User's Guide

电机控制 SDK 通用工程和实验

TEXAS INSTRUMENTS

摘要

本文档介绍了在 MotorControlSDK 中借助通用电机控制实验工程来运行电机驱动评估套件所需的步骤，如何将实验工程迁移到定制板，以及如何将实验工程移植到新的 C2000™ 器件。

内容

插图清单

表格清单

商标

1 引言

本指南中介绍的通用电机控制实验工程不仅能让您对各种电机控制算法进行实验，还可用作您自己设计的参考。通用电机控制解决方案以及实验工程均位于 MotorControl SDK 内。

通用电机控制实验工程提供了一个使用 F28002x、F28003x 和 F280013x 系列 C2000 MCU 的示例。这是包含不同无传感器（FAST™、eSMO、InstaSPIN™-BLDC）和含传感器（增量编码器，霍尔）电机控制技术（FOC，梯形）的构建示例的单个工程，带有可用于各种三相逆变器电机评估套件的系统功能和调试接口。

在该通用电机控制实验工程中，FAST 库（用于估算电机磁通、角、速度和转矩）由 InstaSPIN-FOC™ 实现。该库允许使用已启用 FPU 和支持 C2000Ware-MotorControl-SDK 的 C2000 器件的 InstaSPIN-FOC 的 FAST 观测器。为了使用 FAST 或 InstaSPIN-FOC，用户不再需要使用具有特殊 ROM 内容的 C2000 器件。

在本用户指南中，您将了解如何修改 *user_mtr1.h* 文件，该文件是存储所有用户参数的头文件。其中一些参数可以在运行时通过 CCS 进行操作，但必须更新 *user_mtr1.h* 文件中的参数，才能永久保存在工程中。您将学习如何将该实验迁移到您自己的硬件板，以及通过修改 *hal.h* 和 *hal.c* 文件将该实验工程移植到其他 C2000 MCU 控制器。

实验工程提供了多个接口功能来启动/停止电机以及使用按钮、电位器或 CAN 接口设置基准速度。

电机控制通用实验工程在 MotorControl SDK 文件夹中构建，并使用 C2000Ware 中的文件。MotorControl SDK 软件包括在 C2000 电机控制评估模块 (EVM) 和 TI Designs (TID) 上运行的固件。MotorControl SDK 包含一个 C2000Ware 副本，范围涵盖器件特定的驱动程序和支持软件到完整示例系统应用。

通用 MotorControl 实验需要：

- Code Composer Studio™ v12.0.0 或更高版本
- C2000 编译器 v22.6.0 LTS 或更高版本
- C2000Ware MotorControl SDK V4.01.00 或更高版本

2 电机控制理论

永磁同步电机 (PMSM) 具有一个绕线定子、一个永磁转子组件和用于检测转子位置的内部或外部器件。感测器件提供位置反馈以适当地调整定子基准电压的频率和振幅，从而使磁体组件保持旋转。一个内部永磁转子和外部绕组的组合提供低转子惯性、有效散热和电机尺寸减少等优势。

- 同步电机构造：永磁体被牢牢固定在旋转轴上，生成了一个恒定的转子磁通。这个转子磁通通常具有一个恒定的磁通量。定子绕组通电后可产生旋转电磁场。为了控制旋转的磁场，有必要控制定子电流。

- 根据机器的功率范围和额定速度，转子的实际结构会有所不同。永磁体适合于范围高达几千瓦的同步机器。为了获得更高的额定功率，转子通常由接通直流电的绕组组成。转子的机械结构是针对所需磁极的数量和所需的磁通梯度进行设计的。
- 定子和转子磁通的交感产生了一个转矩。由于定子被牢固地安装在电机架上，而转子可自由旋转，因此转子的旋转将产生一个有用的机械输出，如图 1-1 所示。
- 必须仔细控制转子磁场和定子磁场间的角度，以产生最大扭矩和实现较高的机电转换效率。为了实现这一目的，在同一转速和扭矩条件下，为了尽可能少地消耗电流，在关闭速度环路后需要使用无传感器算法进行微调。
- 旋转中的定子磁场的频率必须与转子永磁磁场的频率相同，否则转子就会经历快速的正负扭矩交替。这会减少最优扭矩产出量，并且在机器部件上产生过多的机械抖动、噪声和机械应力。此外，如果转子因惯性而不能对这些摆动做出响应，那么转子的转动会偏离同步频率，并且对静止转子的平均扭矩（零扭矩）做出响应。这意味着机器会出现一种称为*牵出*的现象。这也是为什么同步机器不能自启动的原因。
- 转子磁场与定子磁场间的角度必须等于 90º 以获得最高的互扭矩产出量。为了产生正确的定子磁场，该同步需要知道转子位置。
- 通过将不同转子相位的输出组合在一起，可将定子磁场设定为任一方向和强度以产生相应的定子磁通。

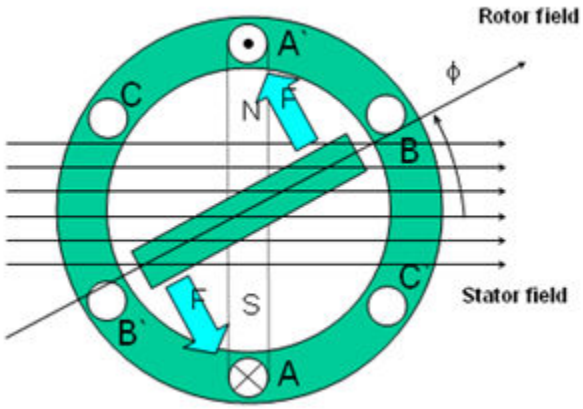


图 2-1. 旋转的定子磁通和转子磁通之间的相互作用产生扭矩

2.1 PMSM 的数学模型和 FOC 结构

PMSM 的无传感器 FOC 结构如图 1-1 所示。在该系统中，eSMO 用于实现 IPMSM 系统的无传感器控制，eSMO 模型是利用反电动势模型和 PLL 模型设计的，用于估算转子位置和转速。

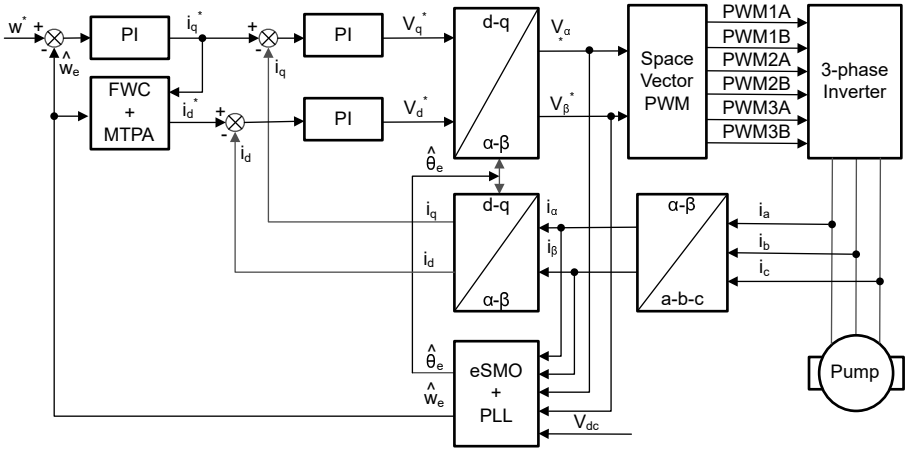


图 2-2. PMSM 系统的无传感器 FOC 结构

IPMSM 由一个三相定子绕组（a、b、c 轴）和用于励磁的永磁体 (PM) 转子组成。电机由标准的三相逆变器进行控制。可以使用相位 a-b-c 量对 IPMSM 进行建模。通过适当的坐标变换，可以得到 d-q 转子坐标系和 α - β 静止坐标系中的动态 PMSM 模型。这些坐标系之间的关系如方程式 1 所示。通用 PMSM 的动态模型可以在 d-q 转子坐标系中写为：

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} R_s + pL_d & -\omega_e L_q \\ \omega_e L_d & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_e \lambda_{pm} \end{bmatrix} \quad (1)$$

其中 v_d 和 v_q 分别是 q 轴和 d 轴定子端电压；i_d 和 i_q 分别是 d 轴和 q 轴定子电流；L_d 和 L_q 分别是 q 轴和 d 轴电感，p 是导数算子，用于简写 $\frac{d}{dt}$ ；λ_{pm} 是永磁体产生的磁链，R_s 是定子绕组的电阻；ω_e 是转子的电角速度。

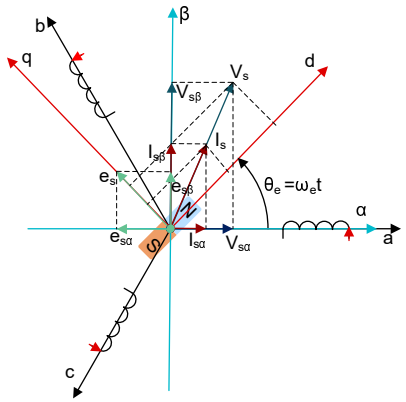


图 2-3. PMSM 建模坐标系的定义

通过使用如图 1-1 所示的 Park 逆变换，PMSM 的动力学可以在 α - β 静止坐标系中建模为：

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} R_s + pL_d & \omega_e\left(L_d - L_q\right) \\ -\omega_e\left(L_d - L_q\right) & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} \quad (2)$$

其中，e_a 和 e_β 是 α - β 轴上扩展电动势 (EEMF) 的分量，可以定义为：

$$\begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} = \left(\lambda_{pm} + \left(L_d - L_q\right) i_d\right) \omega_e \begin{bmatrix} -\sin(\theta_e) \\ \cos(\theta_e) \end{bmatrix} \quad (3)$$

根据方程式 2 和方程式 3，通过等效变换和引入 EEMF 概念，可以将转子位置信息从电感矩阵中解耦出来，从而使 EEMF 成为唯一包含转子磁极位置信息的项。然后可以直接利用 EEMF 相位信息实现转子位置观测。使用定子电流作为状态变量，将 IPMSM 电压公式方程式 4 改写为状态公式：

$$\begin{bmatrix} \dot{i}_\alpha \\ \dot{i}_\beta \end{bmatrix} = \frac{1}{L_d} \begin{bmatrix} -R_s & -\omega_e\left(L_d - L_q\right) \\ \omega_e\left(L_d - L_q\right) & -R_s \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \frac{1}{L_d} \begin{bmatrix} V_\alpha - e_\alpha \\ V_\beta - e_\beta \end{bmatrix} \quad (4)$$

由于定子电流是唯一可以直接测量的物理量，因此在定子电流路径上选择滑动面：

$$S(x) = \begin{bmatrix} \hat{i}_\alpha - i_\alpha \\ \hat{i}_\beta - i_\beta \end{bmatrix} = \begin{bmatrix} \tilde{i}_\alpha \\ \tilde{i}_\beta \end{bmatrix} \quad (5)$$

其中 $\hat{i}_\alpha$ 和 $\hat{i}_\beta$ 是估算的电流，上标 **^** 表示变量为估算值，上标 "~" 表示变量为变量误差，即观测值与实际测量值之间的差异。

2.2 PM 同步电机的磁场定向控制

为了实现更好的动态性能，需要采用更加复杂的控制方案来控制 PM 电机。借助微控制器提供的数学处理能力，我们可以实施先进的控制策略，这些策略使用数学变换将永磁电机中的扭矩生成和磁化功能解耦。这种解耦的扭矩和磁化控制通常称为转子磁通定向控制，或简称为磁场定向控制 (FOC)。

在直流电机中，定子和转子的励磁是独立控制的，产生的扭矩和磁通可以独立调整，如图 1-1 所示。磁场激励强度（例如，磁场激励电流的振幅）决定了磁通的大小。通过转子绕组的电流确定了扭矩是如何生成。转子上的换向器在扭矩产生过程中发挥着有趣的作用。换向器与电刷接触，这个机械构造旨在将电路切换至机械对齐的绕组以产生最大的扭矩。这样的安排意味着，电机的扭矩产生在任何时候都非常接近于最佳情况。这里的关键点是，通过管理绕组以保持转子绕组产生的磁通与定子磁场垂直。

为了实现更好的动态性能，需要采用更加复杂的控制方案来控制 PM 电机。借助微控制器提供的数学处理能力，我们可以实施先进的控制策略，这些策略使用数学变换将永磁电机中的扭矩生成和磁化功能解耦。这种解耦的扭矩和磁化控制通常称为转子磁通定向控制，或简称为磁场定向控制 (FOC)。

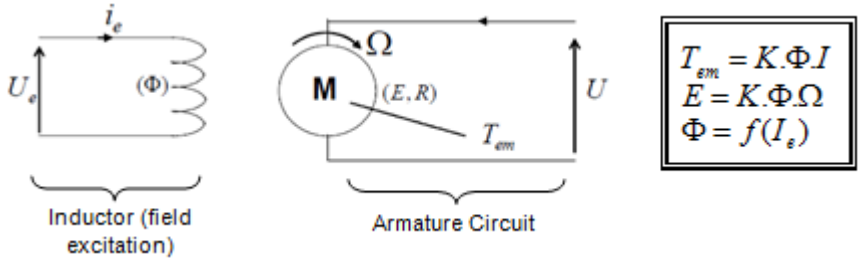


图 2-4. 在直流电机模型中磁通和扭矩是独立控制的

同步和异步电机上的 FOC（也称为矢量控制）旨在分别控制扭矩产生分量和磁化通量分量。利用 FOC 控制，我们能够解耦定子电流的扭矩分量和磁化通量分量。借助于磁化的去耦合控制，定子磁通的扭矩生成分量现在可以

被看成是独立扭矩控制。为了去耦合扭矩和磁通，有必要采用几个数学变换，而这是最能体现微控制器价值的地方。微控制器提供的处理能力可非常快速地执行使这些数学变换。反过来，这意味着控制电机的整个算法可以高速率执行，从而实现了更高的动态性能。除了去耦合，现在一个电机的动态模型被用于很多数量的计算，例如转子磁通角和转子速度。这意味着，它们的影响被计算在内，并且总体控制质量更佳。

根据电磁定律，同步电机中产生的扭矩等于两个现有磁场的矢量叉积，如方程式 6 所示。

$$\tau_{em} = \vec{B}_{stator} \times \vec{B}_{rotor} \tag{6}$$

该表达式表明，如果定子和转子磁场正交，则扭矩最大，这意味着我们需要将负载保持在 90°。如果我们能够始终确保满足这一条件，并且能够正确地对磁通进行定向，将减少扭矩纹波并确保实现更好的动态响应。然而，您需要了解转子的位置：这可以通过位置传感器（诸如递增编码器）实现。对于无法接近转子的低成本应用，采用不同的转子位置观察器策略可无需使用位置传感器。

简而言之，目标是使转子和定子磁通保持正交：例如，目标是将定子磁通与转子磁通的 q 轴对齐，从而与转子磁通正交。为了实现这个目的，控制与转子磁通正交的定子电流分量以产生命令规定的扭矩，并且直接分量被设定为零。定子电流的直接分量可用在某些磁场减弱的情况下，这有抗拒转子磁通的作用，并且减少反电动势，从而实现更高速的运行。

磁场定向控制包括控制由矢量表示的定子电流。该控制基于将三相时间和速度相关系统变换为两坐标（d 和 q 坐标）时不变系统的投影。这些设计导致一个与 DC 机器控制结构相似的结构。磁场定向控制 (FOC) 电机需要两个常数作为输入基准：扭矩分量（与 q 坐标对齐）和磁通分量（与 d 坐标对齐）。由于磁场定向控制只是基于这些投影，因此控制结构将处理瞬时电量。这使得在每次的工作运转过程中（稳定状态和瞬态）均可实现准确控制，并且与受限带宽数学模型无关。因此，FOC 通过以下方式解决了传统方案存在的问题：

- 轻松达到恒定基准（定子电流的扭矩分量和磁通分量）
- 轻松应用直接扭矩控制，这是因为在 (d, q) 坐标系中，扭矩的表达式定义如方程式 7 所示。

$$\tau_{em} \propto \psi_R \times i_{sq} \tag{7}$$

通过将转子磁通 (ψ_R) 的振幅保持在一个固定值，扭矩和扭矩分量 (i_{Sq}) 之间存在线性关系。然后我们可以通过控制定子电流矢量的扭矩分量来控制扭矩。

空间矢量定义和投影

交流电机的三相电压、电流和磁通可根据复数空间矢量进行分析。对于电流，空间矢量可定义如下。假设 i_a、i_b、i_c 是定子的三相即时电流，则复数定子电流矢量的定义如方程式 8 所示。

$$\bar{i}_s = i_a + \alpha i_b + \alpha^2 i_c \tag{8}$$

其中 $\alpha = e^{j\frac{2}{3}\pi}$ 和 $\alpha^2 = e^{j\frac{4}{3}\pi}$ 表示空间运算元。

图 1-1 所示为定子电流的复数空间矢量。

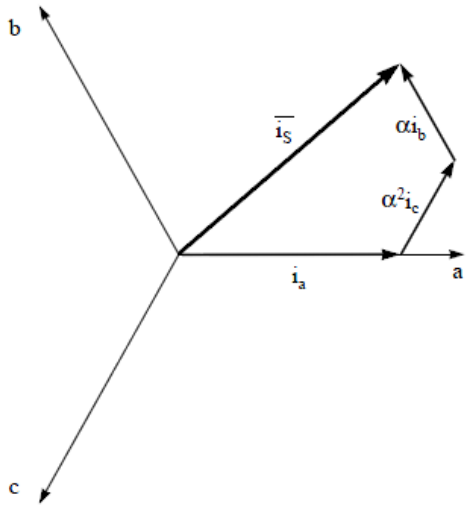


图 2-5. 定子电流空间矢量及其以 (a,b,c) 坐标系表示的分量

其中 (a,b,c) 是三相系统轴。这个电流空间矢量对三相正弦系统进行了描述，但仍需变换为一个两坐标非时变系统。这个变换可拆分为两个步骤：

- $(a, b) \Rightarrow (\alpha, \beta)$（Clarke 变换），输出一个两坐标时变系统
- $(\alpha, \beta) \Rightarrow (d, q)$（Park 变换），输出一个两坐标时不变系统

当 $(a, b) \Rightarrow (\alpha, \beta)$ Clarke 变换

可以使用另外一个仅包含两相 (α , β) 正交轴的坐标系来表示该空间矢量。假设 a 轴和 α 轴方向相同，我们可以得到下面图 1-1 所示的矢量图。

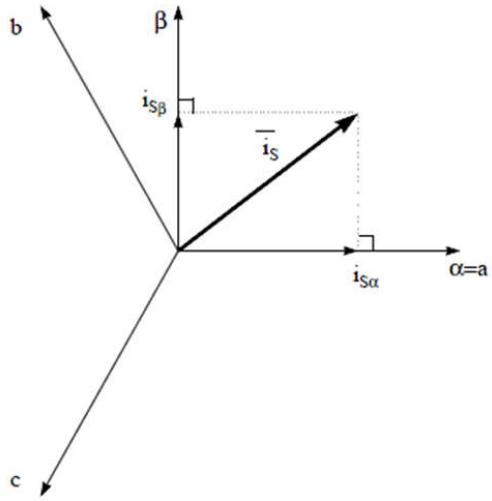


图 2-6. 静止坐标系中的定子电流空间矢量

将三相系统修改为 (α , β) 二维正交系统的投影如方程式 9 所示。

$$i_{s\alpha} = i_a$$
$$i_{s\beta} = \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b \quad (9)$$

两相 (α , β) 电流仍取决于时间和速度。

当 $(\alpha, \beta) \Rightarrow (d, q)$ Park 变换

这是 FOC 内最重要的变换。事实上，该投影在 (d, q) 旋转坐标系中修改了一个两相正交系统 (α , β)。如果我们考虑 d 轴与转子磁通对齐，那么图 1-1 显示了来自该二维坐标系的电流矢量的关系。

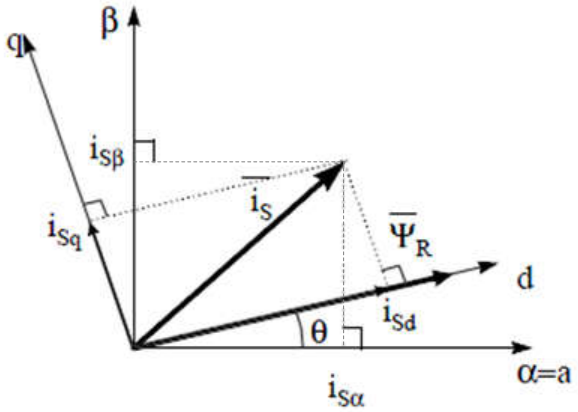


图 2-7. d,q 旋转坐标系中的定子电流空间矢量

电流矢量的磁通和扭矩分量由方程式 10 决定。

$$i_{sd} = i_{s\alpha}\cos(\theta) + i_{s\beta}\sin(\theta)$$
$$i_{sq} = -i_{s\alpha}\sin(\theta) + i_{s\beta}\cos(\theta) \quad (10)$$

其中 θ 是转子磁通位置

这些分量取决于电流矢量（α，β）分量和转子磁通位置；如果我们知道正确的转子磁通位置，那么，通过该投影，d,q 分量就变成一个常量。现在，两个相位电流变换为直流数量（非时变）。此时扭矩控制变得更容易，其中恒定的 i_{sd}（磁通分量）和 i_{sq}（扭矩分量）电流分量单独受到控制。

交流电机 FOC 基本配置方案

图 1-1 总结了使用 FOC 进行扭矩控制的基本配置方案：

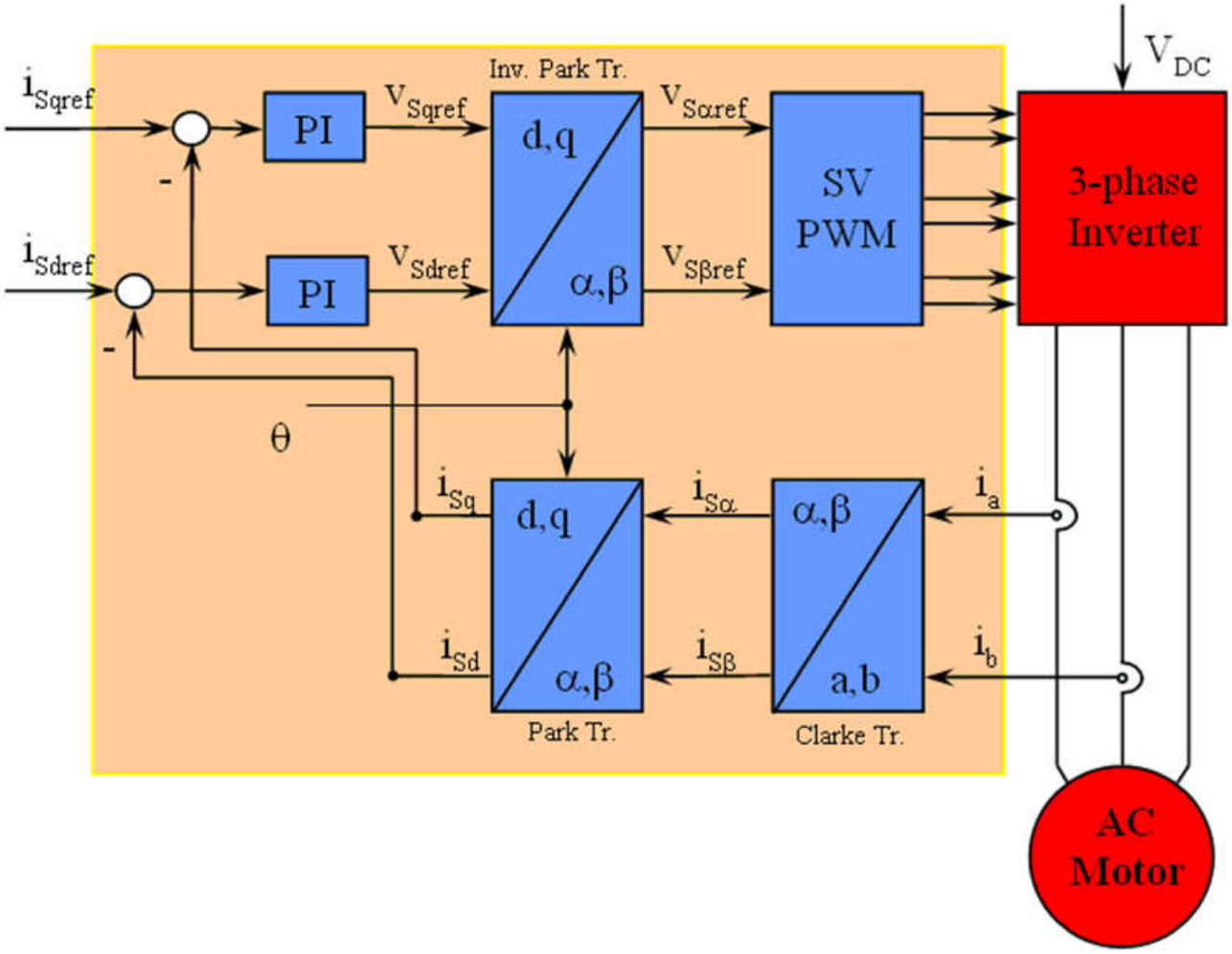


图 2-8. 交流电机 FOC 基本配置方案

测量了两个电机相电流。这些测量值馈入 Clarke 变换模块。这个模块的输出为 $i_{s\alpha}$ 和 $i_{s\beta}$。电流的这两个分量是 Park 变换的输入，该变换给出了 d,q 旋转坐标系中的电流。i_{sd} 和 i_{sq} 分量与基准 i_{sdref}（磁通基准分量）和 i_{sqref}（扭矩基准分量）进行比较。在这一点上，这个控制结构显示了一个有意思的优势：它可被用来控制同步或感应机器，采用的方法就是简单地改变磁通基准并获得转子磁通位置。与在同步永磁电机中一样，转子磁通是固定的，并由磁体确定；所以无需产生转子磁通。因此，当控制一个 PMSM 时，i_{sdref} 应被设定为 0。由于交流感应电机需要生成转子磁通才能运行，因此磁通基准一定不能为零。这很方便地解决了*经典*控制结构的一个主要缺陷：异步驱动至同步驱动的可移植性。当我们使用转速 FOC 时，扭矩命令 i_{sqref} 可以是转速调节器的输出。电流调节器的输出是 Vsdref 和 V_{sqref}；它们进行 Park 逆变换。

这个模块的输出是 $V_{s\alpha ref}$ 和 $V_{s\beta ref}$，它们是 (α，β) 静止正交坐标系中定子矢量电压的分量。这些是空间矢量脉宽调制 (PWM) 的输入。这个块的输出是驱动此反相器的信号。请注意，Park 和 Park 逆变换均需要转子磁通位置。这个转子磁通位置的获得由交流机器的类型 (同步或异步机器) 而定。

转子磁通位置

转子磁通位置的相关知识是 FOC 的核心。事实上，如果该变量存在误差，则转子磁通与 d 轴不对齐，并且定子电流的磁通和扭矩分量 i_{sd} 和 i_{sq} 不正确。图 1-1 显示了 (a, b, c)、(α，β) 和 (*d, q*) 坐标系，以及转子磁通的正确位置和以同步速度随 d,q 坐标旋转的定子电流和定子电压空间矢量。

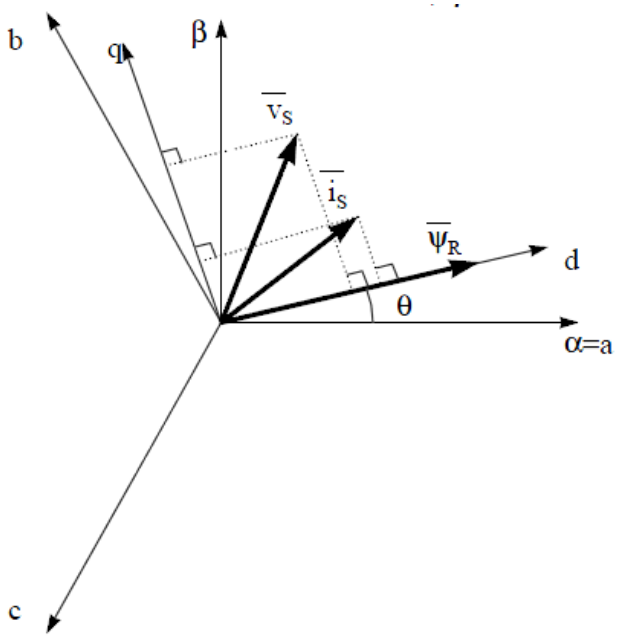


图 2-9. (d, q) 旋转坐标系中的电流、电压和转子磁通空间矢量

如果我们考虑同步或异步电机，转子磁通位置的测量是不同的：

- 在同步电机中，转子转速等于转子磁通转速。然后 θ (转子磁通位置) 由位置传感器或转子速度的积分直接计算。
- 在异步电机中，转子转速不等于转子磁通转速 (存在转差速度)，因此需要使用特定的方法来计算 θ。基本方法是使用一个电流模型，该模型需要 *d, q* 坐标系中的电机模型的两个公式。

理论上，利用适用于 PMSM 驱动的磁场定向控制，可以使用磁通实现对电机扭矩的单独控制，这与直流电机的运行类似。换句话说，转矩和磁通互相之间去耦合。从静止坐标系到同步旋转坐标系间的变量变换需要知道转子位置信息。由于这种变换 (所谓的 Park 变换)，q 轴电流将控制扭矩，而 d 轴电流被强制设置为零。因此，该系统的关键模块是使用增强型滑模观测器 (eSMO) 或 FAST 估算器来估算转子位置。

图 1-1 展示了本文档中 PMSM 电机的无传感器 FOC（使用 eSMO 并具有快速启动功能）的整体方框图。

图 1-1 展示了本文档中 PMSM 电机的无传感器 FOC（使用 eSMO 并具有弱磁控制 (FWC) 和每安培最大扭矩 (MTPA) 功能）的整体方框图。

图 1-1 展示了本文档中 PMSM 电机的无传感器 FOC（使用 FAST 并具有快速启动功能）的整体方框图。

图 1-1 展示了本文档中 PMSM 电机的无传感器 FOC（使用 FAST 并具有弱磁控制 (FWC) 和每安培最大扭矩 (MTPA) 功能）的整体方框图。

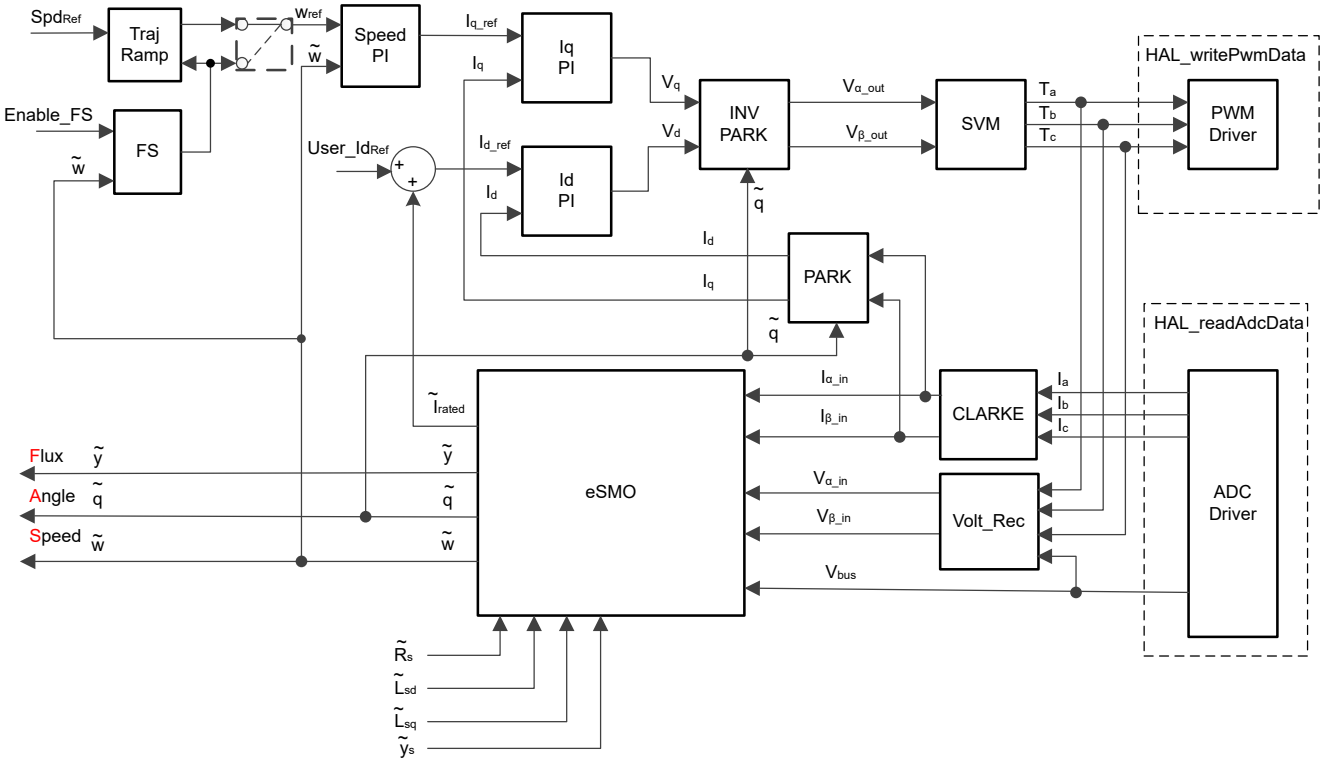


图 2-10. 使用 eSMO 并具有快速启动 (FS) 功能的 PMSM 的无传感器 FOC

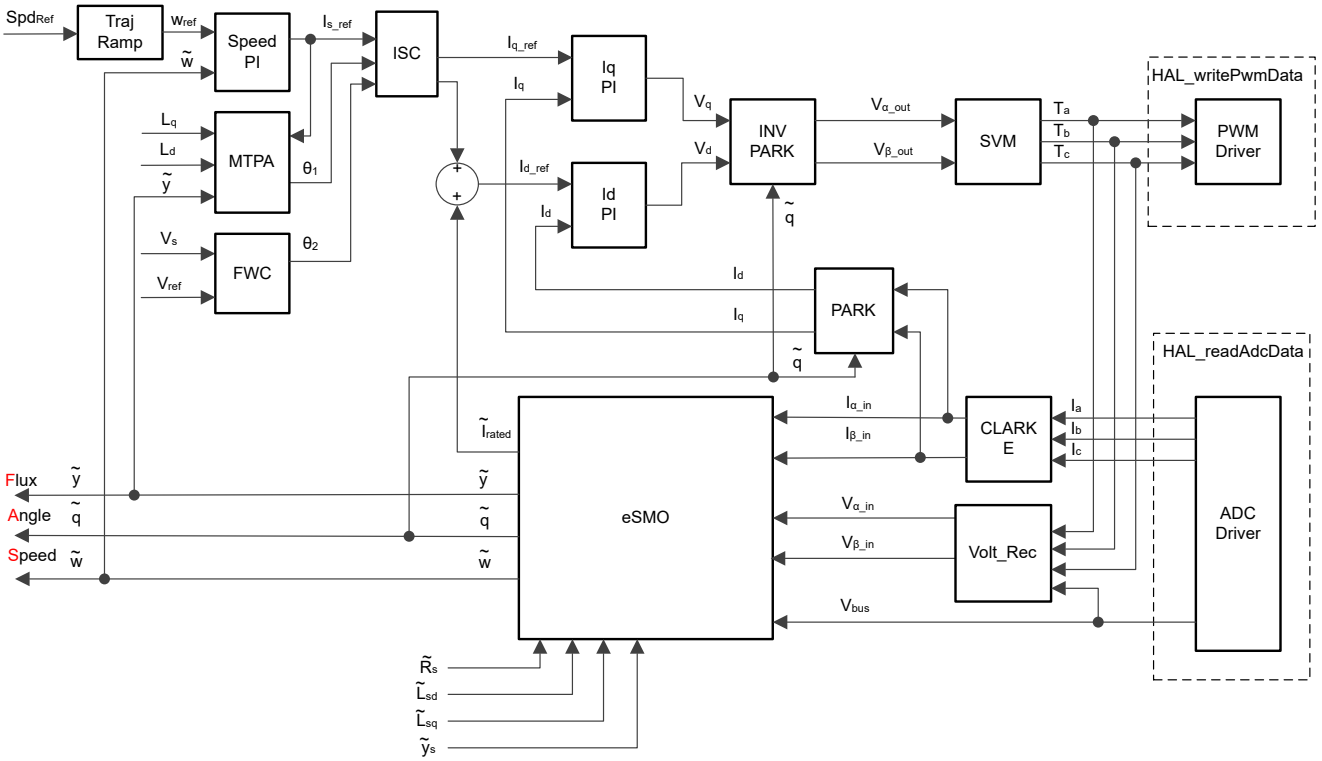


图 2-11. 使用 eSMO 并具有 FWC 和 MTPA 功能的 PMSM 的无传感器 FOC

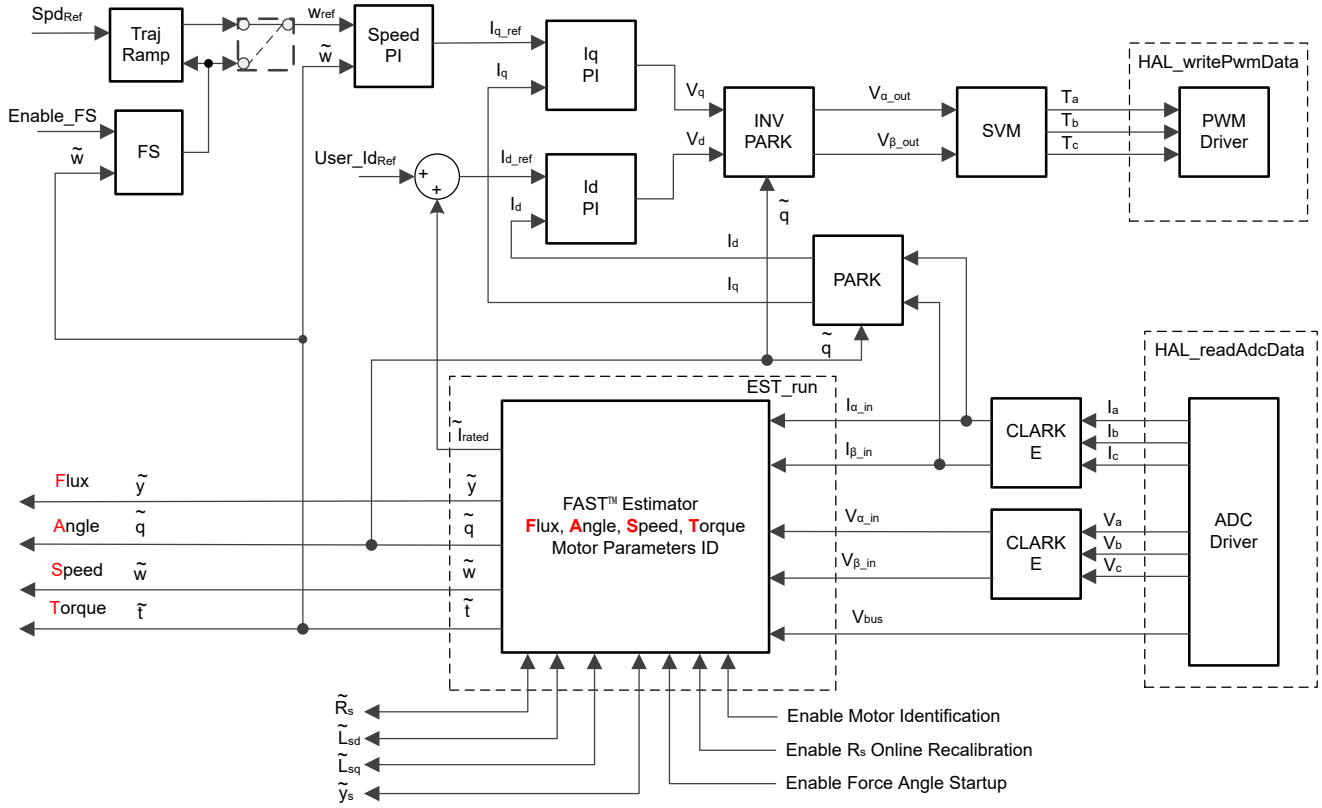


图 2-12. 使用 FAST 并具有快速启动 (FS) 功能的 PMSM 的无传感器 FOC

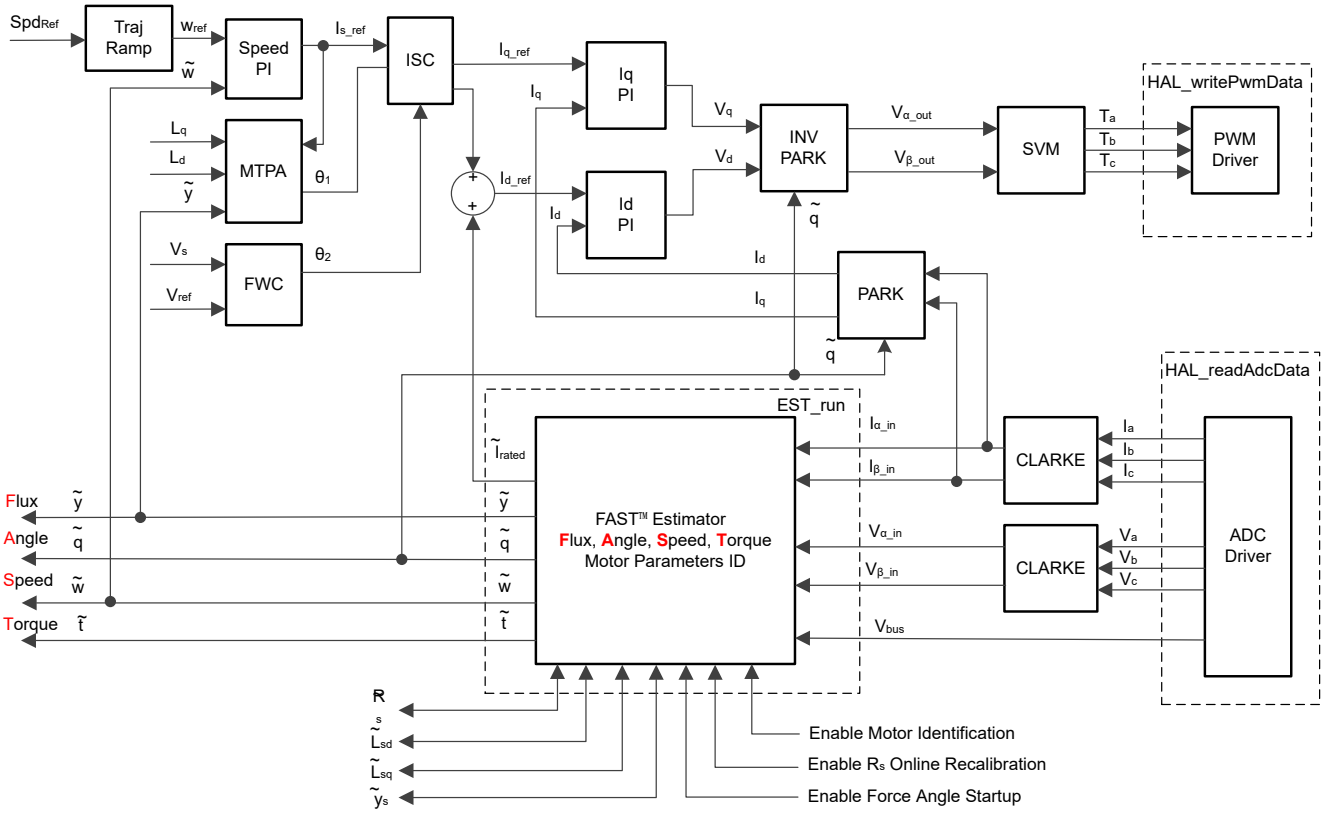


图 2-13. 使用 FAST 并具有 FWC 和 MTPA 功能的 PMSM 的无传感器 FOC

2.3 PM 同步电机的无传感器控制

在家用电器应用中，如果使用机械传感器，将会导致成本、尺寸和可靠性问题增加。为了克服这些问题，无传感器控制方法应运而生，它可以通过多种估算方法在没有机械位置传感器的情况下获得转子转速和位置信息。滑模观测器 (SMO) 因其各种吸引人的特性（包括可靠性、所需的性能和针对系统参数变化的稳健性）而被广泛使用。

2.3.1 具有锁相环的增强型滑模观测器

基于模型的方法用于实现 IPMSM 驱动系统在电机以中高速运行时的无位置传感器控制。模型法通过反电动势或磁链模型估算转子位置。滑动模式观测器是基于滑模控制的观测器设计方法。系统的结构不是固定的，而是根据系统的当前状态有目的地改变，迫使系统按照预定的滑模轨迹运动。其优点包括响应速度快、稳健性高以及对参数变化和干扰不敏感。

2.3.1.1 PMSM 的 ESMO 设计

图 1-1 显示了集成在 SMO 中的传统 PLL。

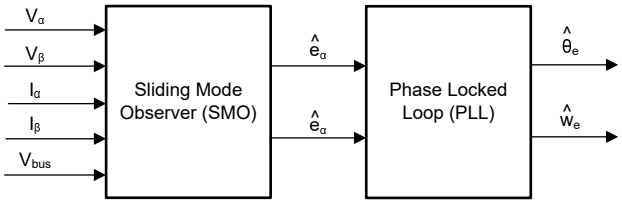


图 2-14. 包含用于 PMSM 的 PLL 的 eSMO 方框图

构建了传统的降阶滑模观测器，其数学模型如图 1-1 所示，方框图如图 1-1 所示。

$$\begin{bmatrix} \hat{i}_\alpha \\ \hat{i}_\beta \end{bmatrix} = \frac{1}{L_d}\begin{bmatrix} -R_s & -\hat{\omega}_e\left(L_d - L_q\right) \\ \hat{\omega}_e\left(L_d - L_q\right) & -R_s \end{bmatrix}\begin{bmatrix} \hat{i}_\alpha \\ \hat{i}_\beta \end{bmatrix} + \frac{1}{L_d}\begin{bmatrix} V_\alpha - \hat{e}_\alpha + z_\alpha \\ V_\beta - \hat{e}_\beta + z_\beta \end{bmatrix} \tag{11}$$

其中 z_α 和 z_β 是滑模反馈分量，其定义为：

$$\begin{bmatrix} z_\alpha \\ z_\beta \end{bmatrix} = \begin{bmatrix} k_\alpha sign\left(\hat{i}_\alpha - i_\alpha\right) \\ k_\beta sign\left(\hat{i}_\beta - i_\beta\right) \end{bmatrix} \tag{12}$$

其中 k_α 和 k_β 是通过李雅普诺夫稳定性分析设计的恒定滑模增益。如果 k_α 和 k_β 是足够大的正值，以保证 SMO 的稳定运行， k_α 和 k_β 应足够大，以保持 $k_\alpha > \max(|e_\alpha|)$ 和 $k_\beta > \max\left(\left|e_\beta\right|\right)$ 。

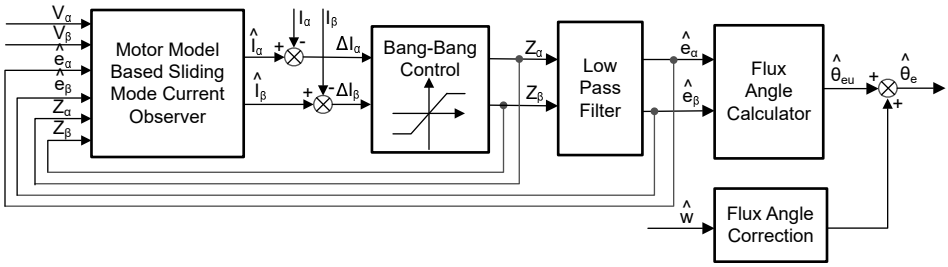


图 2-15. 传统滑模观测器的方框图

α - β 轴上的 EEMF 估算值 ($\hat{e}_\alpha$, $\hat{e}_\beta$) 可通过低通滤波器从不连续开关信号中获得，这些信号为 z_α 和 z_α ：

$$\begin{bmatrix} \hat{e}_\alpha \\ \hat{e}_\beta \end{bmatrix} = \frac{\omega_c}{s + \omega_c}\begin{bmatrix} z_\alpha \\ z_\beta \end{bmatrix} \tag{13}$$

其中 $\omega_c = 2\pi f_c$ 是 LPF 的截止角频率，通常根据定子电流的基频来选择该截止角频率。

因此，转子位置可以直接通过反电动势的反正切计算得出，其定义如下：

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) \tag{14}$$

低通滤波器消除了滑模函数的高频项，从而导致出现相位延迟。可以通过截止频率 ω_c 和反电动势频率 ω_e 之间的关系对其进行补偿，定义为：

$$\Delta\theta_e = -\tan^{-1}\left(\frac{\omega_e}{\omega_c}\right) \tag{15}$$

这样使用 SMO 方法估算的转子位置就为：

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) + \Delta\theta_e \tag{16}$$

