InstaSPIN-FOC™ 和 InstaSPIN-MOTION™

用户指南

Literature Number: ZHCU083F
January 2013–Revised July 2014

内容

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

附图目录

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

附表目录

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

简介

欢迎并感谢您选择德州仪器 (TI) InstaSPIN™ 解决方案。 本文档将提供 InstaSPIN 的详细技术信息，指导您将此解决方案集成到应用中。 本文档的结构汇总如下：

- 简介
  - InstaSPIN-FOC™ 和 FAST™
  - InstaSPIN-MOTION™ 和 SpinTAC™
- 利用 TI 硬件和软件直接运行电机
- 了解软件详细信息，从回顾 API 函数调用到状态图再到调整速度和位置控制环路
- 了解直接影响 InstaSPIN 性能的硬件部分。

上述提供的所有内容均有助于您使用 InstaSPIN-FOC 或 InstaSPIN-MOTION 软件开发出成功的产品。 各个示例项目（实验）是取得成功的关键部分，专为本文档中的主题而设计。 它们不仅可以用于 InstaSPIN 实验，还可用作设计参考。 有关最新的 InstaSPIN-FOC 和 InstaSPIN-MOTION 解决方案与设计资源，以及实用视频，请参见 http://www.ti.com/instaspin。

有关本文档中使用的术语的定义，请参见本文档末尾的附录 A。 以下为最常用的术语：

- FOC：
  - 磁场定向控制
- InstaSPIN-FOC：
  - 通过特定器件（FAST 观测器，FOC，速度和电流环路）上的 TI 片上 ROM 提供完整的无传感器 FOC 解决方案，无需使用任何机械转子传感器即可有效控制电机。
- FAST
  - 统一观测器结构，此结构充分利用那些使用磁通量进行能量转换的所有电机之间的相似性，自动识别所需的电机参数并提供电机反馈信号：磁通、磁通角、电机转轴转速和转矩。
- SpinTAC 运动控制套件：
  - 包括一个高级速度和位置控制器，一个运动引擎，和一个运动序列规划器。 SpinTAC 抗扰速度控制器主动估算且实时补偿系统干扰，从而提升总体产品性能。 SpinTAC 运动电机根据用户定义的参数来计算理想基准信号（具有前馈）。 SpinTAC 支持标准工业曲线以及 LineStream 的专有“平滑轨迹”曲线。 SpinTAC 运动序列规划器运行用户定义的状态转换图，从而轻松设计复杂运动序列。
- InstaSPIN-MOTION:
  - 一款无传感器或传感 FOC 综合解决方案，用于电机控制、运动控制、速度控制和位置控制。 此解决方案以最高效率为运行在不同运动状态转换中的电机应用提供稳健耐用的系统性能。 InstaSPIN-MOTION 包含 FAST 统一软件观测器，可与 LineStream Technologies 公司的 SpinTA 运动控制套件配套使用。
- MotorWare™ 软件：
  - TI 提供的电机控制可扩展软件架构，InstaSPIN-FOC 是其中一部分。

InstaSPIN-FOC 和 InstaSPIN-MOTION 软件可在 TMS320F2806xF、TMS320F2086xM、TMS320F2802xF、TMS320F2805xF 和 TMS320F2805xM 器件系列上使用，未来计划开发更多版本以便在更多器件上使用。 有关更多详细信息，请参见器件专用数据表和器件专用技术参考手册 (TRM)。 InstaSPIN TRM 含有 TI 电机实验室测得的最新性能数据。 本文档（即 《*InstaSPIN 用户指南*》）的不同之处在于它是介绍“如何”在应用中使用 InstaSPIN-FOC 或 InstaSPIN-MOTION 的功能性指南。

无论您是使用 TI 提供的逆变器和电机，还是使用自己的逆变器和电机，本文档都可帮助您了解 TI 的这项功能强大的新型解决方案。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

1.1 InstaSPIN-FOC 和 FAST 概述

TMS320F2806xF（69F、68F 和 62F — 80 或 100 引脚封装）、TMS320F2802xF（26F 和 27F — 48 引脚封装）和 TMS320F2805xF（54F 和 52F — 80 引脚封装）是德州仪器 (TI) 首批包含级联速度和转矩环路所需的 FAST (图 1-1) 评估器和其它电机控制功能的器件，可实现有效的三相磁场定向电机控制 (FOC)。

它们与 F2806xF、F2805xF 和 F2802xF 外设驱动程序一起可构成一个无传感器（也称为自感） InstaSPIN-FOC 解决方案，此解决方案能够识别、调整转矩控制器，并能在几分钟内有效控制电机，而且无需使用任何机械转子传感器。 整个软件包被称为 InstaSPIN-FOC，在 ROM 中提供。 对于 F2806xF 器件，ROM 中包含 FAST 估算器和 FOC 块；对于 F2802xF 器件，ROM 中仅包含 FAST 估算器；而对于 F2805xF 器件，ROM 中包含 FAST 估算器和 FOC 块。 对于 F2806xF 器件，用户还可以选择在用户存储器（闪存或 RAM）中执行所有 FOC 功能，这需要调用 ROM 中的专有 FAST 估算器固件。 对于 F2805xF 器件，用户还可以选择在用户存储器（闪存或 RAM）中执行所有 FOC 功能，这需要调用 ROM 中的专有 FAST 估算器固件。 对于 F2802xF 器件，所有 FOC 块均通过用户存储器（闪存或 RAM）加载和执行，而估算器则通过 ROM 运行。InstaSPIN-FOC 的设计可以灵活适应广泛的系统软件架构。 这种灵活性的范围如图 1-2 和 图 1-3 所示。

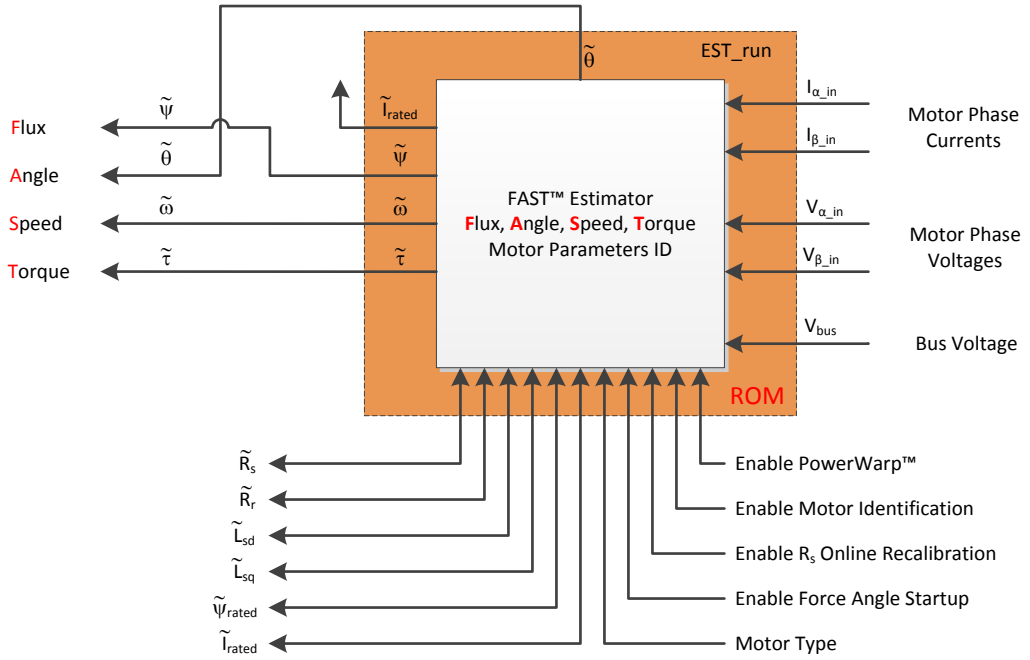


图 **1-1. FAST -** 估算磁通、角度、转速、转矩 **-** 自动电机识别

1.1.1 FAST 估算器特性

- 充分利用所有使用磁通量进行能量转换的电机间的相似性的统一观测器结构
  - 可实现同步电机（无刷直流 (BLDC)，永磁同步电机 (SPM)，内部永磁电机 (IPM)），和异步电机（交流感应电机 (ACIM)）控制
  - 针对内部永磁电机的凸极补偿：观测器在 Ls-d 和 Ls-q 被提供时正确跟踪转子磁通和角度
- 用于控制系统中的独特、高质量反馈信号
  - 用于稳定磁通监视和场强减弱的高质量 磁通信号
  - 与独立于 ACIM 全部转子参数的传统观测器技术相比，在更宽的速度范围内具有出色的转子磁通角估算精度
  - 实时低噪声电机转轴 速度信号
  - 针对负载监视和失衡检测的准确高带宽 转矩信号
- 角度估算器在施加的波形的第一个周期内收敛，与速度无关。
- 在全部功率象限内稳定运行，其中包括发电机象限
- 满转矩时，在低于 1Hz（典型值）的稳定状态速度下可准确估算角度
- 即使在经过零速度的低速反向期间也能保持角度完整性
- 在停转情况下保持角度完整性，从而实现平滑的停转恢复
- 电机识别过程可在 2 分钟（典型值）内测量空载电机所需的电机参数
- “高速数据传输错误纠正 (On-the-fly)”定子电阻重校准（在线 Rs）实时跟踪定子电阻变化，从而实现温度范围内的稳定运行。 这个特性也可被用作电机绕组的温度传感器（需要基点校准）。
- 大大优于传统观测器的转子磁通角度跟踪的瞬态响应
- PowerWarp™ 自适应减少流耗以最大限度地降低组合（转子和定子）铜损耗，而不会影响 ACIM 输出功率级。

