC(CMD_KP, (U16)KP_VALUE >> 8, KP_VALUE);
80      if(bWriteCMD_Status)
81      return 1;
82
```

```
83      bWriteCMD_Status = bWriteCmdToAMC(CMD_KI, KI_VALUE >> 8, KI_VALUE);
84      if(bWriteCMD_Status)
85          return 1;
86
87      bWriteCMD_Status = bWriteCmdToAMC(CMD_DIGITAL_FILTER, 0, DFILTER_BIT);
88      if(bWriteCMD_Status)
89          return 1;
90
91      bWriteCMD_Status = bWriteCmdToAMC(CMD_LOSS_STEP, (U16)LOSS_STEP_VALUE >> 8, LOSS_STEP_VALUE);
92      if(bWriteCMD_Status)
93          return 1;
94
95      bWriteCMD_Status = bWriteCmdToAMC(CMD_RESOLUTION, 0, RESOLUTION_VALUE);
96      if(bWriteCMD_Status)
97          return 1;
98
99      bWriteCMD_Status = bWriteCmdToAMC(CMD_MINIMUM_SPEED, 0, MINIMUM_SPEED);
100     if(bWriteCMD_Status)
101         return 1;
102
103     bWriteCMD_Status = bWriteCmdToAMC(CMD_RAMPUP_ID_ERR, 0, RAMPUP_ID_ERR);
104     if(bWriteCMD_Status)
105         return 1;
106
107     bWriteCMD_Status = bWriteCmdToAMC(CMD_LOCK_ROTOR_DELAY_TIME, 0, LOCK_ROTOR_DELAY_TIME);
108     if(bWriteCMD_Status)
109         return 1;
110
111     bWriteCMD_Status = bWriteCmdToAMC(CMD_OC_MUTE_TIME, 0, OVER_CURRENT_MUTE_TIME);
112     if(bWriteCMD_Status)
113         return 1;
114
115     bWriteCMD_Status = bWriteCmdToAMC(CMD_SHUNT_OFFSET, _bIA_Shunt, _bIB_Shunt);
116     if(bWriteCMD_Status)
117         return 1;
118
119     bWriteCMD_Status = bWriteCmdToAMC(CMD_TRANS_COMPLETE, 0, 0);
120     if(bWriteCMD_Status)
121         return 1;
122
123     bWriteCMD_Status = bReadDataFromAMC(CMD_MCU_READ_AMC_DATA, 0, AMC_DATA_CORE_ID);
124     if(bWriteCMD_Status)
125         return 1;
126
127     return 0;
128 }
129 //---------------------------------------------------------------------
130 //  Initial AMC Procedure
131 //---------------------------------------------------------------------
132 SEG_BIT btInitial_AMCToSpeedIntegral()
133 {
134     SEG_BIT btError_code;
135     U16 wWaitTime;
136
137     Reset_AMC();  // Reset AMC
138
139     Delay1ms(100);
140
141     btError_code = 1;
142     for(wWaitTime = 0; wWaitTime < 3000; wWaitTime++)
143         if((MRX0 & AMC_RESET_READY) == 0x40)
144         {
145             btError_code = 0;
146             break;
```

```
147         }
148     if(btError_code)   // Check Time-Out
149         return(btError_code);
150 
151     btError_code = btTransmit_ParameterToAMC();
152 
153     return btError_code;
154 }
155 //-------------------------------------------------------------------------
156 //  MPU write CMD to AMC
157 //-------------------------------------------------------------------------
158 U8 bWriteCmdToAMC(U8 bCommand, U8 bData_HB, U8 bData_LB)
159 {
160     U16 wWaitTime;
161     SEG_BIT btAMCStandby_Flag;
162 
163     btAMCStandby_Flag = 0;
164     for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
165         if((MRX0 & (AMC_PROCESSING_CMD | AMC_CALCULATING)) == 0x0)
166         {
167             btAMCStandby_Flag = 1;
168             break;
169         }
170     if(!btAMCStandby_Flag)
171         return(AMC_BUSY_TIMEOUT);
172 
173     MTX0 = bCommand | MCU_MAILBOX_INTR_STOP;
174     MTX1 = bData_HB;
175     MTX2 = bData_LB;
176     MTX0 = bCommand & MCU_MAILBOX_INTR_START;
177 
178     Delay10μs(2);
179 
180     for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
181         if((MRX0 & AMC_PROCESSING_CMD) == 0)
182             return(AMC_COMPLETED);
183 
184     return(AMC_WRITE_CMD_ERROR);
185 }
186 
187 //-------------------------------------------------------------------------
188 //  MPU Read Data from AMC
189 //-------------------------------------------------------------------------
190 U8 bReadDataFromAMC(U8 bCommand, U8 bData_HB, U8 bData_LB)
191 {
192     U16 wWaitTime;
193     SEG_BIT btAMCStandby_Flag;
194 
195     btAMCStandby_Flag = 0;
196     for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
197         if((MRX0 & (AMC_PROCESSING_CMD | AMC_CALCULATING)) == 0)
198         {
199             btAMCStandby_Flag = 1;
200             break;
201         }
202     if(!btAMCStandby_Flag)
203         return(AMC_BUSY_TIMEOUT);
204 
205     MTX0 = bCommand | MCU_MAILBOX_INTR_STOP;
206     MTX1 = bData_HB;
207     MTX2 = bData_LB;
208     MTX0 = bCommand & MCU_MAILBOX_INTR_START;
209 
210     Delay10 μs(2);
```

```
211
212     btAMCStandby_Flag = 0;
213     for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
214         if((MRX0 & AMC_PROCESSING_CMD) == 0)
215         {
216             btAMCStandby_Flag = 1;
217             break;
218         }
219     if(!btAMCStandby_Flag)
220         return(AMC_WRITE_CMD_ERROR);
221
222     for(wWaitTime = 0; wWaitTime < 600; wWaitTime++)
223         if((MRX0 & AMC_CALCULATING) == 0)
224             return(AMC_COMPLETED);
225
226     return(AMC_ERRCODE_BUSY_TOUT);
227 }
```

AMC_SpeedIntegral.h

AMC_SpeedIntegral.h 包含命令、参数、AMC数据的索引，以及速度积分算法中控制步骤的常量声明。

AMC_SpeedIntegral.h的内容如下。

其中：

16至18行指示速度积分库的版本信息；

31至37行指示AMC的状态；

41至66行指示速度积分算法的命令；以及

69至70行指示读取AMC数据的索引。

```
1  /*-------------------------------------------------------------------------
2   * Copyright 2012 Fairchild Semiconductor
3   * http://www.fairchildsemi.com
4   *-------------------------------------------------------------------------
5   */
6  #ifndef  _AMC_SpeedIntegral_h_
7  #define  _AMC_SpeedIntegral_h_
8
9  #define SetReg(reg, data)    reg |= (data)
10 #define ClrReg(reg, data)    reg &= ~(data)
11
12 #define READ_MSFR(addr, data)   (MSFRADR = addr, data = MSFRDAT)
13 #define WRITE_MSFR(addr, data)  (MSFRADR = addr, MSFRDAT = data)
14
15 //AMC Core Information define
16 #define SPD_INTEGRAL_AMC_CORE_ID  0x0002
17 #define SPD_INTEGRAL_AMC_CORE_VER 0x0108
18 #define SPD_INTEGRAL_AMC_CORE_ID_IEC     0x0020
19
20 //AMC specific MSFR address
21 #define MSFR_OMEGAH               0x06  // AMC OMEGA high byte
22 #define MSFR_OMEGAL               0x07  // AMC OMEGA low byte
23 #define MSFR_THETA                0x3C  // AMC Theta
24 #define CMD_AMC_CONTROL           0x10
25
26 // MCU mail box status
27 #define MCU_MAILBOX_INTR_STOP     0x01
28 #define MCU_MAILBOX_INTR_START    0xFE
29
```

```
// MRX0 bit description
#define AMC_PROCESSING_CMD        0x02
#define AMC_CALCULATING           0x04
#define AMC_FAULT                 0x08
#define AMC_STARTUP_READY         0x10
#define AMC_LOCK_MOT_READY        0x20
#define AMC_RESET_READY           0x40
#define SAFETY_WARNING            0x80

// MCU CMD list//
// Write data CMD
#define CMD_RUN                   0x10
#define CMD_DUTY                  0x12
#define CMD_ANGLE_SHIFT           0x14
#define CMD_ALIGNMENT_TIME        0x16
#define CMD_ALIGNMENT_WAIT_TIME   0x18
#define CMD_ALIGNMENT_DUTY        0x1A
#define CMD_ALIGNMENT_ANGLE       0x1C
#define CMD_RAMPUP_START_DUTY     0x1E
#define CMD_RAMPUP_END_DUTY       0x20
#define CMD_RAMPUP_ACC_TIME       0x22
#define CMD_FORWARD_RUN_SPEED     0x24
#define CMD_KP                    0x26
#define CMD_KI                    0x28
#define CMD_DIGITAL_FILTER        0x2A
#define CMD_LOSS_STEP             0x2C
#define CMD_RESOLUTION            0x2E
#define CMD_MINIMUM_SPEED         0x30
#define CMD_RAMPUP_ID_ERR         0x32
#define CMD_LOCK_ROTOR_DELAY_TIME 0x34
#define CMD_OC_MUTE_TIME          0x36
#define CMD_SHUNT_OFFSET          0x38
#define CMD_LINK_UP_ANGLE         0x3A
#define CMD_ENABLE_WATCHDOG       0xFA
#define CMD_TRANS_COMPLETE        0xFE
//Read data CMD
#define CMD_MCU_READ_AMC_DATA     0xFC

// MCU Read Data Index
#define AMC_DATA_ID_ERR           0x05
#define AMC_DATA_FW_SPEED         0x06

// AMC status code
#define AMC_COMPLETED             0x00
#define AMC_BUSY_TIMEOUT          0x01
#define AMC_WRITE_CMD_ERROR       0x02
#define AMC_RUNCMD_TIMEOUT        0x03

#define AMC_ERRCODE_PASS          0x80
#define AMC_ERRCODE_BUSY_TOUT     0x81
#define AMC_ERRCODE_CMD_ERROR     0x82
#define AMC_ERRCODE_RUNCMD_TOUT   0x83