在数字控制应用中，需要使用 SMO 的时间离散方程。欧拉法是变换为时间离散观测器的合适方法。在 α - β 坐标中，方程式 17 的时间离散系统矩阵由方程式 17 给出：

$$\begin{bmatrix}\hat{i}_\alpha(n+1)\\ \hat{i}_\beta(n+1)\end{bmatrix}=\begin{bmatrix}F_\alpha\\ F_\beta\end{bmatrix}\begin{bmatrix}\hat{i}_\alpha(n)\\ \hat{i}_\beta(n)\end{bmatrix}+\begin{bmatrix}G_\alpha\\ G_\beta\end{bmatrix}\begin{bmatrix}V_\alpha^*(n)-\hat{e}_\alpha(n)+z_\alpha(n)\\ V_\beta^*(n)-\hat{e}_\beta(n)+z_\beta(n)\end{bmatrix} \tag{17}$$

其中矩阵 $[F]$ 和 $[G]$ 由方程式 18 和方程式 19 给出：

$$\begin{bmatrix}F_\alpha\\ F_\beta\end{bmatrix}=\begin{bmatrix}e^{-\frac{R_S}{L_d}}\\ e^{-\frac{R_S}{L_q}}\end{bmatrix} \tag{18}$$

$$\begin{bmatrix}G_\alpha\\ G_\beta\end{bmatrix}=\frac{1}{R_S}\begin{bmatrix}1-e^{-\frac{R_S}{L_d}}\\ 1-e^{-\frac{R_S}{L_q}}\end{bmatrix} \tag{19}$$

方程式 13 的时间离散形式由方程式 20 给出：

$$\begin{bmatrix}\hat{e}_\alpha(n+1)\\ \hat{e}_\beta(n+1)\end{bmatrix}=\begin{bmatrix}\hat{e}_\alpha(n)\\ \hat{e}_\beta(n)\end{bmatrix}+2\pi f_c\begin{bmatrix}z_\alpha(n)-\hat{e}_\alpha(n)\\ z_\beta(n)-\hat{e}_\beta(n)\end{bmatrix} \tag{20}$$

2.3.1.2 使用 PLL 的转子位置和转速估算

在反正切法中，由于噪声和谐波分量的存在，位置和转速估算的精度会受到影响。为了消除该问题，可使用 PLL 模型对 IPMSM 的无传感器控制结构中的转速和位置进行估算。节 2.3.1.1 中说明了与 SMO 配合使用的 PLL 结构。反电动势估算 $\hat{e}_\alpha$ 和 $\hat{e}_\beta$ 可与 PLL 模型配合使用来估算电机角速度和位置，如图 1-1 所示。

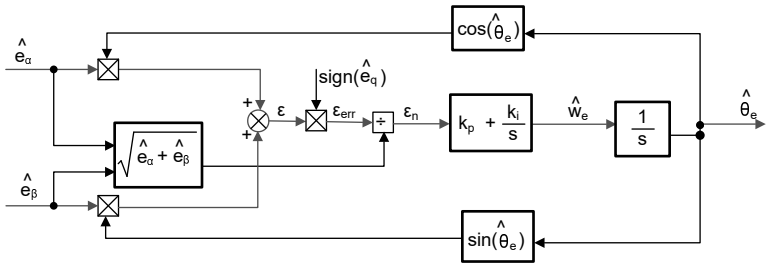


图 2-16. 锁相环位置跟踪器的方框图

由于 $e_\alpha=E\cos(\theta_e)$ ， $e_\beta=E\sin(\theta_e)$ 和 $E=\omega_e\lambda_{pm}$ ，位置误差可定义为：

$$\varepsilon=\hat{e}_\beta\cos(\hat{\theta}_e)-\hat{e}_\alpha\sin(\hat{\theta}_e)=E\sin(\theta_e)\cos(\hat{\theta}_e)-E\cos(\theta_e)\sin(\hat{\theta}_e)=E\sin(\theta_e-\hat{\theta}_e) \tag{21}$$

其中 E 是 EEMF 的幅度，与电机转速成正比 ω_e。当 $(\theta_e-\hat{\theta}_e)<\frac{\pi}{2}$ ，方程式 21 可以简化为

$$\varepsilon=E(\theta_e-\hat{\theta}_e) \tag{22}$$

可以进一步得到 EEMF 归一化后的位置误差：

$$\varepsilon_n=\theta_e-\hat{\theta}_e \tag{23}$$

根据分析，可以得到正交锁相环位置跟踪器的简化方框图，如图 1-1 所示。PLL 的闭环传递函数可表示为方程式 24：

$$\frac{\hat{\theta}_e}{\theta_e} = \frac{k_p s + k_i}{s^2 + k_p s + k_i} = \frac{2\xi\omega_n s + \omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \tag{24}$$

其中 k_p 和 k_i 是标准 PI 调节器的比例增益和积分增益，其固有频率 ω_n 和阻尼比 ξ 已给定：

$$k_p = 2\xi\omega_n, \qquad k_i = \omega_n^2 \tag{25}$$

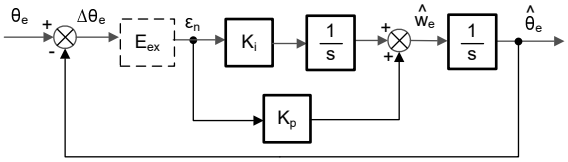


图 2-17. 锁相环位置跟踪器的简化方框图

2.4 电机驱动器的硬件必要条件

用于控制电机的算法利用电机条件的采样测量值，包括直流母线电源电压、每个电机相位上的电压、每个电机相位的电流。需要正确设置一些与硬件相关的参数才能正确识别电机并使用磁场定向控制 (FOC) 有效地运行电机。以下各节说明如何计算采用 FAST 或 eSMO 的压缩机和风扇电机控制的电流标度值、电压标度值和电压滤波器极点。

2.4.1 电机相电压反馈

FAST 估算器需要电机相电压反馈，以在最宽的速度范围内实现更高性能，相电压直接从电机相位测量，而不是使用软件估算。eSMO 依靠软件估算值来表示电压相位，而不使用电机相位电压检测电路。此软件值 (USER_ADC_FULL_SCALE_VOLTAGE_V) 取决于感测电机相电压反馈的电路。图 2-35 展示了如何使用基于电阻分压器的电压反馈电路对电机电压进行滤波和调整以适应 ADC 输入范围。类似的电路用于测量全部三个压缩机和风扇电机以及直流母线。

考虑到 ADC 输入的最大电压为 3.3V，该参考设计中的微控制器可测量的最大相电压反馈可通过公式 92 进行计算。

2.5 额外的控制特性

2.5.1 弱磁 (FW) 和每安培最大扭矩 (MTPA) 控制

永磁同步电机 (PMSM) 因其高功率密度、高效率和宽转速范围而广泛应用于家用电器应用。PMSM 包含两种主要类型：表面贴装式 PMSM (SPM) 和内嵌式 PMSM (IPM)。由于 SPM 电机在扭矩和 q 轴电流之间具有线性关系，因此更易于控制。不过，IPMSM 由于凸极比大而具有电磁扭矩和磁阻扭矩。总扭矩相对于转子角度是非线性的。因此，MTPA 技术可用于 IPM 电机，以优化恒定扭矩区域中的扭矩生成。弱磁控制的目的是优化以达到 PMSM 驱动器的最高功率和效率。弱磁控制可以使电机以其基本转速运行，扩大其运行限值以使转速高于额定转速，并允许在整个转速和电压范围内实现最佳控制。

IPMSM 数学模型的电压公式可以用 d-q 坐标来描述，如方程式 26 和方程式 27 所示。

$$v_d = L_d\frac{di_d}{dt} + R_s i_d - p\omega_m L_q i_q \tag{26}$$

$$v_q = L_q\frac{di_q}{dt} + R_s i_q + p\omega_m L_d i_d + p\omega_m \psi_m \tag{27}$$

图 1-1 显示了 IPM 同步电机的动态等效电路。

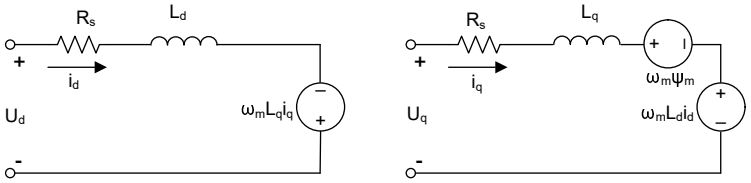


图 2-18. IPM 同步电机的等效电路

IPMSM 产生的总电磁扭矩可以由方程式 28 表示，产生的扭矩包含两个不同的项。第一项对应于扭矩电流 i_q 和永磁体 ψ_m 之间产生的相互反作用力扭矩，而第二项对应于由于 d 轴和 q 轴上的电感不同而产生的磁阻扭矩。

$$T_e = \frac{3}{2}p\left[\psi_m i_q + \left(L_d - L_q\right)i_d i_q\right] \tag{28}$$

在大多数应用中，IPMSM 驱动器具有转速和扭矩约束，这主要是由于分别存在逆变器或电机额定电流以及可用的直流链路电压限制。这些约束可以用数学公式方程式 29 和方程式 30 表示。

$$I_a = \sqrt{i_d^2 + i_q^2} \le I_{max} \tag{29}$$

$$V_a = \sqrt{v_d^2 + v_q^2} \le V_{max} \tag{30}$$

其中 V_{max} 和 I_{max} 是逆变器或电机允许的最大电压和电流。在两级三相电压源逆变器 (VSI) 供电的电机中，可实现的最大相电压受直流链路电压和 PWM 策略的限制。如果采用空间矢量调制 (SVPWM)，则最大电压限制为方程式 31 中所示的值。

$$\sqrt{v_d^2 + v_q^2} \le v_{max} = \frac{v_{dc}}{\sqrt{3}} \tag{31}$$

通常，定子电阻 R_s 在高速运行时可以忽略不计，并且电流的导数在稳态下为零，因此得到方程式 32，如下所示。

$$\sqrt{L_d^2\left(i_d + \frac{\psi_{pm}}{L_d}\right)^2 + L_q^2 i_q^2} \le \frac{V_{max}}{\omega_m} \tag{32}$$

方程式 29 的电流限制在 d-q 平面中产生一个半径为 I_{max} 的圆，而方程式 31 的电压限制产生一个椭圆，其半径 V_{max} 随着转速的增加而减小。必须对得到的 d-q 平面电流矢量进行控制，使其同时遵守电流和电压约束。根据这些约束，可以区分 IPMSM 的三个工作区域，如图 1-1 所示。

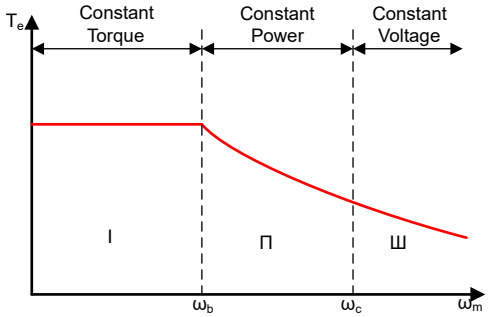


图 2-19. IPMSM 控制工作区域

1. 恒定扭矩区域：可以在该工作区域内实施 MTPA，以确保产生最大扭矩。
2. 恒定功率区域：必须采用弱磁控制，并且在达到电流约束时减小扭矩容量。
3. 恒定电压区域：在这个工作区域，深度弱磁控制使定子电压保持恒定，以尽可能大地产生扭矩。

在恒定扭矩区域，根据方程式 28，IPMSM 的总扭矩包括来自磁链的电磁扭矩和来自以下电感之间凸极的磁阻扭矩： L_d 和 L_q。电磁扭矩与 q 轴电流 i_q 成正比，磁阻扭矩与 d 轴电流 i_d、q 轴电流 i_q 以及 L_d 和 L_q。

SPM 电机的传统矢量控制系统仅通过将命令的 i_d 设置为零来实现非弱磁模式，从而利用电磁扭矩。但 IPMSM 将利用电机的磁阻扭矩，也应控制 d 轴电流。MTPA 控制的目的是计算基准电流 i_d 和 i_q 以尽可能增大产生的电磁扭矩与磁阻扭矩之间的比率。以下各公式显示了 i_d 和 i_q 之间的关系以及定子电流 I_s 的矢量和。

$$I_s = \sqrt{i_d^2 + i_q^2} \tag{33}$$

$$I_d = I_s \cos\beta \tag{34}$$

$$I_q = I_s \sin\beta \tag{35}$$

其中 β 是同步 (d-q) 坐标系中的定子电流角度。方程式 28 可以表示为方程式 36，其中 I_s 替换了 i_d 和 i_q。

方程式 36 表明电机扭矩取决于定子电流矢量的角度，因此：

$$T_e = \frac{3}{2} p I_s \sin\beta \left[\psi_m + (L_d - L_q) I_s \cos\beta \right] \tag{36}$$

当电机扭矩微分等于零时，可以计算出最大效率点。当该微分 $\frac{dT_e}{d\beta}$ 为零（如方程式 37 所示）时，可以找到 MTPA 点。

$$\frac{dT_e}{d\beta} = \frac{3}{2} p \left[\psi_m I_s \cos\beta + (L_d - L_q) I_s^2 \cos 2\beta \right] = 0 \tag{37}$$

接下来，可以通过方程式 38 得出 MTPA 控制的电流角度。

$$\beta_{mtpa} = \cos^{-1}\frac{-\psi_m + \sqrt{\psi_m^2 + 8^*\left(L_d - L_q\right)^2 * I_s^2}}{4^*\left(L_d - L_q\right)^* I_s} \tag{38}$$

因此，可以使用 MTPA 控制的电流角度通过方程式 39 和方程式 40 来表示有效的 d 轴和 q 轴基准电流。

$$I_d = I_s^* \cos\beta_{mtpa} \tag{39}$$

$$I_q = I_s^* \sin\beta_{mtpa} \tag{40}$$

不过，如方程式 38 所示，MTPA 控制的角度 β_{mtpa} 与 d 轴和 q 轴电感有关。这意味着电感的变化会阻碍找到最佳 MTPA 点。为了提高电机驱动器的效率，应在线估算 d 轴和 q 轴电感，但参数 L_d 和 L_q 不易于在线测量，并且受饱和效应的影响。稳健的查找表 (LUT) 方法可确保电气参数变化下的可控性。通常，为了简化数学模型，可以忽略 d 轴和 q 轴电感之间的耦合效应。因此，假设 L_d 仅随 i_d 而变化， L_q 仅随 i_q 而变化。因此，d 轴和 q 轴电感可以分别建模为其 d-q 电流的函数，如方程式 41 和方程式 42 所示。

$$L_d = f_1\left(i_d, i_q\right) = f_1\left(i_d\right) \tag{41}$$

$$L_q = f_2\left(i_q, i_d\right) = f_2\left(i_q\right) \tag{42}$$

为了通过简化方程式 38 来减轻 ISR 计算负担，基于电机参数的常数 K_{mtpa} 改为用方程式 44 表示，其中 K_{mtpa} 在后台循环中使用更新的 L_d 和 L_q 。

$$K_{mtpa} = \frac{\psi_m}{4^*\left(L_q - L_d\right)} = 0.25^*\frac{\psi_m}{\left(L_q - L_d\right)} \tag{43}$$

$$\beta_{mtpa} = cos^{-1}\left(K_{mtpa}/I_s - \sqrt{\left(K_{mtpa}/I_s\right)^2 + 0.5}\right) \tag{44}$$

第二个中间变量 G_{mtpa} 由方程式 45 进行表示，用于进一步简化计算。使用 G_{mtpa} ，MTPA 控制的角度 β_{mtpa} 可以通过方程式 46 进行计算。这两个计算在 ISR 中执行，以获得真实的电流角度 β_{mtpa} 。

$$G_{mtpa} = K_{mtpa}/I_s \tag{45}$$

$$\beta_{mtpa} = cos^{-1}\left(G_{mtpa} - \sqrt{G_{mtpa}^2 + 0.5}\right) \tag{46}$$

在所有情况下，都可以通过作用于直轴电流 i_d 。作为进入该恒定功率工作区域的结果，选择弱磁控制而不是在恒定功率和电压区域中使用的 MTPA 控制。由于最大逆变器电压受到限制，PMSM 电机无法在反电动势 (几乎与永磁场和电机转速成正比) 高于逆变器最大输出电压的转速区域中运行。在 PM 电机中，无法直接控制磁通量。不过，通过添加负 i_d ，由于出现 d 轴电枢反应，空气间隙通量会因为消磁作用而减弱。

考虑到电压和电流约束，电枢电流和端子电压会受到限制，如方程式 29 和方程式 30 所示。逆变器输入电压（直流链路电压）的变化限制了电机的最大输出。此外，最大基波电机电压还取决于所使用的 PWM 方法。在方程式 32 中，IPMSM 有两个因素：一个是永磁值，另一个是电感和磁通电流。

图 1-1 显示了用于实现弱磁的典型控制结构。β_{fW} 是弱磁 (FW) PI 控制器的输出，可生成基准 i_d 和 i_q。在电压幅度达到其限制之前，FW 的 PI 控制器的输入始终为正，因此输出始终在 0 处达到饱和。

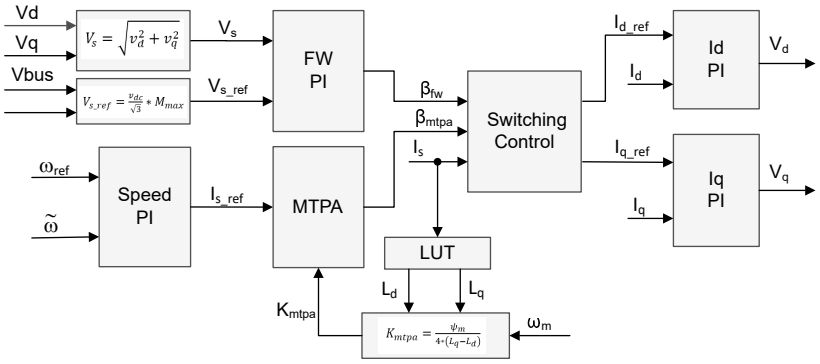


图 2-20. 弱磁和每安培最大扭矩控制的方框图

图 1-1 和图 1-1 显示了基于 FAST 或 eSMO 的 FOC 实现的方框图。这些方框图概述了 FOC 系统功能和变量。电机驱动 FOC 系统中有两个控制模块：一个是 MTPA 控制，一个是弱磁控制。这两个模块根据输入参数分别生成电流角度 β_{mtpa} 和 β_{fW}，如图 1-1 所示。

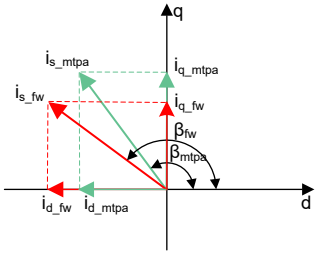


图 2-21. FW 和 MTPA 期间 IPMSM 的电流相量图

切换控制模块用于决定应用哪个角度，然后计算基准 i_d 和 i_q，如方程式 34 和方程式 35 所示。可以根据下面的方程式 47 和方程式 48 来选择电流角度。

$$\beta = \beta_{fw}\, if\, \beta_{fw} > \beta_{mtpa} \tag{47}$$

$$\beta = \beta_{mpta}\, if\, \beta_{fw} < \beta_{mtpa} \tag{48}$$

图 1-1 是显示在主循环和中断中运行采用 FW 和 MPTA 的 InstaSPIN-FOC 所需步骤的流程图。

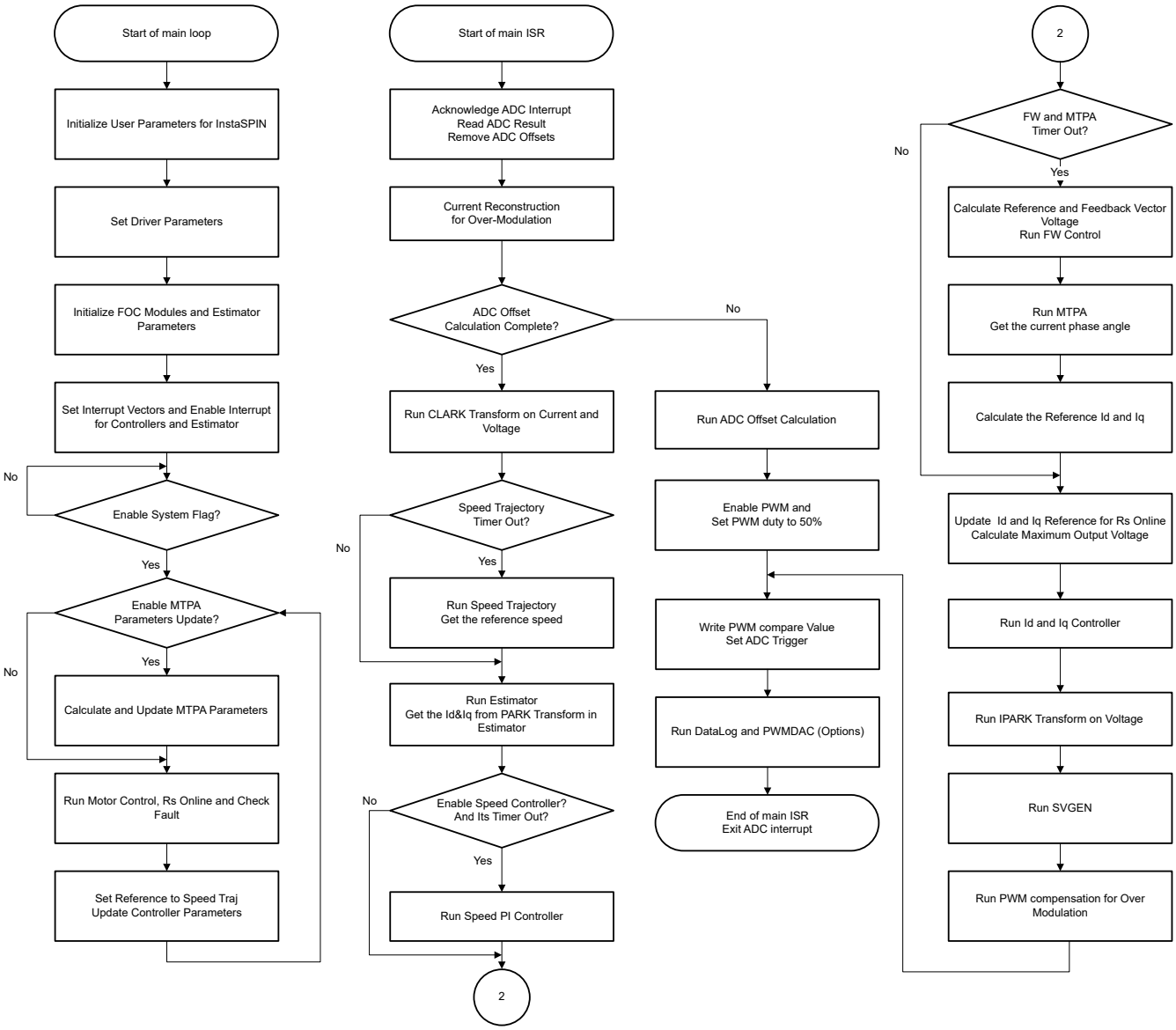


图 2-22. 采用 FW 和 MPTA 的 InstaSPIN-FOC 工程的流程图

2.5.2 快速启动

快速启动功能使驱动器能够确定旋转电机的转速和方向，并以该转速和方向开始输出电压和频率。如果没有快速启动功能，那么驱动器将以零伏和零转速开始其输出，并尝试增加至命令的速度。如果负载的惯性或旋转方向要求电机产生很大的扭矩，则可能会导致过大的电流，并且可能会在驱动器上发生过流跳闸。这些问题可以通过快速启动加以解决。

快速启动是指以除零以外的任何速度启动控制能力，这是空调应用中用于风扇驱动的重要功能。

当电机以正常模式启动时，控制器最初应用 0Hz 的频率并增加至所需的频率。如果驱动器在该模式下启动，并且电机已经以非零频率旋转，则会产生大电流。如果电流限制器反应不够快，则可能会导致过流跳闸。即使电流限制器足够快，可以防止过流跳闸，发生同步和电机达到其所需频率也可能需要不可接受的时长。此外，会在应用上施加更大的机械应力。

在快速启动模式下，驱动器对启动命令的响应是与电机的转速（频率和相位）和电压同步。然后电机加速至命令的频率。该过程可防止过流跳闸并显著减少电机达到其命令的频率所需的时间。由于驱动器以其旋转速度与电机同步并增加至适当的转速，因此几乎不存在机械应力或不存在机械应力。

快速启动功能实现了一种搜索转子转速的算法。该算法搜索与施加到电机上的励磁电流相对应的电机电压。

当电机旋转时，可以通过 BEMF 电压估算转速和位置信息。由于在 InstaSPIN 驱动器中测量定子电压，因此通过开关逆变器可以轻松获得转速和位置。向电机施加零扭矩电流并测量产生的电流和定子电压，然后 InstaSPIN-FOC 模块使用这些信号来估算转子位置和转速。

图 1-1 显示了具有快速启动功能的 FOC 的方框图，快速启动模块输出一个标志来启用或禁用转速闭环控制。设置了零基准扭矩电流，在快速启动功能运行时速度 PI 控制器输出被禁用。

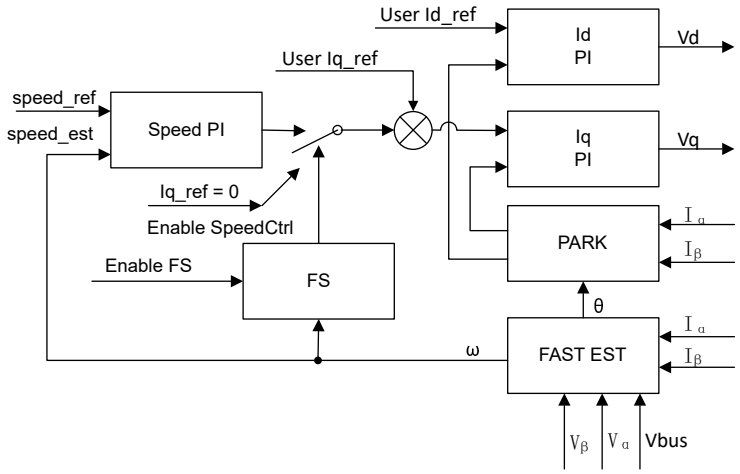


图 2-23. 快速启动控制方框图

如图 1-1 所示，模块例程禁用转速闭环控制，将基准 Iq 设置为零，并在启动期间启用 FOC 模块运行电机。在测量相电流和电压后，该例程运行 InstaSPIN-FOC 并且可以估算实际的电机转速。程序重新启用速度闭环控制，并在快速启动完成后设置转速基准值。

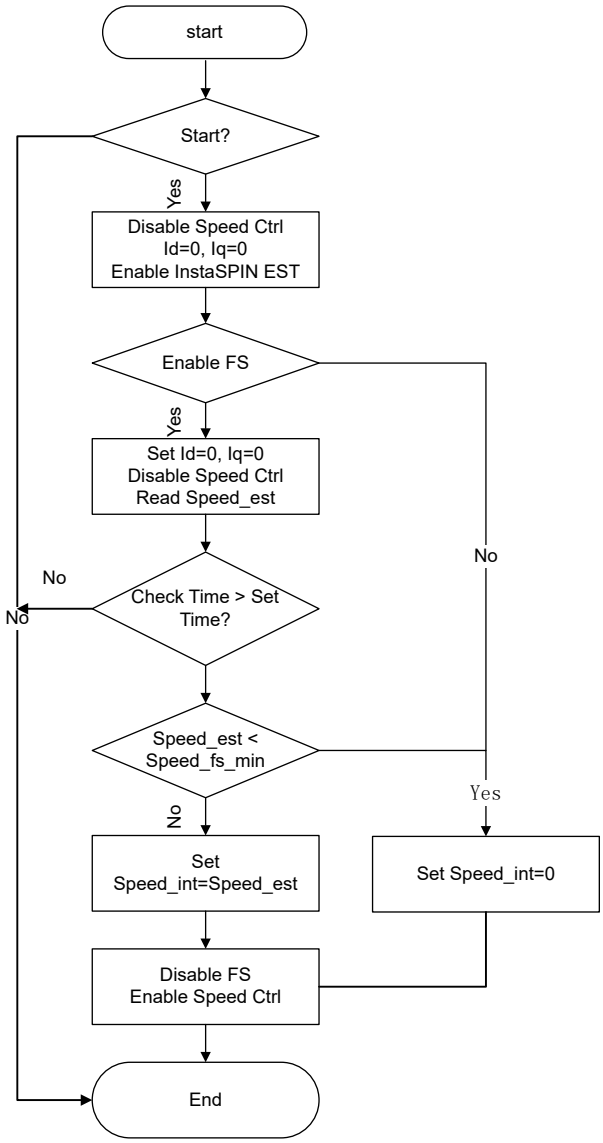


图 2-24. 快速启动模块程序流程图

3 在 TI 硬件套件上运行通用实验

3.1 受支持的 TI 电机评估套件

TMS320F28002x (F28002x)、TMS320F28003x (F28003x) 或 TMS320F280013x (F280013x) 是 C2000™ 实时微控制器系列的成员，具有 IEEE 754 浮点单元 (FPU) 和三角函数加速器 (TMU)。用户可将这些 LaunchPad™ 开发套件或 controlCARD 中的一个与相关电机驱动评估板配合使用，以评估该电机控制实验。

表 3-1 列出了 MotorControl SDK 中该通用电机控制实验工程支持的当前评估套件。

表 3-1. 电机控制 SDK 支持的电机驱动评估套件

电机驱动评估板		C2000 MCU 评估模块	电流检测拓扑	转子位置感应方法	测试电机
器件型号	说明				
DRV8329AEVM	具有 CSD18536KTTT NexFET™ 的 4.5V 至 60V、30A 三相逆变器	LAUNCHXL-F280025C LAUNCHXL-F280039C LAUNCHXL-F2800137	单分流器直流链路电流	基于 FAST 估算器的无传感器 FOC 基于 eSMO 观测器的无传感器 FOC 基于 QEP 编码器的有传感器 FOC 霍尔传感器的有传感器 FOC 无传感器梯形控制	LVSERVOMTR（嵌入式编码器） LVBLDCMTR（嵌入式霍尔传感器）
BOOSTXL-DRV8323RH	具有 CSD88599Q5DC NexFET™ 电源块的 6V 至 54V、15A 三相逆变器	LAUNCHXL-F280025C LAUNCHXL-F280039C LAUNCHXL-F2800137	单个低侧分流器	基于 FAST 估算器的无传感器 FOC 基于 eSMO 观测器的无传感器 FOC 基于 QEP 编码器的有传感器 FOC 基于霍尔传感器的有传感器 FOC	LVSERVOMTR（嵌入式编码器） LVBLDCMTR（嵌入式霍尔传感器）
BOOSTXL-DRV8323RS	具有 CSD88599Q5DC NexFETTM 电源块的 6V 至 54V、15A 三相逆变器	LAUNCHXL-F280025C LAUNCHXL-F280039C LAUNCHXL-F2800137	单个低侧分流器	基于 FAST 估算器的无传感器 FOC 基于 eSMO 观测器的无传感器 FOC 基于 QEP 编码器的有传感器 FOC 基于霍尔传感器的有传感器 FOC	LVSERVOMTR（嵌入式编码器） LVBLDCMTR（嵌入式霍尔传感器）
DRV8316REVM	4.5V 至 35V、8A 峰值电流三相逆变器集成式 MOSFET	LAUNCHXL-F280025C LAUNCHXL-F280039C LAUNCHXL-F2800137	用于三相低侧电流的集成 CSA	基于 FAST 估算器的无传感器 FOC 基于 eSMO 观测器的无传感器 FOC 基于 QEP 编码器的有传感器 FOC 基于霍尔传感器的有传感器 FOC	LVSERVOMTR（嵌入式编码器） LVBLDCMTR（嵌入式霍尔传感器）
DRV8353RS-EVM	具有 CSD19532Q5B 的 9V 至 95V 15A 三相逆变器	LAUNCHXL-F280025C LAUNCHXL-F280039C LAUNCHXL-F2800137	单个低侧分流器	基于 FAST 估算器的无传感器 FOC 基于 eSMO 观测器的无传感器 FOC 基于 QEP 编码器的有传感器 FOC 基于霍尔传感器的有传感器 FOC	LVSERVOMTR（嵌入式编码器） LVBLDCMTR（嵌入式霍尔传感器）
BOOSTXL-3PHGANINV	12V 至 60V、3.5A 三相 GaN 逆变器	LAUNCHXL-F280025C LAUNCHXL-F280039C LAUNCHXL-F2800137	三个基于采样电阻的内嵌式电机相电流检测	基于 FAST 估算器的无传感器 FOC 基于 eSMO 观测器的无传感器 FOC 基于 QEP 编码器的有传感器 FOC 基于霍尔传感器的有传感器 FOC	LVSERVOMTR（嵌入式编码器） LVBLDCMTR（嵌入式霍尔传感器）

表 3-1. 电机控制 SDK 支持的电机驱动评估套件（续）

电机驱动评估板		C2000 MCU 评估模块	电流检测拓扑	转子位置感应方法	测试电机
器件型号	说明				
TMDSHVMTRINSPIN	400V、10A 三相逆变器	TMDSCNCD280025C、TMDSCNCD280039C、TMDSCNCD2800137、带 TMDSADAP180TO1000	单个低侧分流器	基于 FAST 估算器的无传感器 FOC 基于 eSMO 观测器的无传感器 FOC 基于 QEP 编码器的有传感器 FOC	HVPMSMMTR（嵌入式编码器） HVBLDCMTR（嵌入式霍尔传感器）

如果实验设置为使用编码器或基于霍尔的有传感器 FOC，则务必确保以正确的顺序连接物理连接。如果电机、编码器或霍尔线的连接顺序错误，则实验将无法正常运行，可能导致电机无法旋转。对于电机相线，务必确保电机相位连接到逆变器板上的正确相位。对于随 TI 电机控制参考套件提供的电机，提供了正确的相位连接，如表 3-2 中所示。

对于编码器，务必确保 A 连接到 A，B 连接到 B，I 连接到 I。对于霍尔传感器，务必确保 A 连接到 A，B 连接到 B 以及 C 连接到 C。通常还需要 +5V 直流和接地连接。如果使用的霍尔传感器或编码器与表 2-2 中具体列出的传感器或编码器不同，请参阅所用霍尔传感器或编码器的用户手册，以确保正确连接电线。

务必为 ENC 模块的设置和配置提供编码器每旋转一周的时隙数。这使得 ENC 模块能够正确地将编码器信号转换为角度。需要将 *user_mtr1.h* 文件中定义的 USER_MOTOR1_NUM_ENC_SLOTS 常量更新为编码器的正确值。如果此值不正确，则电机会旋转得更快或更慢，具体取决于已设置的值。务必注意，该值应设置为编码器上的时隙数，而不是了解正交精度后得到的计数值。

表 3-2. 用于参考套件和电机的电机相位、编码器或霍尔传感器连接

		LVSERVOMTR	LVBLDCMTR	HVPMSMMTR	HVBLDCMTR
电机相线	U	黑色 (16AWG)	黄色	红色	黄色
	V	红色 (16AWG)	红色	蓝色/黑色	红色
	W	白色 (16AWG)	黑色	白色	黑色
编码器	GND（LAUNCHXL-F280025C/39C/137 的 J12-1）	黑色 (J4-1)	无，不支持基于编码器的有传感器 FOC	黑色	不支持基于编码器的有传感器 FOC
	+5V	红色 (J4-2)		红色	
	I（1I，LAUNCHXL-F280025C/39C/137 的 J12-3）	棕色 (J4-3)		黄色	
	B（1B，LAUNCHXL-F280025C/39C/137 的 J12-4）	橙色 (J4-4)		绿色	
	A（1A，LAUNCHXL-F280025C/39C/137 的 J12-5）	蓝色 (J4-1)		蓝色	
霍尔传感器（LAUCHXL_F280013737 仅具有 J12，霍尔传感器与编码器共享 J12）	GND	黑色 (J10-1)	黑色	不支持基于霍尔传感器的有传感器 FOC	黑色
	+5V	红色 (J10-2)	红色		红色
	A（2I，LAUNCHXL-F280025C/39C 的 J13-3）	灰白色 (J10-3)	蓝色		蓝色
	B（2B，LAUNCHXL-F280025C/39C 的 J13-4）	绿色-白色 (J10-4)	绿色		绿色
	C（2A，LAUNCHXL-F280025C/39C 的 J13-5）	绿色 (J10-5)	白色		白色

立即开始使用 C2000™ 实时控制微控制器 (MCU) 来实现电机控制。

1. 第 1 步：订购所需的电机驱动评估板、C2000 MCU 评估模块和电机，如表 3-1 所示。
2. 第 2 步：下载最新版本的 MotorControl SDK。
3. 第 3 步：下载最新版本的 Code Composer Studio IDE。
4. 第 4 步：按照节 3.2 中的说明设置硬件并运行以下各节中所述的示例实验。
5. 第 5 步：有关您可能提出的任何设计问题的答案，您可以使用 TI C2000 E2E 论坛搜索现有答案或提出自己的问题。

3.2 硬件电路板设置

本节介绍了在将电机驱动器评估板与 C2000 开发工具结合时如何设置用于电机控制的硬件板。以下各节显示了不同电机驱动器评估板上的详细操作过程。

3.2.1 LAUNCHXL-F280025C 设置

LAUNCHXL-F280025C 是一款适用于 TI C2000 实时微控制器系列 F28002x 器件的低成本开发板。该 LaunchPad™ 套件可提供额外引脚用于开发，并支持连接两个 BoosterPack™ 插件模块。

- 硬件文件位于 C2000Ware 的 `<install_location>\boards\LaunchPads\LAUNCHXL_F280025C` 文件夹中。
- 有关 LAUNCHXL-F280025C 的更多详细信息，请参阅 *F28002x 系列 LaunchPad™ 开发套件用户指南*。
- 确保按图 1-1 中所示设置 LAUNCHXL-F280025C 上的开关。
  - 在用于电源和调试 JTAG 的 JP1、JP2、JP3 和 J101 上安装跳线。并在 JP8 上为所用 DAC128S 电路板的电源安装跳线。
  - 对于 **S2**，将 SEL1（左）开关置于 **UP** (1) 以将 GPIO28 和 GPIO29 路由至 BoosterPack 连接器，然后将 SEL2（右）开关置于 **UP** (1)，以将 GPIO16 和 GPIO17 连接至 XDS110 调试器的虚拟 COM 端口。
  - 对于 **S3**，将 SEL1（左）开关置于 **DOWN**，以将 GPIO24 拉低至逻辑 0，并将 SEL2（右）开关置于 **UP**，以将 GPIO32 拉高至逻辑 1，从而将 F280025C 置于等待引导模式，进而降低出现连接问题或先前加载代码执行的风险。
  - 对于 **S4**，如果在工程属性中设置了预定义符号“CMD_CAN_EN”，则将 S4 设置为 **DOWN**（导通），将 GPIO32 和 GPIO33 路由到 CAN 收发器 J14 CAN 接口。
  - 对于 **S5**，请将 QEP1 SEL（左）开关置于 **DOWN** 位置，以将 GPIO44/37/43 路由到 J12 上的 eQEP1 编码器接口，然后将 QEP2 SEL（右）开关置于 **DOWN** 位置，以将 GPIO14/25/26 路由到 J13 上的 eQEP2 霍尔传感器接口。

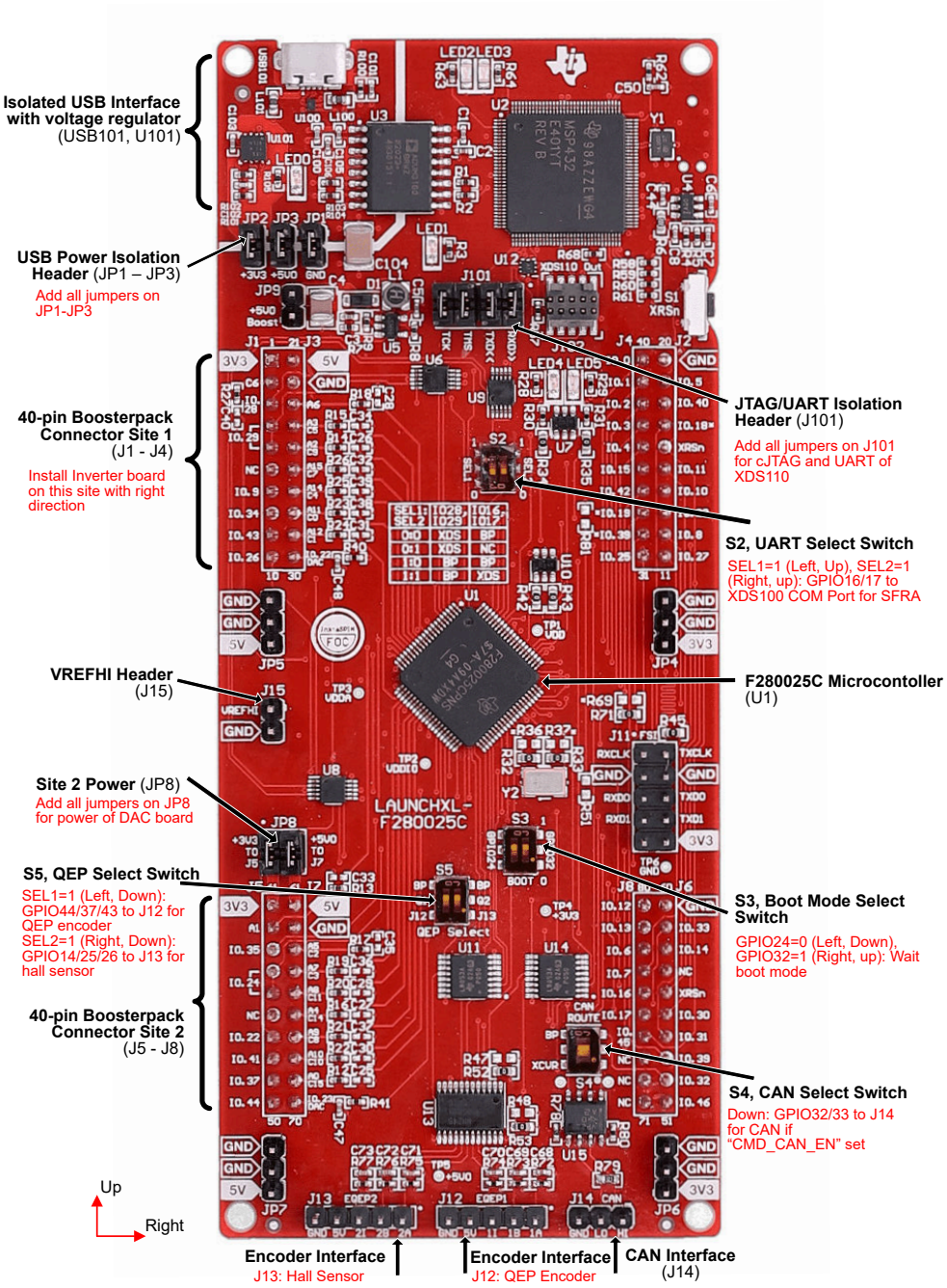


图 3-1. F280025C LaunchPad 电路板概述和开关设置

3.2.2 LAUNCHXL-F280039C 设置

LAUNCHXL-F280039C 是一款适用于 TI C2000 实时微控制器系列 F28003x 器件的低成本开发板。该 LaunchPad™ 套件可提供额外引脚用于开发，并支持连接两个 BoosterPack™ 插件模块。

- 硬件文件位于 C2000Ware 的 `<install_location>\boards\LaunchPads\LAUNCHXL_F280039C` 文件夹中。
- 有关 LAUNCHXL-F280039C 的更多详细信息，请参阅 *C2000™ F28003x 系列 LaunchPad™ 开发套件用户指南*。

- 确保按图 1-1 中所示设置 LAUNCHXL-F280039C 上的开关。
  - 在用于电源和调试 JTAG 的 JP1、JP2 和 J101 上安装跳线。并在 JP8 上为所用 DAC128S 电路板的电源安装跳线。
  - 对于 **S2**，将 SEL1（左）开关置于 **UP** (1) 以将 GPIO28 和 GPIO29 路由至 BoosterPack 连接器，然后将 SEL2（右）开关置于 **UP** (1)，以将 GPIO15 和 GPIO56 连接至 XDS110 调试器的虚拟 COM 端口。
  - 对于 **S3**，将 GPIO24（左）开关置于 **DOWN**，以将 GPIO24 拉至逻辑 0，并将 GPIO32（右侧）开关置于 **UP**，以将 GPIO32 拉高至逻辑 1，从而将 F280039C 置于等待引导模式，进而降低出现连接问题或先前加载代码执行的风险。
  - 对于 **S4**，如果在工程项目属性中设置了预定义符号“CMD_CAN_EN”，则将 S4 设置为 **DOWN**（导通），以将 GPIO4 和 GPIO5 路由到 CAN 收发器接口 J14。否则，将 S4 设为 **UP**（关断），以将 GPIO4 和 GPIO 5 路由至 BoosterPack 连接器。
  - 对于 **S5**，请将 QEP1 SEL（左侧）开关置于 **DOWN** 位置，以将 GPIO40/41/59 连接到 J12 上的 eQEP1 编码器接口，然后将 QEP2 SEL（右侧）开关置于 **DOWN** 位置，以将 GPIO14/55/57 连接到 J13 上的 eQEP2 霍尔传感器接口。

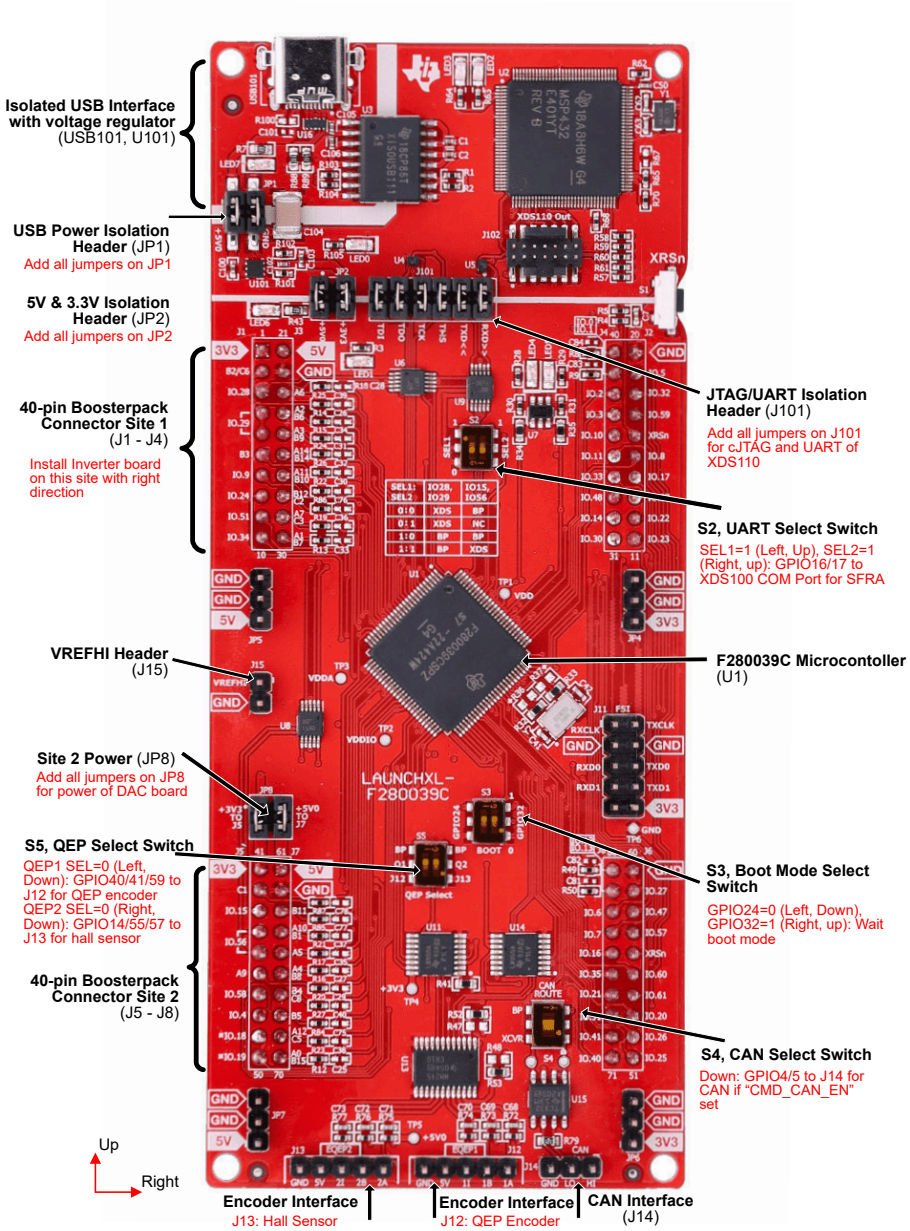


图 3-2. F280039C LaunchPad 电路板概述和开关设置

3.2.3 LAUNCHXL-F2800137 设置

LAUNCHXL-F2800137 是一款适用于 TI C2000 实时微控制器系列 F280013x 器件的低成本开发板。该 LaunchPad™ 套件可提供额外引脚用于开发，并支持连接两个 BoosterPack™ 插件模块。