1.1.2 InstaSPIN-FOC 解决方案特性

- 包括测量无传感器 FOC 系统中转子磁通（幅度和角度）的 FAST 估算器
- 自动转矩（电流）环路调整，具有用户调节选项
- 自动配置速度环增益（Kp 和 Ki）可为绝大多数应用提供稳定运行，还可提供最优瞬态响应所需的用户调节
- 自动或手工场强减弱和场强增强
- 总线电压补偿
- 自动偏移校准确保反馈信号的高质量采样样本

1.1.3 InstaSPIN-FOC 方框图

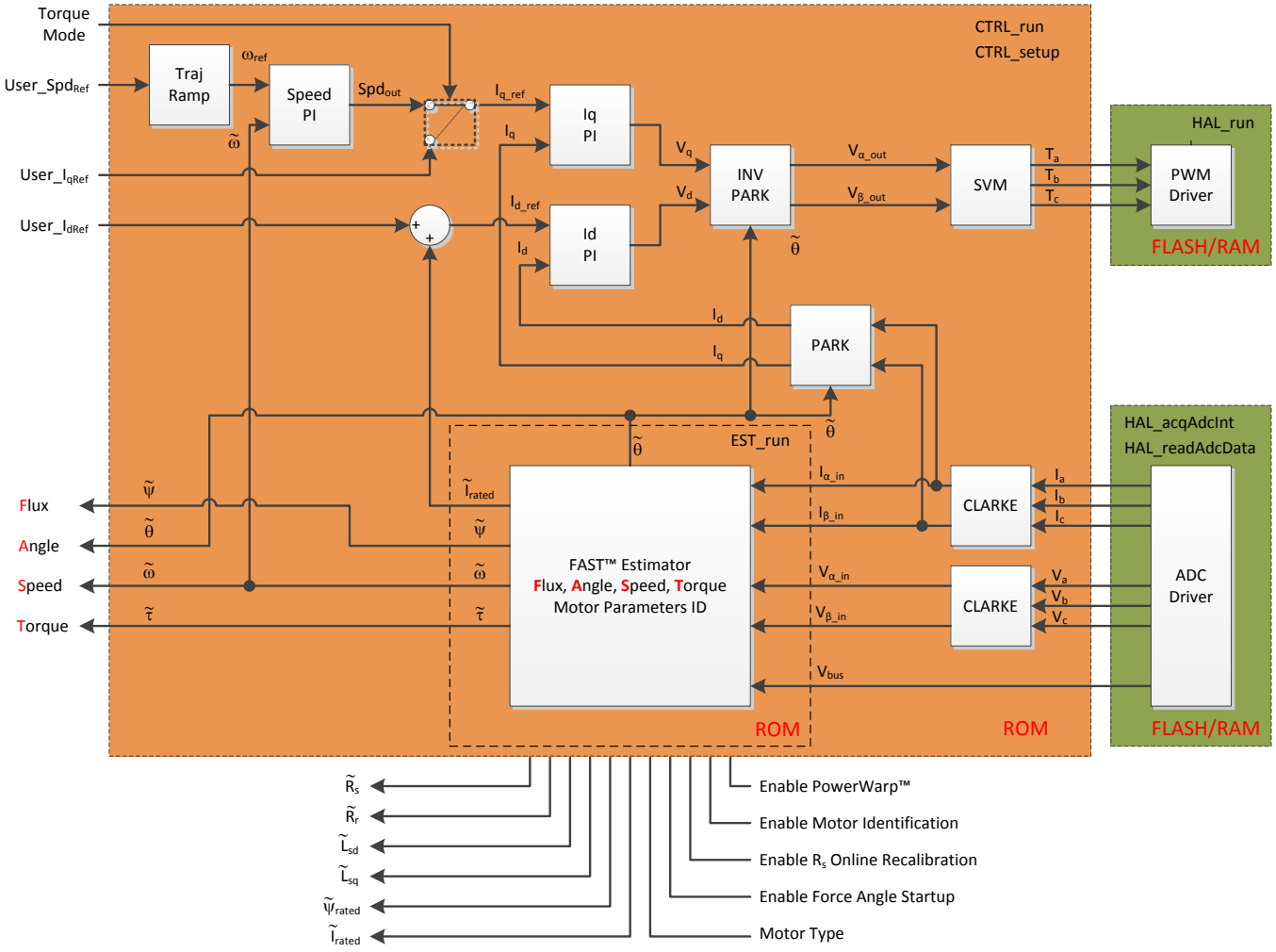


图 **1-2. i. ROM** 中整个 **InstaSPIN-FOC** 包的方框图（**F2802xF** 器件除外）

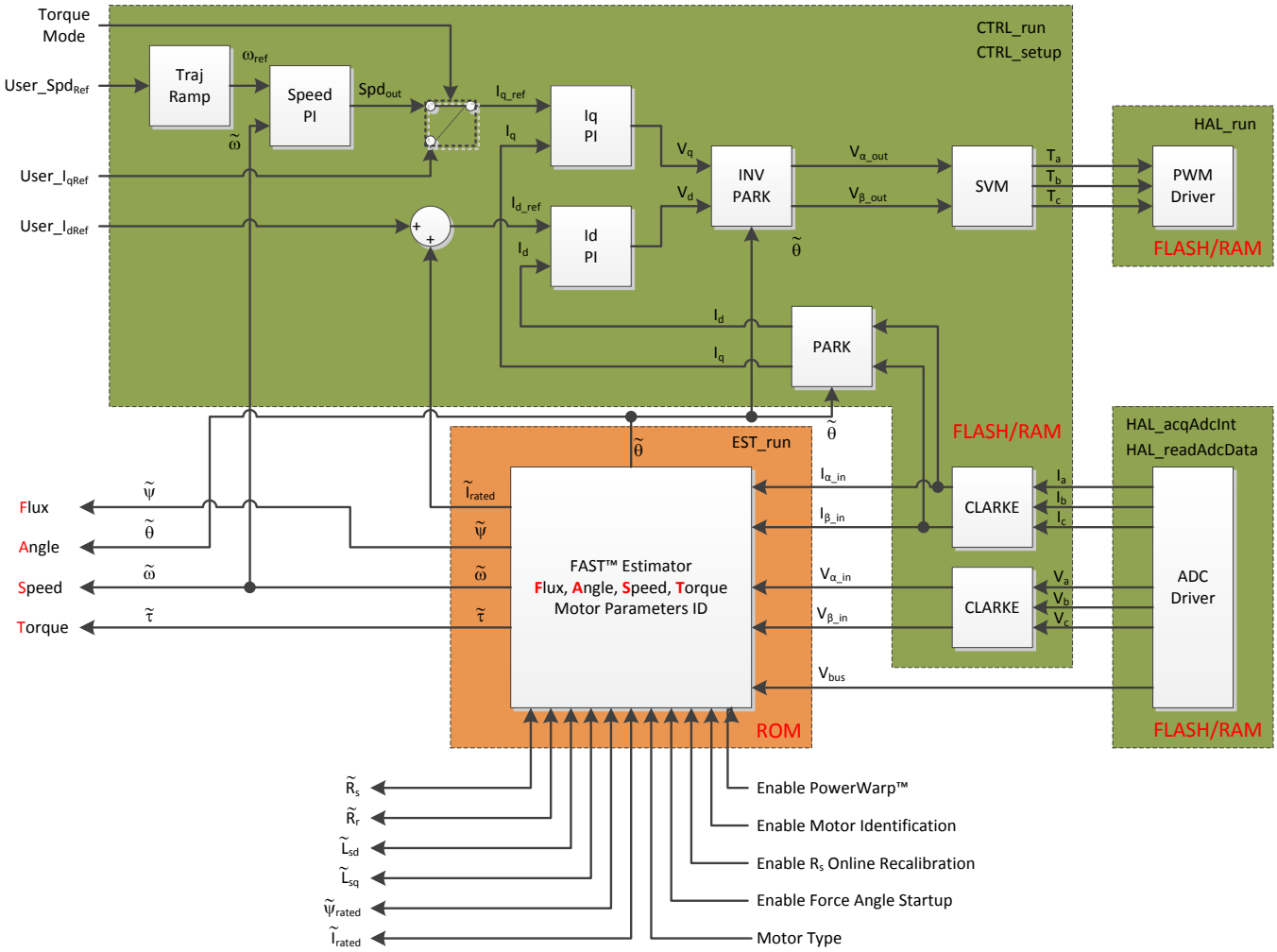


图 **1-3.** 用户内存中的 **InstaSPIN-FOC** 方框图，不包括 **ROM** 中的 **FAST**

1.1.4 将 FAST 估算器与典型解决方案相比较

表 1-1. FAST 估算器与 典型解决方案对比

主题	典型软件传感器和 FOC 解决方案	TI 的 Fast 估算器和 InstaSPIN-FOC 解决方案
电机参数	基于电机模型的观测器在很大程度上取决于电机参数	依赖更少的电机参数，电机的离线参数识别 – 无需数据表！定子电阻的在线参数监视和重新估算
估算器调整	针对每个电机的转速/负载完成多次复杂的观测器调整	无需估算器调整。 一旦识别了电机参数，在转速/转矩动态范围内每次的运行方式都一样
估算器准确度	角度跟踪性能通常只在频率超过 5-10Hz 时才令人满意，这是因为在更高速度时会遇到问题，并且要对场强减弱进行补偿；动态性能受到观测器手动调整的影响；电机停转通常使观测器损毁	FAST 提供可靠的角度跟踪，此角度跟踪在施加波形的一个电周期内收敛，并且能够在频率低于 1Hz 时进行跟踪（取决于模拟感应的质量和分辨率）。角度跟踪展现出出色的瞬态响应（即使在突发负载瞬变导致电机停转时也是如此，这样可以满转矩实现受控重启）。
启动	很难从零速启动或者无法从零速启动。零速时的观测器反馈是不稳定的，这将导致较差的转子角度精度和速度反馈信号。	InstaSPIN-FOC 包括： • 具有强制角的零速度启动 • 启动时 100% 转矩 • FAST 转子磁通角跟踪在一个电周期内收敛 FAST 在通过零速度时完全稳定，从而提供准确速度和角度估计。
电流环路	调整 FOC 电流控制具有挑战性 - 对于新手尤其如此	根据确定的参数自动设定电流控制器的初始调整。 如有需要，用户可以更新增益或使用自己的控制器。完全调整观测器和转矩控制器的识别过程所花费的时间少于 2 分钟
反馈信号	未管理系统偏移和漂移	FAST 包括硬件/软件自动校准和偏移补偿。FAST 需要 2 相电流（针对 100% 和过调制时需要 3 相）、3 相电压来支持完全动态性能，针对电流控制器纹波补偿还需要直流总线电压。FAST 包括一个在线定子电阻跟踪算法
电机类型	针对多个电机的多项技术：标准反电势、滑动模式、凸极跟踪、感应磁通估算器，或“混合模式”观测器	FAST 能够与全部 3 相电机类型、同步和异步电机配套使用，无需考虑负载动态性能。 支持具有不同的 Ls-d 和 Ls-q 的凸极 IPM 电机。包括用于感应电机的 PowerWarp，可实现节能
场强减弱	场强减弱区域对于观测器具有挑战性 - 由于反电势信号变得太强，跟踪性能和稳定性会受到影响	凭借宽范围内磁通估算的稳定性，FAST 估算器支持场强减弱或场强增强应用
电机温度	角度跟踪性能随定子温度的变化而降低	在线定子电阻重校准可提升角度估算的准确性
速度估算	不良的速度估算会导致 FOC 系统损失效率并且动态运行的稳定性下降	高质量低噪声速度估算器，包括感应电机的转差率计算
转矩估算	通常需要转矩和振动传感器	高带宽电机转矩估算器

1.1.5 FAST 提供无传感器 FOC 性能

1.1.5.1 FAST 估算器取代机械传感器