//-----------------------------------------------------------------------
//  Export Function
//-----------------------------------------------------------------------
extern U8 _bIA_Shunt, _bIB_Shunt;

extern SEG_BIT btInitial_AMCToSpeedIntegral();
extern SEG_CODE U8 AS_Table[AS_TABLE_COUNT];
extern U8 bWriteCmdToAMC(U8 bCommand, U8 bData_HB, U8 bData_LB);
extern U8 bReadDataFromAMC(U8 bCommand, U8 bData_HB, U8 bData_LB);
extern void Reset_AMC();
extern void Delay10 μs(unsigned int Counter);
```

```
94 extern void Delay1 ms(unsigned int Counter);
95
96 #endif
```

Parameter_SpeedIntegral.h

由于速度积分算法中电机的启动步骤由 AMC 控制，用户必须向 AMC 提供相应参数值才可启动电机。Parameter_SpeedIntegral.h 提供 AMC 启动电机所需的参数值。该参数值可利用 MCDS IDE 电机调试工具优化。

Parameter_SpeedIntegral.h 的内容如下。

其中，

第 10 行表示每个 AS 调节的速度间隔；

第 11 行表示电机的极数；

第 13 行表示 AS 表的字节数；

第 15 行至第 23 行表示启动参数；

第 26 行至第 34 行表示速度积分算法的 P-I 控制参数；以及

第 37 行至第 51 行表示速度积分算法的正向和反向启动参数。

```
1  /*-------------------------------------------------------------------------
2   * Copyright 2012 Fairchild Semiconductor
3   * http://www.fairchildsemi.com
4   *-------------------------------------------------------------------------
5   */
6
7  #ifndef  _Parameter_SpeedIntegral_h_
8  #define  _Parameter_SpeedIntegral_h_
9
10 #define SPEED_INTERVAL            0x32
11 #define MOTOR_POLE                0x8
12 //-------Parameter setup for AMC ----------------------
13 #define AS_TABLE_COUNT            0x18
14
15 #define ALIGNMENT_DUTY            0x14
16 #define ALIGNMENT_TIME            0x64
17 #define ALIGNMENT_ANGLE           0x0
18 #define ALIGNMENT_WAIT_TIME       0x0
19 #define RAMPUP_START_DUTY         0x20
20 #define RAMPUP_END_DUTY           0x30
21 #define RAMPUP_ACC_TIME           0x19
22 #define RAMPUP_ID_ERR             0x2
23 #define MINIMUM_SPEED             0x0
24
25 // SpeedIntegral parameter
26 #define KP_VALUE                  200
27 #define KI_VALUE                  256  // 256 = 1, 128 = 0.5 , 64 = 0.25
28 #define DFILTER_BIT               1
29 #define LOSS_STEP_VALUE           5    // Speed Integral limit value = Limit_Value * 65536
30 #define RESOLUTION_VALUE          16
31
32 //$ SpeedIntegral parameter for IEC-60730
33 #define LOCK_ROTOR_DELAY_TIME     10  // Resolution: 1 sec/step, 1 means 1 s
34 #define OVER_CURRENT_MUTE_TIME    10  // Resolution: 0.5 sec/step, 1 means 0.5 s
35
36 // SpeedIntegral parameter for Forward wind and Reverse wind start-up
37 #define MOT_SHUNTCUR_TOLERANCE    2      // Shunt current tolerance
38 #define MOT_CURRENT_CHK_TIME      500    // Check time = 500 * 5 µs = 2.5 sec
39 #define MOT_CURRENT_CHK_TIMEOUT   60000  // (1/PWM frequency)/step, (1/15 KHz) * 60000 = 4 sec
40 #define MOT_RUNNING_CHK_TIMEOUT   60000  // 5 µs/step
41 #define REV_SPEED_CHK_TIMEOUT     60000  // 5 µs/step
```

```
42 #define REV_RUNNING_CHK_TIMEOUT   10000  // 5 µs/step
43 #define OMEGA_TO_DUTY_RATIO       140
44 #define FWD_LINKUP_ANGLE_CW       10
45 #defineFWD_LINKUP_ANGLE_CCW       100
46 #defineREV_LINKUP_ANGLE_CW        40
47 #defineREV_LINKUP_ANGLE_CCW       80
48 // SpeedIntegral parameter for Reverse wind startup
49 #define REV_RAMP_UP_DUTY          132
50 #define REV_RAMP_UP_TIME          3000
51 #define REV_RAMP_UP_IDERR         2
52
53 #endif
```

AsTable_SpeedIntegral.h

利用积分算法控制电机时，必须调节超前角度参数，使电流和反电动势相位同步，以便获得最佳效率。调节后的角度随电机速度和负载而变。AsTable_SpeedIntegral.h 提供生成后的 AS_Table。使用 MCDS IDE 软件的电机调试功能，可修改 AS_Table。

AsTable_SpeedIntegral.h 内容如下。

```
1  /*-------------------------------------------------------------------------
2   * Copyright 2013 Fairchild Semiconductor
3   * http://www.fairchildsemi.com
4   *-------------------------------------------------------------------------
5   */
6
7  #ifndef  _AsTable_SpeedIntegral_h_
8  #define  _AsTable_SpeedIntegral_h_
9
10 // Motor AS Table (Max. Speed: 1200, Interval Speed: 50)
11 SEG_CODE U8 AS_Table[AS_TABLE_COUNT]={
12     3, 4, 5, 6, 8, 10, 12, 14, 16, 18,
13     20, 22, 24, 26, 28, 30, 32, 33, 34, 35,
14     36, 37, 38, 38
15     };
16
17 #endif
```

如何使用速度积分

前文讨论了 FCM8531 结构、速度积分算法、参数、命令和通信协议。另外，本节将详细说明如何使用速度积分库部署无传感器电机控制，包括：电机启动、旋转方向控制、旋转速度控制、读取旋转速度、错误判断、看门狗设置、速度闭环控制、FG 信号输出、正向启动和反向启动。

电机启动

在速度积分库中，电机启动由 AMC 处理。用户仅设置启动参数而不用开发程序；AMC 会根据设置的参数启动电机。

MCDS IDE 生成文件之前就已加入 AMC 设置的初始程序代码，包括将参数传输至 AMC 的程序代码。因此，只要 MCNTL 位 0 设为 1 并且通过 CMD_RUN 将 Run=1 传输至 AMC，电机即可启动。反过来，只要将 Run=0 传输至 AMC，电机便停止工作。启动和停止电机的示例程序代码如下。

示例代码（启动电机）

```
#define MOTOR_FREE                  0x00
#define MOTOR_RUN                   0x01
#define MOTOR_CW                    0x02
#define USER_DEFINE_SVMTABLE        0x20
//User program start here. (04)
WRITE_MSFR(MSFR_MCNTL,
USER_DEFINE_SVMTABLE|MOTOR_RUN|MOTOR_CW);
bWriteCmdToAMC(CMD_RUN, 0, MOTOR_RUN);
//User program end here. (04)
```

Example Code (Stopping Motor)

```
//User program start here. (04)
WRITE_MSFR(MSFR_MCNTL,
USER_DEFINE_SVMTABLE|MOTOR_CW);
bWriteCmdToAMC(CMD_RUN, 0, MOTOR_FREE);
//User program end here. (04)
```

旋转方向

如需控制电机的旋转方向，只需改变 MCNTL 位 1，使 PWM 输出序列发生变化即可。

若 MCNTL 位 1 = 1，则 PWM 输出序列为：W -> V -> U;

若 MCNTL 位 0 = 1，则 PWM 输出序列为：U -> V -> W.

示例程序代码如下。

若 MCNTL 位 1 = 1：

```
#define MOTOR_FREE                  0x00
#define MOTOR_RUN                   0x01
#define MOTOR_CW                    0x02
#define USER_DEFINE_SVMTABLE        0x20
//User program start here. (04)
WRITE_MSFR(MSFR_MCNTL,
USER_DEFINE_SVMTABLE|MOTOR_RUN|MOTOR_CW);
bWriteCmdToAMC(CMD_RUN, 0, MOTOR_RUN);
//User program end here. (04)
```

若 MCNTL 位 0 = 1：

```
//User program start here. (04)
WRITE_MSFR(MSFR_MCNTL,
USER_DEFINE_SVMTABLE|MOTOR_RUN);
bWriteCmdToAMC(CMD_RUN, 0, MOTOR_RUN);
//User program end here. (04)
```

旋转速度控制

由于电机启动由 AMC 处理，用户只能在启动步骤完成后通过更改占空比数值控制电机速度。MRX0 的位 4 (STR_Rdy) 可用于指示 AMC 是否已完成启动步骤。STR_Rdy=1 表示启动步骤已完成，因此 MCU 可读取当前占空比值，并改变该值以控制旋转速度。

若占空比数值瞬间增大或减小，则可能会导致旋转错误，如损失步长。建议平滑调节占空比数值，确保电机正常旋转。

改变占空比数值以及提升/降低旋转速度的示例程序代码如下。

```
#define ACCELERATE_SPEED          50
#define AMC_STARTUP_READY         0x10
#define TARGET_DUTY               0x80

if(MRX0 & AMC_STARTUP_READY)
{
    READ_MSFR(MSFR_DUTYA,wCurrrent_Duty.U8[MSB]);
    READ_MSFR(MSFR_DUTYAL,wCurrrent_Duty.U8[LSB])
;
    wCurrrent_Duty.U16 >>= 7;
    if(--_wAccelerate_Cnt <= 0)
    {
        AdjustDuty();
        _wAccelerate_Cnt=ACCELERATE_SPEED;
    }
    bWriteCmdToAMC(CMD_Duty,
        wDuty_Buffer.U8[MSB],wDuty_Buffer.U8[LSB]
);
}
```

```
//-----------------------------------------------
//  Adjust Duty
//-----------------------------------------------
void AdjustDuty()
{
   if(wCurrrent_Duty.U16< wTarget_Duty)
       wCurrrent_Duty.U16++;
   elseif(wCurrrent_Duty.U16> wTarget_Duty)
       wCurrrent_Duty.U16--;
   wDuty_Buffer.U16 = wCurrrent_Duty.U16;
}
```

读取旋转速度

电机在无传感器控制的情况下工作时，无法从传统霍尔传感器反馈获得旋转速度。在速度积分算法中，通过对旋转角速度 (ω) 进行积分可估算转子位置，然后进行变换可获得旋转速度。

MSFR 的 OMEGAH (0x06H) 和 OMEGAL (0x07H) 分别是旋转速度 ω 的高位字节和低位字节，经读取后可根据下列公式计算频率：

$$F = \omega / ((1/ f_S) \times 2^{(32 - RESOLUTION\_VALUE)}) \quad (1)$$

其中：

ω: 旋转速度；

$\mathbf{f_S}$: 速度积分的工作频率 (8 kHz)；以及

RESOLUTION_VALUE: 角度计算的分辨率。

最后，经转换后可获得旋转速度：

$$RPM = F \times 60 / （极数 / 2） \quad (2)$$

示例：

假设对于 8 极电机：RESOLUTION_VALUE=16，读取的 ω=0x01FF。电机的转速为：

$F = 511 / ((1 / (8 \times 10^3)) \times 2^{(32 - 16)})$

$F = 62.4$ Hz

$RPM = 62.4 \times 60 / 4 = 936$

最后，电机的转速为 936 rpm。

示例代码