- 硬件文件位于 C2000Ware 的 `<install_location>\boards\LaunchPads\LAUNCHXL_F2800137` 文件夹中。
- 有关 LAUNCHXL-F2800137 的更多详细信息，请参阅 *C2000™ F2800137 系列 LaunchPad™ 开发套件用户指南*。
- 确保按图 1-1 中所示设置 LAUNCHXL-F2800137 上的开关。
  - 在用于电源和调试 JTAG 的 JP1、JP2 和 J101 上安装跳线。并在 JP8 上为所用 DAC128S 电路板的电源安装跳线。
  - 对于 **S2**，将 SEL1 开关设置到 **UP** (1)，以将 GPIO28 和 GPIO29 路由至 BoosterPack 连接器。通过将 SEL1 设置为 **DOWN** (0)，通过 XDS110 调试器的虚拟 COM 端口，BOOSTXL-3PHGANINV 仅在 LAUNCHXL-F2800137 上提供 SFRA 支持。其他套件需要 GPIO29 启用 DRV 器件。这是由于 LAUNCHXL-F2800137 上的引脚限制。
  - 对于 **S3**，将 GPIO24（左）开关置于 **DOWN**，以将 GPIO24 拉低至逻辑 0，并将 GPIO32（右）开关置于 **UP**，以将 GPIO32 拉高至逻辑 1，从而将 F2800137 置于等待引导模式，进而降低出现连接问题或先前加载代码执行的风险。
  - 对于 **S4**，如果在工程属性中设置了预定义符号“CMD_CAN_EN”，则将 CAN ROUTE 开关设置到 **DOWN** 位置，以将 GPIO4 和 GPIO5 路由到 J14 CAN 接口。将 CAN ROUTE 开关设置到 **UP**（关）位置，以将 GPIO4 和 GPIO 5 连接到 BoosterPack 连接器。
  - 对于 **S5**，请将 QEP1 SEL 开关设置到 **DOWN** 位置，以将 GPIO40/41/39 路由至 eQEP1，用于 J12 上的编码器接口。（LAUNCHXL-F2800137 上没有 J13。默认情况下，霍尔传感器读数不可用。）

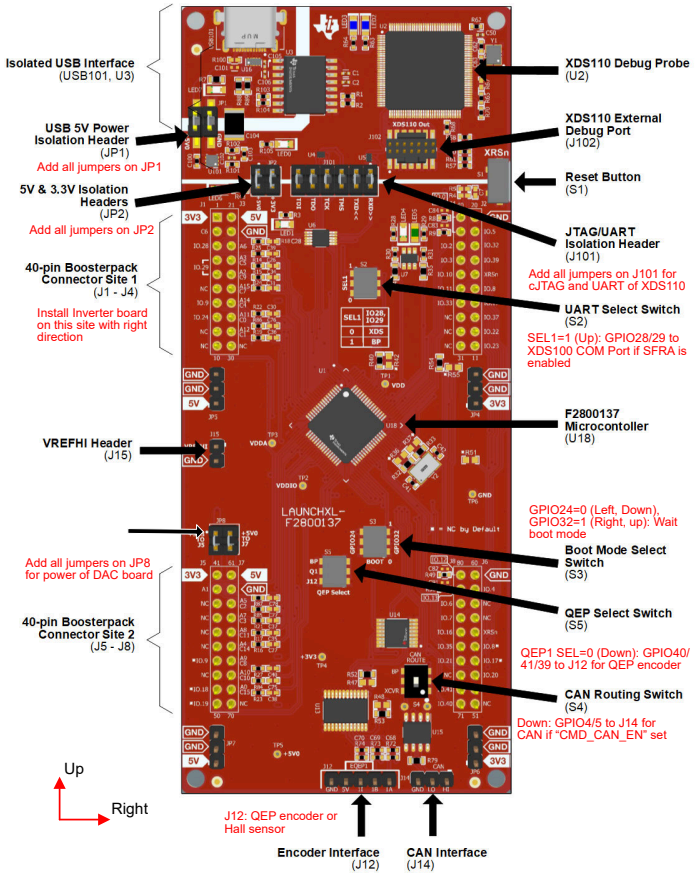


图 3-3. F2800137 LaunchPad 电路板概述和开关设置

3.2.4 TMDSCNCD280025C 设置

TMDSCNCD280025C 是一款适用于 TI C2000™ MCU 系列 TMS320F28002x 器件的低成本评估和开发板。它附带 HSEC180（180 引脚高速边缘连接器），可用于基于 DIMM 的现有 100 引脚 TMDSHVMTRINSPIN（具有 TMDSADAP180TO100 适配器）

- 硬件文件位于 C2000Ware 的 `<install_location>\boards\controlCARDs\TMDSCNCD280025C` 文件夹。
- 有关 TMDSCNCD280025C 的更多详细信息，请参阅 TMS320F280025C controlCARD 信息指南。
- 确保按照图 1-1 中所示设置 TMDSCNCD280025C 上的开关。
  - 对于 S1:A，将 SEL1（左）和 SEL2（右）开关置于 **UP**，以使用卡上 XDS100v2 仿真器和 ISO UART 通信。
  - 对于 S1，将 SEL1（左）和 SEL2（右）开关置于 **DOWN**，以将 GPIO24 和 GPIO25 路由至 HSEC 连接器。
  - 对于 S2，将 SEL1（左）和 SEL2（右）开关置于 **DOWN**，以将 GPIO26 和 GPIO27 路由至 HSEC 连接器。
  - 对于 S3，将开关置于 **UP** 位置，以启用外部晶体。
  - 对于 S4，将 SEL1（左）开关置于 **DOWN**，以将 GPIO24 拉低至逻辑 0，并将 SEL2（右）开关置于 **UP**，以将 GPIO32 拉高至逻辑 1，从而将 F280025C 置于等待引导模式，进而降低出现连接问题或先前加载代码执行的风险。
  - 对于 S5，将 SEL1（左）开关置于 **DOWN**，使 A8/C11 路由至 HSEC 引脚 30，然后将 SEL2（右）开关置于 **UP** 以启用内部电压基准。

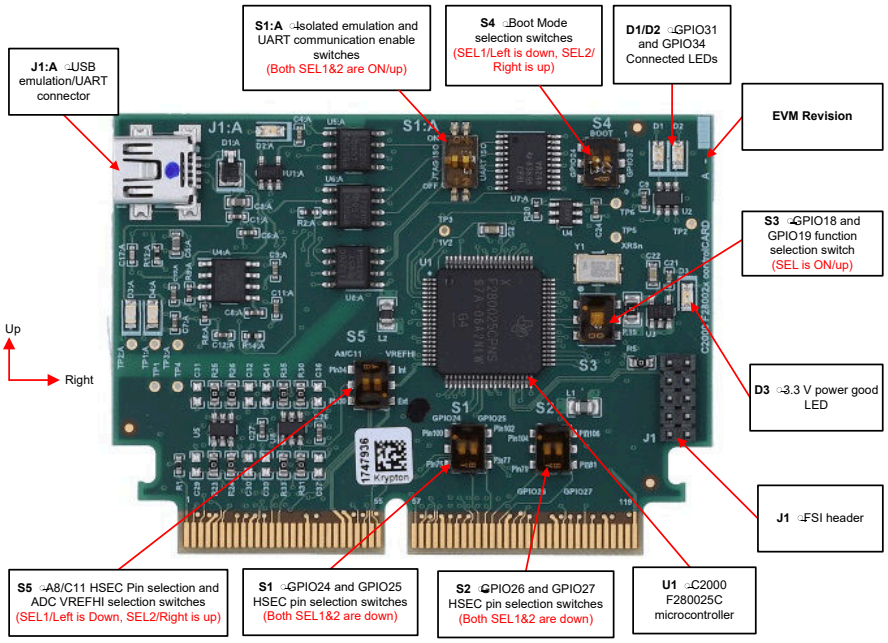


图 3-4. F28002x controlCARD 和开关设置

3.2.5 TMDSCNCD280039C 设置

TMDSCNCD280039C 是一款适用于 TI C2000 MCU 系列 TMS320F28003x 器件的低成本评估和开发板。它附带 HSEC180（180 引脚高速边缘连接器），可用于基于 DIMM 的现有 100 引脚 TMDSHVMTRINSPIN（具有 TMDSADAP180TO100 适配器）

- 硬件文件位于 C2000Ware 的 `<install_location>\boards\controlCARDs\TMDSCNCD280039C` 文件夹。
- 有关 TMDSCNCD280039C 的更多详细信息，请参阅 TMS320F280039C controlCARD 信息指南。
- 确保按照图 1-1 中所示设置 TMDSCNCD280039C 上的开关。
  - 对于 S1:A，将 SEL1（上）和 SEL2（下）开关置于**左侧**，以使用卡上 XDS110 仿真器和 ISO UART 通信。
  - 对于 S1，将开关置于 **UP** 位置，以启用外部晶体。
  - 对于 S2，将 SEL1（左）开关置于 **DOWN**，以将 GPIO24 拉低至逻辑 0，并将 SEL2（右）开关置于 **UP**，以将 GPIO32 拉高至逻辑 1，从而将 F280039C 置于等待引导模式，进而降低出现连接问题或先前加载代码执行的风险。
  - 对于 S3，将开关置于 **DOWN** 位置，以启用内部电压基准。

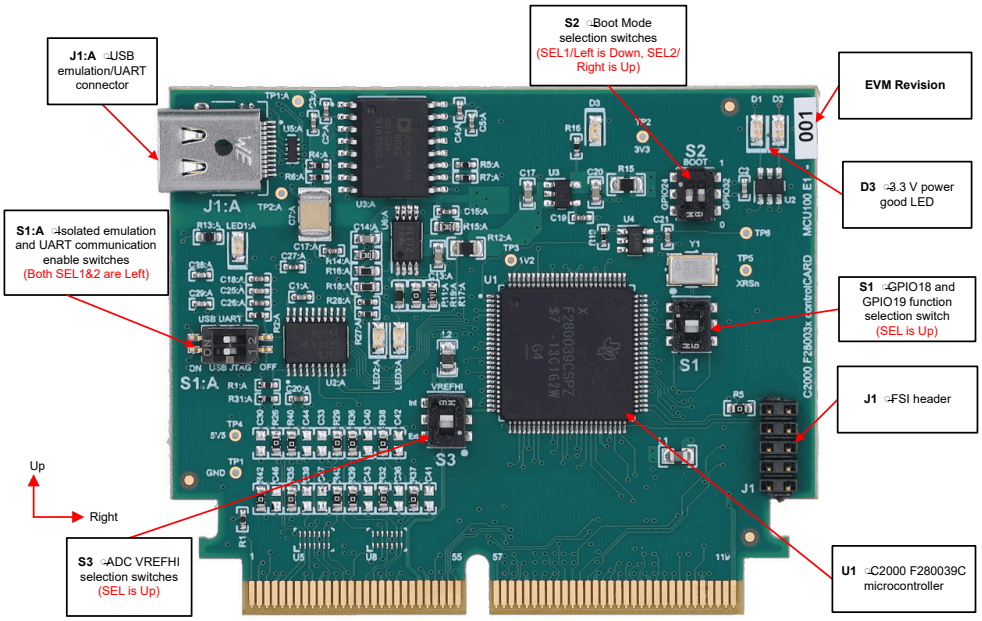


图 3-5. F280039C controlCARD 和开关设置

3.2.6 TMDSCNCD2800137 设置

TMDSCNCD2800137 是一款适用于 TI C2000 MCU 系列 TMS320F280013x 器件的低成本评估和开发板。它附带 HSEC180（180 引脚高速边缘连接器），可用于基于 DIMM 的现有 100 引脚 TMDSHVMTRINSPIN（具有 TMDSADAP180TO100 适配器）

- 硬件文件位于 C2000Ware 的 `<install_location>\boards\controlCARDs\TMDSCNCD2800137` 文件夹。
- 有关 TMDSCNCD2800137 的更多详细信息，请参阅 TMS320F2800137 controlCARD 信息指南。
- 确保按照图 1-1 中所示设置 TMDSCNCD2800137 上的开关。
  - 对于 S1:A，将 SEL1（上）和 SEL2（下）开关置于**左侧**，以使用卡上 XDS110 仿真器和 ISO UART 通信。
  - 对于 S1，将开关置于 **UP** 位置，以将 GPIO35 和 GPIO37 路由至 HSEC 连接器。
  - 对于 S2，将开关置于 **UP** 位置，以启用外部晶体。
  - 对于 S3，将 SEL1（左）开关置于 **DOWN**，以将 GPIO24 拉低至逻辑 0，并将 SEL2（右）开关置于 **UP**，以将 GPIO32 拉高至逻辑 1，从而将 F2800137 置于等待引导模式，进而降低出现连接问题或先前加载代码执行的风险。
  - 对于 S4，将 SEL1（左）开关置于 **DOWN**，使 A8/C11 路由至 HSEC 引脚 30，然后将 SEL2（右）开关置于 **UP** 以启用内部电压基准。

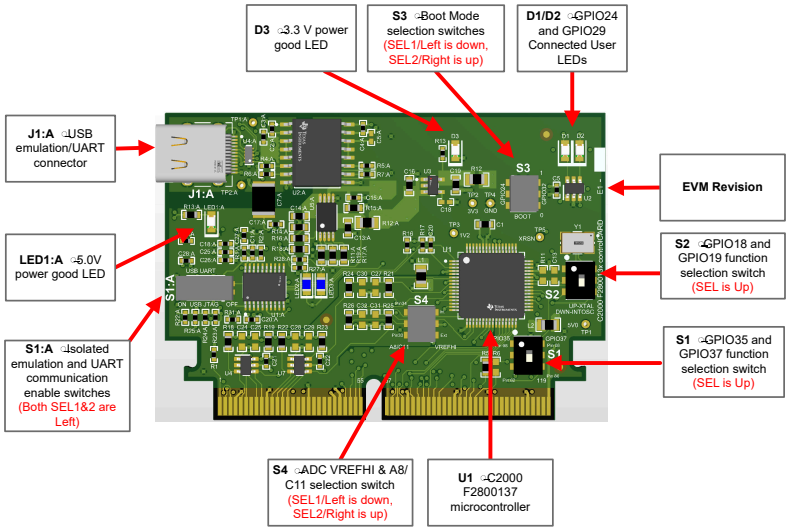


图 3-6. F2800137 controlCARD 和开关设置

3.2.7 TMDSADAP180TO100 设置

TMDSADAP180TO100 适配器允许将 180 引脚 C2000 controlCARD 和现有 100 引脚基于 DIMM 的评估工具结合使用。TMDSCNCD280025C、TMDSCNCD280039C 或 TMDSCNCD2800137 controlCARD 需要在 TMDSHVMTRINSPIN 上使用 TMDSADAP180TO100。

- 硬件文件位于 C2000Ware 的 `<install_location>\boards\controlCARDs\TMDSADAP180TO100` 文件夹。
- 确保开关 TMDSADAP180TO100 按如下所述或按照图 1-1 所示进行设置。
  - S1 开关需要放置在**右侧**，S2、S3 和 S4 开关需要放置在**左侧**。

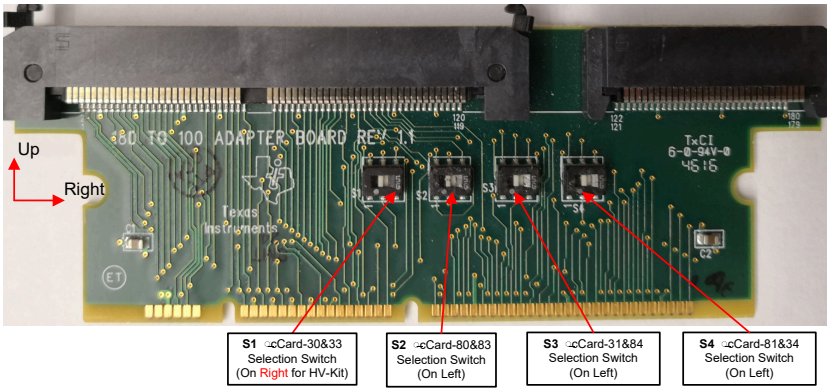


图 3-7. TMDSADAP180TO100 适配器和开关设置

3.2.8 DRV8329AEVM 设置

DRV8329AEVM 是一款基于 DRV8329A 栅极驱动器和 CSD18536KT MOSFET（适用于 BLDC/PMSM 电机）的 4.5V 至 60V、30A 三相无刷直流驱动级。该模块具有独立的直流母线和三相电压检测以及单分流器直流链路电流分流放大器，因此采用 C2000 LaunchPad 的 BLDC/PMSM 控制板适合用于无传感器 InstaSPIN-FOC 算法。驱驱动级受到全面的短路、过热、击穿和欠压保护，并可通过器件 GPIO 与 C2000 MCU 的连接轻松配置。

- DRV8329AEVM 硬件设计文件位于 TI.com 上的 DRV8329AEVM 页面中上。
- 有关 DRV8329AEVM 的更多详细信息，请参阅 EVM 用户指南。
- 确保按照所述完成以下各项，然后将 DRV8329AEVM 连接到 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的站点 1（J1/J3 和 J4/J2），如图 1-1 所示。
  - DRV8329AEVM 连接在 LaunchPad 的顶部，因此需要使用公头转母头适配器将 DRV8329AEVM 上的接头连接到 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 接头。
  - 如果使用 DAC128S085EVM 电路板，请将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J5-42 弯曲 90 度，以避免将该引脚连接到 DAC128S085EVM 电路板。此 EVM 板仅用于调试目的，可监控各种软件变量。
- 按照表 3-2 中所述将电机、编码器和霍尔传感器连接到 DRV8329AEVM 和 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137，如图 1-1 中所示。
- 将由电池或直流电压源提供的 9V 至 54V 电源电压连接到电压电源引脚。只有在节 3.5 中指示时才应接通电源，否则请保持断开连接。

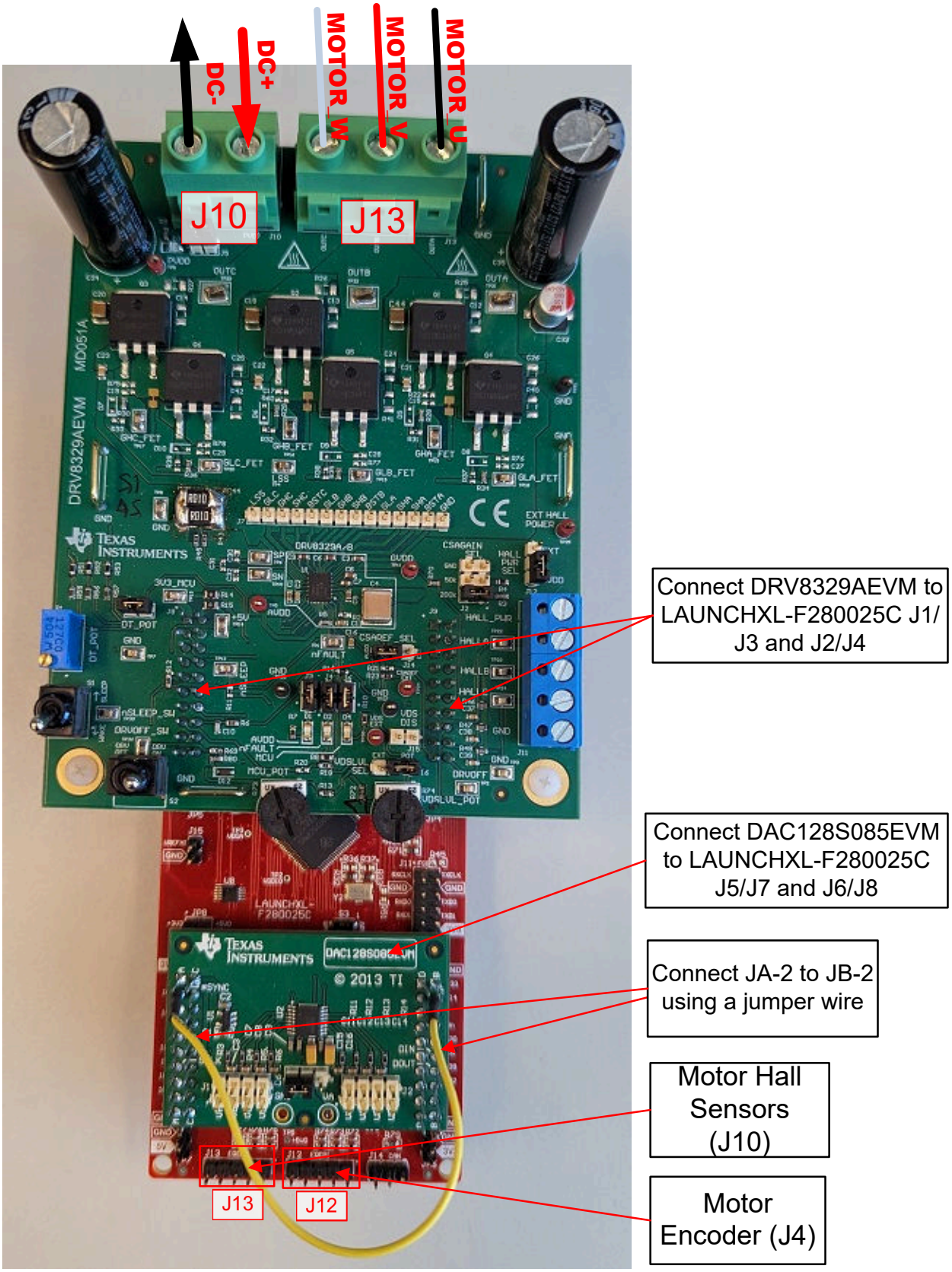


图 3-8. LAUNCHXL-F280025C 已连接至 DRV8329AEVM 和 DAC128S085EVM

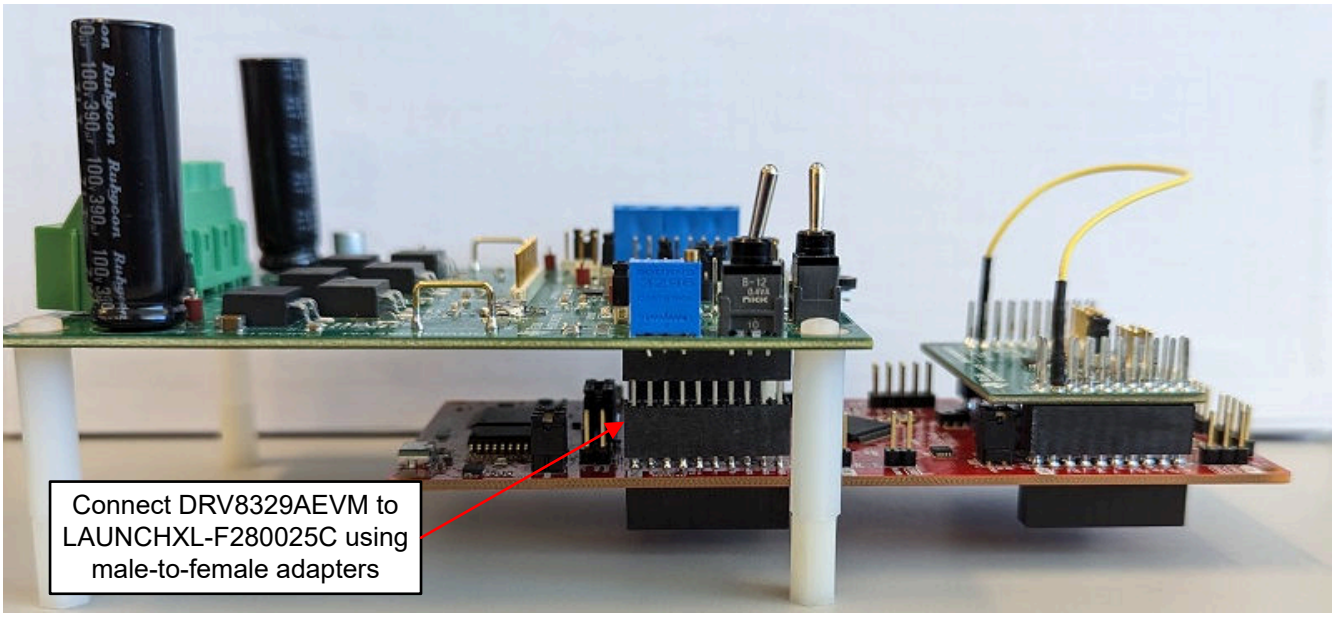


图 3-9. LAUNCHXL-F280025C 已连接至 DRV8329AEVM 和 DAC128S085EVM，侧视图

3.2.9 BOOSTXL-DRV8323RH 设置

BOOSTXL-DRV8323RH 是基于 DRV8323RH 栅极驱动器和 CSD88599Q5DC NexFETTM 电源块的 15A 三相无刷直流驱动级。该模块具有独立的直流母线和三相电压检测以及三个低侧电流分流放大器，因此采用 C2000 LaunchPad™ 的 BLDC/PMSM 控制板适合用于无传感器 InstaSPIN-FOC 算法。

- BOOSTXL-DRV8323RH 硬件文件位于 TI.com 上的 BOOSTXL-DRV8323RH 页面中。
- 有关 BOOSTXL-DRV8323RH 的更多详细信息，请参阅 EVM 用户指南。
- 确保按照所述完成以下各项，然后将 BOOSTXL-DRV8323RH 连接到 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J1/J3 和 J4/J2，如图 1-1 所示。
  - 使用 **47nF** 电容器组装 C9、C10 和 C11。
  - 将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J3-29 和 J3-30 弯曲 90°，以便它们不连接到 BOOSTXL-DRV8323RH，如图 1-1 所示。
  - 如果使用 DAC128S085EVM 电路板，请将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J5-42 弯曲 90 度，以避免将该引脚连接到 DAC128S085EVM 电路板。此 EVM 板仅用于调试目的，可监控各种软件变量。
  - 如果需要在 BOOSTXL-DRV8323RH 上使用电位器功能来设置基准速度，请使用跳线将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J3-29 连接到 BOOSTXL-DRV8323RH 的 J3-11。
- 按照表 3-2 中所述将电机、编码器和霍尔传感器连接到 BOOSTXL-DRV8323RH 和 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 板，如图 1-1 中所示。
  - 由电池或直流电压源提供的 6V 至 40V 电源电压连接到电压电源引脚，如图 1-1 所示。BOOSTXL-DRV8323RH 的最大电源电压额定值为 65V，因此必须注意确保运行期间电压不超过此值。这在减速或制动电机时尤为重要，因为这会导致电源电压大幅上升。只有在节 3.5 中指示时才应接通电源，否则请保持断开连接。

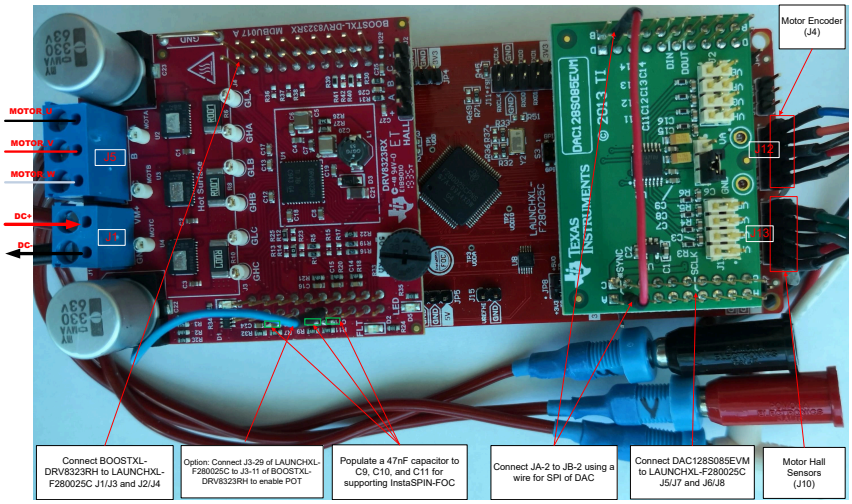


图 3-10. LAUNCHXL-F280025C/F280039C/F2800137 连接到 BOOSTXL-DRV8323RH 和 DAC128S085EVM

LAUNCHXL-F280025C 上的开关和连接（如图 1-1 中所示）可与 BOOSTXL-DRV8323RH、BOOSTXL-DRV8323RS 和 DRV8353RS-EVM 配合使用。

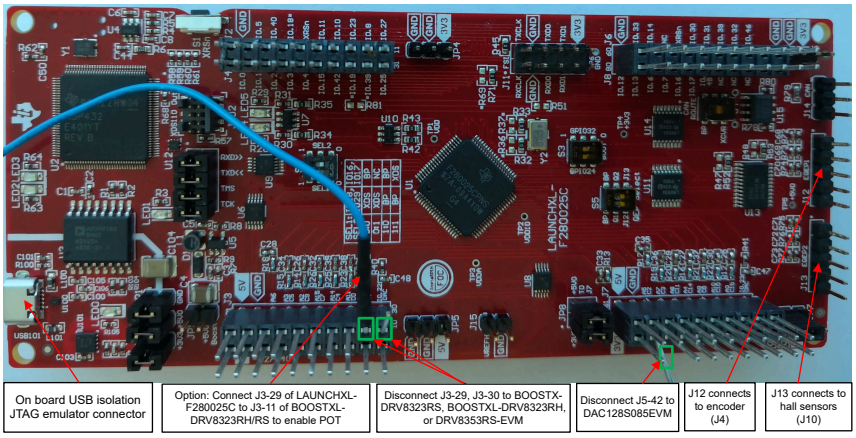


图 3-11. 用于连接 BOOSTXL-DRV 板的 LAUNCHXL-F280025C 板修改

3.2.10 BOOSTXL-DRV8323RS 设置

BOOSTXL-DRV8323RS 是基于 DRV8323RS 栅极驱动器和 CSD88599Q5DC NexFETTM 电源块的 15A 三相无刷直流驱动级。该模块具有独立的直流母线和三相电压检测以及三个低侧电流分流放大器，因此采用 C2000 LaunchPad 的 BLDC/PMSM 控制板适合用于无传感器 InstaSPIN-FOC 算法。该驱动级具有 IDRIVE 配置和故障引脚，还可以通过 C2000 MCU 提供可配置 SPI 来防止出现短路、过热、击穿和欠压等状况。

- BOOSTXL-DRV8323RS 硬件文件位于 TI.com 内的 BOOSTXL-DRV8323RS 页面上。
- 有关 BOOSTXL-DRV8323RS 的更多详细信息，请参阅 EVM 用户指南。
- 确保按照所述完成以下各项，然后将 BOOSTXL-DRV8323RS 连接到 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J1/J3 和 J4/J2，如图 1-1 所示。
  - 使用 47nF 电容器组装 C9、C10 和 C11。
  - 将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J3-29 和 J3-30 弯曲 90°，以使其不连接到 BOOSTXL-DRV8323RS，如图 1-1 所示。

  - 如果使用 DAC128S085EVM 电路板，请将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J5-42 弯曲 90 度，以避免将该引脚连接到 DAC128S085EVM 电路板。此 EVM 板仅用于监控各种软件变量的调试目的，是可选的，对于电机控制则不是必需的。
  - 将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J2-12 弯曲 90°，以使其不连接到 BOOSTXL-DRV8323RS。对于 LAUNCHXL-F280025C 和 LAUNCHXL-F280039C，使用跳线连接 BOOSTXL-DRV8323RS 的 J4-18(nSCS/GAIN) 和 J4-4 以直接使用 C2000 器件的 SPIA-STE，如图 1-1 所示。对于 LAUNCHXL-F2800137，使用跳线连接 BOOSTXL-DRV8323RS 的 J4-18(nSCS/GAIN) 和 J4-16。
  - 如果需要在 BOOSTXL-DRV8323RS 上使用电位器功能来设置基准速度，请使用跳线将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J3-29 连接到 BOOSTXL-DRV8323RS 的 J3-11。
- 按照表 3-2 中所述将电机、编码器和霍尔传感器连接到 BOOSTXL-DRV8323RS 和 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 板，如图 1-1 中所示。
  - 由电池或直流电压源提供的 6V 至 40V 电源电压连接到电压电源引脚，如图 1-1 所示。DRV8323RS 的 ABS 最大电源电压额定值为 65V，因此必须注意确保运行期间电压不超过此值。这在减速或制动电机时尤为重要，因为这会导致电源电压大幅上升。只有在节 3.5 中指示时才应接通电源，否则请保持断开连接。

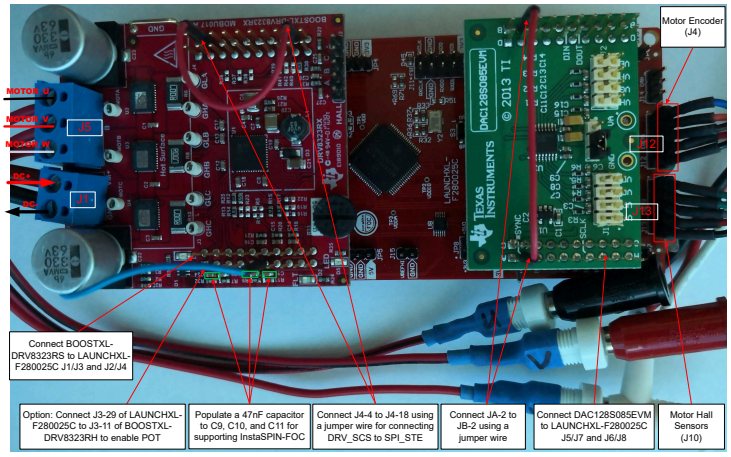


图 3-12. LAUNCHXL-F280025C/F280039C/F2800137 连接到 BOOSTXL-DRV8323RS 和 DAC128S085EVM

3.2.11 DRV8353RS-EVM 设置

DRV8353RS-EVM 是基于 DRV8353RS 栅极驱动器和 CSD19532Q5B NexFETTM 电源块的 15A 三相无刷直流驱动级。该模块具有独立的直流母线和三相电压检测以及三个低侧电流分流放大器，因此采用 C2000 LaunchPad 的 BLDC/PMSM 控制板适合用于无传感器 InstaSPIN-FOC 算法。驱动级受到全面的短路、过热、击穿和欠压保护，并可通过器件 SPI 寄存器与 C2000 MCU 的连接轻松配置。

- DRV8353Rx-EVM 设计文件位于 TI.com 上的 DRV8353RS-EVM 页面中。
- 有关 DRV8353RS-EVM 的更多详细信息，请参阅 EVM 用户指南。
- 确保按照如下所述完成以下各项，然后将 DRV8353RS-EVM 连接到 LAUNCHXL-F280025C 的站点 1（J1/J3 和 J4/J2），如图 1-1 所示。
  - DRV8353RS-EVM 连接在 LaunchPad 的底部，因此需要使用公头转公头适配器将 DRV8353RS-EVM 上的接头连接到 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 接头。需要从 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 上断开与 DRV8353RS-EVM 的引脚 J1-17 和 J1-19 相对应的公头适配器引脚。这可以通过将这些引脚弯曲 90 度来实现。
  - 请勿将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 连接到 DRV8353RS-EVM 的 J3-29 和 J3-30 连接，如图 1-1 所示。
  - 使用跳线将 LAUNCHXL-F280025C 和 LAUNCHXL-F280039C 的 J2-19 连接到 J2-12，或者将 LAUNCHXL-F2800137 的 J2-13 连接到 J2-12，以使 C2000 的 SPI-STE 正确连接到 DRVx 器件，如图 1-1 所示。

  - 如果使用 DAC128S085EVM 电路板，请将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J5-42 弯曲 90 度，以避免将该引脚连接到 DAC128S085EVM 电路板。此 EVM 板仅用于调试目的，可监控各种软件变量。
- 按照表 3-2 中所述将电机、编码器和霍尔传感器连接到 DRV8353RS-EVM 和 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137，如图 1-1 中所示。
- 将由电池或直流电压源提供的 9V 至 54V 电源电压连接到电压电源引脚。只有在节 3.5 中指示时才应接通电源，否则请保持断开连接。

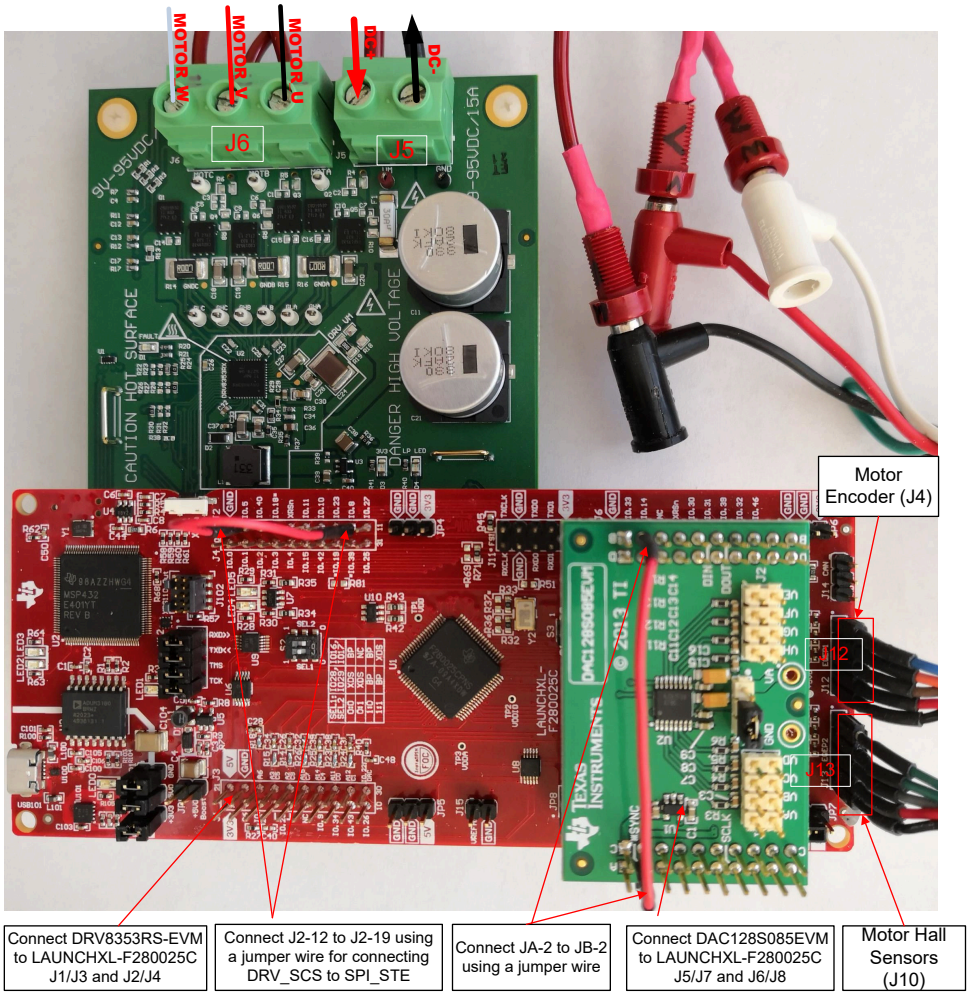


图 3-13. LAUNCHXL-F280025C/F280039C/F2800137 连接到 DRV8353RS-EVM 和 DAC128S085EVM

3.2.12 BOOSTXL-3PHGANINV 设置

BOOSTXL-3PHGANINV 评估模块采用 48V/10A 三相 GaN 逆变器，配备基于分流器的精密直列式相电流检测，从而对精密驱动器（例如，伺服驱动器）进行精准控制。该模块还具有独立的直流母线和三相电压检测功能，因此采用 C2000 LaunchPad™ 的 BLDC/PMSM 控制板专为无传感器 InstaSPIN-FOC 算法而设计。

- TI.com 内的 BOOSTXL-3PHGANINV 页面上提供了硬件文件和更多详细信息。
- 有关 BOOSTXL-3PHGANINV 的更多详细信息，请参阅相应的 *DRV8353Rx-EVM 用户指南*。
- 确保按照所述完成以下各项，然后将 BOOSTXL-3PHGANINV 连接到 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J1/J3 和 J4/J2，如图 1-1 所示。
  - 如果使用 DAC128S085EVM 电路板，请将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J5-42 弯曲 90 度，以避免将该引脚连接到 DAC128S085EVM 电路板。此 EVM 板仅用于调试目的，可监控各种软件变量。
- 按照表 3-2 中所述将电机、编码器和霍尔传感器连接到 BOOSTXL-3PHGANINV 和 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137，如图 1-1 中所示。
- 将由电池或直流电压源提供的 12V 至 54V 电源电压连接到电压电源引脚。按照节 3.5 中的操作说明打开电源。

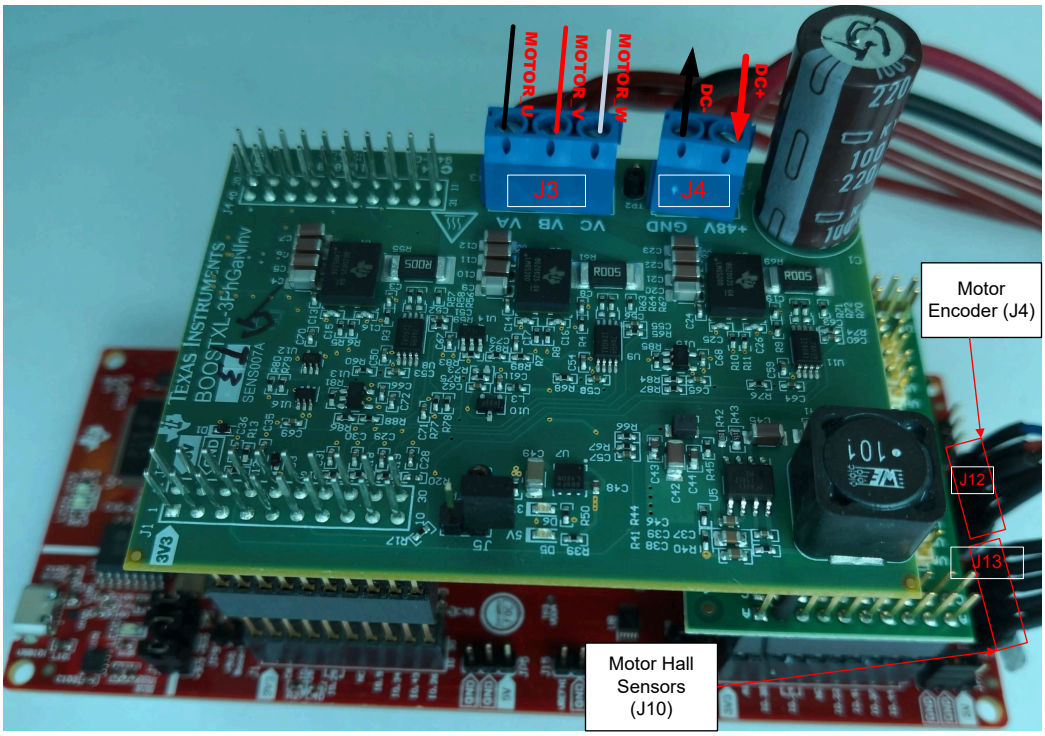


图 3-14. LAUNCHXL-F280025C/F280039C/F2800137 连接到 BOOSTXL-3PHGANINV 和 DAC128S085EVM

3.2.13 DRV8316REVM 设置

DRV8316REVM 提供三个半 H 桥集成式 MOSFET 驱动器，用于驱动具有 8A 峰值电流驱动的三相无刷直流 (BLDC) 电机。DRV8316 器件在模块上集成了用于三相低侧电流测量的电流检测放大器 (CSA) 以及独立直流母线和三相电压检测，因此这个采用 C2000 LaunchPad™ 开发套件的 BLDC/PMSM 控制板适合用于无传感器 InstaSPIN-FOC 算法

- *DRV8316REVM 硬件设计文件*位于 TI.com 内的 DRV8316REVM 页面上。
- 如需详细了解 DRV8316REVM，请参阅 *DRV8316REVM 评估模块*。
- 确保按照所述完成以下各项，然后将 DRV8316REVM 连接到 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 的 J1/J3 和 J4/J2 站点，如图 1-1 所示。
  - 如果使用 DAC128S085EVM 电路板，请将 LAUNCHXL-F280025C 的 J5-42 弯曲 90 度，以避免将该引脚连接到 DAC128S085EVM 电路板。此 EVM 板仅用于调试目的，可监控各种软件变量。
- 按照表 3-2 中所述将电机、编码器和霍尔传感器连接到 DRV8316REVM 和 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137，如图 1-1 中所示。
- 由电池或直流电压源提供的 4.5V 至 24V 电源电压连接到电压电源引脚，如图 1-1 所示。DRV8316REVM 的 ABS 最大电源电压额定值为 40V，因此必须注意确保运行期间电压不超过此值。这在减速或制动电机时尤为重要，因为这会导致电源电压大幅上升。只有在节 3.5 中指示时才应接通电源，否则请保持断开连接。

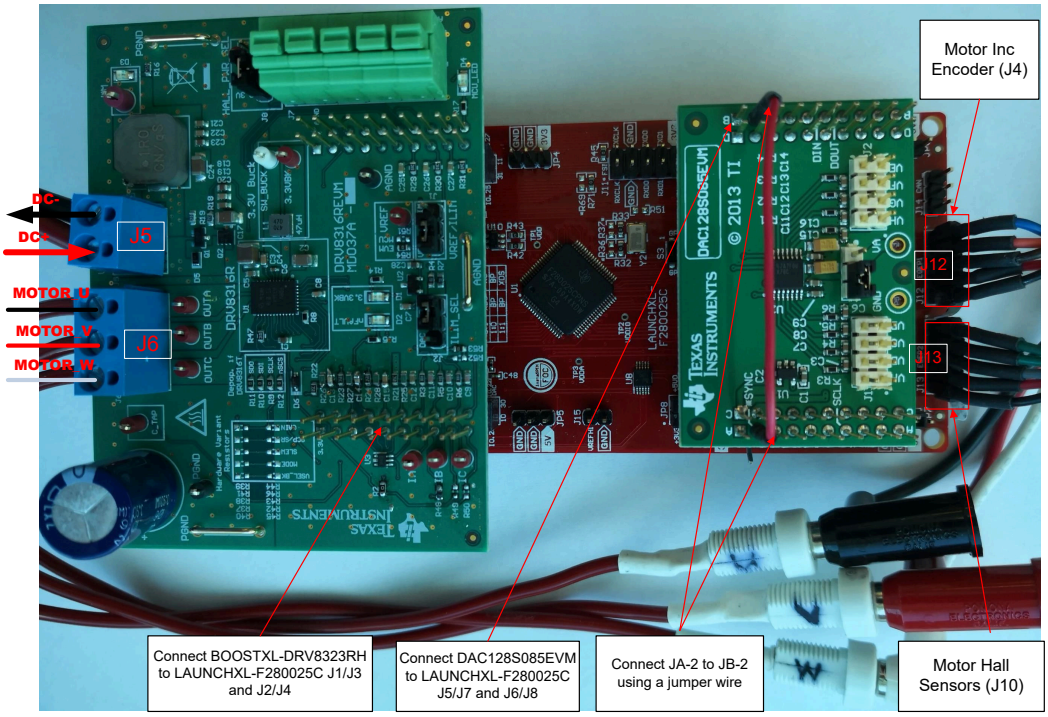


图 3-15. LAUNCHXL-F280025C 已连接至 DRV8316 REVM 和 DAC128S085EVM

3.2.14 TMDSHVMTRINSPIN 设置

警告

- 此评估模块 (EVM) 仅限在实验室环境中工作，并且 TI 不将其视为适合常规消费品用途的最终产品成品。
- 这个 EVM 只能由具有资质的工程师和技术人员使用，这些专业人员熟悉与处理高压电气和机械部件、系统和子系统相关的风险。
- 此 EVM 中存在电压和电流时，如果处理不当，会引发电击、火灾和/或人身伤害。使用设备时必须小心并采取适当的防护措施以避免人身伤害或财产损失。
- 由于存在高电压，在使用 EVM 电子元件时始终要小心！主电源断开后，直流母线电容器会长时间保持充电状态。
- 该 EVM 可接受来自交流电源/壁式电源的电源，仅使用来自壁式电源的火线和中性线，保护性接地未连接（悬空）。电源接地相对于保护性接地端浮动，所有接地平面都是相同的。因此，在将示波器和其他测试设备连接到电路板之前，必须小心谨慎并满足适当的隔离要求。在将接地设备连接到 EVM 时，必须使用隔离变压器。
- 主板上的功率级具有单独的额定值。用户有责任在将这些电源块连接在一起并为主板供电前，确保已完全理解这些额定值（电压、电流和功率等级）并遵守这些额定值。通电后，不得触碰 EVM 以及与 EVM 相连的组件。

TMDSHVMTRINSPIN 是一款基于 DIMM100 controlCARD 的主板评估模块，展示了最常见类型高电压三相电机控制，这些电机包括交流感应 (ACI) 电机、无刷直流 (BLDC) 电机和永磁同步电机 (PMSM)。高电压电机控制套件具有独立的直流母线和三相电压检测功能，因此采用 C2000 LaunchPad™ 开发套件的 BLDC/PMSM 控制板适合与无传感器 InstaSPIN-FOC 算法配合使用。

- 硬件文件位于 C2000WARE-MOTORCONTROL-SDK 的 `<install_location>\solutions\tmdshvmtrinspin\hardware` 文件夹中。

本节介绍了使用通过 MotorControl SDK 提供的软件运行 TMDSHVMTRINSPIN 所需的步骤。该套件随附跳线和开关设置、位置正确，可用于连接 controlCARD。确保这些设置在电路板上是有效的，如下所述，然后将带有 TMDSADAP180TO100 适配器的 controlCARD 插入到 TMDSHVMTRINSPIN 电路板中，如图 1-1 所示。