电机磁场定向控制 (FOC) 可实现出色的转矩控制、更低的转矩纹波，相对于传统交流控制技术，这项技术在很多情况下还可提升效率。 为了实现最佳动态响应，转子磁通基准控制算法更适用于定子磁通基准技术。为了正确运行，这些系统需要知道相对于定子框（通常为相位 A 定子线圈的磁轴）上一个固定点的转子磁通空间角。 传统上，这已经由安装在电机转轴上的机械传感器（例如，编码器或解算器）完成。 这些传感器提供出色的角反馈，但是给系统设计带来很大负担。 如下面 3.5.4 节 中讨论且图示的那样，传感角反馈对系统产生 6 个主要影响：

1. 传感器本身非常昂贵（一个优质解算器的价格通常超过 2500 美元，而大容量集成编码器的价格仅几美元）
2. 安装传感器要求具有娴熟的组装技巧，这会增加劳动力成本
3. 传感器通常需要独立的电源，这会增加系统成本并降低稳定性
4. 传感器是系统中最精密的组件，它可影响系统稳定性，在恶劣环境应用中更是如此
5. 传感器反馈信号通过连接器接回至控制器电路板，这也会增加系统成本并大幅降低稳定性，具体取决于所用的连接器类型。
6. 将传感器信号传回控制器所需的电缆对系统设计人员提出了多重挑战。

- 所用电缆产生额外成本，特别是当电机和控制器之间的距离较远时
- 对于噪声源的敏感性，具有特殊屏蔽层或双绞线的电缆会增加成本
- 出于安全考虑，传感器和相关电缆必须接地。 隔离这些信号通常需要增加额外成本，特别是在处理传感器信号的处理器未接地的情况下

在电机处于封闭环境中的某些应用中（例如，压缩机），由于通过外壳接入馈线会增加成本，因此传感解决方案并不实用。 由于这些原因，FOC 系统设计人员很愿意通过处理那些可在控制器电路板上获得的信号来完全免除对传感器的需要，并且获得转子磁通角信息。 对于同步机器，大多数与执行电机软件模型相关的技术受到控制，以估计反电势波形（转子磁通），然后对这些感测到的波形进行处理，以便提取转子转轴角的估计值，并且推导出它的速度。 对于异步机器，过程有一些复杂，这是因为软件模型（观测器）还必须解决存在于转子和转子磁通之间的转速差。

然而，在这两个情况下，由于反电势波形的振幅与电机速度直接成比例，性能在较低速时会受到影响（假定无磁通减弱）。 由于反电势振幅下降至噪底，或者如果模数转换器 (ADC) 的分辨率不能如实地复制小型反电势信号，那么信号估计失败，并且电机驱动性能受到影响。

为了解决低速问题，我们开发了依靠高频注入的新技术，以角度函数（即，磁凸极）的形式测量磁异，从而实现低至零速的准确角度重建。 然而，这会引入另外一组控制问题。 首先，凸极信号对于异步电机是不存在的，而且对于大多数同步机器来说，这个信号的值也很小（特别是那些具有表面贴装转子磁体的机器）。 对于那些表现出超强凸极信号的电机（例如，IPM 电机），信号经常相对于转子角发生偏移（以负载函数表示），必须对此偏移进行补偿。 最后，这个角度测量技巧只在速度较低时起作用，此时，电机基频不会干扰询问频率。 此控制系统必须创建一个混合控制策略，在低速时使用高频注入跟踪，然后在标称和高速时移动进入基于反电势的观测器。

借助于任一技巧，产生一个稳定软件传感器的过程也是十分具有挑战性的，这是因为从本质上说，这个电机模型（观测器）是其自己的控制系统，此控制系统需要在使用范围内根据每个电机进行调整。 这个调整必须由一个稳定的正向控制环路完成。 所需的是一个稳定的转矩（通常为速度）环路来调整观测器，但是您如何在不具有可用观测器的情况下预先调整您的正向控制呢？ 一个选择就是使用一个用于反馈的机械传感器来创建稳定电流和速度环路，然后调整与机械传感器并联的软件传感器。 然而，机械传感器的使用通常是不现实的。 这一问题已经推迟了用于控制无传感器 FOC 的软件传感器的上市。

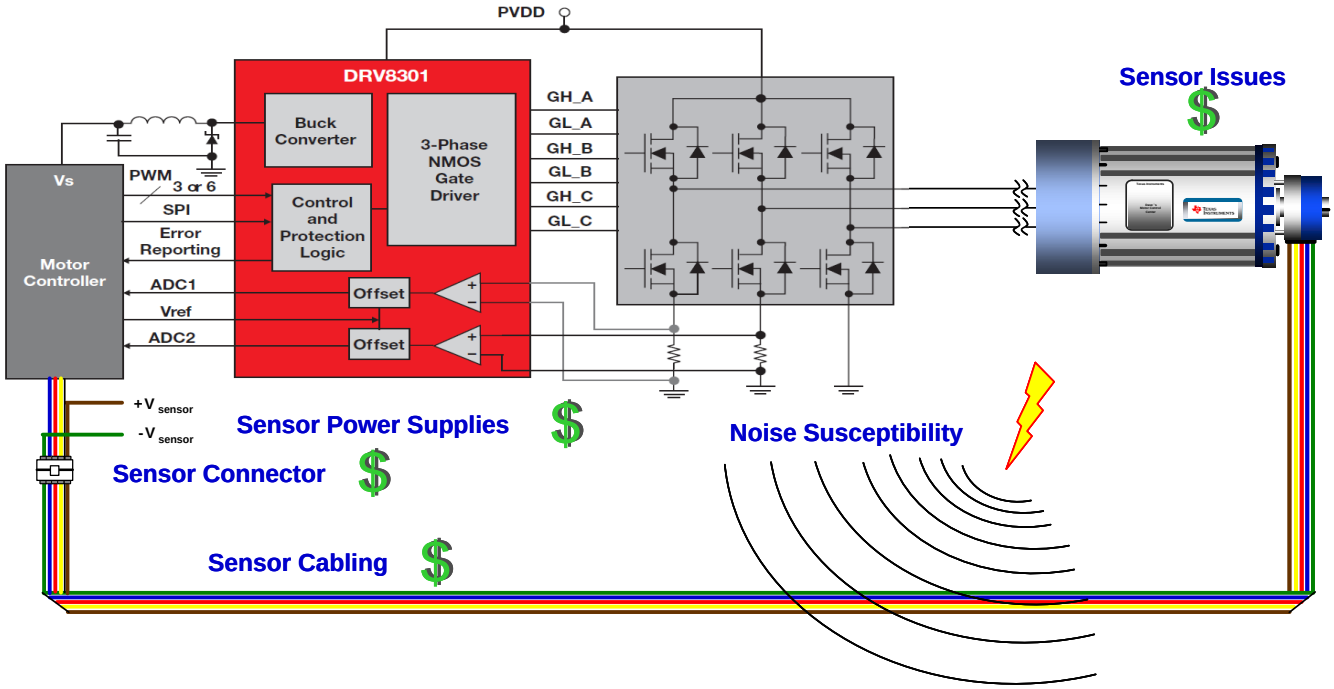


图 1-4. 传感 FOC 系统

总之，这些现有的解决方案全都受到不同问题的影响，其中包括：

- 较差的低速性能（反电势和滑模观测器 (SMO)）
- 较差的高速性能（凸极观测器）
- 不良的动态响应
- 计算强度（多模观测器）
- 参数灵敏度
- 需要观测器调整

无传感器控制革命的最新创新是 InstaSPIN-FOC。 在嵌入在 TI 处理器内的片上 ROM 中提供 C 语言的可调用库，InstaSPIN-FOC 被创建用来解决全部这些问题，以及更多问题。 它减少了系统成本和开发时间，同时提升了三相可变速度电机系统的性能。 这主要通过将机械传感器替代为专有 FAST 估算器来实现。 FAST 是一款估算器，它能够：