```
UU16 _wSpeedBuffer, _wSpeedFREQ;

UU32 dwOMEGABuffer;

READ_MSFR(MSFR_OMEGAH,_wSpeedBuffer.U8[MSB]);
READ_MSFR(MSFR_OMEGAL,_wSpeedBuffer.U8[LSB]);
dwOMEGA_Buffer.U32 = _wSpeedBuffer.U16;
dwOMEGA_Buffer.U32 = dwOMEGA_Buffer.U32*1000 /
                        819;
_wSpeed_FREQ.U16 = dwOMEGA_Buffer.U16[1];
```

AMC 错误判断

在控制电机过程中，若发生电机损失步长，速度积分算法的速度值将不断累积。一旦速度积分值超过 LOSS_STEP_VALUE 的预设值，且 MRX0 位 3（AMC_Fault 标识）= 1，MCU 就会检测 AMC_Fault 标识，并确定下一步。

示例代码

```
#define AMC_FAULT     0x08

if(MRX0 & AMC_FAULT)
{
   WRITE_MSFR(MSFR_MCNTL,
       USER_DEFINE_SVMTABLE|MOTOR_FREE);
   bWriteCmdToAMC(CMD_RUN, 0, MOTOR_FREE);
}
```

看门狗设置

检测到非 IEC 60730-1 B 类速度积分库时，禁用看门狗默认状态。

如需使能看门狗功能，可以先在 MCU 中使能看门狗功能，然后发送“命令 (CMD_ENABLE_WATCHDOG)”到 AMC。

当选中 IEC 60730-1 B 类速度积分库时，看门狗功能由 AMC 自身使能，且无法禁用。

示例代码

```
#define INITIAL_WDTREL0x80

void main(void)
{
   WDTREL = INITIAL_WDTREL;
   WDT = 0;
   SWDT = 0;
   bWriteCmdToAMC(CMD_ENABLE_WATCHDOG, 0, 0);
   bWriteCmdToAMC(CMD_AMC_CONTROL, 0, 0);
   for(;;)
   {
       WDT = 1;
       SWDT = 1;
   }
}
```

速度闭环

如需部署速度闭环控制，请参考 MCDS 提供的示例代码。角速度从 AMC 中读取，转换为电气频率并馈入 PI 控制器。SPEED_CLOSE_LOOP_ENABLE 在 “Program.h” 中定义，使能速度控制。

7 个目标速度在 Program.c 的 VSPVSSpeed_Table[0x7] 中声明，并对应于 0.5~4 V VSP 电压范围，间隔为 0.5 V。

Program.h

```
#ifndef  _PROGRAM_H_
#define  _PROGRAM_H_

//User program start here.(01)
#define Init_CKCON  SVM_data[0]
//Motor State
#define MOTOR_INITIAL        0
#define MOTOR_STATIC         1
#define MOTOR_FORWARD        2
#define MOTOR_RUNNING        3
#define MOTOR_RAMPUPDONE     4
#define MOTOR_REVERSEWINDSTARTUP     -1
#define MOTOR_REVERSEWINDSPEEDDOWN   -2
#defineMOTOR_REVERSEWINDRUNNING      -3

//User define area for MCU use
#define HVDD_MINIMUM_VOLTAGE          0x60
#define HVDD_RECOVRY_MIN_VOLTAGE      0x70
#define VSP_THRESHOLD_VOLTAGE         0x1E
#define HVDD_THRESHOLD_VOLTAGE        0x60
#define AMC_ERROR_CHECK_COUNT         50000
#define ERROR_RECOVERY_TIMES          5

#define HVDD_OVP_LATCH                0xED
#define HVDD_OVP_RELEASE              0xD2
#define SPM_OTP_LATCH                 0xDF
#define SPM_OTP_RELEASE               0x95
//User define area for MCU Speed control use
#define SPEED_CLOSE_LOOP_ENABLE
#define CLOSE_LOOP_DUTY_ACC_SPD      10
#define OPEN_LOOP_DUTY_ACC_SPD       80
#defineSPEED_DOWN_ACC_SPD            5
```

Program.c

```
// VSP to Speed Table ()
SEG_CODE U16 VSPVSSpeed_Table[0x7] = {200, 400,
600, 800, 900, 1000, 1200};
```

FG（频率生成）输出信号

每隔 30 电气角度，AMC 便触发一次 EX10 中断。用户可在 EX10 中断服务例程中切换空闲 I/O 引脚，以生成 FG。

通过读取 FG 信号可获取电机速度。示例代码能在每次电气旋转时产生 6 个脉冲。

示例代码

```
#define INITIAL_IEN2 0x1E
#define INITIAL_HALINT 0x01
SEGBIT _btFGTriggerred = 1;

Void Initial_Interrupt()
{
    IP0 = INITIAL_IP0;
    IP1 = INITIAL_IP1;

    EX0 = 0;
    EX1 = 0;
    IEN1 = INITIAL_IEN1;
    IEN2 = INITIAL_IEN2;
}
void Initial_Hall()
{
    WRITE_MSFR(MSFR_HALMUX, INITIAL_HALMUX);
    WRITE_MSFR(MSFR_HALFLT, INITIAL_HALFLT);
    WRITE_MSFR(MSFR_HALINT, INITIAL_HALINT);
}

Interrupt(ISR_Hall, VECTOR_EX10)
{
    U8 T;
    T = MSFRADR;
    If(btFGTriggerred)
    {
       P1_3 ^= 1;
       btFGTriggerred = 0;
    }
    else
       btFGTriggerred = 1;
    MSFRADR = T;
}
```

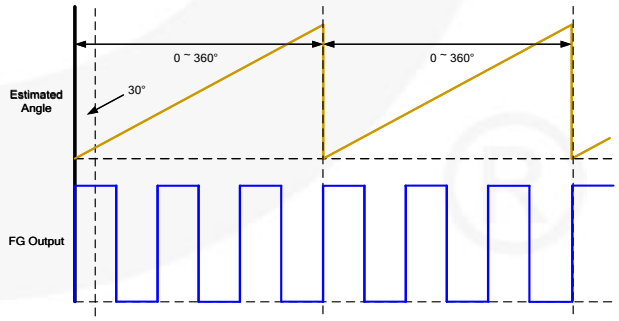


图 26. FG 输出信号

正向和反向启动

除了从静止状态启动电机外，库还提供强大的算法，可从正向旋转和反向旋转启动电机。

可配置多个参数，用户可在各种不同应用中采用这些参数，实现正向和反向的平滑启动。

下图显示正向和反向启动的流程图以及相关参数。

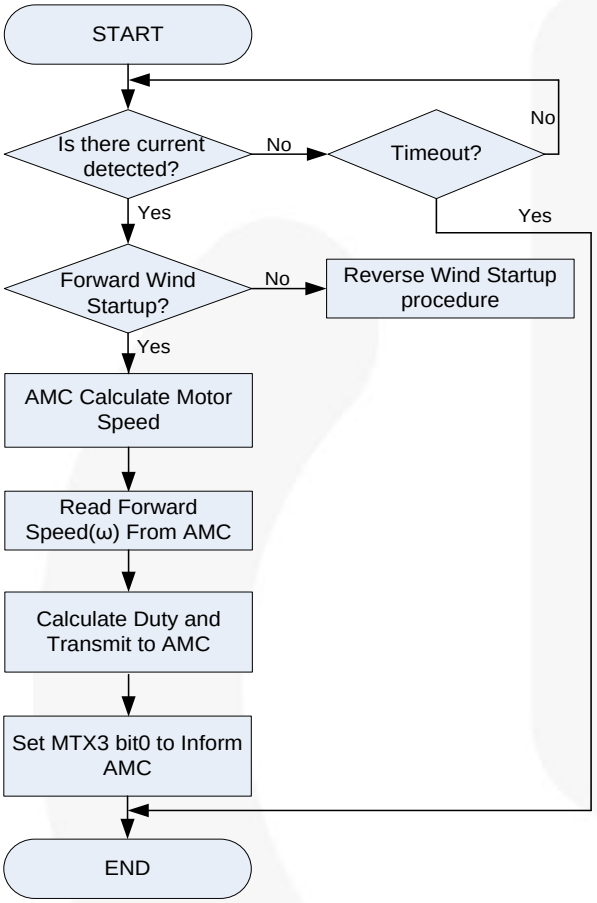


图 27. 正向启动

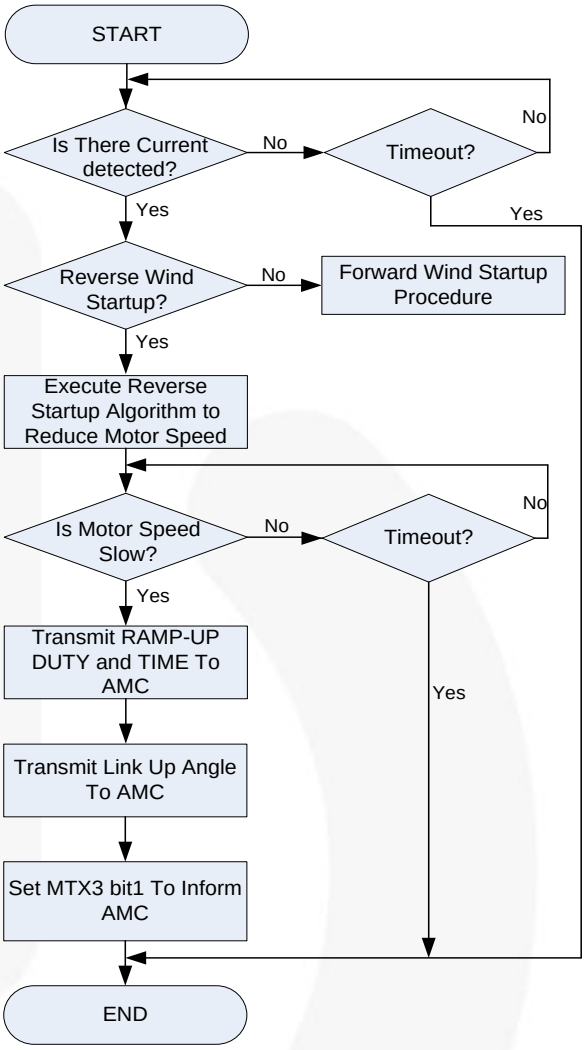


图 28. 反向启动

正向和反向启动参数

库提供 4 个参数，用于正向和反向启动调试。这些参数应当根据不同应用和情况来微调，以获得最佳性能。

表 9. 正向和反向启动参数列表

参数名称	说明	默认值	范围
MOT_RUNNING_CHK_TIMEOUT	通过读取电流检测电阻上的相位电流，可检测电机是否工作。MOT_RUNNING_CHK_TIMEOUT 用来定义时间窗口。如果在时间窗口内无电流回读，则电机将被视为处于静止状态。窗口越大，预期的正向启动速度越低，反之亦然。其单位为 5 μs。	60000	1~65535
OMEGA_TO_DUTY_RATIO	为实现平滑转变，AMC 需具有适当的比率 ω 和占空比，由 OMEGA_TO_DUTY_RATIO 定义。若转变时电流变化较大，则比率也应增大。若电机速度突然下降，则比率也下降。如果转变曲线是平滑的，则该参数的默认值一般不需要改变。	140	1~255
REV_RAMPUP_DUTY	定义使电机反向并从反向转动进入正向旋转所需的占空比。	132	1-255
REV_RAMPUP_TIME	定义使电机反向并从反向转动进入正向旋转所需的时间。REV_RAMPUP_TIME 不应超过 5 s。单位为 1 ms。	5000	1~5000

固件示例

主程序

FCM8531 的 AMC 和 MCU 并行运行。一旦 MCU 完成 AMC 初始化过程，MCU 便可开始发送命令，启动电机，并控制转动速度，或在软件环路中连续调节电流相位。

下文显示示例程序：