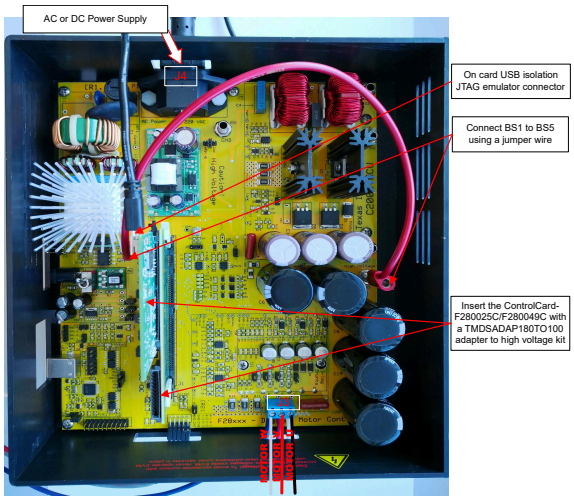


图 3-16. TMDSHVMTRINSPIN 通过 TMDSADAP180TO100 连接到 TMDSCNCD280025C、TMDSCNCD280039C 或 TMDSCNCD2800137

小心

首先验证这些设置，然后再对电路板施加交流电！

- 确保没有任何设备连接至电路板，并且未对电路板施加任何电源。

- 将带 TMDSADAP180TO100 适配器的控制卡插入 [Main]-J1 controlCARD 连接器（如果尚未组装）。
- 确保以下跳线和连接器设置正确实现，如图 1-1 所示。
  - [Main]-J3、J4、J5 和 J8 已组装。
  - 未组装 [Main]-J9 和 [M3]-J5 以便使用具有板载仿真的 controlCARD 来禁用 HVKIT 上的 XDS100。
  - [MAIN]-J7 安装在引脚 2-3（距离 DIMM 100 插槽最远的引脚）之间。
  - 安装 [Main]-BS1 和 [Main]-BS5 之间的香蕉电缆以绕过 PFC。
  - 在 > 150W 的负载下操作电机时，确保套件附带的直流风扇连接到直流风扇跳线 [Main]-J17。
- 获得直流母线电源的两个选项如下，建议使用外部 15V 直流电源。
  - 如果使用来自外部 15V 直流电源的 +15V 电压，则不会组装 [Main]-J2。确认 [M6]-SW1 处于“Off”位置，将 15V 直流电源连接到 [M6]-JP1。
  - 如果使用来自辅助电源模块的 +15V 电源，则 [Main]-J2 在电桥和中间引脚之间组装了一个跳线。
- 打开 [M6]-SW1。现在，[M6]-LD1 应该接通。请注意，控制卡 LED 也会点亮，这表示控制卡正在由电路板供电。
- 使用香蕉插头线将 [Main]-BS1 和 BS5 相互连接，然后将交流电源或直流电源的一端连接到 [Main]-P1。
- 将电机、编码器和霍尔传感器连接到表 3-2 中所述的套件，如图 1-1 中所示。
- 将电源电压从交流或直流电压源连接到电源引脚。只有在节 3.5 中指示时才应接通电源，否则请保持断开连接。

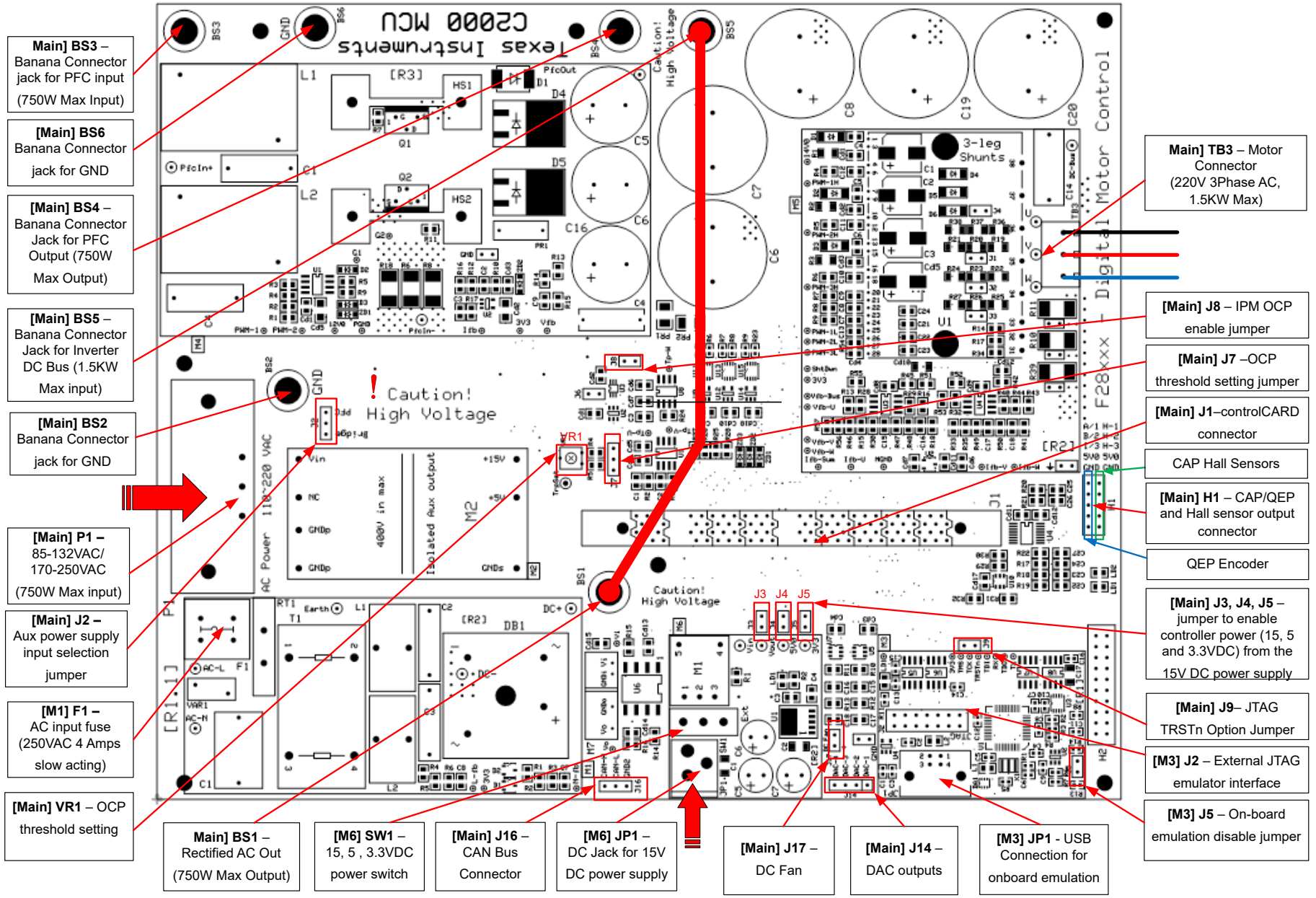


图 3-17. TMDSHVMTRINSPIN 套件跳线和连接器图

表 3-3 显示了电路板上可用的各种连接。电路板上这些连接的位置如图 1-1 所示。

表 3-3. 关键跳线、连接器说明

[Main]-P1	交流输入连接器 (110V – 220V AC)
[Main]-TB3	用于连接电机的端子块
[Main]-BS1	用于从交流整流器输出的香蕉插孔
[Main]-BS2，BS6	针对接地 (GND) 连接的香蕉插孔
[Main]-BS3	用于为 PFC 级连接输入电压的香蕉插孔，这通常是通过 [Main]-BS1 连接器对交流电压进行整流。
[Main]-BS4	香蕉插孔用于将负载连接到 PFC 级的输出，当使用 PFC+电机时，PFC 级的输出将连接到逆变器总线的输入，即 [Main]-BS5
[Main]-BS5	用于为逆变器提供直流母线电压输入的香蕉插孔
[Main]-J2	辅助电源模块输入电压选择跳线， • 当跳线连接到桥位置时，辅助电源模块从交流整流器电桥输出提供电源。 • 当跳线连接到 PFC 位置时，辅助电源模块从 PFC 级的输出端提供电源。
[Main]-J3，J4，J5	跳线 J3、J4 和 J5 分别用于为电路板从 15V 直流电源提供 15V、5V 和 3.3V 电源。
[Main]-J7	J7 用于选择过流保护阈值源
[Main]-J8	J8 用于启用/禁用 IPM 过流保护
[Main]-J9	JTAG TRSTn 将跳线断开，组装跳线将启用 JTAG 到微控制器的连接。当不需要 JTAG 连接时，例如从闪存引导时，需要拆下跳线。
[Main]-J14	PWMDAC 输出：提供了由一阶低通滤波器所连接的 PWM 引起的电压输出。引脚 1、2、3 和 4 分别被连接至低通滤波的 PWM 输出引脚以在示波器上观察系统变量。
[Main]-J16	隔离式 CAN 总线连接器
[Main]-J17	为连接到 IPM 散热器的直流风扇（随电路板提供）供电的连接器。
[Main]-H1	QEP 连接器：与 0-5V QEP 传感器连接，以收集有关电机速度和位置的信息。 电容/霍尔效应传感器连接器：与 0-5V 传感器连接，以收集有关电机速度和位置的信息。
[M1]-F1	交流输入保险丝
[M3]-JP1	针对板载仿真的 USB 连接
[M3]-J2	外部 JTAG 接口：这个连接器提供到 JTAG 仿真引脚的访问权。如果需要外部仿真，将一个跳线放置在 [M3]-J5 并将仿真器连接到主板。若要为仿真逻辑供电，USB 连接器仍需要连接到 [M3]-JP1。
[M3]-J5	板载仿真禁用跳线：在此处放置一个跳线来禁用板载仿真器并提供到外部接口的访问。

3.3 实验软件实现

1. 从 Code Composer Studio (CCS) 集成开发环境 (IDE) 工具文件夹下载 Code Composer Studio 并进行安装。建议使用版本 10.4 或更高版本。有关 CCS 安装和实现的更多详细信息，请参阅 CCS 用户指南。
2. 通过 TI 提供的链接下载并安装 C2000WARE-MOTORCONTROL-SDK 软件包，并在其默认文件夹中安装该 Motor Control SDK 软件。可以通过两种方法之一安装 C2000WARE-MOTORCONTROL-SDK：
   a. 通过 C2000Ware MotorControl SDK 工具文件夹下载软件。
   b. 转至 CCS 的“View”→“Resource Explorer”下。在 TI Resource Explorer 下，转至“Software”→“C2000Ware_MotorControl_SDK”，然后点击安装按钮。
3. 安装完成后，关闭 CCS 并创建一个新的工作区以导入工程。本示例实验的软件工程位于 C2000Ware 电机控制 SDK 文件夹内，该文件夹位于 `<install_location>\C2000Ware_MotorControl_SDK_<version>\solutions\universal_motorcontrol_lab`。按照以下各节使用不同的增量构建来构建和运行该代码，如以下部分所述。

3.3.1 导入和配置工程

示例实验是一个通用工程，支持可与 F280025C、F280039C 或 F2800137 C2000 MCU 器件配合使用的各种 TI EVM 电机驱动器套件。用户可以通过设置实验工程的构建配置和属性来运行不同的 TI EVM 套件。在以下各节中，将 LAUNCHXL-F280025C、LAUNCHXL-F280039C 或 LAUNCHXL-F2800137 与 BOOSTXL-DRV8323RS 实验结合使用，以展示如何导入和运行此套件上的示例实验。

1. 依次点击“Project”➔“Import CCS Projects...”，在 CCS 中导入工程，然后点击“Browse...”按钮选择搜索目录：
   a. 基于 F28002x 的实验：`<install_location>\solutions\universal_motorcontrol_lab\f28002x\ccs\motor_control\`，从而选择“universal_motorcontrol_lab_f28002x”工程。
   b. 基于 F28003x 的实验：`<install_location>\solutions\universal_motorcontrol_lab\f28003x\ccs\motor_control\`，从而选择“universal_motorcontrol_lab_f28003x”工程。
   c. 基于 F280013x 的实验：`<install_location>\solutions\universal_motorcontrol_lab\f280013x\ccs\motor_control\`，从而选择“universal_motorcontrol_lab_f280013x”工程。
2. 该实验工程可配置为在各种电机驱动器套件上运行。通过右键点击导入的工程名称并选择正确的构建配置（例如 **Flash_lib_DRV8323RS_3SC**），可以选择其中一个套件，如图 1-1 所示。
3. 通过右键点击导入的工程名称来配置工程以选择工程中的支持函数，然后点击“Properties”命令为工程设置预定义符号，如图 1-1 所示。
   a. 通过在名称中删除或添加“_N”，可以激活或禁用预定义符号。例如，将“MOTOR1_FWC_N”中的“_N”删除（使其变为“MOTOR1_FWC”）可启用弱磁控制，而将“MOTOR1_FWC”符号名称更改为“MOTOR1_FWC_N”可为电机 1（压缩机）禁用弱磁控制功能。
   b. 根据电机和硬件板，通过启用上述的相关预定义符号来选择正确的支持电机控制算法。表 3-4 展示了支持算法和相关电机矩阵。
   c. 通过启用预定义符号来选择正确的支持函数，如图 1-1 所示。
4. 选择正确的目标配置文件 (.ccxml)（如图 1-1 所示），方法是右键点击文件名，在弹出菜单中选择“Set as Active Target Configuration”和“Set as Default Target Configuration”。
   a. TMS320F280025C_LaunchPad.ccxml 适用于基于 LAUNCHXL-F280025C 的硬件套件。
   b. TMS320F280025C.ccxml 适用于基于 TMDSCNCD280025C 的硬件套件。

5. 在 *user_mtr1.h* 和 *user_common.h* 文件中选择或定义正确的电机模型。这些文件位于工程浏览器窗口中的 src_board 文件夹下。电机定义了 *user_mtr1.h* 文件中从第 921 行开始的部分。取消注释与被测试电机相对应的 #define，并确保其余 #define 电机仍保持注释状态。确保代码中的电机参数与所连接电机的规格相匹配。
6. 按照节 3.2 中所述设置硬件套件，将电机、编码器和/或霍尔传感器连接到套件。

表 3-4. 示例实验中的支持算法、功能和电机矩阵

算法或功能	预定义符号	LaunchPad						controlCARD
		DRV8329AEVM	BOOSTXL-DRV8323RH	BOOSTXL-DRV8323RS	DRV8353RS-EVM	BOOSTXL-3PHGANINV	DRV8316REVM	TMDSHVMTRINSPIN
基于 FAST 的无传感器 FOC	MOTOR1_FAST	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，HVPMSMMTR
基于 eSMO 的无传感器 FOC	MOTOR1_ESMO	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，HVPMSMMTR
基于 QEP 编码器的含传感器 FOC	MOTOR1_ENC QEP_ENABLE	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，LVSERVOMTR	✓，HVPMSMMTR
基于霍尔传感器的含传感器 FOC	MOTOR1_HALL HALL_ENABLE HALL_CAL	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR，LVBLDCMTR	✓，LVSERVOMTR，LVBLDCMTR	✓，HVBLDCMTR#
梯形 InstaSPIN-BLDC	MOTOR1_ISBLDC	✓，LVSERVOMTR，LVBLDCMTR	✖	✖	✖	✖	✖	✖
单分流器电流检测	MOTOR1_DCLINKSS	✓，LVSERVOMTR，LVBLDCMTR	✖	✖	✖	✖	✖	✖
使用图形工具时的数据日志 [1]	DATALOGF2_EN	✓	✓	✓	✓	✓	✓	✓
PWMDAC	EPWMDAC_MODE	✖	✖	✖	✖	✖	✖	✓
外部 DAC	DAC128S_ENABLE	✓	✓	✓	✓	✓	✓	✖
SFRA 工具	SFRA_ENABLE	✓	✓	✓	✓	✓	✓	✓
使用图形工具时的阶跃响应	STEP_RP_EN	✓	✓	✓	✓	✓	✓	✓

[1] 通用电机控制实验中的 Datalog 实现需要 DMA。并非所有 C2000 器件都具有 DMA。请参阅特定器件数据表以确定 DMA 支持。

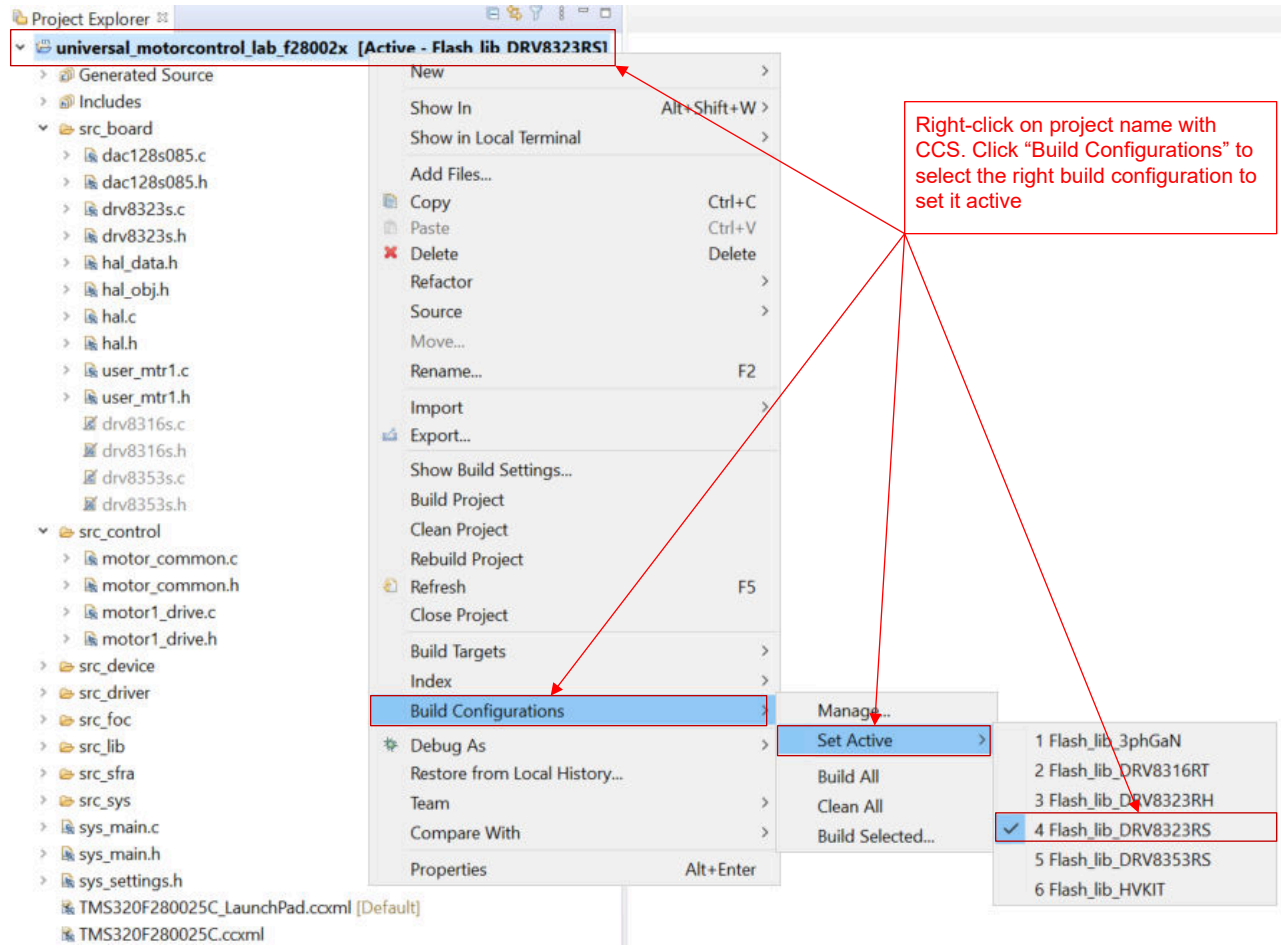


图 3-18. 在 CCS 中选择合适的构建配置

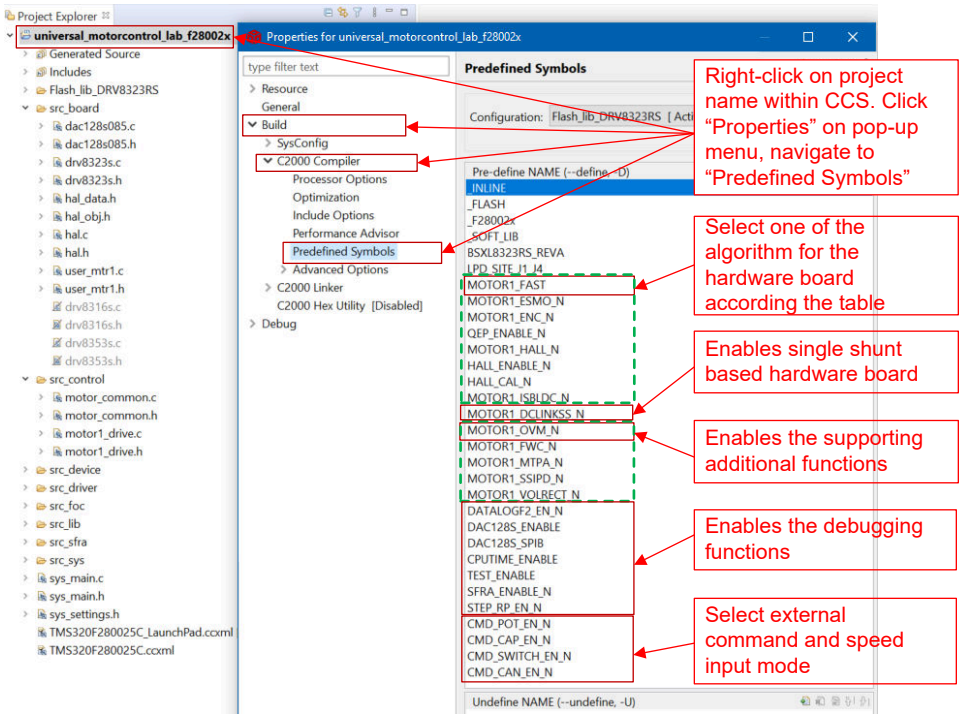


图 3-19. 在工程属性中选择所需的预定义符号

3.3.2 实验工程结构

工程总体结构如图 1-1 所示。器件外设配置基于 C2000Ware driverlib。如果用户需要将参考设计软件迁移到定制板或不同 C2000 器件上，则只需更改 *hal.c* 和 *hal.h* 中的代码和定义以及 *user_mtr1.h* 中的参数。

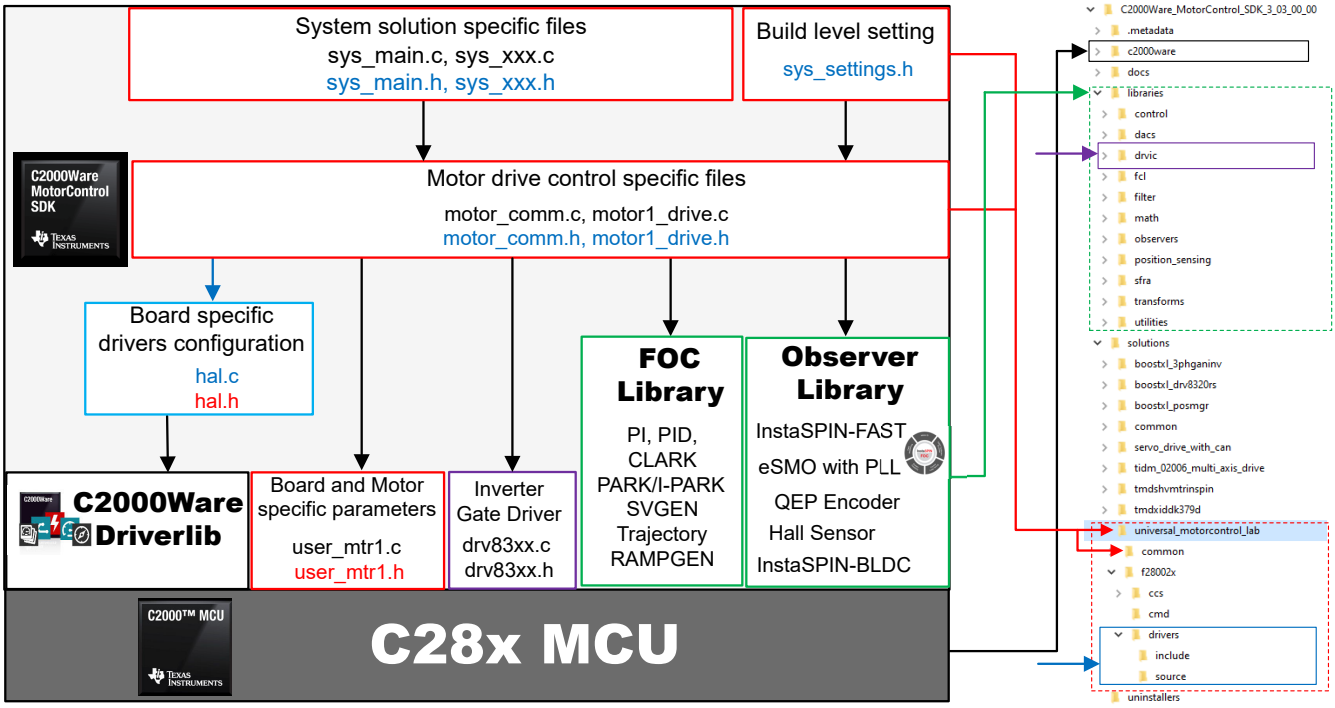


图 3-20. 工程结构概览

将工程导入 CCS 后，CCS 内将显示工程浏览器，如图 1-1 所示。

文件夹 *src_foc* 包含典型的 FOC 模块，其中包括 Park、Clark 以及逆向 Park 和 Clark 变换、PID 和由电机驱动算法组成的估算器，独立于特定器件和电路板。

src_lib 文件夹包含 InstaSPIN-FOC 库和并非特定于任何特定器件或电路板的数学库。

src_control 文件夹包含电机驱动控制文件，这些文件在中断服务例程和后台任务中调用电机控制核心算法函数。

文件夹 *src_sys* 包含为系统控制保留的一些文件，这些文件独立于特定的器件或电路板。用户可以添加自己的用于系统控制、通信等功能的代码。

特定于电路板、特定于电机和特定于器件的文件位于 *src_board* 文件夹中。这些文件包含特定于器件的驱动程序，用于运行解决方案。如果用户希望为自己的电路板迁移工程或迁移到其他 C2000 器件，则只需根据电路板的器件外设使用情况更改 *hal.c*、*hal.h* 和 *user_mtr1.h* 文件。

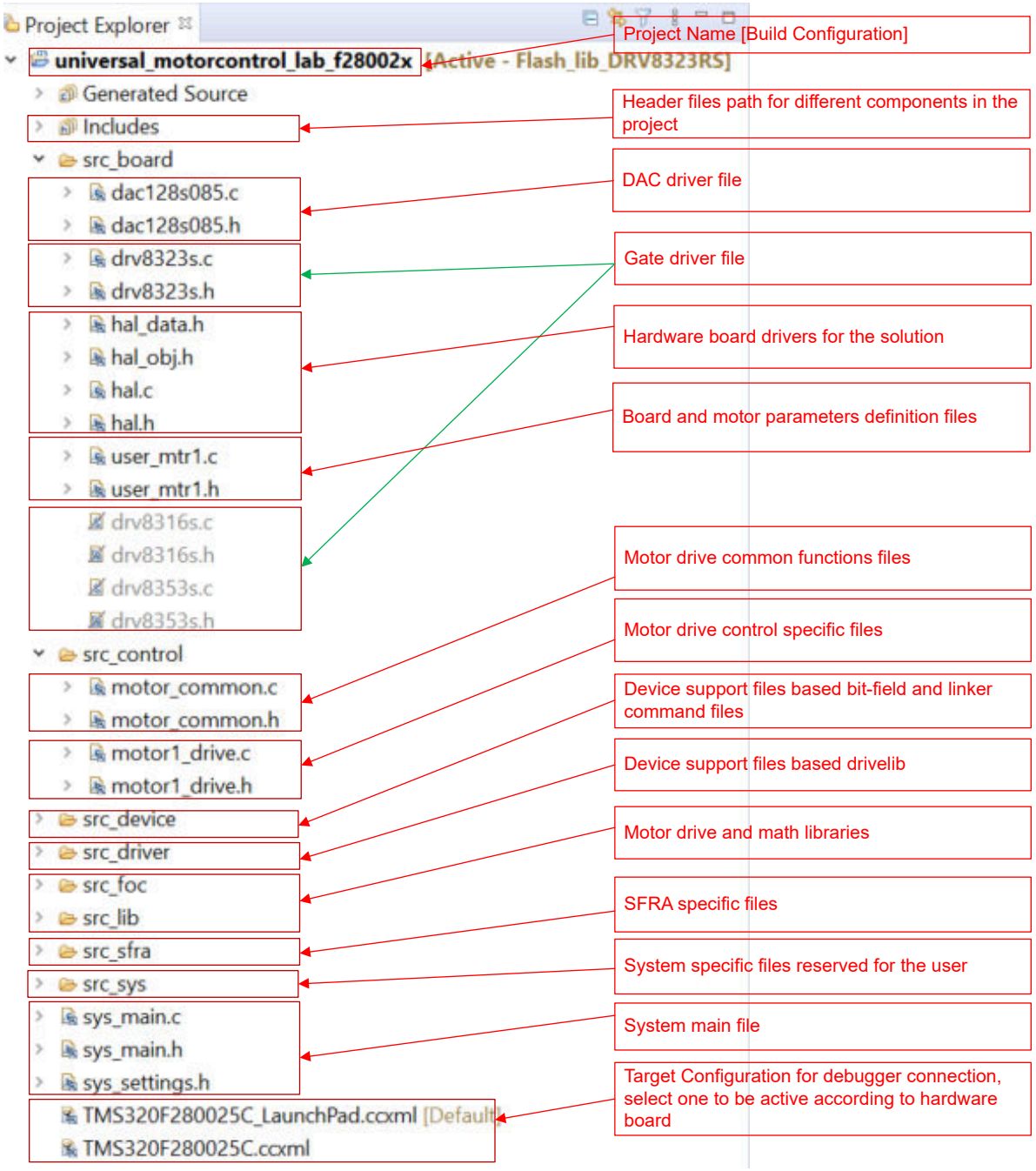


图 3-21. 示例实验的 Project Explorer 视图

3.3.3 实验软件概述

图 1-1 展示了固件的工程软件流程图，其中包括一个用于实时电机控制的 ISR、一个主循环用于在后台循环中更新电机控制参数。ISR 将由 ADC 转换结束 (EOC) 触发。

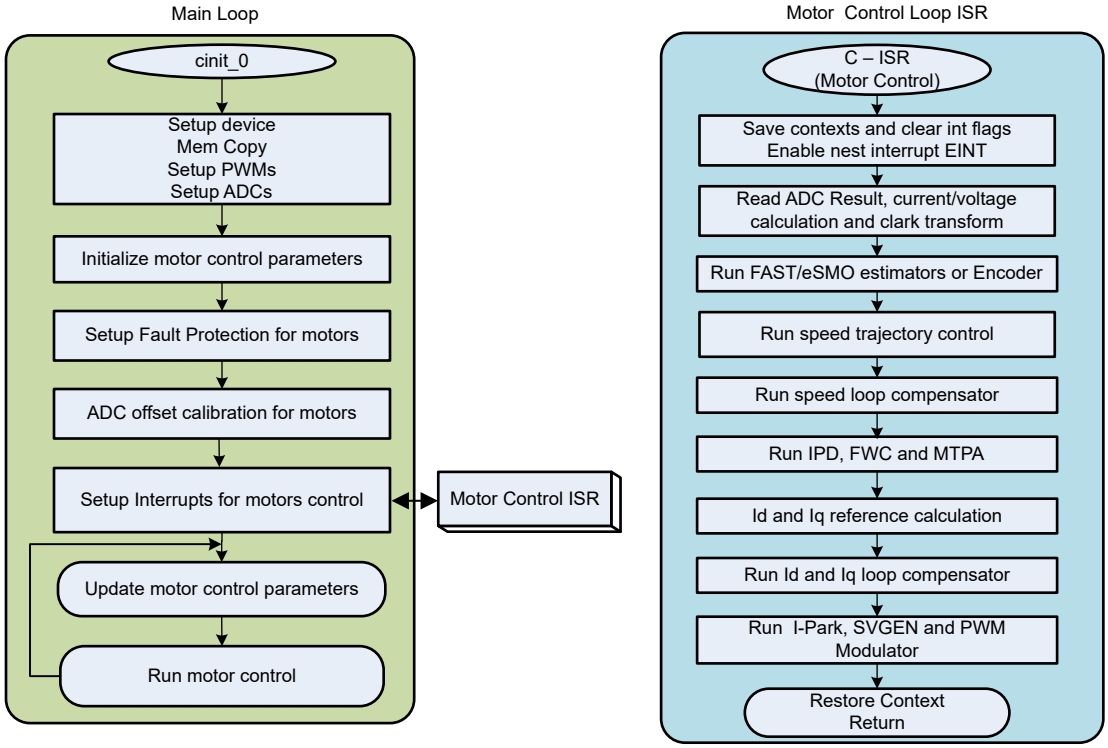


图 3-22. 工程软件流程图

为了简化系统开发和设计，将该软件组织为四个增量构建，这使得学习和熟悉电路板和软件变得更加容易。这个方法对也适用于调试和测试电路板。

表 3-5 列出了要在该实验工程中使用的框架模块。

表 3-5. 在实验工程中使用电机控制模块

模块名称	说明	算法
ANGLE_GEN_run	用于开环运行的斜坡角度发生器	eSMO、ENC、HALL
CLARKE_run	针对电流或电压的 Clarke 变换	FAST、eSMO、ENC、HALL
collectRMSData、calculateRMSData	收集采样值以计算相电流和电压的 RMS 值	FAST、eSMO、ENC、HALL
DAC128S_writeData	使用 SPI 将软件变量转换并发送到外部 DAC	所有算法
DATALOGIF_update	将实时值存储到中，以便使用图形工具显示	所有算法
ENC_run	根据编码器计算转子角度	ENC
ESMO_run	增强滑模观测器 (eSMO) 以实现无传感器 FOC	eSMO
EST_run	适用于无传感器 FOC 的 FAST 估算器	FAST
EST_runTraj	用于电机识别的电流和速度轨迹发生器	FAST
EST_setupTrajState	用于电机识别的电流和速度轨迹发生器设置	FAST
HAL_readADCData	以浮点格式返回 ADC 转换值	所有算法
HAL_writePWMDACData	将软件变量转换为 PWM 信号	所有算法
HAL_writePWMData	用于电机的 PWM 驱动	所有算法
HALL_run	根据霍尔传感器计算转子角度和速度	HALL
IPARK_run	Park 逆变换	FAST、eSMO、ENC、HALL
PARK_run	Park 变换	FAST、eSMO、ENC、HALL
PI_run	针对电流和速度的 PI 稳压器	所有算法
SPDCALC_run	基于来自编码器信号的角度进行速度测量	ENC
SPDFR_run	基于来自观测器的角度测量速度	eSMO
SVGEN_run	具有正交控制的空间矢量 PWM	FAST、eSMO、ENC、HALL
TRAJ_run	用于设置速度基准的轨迹	所有算法
VS_FREQ_run	生成具有 v/f 曲线的矢量电压	FAST、eSMO、ENC、HALL

表 3-6 总结了在每个增量系统构建中测试的模块。

表 3-6. 每个增量构建使用的电机控制模块

软件模块	DMC_LEVEL_1	DMC_LEVEL_2	DMC_LEVEL_3	DMC_LEVEL_4	
	50% PWM 占空比，验证 ADC 失调电压校准、PWM 输出和相移	开环控制，用于验证电机电流和电压检测信号	闭合电流环路，用于验证电流 在电路板上检测和使用 PID 控制器进行电流控制	采用估算器/观测器的闭环运行	使用 FAST 估算器识别电机参数
HAL_readADCData	√ √	√ 1	√	√	√
HAL_writePWMData	√ √	√	√	√	√
ANGLE_GEN_run		√ √	√	√ (eSMO、ENC、HALL)*	
VS_FREQ_run		√ √			
CLARKE_run (电流)		√	√	√	√
CLARKE_run (电压)		√	√ (FAST)* 2	√ (FAST)*	√ (FAST)*
TRAJ_run		√ √	√	√ √	
EST_run		√ (FAST)*	√ (FAST)*	√ √ (FAST)*	√ √ (FAST)*
EST_setupTrajState					√ √ (FAST)*
EST_runTraj					√ √ (FAST)*
ESMO_run		√ (eSMO)*	√ (eSMO)*	√ √ (eSMO)*	
SPDFR_run		√ (eSMO)*	√ (eSMO)*	√ √ (eSMO)*	
ENC_run		√ (ENC)*	√ (ENC)*	√ √ (ENC)*	
SPDCALC_run		√ (ENC)*	√ (ENC)*	√ √ (ENC)*	
HALL_run		√ (HALL)*	√ (HALL)*	√ √ (HALL)*	
PARK_run		√	√	√	√
PI_run (Id)			√ √	√	√
PI_run (Iq)			√ √	√	√
PI_run (速度)				√ √	√
IPARK_run		√ √	√	√	√
SVGEN_run		√ √	√	√	√
HAL_writePWMDACData		√** 3	√**	√**	√**
DATALOGIF_update		√	√	√	√
DAC128S_writeData		√**	√**	√**	√**

1. √ 表示使用此模块。√ √ 表示此模块正在测试中。
2. √ (FAST)* 表示此模块仅供 FAST 使用。√ (eSMO)* 表示此模块仅供 eSMO 使用。√ (ENC)* 表示此模块仅供 ENC 使用。√ (HALL)* 表示此模块仅供 HALL 使用。
3. √** 表示此模块受某些硬件套件的支持，如表 3-1 所示。

通用实验工程可以单独使用其中一种 FOC 算法进行电机控制，或同时使用两种 FOC 算法，如表 3-7 所示。如果在实验工程中实现了两种算法，则可以快速平滑地切换正在使用的估算器。

表 3-7. 支持实验工程中的估算器算法

	FAST(MOTOR1_FAST)	eSMO (MOTOR1_ESMO)	ENCODER (MOTOR1_ENC)	HALL (MOTOR1_HALL)	ISBLDC (MOTOR1_ISBLDC)
FAST	√ 1	√	√	√	×
eSMO	√	√	√	×	×
编码器	√	√	√	×	×
HALL	√	×	×	√	×
ISBLDC	×	×	×	×	√

1. √ 意味着这两种算法可以在工程中同时使用。× 意味着这两种算法不能在工程中同时使用。

3.4 监控反馈或控制变量

可以通过多种方法来监控连续反馈或控制变量，例如带图形工具的数据日志、带示波器的 PWMDAC、带示波器的外部 DAC。

3.4.1 使用 DATALOG 函数

DATALOG 模块将两个用户可选软件变量的实时值存储在 C2000 MCU 上提供的数据 RAM 中，如图 1-1 所示。通过将模块输入 iptr[0] 和 iptr[1] 配置为两个变量的地址来选择这两个变量。两个 RAM 缓冲区位置（其中存储了数据值）的起始地址存储在 datalogBuff1[0] 和 datalogBuff1[1] 中。这些 Datalog 缓冲区是包含值触发数据的大型数组，这些数据随后可以显示在图形中。DATALOG 预分频器是可配置的，这使得 dlog 函数只能从每个预分频样本中记录一个。默认预分频器设置为 10，但可以通过修改 *datalogIF.h* 文件中 DATA_LOG_SCALE_FACTOR 定义的值进行更改。直接存储器存取 (DMA) 用于将所选软件变量的值传输到 RAM 中的 DATALOG 缓冲区。

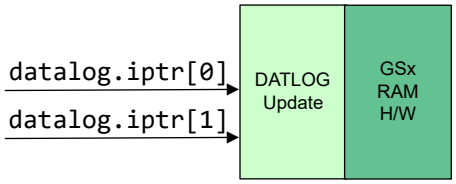


图 3-23. DATALOG 模块方框图

要启用 DATALOG 功能，必须在工程属性中添加预定义符号 DATALOGF2_EN，如图 1-1 所示。

以下代码演示了一个 DATALOG 对象和句柄的声明。此代码位于 *datalogIF.c* 文件中。

```
DATALOG_Obj datalog;
DATALOG_Handle datalogHandle;       //!< the handle for the Datalog object
```

下面的代码演示了 DATALOG 对象、句柄和参数的初始化和设置。此代码位于 *sys_main.c* 文件中。

```
// Initialize Datalog
datalogHandle = DATALOGIF_init(&datalog, sizeof(datalog));
DATALOG_Obj *datalogObj = (DATALOG_Obj *)datalogHandle;

HAL_setupDMAforDLOG(halHandle, 0, &datalogBuff1[0], &datalogBuff1[1]);
HAL_setupDMAforDLOG(halHandle, 1, &datalogBuff2[0], &datalogBuff2[1]);
```

下面的代码演示了两个模块输入 iptr[0] 和 iptr[1] 的配置，以指向两个变量的地址。datalog 模块输入指向不同的系统变量，具体取决于构建级别。此代码位于 *sys_main.c* 文件中：

```
datalogObj->iptr[0] = &motorVars_M1.adcData.I_A.value[0];
datalogObj->iptr[1] = &motorVars_M1.adcData.I_A.value[1];
```

以下代码演示了在 **motor1ctrlISR()** 中断执行期间使用新数据定期更新 datalog 缓冲区。此代码位于 *motor1_drive.c* 文件中。

```
if(DATALOGIF_enable(datalogHandle) == true)
{
    DATALOGIF_updateWithDMA(datalogHandle);

    // Force trig DMA channel to save the data
    HAL_trigDMAforDLOG(halHandle, 0);
    HAL_trigDMAforDLOG(halHandle, 1);
}
```

备注

如果没有足够的 RAM，数据日志将不会用在器件上的软件中。

datalog 模块与图形工具一同使用，该工具提供了一种直观检查变量并判断系统性能的方法。CCS 中提供了图形工具，可以各种图形类型显示数据数组。数据数组以各种格式存储在器件的存储器中。

当工程处于调试模式时，打开并设置时间图窗口来绘制数据日志缓冲区，如图 1-1 中所示。或者，用户可以导入位于工程文件夹中的图形配置文件。要导入这些文件，请点击：Tools -> Graph -> DualTime…，选择“import”并通过浏览找到以下位置 `<install_location>\solutions\universal_motorcontrol_lab\common\debug`，然后选择 *motor_datalog_fp2.graphProp* 文件。点击“OK”，这样会将 Graphs 添加到调试视图中。点击图形选项卡左上角的“Continuous Refresh”按钮  。

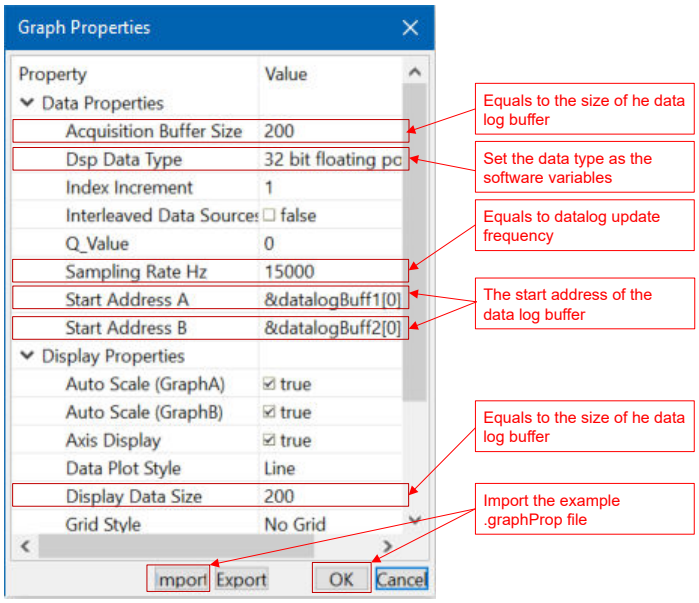


图 3-24. 图形窗口设置

3.4.2 使用 PWMDAC 函数

PWMDAC 模块使用 ePWM 6A、6B、7A 和 7B 将软件变量转换为 PWM 信号，如图 1-1 所示。PWMDAC 模块只受高压套件 (TMDSHVMTRINSPIN) 支持，因为该模块具有额外的 PWM 输出，电路板上提供 RC 滤波器。如果将 PWMDAC 模块与不支持 PWMDAC 模块的电机驱动器板一起使用，那么 PWM 信号被路由到 C2000 LaunchPad 上的备用 PWM，并且用户需要在这些引脚上添加 RC 滤波器才能利用 PWMDAC 解决方案。

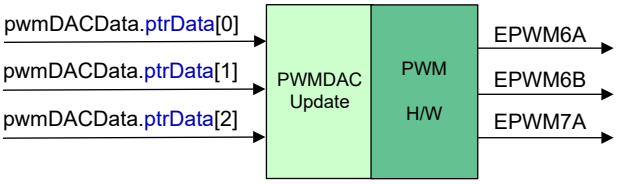


图 3-25. PWMDAC 模块方框图

PWMDAC 模块可被用于通过外部低通滤波器来查看相关引脚输出上的信号，此信号由变量表示。因此，需要使用外部低通滤波器来查看图 1-1 中所示的实际信号波形。（1 阶）RC 低通滤波器用于滤除嵌入在实际低频信号中的高频分量。要选择 R 和 C 值，时间常数可以用截止频率 (f_c) 表示，如以下公式方程式 49 和 方程式 50 所示。

$$\tau = RC = \frac{1}{2\pi f_c} \tag{49}$$

$$f_c = 2\pi RC \tag{50}$$

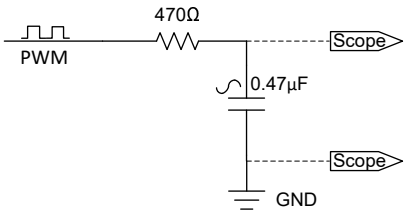


图 3-26. 连接到 C2000 MCU PWM 引脚的外部 RC 低通滤波器

若要启用 ePWM DAC 功能，必须在工程属性中添加预定义符号 EPWMDAC_MODE，如图 1-1 所示。

下面的代码演示了 PWMDAC 对象的声明。此代码位于 *sys_main.c* 文件中。

```
HAL_PWMDACData_t pwmDACData;
```

下面的代码展示了 PWMDAC 对象、句柄和参数的初始化和设置。四个模块输入 ptrData[0]、ptrData[1]、ptrData[2] 和 ptrData[3] 被配置为指向四个变量的地址。PWMDAC 模块输入指向不同的系统变量，这取决于构建级别。此代码位于 *sys_main.c* 文件中。

```
// set DAC parameters
pwmDACData.periodMax =
              PWMDAC_getPeriod(halHandle->pwmDACHandle[PWMDAC_NUMBER_1]);

pwmDACData.ptrData[0] = &motorVars_M1.anglePLL_rad;           // PWMDAC1
... ...
pwmDACData.ptrData[1] = &motorVars_M1.angleENC_rad;           // PWMDAC2
... ...
pwmDACData.ptrData[2] = &motorVars_M1.angleENC_rad;           // PWMDAC3
... ...
pwmDACData.ptrData[3] = &motorVars_M1.adcData.I_A.value[0];   // PWMDAC4

pwmDACData.offset[0] = 0.5f;
pwmDACData.offset[1] = 0.5f;
pwmDACData.offset[2] = 0.5f;
pwmDACData.offset[3] = 0.5f;

pwmDACData.gain[0] = 1.0f / MATH_TWO_PI;
pwmDACData.gain[1] = 1.0f / MATH_TWO_PI;
pwmDACData.gain[2] = 1.0f / MATH_TWO_PI;
pwmDACData.gain[3] = 4096.0f / USER_MOTOR1_OVER_CURRENT_A;
```

以下代码演示了在 **motor1ctrlISR()** 中断执行期间使用新数据更新 PWM 输出。此代码位于 *motor1_drive.c* 文件中。

```
// connect inputs of the PWMDAC module. HAL_writePWMDACData(halHandle, &pwmDACData);
```

3.4.3 使用外部 DAC 板

DAC128S 模块可将最多 8 个软件变量转换为 12 位整数值，并通过 SPI 将数据传输至 DAC128S085EVM 上的数模转换器 (DAC)，如图 1-1 所示。