- 与所有三相电机高效协同工作，并将同步/异步、凸极/非凸极和永磁/非永磁/感应磁体之间的差异考虑在内
- 对于多种应用，大幅提升了整个工作频率和负载范围内的性能和稳定性
- 解决了传统 FOC 系统的手工调整困难：
  - 观测器和估算器，完全免去了所需调整
  - 电流环路稳压器，大大减少了所需调整
- 消除或减少了电机参数变化的影响
- 在少于 2 分钟的时间里，为大多数电机自动设计一个稳定且实用的控制系统。

1.1.5.2 对性能十分关键的转子角准确度

为什么很多使用机械传感器的应用需要转子磁通角的精确估算？

对于三相电机的高效控制，其目的是在*定子*上创建一个旋转磁通矢量，此定子与相对于转子的*理想方向*对齐，这样，*转子*磁场在创建所需转矩并使用最少量电流的同时跟随定子磁场。

- 定子：电机上连接至由微控制器控制的逆变器的固定部分
- 理想方向：对于非凸极同步电机，为 90 度；对于凸极电机，略大于 90 度，而对于异步电机则略小于 90 度，因为电流矢量的一部分也被用来产生转子磁通
- 转子：电机的旋转部分，在转轴上产生工作转矩

为了实现这一目的，您需要从电机上获得以下信息：

- 每相所耗电流
- 转子磁通磁场的精确相对角（通常在 ±3 度电角以内），以便能够正确定向定子磁场
- 对于速度环路，您还需要知道转子速度。

1.2 InstaSPIN-MOTION 和 SpinTAC 概述

InstaSPIN-MOTION [TMS320F2806xM（69M 和 68M — 80 引脚或 100 引脚封装）和 TMS320F2805xM（54M 和 52M — 80 引脚封装）] 是德州仪器 (TI) 的特色器件，首次将 TI 32 位 C2000™ Piccolo™ 微控制器与综合电机、运动、速度、和位置控制软件组合在一起。 InstaSPIN-MOTION 以最高效率为运行在不同运动状态转换中的电机应用提供稳健耐用的速度和位置控制。 InstaSPIN-MOTION 是您自己的单片运动控制专家。

InstaSPIN-MOTION 是一款无传感器或传感 FOC 解决方案，此解决方案能够在几分钟的时间内识别、调整并且控制电机。 InstaSPIN-MOTION 具有 FAST 优质软件传感器和 SpinTAC 运动控制套件（图 1-5）。 核心算法被嵌入到 TI 32 位 C2000 Piccolo 微控制器 (MCU) 上的只读存储器 (ROM) 内。

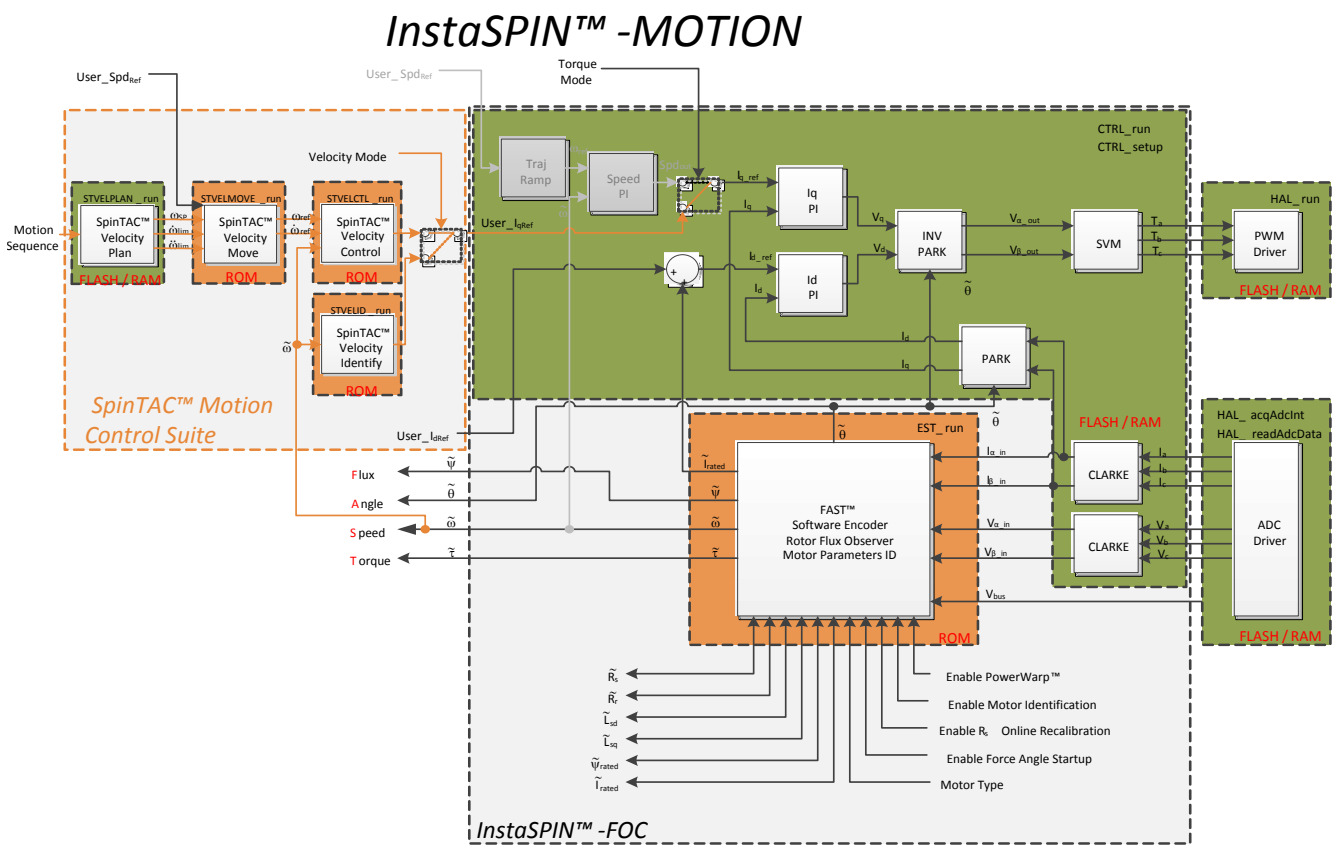


图 1-5. InstaSPIN-MOTION = C2000 Piccolo 微控制器 + FAST 软件传感器（可选）+ 自动调整内部转矩控制器 + SpinTAC 运动控制套件

InstaSPIN-MOTION 十分适合于需要精确速度和位置控制，最小干扰的应用，或者要进行多状态变换或经历动态速度或负载变化的应用。

表 1-2 提供将大大得益于 InstaSPIN-MOTION 的应用示例。

表 1-2. InstaSPIN-MOTION 应用示例

应用特性	示例
精确的速度控制	• 工业用风扇 • 传送带系统 • 升降机/自动扶梯 • 汽车车体部件（电动车窗、天窗等） • 光盘驱动器/硬盘驱动器 • 医用混合
精确的位置控制	• 监控系统 • 封装系统 • 医用机器人 • 平衡环系统 • 纺织/缝纫机器
最小干扰	• 牙科工具 • 电动工具 • 安全门

表 1-2. InstaSPIN-MOTION 应用示例 (continued)

应用特性	示例
进行多状态变换/动态变化	• HVAC 泵、风扇和风机 • 发电机 • 空调压缩机 • 洗衣机 • 运动器材 • 医用泵

1.2.1 InstaSPIN-MOTION 关键功能和优势

InstaSPIN-MOTION 用最大限度提高系统性能且尽可能减少设计工作的解决方案来取代低效率、之前的设计技术。 通过嵌入与芯片相关的电机专业技术，InstaSPIN-MOTION 使得用户能够专注于优化他们的应用，而不是纠结于运动控制。

InstaSPIN-MOTION 提供以下核心功能：

- FAST 统一软件观测器，此观测器充分利用那些使用磁通量用于能量转换的所有电机之间的相似性。FAST 估算器测量转子磁通（幅度，角度和速度）以及一个无传感器 FOC 系统中的转轴转矩。
- 电机参数识别，被用来调整 FAST 软件观测器，并且初始化针对 FOC 系统的 Iq 和 Id 控制的最内部电流（转矩）PI 控制器。
- SpinTAC，LineStream Technologies 公司的综合性运动控制套件（请见图 1-6），简化了调整并且确保动态速度和位置范围内的最优性能。

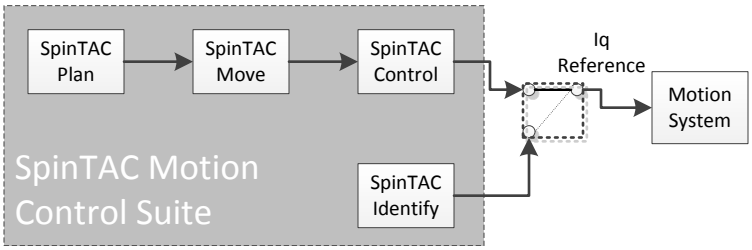


图 1-6. SpinTAC 运动控制套件组件

1.2.1.1 FAST 统一软件观测器

FAST 统一观测器结构充分利用那些使用磁通量用于能量转换的所有电机之间的相似性：

- 支持同步电机（BLDC、SPM、IPM）和异步电机 (ACIM) 控制。
- 针对内部永磁电机提供凸极补偿：在已提供 Ls-d 和 Ls-q 时观测器可正确跟踪转子磁通和角度。