```
C:\Fairchildsemi\MCDS2\Examples\KeilC\Example_SpeedIntegral_UL\
```

若已安装 MCDS IDE 软件。示例程序必须与 FCM8531 评估板一起使用，以保证正常工作。
http://www.fairchildsemi.com/applications/motor-control/solutions/bldc-pmsm-controller/.

图 29 显示的是 MCU 主程序中的初始化过程流程图。

图 30 显示的是 MCU 主程序中的控制环路流程图。

初始化过程初始化 PWM 频率、保护点、中断等参数。

该过程包含校准电流检测电阻、复位 AMC、发送电流检测电阻的校准值和相关参数至 AMC、测试 MCU 和 AMC 之间的通信、检查库信息，以及发送命令 FREE 以解锁 IEC 60730-1 B 类保护，从而进入主控环路。

初始化过程仅在主程序启动时执行一次，之后执行后续控制环路。

控制环路是一种无限环路结构，可响应评估板的开始切换、正向、反向和停止状态，并开始执行相应的启动过程。一旦完成电机启动，便可控制旋转速度，调节超前角。使能 AMC_Fault 或各种保护机制。若发生异常情况，则 MCU 可发出 FREE 命令，使电机停止转动。异常情况移除后，可发送 RUN 命令，使电机再次转动。

选中 IEC 60730-1 B 类库时，看门狗使能。为了避免系统由于看门狗定时器溢出而复位，应定期复位看门狗定时器。

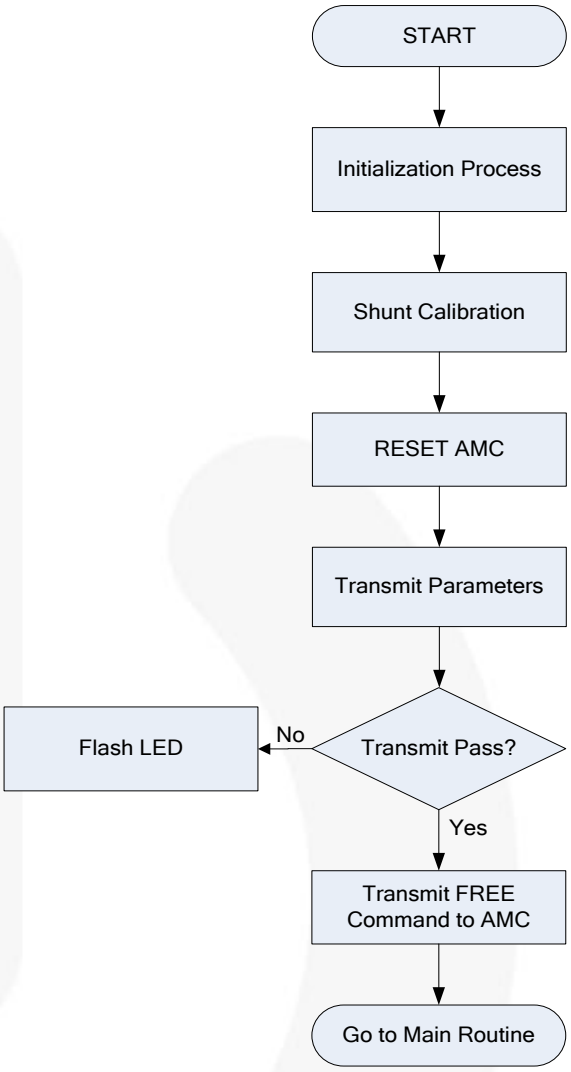


图 29. 初始化流程

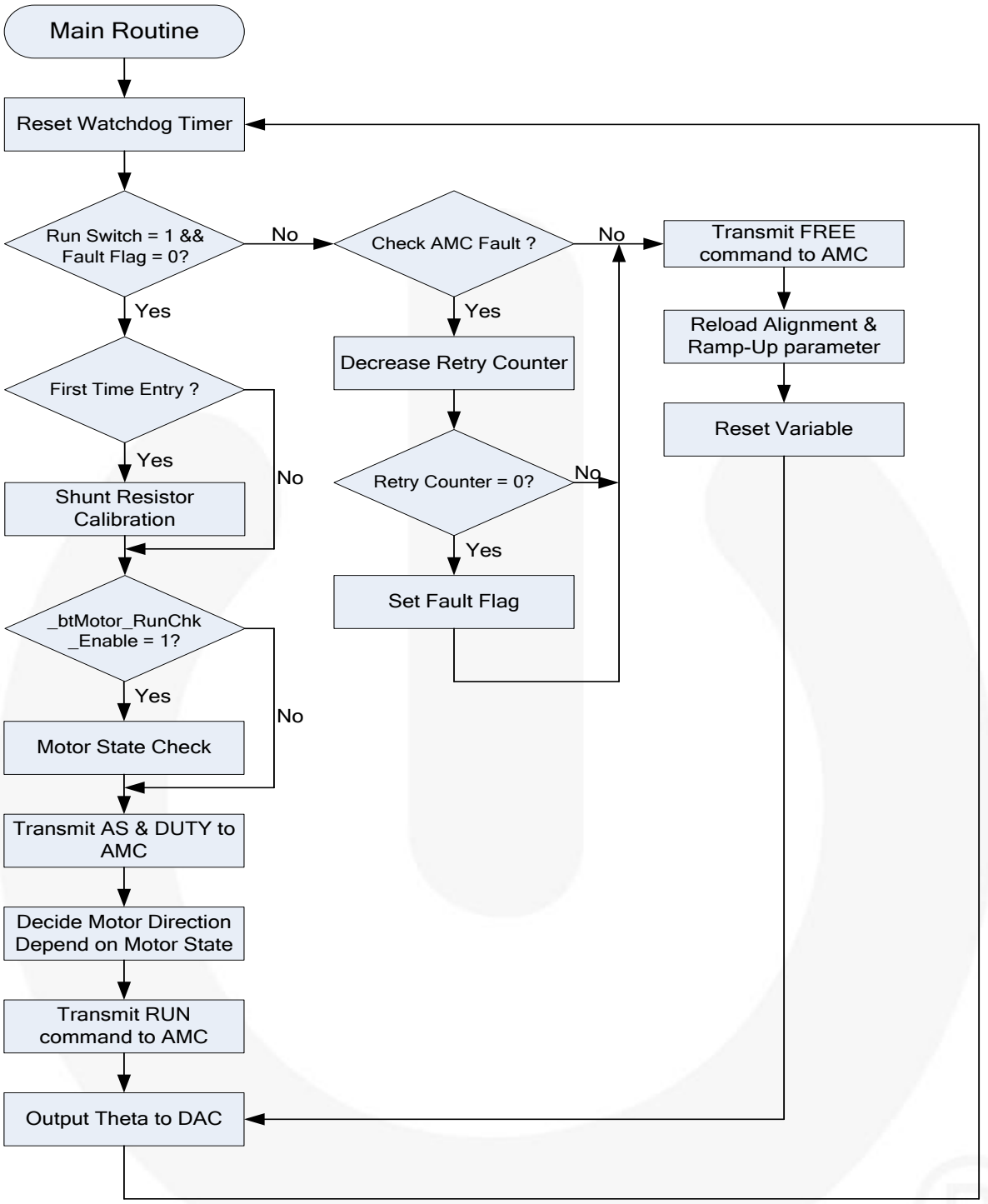

Main Routine
Reset Watchdog Timer
Run Switch = 1 && Fault Flag = 0?
No
Check AMC Fault ?
No
Transmit FREE command to AMC
Yes
Yes
Reload Alignment & Ramp-Up parameter
First Time Entry ?
Decrease Retry Counter
Reset Variable
Yes
No
Retry Counter = 0?
No
Shunt Resistor Calibration
Yes
Set Fault Flag
_btMotor_RunChk _Enable = 1?
Yes
No
Motor State Check
Transmit AS & DUTY to AMC
Decide Motor Direction Depend on Motor State
Transmit RUN command to AMC
Output Theta to DAC

图 30. 主控制环路

主程序代码如下。

```
/*---------------------------------------------------------------------
 * Copyright 2012 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *
 *
 *
 * Create Date: 2013/11/11 18:18:11
 * Create by MCDS IDE Version 2.00
 *---------------------------------------------------------------------
 */

//---------------------------------------------------------------------
//  Includes
//---------------------------------------------------------------------
#include "compiler-define.h"
#include "FCM8531.h"
#include "MSFR-define.h"
#include "Program.h"
#include "MCS51.h"
#include "Parameter_SpeedIntegral.h"
#include "AMC_SpeedIntegral.h"