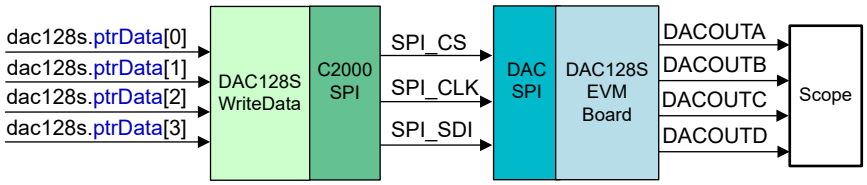


图 3-27. DAC128S 模块方框图

DAC128S085EVM 可以连接到 LaunchPad，如图 1-1 所示。图 1-1 展示了 DAC128S 的主要连接。

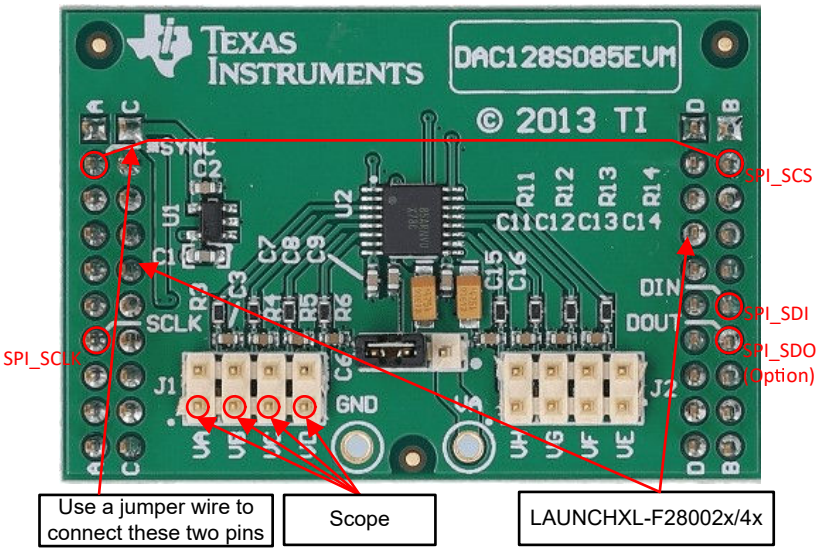


图 3-28. DAC128S085EVM 评估板

表 3-8. 使用 DAC128S085EVM 时需要进行硬件更改

LaunchPad 器件	所需硬件更改
F28002x、F28003x	跳线 C2000 SPI_STE (SCS) 引脚 JB-2 至 DAC128S085EVM 上的 SYNC 引脚 JA-2，如图 1-1 所示。
F280013x	(1) 器件只有 1 个 SPI 模块，该模块在 BoosterPack 站点 1 和站点 2 之间共享。这需要在 LaunchPad 上填充 0Ω 电阻器来将 SPI 信号连接到 DAC128S085EVM 连接到的 BoosterPack 站点 2。有关特定的电阻器标识符，请参阅 LAUNCHXL-F2800137 原理图中的 *SPI 路由* 部分。 (2) 跳线将 C2000 SPI_STE (SCS) 引脚接至 DAC128S085EVM 上的 SYNC 引脚 JA-2。C2000 SPI_STE 引脚使用情况将取决于所使用的逆变器 BoosterPack： 1. GPIO19 用于 DRV8323RS、DRV8353RS、DRV8316 和三相 GaN 逆变器 EVM。这将需要使用 F280013x 的内部振荡器并更改硬件。有关详细信息，请参阅 LAUNCHXL-F2800137 原理图的 *振荡器* 部分。 2. GPIO37 用于所有其他逆变器 EVM。

使用 DAC128S085EVM 时需要进行硬件更改。要启用 DAC128S 功能，必须在工程属性中添加预定义的符号 DAC_128S_ENABLE，如图 1-1 所示。

下面的代码演示了 DAC128S 对象的声明。此代码位于 *sys_main.c* 文件中。

```
DAC128S_Handle   dac128sHandle;        //!< the DAC128S interface handle
DAC128S_Obj      dac128s;              //!< the DAC128S interface object
```

DAC128S085 具有 8 通道 12 位数模转换器 (DAC)，因此用户可以通过更改 *dac128s085.h* 文件中 DAC_EN_CH_NUM define 的值将输出数设置为 1 到 8 之间的值。尽管 *sys_main.c* 文件将 8 个 ptrData[] 模块输入初始化为 8 个不同的变量地址，但在代码执行期间实际发送和使用的模块输入数量将由 *dac128s085.h* 文件中定义的 DAC_EN_CH_NUM 常量的值决定（如下所示）。大多数示波器只有四个探针，因此使用 8 个 DAC128S085 输出中的 4 个是本示例中的默认设置。使用 4 个输出是本示例实验中的默认设置，因为大多数示波器只有四个探针。更多的输出将占用更多的 ISR 时间来转换和传输数据，这可能会对用于其他任务的时间产生负面影响，如果用户希望使用不止 4 个输出，必须考虑这一因素。

```
#define DAC_EN_CH_NUM                    (4)          // 1~8
```

下面的代码演示了 DAC128S 对象、句柄和参数的初始化和设置。代码会配置八个模块输入 ptrData[0] - ptrData[7]，以指向八个不同软件变量的地址，但实际使用的模块输入数量由 *dac128s085.h* 文件中定义的 DAC_EN_CH_NUM 常量的值决定。dac128s 数据指向不同的系统变量，具体取决于构建级别和定义的控制算法。此代码可在 *sys_main.c* 文件中找到。

```
// initialize the DAC128S
dac128sHandle = DAC128S_init(&dac128s);

// setup SPI for DAC128S
DAC128S_setupSPI(dac128sHandle);
... ...
// Build_Level_2 or Level_3, verify the estimator
dac128s.ptrData[0] = &motorVars_M1.angleGen_rad;                // CH_A
dac128s.ptrData[1] = &motorVars_M1.angleEST_rad;                // CH_B
dac128s.ptrData[2] = &motorVars_M1.anglePLL_rad;                // CH_C
dac128s.ptrData[3] = &motorVars_M1.adcData.I_A.value[0];        // CH_D
dac128s.ptrData[4] = &motorVars_M1.adcData.V_V.value[0];        // CH_E, N/A
dac128s.ptrData[5] = &motorVars_M1.adcData.I_A.value[1];        // CH_F, N/A
dac128s.ptrData[6] = &motorVars_M1.adcData.I_A.value[2];        // CH_G, N/A
dac128s.ptrData[7] = &motorVars_M1.adcData.V_V.value[1];        // CH_H, N/A

dac128s.gain[0] = 4096.0f / MATH_TWO_PI;
dac128s.gain[1] = 4096.0f / MATH_TWO_PI;
dac128s.gain[2] = 4096.0f / MATH_TWO_PI;
dac128s.gain[3] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;
dac128s.gain[4] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V;
dac128s.gain[5] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;
dac128s.gain[6] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;
dac128s.gain[7] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V;
```

```
dac128s.offset[0] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[1] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[2] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[3] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[4] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[5] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[6] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[7] = (uint16_t)(0.5f * 4096.0f);
```

下面的代码演示了在 **motor1ctrlISR()** 中断执行期间通过 SPI 定期使用新数据更新 DAC128S 电路板。此代码位于 *motor1_drive.c* 文件中。实际更新的 DAC 输出数量将取决于 DAC_EN_CH_NUM 常数的值。

```
// Write the variables data value to DAC128S085
DAC128S_writeData(dac128sHandle);
```

3.5 使用不同的构建级别循序渐进地运行工程

分多个阶段对此系统逐步测试和验证，以便最终系统可以让人放心地运转。要选择特定的构建选项，请在 *sys_settings.h* 文件中将 DMC_BUILDLEVEL 定义的值更改为所需的 DMC_LEVEL_X 选项。选中构建选项后，右键点击工程名称并点击 *Rebuild Project* 编译工程。

3.5.1 级别 1 增量构建

此构建级别的目标：

- 使用 HAL 对象为电机驱动器硬件初始化 MCU 的外设。
- 验证 PWM 和 ADC 驱动器模块
- 确认 ADC 偏移验证
- 熟悉 CCS 的操作。有关 CCS 的更多详细信息，请参阅 CCS 用户指南。

在该构建级别中，电路板以开环模式执行（采用固定 PWM 占空比）。占空比设置为 50%。该构建级别验证来自功率级的反馈值检测以及 PWM 栅极驱动器的运行，并确保没有硬件问题。此外，可以在该构建级别中执行输入和输出电压检测校准。在此过程中，电机必须保持断开。图 1-1 显示了该构建级别的软件方框图。

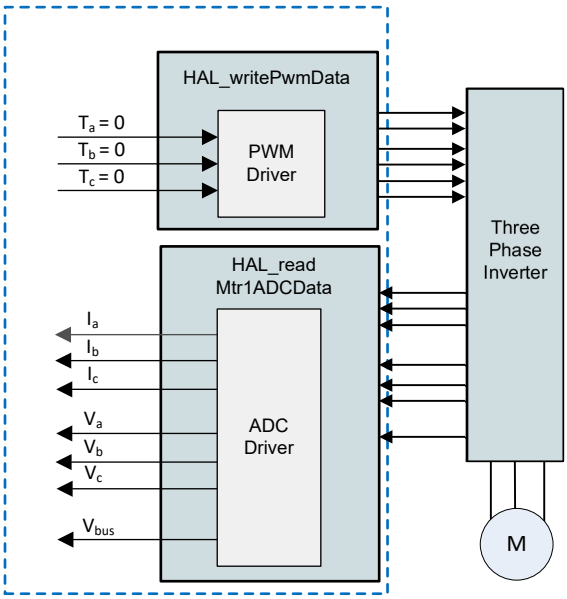


图 3-29. 构建级别 1 软件方框图 - 偏移验证

3.5.1.1 构建和加载工程

1. 按照节 3.2 中所述设置电机驱动器硬件板和 C2000 LaunchPad 或 controlCARD，但在此构建级别中，电机不应连接到电机驱动器板。注意：确保 LaunchPad 上的 S2 开关配置正确尤为重要，因为将驱动器的使能引脚连接到 LaunchPad 的 GPIO 29 是必需的。
2. 将 USB 电缆从计算机连接到 C2000 LaunchPad 或 controlCARD 上的板载 USB 连接器，以启用到 C2000 器件的隔离 JTAG 仿真。
3. 按照节 3.2 中所述，通过向总线电压输入端子施加适当的电压来为电机驱动器板供电。
4. 将通用电机控制实验工程导入 CCS 并按照节 3.3.1 中所述选择正确的构建配置。打开 *sys_settings.h* 文件，将 DMC_BUILDLEVEL 设置为 DMC_LEVEL_1。这将确保将工程配置为运行第一个增量构建。
5. 在 Project Explorer 窗口中，通过右键点击所需的目标配置文件名并选择 *Set as Active Target Configuration*，确保将正确的目标配置文件设置为 Active。建议也通过右键点击文件名并选择 *Set as Default Target Configuration* 来将所需目标配置文件设置为默认值。之所以这样做，一个原因是没有用于显示哪个文件处于活动状态的可见指示符，但如果将文件设置为默认值，则 [default] 指示符会出现在工程浏览器窗口中文件名的旁边。将文件设置为默认值也会导致默认情况下使用该文件，除非另一个配置文件专门设置为 Active。您也可以通过以下方法将目标配置链接到工作区中的工程：前往 *View* > *Target Configurations*，右键点击“Target Configurations”视图中的目标配置名称并选择 *Link to Project*。
6. 右键点击工程名称，然后点击 *Rebuild Project*。监视控制台窗口。工程中的任何错误都将显示在控制台窗口中。
7. 成功完成构建后，点击“Debug”按钮  ，或依次点击 *Run* → *Debug*。现在，IDE 将自动连接至目标，将输出文件载入到器件内并换到调试视图。右上角会显示“CCS Debug”图标，该图标表明用户现在处于“Debug Perspective”视图中。程序应该在 main() 的开始处暂停。

> **小心**
> 在将代码载入闪存时请勿点击 *Cancel*、关闭板的电源或断开仿真器。

3.5.1.2 设置调试环境窗口

在调试代码时观察局部和全局变量是标准调试做法。在 CCS 中有多种不同的方法来实现这一做法，例如存储器视图和监视视图。此外，CCS 能够制作时域（和频域）图。该功能允许用户使用图形工具查看波形。有关如何设置和配置图形工具的信息，请参见节 3.4.1。有关设置表达式窗口的信息，请参阅以下说明。

1. 设置监视窗口：点击菜单栏上的 *View* → *Expressions* 打开一个“Expressions”监视窗口。在“Expressions”窗口中点击 *Add new expression*，输入变量的名称，然后按 Enter，即可将变量添加到“Expressions”窗口。显示变量值所用的数字格式基于声明变量时与变量关联的数字格式。通过右键点击变量，导航至 *Number Format* 并选择所需的格式，可以为特定变量更改所需的数字格式。
2. 或者，可以通过右键点击“Expressions”窗口并点击“Import”将一组变量导入到“Expressions”窗口中，然后浏览至工程目录（`<install_location>\solutions\universal_motorcontrol_lab\common\debug\`），选择 *universal_lab_level1.txt* 文件，然后点击“OK”以导入图 1-1 中所示的变量。

> **备注**
> 此时主代码中的某些变量尚未初始化，可能包含一些无用的值。

备注

结构变量 *motorVars_M1* 引用了大多数与控制电机驱动相关的变量。

3. 点击“Expressions Window”选项卡右上角的“Continuous Refresh”按钮  ，启用微控制器的数据定期捕捉功能。通过点击“View Menu”按钮（“Expressions”窗口右上角的 3 个点），您可以选择 *Continuous Refresh Interval* 并编辑“Expressions”窗口的刷新率。请注意，将间隔时间选的过短会影响性能。

3.5.1.3 运行代码

1. 通过按“run”  按钮来运行工程，或点击“Debug”选项卡中的 *Run* → *Resume*。
2. 现在工程应该运行，而图和监视窗口中的值应该持续更新。
3. 在监视窗口中看到 *systemVars.flagEnableSystem* 自动设置为 1 后，在“Expressions”窗口中，将 *motorVars_M1.flagEnableRunAndIdentify* 变量设置为 1。
4. 工程现在应该已运行，在使用该工程时图和表达式窗口中的值应不断更新，如图 1-1 所示。您可能需要根据您的偏好来调整窗口大小。
5. 在监视视图中，如果没有故障，变量 *motorVars_M1.flagRunIdentAndOnLine* 应自动设置为 1。*ISRCount* 应不断增加。如果 *ISRCount* 没有增加，请确保已定义 TEST_ENABLE 预定义。如果未定义此预定义，则 *ISRCount* 不会在代码中增加。
6. 检查电机驱动器板的校准偏移。电机相电流检测值的偏移值应约等于 ADC 标度电流的一半，相电压偏移应约等于 0.5，如图 1-1 所示。
7. 如果使用图形工具，图中显示的变量是 u 相和 v 相的相电流。它们的幅度应接近于 0。
8. 展开和检查 *MotorVars_M1.faultMtrPrev.bit* 结构，以确保未设置故障标志。
9. 使用示波器探测用于电机驱动控制的 PWM 输出。在此构建级别中，三个 PWM 的占空比设置为 50%。预期的 PWM 输出波形如图 1-1 所示。PWM 开关频率将与在 *user_mtr1.h* 文件中为 USER_M1_PWM_FREQ_kHz 定义设置的值相同。
10. 将变量 *motorVars_M1.flagEnableRunAndIdentify* 变量设为 0 以停用 PWM。
11. 如果之前的任何步骤产生意外结果，则需要进行额外的调试。需要检查的几个事项是：
    a. 确保正确设置 DRV 板，并在 EVM 上组装适当的电容器/电阻器。
    b. 确保正在使用的电机驱动器板与构建配置中选择的板相同（请参阅 1）。
    c. 确保设置了正确的预定义。
    d. 确保在 C2000 MCU Lunchpad/ControlCARD 上正确配置开关，如节 3.2 所述。
12. 完成上述步骤后，现在可以停止控制器并终止调试连接。通过首先点击工具栏上的“Halt”按钮  ，或依次点击 *Target* → *Halt* 来完全停止控制器。最后，通过点击  按钮或依次点击 *Run* → *Reset* → CPU Reset 来复位控制器。
13. 通过点击“Terminate Debug Session”按钮  或点击 *Run* → *Terminate* 来关闭 CCS 调试会话。这将暂停程序并从 MCU 上断开 Code Composer Studio。
14. 每次用户重新更改或运行代码时，都不必终止调试会话。但可以遵循以下过程。重新构建工程后，按下  按钮或依次点击 *Run* → *Reset* → CPU Reset，然后按  按钮或点击 *Run* → *Restart*。如果目标器件或配置发生更改，则必须在关闭 CCS 之前终止工程。

图 3-30. 构建级别 1：“Expressions”窗口中的变量

图 1-1 展示了在栅极驱动输入端具有死区输出的 C2000 PWM。

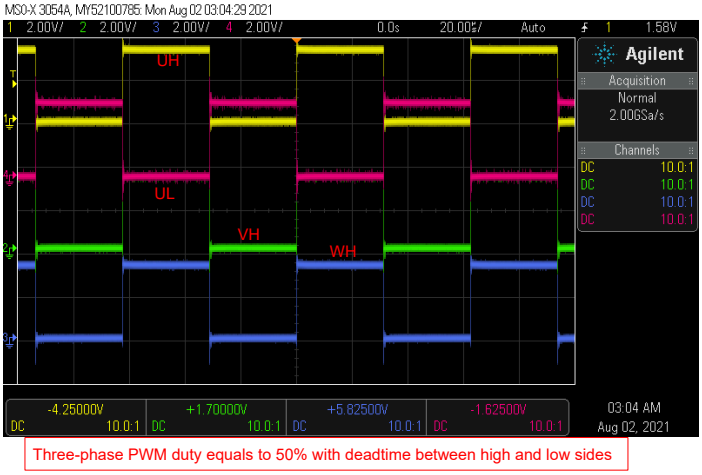


图 3-31. 构建级别 1：PWM 输出波形

3.5.2 级别 2 增量构建

了解该构建级别中的目标：

- 实现简单的电机标量 v/f 控制以驱动双电机，从而验证电流和电压检测电路以及栅极驱动器电路。
- 测试用于电机控制的 InstaSPIN-FOC FAST 或 eSMO 模块。

在该构建级别，系统以开环控制方式运行，因此 ADC 值仅用于验证和确认，不会实际用在电机的控制环路中。图 1-1 展示了该构建级别的软件流程。

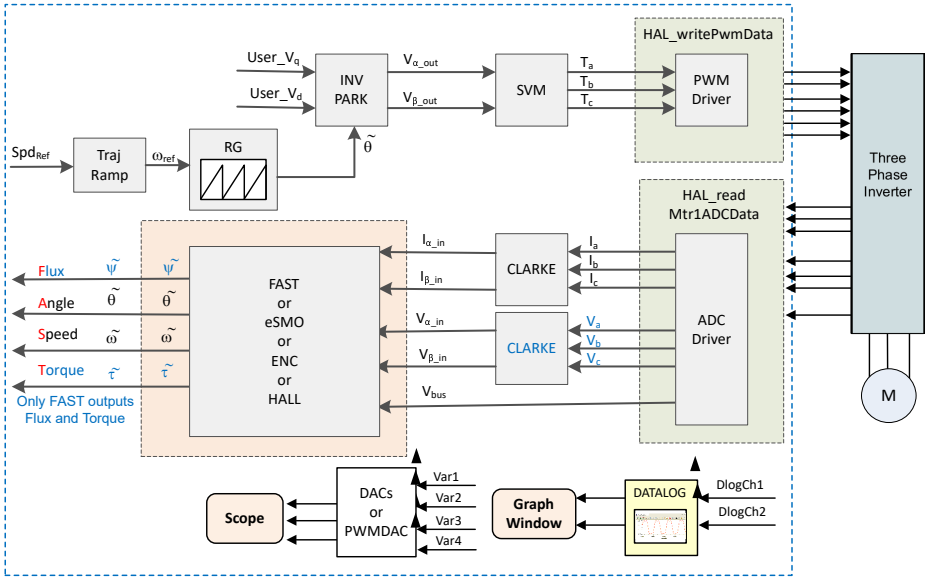


图 3-32. 构建级别 2 软件方框图 – 开环控制

3.5.2.1 构建和加载工程

将电机连接到电机驱动器评估板上的相应端子。按照节 3.5.1.1 中的步骤 2-7 构建和加载工程。在步骤 4 中，将 DMC_BUILDLEVEL 设置为 DMC_LEVEL_2。

> **小心**
> 在将代码载入闪存时请勿点击 *Cancel*、关闭板的电源或断开仿真器。

3.5.2.2 设置调试环境窗口

按照节 3.5.1.2 中的步骤将变量导入“Expressions”窗口。对于构建级别 2，选择 *universal_lab_level2.txt* 文件。此时将显示“Expressions”窗口，如图 1-1 所示。

3.5.2.3 运行代码

1. 为适当的电源加电，并逐渐增加电源的输出电压以获得适当的直流母线电压。
2. 如果使用图形工具，则实验 2 使用与实验 1 相同的图形配置和参数来监控 2 个相电流。
3. 通过点击  按钮来运行工程，或点击“Debug”选项卡中的 *Run → Resume*。*systemVars.flagEnableSystem* 应在固定时间后设为 *1*，这意味着已完成偏移校准。故障标志 *motorVars_M1.faultMtrUse.all* 应等于 *0*。如果情况并非如此，用户应按照节 3.5.1.3 中所述仔细检查实验 1 中的电流和电压检测电路。
4. 要验证电机逆变器的电流和电压检测电路，请在“Expressions”窗口中将变量 *motorVars_M1.flagEnableRunAndIdentify* 设置为 *1*，如图 1-1 所示。电机将以电压/频率 (v/f) 开环运行。如果电机旋转不平稳，请根据电机规格调整 *user_mtr1.h* 文件中的 v/f 曲线参数，如下所示。有关调整 v/f 曲线参数的更多详细信息，请参阅 7.a。注意：修改这些参数将需要重新编译工程。有关在调试模式下重新编译工程的更多信息，请参阅 14 的步骤 14。

```
#define USER_MOTOR1_FREQ_LOW_HZ             (5.0)           // Hz
#define USER_MOTOR1_FREQ_HIGH_HZ            (400.0)         // Hz
#define USER_MOTOR1_VOLT_MIN_V              (1.0)           // Volt
#define USER_MOTOR1_VOLT_MAX_V              (24.0)          // Volt
```

5. *motorVars_M1.speedRef_Hz* 变量用于设置电机的速度基准。在“Expressions”窗口中检查 *motorVars_M1.speed_Hz* 变量的值，以保持电机转速 (*motorVars_M1*speed_Hz) 接近于基准速度 (*motorVars_M1.speedRef_Hz)*，如图 1-1 所示。
6. 在此构建级别中，需要验证电流检测、电压检测、转子角度估算器和发生器。这可以使用 PWMDAC 或 DAC128S 模块并借助示波器来完成，如节 3.4.2 或节 3.4.3 所述。此外，DATALOG 模块可用于查看这些感应波形。有关使用 DATALOG 查看电流、电压和角度信号的更多信息，请参阅步骤 7。如果使用 DAC128S 模块，请配置 dac128s.ptrdata[] 输入，使其与以下每个子部分中显示的代码相对应。要进行代码修改，可以取消注释用于 DAC128S 初始化的相应代码部分，并注释掉该小节中不需要的 DAC128S 初始化的其余部分。在修改代码以与每个小节相对应后，需要重新编译工程。有关在调试模式下重新编译工程的更多信息，请参阅 14 的步骤 14。
   a. 要监控的第一组参数是相电流。为此，需要对如下所示的代码部分取消注释，代码部分用于将 dac128s 指针数据设置为 3 个相电流以及角度估算器变量。此代码位于 *sys_main.c* 文件中。预期电流波形应与图 1-1 中示波器上显示的波形类似。在 DAC128S 输出上测量的电流波形应该几乎与电流探头捕获的相应相电流波形相同。如果是这种情况，则表示电流检测电路适用于电机控制。如果情况并非如此，则可能需要调整 *user_mtr1.h* 文件中的 v/f 参数，如 7.a 中所述。

```
dac128s.ptrData[0] = &motorVars_M1.adcData.I_A.value[0];     // CH_A
dac128s.ptrData[1] = &motorVars_M1.adcData.I_A.value[1];     // CH_B
dac128s.ptrData[2] = &motorVars_M1.adcData.I_A.value[2];     // CH_C
dac128s.ptrData[3] = &motorVars_M1.angleGen_rad;             // CH_D
dac128s.ptrData[4] = &motorVars_M1.angleEST_rad;                // CH_E, N/A
dac128s.ptrData[5] = &motorVars_M1.adcData.V_V.value[0];        // CH_F, N/A
dac128s.ptrData[6] = &motorVars_M1.adcData.V_V.value[1];        // CH_G, N/A
dac128s.ptrData[7] = &motorVars_M1.adcData.V_V.value[2];        // CH_H, N/A

dac128s.gain[0] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;
dac128s.gain[1] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;
dac128s.gain[2] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;
dac128s.gain[3] = 4096.0f / MATH_TWO_PI;
dac128s.gain[4] = 4096.0f / MATH_TWO_PI;                              // NA
dac128s.gain[5] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V; // N/A
dac128s.gain[6] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V; // N/A
dac128s.gain[7] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V; // N/A
... ...
dac128s.offset[0] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[1] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[2] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[3] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[4] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[5] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[6] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[7] = (uint16_t)(0.5f * 4096.0f); // N/A
```

b. 要监控的第二组参数是相电压。为此，对于将 dac128s 指针数据设置为指向相电压的代码部分，应取消注释，如下所示。此代码位于 *sys_main.c* 文件中。DAC128S 模块的相电压检测的输出波形形状应与图 1-1 或图 1-1 中所示的图像类似。如果是这种情况，则表示电压检测电路按预期工作。请注意，振幅会随电源电压和使用的电机而变化，但波形的形状应该是相同的。请注意相位波形根据所使用的 SVM 模式而存在的差异。对于通用 SVM 模式，波形的上峰和下峰有电压骤降；而对于最小 SVM 模式，电压骤降仅出现在上峰，但在下峰保持平缓。要更改 SVM 模式，请从“Expressions”窗口中的 *motorVars_M1.svmMode* 枚举中选择 SVM_COM_C 或 SVM_MIN_C。

```
dac128s.ptrData[0] = &motorVars_M1.adcData.V_V.value[0];     // CH_A
dac128s.ptrData[1] = &motorVars_M1.adcData.V_V.value[1];     // CH_B
dac128s.ptrData[2] = &motorVars_M1.adcData.V_V.value[2];     // CH_C
dac128s.ptrData[3] = &motorVars_M1.angleGen_rad;             // CH_D
dac128s.ptrData[4] = &motorVars_M1.angleEST_rad;                // CH_E, N/A
dac128s.ptrData[5] = &motorVars_M1.adcData.I_A.value[0];        // CH_F, N/A
dac128s.ptrData[6] = &motorVars_M1.adcData.I_A.value[1];        // CH_G, N/A
dac128s.ptrData[7] = &motorVars_M1.adcData.I_A.value[2];        // CH_H, N/A

dac128s.gain[0] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V;
dac128s.gain[1] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V;
dac128s.gain[2] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V;
dac128s.gain[3] = 4096.0f / MATH_TWO_PI;
dac128s.gain[4] = 4096.0f / MATH_TWO_PI;                              // N/A
dac128s.gain[5] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A; // N/A
dac128s.gain[6] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A; // N/A
dac128s.gain[7] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A; // N/A
... ...
dac128s.offset[0] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[1] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[2] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[3] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[4] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[5] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[6] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[7] = (uint16_t)(0.5f * 4096.0f); // N/A
```

c. 要监控的第三组参数是角度发生器和角度估算器参数。为此，对于将 dac128s 指针数据设置为指向相应变量的代码部分，应取消注释，如下所示。此代码位于 *sys_main.c* 文件中。角度发生器和估算器波形的角度应与图 1-1 中所示的示波器波形相似。请注意，力角发生器的角度与 FAST 或 eSMO 估算器的估算转子角度非常相似。这表示 FAST 或 eSMO 估算器可按预期使用电机参数以及采样电流和电压信号。

```
dac128s.ptrData[0] = &motorVars_M1.angleGen_rad;            // CH_A
dac128s.ptrData[1] = &motorVars_M1.angleEST_rad;            // CH_B
dac128s.ptrData[2] = &motorVars_M1.anglePLL_rad;            // CH_C
dac128s.ptrData[3] = &motorVars_M1.adcData.I_A.value[0];    // CH_D
dac128s.ptrData[4] = &motorVars_M1.adcData.V_V.value[0];        // CH_E, N/A
dac128s.ptrData[5] = &motorVars_M1.adcData.I_A.value[1];        // CH_F, N/A
dac128s.ptrData[6] = &motorVars_M1.adcData.I_A.value[2];        // CH_G, N/A
dac128s.ptrData[7] = &motorVars_M1.adcData.V_V.value[1];        // CH_H, N/A

dac128s.gain[0] = 4096.0f / MATH_TWO_PI;
dac128s.gain[1] = 4096.0f / MATH_TWO_PI;
dac128s.gain[2] = 4096.0f / MATH_TWO_PI;
dac128s.gain[3] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;
dac128s.gain[4] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V; // N/A
dac128s.gain[5] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A; // N/A
dac128s.gain[6] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A; // N/A
dac128s.gain[7] = 2.0f * 4096.0f / USER_M1_ADC_FULL_SCALE_VOLTAGE_V; // N/A

dac128s.offset[0] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[1] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[2] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[3] = (uint16_t)(0.5f * 4096.0f);
dac128s.offset[4] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[5] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[6] = (uint16_t)(0.5f * 4096.0f); // N/A
dac128s.offset[7] = (uint16_t)(0.5f * 4096.0f); // N/A
```

7. 如果将 DATALOG 模块与图形工具配合使用来检查电流检测信号、电压检测信号和角度输出，请按照下述步骤操作。关于 DATALOG 模块的更多信息，请参阅节 3.4.1。注意：在修改代码后，有必要在以下每一步之间重新编译工程。有关在调试模式下重新编译工程的更多信息，请参阅 14 的步骤 14。

a. 要使用 DATALOG 模块测试 U 和 V 相电流，必须在 *sys_main.c* 文件中设置以下代码。

备注

默认情况下，已为构建级别 2 配置此代码。图形工具上显示的相电流采样信号波形如图 1-1 所示。

```
datalogObj->iptr[0] = &motorVars_M1.adcData.I_A.value[0];
datalogObj->iptr[1] = &motorVars_M1.adcData.I_A.value[1];
```

b. 要使用 DATALOG 模块测试 U 相和 V 相的相电压，必须在 *sys_main.c* 文件中设置以下代码并将其修改为如下所示的代码。图形工具上的相电压采样信号波形如图 1-1 所示。

```
datalogObj->iptr[0] = &motorVars_M1.adcData.V_V.value[0];
datalogObj->iptr[1] = &motorVars_M1.adcData.V_V.value[1];
```

c. 将 DATALOG 模块的四个输入配置为以下代码。图形工具上力角生成器或估算器波形的角度，如图 1-1 中所示。请注意，力角发生器的角度与 FAST 或 eSMO 估算器的估算转子角度非常相似。

```
datalogObj->iptr[0] = &motorVars_M1.angleFOC_rad;
datalogObj->iptr[1] = &motorVars_M1.angleEST_rad;
```

8. 通过减小变量 *motorVars_M1.overCurrent_A* 的值来验证过流故障保护，过流保护由 CMPSS 模块实现。如果 *motorVars_M1.overCurrent_A* 设为小于电机相电流实际值的值，则会触发过流故障，PWM 输出会被禁用，*motorVars_M1.flagEnableRunAndIdentify* 将被设置为 *0*，*motorVars_M1.faultMtrUse.all* 将被设置为 *0x10 (16)*，如*图 1-1* 所示。
9. 将变量 *motorVars_M1.flagEnableRunAndIdentify* 设为 0 停止运行电机。
10. 完成后，现在可以停止控制器，并终止调试连接。通过首先点击工具栏上的“Halt”按钮 或点击 *Target → Halt* 来完全停止控制器。最后，通过点击 或点击 *Run → Reset* 来重置控制器。
11. 通过点击“Terminate Debug Session” 或点击 *Run → Terminate* 来关闭 CCS 调试会话。
12. 关闭逆变器套件的电源。

图 3-33. 构建级别 2：“Expressions”窗口中的变量

在“Expression”表达式窗口中调整 *motorVars_M1.overCurrent_A* 的值，以触发过流故障，如图 1-1 所示。

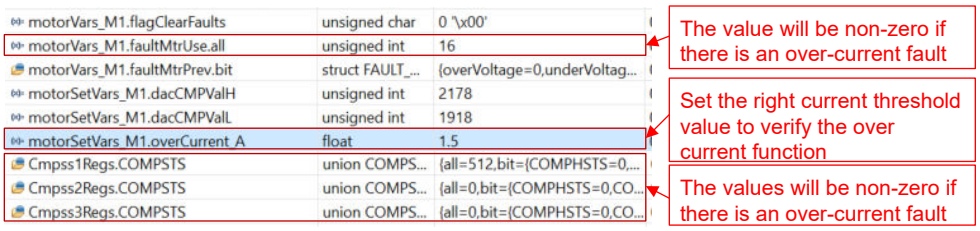


图 3-34. 构建级别 2：电流保护设置

使用 DAC128S085EVM 与示波器来监测电机的三相检测电流，并使用电流探头将采样值与测量值进行比较，如图 1-1 所示。

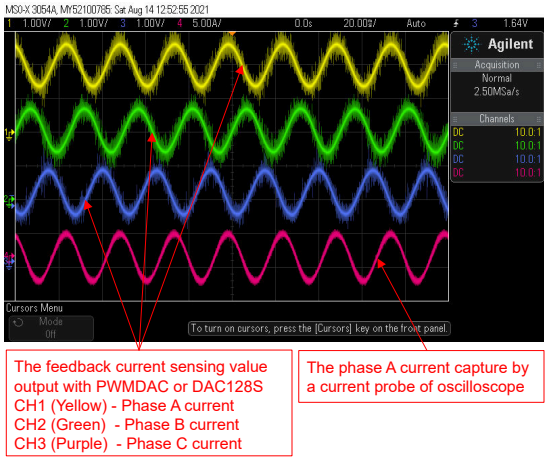


图 3-35. 构建级别 2：电机相电流波形

使用 DAC128S085EVM 和示波器来监测电机的三相检测电压，并通过将 *motorVars_M1.svmMode* 设置为 SVM_COM_C 来使用共模 SVPWM，如图 1-1 所示。

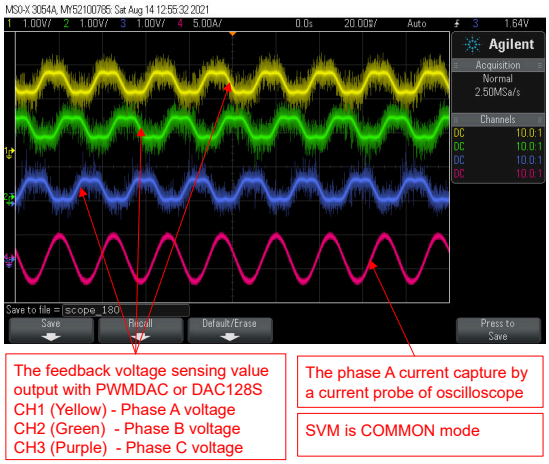


图 3-36. 构建级别 2：使用共模 SVM 的电机相电压波形

搭配使用 DAC128S085EVM 和示波器来监测电机的三相检测电压，并通过将 *motorVars_M1.svmMode* 设置为 SVM_MIN_C 来使用最小模式 SVPWM，如图 1-1 所示。

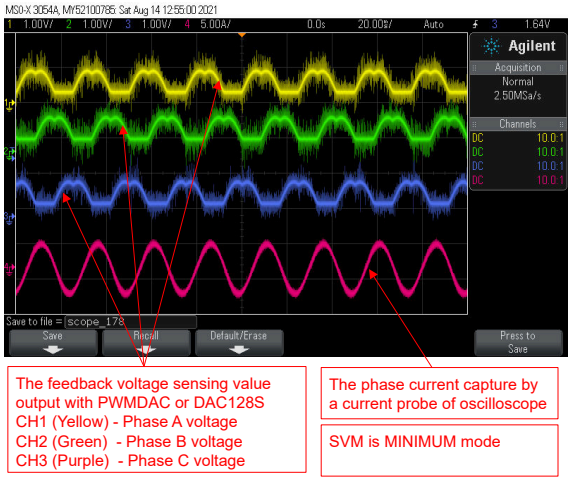


图 3-37. 构建级别 2：使用最小 SVM 模式的电机相电压波形

搭配使用 DAC128S085EVM 与示波器来从角度发生器监控电机的转子角度和 FAST 估算器的角度，如图 1-1 所示。

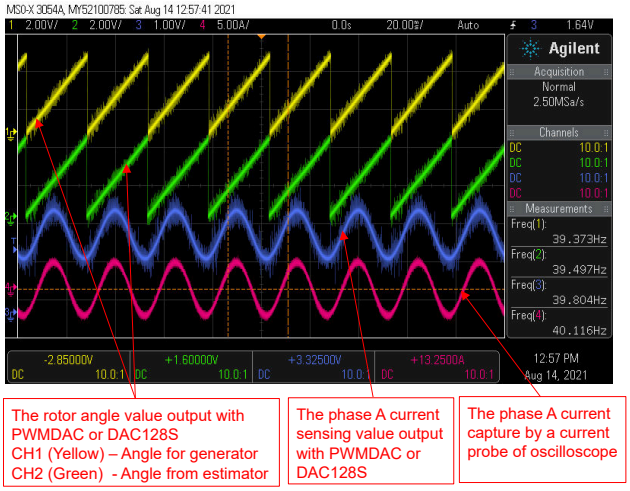


图 3-38. 构建级别 2：电机转子角度和相电流波形

将 DATALOG 与图形工具一起使用，以监测电机的三相检测电流，如图 1-1 所示。

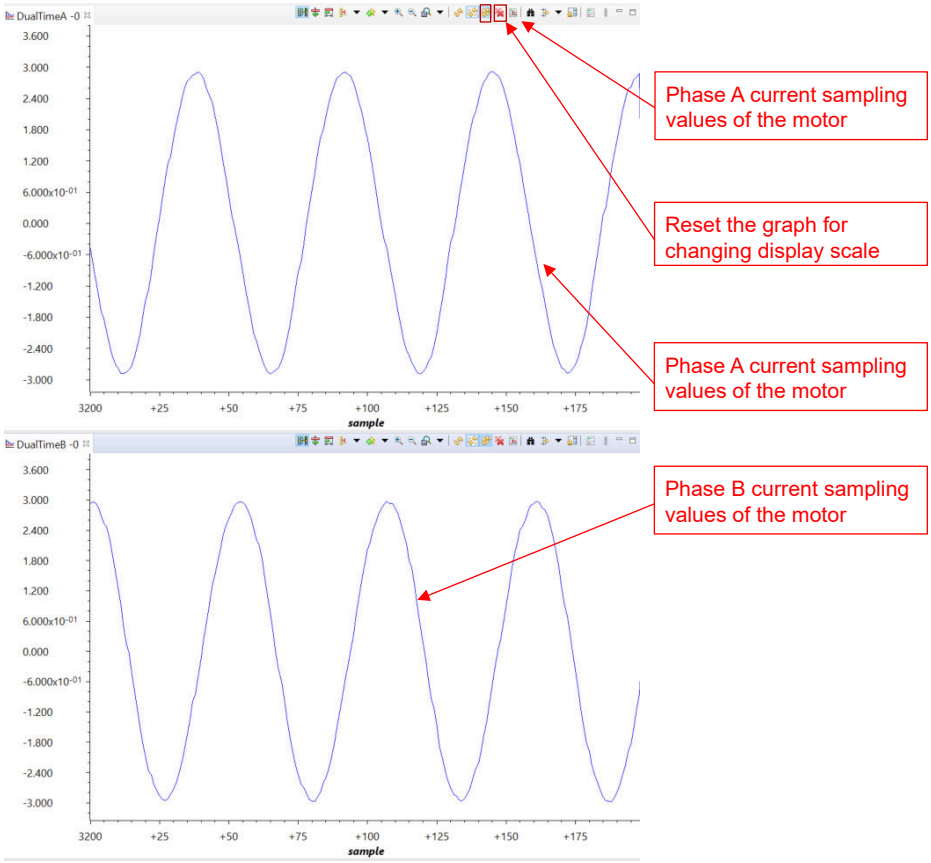


图 3-39. 构建级别 2：使用图形工具时的电机相电流波形

使用数据记录器和图形工具监测电机的三相检测电压，如图 1-1 所示。

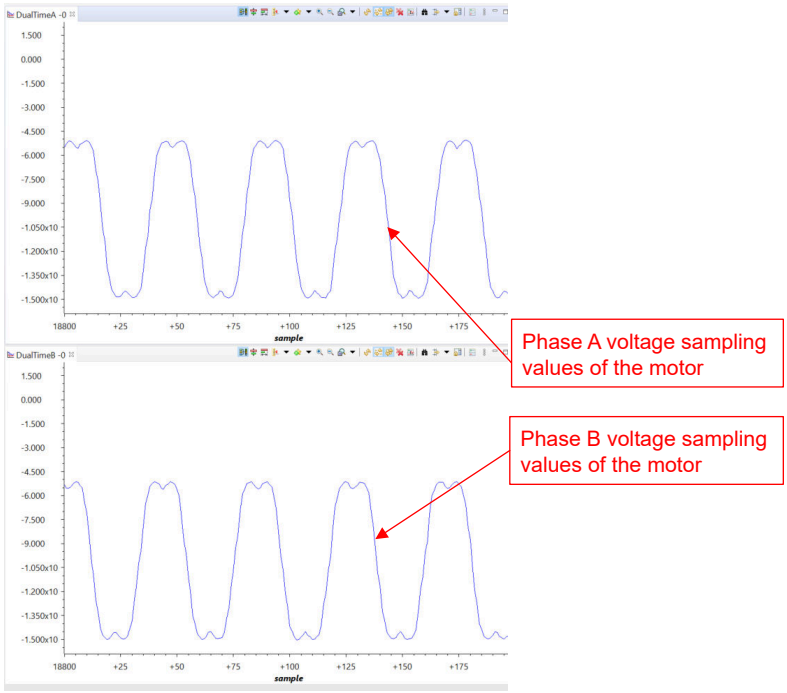


图 3-40. 构建级别 2：使用图形工具时的电机相电压波形

将 Datalog 与图形工具配合使用，从角度发生器监视电机的转子角度以及 FAST 估算器的角度，如图 1-1 所示。

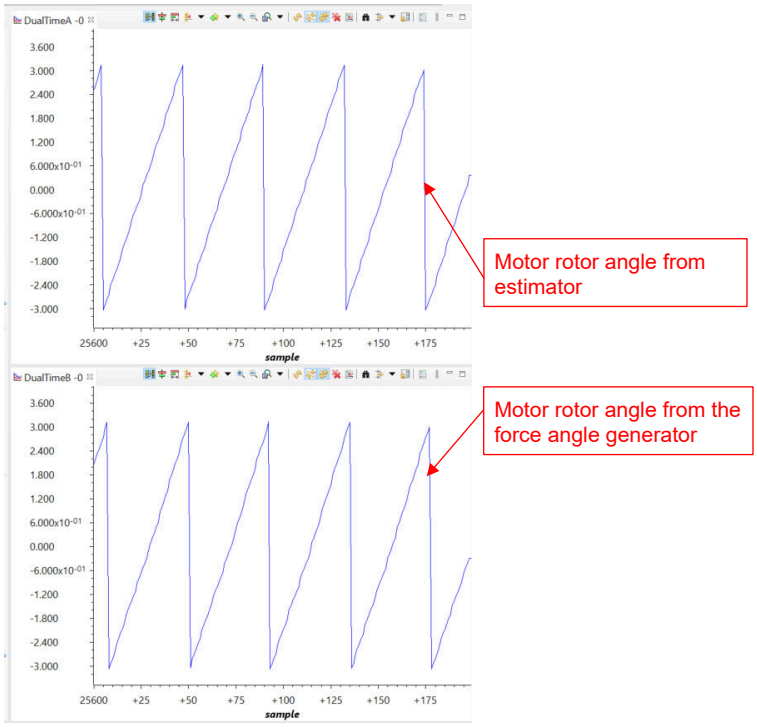


图 3-41. 构建级别 2：使用图形工具时的电机转子角度波形

3.5.3 级别 3 增量构建

了解该构建级别中的目标：

- 评估电机的闭合电流环路运行。
- 验证电流检测参数设置

在这个构建级别，电机由 i/f 控制进行控制，转子角度由斜坡发生器模块生成。图 1-1 展示了该构建级别的软件流程。

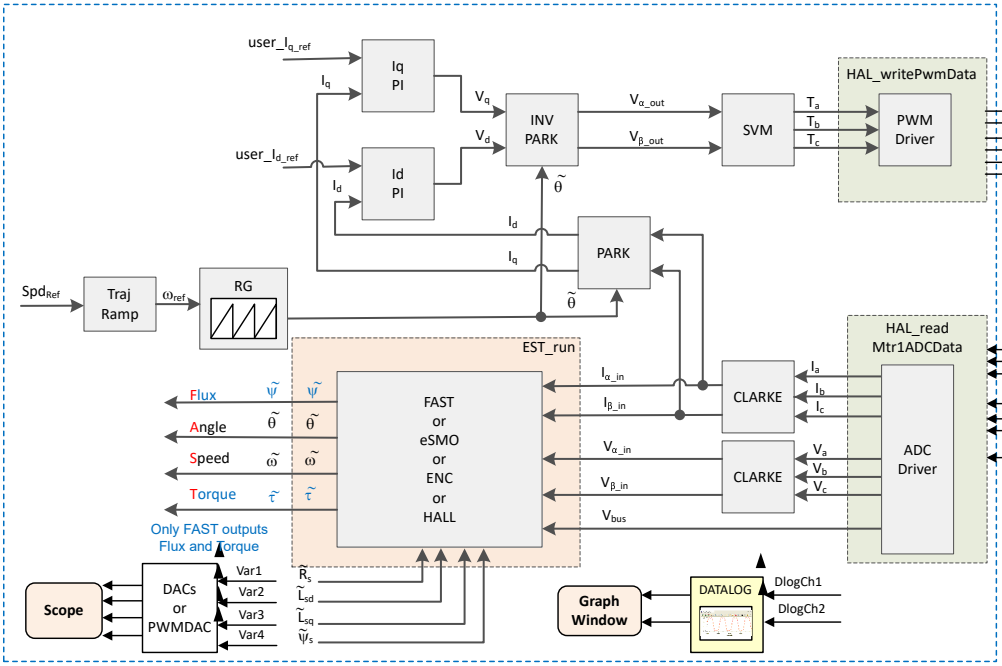


图 3-42. 构建级别 3 软件方框图 – 电流闭环控制

3.5.3.1 构建和加载工程

将电机连接到电源逆变器板上的相关端子。按照节 3.5.1.1 中的操作步骤，通过在 *sys_settings.h* 文件中将 DMC_BUILDLEVEL 设置为 DMC_LEVEL_3 来构建和加载工程。

> **小心**
> 在将代码载入闪存时请勿点击 *Cancel*、关闭板的电源或断开仿真器。

3.5.3.2 设置调试环境窗口

通过选择 *universal_lab_level3.txt*，按照节 3.5.1.2 中的操作步骤将变量导入“Expressions”窗口。此时将显示“Expressions”窗口，如图 1-1 所示。

3.5.3.3 运行代码

1. 打开交流或直流电源，逐渐增加电源上的输出电压以获得适当的直流母线电压。
2. 通过点击按钮 来运行工程，或点击“Debug”选项卡中的 *Run → Resume*。
   systemVars.flagEnableSystem 应在固定时间后设为 *1*，这意味着已完成偏移校准。故障标志 *motorVars_M1.faultMtrUse.all* 应等于 *0*，否则用户必须检查电流和电压检测电路，如节 3.5.1 中所述。
3. 要验证运行电机时进行电流闭环控制，请在“Expressions”窗口中将变量 *motorVars_M1.flagEnableRunAndIdentify* 设置为 *1*，如图 1-1 中所示。电机在运行时将使用来自角度发生器的角度以在 *motorVars_M1.speedRef_Hz* 变量中设置的速度进行闭环控制。在“Expressions”窗口中检查 *motorVars_M1.speed_Hz* 的值，两个变量的值应非常接近。

4. 将示波器探头连接到 EPWMDAC 或 DAC128S 输出和电机相线，以探测角度和电流信号以及电流。示波器上的这些波形如图 1-1 所示。在“Expressions”窗口中更改 *motorVars_M1.Idq_set_A.value[1]* 以设置基准扭矩电流，相应地，电机相电流将以相同的百分比增加。
5. 如果电机无法以电流闭环运行并出现过流故障，则检查是否根据硬件板正确设置了 *motorVars_M1.adcData.current_sf* 的符号和 *userParams_M1.current_sf* 的值。这两个变量的值都与 *user_mtr1.h* 文件中的定义常量 USER_M1_ADC_FULL_SCALE_CURRENT_A 相关。
6. 将变量 *motorVars_M1.flagEnableRunAndIdentify* 设为 0 停止运行电机。
7. 完成后，现在可以停止控制器，并终止调试连接。通过首先点击工具栏上的“Halt”按钮 或点击 *Target → Halt* 来完全停止控制器。最后，通过点击 或点击 *Run → Reset* 来重置控制器。
8. 通过点击“Terminate Debug Session” 或点击 *Run → Terminate* 来关闭 CCS 调试会话。