FAST 为控制系统提供独特的高质量电机反馈信号：

- 用于稳定磁通监视和场强减弱的高质量磁通信号。
- 与独立于 ACIM 全部转子参数的传统观测器技术相比，在更宽的速度范围内具有出色的转子磁通角估算精度。
- 实时低噪声电机转轴速度信号。
- 针对负载监视和失衡检测的准确高带宽转矩信号。

FAST 可替代机械编码器和解算器，加快控制系统设计：

- 施加波形第一个周期内的角度估算器收敛，与速度无关。

- 全部功率象限内的稳定运行，其中包括发电机象限。
- 满转矩时，稳定状态速度下（典型值低于 1Hz）的准确角度估算。
- 即使在经历零速度的低速反向期间也可保持角度完整性。
- 停转情况下保持角度完整性，从而实现平稳的停转恢复。
- 电机识别可在 2 分钟（典型值）内测量空载电机所需的电机参数。
- *高速数据传输错误纠正 (On-the-fly)* 定子电阻重校准（在线 Rs）实时跟踪定子电阻变化，从而实现温度范围内的稳定运行。 这个特性也可被用作电机绕组的温度传感器（需要基点校准）。
- 大大优于传统观测器的转子磁通角度跟踪的瞬态响应。
- PowerWarp自适应减少流耗以最大限度地将组合（转子和定子）铜损耗降到最低，而不会影响 ACIM 输出功率级。

1.2.1.2 SpinTAC 运动控制套件

SpinTAC 最大限度地减少您在定义电机旋转方式时所花费的时间，并且确保您的电机以最佳水平运转，从而实现理想性能。 关键优势包括：

- **经简化的调整** - 使用单个可轻松评估的参数即在整个位置和速度运行范围内调整系统。
- **直观轨迹规划** - 轻松设计和执行复杂运动序列。
- **机械可靠运动** - 根据您系统的机械限制来优化速度转换。
- **理想控制** - 基于 LineStream 公司已获专利的主动抗扰控制，受益于市面上最准确的速度和位置控制。

在 SpinTAC 运动控制套件中包含四个组成部分：Identify（识别），Control（控制），Move（移动）和 Plan（规划）。 这些组成部分中的每一个都用于速度和位置解决方案。

1.2.1.2.1 IDENTIFY

SpinTAC Identify 估算惯性（一个物体围绕轴旋转加速的阻力）。 系统的惯性越大，电机加速或减速所需的转矩也就越大。 SpinTAC 控制器使用系统的惯性值来提供最准确的系统控制。 SpinTAC Identify 通过使电机在应用中旋转并且测量反馈来自动测量系统惯性。

1.2.1.2.2 CONTROL

SpinTAC Control 是一款高级速度和位置控制器，此控制器特有主动抗扰控制 (ADRC)，可实时估算并补偿系统干扰。 SpinTAC 自动补偿由以下因素导致的不良系统行为：

- 不确定性（例如，谐振模式）
- 非线性摩擦
- 负载变化
- 环境变化

SpinTAC Control 能够提供优于 PI 控制器的干扰抑制功能和轨迹跟踪性能，并且能够承受宽范围的惯性变化。 这意味着 SpinTAC 可提升精度和系统性能，并且最大限度地减少机械系统压力。

借助于单系数调整，SpinTAC 控制器使得用户能够快速地将他们的速度和位置控制从软响应测试和调整为硬响应。 这个单个增益（带宽）通常在应用的整个可变速度、位置和负载范围内起作用，从而减少了基于 PI 的多变量系统中的典型复杂度和系统调整时间。 一个单个参数控制位置和速度。 这些系统常常需要十二个或更多的已调整系数集合来处理全部可能的动态情况。

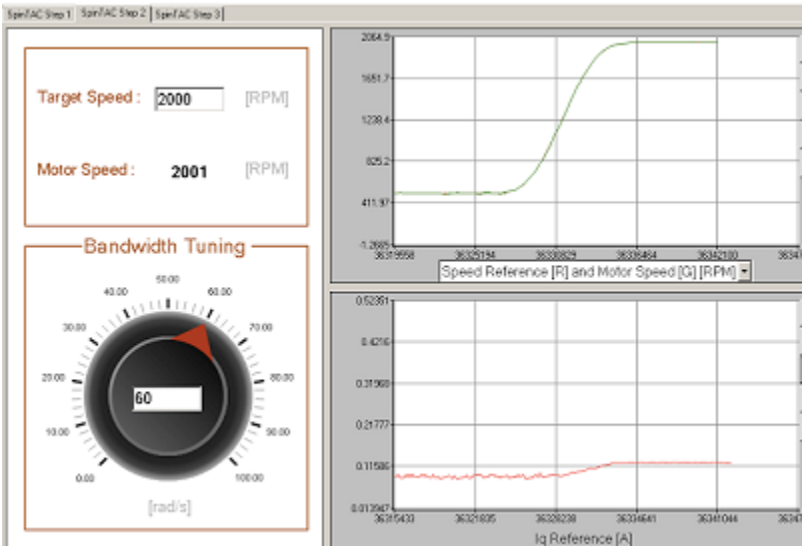


图 **1-7.** 简单调整接口

通过 InstaSPIN-MOTION（F2805xM 和 F2806xM）GUI（请参见图 1-7）连同 InstaSPIN-MOTION 快速入门指南，用户能够使用 TI 评估套件以及 TI 提供的电机，或者自己的电机快速评估 InstaSPIN-MOTION（速度控制）。 GUI 的设计可指导您快速完成 InstaSPIN-MOTION 评估过程。 您可以从 ti.com/tool/motorkitscncd69miso 免费获得 GUI。 确定 InstaSPIN-MOTION 适用于相关应用后，可立即使用基于 MotorWare 的项目连同本文档来设计项目并进行性能测试。

1.2.1.2.3 MOVE

SpinTAC Move 通过计算点 A 和点 B 之间的最快路径来提供一个速度或位置到另外一个速度或位置平稳转换的简单方法。SpinTAC 移动生成一个基于启动速度或位置、所需速度或位置，以及针对加速和急动的已配置系统限制的系统配置。 急动代表加速度变化率。 更大的加速度变化率将增加更快速率时的加速度。 步进，或两个点之间的突然运动会导致系统振荡。 步进越大，这个趋势越明显。 对于急动的控制可使速度角变得圆滑，从而减少振荡。 因此，加速度可被设定的更高一些。 控制您系统中的急动将减少您系统组件上的机械应力，并且能够获得更好的稳定性和更少的故障部件。

与预先定义的查询表相反，SpinTAC Move 在处理器上运行，从而比传统解决方案占用更少的存储器。 除了行业标准梯形曲线和 s 曲线 SpinTAC 还提供一个专有的 s 曲线，此曲线比 s 曲线更平滑，并且使得用户能够限制突然的机械运动。

Signals	Trapezoidal	s-Curve	st-Curve
Position	Smooth	Smooth	Smooth
Velocity	Continuous	Smooth	Smooth
Acceleration	Bounded	Continuous	Smooth
Jerk	Infinite	Bounded	Continuous

图 **1-8. SpinTAC Move** 中的可用曲线

图 1-8 描述了可在 SpinTAC Move 内使用的曲线。 LineStream 专有 st 曲线通过使系统配置的加速变得平滑来提供最平稳的机械运动。 对于大多数应用，st 曲线代表最佳机械运动系统配置。

1.2.1.2.4 PLAN

SpinTAC Plan 可简单设计和执行复杂的机械运动序列。 轨迹规划特性允许用户快速构建不同的运动状态（A 点至 B 点），然后通过基于状态的逻辑将它们连在一起。 SpinTAC Plan 可被用来执行一个针对几乎全部应用的机械运动序列。公式 76 显示一个洗衣机的机械运行序列，而 图 1-10 显示一个车库门的机械运行序列。 可使用 SpinTAC Plan 来轻松设计这些机械运动序列。 一旦设计完成，轨迹被直接嵌入到微控制器上的 C 语言代码中。