//User program start here.(01)
#if defined SDCC
#include "MCS51.c"
#include "MotorCtrl.c"
#endif
#include "MotorCtrl.h"
#include "Utility_SpeedIntegral.h"
#include "Utility_SpeedIntegral_Advance.h"
//User program end here.(01)

//---------------------------------------------------------------------
//  Initial Const
//---------------------------------------------------------------------
#define INITIAL_SLEEP     0x04  // Set use 30 MHz

//User program start here.(09)

//User program end here.(09)

//---------------------------------------------------------------------
//  Main Routine
//---------------------------------------------------------------------
#if defined SDCC
    SEG_BIT __at (0x30) btInit_AMCStatus;
#else
    SEG_BIT btInit_AMCStatus;
#endif
void main(void)
{
    //User variable start here.(0A)
    //User variable end here.(0A)
    CKCON = 0;
    WRITE_MSFR(MSFR_SLEEP, INITIAL_SLEEP);
    MD0 = Init_SVM_Table;

    Initial_Procedure();
    //User program start here.(02)

    EA = 1;
    Initial_MCU_Variable();
```

```
    Shunt_Calibration();

    //User program end here.(02)
    EA = 1;
    btInit_AMCStatus = btInitial_AMCToSpeedIntegral();

    //User program start here.(03)

    if(btInit_AMCStatus)
        Fault_Led_Flash();

    Delay1ms(100);

    //User program end here.(03)
    bWriteCmdToAMC(CMD_AMC_CONTROL, 0, 0);
    for(;;)
    {
        WDT_Clear();
        //User program start here.(04)
        if(_btMotorRun_Enable && (!_btDIRChg_Flag) && _btHVDD_Level_Pass && !_btFaultOccured_Flag)
        {
            if(_bMotor_State == MOTOR_INITIAL)
            {
                Shunt_Calibration();
                bWriteCmdToAMC(CMD_SHUNT_OFFSET, _bIA_Shunt, _bIB_Shunt);
            }
            if(_btMotor_RunChk_Enable)
            {
                Check_Motor_Running();
                if(_btBrake_Enable)
                    Motor_Brake();
            }
            Transmit_New_DUTYANDAS_To_AMC();
            Decide_Motor_Rotation();
            Set_Motor_State(MOTOR_RUN);
        }
        else
        {
            Check_AMCFaultRetryCounter();
            Set_Motor_State(MOTOR_FREE);
            Reload_Alignment_And_RampUp_Parameter();
            Reset_MCU_Variable();
        }
        OutputThetatoDAC();
        //User program end here.(04)
    }
    //User program start here.(05)

    //User program end here.(05)
}
```

定时器 0 程序

定时器 0 用来监控硬件开关状态并检查故障标识。图 31 显示的是使用定时器 0 的流程图。

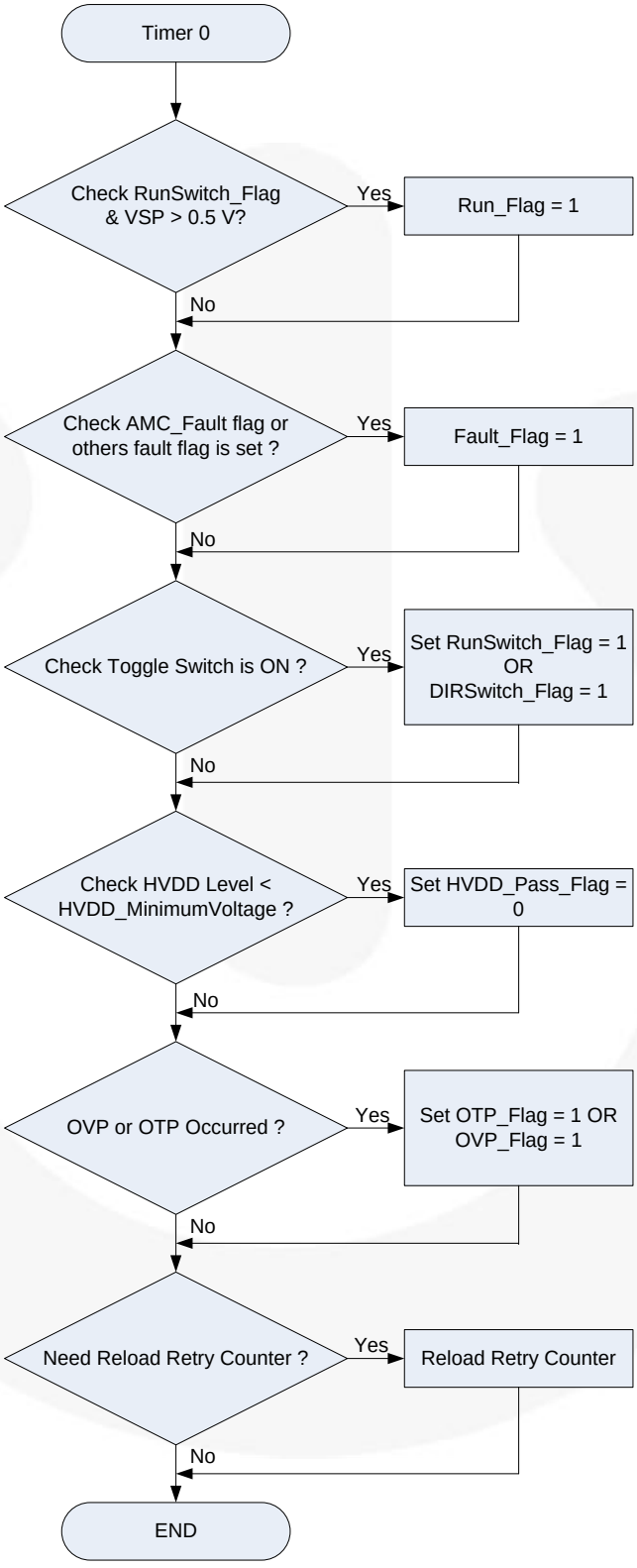


图 31. 定时器 0 流程图

定时器 0 示例程序代码如下。

```
/*-------------------------------------------------------------------------
 * Copyright 2012 Fairchild Semiconductor
 * http://www.fairchildsemi.com
*-------------------------------------------------------------------------
*/
//-------------------------------------------------------------------------
//  Interrupt Service Routine
//-------------------------------------------------------------------------
// Timer0 ISR
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)
    _bTimer0Counter++;
    _wVsp_Input.U16 >>= 7;
    wVSPFilter.U32 = 3 * wVSPFilter.U32 + _wVsp_Input.U16;
    wVSPFilter.U32 >>= 2;
    wVSPResult = wVSPFilter.U16[LSB];

    _btMotorRun_Enable = ((_btRUNCmd_Flag == 0) & (wVSPResult >= VSP_THRESHOLD_VOLTAGE));

    _btFaultOccured_Flag = (((MRX0 & AMC_FAULT) == 0x08) || (_bRecovery_Times == 0)
                    ||(_btOverVoltage_Protect|_btOverTemperature_Protect)||_btShortCircuit_Protect);

    Toggle_Switch_Check();

    Check_HVDD_UV();

    Check_OVP_OTP();

    Check_Reload_RetryCounter();

    if(_bT0DownCounter)  _bT0DownCounter--;

    TR0 = 1;
    //User program end here.(13)
    MSFRADR = bBackupADR;

}
```

定时器 1 程序

定时器 1 用来控制电机速度。示例程序的流程图如所示图 32。

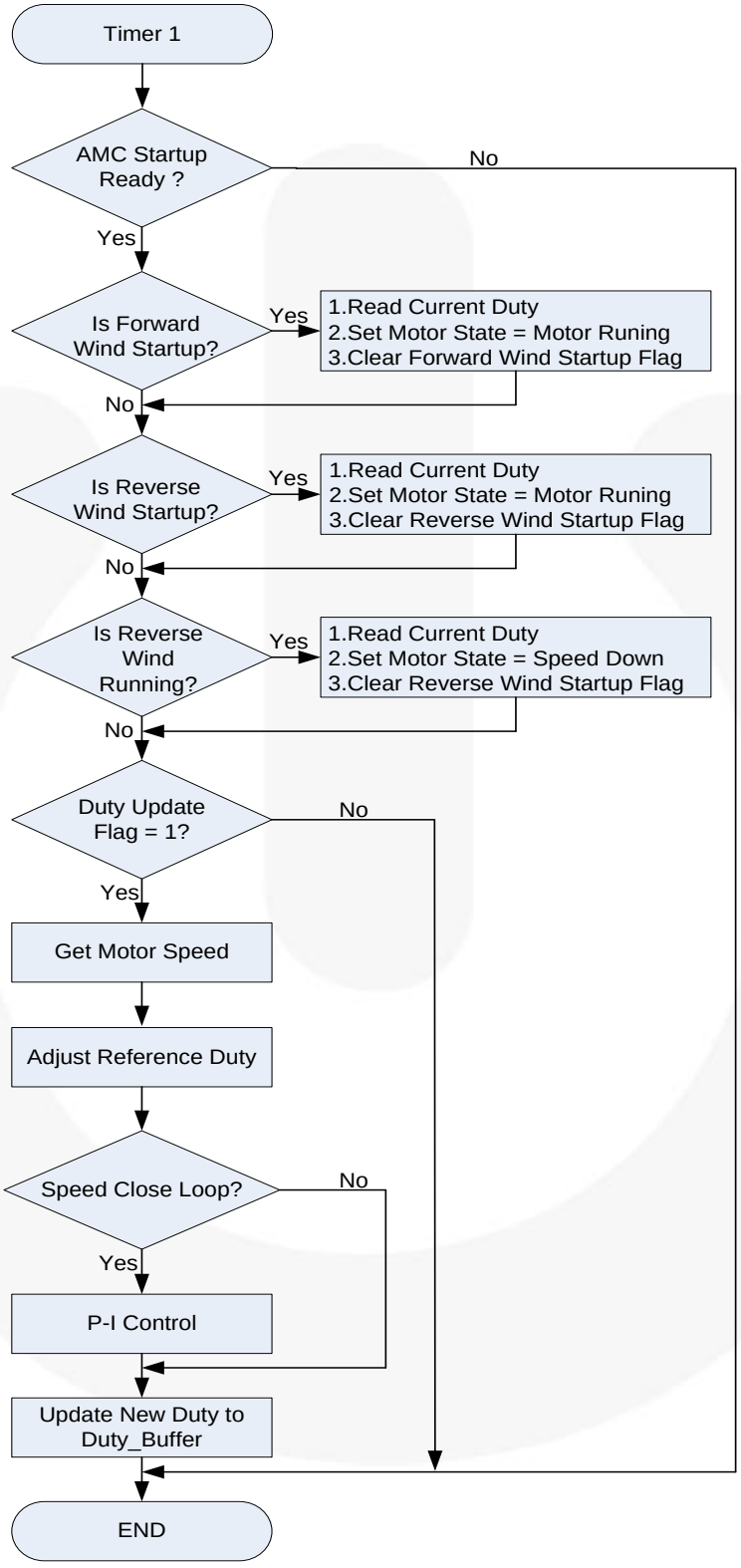


图 32. 定时器 1 流程图

仅在 AMC 完成启动过程后，速度控制才可用。可从 AMC 读取当前速度。如果使能了闭环控制，则当前速度与目标速度的速度差可用来进行 PI 控制，更新占空比值。如果未使能闭环速度控制，则占空比值直接响应 ADC 的 VSP 输入电压。

定时器 1 示例程序代码如下。