图 3-43. 构建级别 3：“Expressions”窗口中的变量

搭配使用 DAC128S085EVM 与示波器来从角度发生器监控电机的转子角度和 FAST 估算器的转子角度，以及电机的相电流，如图 1-1 所示。

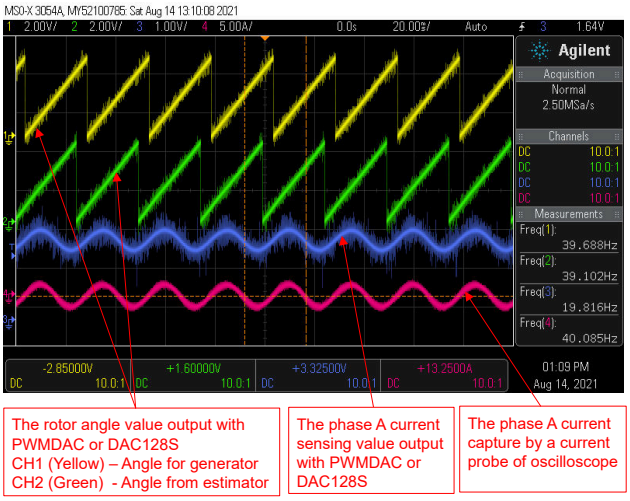


图 3-44. 构建级别 3：示波器上的电机转子角度和相电流波形

3.5.4 级别 4 增量构建

了解该构建级别中的目标：

- 使用 FAST 估算器评估电机识别。
- 使用基于 FAST 的无传感器 FOC、基于 eSMO 的无传感器 FOC、基于编码器的有传感器 FOC 或基于霍尔的有传感器 FOC 评估整个电机驱动器。
- 评估其他特性，例如弱磁控制、快速启动、MTPA 和制动。

在此构建级别，外部速度环路是闭合的，内部电流环路用于电机，使得转子角度来自 FAST、eSMO、编码器或霍尔传感器模块。图 1-1 展示了该构建级别的软件流程。

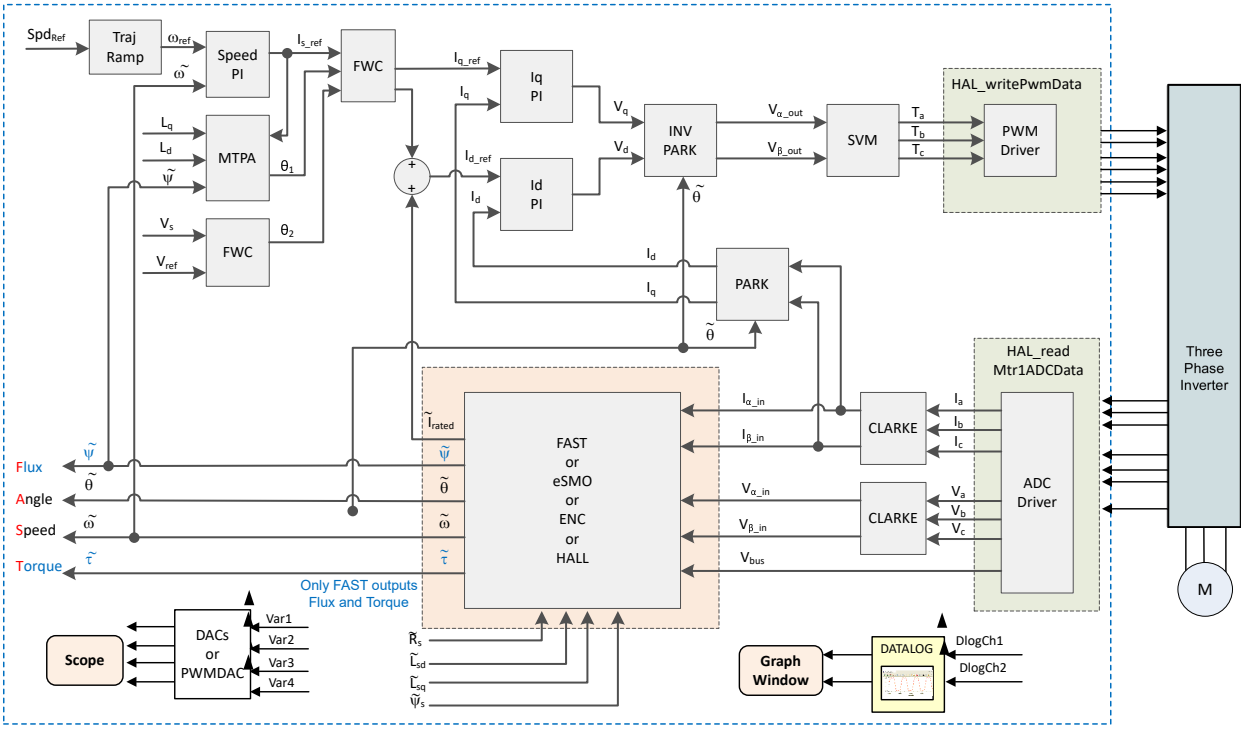


图 3-45. 构建级别 4 软件方框图 – 转速和电流闭环控制

3.5.4.1 构建和加载工程

将电机连接到电源逆变器板上的相关端子。按照节 3.5.1.1 中的操作步骤，通过在 *sys_settings.h* 文件中将 DMC_BUILDLEVEL 设置为 DMC_LEVEL_4 来构建和加载工程。

> **小心**
> 在将代码载入闪存时请勿点击 *Cancel*、关闭板的电源或断开仿真器。

3.5.4.2 设置调试环境窗口

通过选择 *universal_lab_level4.txt*，按照节 3.5.1.2 中的操作步骤将变量导入“Expressions”窗口。此时将显示“Expressions”窗口，如图 1-1 所示。

3.5.4.3 运行代码

1. 将交流电源输出设置为 0V (50/60Hz)，打开交流电源，将输入电压从 0V 缓慢增加至 220V 交流。
2. 必须在头文件 *user_mtr1.h* 中定义所需的电机参数，如以下示例代码所示。如果用户不太了解电机参数，那么在示例实验中使用 FAST 估算器的情况下，可以使用电机识别来获得电机参数。

```
#define USER_MOTOR1_TYPE MOTOR_TYPE_PM
#define USER_MOTOR1_NUM_POLE_PAIRS (4)
#define USER_MOTOR1_Rr_Ohm (NULL)
#define USER_MOTOR1_Rs_Ohm (0.38157931f)
#define USER_MOTOR1_Ls_d_H (0.000188295482f)
#define USER_MOTOR1_Ls_q_H (0.000188295482f)
#define USER_MOTOR1_RATED_FLUX_VpHz (0.0396642499f)
```

3. 将 *sys_main.c* 文件中的 *userParams.flag_bypassMotorId* 值更改为“false”以启用电机识别并作为以下示例代码。

```
// false->enable identification, true->disable identification
    userParams_M1.flag_bypassMotorId = false;
```

4. 根据电机规格在 *user_mtr1.h* 文件中设置正确的识别值。

```
#define USER_MOTOR1_RES_EST_CURRENT_A       (1.5f)      // A - 10~30% of rated current of the
motor
#define USER_MOTOR1_IND_EST_CURRENT_A       (-1.0f)     // A - 10~30% of rated current of the
motor, just enough to enable rotation
#define USER_MOTOR1_MAX_CURRENT_A           (4.5f)      // A - 30~150% of rated current of the
motor
#define USER_MOTOR1_FLUX_EXC_FREQ_Hz        (40.0f)     // Hz - 10~30% of rated frequency of
the motor
```

5. 重新构建工程并将代码加载到控制器中，通过点击按钮  来运行工程，或点击“Debug”选项卡中的 *Run → Resume*。经过固定的时长后，*systemVars.flagEnableSystem* 应设置为 *1*，这意味着偏移校准已完成并且浪涌电源继电器已开启。故障标志 *motorVars_M1.faultMtrUse.all* 应等于 *0*，否则用户应检查电流和电压检测电路，如节 3.5.1 中所述。
6. 在“Expressions”窗口中将变量 *motorVars_M1.flagEnableRunAndIdentify* 设置为 *1*（如图 1-1 所示），此时将执行电机识别，整个过程需要大约 150s。*motorVars_M1.flagEnableRunAndIdentify* 等于 *0* 并且电机停止后，电机参数即被识别。复制“watch”窗口中的变量值，替换 *user_mtr1.h* 文件中定义的电机参数，如下所示：
   - USER_MOTOR1_Rs_Ohm = motorSetVars_M1.Rs_Ohm 的值
   - USER_MOTOR1_Ls_d_H = motorSetVars_M1.Ls_d_H 的值
   - USER_MOTOR1_Ls_q_H = motorSetVars_M1.Ls_q_H 的值
   - USER_MOTOR1_RATED_FLUX_VpHz = motorSetVars_M1.flux_VpHz 的值
7. 成功识别电机参数后，将 *userParams_M1.flag_bypassMotorId* 值设置为“true”以禁用识别，重新构建工程并将代码加载到控制器中。

8. 按照以下步骤，该示例可以支持在线识别电机，而无需重新加载代码。
   a. 将 *motorVars_M1.flagEnableRunAndIdentify* 设为 *0* 停止运行电机。
   b. 将 *motorVars_M1.flagEnableMotorIdentify* 设置为“1”以启用识别。
   c. 将 *motorVars_M1.flagEnableRunAndIdentify* 设为 *1* 开始识别电机参数。*motorVars_M1.flagEnableMotorIdentify* 将自动设为“0”，这意味着识别正在进行。
   d. 如上面的步骤 6 所述，将识别新的电机参数。
9. 识别完整的电机参数或在 *user_mtr1.h* 文件中设置正确的电机参数后。要开始运行电机，请执行以下步骤。
   a. 再次将变量 *motorVars_M1.flagEnableRunAndIdentify* 设为 *1* 以开始运行电机。
   b. 将目标速度值设为变量 *motorVars_M1.speedRef_Hz*，并观察电机轴速度如何随设定速度改变。
   c. 要改变加速度，请在变量 *motorVars_M1.accelerationMax_Hzps* 中输入不同的加速度值。
   d. 如节 3.4.2 或节 3.4.3 所述，使用 PWMDAC 或 DAC128S 模块显示监视变量。电机角度和电流波形如图 1-1 所示。
10. FOC 系统电流控制器的默认比例增益 (Kp) 和积分增益 (Ki) 在函数 *setupControllers()* 中计算。调用 *setupControllers*() 后，全局变量 motorSetVars_M1.Kp_Id、motorSetVars_M1.Ki_Id、motorSetVars_M1.Kp_Iq 和 motorSetVars_M1.Ki_Iq 将使用新计算得出的 Kp 和 Ki 增益进行初始化。如图 1-1 中所示，调整“Expressions”窗口中这四个变量的 Kp 和 Ki 值，使电流控制器实现预期的电流控制带宽和响应。Kp 增益会产生一个零点，以抵消电机定子的极点，可以轻松地计算得出。Ki 增益可调整电流控制器-电机系统的带宽。如果需要一个速度控制系统来实现特定阻尼，电流控制器的 Kp 增益将与速度控制系统的时间常数相关。
11. FOC 系统速度控制器的默认比例增益 (Kp) 和积分增益 (Ki) 也在函数 *setupControllers()* 中计算。调用 *setupControllers()* 后，全局变量 motorSetVars_M1.Kp_spd 和 motorSetVars_M1.Ki_spd 将使用新计算得出的 Kp 和 Ki 增益进行初始化。如图 1-1 中所示，调整“Expressions”监视窗口中这两个变量的 Kp 和 Ki 值，使速度控制器实现预期的电流控制带宽和响应。与调整电流控制器相比，调整速度控制器的未知情况更多，默认计算得出的 Kp 和 Ki 只是参考值作为起点。
12. 将变量 *motorVars_M1.flagEnableRunAndIdentify* 设为“0”停止运行电机。
13. 完成后，现在可以停止控制器，并终止调试连接。通过首先点击工具栏上的“Halt”按钮 或点击 *Target* → *Halt* 来完全停止控制器。最后，通过点击 或点击 *Run* → *Reset* 来重置控制器。
14. 通过点击“Terminate Debug Session” 或点击 *Run* → *Terminate* 来关闭 CCS 调试会话。

图 3-46. 构建级别 4：“Expressions”窗口中的变量

当电机正向旋转时，将 DAC128S085EVM 与示波器结合使用可监测来自 FAST 估算器的电机转子角度、电机的反馈速度以及电机的相电流，如图 1-1 所示，方法是将 *motorVars_M1.speedRef_Hz* 设置为正基准值。

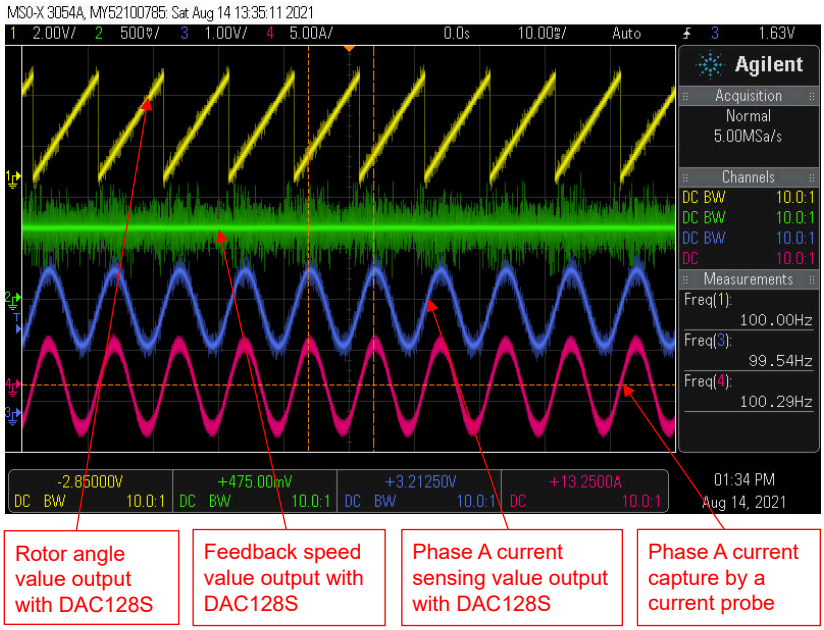


图 3-47. 构建级别 4：采用 FAST 的转子角度，正向移动时的相电流波形

如节 3.3.2 所示，在示例实验中可以支持多种 FOC 算法。用户可以在实验工程中使用一个或两个算法进行电机控制，如表 3-7 所示。

用户可以通过在工程属性中添加预定义名称 *MOTOR1_FAST* 和 *MOTOR1_ESMO*，在工程中同时实施 FAST 和 eSMO 估算器，如节 3.3.1 中所述。按照上述操作步骤重新构建、加载和运行工程。设置将如图 1-1 中所示。

- systemVars.estType 值等于 EST_TYPE_FAST_ESMO，这意味着 FAST 和 eSMO 估算器在该工程中处于启用状态。
- motorVars_M1.estimatorMode 等于 ESTIMATOR_MODE_FAST，这意味着 FAST 估算器正在用于无传感器 FOC；等于 ESTIMATOR_MODE_ESMO，这意味着 eSMO 估算器正在用于无传感器 FOC。
- 图 1-1 显示了 FAST 和 eSMO 的估算转子角度。通过将 motorVars_M1.speedRef_Hz 设置为*正*值，电机在 FAST *正向*旋转的情况下运行。
- 图 1-1 显示了 FAST 和 eSMO 的估算转子角度。通过将 motorVars_M1.speedRef_Hz 设置为*负*值，电机在 FAST *逆向*旋转的情况下运行。
- 用户可以将值更改为 ESTIMATOR_MODE_ESMO 以选择无传感器 FOC 的 eSMO 估算器。用户还可以更改值以动态地使用估算器进行切换。

当电机正向旋转时，将 DAC128S085EVM 与示波器结合使用可监测来自 FAST 和 eSMO 估算器的电机转子角度、电机的相电流，如图 1-1 所示，方法是将 *motorVars_M1.speedRef_Hz* 设置为正基准值。

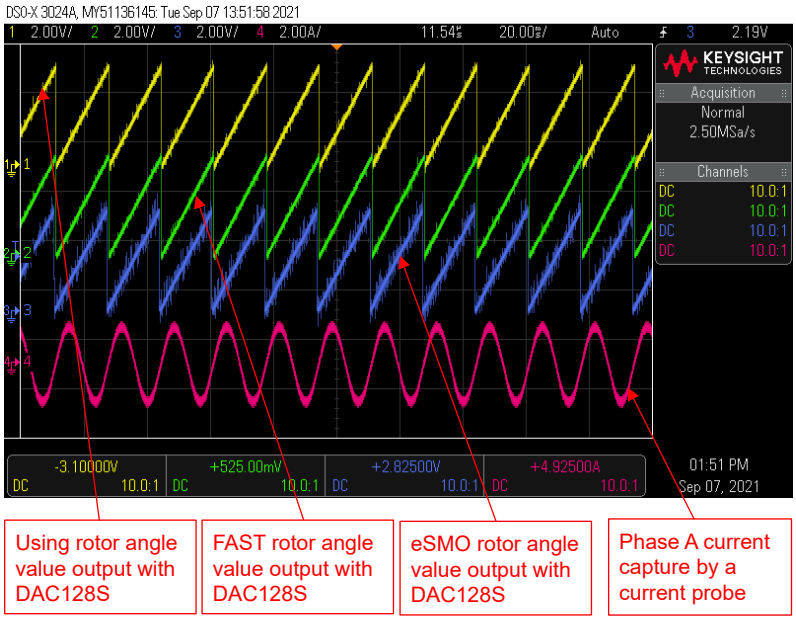


图 3-48. 构建级别 4：采用 FAST 和 eSMO 的转子角度，正向旋转时的相电流波形

当电机逆向旋转时，将 DAC128S085EVM 与示波器结合使用可监测来自 FAST 和 eSMO 估算器的电机转子角度、电机的相电流，如图 1-1 所示，方法是将 *motorVars_M1.speedRef_Hz* 设置为负基准值。

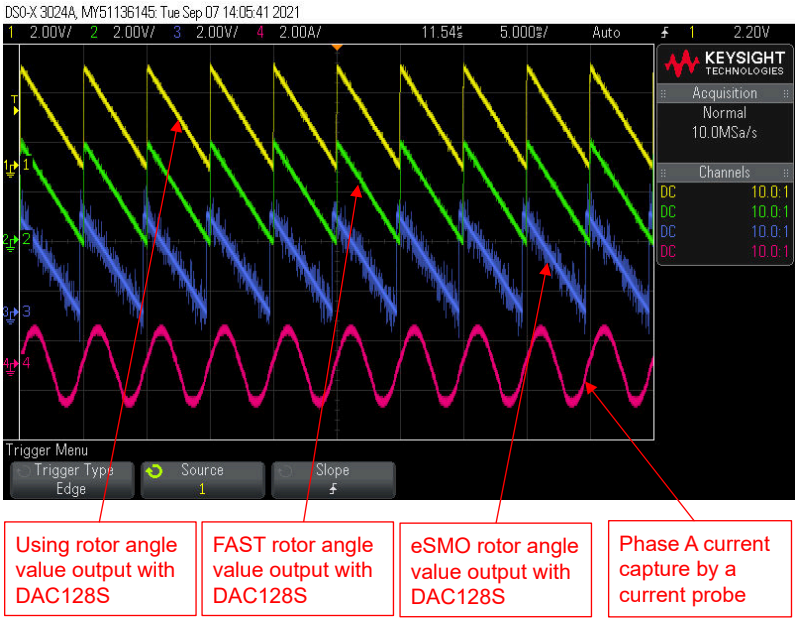


图 3-49. 构建级别 4：采用 FAST 和 eSMO 的转子角度，逆向旋转时的相电流波形

用户可以通过在工程属性中添加预定义名称 *MOTOR1_FAST* 和 *MOTOR1_ENC*，在工程中同时实施 FAST 和编码器估算器，如节 3.3.1 中所述。按照上述操作步骤重新构建、加载和运行工程。

- systemVars.estType 值等于 EST_TYPE_FAST_ENC，这意味着 FAST 和编码器估算器在该工程中处于启用状态。
- motorVars_M1.estimatorMode 等于 ESTIMATOR_MODE_FAST，这意味着 FAST 估算器正在用于无传感器 FOC；等于 ESTIMATOR_MODE_ENC，这意味着编码器估算器正在用于有传感器 FOC。
- 图 1-1 展示了 FAST 和编码器的估算转子角度。通过将 motorVars_M1.speedRef_Hz 设置为正值，电机在 FAST 正向旋转的情况下运行。
- 用户可以将值更改为 ESTIMATOR_MODE_ENC，以选择有传感器 FOC 的编码器估算器。用户还可以更改值以动态地使用估算器进行切换。

当电机正向旋转时，将 DAC128S085EVM 与示波器结合使用可监测来自 FAST 估算器和编码器的电机转子角度、电机的相电流，如图 1-1 所示，方法是将 *motorVars_M1.speedRef_Hz* 设置为正基准值。

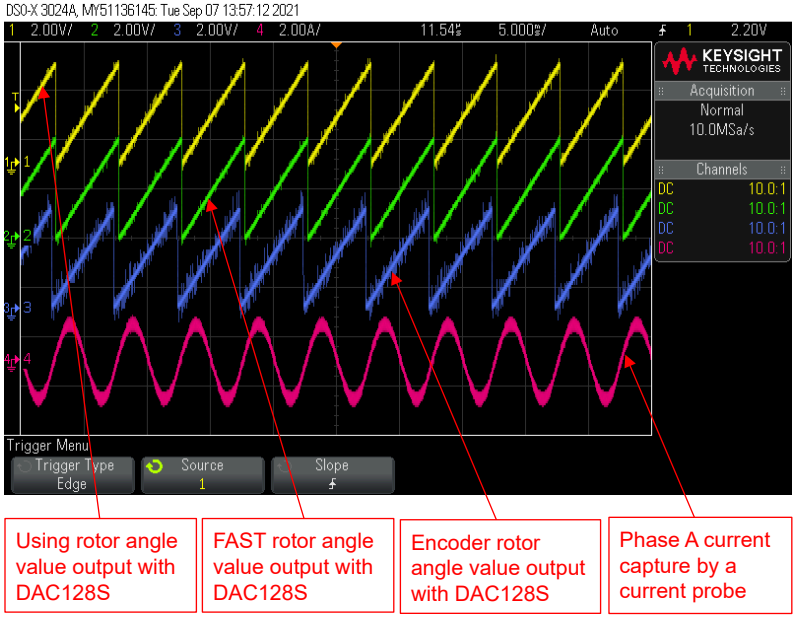


图 3-50. 构建级别 4：采用 FAST 和编码器的转子角度，正向旋转时的相电流波形

用户可以通过在工程属性中添加预定义名称 *MOTOR1_FAST* 和 *MOTOR1_HALL*，在工程中同时实施 FAST 和霍尔传感器估算器，如节 3.3.1 中所述。按照上述操作步骤重新构建、加载和运行工程。

- systemVars.estType 值等于 EST_TYPE_FAST_HALL，这意味着 FAST 和霍尔传感器估算器在该工程中处于启用状态。
- motorVars_M1.estimatorMode 等于 ESTIMATOR_MODE_FAST，这意味着 FAST 估算器正在用于无传感器 FOC；等于 ESTIMATOR_MODE_HALL，这意味着霍尔传感器估算器正在用于有传感器 FOC。
- 图 1-1 中显示了 FAST 和霍尔传感器的估算转子角度。通过将 motorVars_M1.speedRef_Hz 设置为正值，电机在 FAST 正向旋转的情况下运行。
- 图 1-1 中显示了 FAST 和霍尔传感器的估算转子角度。通过将 motorVars_M1.speedRef_Hz 设置为负值，电机在霍尔传感器逆向旋转的情况下运行。
- 用户可以将值更改为 ESTIMATOR_MODE_HALL 以选择有传感器 FOC 的霍尔传感器估算器。用户还可以更改值以动态地使用估算器进行切换。

当电机正向旋转时，将 DAC128S085EVM 与示波器结合使用可监测来自 FAST 估算器和霍尔传感器的电机转子角度、电机的相电流，如图 1-1 所示，方法是将 *motorVars_M1.speedRef_Hz* 设置为正基准值。

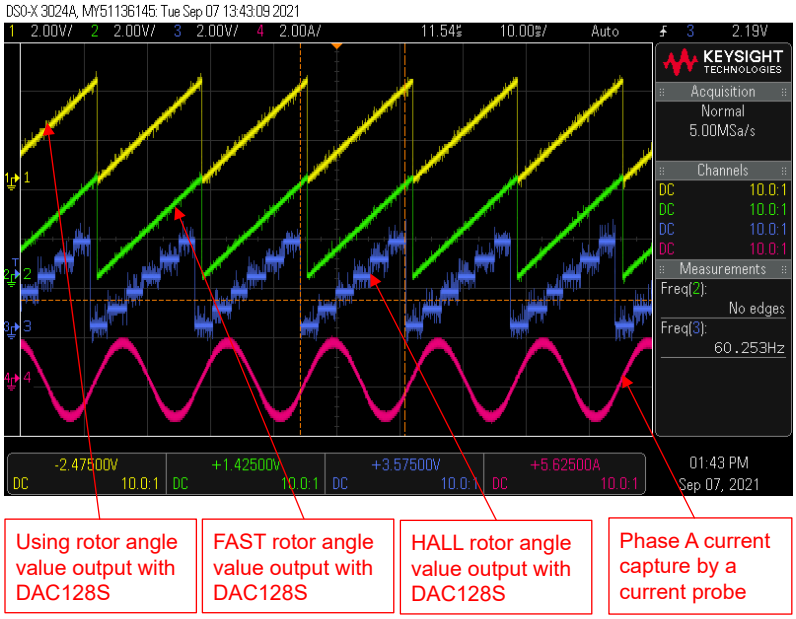


图 3-51. 构建级别 4：采用 FAST 和霍尔传感器的转子角度，正向旋转时的相电流波形

当电机逆向旋转时，将 DAC128S085EVM 与示波器结合使用可监测来自 FAST 估算器和霍尔传感器的电机转子角度、电机的相电流，如图 1-1 所示，方法是将 *motorVars_M1.speedRef_Hz* 设置为负基准值。

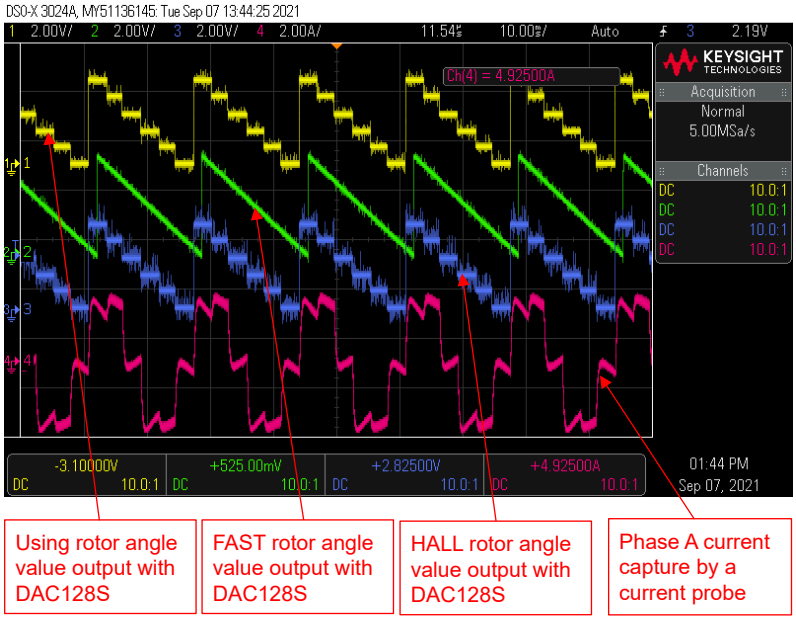


图 3-52. 构建级别 4：采用 FAST 和霍尔传感器的电机转子角度，逆向旋转时的相电流波形

，用户可以通过在工程属性中添加预定义名称 *MOTOR1_ESMO* 和 *MOTOR1_ENC*，在工程中同时实施 eSMO 和编码器估算器，如节 3.3.1 中所述。按照上述操作步骤重新构建、加载和运行工程。

- systemVars.estType 值等于 EST_TYPE_ESMO_ENC，这意味着 eSMO 和编码器估算器在该工程中处于启用状态。
- motorVars_M1.estimatorMode 等于 ESTIMATOR_MODE_ESMO，这意味着 eSMO 估算器正在用于无传感器 FOC；等于 ESTIMATOR_MODE_ENC，这意味着编码器估算器正在用于有传感器 FOC。
- 来自 eSMO 和编码器的估算转子角度如图 1-1 所示。通过将 motorVars_M1.speedRef_Hz 设置为正值，电机在 eSMO 正向旋转的情况下运行。
- 用户可以将值更改为 ESTIMATOR_MODE_ENC，以选择有传感器 FOC 的编码器估算器。用户还可以更改值以动态地使用估算器进行切换。

当电机正向旋转时，将 DAC128S085EVM 与示波器结合使用可监测来自 eSMO 估算器和编码器的电机转子角度、电机的相电流，如图 1-1 所示，方法是将 *motorVars_M1.speedRef_Hz* 设置为正基准值。

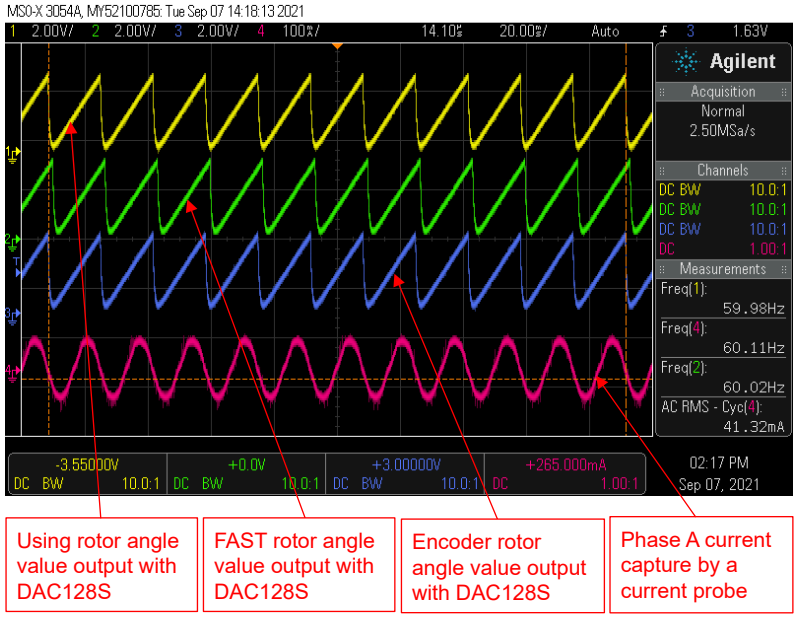


图 3-53. 构建级别 4：采用 eSMO 和编码器的转子角度，正向旋转时的相电流波形

4 构建定制板

4.1 构建新的定制板

本节讨论了用户如何设计自己的应用板来驱动电机，以及如何迁移电机控制 SDK 软件以便与自己的板一起使用。

4.1.1 硬件设置

如果使用定制板，请确保 C2000 微控制器和栅极驱动器的电源正确，并且 JTAG 仿真器可以成功连接。按照以下部分中的说明修改参考代码以与定制电路板兼容，然后运行从构建级别 1 开始的代码，并按照节 3.5 中所示的方法构建级别 4。

4.1.2 将参考代码迁移到定制电路板

要将参考代码迁移到新的 TI 电机驱动器套件或定制电路板，用户需要根据电机驱动器电路在 *user_mtr1.h* 文件中配置硬件参数和电机控制参数，在 *hal.h* 和 *hal.c* 文件中配置相关外设，如以下各节所述。

下面的方框图总结了用于配置电机控制设置和 C2000 MCU 外设的函数调用（图 1-1）。

在该实验工程中，有几个只被调用一次的 HAL 函数，它们与硬件的配置有关。所有这些函数都涉及外设或电机驱动器 IC 的配置。

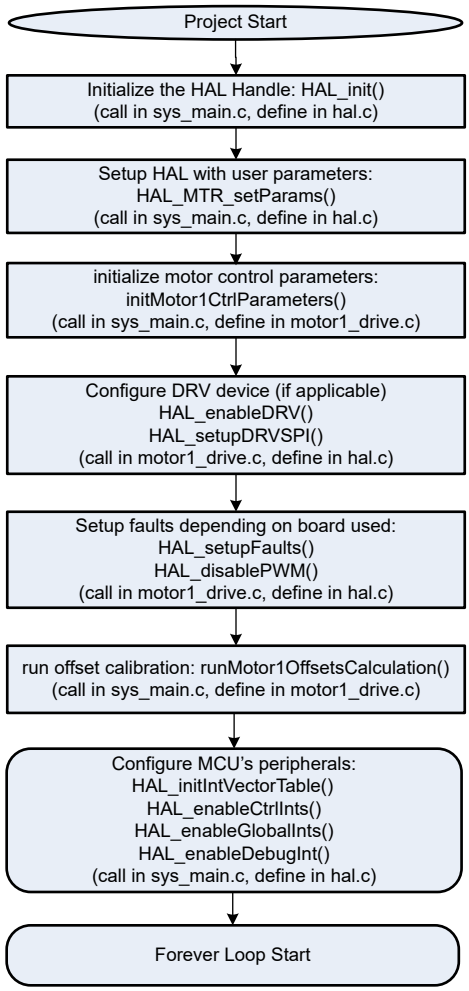


图 4-1. HAL 配置和电机控制设置方框图

4.1.2.1 设置硬件板参数

user_mtr1.h 文件用于存储所有用户参数以进行电机控制。模数转换器输入端的最大相电流和相电压值取决于硬件，应基于电流和电压检测电路以及 ADC 输入端口的调节。相电流传感器和相电压传感器的数量也在 *user_mtr1.h* 文件中定义。这些值取决于硬件。

user_mtr1.h 文件中定义的所有可配置参数都可以使用 `Motor Control Parameters Calculation.xlsx` Excel® 电子表格进行计算。此文件包含在以下文件夹中的工程文件中：`\solutions\universal_motorcontrol_lab\doc`。将标记为***粗体*** 的参数复制到 *user_mtr1.h* 文件中，如以下代码所示。

```
//! \brief Defines the maximum voltage at the AD converter
#define USER_M1_ADC_FULL_SCALE_VOLTAGE_V         (57.52845691f)

//! \brief Defines the analog voltage filter pole location, Hz
#define USER_M1_VOLTAGE_FILTER_POLE_Hz           (680.4839141f)     // 47nF

//! \brief Defines the maximum current at the AD converter
#define USER_M1_ADC_FULL_SCALE_CURRENT_A         (47.14285714f)     // gain=10
```

4.1.2.2 修改电机控制参数

user_mtr1.h 中提供的用于 PMSM/BLDC 电机的参数如以下代码所示。如果使用了 FAST 模块，则可以从电机数据表或通过 FAST 识别过程来识别电机参数。

```
#define USER_MOTOR1_TYPE                    MOTOR_TYPE_PM
#define USER_MOTOR1_NUM_POLE_PAIRS          (4)

#define USER_MOTOR1_Rs_Ohm                  (0.38157931f)
#define USER_MOTOR1_Ls_d_H                  (0.000188295482f)
#define USER_MOTOR1_Ls_q_H                  (0.000188295482f)
#define USER_MOTOR1_RATED_FLUX_VpHz         (0.0396642499f)
#define USER_MOTOR1_MAX_CURRENT_A           (6.0f)
```

4.1.2.3 更改引脚分配

位于 *hal.c* 文件中的 HAL_setupGPIOs() 函数会配置 GPIO 引脚的功能，并根据使用的硬件电机驱动器板/套件设置指定引脚的方向和模式。要修改定制板（当前没有通用实验室代码支持的 TI 电机驱动器 EVM）的代码，或用于不同 C2000MCU 的代码，需要更改这些 GPIO 分配以与电机驱动器板正确对应。请小心为指定的引脚设置正确的焊盘配置，特别是对于将被用作 PWM 输出的 GPIO。EPWM1A GPIO0 引脚的配置示例如下所示。

```
// GPIO0->EPWM1A->M1_UH*
GPIO_setPinConfig(GPIO_0_EPWM1_A);
GPIO_setDirectionMode(0, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(0, GPIO_PIN_TYPE_STD);
```

4.1.2.4 配置 PWM 模块

HAL 模块用于配置 PWM 通道。*hal.h* 文件中定义了用于电机控制器 PWM 输入的 PWM 通道的基地址，并在 *hal.c* 文件中为 PWM 句柄分配了基地址。LAUNCHXL-F280025C 与 BOOSTXL-DRV8323RS 之间 PWM 信号的连接图如图 1-1 所示。

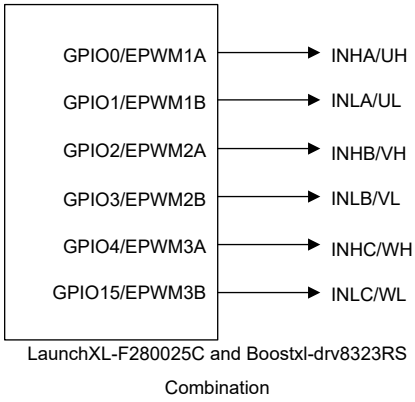


图 4-2. PWM 连接图

用于配置 PWM 信号的代码如下所示，该代码取自位于 solutions\universal_motorcontrol_lab\f28002x\drivers\include 和 \source 文件夹的 *hal.h* 和 *hal.c* 文件。与电机驱动器板相关的变化和与 MCU 相关的变化以**粗体**突出显示。

1. PWM 模块的基地址在 *hal.h* 文件中定义，如下所示。

```
//! \ Motor 1
#define MTR1_PWM_U_BASE          EPWM1_BASE
#define MTR1_PWM_V_BASE          EPWM2_BASE
#define MTR1_PWM_W_BASE          EPWM6_BASE
```

2. 在位于 *hal.c* 文件中的 HAL_setupGpios() 函数内将 GPIO 设置为 PWM 输出。

```
// GPIO0->EPWM1A->M1_UH*
GPIO_setPinConfig(GPIO_0_EPWM1_A);
GPIO_setDirectionMode(0, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(0, GPIO_PIN_TYPE_STD);

// GPIO1->EPWM1B->M1_UL*
GPIO_setPinConfig(GPIO_1_EPWM1_B);
GPIO_setDirectionMode(1, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(1, GPIO_PIN_TYPE_STD);

// GPIO2->EPWM2A->M1_VH*
GPIO_setPinConfig(GPIO_2_EPWM2_A);
GPIO_setDirectionMode(2, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(2, GPIO_PIN_TYPE_STD);

// GPIO3->EPWM2B->M1_VL*
GPIO_setPinConfig(GPIO_3_EPWM2_B);
GPIO_setDirectionMode(3, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(3, GPIO_PIN_TYPE_STD);

// GPIO4->EPWM3A->M1_WH*
GPIO_setPinConfig(GPIO_4_EPWM3_A);
GPIO_setDirectionMode(4, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(4, GPIO_PIN_TYPE_STD);

// GPIO15->EPWM3B->M1_WL*
GPIO_setPinConfig(GPIO_15_EPWM3_B);
GPIO_setDirectionMode(15, GPIO_DIR_MODE_OUT);
GPIO_setPadConfig(15, GPIO_PIN_TYPE_STD);
```

3. 以下代码将 PWM 模块的相应基地址分配给 *hal.c* 文件中 HAL_MTR1_init() 函数的 PWM 句柄。在调整代码以适应新电路板或 C2000 MCU 时，不需要更改以下代码，这些代码仅演示了如何在代码中初始化 PWM 句柄。

```
// initialize PWM handles for Motor 1
obj->pwmHandle[0] = MTR1_PWM_U_BASE;        //!< the PWM handle
obj->pwmHandle[1] = MTR1_PWM_V_BASE;        //!< the PWM handle
obj->pwmHandle[2] = MTR1_PWM_W_BASE;        //!< the PWM handle
```

4. 下面的代码演示了在位于 *hal.c* 文件中的 HAL_setupPWMs() 函数内发生的 PWM 配置。请注意，设置 PWM 频率所需的 PWM 周期 (USER_M1_PWM_TBPRD_NUM)、SOC 事件预分频数 (USER_M1_PWM_TBPRD_NUM) 和死区值 (MTR1_PWM_DBRED_CNT、MTR1_PWM_DBFED_CNT) 在 *hal.h* 文件中定义。可以根据硬件板和控制要求更改这些定义的值。PWM 计数器模式和 PWM 动作限定符输出需要根据硬件板进行设置。

```
void HAL_setupPWMs(HAL_MTR_Handle handle)
{
    HAL_MTR_Obj *obj = (HAL_MTR_Obj *)handle;
    uint16_t cnt;
    uint16_t pwmPeriodCycles = (uint16_t)(USER_M1_PWM_TBPRD_NUM);
    uint16_t numPWMTicksPerISRTick = USER_M1_NUM_PWM_TICKS_PER_ISR_TICK;
    ... ...
    for(cnt=0; cnt<3; cnt++)
    {
        // setup the Time-Base Control Register (TBCTL)
        EPWM_setTimeBaseCounterMode(obj->pwmHandle[cnt], EPWM_COUNTER_MODE_UP_DOWN);
        ... ...
        // setup the Action-Qualifier Output A Register (AQCTLA)
        EPWM_setActionQualifierAction(obj->pwmHandle[cnt], EPWM_AQ_OUTPUT_A,
            EPWM_AQ_OUTPUT_HIGH, EPWM_AQ_OUTPUT_ON_TIMEBASE_UP_CMPA);
        EPWM_setActionQualifierAction(obj->pwmHandle[cnt], EPWM_AQ_OUTPUT_A,
            EPWM_AQ_OUTPUT_HIGH, EPWM_AQ_OUTPUT_ON_TIMEBASE_PERIOD);
        EPWM_setActionQualifierAction(obj->pwmHandle[cnt], EPWM_AQ_OUTPUT_A,
            EPWM_AQ_OUTPUT_LOW, EPWM_AQ_OUTPUT_ON_TIMEBASE_DOWN_CMPA);
        EPWM_setActionQualifierAction(obj->pwmHandle[cnt], EPWM_AQ_OUTPUT_A,
            EPWM_AQ_OUTPUT_LOW, EPWM_AQ_OUTPUT_ON_TIMEBASE_ZERO);
        ... ...
        // setup the Dead-Band Generator Control Register (DBCTL)
        EPWM_setDeadBandDelayMode(obj->pwmHandle[cnt], EPWM_DB_RED, true);
        EPWM_setDeadBandDelayMode(obj->pwmHandle[cnt], EPWM_DB_FED, true);

        // select EPWMA as the input to the dead band generator
        EPWM_setRisingEdgeDeadBandDelayInput(obj->pwmHandle[cnt], EPWM_DB_INPUT_EPWMA);

        // configure the right polarity for active high complementary config.
        EPWM_setDeadBandDelayPolarity(obj->pwmHandle[cnt], EPWM_DB_RED,
EPWM_DB_POLARITY_ACTIVE_HIGH);
        EPWM_setDeadBandDelayPolarity(obj->pwmHandle[cnt], EPWM_DB_FED,
EPWM_DB_POLARITY_ACTIVE_LOW);

        // setup the Dead-Band Rising Edge Delay Register (DBRED)
        EPWM_setRisingEdgeDelayCount(obj->pwmHandle[cnt], MTR1_PWM_DBRED_CNT);

        // setup the Dead-Band Falling Edge Delay Register (DBFED)
        EPWM_setFallingEdgeDelayCount(obj->pwmHandle[cnt], MTR1_PWM_DBFED_CNT);
        ... ...
    }
    ... ...
    // setup the Event Trigger Selection Register (ETSEL)
    EPWM_setInterruptSource(obj->pwmHandle[0], EPWM_INT_TBCTR_ZERO);
    EPWM_enableInterrupt(obj->pwmHandle[0]);
    EPWM_setADCTriggerSource(obj->pwmHandle[0], EPWM_SOC_A, EPWM_SOC_TBCTR_D_CMPC);
    EPWM_enableADCTrigger(obj->pwmHandle[0], EPWM_SOC_A);
    ... ...

    return;
}  // end of HAL_setupPWMs() function
```

5. 以下代码位于 *hal.h* 文件中，定义了 ADC 转换启动触发源。此 ePWM SOC 触发器必须对应于第 4 步所示代码中启用的同一 ePWM SOC，以及与 pwmHandle[0] 相关联的同一 ePWM。这是因为在第 4 步中使用了这些引脚来设置触发源。在这种情况下，EPWM1 A 用作 ADC 的 SOC。

```
// Three-shunt
#define MTR1_ADC_TRIGGER_SOC      ADC_TRIGGER_EPWM1_SOCA // EPWM1_SOCA
```

4.1.2.5 配置 ADC 模块

与前面的 PWM 部分类似，对于通用电机控制实验不支持的定制电路板、TI 电机控制套件或 C2000 MCU 也可以更改 ADC 连接。HAL 模块会配置 ADC 通道，使其与电机驱动器板正确对应。例如，LAUNCHXL-F280025C 和 BOOSTXL-DRV8323RS 组合的连接图如图 1-1 所示。以下步骤介绍了 ADC 模块配置，电路板特定的潜在更改以粗体突出显示。步骤 1 和步骤 2 对于配置新的电机驱动器板或不同的 C2000 MCU 以运行电机至关重要。

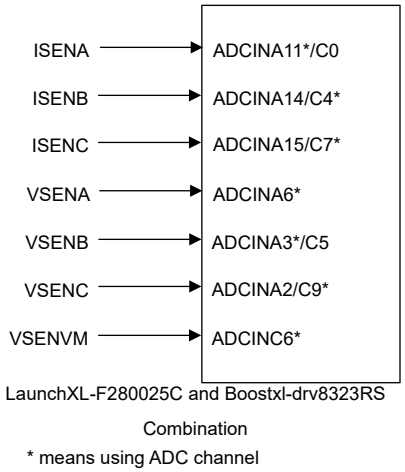


图 4-3. ADC 连接图

以下代码取自位于 solutions\universal_motorcontrol_lab\f28002x\drivers\include and \source 文件夹中的 *hal.h* 和 *hal.c* 文件。

1. 以下代码演示了 *hal.h* 文件中 ADC 模块的基地址、分配的通道和 SOC 的定义。请注意，对于 SOC 编号，多个 ADC 可以与同一 SOC 编号相关联，只要它们属于不同的 ADC 模块（在以下用例中，是模块 A 和模块 C）。尽量尝试对所有电流和所有电压进行采样时使其尽可能靠近，因此在配置 SOC 编号时应注意这一点。

```
#define MTR1_IU_ADC_BASE          ADCA_BASE                 // ADCA-A11*/C0
#define MTR1_IV_ADC_BASE          ADCC_BASE                 // ADCC-A14/C4*
#define MTR1_IW_ADC_BASE          ADCC_BASE                 // ADCC-A15/C7*
#define MTR1_VU_ADC_BASE          ADCA_BASE                 // ADCA-A6*
#define MTR1_VV_ADC_BASE          ADCA_BASE                 // ADCC-A3*/C5
#define MTR1_VW_ADC_BASE          ADCC_BASE                 // ADCA-A2/C9*
#define MTR1_VDC_ADC_BASE         ADCC_BASE                 // ADCC-C6*
#define MTR1_POT_ADC_BASE         ADCA_BASE                 // ADCA-A12*/C1

#define MTR1_IU_ADCRES_BASE       ADCARESULT_BASE           // ADCA-A11*/C0
#define MTR1_IV_ADCRES_BASE       ADCCRESULT_BASE           // ADCC-A14/C4*
#define MTR1_IW_ADCRES_BASE       ADCCRESULT_BASE           // ADCC-A15/C7*
#define MTR1_VU_ADCRES_BASE       ADCARESULT_BASE           // ADCA-A6*
#define MTR1_VV_ADCRES_BASE       ADCARESULT_BASE           // ADCC-A3*/C5
#define MTR1_VW_ADCRES_BASE       ADCCRESULT_BASE           // ADCA-A2/C9*
#define MTR1_VDC_ADCRES_BASE      ADCCRESULT_BASE           // ADCC-C6*
#define MTR1_POT_ADCRES_BASE      ADCARESULT_BASE           // ADCA-A12*/C1

#define MTR1_IU_ADC_CH_NUM        ADC_CH_ADCIN11            // ADCA-A11*/C0
#define MTR1_IV_ADC_CH_NUM        ADC_CH_ADCIN4             // ADCC-A14/C4*
#define MTR1_IW_ADC_CH_NUM        ADC_CH_ADCIN7             // ADCC-A15/C7*
#define MTR1_VU_ADC_CH_NUM        ADC_CH_ADCIN6             // ADCA-A6*
#define MTR1_VV_ADC_CH_NUM        ADC_CH_ADCIN3             // ADCC-A3*/C5
#define MTR1_VW_ADC_CH_NUM        ADC_CH_ADCIN9             // ADCA-A2/C9*
#define MTR1_VDC_ADC_CH_NUM       ADC_CH_ADCIN6             // ADCC-C6*
#define MTR1_POT_ADC_CH_NUM       ADC_CH_ADCIN12            // ADCA-A12*/C1
```

```
#define MTR1_IU_ADC_SOC_NUM      ADC_SOC_NUMBER1          // ADCA-A11*/C10-SOC1-PPB1
#define MTR1_IV_ADC_SOC_NUM      ADC_SOC_NUMBER1          // ADCC-A14/C4* -SOC1-PPB1
#define MTR1_IW_ADC_SOC_NUM      ADC_SOC_NUMBER2          // ADCC-A15/C7* -SOC2-PPB2
#define MTR1_VU_ADC_SOC_NUM      ADC_SOC_NUMBER4          // ADCA-A6*     -SOC4
#define MTR1_VV_ADC_SOC_NUM      ADC_SOC_NUMBER5          // ADCC-A3*/C5  -SOC5
#define MTR1_VW_ADC_SOC_NUM      ADC_SOC_NUMBER5          // ADCA-A2/C9*  -SOC5
#define MTR1_VDC_ADC_SOC_NUM     ADC_SOC_NUMBER6          // ADCC-C6*     -SOC6
#define MTR1_POT_ADC_SOC_NUM     ADC_SOC_NUMBER6          // ADCA-A12*/C1 -SOC6

#define MTR1_IU_ADC_PPB_NUM      ADC_PPB_NUMBER1          // ADCA-A11*/C10-SOC1-PPB1
#define MTR1_IV_ADC_PPB_NUM      ADC_PPB_NUMBER1          // ADCC-A14/C4* -SOC1-PPB1
#define MTR1_IW_ADC_PPB_NUM      ADC_PPB_NUMBER2          // ADCC-A15/C7*- SOC2-PPB2
```