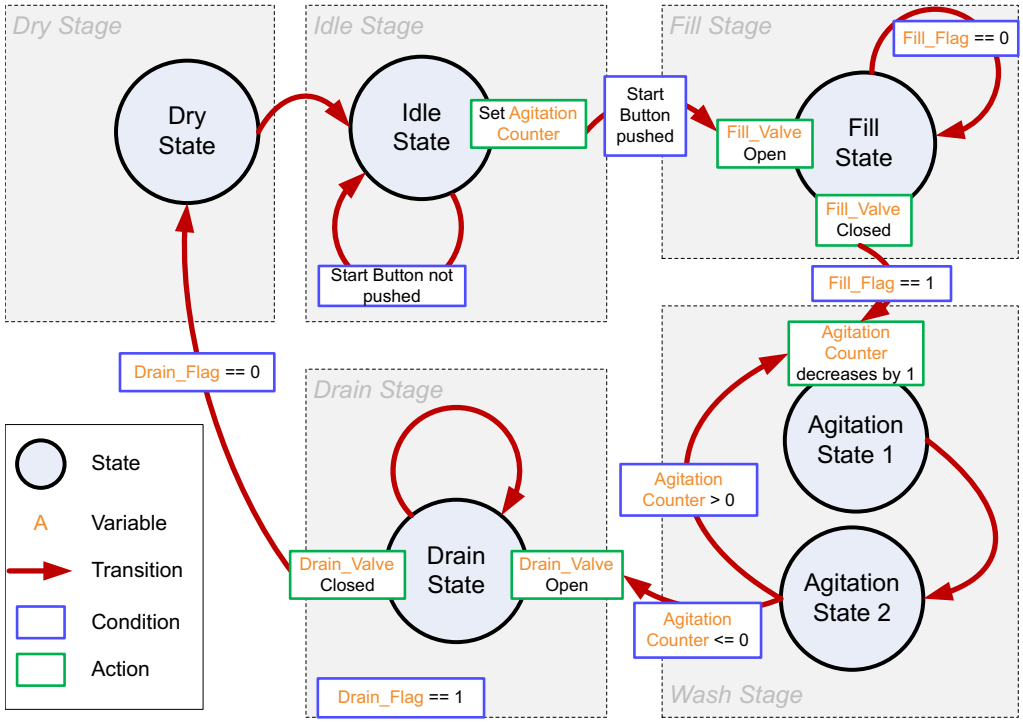


图 **1-9.** 针对一台洗衣机的状态转换图

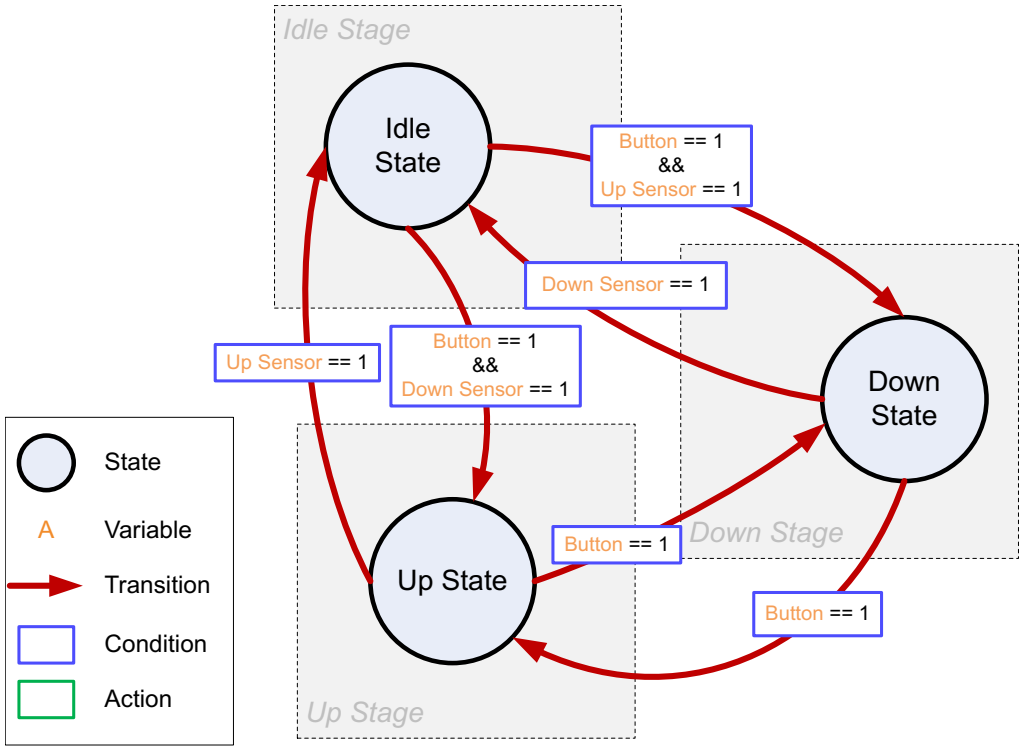


图 **1-10.** 针对一扇车库门系统的状态转换图

1.2.1.3 其它 InstaSPIN-MOTION 特性

- 自动转矩（电流）环路调整，具有用户调节选项
- 自动或手动减弱场强和增强场强
- 总线电压补偿
- 自动偏移校准可确保反馈信号采样的质量

1.2.2 InstaSPIN-MOTION 方框图

InstaSPIN-MOTION 采用模块化设计。 客户可确定他们系统中所包含的函数。 FAST 观测器驻留在 ROM 中。 SpinTAC 库的核心控制算法嵌入在 ROM 内，可通过应用程序接口 (API) 根据用户代码访问这些函数。

InstaSPIN-MOTION 支持大量的系统设计。 InstaSPIN-MOTION 针对无传感器 FOC 系统使用 FAST 软件编码器 [有关更多信息，请参见《*TMS320F2802xF InstaSPIN-FOC 技术参考手册*》（文献编号 SPRUHP4）、《*TMS320F2805xF InstaSPIN-FOC 技术参考手册*》（文献编号 SPRUHW0）和《*TMS320F2806xF InstaSPIN-FOC 技术参考手册*》（文献编号 SPRUHI9）]。 InstaSPIN-MOTION 还支持利用机械传感器（例如，编码器、解算器）的解决方案。 这些情况将在下文介绍。

注意，8.6 节、图 1-12、图 1-13 和图 1-14 中使用的变量定义如下：

- θ_{Qep}：编码器的位置角信号
- θ_{M}：SpinTAC 位置控制器中要使用的格式化锯齿位置信号
- θ_{SP}：Sawtooth 位置移动生成的锯齿位置基准信号
- ω_{lim}：速度限值（用于生成位置系统配置）
- $\dot{\omega}_{lim}$：加速度限值
- $\ddot{\omega}_{lim}$：急动限值
- ω_{Ref}：速度基准
- $\dot{\omega}_{Ref}$：加速度基准
- $\tilde{\tau}_{r}$：电机时间常量

情况 1：借助 FAST 软件编码器的 InstaSPIN-MOTION 速度控制

在这个情况下（请见 8.6 节 和 图 1-12），SpinTAC 速度控制接收来自 FAST 估算器的速度估计值，并且生成转矩基准信号。 这对于用户存储器（请参见8.6 节）中或 ROM（请参见图 1-12）中的 InstaSPIN-MOTION 同样适用。 SpinTAC 机械运动控制套件提供运动序列状态机，生成基准轨迹并且控制系统速度。