```
/*-------------------------------------------------------------------------
 * Copyright 2012 Fairchild Semiconductor
 * http://www.fairchildsemi.com
*-------------------------------------------------------------------------
*/
//-------------------------------------------------------------------------
//  Interrupt Service Routine
//-------------------------------------------------------------------------
// Timer1 ISR
INTERRUPT(ISR_T1, VECTOR_ET1)

{
    U8 bBackupADR;
    UU16 V;
    //User variable start here.(3 A)
#ifdef SPEED_CLOSE_LOOP_ENABLE
    UU16 wSpeedError_Temp;
#endif
    //User variable end here.(3 A)

    bBackupADR = MSFRADR;
    //Initial Timer1
    V.U16 = INITIAL_T1_INTERVAL;
    TH1 = V.U8[MSB];
    TL1 = V.U8[LSB];
    //User program start here.(14)
    _wTarget_Duty = wVSPResult;
    _wTarget_Speed_Value = wGet_TargetSpeed(_wTarget_Duty);

    if((MRX0 & AMC_STARTUP_READY) == 0)
    {
       GetMotor_Speed();
       ReadCurrentDuty();
    }
    else
    {
       if((_bMotor_State == MOTOR_FORWARD) || (_bMotor_State == MOTOR_STATIC))
       {
           ReadCurrentDuty();
           _bMotor_State = MOTOR_RUNNING;
           MTX3 &= CLR_FORWARDSTARTUP_STATE;
       }
       if(_bMotor_State == MOTOR_REVERSEWINDSTARTUP)
       {
           ReadCurrentDuty();
           _bMotor_State = MOTOR_RUNNING;
           MTX3 &= CLR_BACKWARDSTARTUP_STATE;
       }
       if(_bMotor_State == MOTOR_REVERSEWINDRUNNING)
       {
           ReadCurrentDuty();  // If yes, read current duty from MSFR
           _bMotor_State = MOTOR_REVERSEWINDSPEEDDOWN;
           MTX3 &= CLR_FORWARDSTARTUP_STATE;
       }

       if(_btMotor_DutyUpdate_Enable == 0)
       {
```

```
            if(--_wAdjustDutyLoop_Cnt == 0)
            {
                 GetMotor_Speed();

                 Adjust_ReferenceValue();

                 OMEGA_To_Frequency(_wSpeedBuffer.U16);

#ifdef SPEED_CLOSE_LOOP_ENABLE
                 _wError_temp.S16 = _wReference_Value - _wSpeed_FREQ.U16;

                 if((_wSpeed_FREQ.U16 > MINIMUM_SPEED_FREQ) &&
                     (_wError_temp.S16 < 10) && (_wError_temp.S16 > -10))
                     _wError_temp.S16 = 0;

                 wSpeedError_Temp.S16 = _wError_temp.S16 << 7;

                 if(_wError_temp.S16 < 0)
                 {
                     if(!(wSpeedError_Temp.U8[MSB] & 0x80))
                         wSpeedError_Temp.S16 = -wSpeedError_Temp.S16;
                 }
                 else
                 {
                     if(wSpeedError_Temp.U8[MSB] & 0x80)
                         wSpeedError_Temp.S16 = -wSpeedError_Temp.S16;
                 }

                 _dwKI_Buffer.S32 += wSpeedError_Temp.S16;

                 if(_dwKI_Buffer.S16[MSB] > 511)
                     _dwKI_Buffer.S16[MSB] = 511;
                 if(_dwKI_Buffer.S16[MSB] < -511)
                     _dwKI_Buffer.S16[MSB] = -511;


                 if(_wError_temp.S16 > 30)
                     _wError_temp.S16 -= 30;
                 else if(_wError_temp.S16 < -30)
                     _wError_temp.S16 += 30;
                 else _wError_temp.S16 = 0;

                 _wKP_Buffer.S16 = _wError_temp.S16 >> 2;

                 if(_wKP_Buffer.S16 > 100)
                     _wKP_Buffer.S16 = 100;
                 if(_wKP_Buffer.S16 < -100)
                     _wKP_Buffer.S16 = -100;

                 _dwKPKI_Buffer.S32 = _wKP_Buffer.S16 + _dwKI_Buffer.S16[MSB];

                 if(_dwKPKI_Buffer.S16[LSB] > 511)
                     _dwKPKI_Buffer.S16[LSB] = 511;
                 if(_dwKPKI_Buffer.S16[LSB] < -511)
                     _dwKPKI_Buffer.S16[LSB] = -511;

                 _dwDutyFilter.S32 = 3 * _dwDutyFilter.S32 + _dwKPKI_Buffer.S16[LSB];
                 _dwDutyFilter.S32 >>= 2;
                 _wDuty_Buffer.S16 =  _dwDutyFilter.S16[LSB];
#else
                 _wDuty_Buffer.S16 = _wReference_Value;
#endif
                 if(_wDuty_Buffer.S16 > 0x1FF)
                       _wDuty_Buffer.S16 = 0x1FF;
```

```
            if(!(_bMotor_State == MOTOR_REVERSEWINDSPEEDDOWN))
            {
                    if(_wDuty_Buffer.S16 < 25)
                    _wDuty_Buffer.S16 = 25;
            }
            else
            {
                   if(_wDuty_Buffer.S16 < 0)
                     _wDuty_Buffer.S16 = 0;
            }

              _btMotor_DutyUpdate_Enable = 1;
              if(_bMotor_State == MOTOR_REVERSEWINDSPEEDDOWN)
                  _wAdjustDutyLoop_Cnt = SPEED_DOWN_ACC_SPD;
              else
                  _wAdjustDutyLoop_Cnt = _wAdjustDutyLoopcCnt_Temp;
          }
      }
  }