2. 下面的代码展示了 *hal.h* 文件中 ISR 中断源的定义。

```
// interrupt
#define MTR1_PWM_INT_BASE        MTR1_PWM_U_BASE          // EPWM1

#define MTR1_ADC_INT_BASE        ADCA_BASE                // ADCA-A14 -SOC4
#define MTR1_ADC_INT_NUM         ADC_INT_NUMBER1          // ADCA_INT1-SOC4
#define MTR1_ADC_INT_SOC         ADC_SOC_NUMBER4          // ADCA_INT1-SOC4

#define MTR1_PIE_INT_NUM         INT_ADCA1                // ADCA_INT1-SOC4
#define MTR1_INT_ACK_GROUP       INTERRUPT_ACK_GROUP1     // ADCA_INT1-CPU_INT1
```

3. ADC 模块在位于 *hal.c* 文件中的 HAL_setupADCs() 函数中设置。下面显示了用于对电位器进行采样的 ADC 设置示例。如果用户希望使用额外的 ADC 通道来对额外的信号进行采样，则需要使用与以下代码类似的方式设置 ADC 通道。

```
// POT_M1 ADC_setupSOC(MTR1_POT_ADC_BASE, MTR1_POT_ADC_SOC_NUM, MTR1_ADC_TRIGGER_SOC,
MTR1_POT_ADC_CH_NUM, MTR1_ADC_V_SAMPLEWINDOW);
```

4. ADC 结果在 *hal.h* 文件中的 HAL_readMtr1ADCData() 函数中读取。POT 输入读取 ADC 结果的示例如下所示。如果用户添加了额外的 ADC 通道以进行额外的信号采样，则需要添加与下面所示内容类似的代码，以读取所添加 ADC 通道的结果。这还需要修改 HAL_ADCData_t 结构以存储数据。

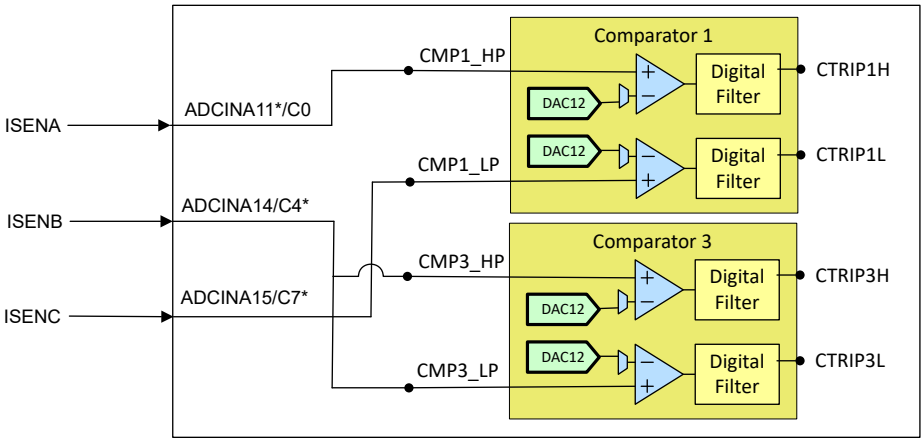
```
// read POT adc value
pADCData->potAdc = ADC_readResult(MTR1_POT_ADCRES_BASE, MTR1_POT_ADC_SOC_NUM);
```

4.1.2.6 配置 CMPSS 模块

CMPSS 模块用于对相电流进行过流监测。可使用 CMPSS DAC 设置阈值，如果电流检测放大器的输出超过该阈值，则 CMPSS 输出会发生跳变。

如果使用定制电机驱动器板，或者将代码迁移到当前通用电机控制实验不支持的 C2000 MCU 或 TI 电机驱动器 EVM，则需要根据电机驱动器和 C2000 MCU 连接在 *hal.h* 文件中正确修改 ADC 引脚和 CMPSS 模块之间的连接。有关 CMPSS 模块内部连接的更多详细信息，请参阅 *TMS320F28002x 实时微控制器技术参考手册（修订版 A）* 中的*模拟引脚和内部连接* 表。

HAL 模块根据所使用的电机驱动器板配置 CMPSS 模块。例如，LAUNCHXL-F280025C 和 BOOSTXL-DRV8323RS 之间的连接图如图 1-1 所示。以下步骤介绍了 CMPSS 模块的配置（特定于电路板或特定于 MCU 的更改以**粗体**显示）。



LaunchXL-F280025C and Boostxl-drv8323RS Combination

* means using ADC channel

图 4-4. CMPSS 连接图

1. 下面的代码显示了 3 个相电流 CMPSS 模块的基地址定义。此代码位于 *hal.h* 文件中。

```
#define MTR1_CMPSS_U_BASE       CMPSS1_BASE
#define MTR1_CMPSS_V_BASE       CMPSS3_BASE
#define MTR1_CMPSS_W_BASE       CMPSS1_BASE
```

2. 以下代码显示了用于将所需 ADC 输入分配给正确 CMPSS 模块的定义。此代码位于 *hal.h* 文件中。每个 CMPSS 比较器都有一个高电平比较器和一个低电平比较器，因此信号必须适当地多路复用到所需比较器的所需输入。有关这些连接的更多信息，请参阅正在使用的微控制器技术参考手册中的“模拟引脚和内部连接”表。注意：对于 LAUNCHXL F280025C，电机驱动器电流检测引脚可用的 CMPSS 模块是 CMPSS3 和 CMPSS1，因此 U 相电流和 W 相电流都需要共享 CMPSS1。这会导致 U 相电流仅在正过流时跳闸 CMPSS，而 W 相电流仅在负过流时跳闸 CMPSS。由于 V 相有一个专用的 CMPSS，因此将检测该相位上的过流是否存在正电流和负电流。如果修改代码以支持 C2000 MCU 器件，该器件具有用于每个相电流输入的单独 CMPSS 模块（例如，在 F280049C LaunchPad 中），则通过将同一相电流 ADC 输入多路复用到相应 CMPSS 模块的高输入和低输入，可以将代码配置为在每个相位上同时出现高过流和低过流。

```
// CMPSS
// For single phase current sensing, DRV8323RH and RS only
#define MTR1_IDC_CMPHP_SEL      ASYSCTL_CMPHPMUX_SELECT_3    // CMPSS3-A14/C4*
#define MTR1_IDC_CMPLP_SEL      ASYSCTL_CMPLPMUX_SELECT_3    // CMPSS3-A14/C4*

#define MTR1_IDC_CMPHP_MUX      4                            // CMPSS3-A14/C4*
#define MTR1_IDC_CMPLP_MUX      4                            // CMPSS3-A14/C4*

// For three-phase current sensing
#define MTR1_IU_CMPHP_SEL       ASYSCTL_CMPHPMUX_SELECT_1    // CMPSS1-A11
#define MTR1_IU_CMPLP_SEL       ASYSCTL_CMPLPMUX_SELECT_1    // CMPSS1-A11, N/A

#define MTR1_IV_CMPHP_SEL       ASYSCTL_CMPHPMUX_SELECT_3    // CMPSS3-C4
#define MTR1_IV_CMPLP_SEL       ASYSCTL_CMPLPMUX_SELECT_3    // CMPSS3-C4

#define MTR1_IW_CMPHP_SEL       ASYSCTL_CMPHPMUX_SELECT_1    // CMPSS1-C7, N/A
#define MTR1_IW_CMPLP_SEL       ASYSCTL_CMPLPMUX_SELECT_1    // CMPSS1-C7

#define MTR1_IU_CMPHP_MUX       1                            // CMPSS1-A11
#define MTR1_IU_CMPLP_MUX       1                            // CMPSS1-A11

#define MTR1_IV_CMPHP_MUX       4                            // CMPSS3-C4
#define MTR1_IV_CMPLP_MUX       4                            // CMPSS3-C4

#define MTR1_IW_CMPHP_MUX       3                            // CMPSS1-C7
#define MTR1_IW_CMPLP_MUX       3                            // CMPSS1-C7
```

3. 以下代码演示了 CMPSS 模块的设置，该设置在 *hal.c* 文件的 HAL_setupCMPSSs () 函数中进行。在此示例中，CMPSS1_HP 链接到 U 相电流，CMPSS1_LP 链接到 W 相电流，因此将 CMPSS1 配置两次以简化代

码。在修改通用电机控制实验时，以下代码可能不需要修改，但务必确保正确配置上一步中的代码，以便 HAL_setupCMPSSs() 函数正确设置 CMPSS 模块。

```
void HAL_setupCMPSSs(HAL_MTR_Handle handle)
{
    HAL_MTR_Obj *obj = (HAL_MTR_Obj *)handle;
... ...
    uint16_t cmpsaDACH = MTR1_CMPSS_DACH_VALUE;
    uint16_t cmpsaDACL = MTR1_CMPSS_DACL_VALUE;
... ...
    ASysCtl_selectCMPHPMux(MTR1_IU_CMPHP_SEL, MTR1_IU_CMPHP_MUX);
    ASysCtl_selectCMPLPMux(MTR1_IU_CMPLP_SEL, MTR1_IU_CMPLP_MUX);

    ASysCtl_selectCMPHPMux(MTR1_IV_CMPHP_SEL, MTR1_IV_CMPHP_MUX);
    ASysCtl_selectCMPLPMux(MTR1_IV_CMPLP_SEL, MTR1_IV_CMPLP_MUX);

    ASysCtl_selectCMPHPMux(MTR1_IW_CMPHP_SEL, MTR1_IW_CMPHP_MUX);
    ASysCtl_selectCMPLPMux(MTR1_IW_CMPLP_SEL, MTR1_IW_CMPLP_MUX);

    for(cnt=0; cnt<3; cnt++)
    {
        // Enable CMPSS and configure the negative input signal to come from the DAC
        CMPSS_enableModule(obj->cmpssHandle[cnt]);

        // NEG signal from DAC for COMP-H
        CMPSS_configHighComparator(obj->cmpssHandle[cnt], CMPSS_INSRC_DAC);

        // NEG signal from DAC for COMP-L
        CMPSS_configLowComparator(obj->cmpssHandle[cnt], CMPSS_INSRC_DAC);

        // Configure the output signals. Both CTRIPH and CTRIPOUTH will be fed by
        // the asynchronous comparator output.
        // Dig filter output ==> CTRIPH, Dig filter output ==> CTRIPOUTH
        CMPSS_configOutputsHigh(obj->cmpssHandle[cnt],
                                CMPSS_TRIP_FILTER |
                                CMPSS_TRIPOUT_FILTER);

        // Dig filter output ==> CTRIPL, Dig filter output ==> CTRIPOUTL
        CMPSS_configOutputsLow(obj->cmpssHandle[cnt],
                               CMPSS_TRIP_FILTER |
                               CMPSS_TRIPOUT_FILTER |
                               CMPSS_INV_INVERTED);

        // Configure digital filter. For this example, the maxiumum values will be
        // used for the clock prescale, sample window size, and threshold.
        CMPSS_configFilterHigh(obj->cmpssHandle[cnt], 32, 32, 30);
        CMPSS_initFilterHigh(obj->cmpssHandle[cnt]);

        // Initialize the filter logic and start filtering
        CMPSS_configFilterLow(obj->cmpssHandle[cnt], 32, 32, 30);
        CMPSS_initFilterLow(obj->cmpssHandle[cnt]);

        // Set up COMPHYSCTL register
        // COMP hysteresis set to 2x typical value
        CMPSS_setHysteresis(obj->cmpssHandle[cnt], 1);

        // Use VDDA as the reference for the DAC and set DAC value to midpoint for
        // arbitrary reference
        CMPSS_configDAC(obj->cmpssHandle[cnt],
                    CMPSS_DACREF_VDDA | CMPSS_DACVAL_SYSCLK | CMPSS_DACSRC_SHDW);

        // Set DAC-H to allowed MAX +ve current
        CMPSS_setDACValueHigh(obj->cmpssHandle[cnt], cmpsaDACH);

        // Set DAC-L to allowed MAX -ve current
        CMPSS_setDACValueLow(obj->cmpssHandle[cnt], cmpsaDACL);

        // Clear any high comparator digital filter output latch
        CMPSS_clearFilterLatchHigh(obj->cmpssHandle[cnt]);

        // Clear any low comparator digital filter output latch
        CMPSS_clearFilterLatchLow(obj->cmpssHandle[cnt]);
    }
... ...
    return;
}
```

4.1.2.7 配置故障保护函数

当发生某些故障（例如电机驱动器跳闸引起的故障引脚或在 CMSS 模块上检测到过流事件）时，将采取措施来停止 MCU 的输出 PWM。为实现此目的，对 ePWM 多路复用器和 ePWM 跳闸区域进行配置以采取适当的操作。本节将介绍实现此目的的代码。在使用自定义电机驱动器板、使用通用电机控制实验尚不支持的 TI 电机驱动器 EVM 或为不同的 C2000 MCU 配置代码时，必须对此代码进行适当修改。可能需要的代码更改用**粗体**突出显示。有关 X-BAR 的更多详细信息，请参阅 TMS320F28002x 实时微控制器技术参考手册中的 ePWM X-BAR 多路复用器配置表和输出 X-BAR 多路复用器配置表。

1. 以下代码定义了电机驱动器故障 GPIO 输入。此代码位于 *hal.h* 文件中。

```
//! \brief Defines the gpio for the nFAULT of Power Module
#define MTR1_PM_nFAULT_GPIO     34
```

2. 下面的定义用于设置 X-BAR，以将来自 CMPSS 模块和故障 GPIO 的信号链接到 ePWM 跳闸区子模块。此代码位于 *hal.h* 文件中。请注意，在下面的代码中，相位 U 电流 X-BAR ePWM 多路复用器配置为 CTRIPL，而相位 W 电流 X-BAR ePWM 多路复用器配置为 CTRIPH，但相位 V 电流 X-BAR ePWM 多路复用器配置为 CTRIPH_OR_L。这是因为 W 相和 U 相电流输入共用同一个 CMPSS 模块（模块 1），而 V 相电流输入有其自己的专用 CMPSS 模块（模块 3），从而使其可以配置为在同一相电流输入上输出高电平或低电平。如果使用允许电流输入具有专用 CMPSS 模块的 C2000 MCU 或电机驱动板，则需要将 U 相和 W 相电流的 ePWM 多路复用器配置更改为 CTRIPH_OR_L，类似于 V 相的当前配置方式。

```
#define MTR1_XBAR_TRIP_ADDRL    XBAR_O_TRIP7MUX0TO15CFG
#define MTR1_XBAR_TRIP_ADDRH    XBAR_O_TRIP7MUX16TO31CFG

#define MTR1_IDC_XBAR_EPWM_MUX XBAR_EPWM_MUX05_CMPSS3_CTRIPL   // CMPSS3-LP, single shunt only
#define MTR1_IDC_XBAR_MUX       XBAR_MUX05 // CMPSS3-LP, single shunt only

#define MTR1_IU_XBAR_EPWM_MUX XBAR_EPWM_MUX00_CMPSS1_CTRIPH // CMPSS1-HP
#define MTR1_IV_XBAR_EPWM_MUX XBAR_EPWM_MUX04_CMPSS3_CTRIPH_OR_L // CMPSS3-HP&LP
#define MTR1_IW_XBAR_EPWM_MUX XBAR_EPWM_MUX01_CMPSS1_CTRIPL       // CMPSS1-LP

#define MTR1_IU_XBAR_MUX        XBAR_MUX00          // CMPSS1-HP
#define MTR1_IV_XBAR_MUX        XBAR_MUX04          // CMPSS3-HP&LP
#define MTR1_IW_XBAR_MUX        XBAR_MUX01 // CMPSS1-LP

#define MTR1_XBAR_INPUT1 XBAR_INPUT1
#define MTR1_TZ_OSHT1 EPWM_TZ_SIGNAL_OSHT1
#define MTR1_XBAR_TRIP XBAR_TRIP7
#define MTR1_DCTRIPIN EPWM_DC_COMBINATIONAL_TRIPIN7
```

3. 以下代码为相电流和电机故障引脚配置 ePWM X-BAR 和跳闸信号。此操作在 *hal.c* 文件中的 HAL_setupMtrFaults() 函数内完成。在修改通用电机控制实验时，可能不需要修改以下代码，但务必确保正确配置上一步中的代码，以便设置正确的多路复用器和跳闸值。

```
void HAL_setupMtrFaults(HAL_MTR_Handle handle)
{
    ... ...
    // Configure TRIP7 to be CTRIP5H and CTRIP5L using the ePWM X-BAR
    XBAR_setEPWMMuxConfig(MTR1_XBAR_TRIP, MTR1_IU_XBAR_EPWM_MUX);

    // Configure TRIP7 to be CTRIP1H and CTRIP1L using the ePWM X-BAR
    XBAR_setEPWMMuxConfig(MTR1_XBAR_TRIP, MTR1_IV_XBAR_EPWM_MUX);

    // Configure TRIP7 to be CTRIP3H and CTRIP3L using the ePWM X-BAR
    XBAR_setEPWMMuxConfig(MTR1_XBAR_TRIP, MTR1_IW_XBAR_EPWM_MUX);

    // Disable all the mux first
    XBAR_disableEPWMMux(MTR1_XBAR_TRIP, 0xFFFF);

    // Enable Mux 0 OR Mux 4 to generate TRIP
    XBAR_enableEPWMMux(MTR1_XBAR_TRIP, MTR1_IU_XBAR_MUX | MTR1_IV_XBAR_MUX | MTR1_IW_XBAR_MUX);

    ... ...

    // configure the input x bar for TZ2 to GPIO, where Over Current is connected
    XBAR_setInputPin(INPUTXBAR_BASE, MTR1_XBAR_INPUT1, MTR1_PM_nFAULT_GPIO);
    XBAR_lockInput(INPUTXBAR_BASE, MTR1_XBAR_INPUT1);
```

```
    for(cnt=0; cnt<3; cnt++)
    {
        EPWM_enableTripZoneSignals(obj->pwmHandle[cnt],
                                   EPWM_TZ_SIGNAL_CBC6);

        //enable DC TRIP combinational input
        EPWM_enableDigitalCompareTripCombinationInput(obj->pwmHandle[cnt],
                                              MTR1_DCTRIPIN, EPWM_DC_TYPE_DCAH);

        EPWM_enableDigitalCompareTripCombinationInput(obj->pwmHandle[cnt],
                                              MTR1_DCTRIPIN, EPWM_DC_TYPE_DCBH);

        // Trigger event when DCAH is High
        EPWM_setTripZoneDigitalCompareEventCondition(obj->pwmHandle[cnt],
                                                     EPWM_TZ_DC_OUTPUT_A1,
                                                     EPWM_TZ_EVENT_DCXH_HIGH);

        // Trigger event when DCBH is High
        EPWM_setTripZoneDigitalCompareEventCondition(obj->pwmHandle[cnt],
                                                     EPWM_TZ_DC_OUTPUT_B1,
                                                     EPWM_TZ_EVENT_DCXL_HIGH);

        // Configure the DCA path to be un-filtered and asynchronous
        EPWM_setDigitalCompareEventSource(obj->pwmHandle[cnt],
                                          EPWM_DC_MODULE_A,
                                          EPWM_DC_EVENT_1,
                                          EPWM_DC_EVENT_SOURCE_FILT_SIGNAL);

        // Configure the DCB path to be un-filtered and asynchronous
        EPWM_setDigitalCompareEventSource(obj->pwmHandle[cnt],
                                          EPWM_DC_MODULE_B,
                                          EPWM_DC_EVENT_1,
                                          EPWM_DC_EVENT_SOURCE_FILT_SIGNAL);

        EPWM_setDigitalCompareEventSyncMode(obj->pwmHandle[cnt],
                                            EPWM_DC_MODULE_A,
                                            EPWM_DC_EVENT_1,
                                            EPWM_DC_EVENT_INPUT_NOT_SYNCED);

        EPWM_setDigitalCompareEventSyncMode(obj->pwmHandle[cnt],
                                            EPWM_DC_MODULE_B,
                                            EPWM_DC_EVENT_1,
                                            EPWM_DC_EVENT_INPUT_NOT_SYNCED);

        // Enable DCA as OST
        EPWM_enableTripZoneSignals(obj->pwmHandle[cnt], EPWM_TZ_SIGNAL_DCAEVT1);

        // Enable DCB as OST
        EPWM_enableTripZoneSignals(obj->pwmHandle[cnt], EPWM_TZ_SIGNAL_DCBEVT1);

        // What do we want the OST/CBC events to do?
        // TZA events can force EPWMxA
        // TZB events can force EPWMxB
        EPWM_setTripZoneAction(obj->pwmHandle[cnt],
                               EPWM_TZ_ACTION_EVENT_TZA,
                               EPWM_TZ_ACTION_LOW);

        EPWM_setTripZoneAction(obj->pwmHandle[cnt],
                               EPWM_TZ_ACTION_EVENT_TZB,
                               EPWM_TZ_ACTION_LOW);
    }
    ... ...
    return;
} // end of HAL_setupMtrFaults() function
```

4.1.3 向电机控制工程中添加附加功能

示例实验工程提供了多个接口功能来启动/停止电机以及使用按钮、电位器或 SCI 或 CAN 等通信总线设置基准速度。

4.1.3.1 添加按钮功能

在按下按钮时，读取按钮以使电机运行、停止或只是更改全局变量的状态通常很有用。例如，用户可将 GPIO23 连接到用以启动/停止电机的按钮。为此，请在工程构建属性中启用预定义符号 CMD_SWITCH_EN，如图 1-1 所示。GPIO 状态将分配给 motorVars_M1.flagEnableRunAndIdentify。详细步骤如下。

1. 在 *hal.h* 文件中定义 GPIO 编号

```
#define MTR1_CMD_SWITCH_GPIO    23
```

2. 在 *hal.c* 文件的 HAL_setupGpios() 函数中配置 GPIO，以允许该引脚作为输入。

```
// GPIO23->Command Switch Button
GPIO_setPinConfig(GPIO_23_GPIO23);
GPIO_setDirectionMode(23, GPIO_DIR_MODE_IN);
GPIO_setPadConfig(23, GPIO_PIN_TYPE_PULLUP);
GPIO_setQualificationMode(23, GPIO_QUAL_3SAMPLE);
GPIO_setQualificationPeriod(23, 4);
```

3. 在 *motor_common.c* 文件中的 updateCmdSwitch() 中使用数字过滤器读取 GPIO，如下所示。

```
if(GPIO_readPin(MTR1_CMD_SWITCH_GPIO) == 0)
{
    objMtr->cmdSwtich.lowTimeCnt++;

    if(objMtr->cmdSwtich.lowTimeCnt > objMtr->cmdSwtich.delayTimeSet)
    {
        objMtr->cmdSwtich.flagCmdRun = true;
    }

    if(objMtr->cmdSwtich.highTimeCnt > 0)
    {
        objMtr->cmdSwtich.highTimeCnt--;
    }
}
else
{
    objMtr->cmdSwtich.highTimeCnt++;

    if(objMtr->cmdSwtich.highTimeCnt > objMtr->cmdSwtich.delayTimeSet)
    {
        objMtr->cmdSwtich.flagCmdRun = false;
    }

    if(objMtr->cmdSwtich.lowTimeCnt > 0)
    {
        objMtr->cmdSwtich.lowTimeCnt--;
    }
}
```

4. 将状态关联到 *motor_common.c* 文件的 updateCmdSwitch() 中的电机启动/停止变量。

```
if((objMtr->cmdSwtich.flagEnablCmd == true) && (objMtr->faultMtrUse.all == 0))
{
    objMtr->flagEnableRunAndIdentify = objMtr->cmdSwtich.flagCmdRun;
}
```

4.1.3.2 添加电位器读取功能

通常使用电位器来允许电机运行、停止和设置基准速度。例如，用户可将 ADCA12 连接到电位器。为此，请在工程构建属性中启用预定义符号 CMD_POT_EN，如图 1-1 所示。读取 ADC 的结果将转换为速度值，并分配给变量 motorVars_M1.flagEnableRunAndIdentify，用于启动/停止电机和 motorVars_M1.speedRef_Hz 设置基准速度。详细步骤如下。

1. 在 *hal.h* 文件中定义用于连接到电位计的 ADC 通道，如下所示：

```
#define MTR1_POT_ADC_BASE       ADCA_BASE
#define MTR1_POT_ADCRES_BASE    ADCARESULT_BASE
#define MTR1_POT_ADC_CH_NUM     ADC_CH_ADCIN12
#define MTR1_POT_ADC_SOC_NUM    ADC_SOC_NUMBER6
```

2. 在 *hal.c* 文件的函数 HAL_setupAdcs() 中配置 ADC 通道，如下所示：

```
// POT_M1
ADC_setupSOC(MTR1_POT_ADC_BASE, MTR1_POT_ADC_SOC_NUM, MTR1_ADC_TIGGER_SOC,
             MTR1_POT_ADC_CH_NUM, MTR1_ADC_V_SAMPLEWINDOW);
```

3. 读取 ADC 结果寄存器并按如下方式对 *hal.h* 文件的 HAL_readMtr1ADCData() 中的值进行缩放：

```
// read POT adc value
pADCData->potAdc = ADC_readResult(MTR1_POT_ADCRES_BASE, MTR1_POT_ADC_SOC_NUM);
```

4. 将 ADC 值转换为 *motor_common.c* 文件的 updateExtCmdPotFreq() 中的速度值。

4.1.3.3 添加 CAN 功能

可以将 CAN 功能添加到实验工程中，为用户提供用于发送启动/停止命令并获取运行状态反馈的通信总线。要利用此功能，请在工程构建属性中启用预定义符号 CMD_CAN_EN，如图 1-1 中所示。详细步骤如下所示。

1. 将 CAN 源文件添加到工程中。右键点击 Project Explorer 窗口内的工程名称，并选择“Add Files”。接下来，导航到以下文件夹并选择“Link to Files”。

   `<install_location>\c2000ware\driverlib\f28002x\driverlib\can.c`
2. 编辑 HAL_Obj 以在 *hal_obj.h* 头文件中添加 canHandle。

```
typedef struct _HAL_Obj_ { uint32_t adcHandle[2]; //!< the ADC handles ... ... uint32_t
canHandle; //!< the CAN handle ... ... } HAL_Obj;
```

3. 在 *hal.c* 文件中的 HAL_init() 函数中初始化 CAN 句柄。

```
HAL_Handle HAL_init(void *pMemory,const size_t numBytes)
{
... ...
    // initialize CAN handle
    obj->canHandle = CANA_BASE;             //!< the CAN handle
... ...
    return(handle);
} // end of HAL_init() function
```

4. 对 *communication.h* 文件中的 CAN 设置函数进行原型设计，如下代码所示。

```
//! \brief     Sets up the CANA
//! \param[in] handle  The hardware abstraction layer (HAL) handle
extern void HAL_setupCANA(HAL_Handle handle);
```

5. 在 *communication.c* 文件中将 CAN 设置函数定义为以下代码。

```
void HAL_setupCANA(HAL_Handle halHandle)
{
    HAL_Obj *obj = (HAL_Obj *)halHandle;

    // Initialize the CAN controller
    CAN_initModule(obj->canHandle);

    // Set up the CAN bus bit rate to 200kHz
    // Refer to the Driver Library User Guide for information on how to set
    // tighter timing control. Additionally, consult the device data sheet
    // for more information about the CAN module clocking.
    CAN_setBitRate(obj->canHandle, DEVICE_SYSCLK_FREQ, 500000, 16);

    // Initialize the transmit message object used for sending CAN messages.
    // Message Object Parameters:
    //      Message Object ID Number: 1
    //      Message Identifier: 0x1
    //      Message Frame: Standard
    //      Message Type: Transmit
    //      Message ID Mask: 0x0
    //      Message Object Flags: Transmit Interrupt
    //      Message Data Length: 8 Bytes
    CAN_setupMessageObject(CANA_BASE, TX_MSG_OBJ_ID, 0x1, CAN_MSG_FRAME_STD,
                           CAN_MSG_OBJ_TYPE_TX, 0, CAN_MSG_OBJ_TX_INT_ENABLE,
                           MSG_DATA_LENGTH);

    // Initialize the receive message object used for receiving CAN messages.
    // Message Object Parameters:
    //      Message Object ID Number: 2
    //      Message Identifier: 0x1
    //      Message Frame: Standard
    //      Message Type: Receive
    //      Message ID Mask: 0x0
    //      Message Object Flags: Receive Interrupt
    //      Message Data Length: 8 Bytes
    CAN_setupMessageObject(obj->canHandle, RX_MSG_OBJ_ID, 0x1, CAN_MSG_FRAME_STD,
                           CAN_MSG_OBJ_TYPE_RX, 0, CAN_MSG_OBJ_RX_INT_ENABLE,
                           MSG_DATA_LENGTH);

    // Start CAN module operations
    CAN_startModule(obj->canHandle);

    return;
}  // end of HAL_setupCANA() function
```

备注

要根据系统需求初始化各种 CAN 模块参数，以上内容仅为简单参考。

6. 在 *hal.c* 文件的 HAL 设置时钟函数 HAL_setupPeripheralClks() 中启用相应的 CAN 外设时钟。

```
   SysCtl_enablePeripheral(SYSCTL_PERIPH_CLK_CANA);
```

7. 将相关的 GPIO 配置为 *hal.c* 文件的 HAL_setupGpios() 函数中的 CAN 函数。

```
    // GPIO33->CAN_TX
    GPIO_setPinConfig(GPIO_32_CANA_TX);
    GPIO_setDirectionMode(32, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(32, GPIO_PIN_TYPE_STD);
    GPIO_setQualificationMode(32, GPIO_QUAL_ASYNC);

    // GPIO33->CAN_RX
    GPIO_setPinConfig(GPIO_33_CANA_RX);
    GPIO_setDirectionMode(33, GPIO_DIR_MODE_IN);
    GPIO_setPadConfig(33, GPIO_PIN_TYPE_STD);
    GPIO_setQualificationMode(33, GPIO_QUAL_ASYNC);
```

8. 对 *communication.h* 文件中的 CAN 中断设置函数进行原型设计。

```
extern void HAL_enableCANInts(HAL_Handle handle);
```

9. 定义 CAN 中断设置函数。

```
void HAL_enableCANInts(HAL_Handle handle)
{
    HAL_Obj *obj = (HAL_Obj *)handle;

    // Enable CAN test mode with external loopback
//     CAN_enableTestMode(CANA_BASE, CAN_TEST_EXL);    // Only for debug

    // Enable interrupts on the CAN peripheral.
    CAN_enableInterrupt(obj->canHandle, CAN_INT_IE0 | CAN_INT_ERROR |
                        CAN_INT_STATUS);


    // enable the PIE interrupts associated with the CAN interrupts
    Interrupt_enable(INT_CANA0);


    CAN_enableGlobalInterrupt(obj->canHandle, CAN_GLOBAL_INT_CANINT0);

    // enable the cpu interrupt for CAN interrupts
    Interrupt_enableInCPU(INTERRUPT_CPU_INT9);

    return;
} // end of HAL_enableCANInts() function
```

10. 调用 *sys_main.c* 文件中的 CAN 设置函数和 CAN 中断设置函数。

```
    // setup the CAN
    HAL_setupCANA(halHandle);
    // setup the CAN interrupt
    HAL_enableCANInts(halHandle);
```

11. 对 *communication.h* 文件中的 CAN 中断服务 (ISR) 例程进行原型设计。

```
extern __interrupt void canaISR(void);
```

12. 将 CAN 中断服务 (ISR) 例程矢量添加到 *communication.c* 文件中 initCANCOM() 内的 PIE 表中。

```
    Interrupt_register(INT_CANA0, &canaISR);
```

13. 为了通过在 *communication.c* 文件中添加以下代码来加快执行速度，将 CAN ISR 代码放置在闪存中并从 RAM 运行。

```
#pragma CODE_SECTION(canaISR, ".TI.ramfunc");
```

14. 在 *communication.c* 文件中定义 CAN 中断例程 canaISR()。定义经过原型设计并传递到 PIE 向量表的 canaIS() 函数。下面的示例代码提供了一个使用 CAN 接收/发送消息数据的函数，并清除 PIE 中的中断，从而允许再次触发 CAN 中断。

```
__interrupt void canaISR(void) { ... ... // Check if the cause is the transmit message object
1 // Check if the cause is the transmit message object 1 else if(status == TX_MSG_OBJ_ID)
{ // // Getting to this point means that the TX interrupt occurred on // message object 1, and
the message TX is complete. Clear the // message object interrupt. //
CAN_clearInterruptStatus(CANA_BASE, TX_MSG_OBJ_ID); // Increment a counter to keep track of how
many messages have been // sent. In a real application this could be used to set flags to //
indicate when a message is sent. canComVars.txMsgCount++; // Since the message was sent, clear
any error flags. canComVars.errorFlag = 0; } // Check if the cause is the receive message
object 2 else if(status == RX_MSG_OBJ_ID) { // // Get the received message //
CAN_readMessage(halHandle->canHandle, RX_MSG_OBJ_ID, (uint16_t *)
(&canComVars.rxMsgData[0])); // Getting to this point means that the RX interrupt occurred
on // message object 2, and the message RX is complete. Clear the // message object interrupt.
CAN_clearInterruptStatus(halHandle->canHandle, RX_MSG_OBJ_ID); canComVars.rxMsgCount++;
canComVars.flagRxDone = true; // Since the message was received, clear any error flags.
canComVars. errorFlag = 0; } ... ... // Clear the global interrupt flag for the CAN interrupt
line CAN_clearGlobalInterruptStatus(halHandle->canHandle, CAN_GLOBAL_INT_CANINT0); //
Acknowledge this interrupt located in group 9 Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP9);
return; }
```

15. 在 *communication.h* 文件和 *communication.c* 文件中分别定义 updateCANCmdFreq()，并对其进行原型设计。对 CAN 总线接收和发送的数据进行处理并链接到 updateCANCmdFreq() 中的电机控制变量。

```
void updateCANCmdFreq(MOTOR_Handle handle)
{
...
}
```

16. 在无限循环中调用 updateCANCmdFreq()。

```
updateCANCmdFreq(motorHandle_M1);

if((motorVars_M1.cmdCAN.flagEnablCmd == true) && (motorVars_M1.faultMtrUse.all == 0))
{
    canComVars.flagCmdTxRun = motorVars_M1.cmdCAN.flagCmdRun;
    canComVars.speedSet_Hz = motorVars_M1.cmdCAN.speedSet_Hz;

    if(motorVars_M1.cmdCAN.flagEnablSyncLead == true)
    {
        motorVars_M1.flagEnableRunAndIdentify = motorVars_M1.cmdCAN.flagCmdRun;
        motorVars_M1.speedRef_Hz = motorVars_M1.cmdCAN.speedSet_Hz;
    }
    else
    {
        motorVars_M1.flagEnableRunAndIdentify = canComVars.flagCmdRxRun;
        motorVars_M1.speedRef_Hz = canComVars.speedRef_Hz;
    }
}
```

4.2 支持新的 BLDC 电机驱动器板

C2000 MCU 可与 BLDC 电机驱动器搭配使用，用于驱动三相 BLDC 或 PMSM 电机应用。此通用实验工程可支持各种预定义的 BLDC 电机驱动器。用户可以参考实验工程中的示例代码，并按照本节中所述的步骤来实现更新的或不受支持的 BLDC 电机驱动器。本节以带 SPI 的 DRV8323RS 为例。

1. 设计适用于新 BLDC 电机驱动器 EVM 板的驱动器文件。

   如果 BLDC 电机驱动器支持 SPI，请参阅现有的 BLDC 电机驱动器文件（*drv8323s.h* 和 *drv8323s.c*），并根据需要在 *drv8xxx.h* 和 *drv8xxx.c* 文件中更改寄存器和 API 函数定义。BLDC 电机驱动器寄存器映射的详细说明可在 BLDC 电机驱动器器件的数据表中找到。

   为驱动程序文件创建一组新的文件夹，与 DRV8323 一样（“\libraries\drvic\drv8323\include”和“\libraries\drvic\drv8323\source”）。
2. 将 BLDC 电机驱动器源文件添加到电机控制工程中。

   首先，将 BLDC 电机驱动器源文件添加到您正在处理的工程中。添加文件的方法有两种。

   使用编辑器打开 *universal_motorcontrol_lab.projectspec* projectspec 文件，按如下所示将文件添加到工程中。

```
<file action="link" path="SDK_ROOT/libraries/drvic/drv8323/source/drv8323s.c"
targetDirectory="src_board" applicableConfigurations="Flash_lib_DRV8323RS" />
<file action="link" path="SDK_ROOT/libraries/drvic/drv8323/include/drv8323s.h"
targetDirectory="src_board" applicableConfigurations="Flash_lib_DRV8323RS" />
```

   或者，右键点击 Project Explorer 窗口内的工程名称，并选择“Add Files”。接下来，导航至以下文件夹并从“\libraries\drvic\drv8323\source”中选择设计的驱动程序文件，然后选择“Link to Files”。
3. 将头文件添加到 *hal_obj.h* 文件的 include 列表中。

```
#include "drv8323s.h"
```

   为了确保可以正确找到头文件，请在“Project Properties”-> “Build”-> “C2000 Compiler”-> “Include Options”-> “Add dir to #include search path”中添加指向头文件的目录。

   或者，通过在 *universal_motorcontrol_lab.projectspec* projectspec 文件中添加以下内容，将目录添加到头文件中。

```
-I${SDK_ROOT}/libraries/drvic/drv8323/include
```

4. 编辑 HAL_Obj 以添加 drvic 接口句柄和 SPI 句柄。

   请参阅 DRV8323 文件，按如下所示添加支持代码。

   在 *hal_obj.h* 文件中添加定义。

```
#define DRAdd the defines in hal_obj.h fileVIC_Obj              DRV8323_Obj
#define DRVIC_VARS_t            DRV8323_VARS_t
#define DRVIC_Handle            DRV8323_Handle
#define DRVICVARS_Handle        DRV8323VARS_Handle

#define DRVIC_init              DRV8323_init
#define DRVIC_enable            DRV8323_enable
#define DRVIC_writeData         DRV8323_writeData
#define DRVIC_readData          DRV8323_readData

#define DRVIC_setupSPI          DRV8323_setupSPI

#define DRVIC_setSPIHandle      DRV8323_setSPIHandle
#define DRVIC_setGPIOCSNumber   DRV8323_setGPIOCSNumber
#define DRVIC_setGPIOENNumber   DRV8323_setGPIOENNumber
```

   将 drvic 接口句柄和 SPI 句柄添加到 HAL_Obj：

```
  uint32_t       spiHandle;          //!< the SPI handle

  DRVIC_Handle   drvicHandle;        //!< the drvic interface handle
  DRVIC_Obj      drvic;              //!< the drvic interface object

  uint32_t       gateEnableGPIO;
  // BSXL8353RS_REVA
```

5. 配置 SPI 以与 BLDC 电机驱动器进行通信。

   将电机驱动器与 SPI 配合使用时，必须从 MCU 正确配置 SPI，以匹配与 BLDC 电机驱动器器件正确通信所需的格式。

   在 *hal.c* 文件的 HAL_setupGPIOs() 中为 SPI 函数配置相关 GPIO。确保查看 BLDC 电机驱动器数据表，以确定每个 SPI 引脚是否需要外部上拉或下拉电阻器，或者是否配置为推挽引脚。

```
    // GPIO5->Connect to GPIO5 using a jumper wire->M1_DRV_SCS
    GPIO_setPinConfig(GPIO_5_SPIA_STE);
    GPIO_setDirectionMode(5, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(5, GPIO_PIN_TYPE_STD);

    // GPIO9->M1_DRV_SCLK*
    GPIO_setPinConfig(GPIO_9_SPIA_CLK);
    GPIO_setDirectionMode(9, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(9, GPIO_PIN_TYPE_PULLUP);

    // GPIO10->SPIA_SOMI->M1_DRV_SDO*
    GPIO_setPinConfig(GPIO_10_SPIA_SOMI);
    GPIO_setDirectionMode(10, GPIO_DIR_MODE_IN);
    GPIO_setPadConfig(10, GPIO_PIN_TYPE_PULLUP);

    // GPIO11->SPIA_SIMO->M1_DRV_SDI*
    GPIO_setPinConfig(GPIO_11_SPIA_SIMO);
    GPIO_setDirectionMode(11, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(11, GPIO_PIN_TYPE_PULLUP);
```

   在 *hal.c* 文件中的 HAL_setupSPI() 中为波特率、数据帧配置 SPI 控制寄存器：

```
    // Must put SPI into reset before configuring it
    SPI_disableModule(obj->spiHandle);

    // SPI configuration. Use a 500kHz SPICLK and 16-bit word size, 25MHz LSPCLK
    SPI_setConfig(obj->spiHandle, DEVICE_LSPCLK_FREQ, SPI_PROT_POL0PHA0,
                  SPI_MODE_MASTER, 400000, 16);

    SPI_disableLoopback(obj->spiHandle);

    SPI_setEmulationMode(obj->spiHandle, SPI_EMULATION_FREE_RUN);

    SPI_enableFIFO(obj->spiHandle);
    SPI_setTxFifoTransmitDelay(obj->spiHandle, 0x10);

    SPI_clearInterruptStatus(obj->spiHandle, SPI_INT_TXFF);

    // Configuration complete. Enable the module.
    SPI_enableModule(obj->spiHandle);
```

6. 为其他输入和输出引脚配置 GPIO，例如 ENABLE、nFAULT。您可以参考 *hal.c* 文件的 HAL_setupGPIOs()、HAL_setupGate() 中的示例代码以及 *hal.h* 文件中的定义，如下所示：

```
//! \brief Defines the gpio for enabling Power Module
#define MTR1_GATE_EN_GPIO       29

//! \brief Defines the gpio for the nFAULT of Power Module
#define MTR1_PM_nFAULT_GPIO     34
```

7. 在 *hal.c* 文件的 HAL_MTR_setParams() 中调用 HAL_setupSPI() 和 HAL_setupGate() 函数：

```
    // setup the spi for drv8323/drv8353/drv8316
    HAL_setupSPI(handle);

    // setup the drv8323s/drv8353s/drv8316s interface
    HAL_setupGate(handle);
```

8. 按如下所示调用 *motor1_drive.c* 文件中的驱动程序函数：

```
// turn on the DRV8323/DRV8353/DRV8316 if present
HAL_enableDRV(obj->halMtrHandle);

// initialize the DRV8323/DRV8353/DRV8316 interface
HAL_setupDRVSPI(obj->halMtrHandle, &drvicVars_M1);
```

9. 如果需要，更改 BLDC 电机驱动器的默认设置值。

```
drvicVars_M1.ctrlReg05.bit.VDS_LVL = DRV8323_VDS_LEVEL_1P700_V;
drvicVars_M1.ctrlReg05.bit.OCP_MODE = DRV8323_AUTOMATIC_RETRY;
drvicVars_M1.ctrlReg05.bit.DEAD_TIME = DRV8323_DEADTIME_100_NS;
drvicVars_M1.ctrlReg06.bit.CSA_GAIN = DRV8323_Gain_10VpV;

drvicVars_M1.ctrlReg06.bit.LS_REF = false;
drvicVars_M1.ctrlReg06.bit.VREF_DIV = true;
drvicVars_M1.ctrlReg06.bit.CSA_FET = false;

drvicVars_M1.writeCmd = 1;
HAL_writeDRVData(obj->halMtrHandle, &drvicVars_M1);
```

4.3 将参考代码移植到新的 C2000 MCU

电机控制通用实验工程可以移植到其他支持 FPU 和 TMU 的 C2000 MCU 控制器。以下步骤详细说明了如何移植实验练习代码。F28004x MCU 用作新目标 C2000 MCU 的示例。若要调整支持 SysConfig 的实验版本，请在以下全部说明中将 `universal_motorcontrol_lab` 替换为 `universal_motorcontrol_syscfg`。

1. 浏览到 `<install_location>\solutions\universal_motorcontrol_lab` 文件夹，然后选择现有的器件特定实验文件夹之一。本示例中将使用“f28002x”文件夹，但可以使用任何文件夹。
2. 在同一 universal_motorcontrol_lab 文件夹中创建所选器件特定实验的副本，并将名称更改为“f28004x”。`<install_location>\solutions\universal_motorcontrol_lab\f28004x` 将是您最终移植的实验的位置，在下面会称为 `<f28004x_lab_location>`。
3. 编译器使用 cmd 文件来映射 C2000 MCU 的存储器。浏览至 `<f28004x_lab_location>\cmd` 文件夹，并更新 *f28002x_flash_lib_is.cmd* 文件的名称，以反映新的 f28004x 器件。请注意，该文件夹中存在多个其他 cmd 文件，它们在默认情况下未使用且可以忽略。
4. 通用电机控制实验使用 C2000Ware 器件驱动程序文件 *device.c/h* 中提供的引脚定义。必须针对新器件更新这些定义。
   a. 导航至新 C2000 MCU 的 C2000Ware 通用器件支持文件夹，位于 `<install_location>\c2000ware\device_support\f28004x`。在 `...\common\include` 子文件夹中找到 *device.h* 文件。
   b. 将 *device.h* 文件复制到 `<f28004x_lab_location>\drivers\include` 文件夹，替换现有文件。
   c. 导航回新 C2000 MCU 的 C2000Ware 通用器件支持文件夹。在 `...\common\source` 子文件夹中找到 *device.c* 文件。
   d. 将 *device.c* 文件复制到 `<f28004x_lab_location>\drivers\source` 文件夹，替换现有文件。
5. 浏览到 `<f28004x_lab_location>\ccs\motor_control` 文件夹，并使用编辑器打开 projectspec 文件。CCS 使用该文件在用户工作区中生成工程文件夹，并包含对特定于器件的 C2000Ware 源文件的引用。
   a. 如果使用支持 SysConfig 的实验版本，请更新下面的**粗体文本**以指示新 C2000 MCU 的软件包。这行文本可在文件的器件定义部分中找到，这应该是第一个部分。

```
sysConfigBuildOptions --product ${C2000WARE_ROOT}/.metadata/sdk.json --device F28002x --
package 80QFP --part F28002x_80QFP"
```

b. 某些 C2000 MCU 具有与其他 C2000 MCU 不同的功能。更新此文件以反映这些差异。例如，F28002x MCU 支持快速整数除法 (FINTDIV)，而 F28004x 则不支持。相关的处理器选项是“idiv_support”项。查找并删除所有“--idiv_support=idiv0”实例，因为 F28004x 不支持此函数。
   i. 如果您不确定需要进行哪些更改，请参阅 TMS320C28x 优化 C/C++ 编译器 v22.6.0.LTS 用户指南第 2.3 节 *使用选项更改编译器的行为* 的表 2-1 *处理器选项*，其中详细介绍了每个选项的用途。
   ii. 确定该器件与您选择的新 C2000 MCU 之间存在哪些差异。有关此过程的帮助，请参阅 C2000 实时控制外设参考指南，其中介绍了器件和外设版本之间的差异。
   iii. 根据需要在 projectspec 文件中进行调整。
c. 找到文件中的所有“28002x”实例并替换为“28004x”。
d. 在文件中找到“280025C”的所有实例。
   i. 第一个结果应在文件开头附近，指定工程器件。更新粗体文本以正确显示为该工程选择的新 C2000 MCU。

```
<project
    name="universal_motorcontrol_lab_f28004x"
    device="TMS320F280025C"
```

   ii. 最后两个结果应位于文件的最后一个部分。以下摘录可在 ccxml 文件“复制文件”操作的“路径”中找到。

```
/TMS320F280025C_LaunchPad.ccxml
/TMS320F280025C.ccxml
```

   iii. 更新粗体文本以正确指示新 C2000 MCU 系列器件的通用目标配置文件，这些文件可以在之前引用的器件支持文件夹中的`...\common\targetConfigs` 下找到。对于所有 F28004x 器件，粗体文本应更改为“**TMS320F280049C**”。
   iv. “280025C”的所有其他结果应位于注释中。为了确保文档准确性，建议更新这些内容，但并不重要。