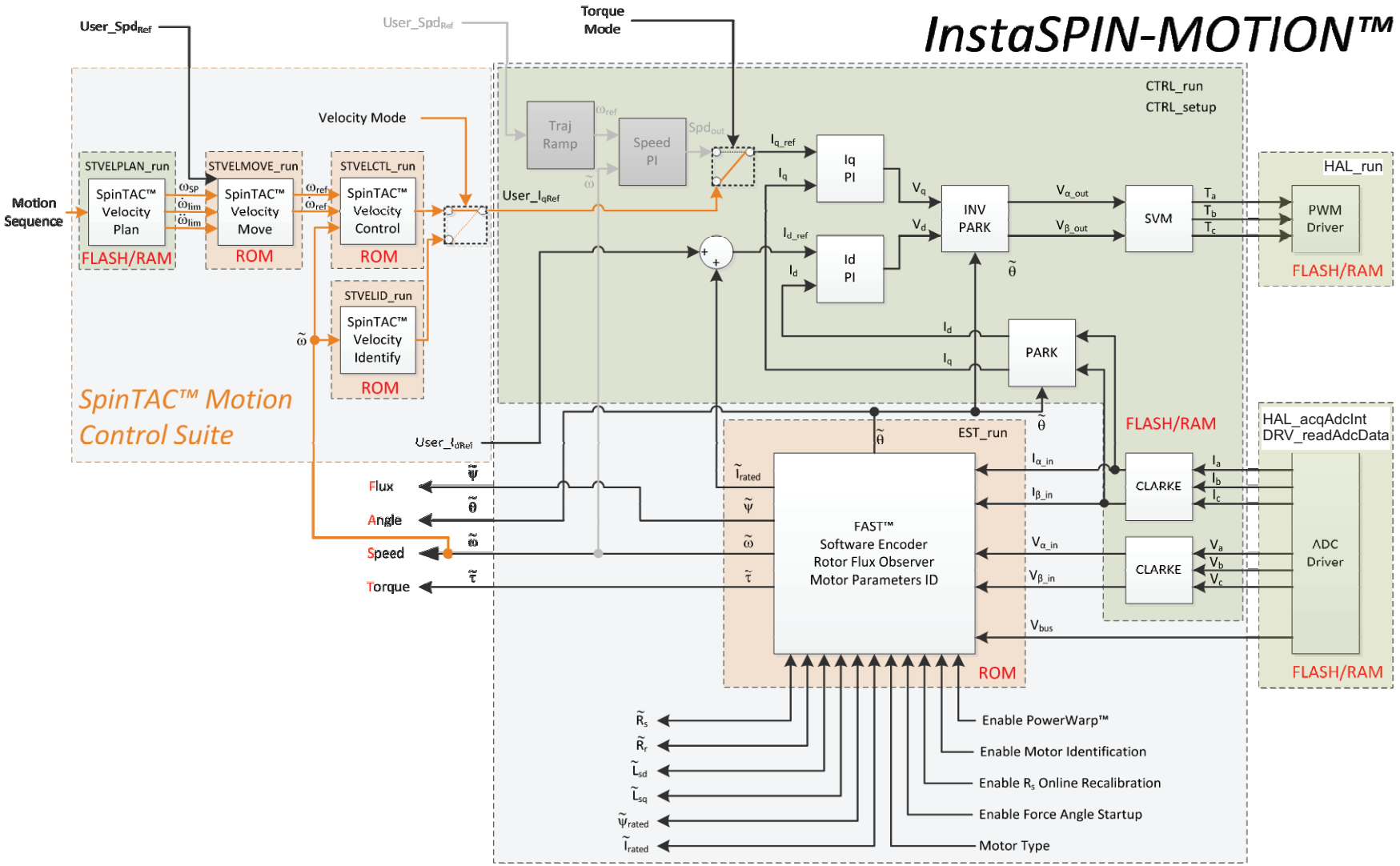


图 **1-11.** 除了 **ROM** 中的 **FAST** 和 **SpinTAC**，用户内存中的 **InstaSPIN-MOTION**

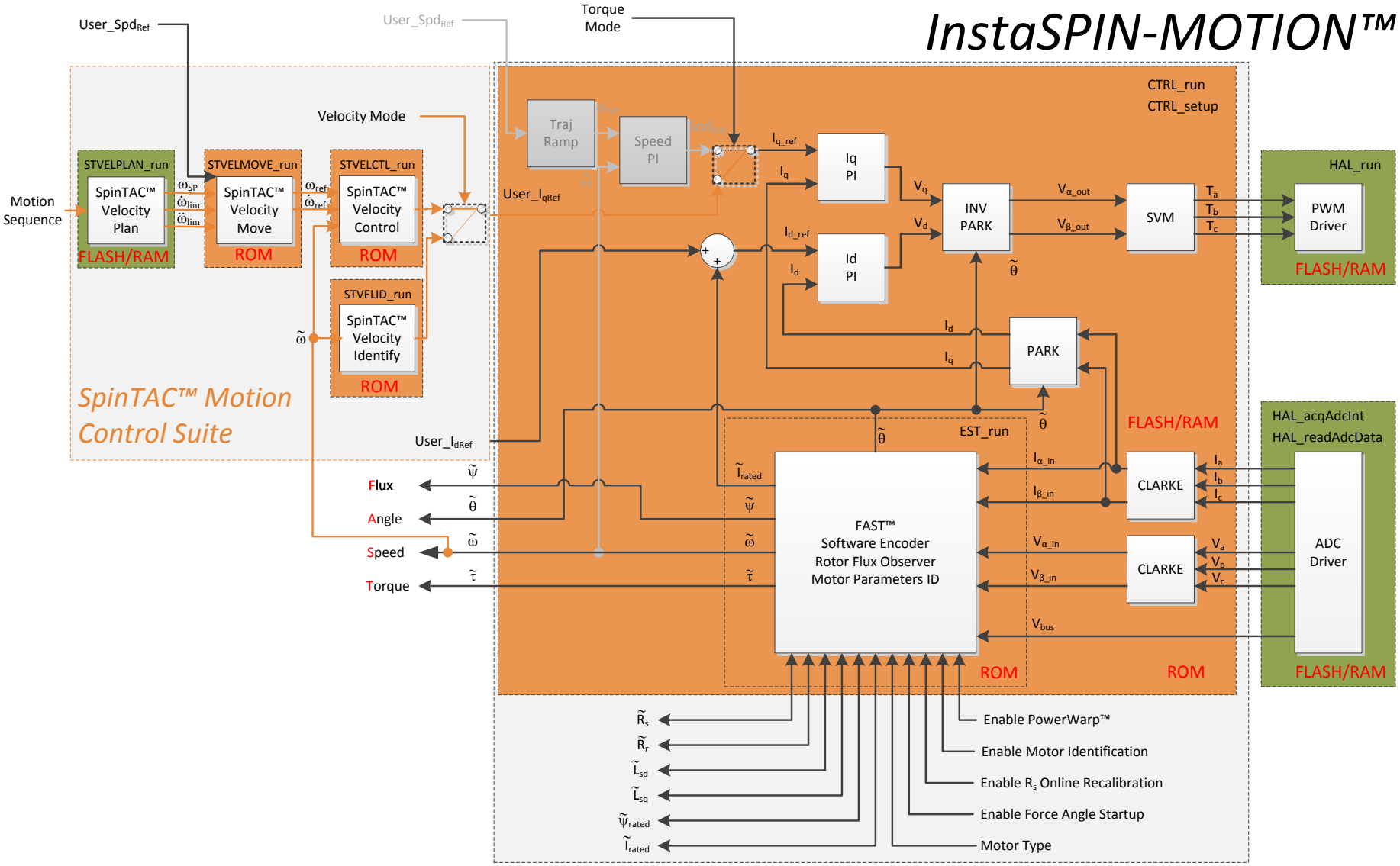


图 1-12. ROM 中的 InstaSPIN-MOTION

情况 **2**：借助机械传感器的 **InstaSPIN-MOTION** 速度控制

虽然它对于很多应用来说机具吸引力且划算的无传感器解决方案，有很多应用需要严格且高精度的机械传感器。 对于这些应用来说（请参见图 1-13），正交编码器提供位置信息，此信息随后通过 SpinTAC 位置转换器转换为速度反馈。 SpinTAC 速度控制接收速度反馈，然后通过 IqRef 生成转矩基准信号。 SpinTAC 运动控制套件提供运动序列状态机，生成基准轨迹并控制系统速度。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

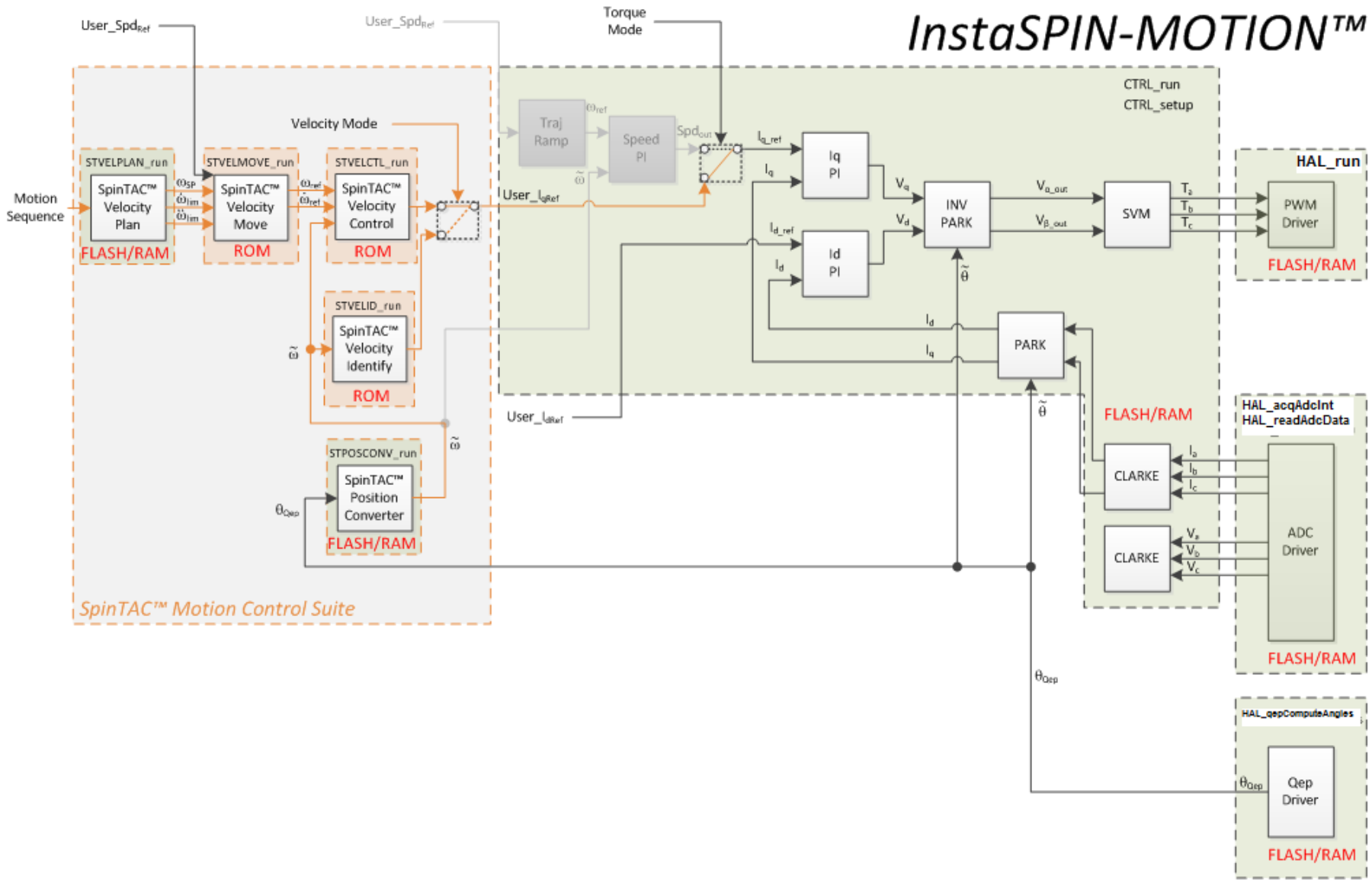


图 **1-13.** 使用机械传感器进行的 **InstaSPIN-MOTION** 速度控制

情况 3：使用机械传感器和冗余 FAST 软件传感器进行的 InstaSPIN-MOTION 位置控制

有很多需要高精度位置控制的应用。 对于这些应用，很难去平衡很多所需的调整参数。 InstaSPIN-MOTION 具有位置和速度单变量组合调节功能，可实现准确位置、速度和转矩控制。 这一特性简化了调整工作，并且使您能够专注于您的应用，而不是将精力放在电机调整方面。 位置应用需要一个机械传感器，以便准确识别零速和极低速时的电机角度。 FAST 软件编码器可在位置控制应用中实现冗余；这一特性可在机械编码器出现故障时提供安全保护（请参见图 1-14）。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