  TR1 = 1;
  //User program end here.(14)
  MSFRADR = bBackupADR;
}
```

电机调试工具

为了在所有电机控制设计中获得最佳效率，MCDS IDE 软件提供电机调试工具（功能），缩短开发时间和周期。电机调试工具可用来设置并调节电机的静态和动态参数及其实时负载，从而实现对响应的监控。在设计的起始阶段，需利用电机调试过程找到最佳电机参数。

电机调试工具为速度积分库提供三个主要过程：项目设置、启动调试和 AS 调试。

项目设置

项目设置界面如图 33 所示。用户可以调节电流保护水平，并设置 PWM 频率以满足应用要求。

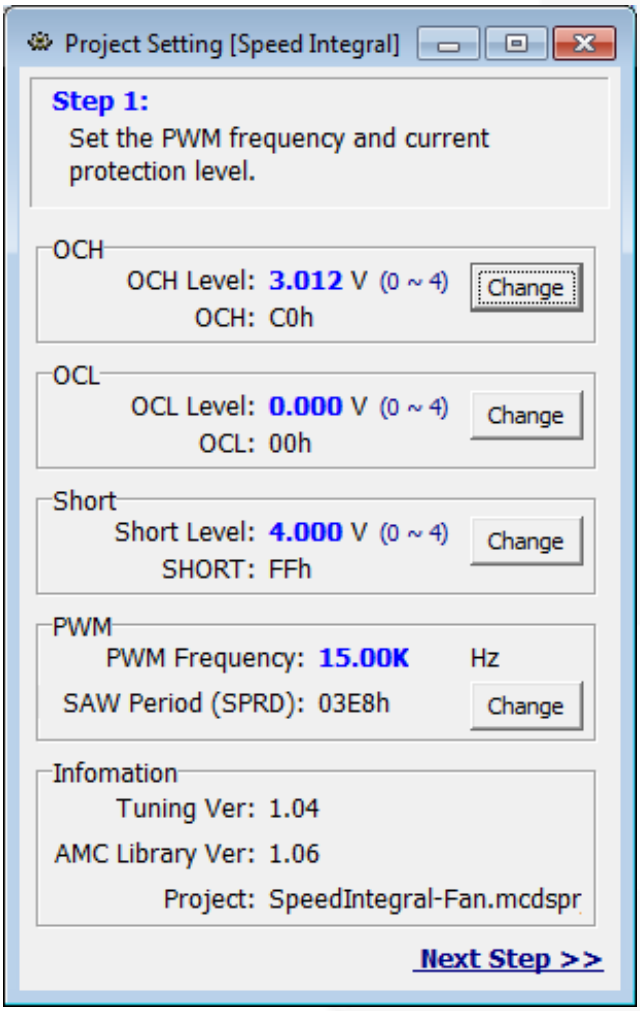


图 33. 项目设置

启动调试

启动调试界面如图 34 所示。用户应当微调相关参数，直至电机可平滑并重复启动。

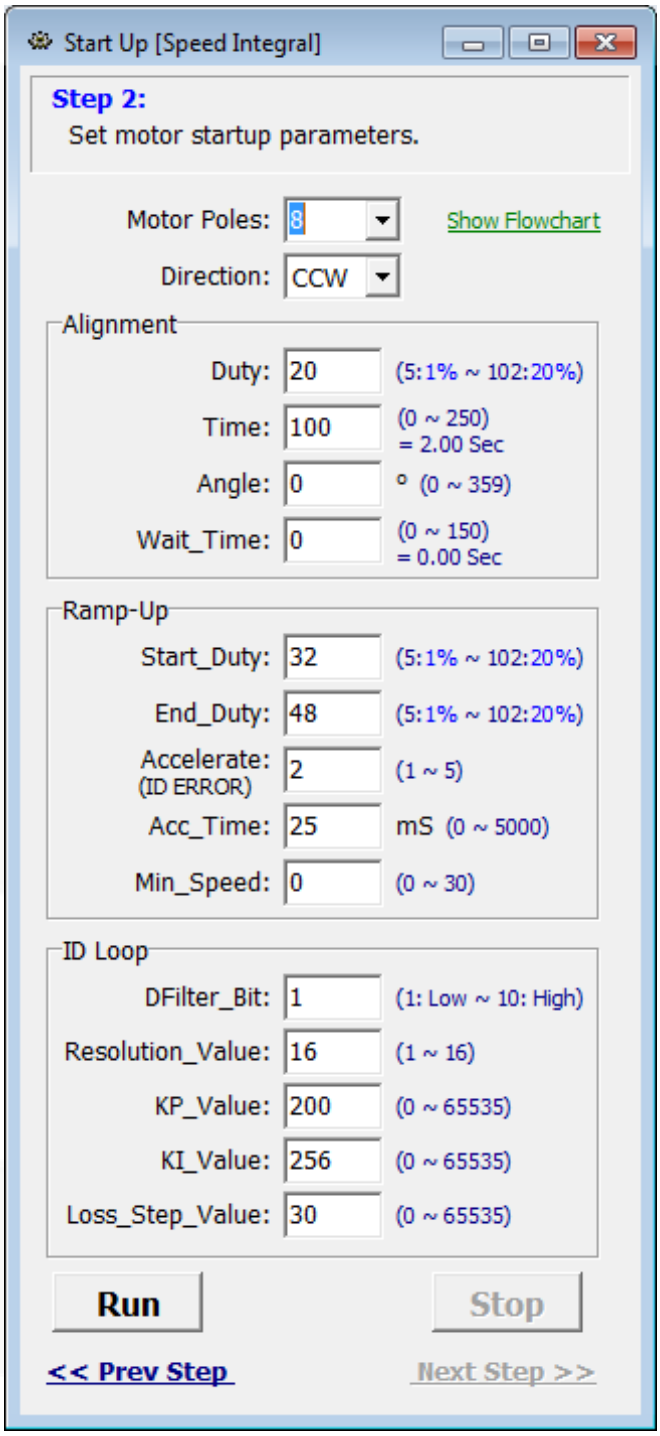


图 34. 启动调试

AS 调试

在速度闭环中执行 AS 调试，如图 35 所示。用户可设置最大速度和速度积分，以获得速度阵列。此后，AS 调试便可为每个速度段找到最佳超前角。

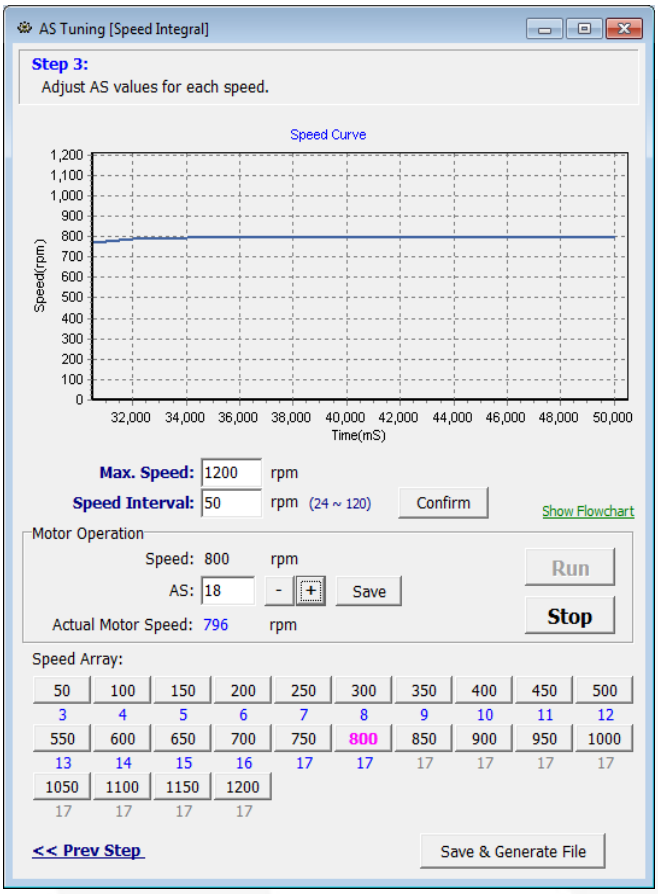


图 35. AS 调试

附录 A

短路保护 (SCP)

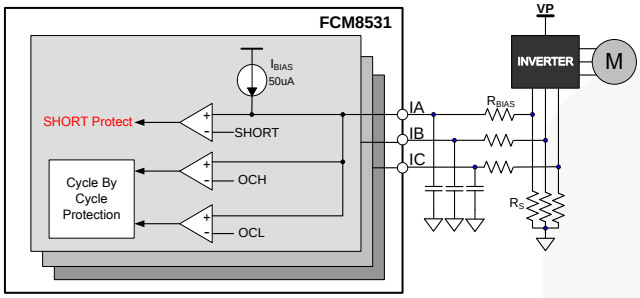


图 36. 电流保护框图

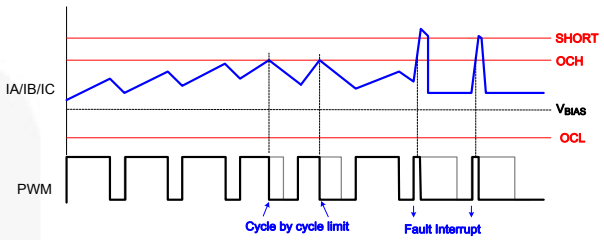


图 37. 电流保护原理示意图

表 10. 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)	Reserved		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 36 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 37），通过逐周期限流即时保护 PWM，确保过载电流不损坏三相功率组件。此外，电流与 MSFR SHORT 寄存器数值进行比较，并且可以对其编程。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)

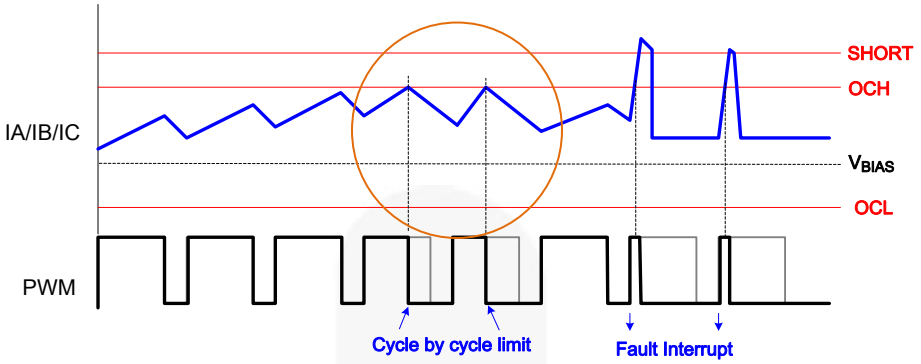


图 38. 过流保护

表 11. 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 12. 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（过流相位）保护使能
OCCL_EA：OCC 保护使能
OCBL_EA：OCB 保护使能
OCAL_EA：A OCA（过流相位）保护使能

表 13. 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 标志，读取后自动复位
OCBH：相位 B OCH 标志，读取后自动复位
OCAH：相位 A OCH 标志，读取后自动复位
OCCL：相位 C OCL 标志，读取后自动复位
OCBL：相位 B OCL 标志，读取后自动复位
OCAL：相位 A OCL 标志，读取后自动复位

表 14. DAC0 寄存器

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

表 15. DAC1 寄存器

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

因电机轴承老化或电机转子锁定所引起的未预期重载会产生巨大的瞬时电流，可能损坏电机和电源组件。为了防止损坏，FCM8531 在检测到过流之后可立即关断 PWM 信号。

请遵循下列步骤，设置过流保护参数：

1. 禁用过流保护；
2. 设置去抖时间；
3. 设置保护电平；
4. 使能过流保护；以及
5. 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)

多个 ADC 输入可用来感测外部信号，比如 IA、IB、IC、VA、VB、VC 等。本例中，ADC 输入用来感测 DC 总线电压，并且示例程序代码可用来提供过压保护。

反电动势可造成电机突然降速或制动，使直流总线上产生过多电压。过大的电压会损坏驱动电源组件和电容。需要部署良好的保护方案。

建议电路见图 39。ADC3 通过软件感测，监控直流总线电压。若软件检测到过压条件，则关断 PWM 信号输出。过压保护持续起作用，直到直流总线电压返回正常电平。

若要避免保护电平附近发生 OVP “回弹”，可在程序中定义电压滞回范围。

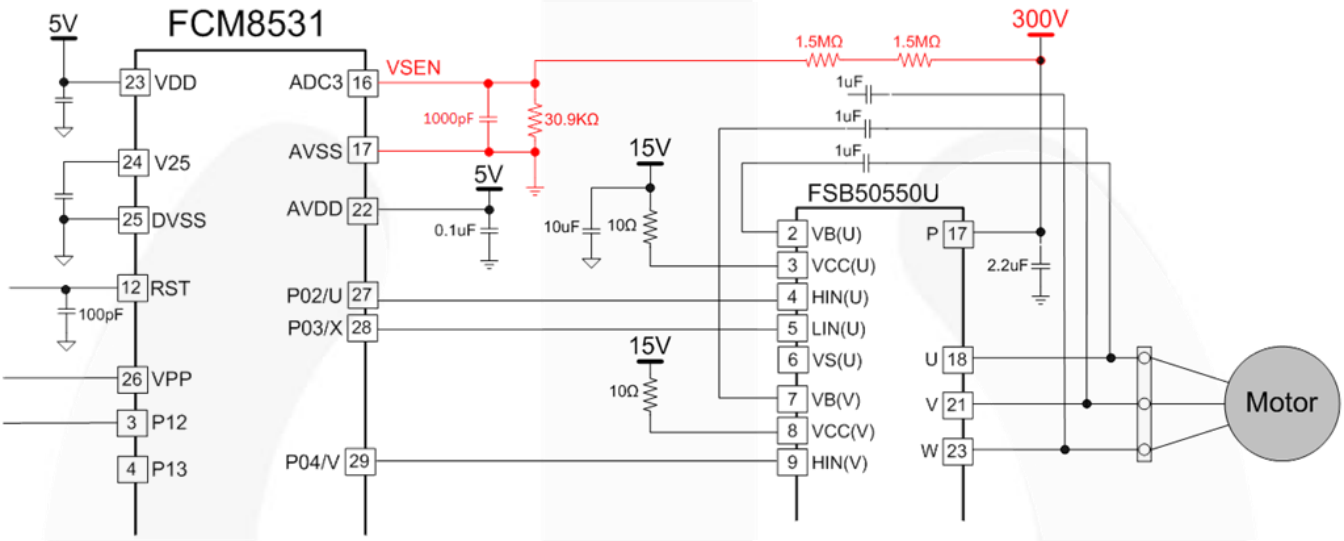


图 39. 过压保护电路

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，此处省略重复性声明。）