6. 将“universal_motorcontrol_lab_f28004x”工程导入 CCS。
   a. 请注意，导入工程后，CCS 可能会显示错误，指示未找到 *f28004x_headers_nonbios.cmd* 文件。此错误不会影响性能，但可能会增加调试的难度。在该文件中执行的内存分配只被调试环境监视窗口使用，在节 3.5.1.2 的增量式构建阶段中进行了说明。
   b. 若要充分利用调试环境监视窗口，请遵循本节中与 *f28002x_flash_lib_is.cmd* 文件以及 *f28002x_headers_nonbios.cmd* 文件相关的所有说明。
7. 打开 cmd 文件并根据所选的器件更改存储器映射。对于完全不熟悉此类型文件的用户，请参阅 TI 链接器命令文件入门，了解深入介绍和基本使用指南。
   a. 与针对旧器件修改 cmd 文件相比，修改某个通用 C2000Ware cmd 文件（例如 *28004x_generic_flash_lnk.cmd* 文件）可能更容易。这些文件位于器件支持文件夹的 `...\common\cmd` 子文件夹中。在这种情况下，应使用工程的原始 cmd 文件作为参考。
   b. 如果使用 *f28004x_headers_nonbios.cmd* 文件，则通用 C2000Ware cmd 文件位于器件支持文件夹的 `...\headers\cmd` 子文件夹中。
8. 按照基于 F28004x 的硬件套件的节 4.1.2 中所述，修改 GPIO、PWM、ADC 和 CMPSS 模块以及 *hal.h* 文件中的定义。
9. 重新编译实验工程。工程中的所有错误或警告将显示在 CCS 控制台窗口中。按照消息提示修复所有错误或警告。器件之间的 driverlib API 有一些差异，此时必须考虑到这一点。
10. 要添加函数以配置和使用新 C2000 MCU 中存在但在这些文件的原始 C2000 MCU 源代码中不存在的外设，请参阅 C2000Ware 或 MotorControlSDK 中的示例函数。例如，F28004x 具有可编程增益放大器 (PGA)，而 F28002x 没有。
11. 使用不同的构建级别以增量方式运行工程，以测试和验证功能。
12. 如果要多次导入该工程，则最好更新工程的源文件，以便只需执行一次这些更改。导航至 `<f28004x_lab_location>` 文件夹。
   a. 在 `...\cmd` 子文件夹中，将 *f28004x_flash_lib_is.cmd* 文件替换为所导入 F28004x 工程中的更新文件。
   b. 在 `...\drivers\source` 子文件夹中，将 *hal.c* 文件替换为所导入的 F28004x 工程中的更新文件。
   c. 在 `...\drivers\include` 子文件夹中，将 *hal.h* 文件替换为所导入 F28004x 工程中的更新文件。

A 附录 A. 电机控制参数

通用电机控制实验定义了许多会影响系统性能的参数。下面列出了 user_mtr1.h 文件中可用于用户操作的参数。其中一些参数被描述为推导参数，通常不应更改，这些参数的值取决于一个或多个其他参数。

- **硬件参数：**
  - **USER_M1_NOMINAL_DC_BUS_VOLTAGE_V**

    此参数定义了标称直流母线电压，单位为伏特 (V)。

    此参数的值必须大于额定电机电压，并且小于硬件的最大检测直流母线电压。
  - **USER_M1_ADC_FULL_SCALE_VOLTAGE_V**

    此参数定义了 ADC 输入端的最大电压，单位为伏特 (V)。此值由最大 ADC 读数表示。

    此参数用于缩放 ADC 读数，应设置为 ADC 输入端的最大预期电压。
  - **USER_M1_ADC_FULL_SCALE_CURRENT_A**

    此参数定义了 ADC 输入端的最大电流，单位为安培 (A)。此值由最大 ADC 读数表示。

    此参数用于缩放 ADC 读数，应设置为 ADC 输入端的最大预期电流。
  - **USER_M1_VOLTAGE_FILTER_POLE_Hz**

    此参数定义了模拟滤波器极点位置（单位为 Hz）。

    此参数必须设置为与硬件电压反馈滤波器的滤波器极点位置相匹配。
  - **USER_M1_VOLTAGE_FILTER_POLE_rps**

    该推导参数是 USER_M1_VOLTAGE_FILTER_POLE_Hz 的弧度/秒转换。
  - **USER_M1_SIGN_CURRENT_SF**

    此参数定义源自 USER_M1_ADC_FULL_SCALE_CURRENT_A 的电流比例因子的符号（正或负）。

    如果运算放大器的同相 (+) 引脚在电流反馈电路中接地，则该参数的值为 -1.0f。如果反相引脚 (-) 接地，则该值为 +1.0f。
  - **USER_M1_NUM_CURRENT_SENSORS**

    此参数定义硬件中存在的电流传感器的数量。

    本实验中的计算假设使用 2 个或 3 个电流传感器。因此，较高或较低的值将导致编译错误。
  - **USER_M1_NUM_VOLTAGE_SENSORS**

    此参数定义硬件中存在的电压相位传感器的数量。

    本实验中的计算假设有 3 个电压传感器。因此，较高或较低的值将导致编译错误。
  - **USER_M1_Ix_OFFSET_AD**

    USER_M1_Ix_OFFSET_AD 参数，其中“x”是 A、B 或 C，表示每个相应相位的 ADC 电流偏移，由硬件定义。

    该值应接近 2048。

    该实验能够通过“flagEnableOffsetCalc”标志自动进行偏移校准。如果设置，则在校准过程完成后，此参数的新值将加载到 motorVars_M1.adcData.offset_I_ad.value[2:0] 中。必须手动更新 user_mtr1.h 中的值。

    USER_M1_IA_OFFSET_AD；USER_M1_IB_OFFSET_AD；USER_M1_IC_OFFSET_AD
  - **USER_M1_Vx_OFFSET_SF**

    USER_M1_Vx_OFFSET_SF 参数，其中“x”是 A、B 或 C，表示每个相应相位的 ADC 电压偏移，由硬件定义。这些参数仅用于 InstaSPIN-FOC FAST 和 InstaSPIN-BLDC。

    该值应接近 0.5。

该实验能够通过“flagEnableOffsetCalc”标志自动进行偏移校准。如果设置，校准过程完成后，此参数的新值将加载到 motorVars_M1.adcData.offset_V_sf.value[2:0] 中。必须手动更新 user_mtr1.h 中的值。

USER_M1_VA_OFFSET_SF；USER_M1_VB_OFFSET_SF；USER_M1_VC_OFFSET_SF

- **USER_M1_IDC_OFFSET_AD**

  此参数定义单分流器硬件的 ADC 电流偏移。

  与 USER_M1_Ix_OFFSET_AD 一样，可通过“flagEnableOffsetCalc”标志校准该值。新值加载到 motorVars_M1.adcData.offset_Idc_ad 中。

- **时序参数：**
  - **USER_SYSTEM_FREQ_Hz**

    此推导参数（位于 user_common.h 中）是器件 SysClk 频率 DEVICE_SYSCLK_FREQ 的浮点转换，以 Hz 为单位。DEVICE_SYSCLK_FREQ 可在 device.h 文件中找到。
  - **USER_M1_PWM_FREQ_kHz**

    此参数定义了 ePWM 的频率，以 kHz 为单位。更改该值会改变控制中断频率。
  - **USER_M1_PWM_PERIOD_usec**

    此推导参数是 ePWM 频率 USER_M1_PWM_FREQ_kHz 的周期，以微秒为单位。
  - **USER_M1_ISR_FREQ_Hz**

    此推导参数定义了控制中断频率（单位为 Hz）。

    此参数源自 PWM 频率 USER_M1_PWM_FREQ_kHz。
  - **USER_M1_ISR_PERIOD_usec**

    此推导参数是控制中断频率 USER_M1_ISR_FREQ_Hz 的周期，以微秒为单位。
  - **USER_M1_NUM_PWM_TICKS_PER_ISR_TICK**

    此参数定义每次中断的 PWM 周期数。该值将控制中断频率分频。

    本实验假定此值介于 1 和 3 之间（包括 1 和 3）。
  - **USER_M1_NUM_ISR_TICKS_PER_SPEED_TICK**

    此参数定义了每个速度控制器时钟节拍的中断时钟节拍数。此值将速度控制器触发速率分频。这会影响系统响应时间。
- **采样参数：**
  - **USER_M1_CURRENT_SF**

    此导出参数计算电流读数的 ADC 结果位相对于实际电流值（安培 (A)）的比例。此计算假定使用 12 位 ADC。

    此参数源自 ADC 满量程电流 USER_M1_ADC_FULL_SCALE_CURRENT_A。
  - **USER_M1_VOLTAGE_SF**

    这个推导参数计算电压读数的 ADC 结果位相对于以伏特 (V) 为单位的实际电压值的比例。此计算假定使用 12 位 ADC。此参数仅供 InstaSPIN-FOC FAST 和 eSMO 估算器使用。

    此参数是从 ADC 满量程电压 USER_M1_ADC_FULL_SCALE_VOLTAGE_V 推导出的。
  - **USER_M1_CURRENT_INV_SF**

    对于 12 位 ADC，此推导参数等效于 1 / USER_M1_CURRENT_SF。

    此参数源自 ADC 满量程电流 USER_M1_ADC_FULL_SCALE_CURRENT_A。
  - **USER_M1_DCLINKSS_MIN_DURATION**

    此参数定义了单分流器 ADC 读数的最短时钟周期。

    motor1_drive.c 文件中介绍了此参数值的计算。
  - **USER_M1_DCLINKSS_SAMPLE_DELAY**

该参数定义了在获得单分流器 ADC 读取以考虑信号传播问题之前的时钟周期延迟。

motor1_drive.c 文件中介绍了此参数值的计算。

- **电机属性：**
  - **USER_MOTOR1_TYPE**

    此参数定义电机类型。

    此参数的两个有效值为 MOTOR_TYPE_PM（对于 BLDC、PMSM、SMPM 或 IPM 电机）和 MOTOR_TYPE_INDUCTION（对于异步 ACI 电机）
  - **USER_MOTOR1_NUM_POLE_PAIRS**

    此参数定义了电机极对数。当使用 InstaSPIN-FOC FAST 估算器时，此值被用来计算电机输出功率。
  - **USER_MOTOR1_Rr_Ohm**

    此参数定义了感应电机的电机转子电阻，单位为欧姆。对于非感应电机，设定为 NULL。此参数仅供 InstaSPIN-FOC FAST 使用。
  - **USER_MOTOR1_Rs_Ohm**

    此参数定义了电机定子电阻，单位为欧姆。
  - **USER_MOTOR1_Ls_d_H**

    此参数定义了直接方向上电机的定子电感，单位为亨利 (H)。对于 PM，这是平均定子电感。
  - **USER_MOTOR1_Ls_q_H**

    此参数定义了正交方向上电机的定子电感，单位为亨利 (H)。对于 PM，这是平均定子电感。
  - **USER_MOTOR1_RATED_FLUX_VpHz**

    此参数以 V/Hz（或 V*s，韦伯）为单位定义了电机的额定磁通。
  - **USER_MOTOR1_MAGNETIZING_CURRENT_A**

    此参数定义了直接方向上电机的额定电流值，单位为安培，仅适用于感应电机。对于其他电机，设置为 NULL。此参数仅供 InstaSPIN-FOC FAST 使用。
  - **USER_MOTOR1_MAX_CURRENT_A**

    此参数定义了电机的最大电流，以 A 表示。这会影响过流处理。
  - **USER_MOTOR1_INERTIA_Kgm2**

    该参数定义了与电机刚性耦合的质量惯性矩，单位为 kg/m^2。这有助于进行速度控制器增益常数计算。
  - **USER_M1_VD_SF**

    此参数定义 ID 电流控制器 Vd 的初始最大值。请参阅 USER_M1_MAX_VS_MAG_PU 以了解更多详细信息。

    此值必须介于 0.1 到 0.95 之间。
  - **USER_M1_MAX_VS_MAG_PU**

    此参数定义了标幺 V 中电压矢量 Vs 的最大幅值。

    根据矢量的定义，电压矢量 Vs 的幅度与 Vd 和 Vq 轴分量之间的关系如下：

    $Vs = \sqrt{(Vd^2 + Vq^2)}$

    Vd 由最终应用确定，对于某些电机，0 值是有效的选择。在本示例中，根据使用的是闭环控制还是开环控制，对此进行了不同的定义。在任一种情况下，Vs 和 Vd 的计算都在下面提供。

    在开环控制下，在电压-频率表征曲线中将 Vd 设置为 0.3。

    在闭环控制中，Vd 由 PI 控制器动态确定。|Vd| 的最大幅度最初设置为 USER_M1_VD_SF * USER_MOTOR1_RATED_VOLTAGE_V。在电机运行期间，它设为 USER_MOTOR1_RATED_VOLTAGE_V。

    在开环控制中，Vs 设置为 USER_M1_MAX_VS_MAG_PU。所有运算均采用标幺值。

在闭环控制中，Vs 设置为 USER_M1_MAX_VS_MAG_PU * obj->adcData.VdcBus_V（直流母线电压）。所有操作均以伏特为单位。

- **USER_M1_MAX_ACCEL_Hzps**

  此参数定义了估算速度曲线的最大加速幅度（单位为 Hz/s）。仅在 InstaSPIN 电机 ID 期间使用。

  此值通常应保留为默认值。
- **InstaSPIN-FOC FAST 参数：**
- **USER_MOTOR1_RES_EST_CURRENT_A**

  此参数定义了要用于定子电阻估算的最大电流值，单位为安培。

  该值应设置为电机额定相电流的 10% 至 40%。如果电机在整个斜升过程中不旋转，则以 5% 为增量增大该电流，直至电机在整个斜升过程中旋转。
- **USER_MOTOR1_IND_EST_CURRENT_A**

  此参数定义了用于定子电感估算的最大电流值，单位为安培。

  该值应设置为电机额定相电流的 10% 至 20%，刚好能够使电机旋转。
- **USER_MOTOR1_FLUX_EXC_FREQ_Hz**

  此参数定义了磁通激励频率，用于估算电机识别期间的定子电感和磁通，单位为 Hz。

  通常，电机额定频率的 5% 至 20% 足够高，足以估算定子电感为 ~10-20 微亨或更高的 PMSM 电机。如果电感为几 µH，建议使用较高频率（高达 60Hz）来估算超低电感。
- **USER_M1_R_OVER_L_EXC_FREQ_Hz**

  此参数定义了以 Hz 为单位的 R/L 激励频率，在电机识别期间用于估算定子电阻和电感的初始值。这用于计算电流控制器增益。

  默认情况下，该参数设置为 300Hz。
- **USER_M1_IDRATED_FRACTION_FOR_L_IDENT**

  此参数定义要在电感估算期间使用的 Id 额定值（小数表示形式）。

  不应更改此参数的默认值 0.5。
- **USER_M1_SPEEDMAX_FRACTION_FOR_L_IDENT**

  此参数定义要在电感估算期间使用的最大速度（小数表示形式）。

  不应更改此参数的默认值 1.0。
- **USER_M1_R_OVER_L_KP_SF**

  此参数定义 FOC 电流控制器中 Kp 的比例因子。

  不应更改此参数的默认值 0.02。
- **USER_M1_IDRATED_DELTA_A**

  此参数定义了估算期间要使用的 Id 差值电流，单位为 A。

  不应更改此参数的默认值 0.0001。
- **USER_M1_FORCE_ANGLE_FREQ_Hz**

  如果启用了强制角启动选项，则此参数定义强制角启动频率，单位为 Hz。此值在电机启动期间使用。

  典型强制角启动速度为 +-1Hz。
- **USER_MOTOR1_FREQ_NEARZEROLIMIT_Hz**

  如果启用了强制角启动选项，则此参数定义使用强制角启动过程的速度范围。

  对于大多数应用，默认值 5Hz 就足够了。
- **USER_M1_PW_GAIN**

  此参数定义了用于计算感应电机 Id 参考值的 PowerWarp 增益。

不应更改此参数的默认值 1.0。

- **USER_M1_DCBUS_POLE_rps**

  此参数定义了软件直流母线滤波器的极点位置（单位为 rad/s）。

  不应更改此参数的默认值 100。

- **USER_M1_SPEED_POLE_rps**

  此参数定义了软件频率估算器滤波器的极点位置（单位为 rad/s）。

  对于大多数应用，默认值 100rps 就足够了。对于高速电机，将该值增加到 500rps 可以提高性能。

- **USER_M1_EST_FLUX_HF_SF**

  此参数定义磁通估算的比例因子。

  此参数的范围是 0.1 至 1.25。对于大多数 PMSM 电机，值 1.0 就足够了。对于频率更高、电感更低的电机，可以使用较低的值。

- **USER_M1_EST_BEMF_HF_SF**

  此参数定义了估算反电动势 (BEMF) 的比例因子。

  如果使用 InstaSPIN-FOC FAST 估算器，则不应更改此参数的默认值 1。

  否则，此参数的范围是 0.5 至 1.25。对于大多数 PMSM 电机，值 1.0 就足够了。对于频率更高、电感更低的电机，可以使用较低的值。

- **USER_MOTOR1_Ls_d_COMP_COEF**

  此参数定义 Ls d 轴补偿系数。电机电感随施加的定子电流的振幅而降低。为了简化计算，假设存在线性关系。

  其中 Ls 是电机定子电感 Ld，Ls^1 是补偿后的电机定子电感，

  Ls^1 = Ls * (1 -(Ls Compensation Coefficient))

  应根据电机制造商提供的电感与电流模型设置此参数。

- **USER_MOTOR1_Ls_q_COMP_COEF**

  此参数与 USER_MOTOR1_Ls_d_COMP_COEF 相同，只不过它引用的是 Lq 而非 Ld。

- **USER_MOTOR1_Ls_MIN_NUM_COEF**

  此参数定义了电机的最小电感。

  应根据电机制造商提供的电感与电流模型设置此参数。

- **USER_MOTOR1_RSONLINE_WAIT_TIME**

  当电机运行在闭环中时，Rs 在线校准被用来重新校准定子电阻 Rs。但是，考虑到 Rs 随时间缓慢变化，并且 Rs 在线校准过程会注入不会生成扭矩的 d 轴电流，因此该过程只会间歇性地启用。

  此参数定义了 5ms 内 Rs 在线校准周期之间的等待时间。

  此参数的默认值通常就足够了。

- **USER_MOTOR1_RSONLINE_WORK_TIME**

  此参数定义了 Rs 在线校准周期的持续时间，以 5ms 为单位。

  此参数的默认值通常就足够了。

- **开环电压/频率曲线参数：**
  - **USER_MOTOR1_FREQ_LOW_Hz**

    此参数定义了开环电压/频率曲线的低频边界，单位为 Hz。

    此值定义了电机电压/频率曲线线性区域的下限。在线性区域内，计算实现选定电机转速所需的施加电压极其简单。超过下限后，该关系不再是线性的。

    此参数应设置为额定电机频率的大概 10%。

  - **USER_MOTOR1_FREQ_HIGH_Hz**

此参数定义了开环电压/频率曲线的高频边界，单位为 Hz。

此值定义了电机电压/频率曲线线性区域的上限。在线性区域内，计算实现选定电机转速所需的施加电压极其简单。电压的增加不应超过电机的额定电压。

此参数应设置为电机的额定频率。

- **USER_MOTOR1_VOLT_MIN_V**

  此参数定义了在电机中生成 USER_MOTOR1_FREQ_LOW_Hz 速度的电压值，单位为 V。

  此参数应设置为额定电机电压的 15% 左右。
- **USER_MOTOR1_VOLT_MAX_V**

  此参数定义了在电机中生成 USER_MOTOR1_FREQ_HIGH_Hz 速度的电压值，单位为 V。

  此参数应设置为额定电机电压。

- **电机编码器参数：**
  - **USER_MOTOR1_NUM_ENC_SLOTS**

    此参数定义了电机正交编码器中的编码器槽数。

    应根据使用的编码器设置此参数。
  - **USER_MOTOR1_ENC_POS_MAX**

    此推导参数定义了 eQEP 模块的位置计数器的最大值。

    此参数从编码器插槽数 (USER_MOTOR1_NUM_ENC_SLOTS) 推导得出。
  - **USER_MOTOR1_ENC_POS_OFFSET**

    此参数定义位置零处编码器的偏移量。
- **eSMO 估算器参数：**
  - **USER_MOTOR1_KSLIDE_MAX**

    此参数定义了时间相关的 eSMO Kslide 增益变量的最大值。

    此参数必须设置为大于最大电机电频率下的最大反电动势（标幺值）。默认值对于大多数应用而言已经足够了。此参数可能在 0.1 和 10 之间变化。
  - **USER_MOTOR1_KSLIDE_MIN**

    此参数定义了时间相关的 eSMO Kslide 增益变量的初始值。

    此参数必须设置为大于最大电机电频率下的最大反电动势（标幺值）。默认值对于大多数应用而言已经足够了。此参数可能在 0.1 和 10 之间变化。
  - **USER_MOTOR1_PLL_KP_MAX**

    此参数定义 eSMO PLL 的最大增益。

    此参数必须设置为大于最大电机电频率下的最大反电动势（标幺值）。默认值对于大多数应用而言已经足够了。此参数可能在 0.1 和 10 之间变化。
  - **USER_MOTOR1_PLL_KP_MIN**

    此参数定义 eSMO PLL 的最小增益。

    此参数必须设置为大于最大电机电频率下的最大反电动势（标幺值）。默认值对于大多数应用而言已经足够了。此参数可能在 0.1 和 5 之间变化。
  - **USER_MOTOR1_PLL_KP_SF**

    此参数定义应用于每个单元中电机转速的比例因子。这控制 eSMO PLL 的增益。

    此参数应设置为 (USER_MOTOR1_PLL_KP_MAX - USER_MOTOR1_PLL_KP_MIN) / (fscale * fmax)。在大多数情况下、默认值就足够了。
  - **USER_MOTOR1_BEMF_THRESHOLD**

    此参数定义 eSMO 滑动模式控制器估算电流误差的阈值。

此参数应等于电机最大反电动势 (BEMF) 除以电机额定电压，范围应在 0.3 到 0.5 之间。在大多数情况下、默认值 0.5 就足够了。

- **USER_MOTOR1_BEMF_KSLF_FC_Hz**

  该参数定义 eSMO 估算反电动势的低通滤波器的截止频率。

  此参数等于截止频率 * 2 * PI() * Ts（时间刻度）。在大多数情况下、默认值就足够了。
- **USER_MOTOR1_THETA_OFFSET_SF**

  此参数定义应用于反电动势滤波器补偿系数的系数。

  此参数应设置在 0.5 到 1.5 之间（包括 0.5 和 1.5）。大多数情况下，默认值 1 就足够了。
- **USER_MOTOR1_SPEED_LPF_FC_Hz**

  此参数设置用于计算速度的低通滤波器的截止频率。如果将此滤波器应用于标幺值，则此参数仅影响噪声非常高的情况下的 eSMO 输出。

  大多数情况下，默认值 200Hz 就足够了。否则，此参数应设置在 100 和 400 之间。

- **InstaSPIN-BLDC 参数：**
  - **USER_MOTOR1_RAMP_START_Hz**

    此参数定义了电机开环运行时的最小速度，单位为 Hz。

    此参数的具体值应基于电机或系统性能要求测试。该值可能为额定电机转速的 0.5%-1%。
  - **USER_MOTOR1_RAMP_END_Hz**

    此参数定义了何时开始考虑换向的 BEMF 过零点，单位为 Hz。

    此参数的具体值应基于电机或系统性能要求测试。该值可能为额定电机转速的 10%-20%。
  - **USER_MOTOR1_RAMP_DELAY**

    此参数定义系统在开环中运行电机的时长，单位为秒。

    此参数的具体值应基于电机或系统性能要求测试。该值可以在 5-20s 之间。
  - **USER_MOTOR1_ISBLDC_INT_MAX**

    此参数定义用于换向点检测的积分电压的最大阈值，以标称值的百分比表示。

    此参数的具体值应基于电机或系统性能要求测试。该值可以介于 0.01-0.15 之间。
  - **USER_MOTOR1_ISBLDC_INT_MIN**

    此参数定义用于换向点检测的积分电压的最小阈值，以标称值的百分比表示。

    此参数的具体值应基于电机或系统性能要求测试。该值可以介于 0.005-0.1 之间。
  - **USER_MOTOR1_ISBLDC_I_START_A**

    此参数定义了以开环方式启动电机的电流，单位为 A。

    此参数的具体值应基于电机或系统性能要求测试。该值可能介于额定电流的 5%-20% 之间。
  - **USER_MOTOR1_ISBLDC_DUTY_START**

    此参数定义了在开环中启动电机的 PWM 占空比，以百分比表示。

    此参数的具体值应基于电机或系统性能要求测试。该值可以介于 5%-20% 占空比之间。
- **弱磁控制 (FWC) 参数：**
  - **USER_M1_FWC_KP**

    此参数定义了 FWC PI 稳压器的 Kp 增益。

    此参数的具体值应基于电机或系统性能要求测试。该值可以介于 0.01-0.1 之间。
  - **USER_M1_FWC_KI**

    此参数定义了 FWC PI 稳压器的 Ki 增益。

    此参数的具体值应基于电机或系统性能要求测试。该值可以介于 0.01-0.1 之间。

- **USER_M1_FWC_MAX_ANGLE**

  此参数定义了 FWC 的最大矢量角度，单位为度。

  此参数应定义在 0 至 -45 度之间。对于 FWC，d 轴上的较大负值会设置较高的负电流。
- **USER_M1_FWC_MIN_ANGLE**

  此参数定义了 FWC 的最小矢量角度，单位为度。

  不应更改此参数的默认值 0。

- **启动和运行状态参数：**
  - **USER_MOTOR1_RATED_VOLTAGE_V**

    此参数定义了电机的额定电压，单位为 V。

    此参数可以设置为 USER_M1_NOMINAL_DC_BUS_VOLTAGE_V / sqrt(2)。

- **USER_MOTOR1_FREQ_MAX_Hz**

  此参数定义了电机的最大转速，单位为 Hz。

  此参数是系统相关的，并且有可能高于电机的额定速度。通常，电机转速为额定转速的 100% 至 150%。应根据系统性能要求确定具体的值。
- **USER_MOTOR1_ALIGN_CURRENT_A**

  此参数定义了用于运行电机转子对齐的电流，以 A 表示。

  此参数应设置为电机额定电流的 5%-50%。应根据系统性能要求确定具体的值。
- **USER_MOTOR1_FLUX_CURRENT_A**

  此参数定义了用于在强制开环中运行电机的电流，单位为 A。

  此参数应设置为电机额定电流的 5%-50%。应根据系统性能要求确定具体的值。
- **USER_MOTOR1_STARTUP_CURRENT_A**

  此参数定义了电流 (A)，用于在速度低于定义的启动速度 (USER_MOTOR1_SPEED_START_Hz) 时闭环运行电机。

  此参数应设置为额定电机电流的 10%-100%。
- **USER_MOTOR1_TORQUE_CURRENT_A**

  此参数定义了最初为电机启动电流分配的值，单位为 A。
- **USER_MOTOR1_SPEED_START_Hz**

  此参数定义了电机启动速度阈值，单位为 Hz。

  此参数应设置为高于电机最低转速，通常下降到额定电机转速的 10%-50% 之间。
- **USER_MOTOR1_SPEED_FORCE_Hz**

  此参数定义了电机的强制开环速度阈值，单位为 Hz。

  此参数应设置为高于电机最低转速，通常下降到额定电机转速的 5%-30% 之间。
- **USER_MOTOR1_ACCEL_START_Hzps**

  此参数定义启动期间的电机加速度。

  此参数应设置为小于电机的最大加速度 (USER_MOTOR1_ACCEL_MAX_Hzps)，否则应根据系统性能要求进行设置。
- **USER_MOTOR1_ACCEL_MAX_Hzps**

  此参数定义电机的最大加速度。

  应根据系统性能要求和电机硬件限制设置此参数。
- **USER_MOTOR1_SPEED_FS_Hz**

  此参数定义了电机的快速启动速度阈值，单位为 Hz。

  此参数应设置为高于电机最低转速，通常下降到额定电机转速的 0.1%-5% 之间。
- **USER_MOTOR1_BRAKE_CURRENT_A**

  启用制动后，此参数定义制动状态期间的电机电流。

  根据系统性能要求，此参数应设置在额定电机电流的 10% 到 50% 之间。

- **USER_MOTOR1_BRAKE_TIME_DELAY**

  此参数定义了电机制动状态的延迟，以 5ms 为单位。

  此参数应根据系统性能要求进行设置，通常在 1-300 秒之间，转换为 5ms 周期。
- **USER_M1_STOP_WAIT_TIME_SET**

  此参数定义了从停止状态启动电机所需的最短等待时间，周期为 5ms。

  此参数应设置为大于 0.1s，并转换为 5ms 周期。
- **USER_M1_RESTART_WAIT_TIME_SET**

  此参数定义了从故障状态启动电机所需的最短等待时间，周期为 5ms。

  此参数应设置为大于 0.1s，并转换为 5ms 周期。
- **USER_M1_START_TIMES_SET**

  此参数定义了电机转子对齐状态的持续时间，以 5ms 为单位。

  此参数应设置为 1s-30s 之间，并转换为 5ms 周期。

- **电机保护参数：**
  - **USER_M1_IS_OFFSET_AD_DELTA**

    此参数定义相电流的 ADC 偏移的误差阈值。如果启用了失调电压校准，系统会检查以确保电流测量电路的所有 ADC 失调值都处于初始定义参数 (USER_M1_Ix_OFFSET_AD) 的差值范围内。如果计算出的 ADC 失调电压超出该相对于初始定义参数的差值，则表明存在软件错误或硬件故障，并且系统进入故障处理状态。使用单分流器测量时，不使用此参数。

    此参数应设置在 0 到 200（包括 0 到 200）之间，以在仍然捕捉错误的同时提供足够的范围。为了实现更低的误差差值，可能需要精度更高的检测电路。
  - **USER_M1_VA_OFFSET_SF_DELTA**

    此参数定义相电压的 ADC 偏移的误差阈值。如果启用了失调电压校准，系统会检查以确保电压测量电路的所有 ADC 失调值都在相对于初始定义参数 (USER_M1_Vx_OFFSET_SF) 的差值范围内。如果计算出的 ADC 失调电压超出该相对于初始定义参数的差值，则表明存在软件错误或硬件故障，并且系统进入故障处理状态。此参数仅用于 InstaSPIN-FOC FAST 和 InstaSPIN-BLDC。

    此参数应设置在 0 到 200（包括 0 到 200）之间，以在仍然捕捉错误的同时提供足够的范围。为了实现更低的误差差值，可能需要精度更高的检测电路。
  - **USER_M1_OVER_VOLTAGE_FAULT_V**

    此参数定义了电机和逆变器硬件的过压阈值，单位为伏特。大于该值的直流母线电压表示软件错误或硬件故障导致计算出的直流母线电压超出用户定义的安全或预期范围，系统进入故障处理状态。

    此参数应设置为小于满标量程 ADC 电压 (USER_M1_ADC_FULL_SCALE_VOLTAGE_V) 的硬件相关值。
  - **USER_M1_OVER_VOLTAGE_NORM_V**

    此参数定义了指示直流母线过压故障（请参阅 USER_M1_OVER_VOLTAGE_FAULT_V）不再存在的下限阈值 (V)。

    此参数应设置为小于过压故障阈值 (USER_M1_OVER_VOLTAGE_FAULT_V) 的硬件相关值。通常，这会比过压故障阈值低几伏特，以确保系统可靠地返回到安全或预期状态。

- **USER_M1_UNDER_VOLTAGE_FAULT_V**

  此参数定义了电机和逆变器硬件的欠压阈值，单位为伏特。大于该值的直流母线电压表示软件错误或硬件故障导致计算出的直流母线电压低于用户定义的安全或预期范围，系统进入故障处理状态。

  此参数通常应设置为与硬件相关的值，比使电机旋转所需的最低电压高几伏特。
- **USER_M1_UNDER_VOLTAGE_NORM_V**

  此参数定义了用于指示直流母线欠压故障（请参阅 USER_M1_UNDER_VOLTAGE_FAULT_V）不再存在的上限阈值。

  此参数应设置为大于欠压故障阈值 (USER_M1_UNDER_VOLTAGE_FAULT_V) 的硬件相关值。通常，这会比欠压故障阈值高几伏，以确保系统可靠地返回到安全或预期状态。
- **USER_M1_LOST_PHASE_CURRENT_A**

  此参数定义了丢失相位故障的电流阈值，单位为 A。超过此错误的最小速度阈值 (USER_M1_FAIL_SPEED_MIN_HZ) 后，任何相电流的绝对值小于该值即表示软件错误或硬件故障，系统进入故障处理状态。

  根据系统性能要求，应将此参数设置为取决于电机的值。通常，该值介于电机额定电流的 0.1% 到 10% 之间。
- **USER_M1_UNBALANCE_RATIO**

  此参数定义了指示相位失衡故障的比率阈值。如果最大与最小 RMS 相电流值之比超过该比率，则表示存在软件错误或硬件故障，系统进入故障处理状态。

  根据系统性能要求，应将此参数设置为取决于电机的值。通常，此值是介于 5% 和 25% 之间的百分比值。对于此应用而言，默认值 20% 通常应该足够了。
- **USER_MOTOR1_OVER_CURRENT_A**

  此参数定义了电机的过流阈值，以 A 表示。此值用于计算 CMPSS 外设的 DAC 值，进而触发 PWM 的跳闸区故障处理。

  此参数应设置为与电机和系统相关的值，通常为额定电机电流的 50% 至 300%。

  注意：如果此参数的值设置为大于电机的最大峰值电流值（在本例中定义为 ADC 满量程电流的 47.5%），则会忽略此参数，而使用最大峰值电流。
- **USER_M1_OVER_LOAD_POWER_W**

  此参数定义了电机和逆变器硬件的过功率阈值，以 W 为单位。在电机运行状态下，如果电机的计算功率高于此值，则表示出现软件错误或硬件故障，系统进入故障处理状态。

  此参数应设置为与电机相关的值，通常为电机额定功率的 50% 至 200%。此值必须小于电机在不超过热额定值的情况下可产生的最大输出功率。
- **USER_M1_STALL_CURRENT_A**

  当计算得出的电机转速低于速度阈值 (USER_M1_FAIL_SPEED_MIN_HZ) 时，该参数定义了 RMS 电机定子电流的电流阈值，单位为 A。当电机电流高于故障检查阈值（请参阅 USER_M1_FAULT_CHECK_CURRENT_A）时，这表示电机定子电流的最小值，而低于此参数的值则表示失败的电机启动故障情况。高于此参数的值始终表示存在电机失速故障情况。

  此参数应设置为与电机相关的值，通常为电机额定电流的 50% 至 300%。此值应低于电机的最大峰值电流，在本例中定义为 ADC 满量程电流的 47.5%。

  备注

  电机启动失败和电机失速情况不应该都处于活动状态。
- **USER_M1_FAULT_CHECK_CURRENT_A**

  此参数定义了多个电机故障的电流阈值，以 A 表示。当 RMS 电机定子电流低于该值时，不会检查不平衡相电流故障、过速故障和启动失败故障。

  根据系统性能要求，应将此参数设置为取决于电机的值。该值通常为额定电机电流的 0.1% 至 20%。

- **USER_M1_FAIL_SPEED_MAX_HZ**

  此参数定义了电机的最大速度，单位为 Hz。如果计算得出的速度高于此值，则表示存在软件错误或硬件故障，系统将进入故障处理状态。

  根据系统性能要求，应将此参数设置为取决于电机的值。此值通常为电机额定速度的 200% 至 500%。
- **USER_M1_FAIL_SPEED_MIN_HZ**

  此参数定义了电机最小速度阈值，单位为 Hz。仅在电机高于或低于该值时，才会检查某些故障。仅当电机转速低于该值时，才会检查失速电流和电机启动失败故障。仅当电机转速高于此值时，才会检查失相故障。

  根据系统性能要求，应将此参数设置为取决于电机的值。此值通常为电机额定转速的 1% 到 10%，且必须高于电机最低转速。
- **USER_M1_VOLTAGE_FAULT_TIME_SET**

  此参数定义了设置或清除过压和欠压故障的持续时间，以 5ms 为周期。如果检测到故障情况的时间超过此时间，则错误标志会被置位。如果未出现故障情况的时间超过此时间，ERROR 标志位被复位。

  应根据系统性能要求设置此参数。该值通常介于 0.02s 到 3.0s 之间。对于大多数应用而言，默认值应该足够了。
- **USER_M1_OVER_LOAD_TIME_SET**

  此参数定义了设置过载电源故障的持续时间，以 5ms 为单位。如果检测到故障情况的时间超过此时间，则错误标志会被置位。

  应根据系统性能要求设置此参数。该值通常介于 0.01s 到 1.0s 之间。对于大多数应用而言，默认值应该足够了。
- **USER_M1_STALL_TIME_SET**

  此参数定义了设置电机失速电流故障的持续时间，以 5ms 为单位。如果检测到故障情况的时间超过此时间，则错误标志会被置位。

  应根据系统性能要求设置此参数。该值通常介于 0.01s 到 2.0s 之间。对于大多数应用而言，默认值应该足够了。
- **USER_M1_UNBALANCE_TIME_SET**

  此参数定义了设置电机不平衡相电流故障的持续时间，周期为 5ms。如果检测到故障情况的时间超过此时间，则错误标志会被置位。

  应根据系统性能要求设置此参数。该值通常介于 0.05s 到 5.0s 之间。对于大多数应用而言，默认值应该足够了。
- **USER_M1_LOST_PHASE_TIME_SET**

  此参数定义了设置电机丢相电流故障的持续时间，周期为 5ms。如果检测到故障情况的时间超过此时间，则错误标志会被置位。

  应根据系统性能要求设置此参数。该值通常介于 0.05s 到 5.0s 之间。对于大多数应用而言，默认值应该足够了。
- **USER_M1_OVER_SPEED_TIME_SET**

  此参数定义了设置电机超速故障的持续时间，周期为 5ms。如果检测到故障情况的时间超过此时间，则错误标志会被置位。

  应根据系统性能要求设置此参数。该值通常介于 0.05s 到 5.0s 之间。对于大多数应用而言，默认值应该足够了。
- **USER_M1_STARTUP_FAIL_TIME_SET**

  此参数定义了设置失败电机启动故障的持续时间，周期为 5ms。如果检测到故障情况的时间超过此时间，则错误标志会被置位。

  应根据系统性能要求设置此参数。此值通常介于 1.0s 至 20.0s 之间。对于大多数应用而言，默认值应该足够了。
- **USER_M1_OVER_CURRENT_TIMES_SET**

此参数定义了设置电机过流故障的持续时间，周期为 5ms。如果检测到故障情况的时间超过此时间，则错误标志会被置位。

应根据系统性能要求设置此参数。此值通常介于 0.01s 至 0.5s 之间。对于大多数应用而言，默认值应该足够了。

- **PI 稳压器增益调整参数：**
  - *通用 PI 调优注意事项*

速度和电流调节器的典型初始 Kp 和 Ki 是根据电机参数计算的。增益的这些运行时值将根据启动后的电机运行速度或矢量电流进行更改。

PI 稳压器的增益根据以下规则设置：

Kp 增益	Ki 增益	条件
Kp* = Kp * G(p_start)	Ki* = Ki * G(i_start)	电机启动
Kp* = Kp * G(p_low)	Ki* = Ki * G(i_low)	启动后和条件为低电平
Kp* = Kp * G(p_high)	Ki* = Ki * G(i_high)	启动后和条件为高电平
Kp* = Kp * (G(p_low) + H(p_low) * (condition state))	Ki* = Ki * (G(i_low) + H(i_low) * (condition state))	没有匹配的任何其他条件

其中：

- H(p_slope) = (G(p_high) - G(p_low)) / (condition range)
- H(i_slope) = (G(i_high) - G(i_low)) / (condition range)
- 条件低电平 =
  - 速度调节器：ω(e) < ω(e_low)
  - 电流调节器：i(s) < i(s_low)
- 条件高电平=
  - 速度调节器：ω(e) > ω(e_high)
  - 电流调节器：i(s) > i(s_high)
- 条件状态 =
  - 速度调节器：ω(e) - ω(e_low)
  - 电流调节器：i(s) -i(s_low)
- 条件范围 =
  - 速度调节器：ω(e_high) - ω(e_low)
  - 电流调节器：i(s_high) -i(s_low)

- **USER_MOTOR1_GAIN_SPEED_LOW_Hz**

  此参数定义了用于调整速度 PI 稳压器 Kp* 和 Ki* 的低速阈值。

  根据电机和应用要求，此参数应该被设定在电机额定速度的 10% 至 30% 之间。
- **USER_MOTOR1_GAIN_SPEED_HIGH_Hz**

  此参数定义了用于调整速度 PI 稳压器 Kp* 和 Ki* 的高速阈值。

  根据电机和应用要求，此参数应该被设定在电机额定速度的 60% 至 100% 之间。
- **USER_MOTOR1_Kx_SPD_START_SF**

  其中 Kx 为 Kp 或 Ki，此参数定义了启动时的增益系数 Gx_start，以调整速度 PI 稳压器的 Kx*。

  根据电机和应用要求，该参数应设置在 0.1 至 2.0 之间，在大多数应用中应低于 Gx_low。

  USER_MOTOR1_Kp_SPD_START_SF；USER_MOTOR1_Ki_SPD_START_SF
- **USER_MOTOR1_Kx_SPD_LOW_SF**

  其中 Kx 为 Kp 或 Ki，此参数定义了增益系数 Gx_low，以调整速度 PI 稳压器的 Kx*。

  根据电机和应用要求，该参数应设置在 0.1 至 10.0 之间，并且通常应高于 Gx_high。

  USER_MOTOR1_Kp_SPD_LOW_SF；USER_MOTOR1_Ki_SPD_LOW_SF
- **USER_MOTOR1_Kx_SPD_HIGH_SF**

  其中 Kx 为 Kp 或 Ki，此参数定义了增益系数 Gx_high，以调整速度 PI 稳压器的 Kx*。

  根据电机和应用要求，该参数应设置在 0.1 至 5.0 之间，并且通常应低于 Gx_low。

  USER_MOTOR1_Kp_SPD_HIGH_SF；USER_MOTOR1_Ki_SPD_HIGH_SF
- **USER_MOTOR1_GAIN_IQ_LOW_A**

  此参数定义低电流阈值 is_low 以调整 q 轴电流 PI 控制器的增益。

  根据系统性能、电机和应用要求，此参数应设置为电机额定电流的 10% 至 50%。
- **USER_MOTOR1_GAIN_IQ_HIGH_A**

  此参数定义高电流阈值 is_high，以调整 q 轴电流 PI 控制器的增益。

  根据系统性能、电机和应用要求，此参数应设置为电机额定电流的 50% 至 100%。
- **USER_MOTOR1_Kx_IQ_START_SF**

  其中 Kx 为 Kp 或 Ki，该参数定义了增益系数 Gx_start，用于在启动期间调整 q 轴电流 PI 控制器的增益。

  根据系统性能、电机和应用要求，此参数应设置为 0.1 至 5.0。

  USER_MOTOR1_Kp_IQ_START_SF；USER_MOTOR1_Ki_IQ_START_SF
- **USER_MOTOR1_Kx_IQ_LOW_SF**

  其中 Kx 为 Kp 或 Ki，该参数定义了增益系数 Gx_low，用于在启动期间调整 q 轴电流 PI 控制器的增益。

  根据系统性能、电机和应用要求，该参数应设置在 0.1 至 10.0 之间，并且应高于 Gx_high。

  USER_MOTOR1_Kp_IQ_LOW_SF；USER_MOTOR1_Ki_IQ_LOW_SF
- **USER_MOTOR1_Kx_IQ_HIGH_SF**

  其中 Kx 为 Kp 或 Ki，该参数定义了增益系数 Gx_high，用于在启动期间调整 q 轴电流 PI 控制器的增益。

  根据系统性能、电机和应用要求，该参数应设置在 0.1 至 5.0 之间，并且应低于 Gx_low。

  USER_MOTOR1_Kp_IQ_HIGH_SF；USER_MOTOR1_Ki_IQ_HIGH_SF
- **USER_MOTOR1_Kx_ID_SF**

  其中 Kx 为 Kp 或 Ki，此参数定义了增益系数 Gx，用于调整 d 轴电流 PI 控制器的增益。

  根据系统性能、电机和应用要求，此参数应设置为 0.1 至 5.0。

USER_MOTOR1_Kp_ID_SF；USER_MOTOR1_Ki_ID_SF

- **电位器控制参数：**
  - **USER_M1_POT_ADC_MIN**

    启用电位器速度控制后，此参数定义电位器允许的最小 ADC 值。如果 ADC 读数低于该值，则将电机转速设置为 0。

    此参数的设置应确保在打算禁用电机时 ADC 噪声不会导致大于此值。
  - **USER_M1_POT_ADC_MAX**

    启用电位器速度控制后，此参数定义电位器允许的最大 ADC 值。如果 ADC 读数高于此值，则电机转速被设置为最大值。

    此参数应设置为 USER_M1_POT_ADC_MIN 的最大可能 ADC 值（本例中为 4096）偏移。在本例中，即 4096U-200U。
  - **USER_M1_POT_SPEED_SF**

    启用电位器速度控制后，该推导参数定义 ADC 读数与电机转速之间的精确关系。

    此参数根据最小和最大电位器 ADC 值（USER_M1_POT_ADC_MIN 和 USER_M1_POT_ADC_MAX）以及电机最大速度 (USER_MOTOR1_FREQ_MAX_Hz) 得出。
  - **USER_M1_POT_SPEED_MIN_Hz**

    启用电位器速度控制后，此推导参数定义了电机的最小频率，单位为 Hz。如果电位器 ADC 读数产生低于此速度的非零值，则电机转速改为该值。

    此参数设置为电机最大速度的 10% (USER_MOTOR1_FREQ_MAX_Hz)。
  - **USER_M1_POT_SPEED_MAX_Hz**

    启用电位器速度控制后，此推导参数定义了电机的最大频率，单位为 Hz。如果电位器 ADC 读数产生高于此速度的值，则电机转速改为该值。

    此参数设置为电机最大速度的 50% (USER_MOTOR1_FREQ_MAX_Hz)。
  - **USER_M1_WAIT_TIME_SET**

    启用电位器速度控制时，此参数定义 1ms 内速度更新之间的等待时间。

    此参数应设置为允许快速调整的值，同时不允许噪声对计算产生过度影响。默认值为 500ms，这对于大多数应用而言已经足够了。
- **速度脉冲控制参数：**
  - **USER_M1_SPEED_CAP_MIN_Hz**

    启用 cmd 脉冲速度控制后，允许通过输入 GPIO 上的速度脉冲计算电机转速，此参数定义了电机的最小频率，以 Hz 为单位。当计算得出的电机转速低于此值时，电机转速改为 0。

    此参数应根据电机的最小转速进行设置，默认情况下不会自动获得。
  - **USER_M1_SPEED_CAP_MAX_Hz**

    启用 cmd 脉冲速度控制后，允许通过输入 GPIO 上的速度脉冲计算电机转速，此参数定义了电机的最大频率，以 Hz 为单位。当计算得出的电机转速高于此值时，电机转速改为 0。

    此参数应根据最大额定电机转速进行设置，默认情况下不自动获得。

    备注

    计算出的速度高于最大值并不会像电位器控制模式下那样将速度设置为最大值，而是会禁用电机。
  - **USER_M1_CAP_WAIT_TIME_SET**

    启用 cmd 脉冲速度控制后，允许通过输入 GPIO 上的速度脉冲计算电机转速，此参数定义了速度更新之间的等待时间为 1ms。

    此参数应设置为允许快速调整的值，同时不允许噪声对计算产生过度影响。默认值为 200ms，这对于大多数应用而言已经足够了。

- **开关控制参数：**
  - **USER_M1_SWITCH_WAIT_TIME_SET**

    启用 cmd 开关控制时，为了允许使用开关来启用或禁用电机，此参数定义了 1ms 内开关状态更新之间的最小等待时间。

    此参数应设置为允许快速调整的值，同时不允许噪声对计算产生过度影响。默认值为 50ms，这对于大多数应用而言已经足够了。

参考资料

- 德州仪器 (TI)：*C2000™ 实时控制微控制器 (MCU) 使用入门*
- 德州仪器 (TI)：*使用 C2000 实时微控制器的基本开发指南*
- 德州仪器 (TI)：*TMS320F28xx 和 TMS320F28xxx 的硬件设计指南*
- 德州仪器 (TI)：*C2000 MCU JTAG 连接调试*
- 德州仪器 (TI)：*使用 PWM 输出作为 TMS320F280x 上的数模转换器*
- 德州仪器 (TI)：*使用单一直流链路分流器的 PMSM 无传感器 FOC*
- 德州仪器 (TI)：*C2000™ 软件频率响应分析器 (SFRA) 库用户指南*
- 德州仪器 (TI)：*C2000Ware 电机控制 SDK 入门指南*
- 有关 C2000 实时 MCU 的一般信息 - C2000™ 概述
- C2000 产品 - C2000™ 产品
- C2000 设计和开发资源 - C2000™ 设计和开发

修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision * (September 2021) to Revision A (April 2024)	Page
• 添加了节 2	3
• 更新了 节 3.1：添加了对以下 TI 硬件的支持：F28003x LP & CC、F280013x LP & CC 和 DRV8329AEVM。	24
• 更新了节 3.2。添加了对以下 TI 硬件的支持：F28003x LP & CC、F280013x LP & CC 和 DRV8329AEVM。	27
• 更新了节 3.4。为清晰起见，对说明进行了调整。	55
• 更新了节 3.5 为清晰起见，对说明进行了调整。	60
• 更新了节 4。为清晰起见，对说明进行了调整。	83
• 添加了附录 A	103

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265