```
/*---------------------------------------------------------------------------
 * Copyright 2012 Fairchild Semiconductor
 * http://www.fairchildsemi.com
 *---------------------------------------------------------------------------
*/
#define HVDD_OVP_LATCH              0xED          // DC 440 V OVP Latch
#define HVDD_OVP_RELEASE            0xD2          // DC 380 V 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)

与 OVP 类似的方案同样用于 OTP，但输入源不同。将 NTC 电阻和正常电阻级联，形成分压器。感测电压通过软件读取并处理，然后与迟滞窗口比较，以便在发生 OTP 时关断 PWM 信号，在释放 OTP 时开启 PWM 信号。

图 40 中的推荐电路为评估板设计。

过温保护的示例程序代码列于下文。（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）

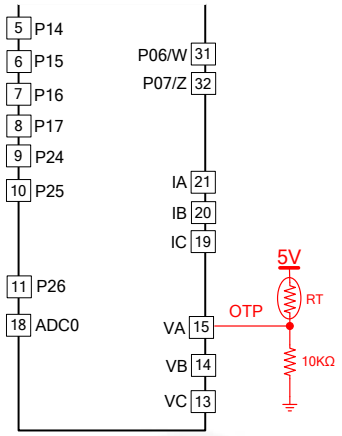


图 40. 过温保护电路

```
/*---------------------------------------------
 * 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.(1 A)
    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.(1 A)
    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;
        }
    }
}
```

如何使用 AMC 并符合 IEC 60730-1 B 类标准

若要保证安全功能正常工作，用户必须检查硬件，并修改 MCU 固件，以便符合下述要求。

硬件：

1. 短路：

   连接 PWM 引脚时，可能发生短路情况。用户必须在 PWM 引脚上（高电平、低电平引脚）执行硬件故障模式有效性分析 (FMEA)。

2. 锁定转子检测：

   FCM8531 的 PWM 输出信号由 Theta 确定（MSFR地址：3Ch).转子在三种情况下锁定：

   - Theta 估算的速度超过 500 Hz 并持续 20 ± 4 秒。
   - Theta 估算的速度低于 0.05 Hz (±20%)。
   - Theta 估算的速度保持恒定 (0.05 Hz ~ 500 Hz)。

   前两种条件可通过安全功能检测并提供保护。第三种条件无法检测。用户必须执行硬件 FMEA。

3. 电流感测电阻：

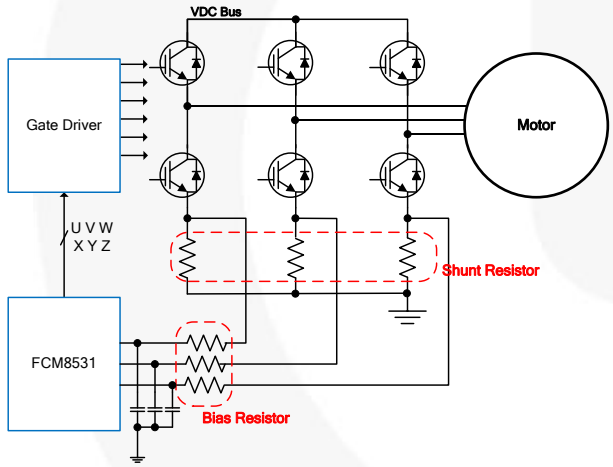


图 41. 电机驱动器界面

图 41 所示电路中，三个电流检测电阻用来感测相应的三相电流。电阻误差容限设计为低于 ± 1%。

电流检测电阻可通过下式确定：

感测电压 (V) = IA(A) * 电阻 (Ω)

相应的电机功率可通过下式计算：

电机功率 $W = 3 * Va * Ia * \cos\theta$

其中，θ 是 Va 和 Ia 的相位差：

$$Va = \frac{\frac{\sqrt{3}}{2} * 1.15}{\sqrt{2} * \sqrt{3}} VDC = \frac{1.15}{2\sqrt{2}} VDC$$

$$Ia = \frac{IA}{\sqrt{2}}$$

Va 是相位电压 (RMS)，IA 是相位电流幅度，Ia 是相位电流 (RMS)。

表 16 和表 17 供参考，使用感测电压 ±1.5 V 和 ±1.0 V。

表 16. 感测电压 ±1.5 V（偏置为 2 V）

相位电流最大值 IA (A)	电阻 (Ω)
0. 25	6. 000
0. 50	3. 000
1. 00	1. 500
2. 00	0. 750
3. 00	0. 500
4. 00	0. 375
5. 00	0. 300
6. 00	0. 250
7. 00	0. 214
8. 00	0. 188
9. 00	0. 167
10. 00	0. 150
15. 00	0. 100
20. 00	0. 075

表 17. 感测电压 ±1.0 V（偏置为 2 V）

相位电流最大值 IA (A)	电阻 (EN)
0. 25	4. 000
0. 50	2. 000
1. 00	1. 000
2. 00	0. 500
3. 00	0. 333
4. 00	0. 250
5. 00	0. 200
6. 00	0. 167
7. 00	0. 143
8. 00	0. 125
9. 00	0. 111
10. 00	0. 100
15. 00	0. 067
20. 00	0. 050

4. 偏置电阻：

   在电流感测界面中，FCM8531 为电流感测操作提供的偏置电流具有固定 2 V 偏置电压。其中，2 V 表示 0 A；此外，0 V 或 4 V 表示最大电流值。偏置电阻设计为 40 kΩ ± 1%。

固件：

1. 以适当的周期使能看门狗定时器。
2. 为过流检测时间设置合适的数值。
3. 为锁定转子重启延迟设置适当的数值。

看门狗定时器

FCM8531 提供内置看门狗功能，一旦使能并检测微电子故障后便无法禁用。看门狗定时器必须在设定的时间周期内手动清零，避免重复复位过程。清零看门狗计数器的命令行为：

```
SFR 0xA8 bit6 (WDT) =1;
SFR 0xB8 bit6 (SWDT) =1;
```

必须按照顺序设置。

根据 FCM8531 数据手册，WDTREL 确定看门狗计数器的重载值和源频率——可在 SFR 0x86 中设置。WDTREL 的功能如下：

- 提供系统频率的 16 分频选项，并馈入看门狗计数器。
- 设置看门狗重载值（0x00 至 0x7F）。

最大频率可设为 12×32×16×256×128 = 201,326,592 时钟周期，等效值为 6.7109 s。最小频率可设为 12×32=384 时钟周期，等效值为 12.672 ms。最小和最大时间均满足风险时间要求。

为了避免异常复位，看门狗时间必须设置为 1.0 s 以上。

- IC 默认设置 (WDTREL)：
- WDPS: 0（正常量程）；
- WDTH: 0x00;
- 复位时间：0.4152 ms。

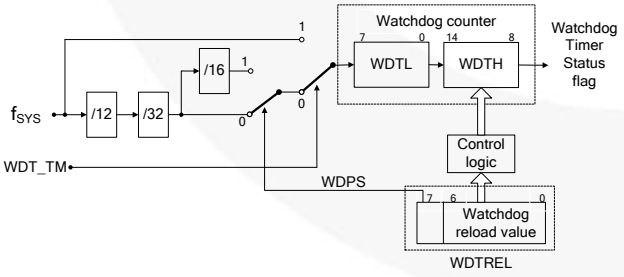


图 42. 看门狗框图

IEC 过流检测时间

通过获取三个独立的相位电流值——分别是 IA、IB 和 IC，实现 IEC 过流检测，以便确定三相电流值中的任意电流值是否超过过流阈值（IEC OCH 电平、IEC OCL 电平）。检测到 IEC 过流时，开启定时器。若定时器超过预定义阈值，则 PWM 输出关断。

命令行为 CMD_OC_MUTE_TIME；

邮箱中的地址为 036h；

邮箱发送时间值的命令为：

bWriteCmdToAMC(CMD_OC_MUTE_TIME, 0, OC_MUTE_TIME)；

默认为 3 s。范围为 0 至 53 s。，

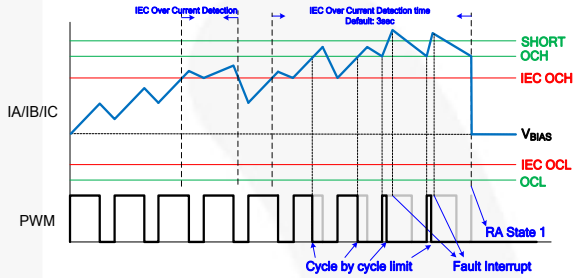


图 43. IEC 过流检测

锁定转子重启延迟

检测到锁定转子时，延迟期间不允许执行重启命令。用户可手动将延迟设为 0 至 255 s。

命令行为 CMD_LOCK_ROTOR_DELAY_TIME。

邮箱中的地址为 034h；

邮箱发送时间值的命令为：

bWriteCmdToAMC(CMD_LOCK_ROTOR_DELAY_TIME, 0, LOCK_ROT_DLY_TIME)；

默认为 10 s。范围为 0 至 255 s。，

错误消息

发生故障时，将 SFR ECL 和邮箱 MRX0 [7] 设为高电平，可输出错误消息。

根据触发的错误源，错误被归类为 RA 1 级和 RA 2 级，消息分别如下所示：

RA 1 级

保护：PWM 关断；

恢复：CMD_RUN 命令设为 FREE，或者 MCNTL[0] 设为低电平。

错误消息	情景
0xBB	检测到 ADC 引脚短路
0xCC	检测到 Over-Current
0xDD	锁定转子检测
0xEE	邮箱通信太频繁
0xFF	不适当的采样补偿值

RA 2 级

保护：PWM 关断，AMC 停止。

恢复：IC 硬件复位（复位引脚 =1）或重启电源。

错误消息	情景
0x11	SFR 检查失败
0x22	程序計數器，檢查失敗
0x33	時鐘檢查失敗
0x44	RC時鐘，檢查失敗
0x55	RAM 检查失败
0x66	ROM 检查失败
0x77	PWM 检查失败
0x88	ADC 检查失败
0x99	堆栈指针检查失败
0xAA	芯片版本检查失败

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-8205 — FCM8531 AMC 库：霍尔接口
AN-8206 — FCM8531 AMC 库：滑动模式
AN-8207 — 飞兆电机控制开发系统 (MCDS) 集成式开发环境 (IDE)