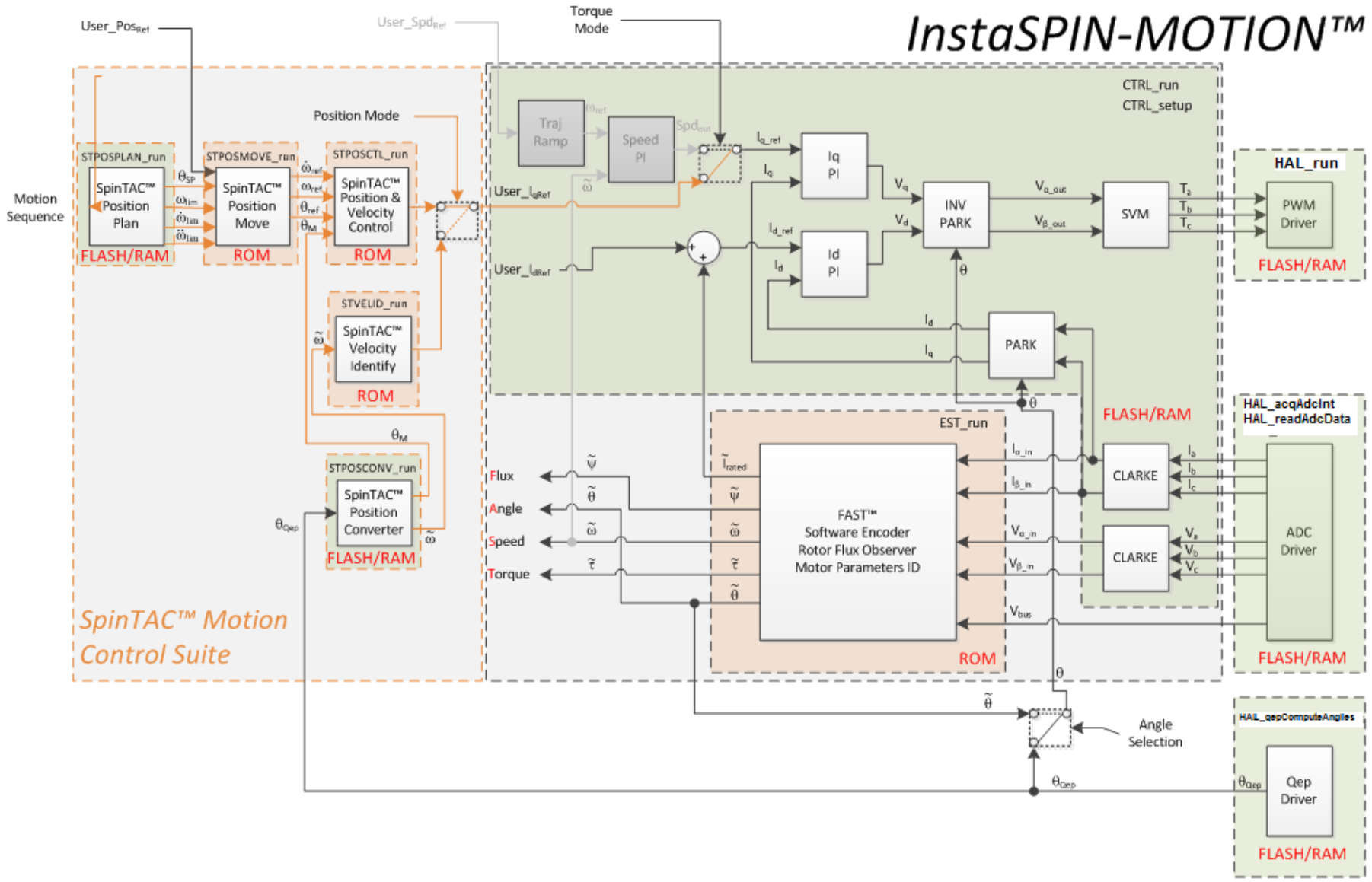


图 **1-14.** 使用机械传感器和冗余 **FAST** 软件传感器进行的 **InstaSPIN-MOTION** 位置控制

1.2.3 应用示例

InstaSPIN-MOTION 十分适合于需要精确的速度和位置控制以及最小干扰的应用，以及要进行多状态变换或经历动态变化的应用。 下文提供几个示例。

1.2.3.1 跑步机传送带：不同速度和负载下的平滑运动。

持续速度控制对于跑步机传送带十分关键。 当有人在跑步机上跑动时，他们的跨步动作会干扰传送带的运动。 如果驱动传送带的电机不能快速提供足够的转矩来克服干扰，那么跑步者的跨步动作将被打断。 这个问题在用户将变速运动作为他们锻炼方式的一部分时加剧。 如果传送带无法平滑加速或减速，那么看起来就好像跑步机无法正常工作。 此外，低速时，当用户在传送带上行走时，他们的体重会使得传送带停止。

InstaSPIN-MOTION 被应用于使用 4HP，220V 交流感应电机的商用跑步机，以驱动传送带。 此跑步机的调速范围已经过测试：最低速 42rpm，最高速 3300rpm。

客户会发现，InstaSPIN-MOTION 的高级控制器自动补偿干扰，从而在跑动和速度变化的情况下保持运行速度不变。 低速时，此控制器防止传送带在负载施加时停止。 此外，一个单个增益被用来控制整个工作范围。

1.2.3.2 视频摄像机：低速时平滑运动和位置准确度

高端安保和会议室摄像机以极低速运转（例如，0.1rpm），并且需要准确且平滑的位置控制来转动，倾斜和缩放。 很难在低速时对驱动这些摄像机的电机进行调节，并且他们通常需要最少四个调整集。 此外，启动时会有跳动，这会导致抖动或未聚焦画面。

InstaSPIN-MOTION 被应用于由具有磁编码器的 2 极 BLDC 电机驱动的高精度安保摄像机。 InstaSPIN-MOTION 能够使用一个在整个工作范围内都有效的单一调整参数来控制速度和位置。 SpinTAC Move 被用来控制电机急动，从而获得平滑启动。

1.2.3.3 洗衣机：低速时的平滑运动和位置准确度

周期变换、负载变化和环境干扰会导致严重的电机磨损。 自动、实时的干扰减少能够延长电机的使用寿命和性能。

例如，让我们看一看洗衣机。图 6-59 显示针对一个标准洗衣机三个阶段的机械运动系统配置。 第一阶段代表搅动周期，在 250rpm 到 -250rpm 之间重复旋转。 第二和第三阶段代表两个不同的旋转周期。 第二阶段的旋转速度为 500rpm，而第三阶段旋转速度为 2000rpm。 此系统配置可通过 SpinTAC Plan 轻松创建。

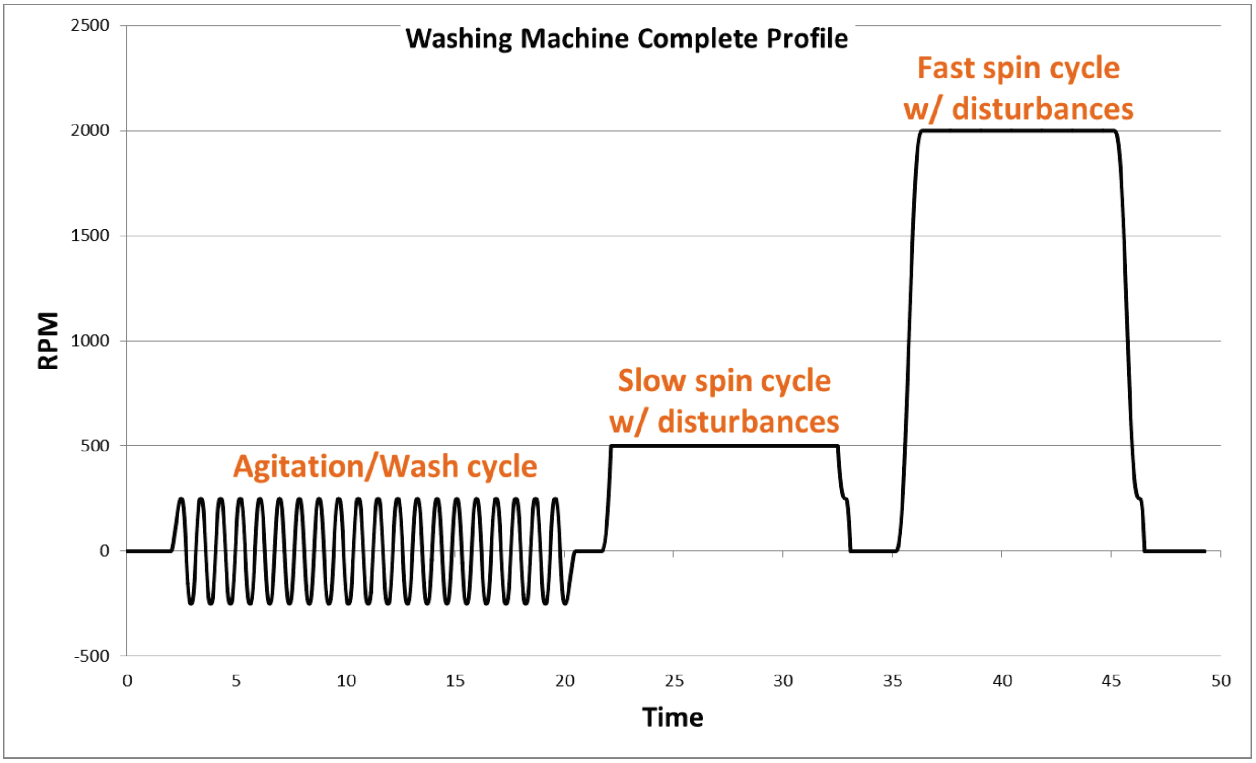


图 **1-15.** 洗衣机系统配置

InstaSPIN-MOTION 被施加到一个洗衣机应用中。 SpinTAC Plan 轨迹规划特性用于快速构建不同的运动状态（速度 A 至速度 B），然后通过基于状态的逻辑将它们连在一起。

洗衣机应用运行了两次，一次使用标准 PI 控制器，一次使用 LineStream 公司的 SpinTAC 控制器。 然后，在参考曲线上标出数据进行比较。

1.2.3.3.1 搅动周期

搅动期间，电机在 250rpm 和 -250rpm 设定点之间切换 20 次。 图 1-16 中的结果表示 InstaSPIN-MOTION 更加贴近参考曲线。 此外，PI 的最大误差为 91rpm (341 - 250 = 91rpm)，而 InstaSPIN-MOTION 的最大误差为 30rpm (280 - 250 = 30rpm)。

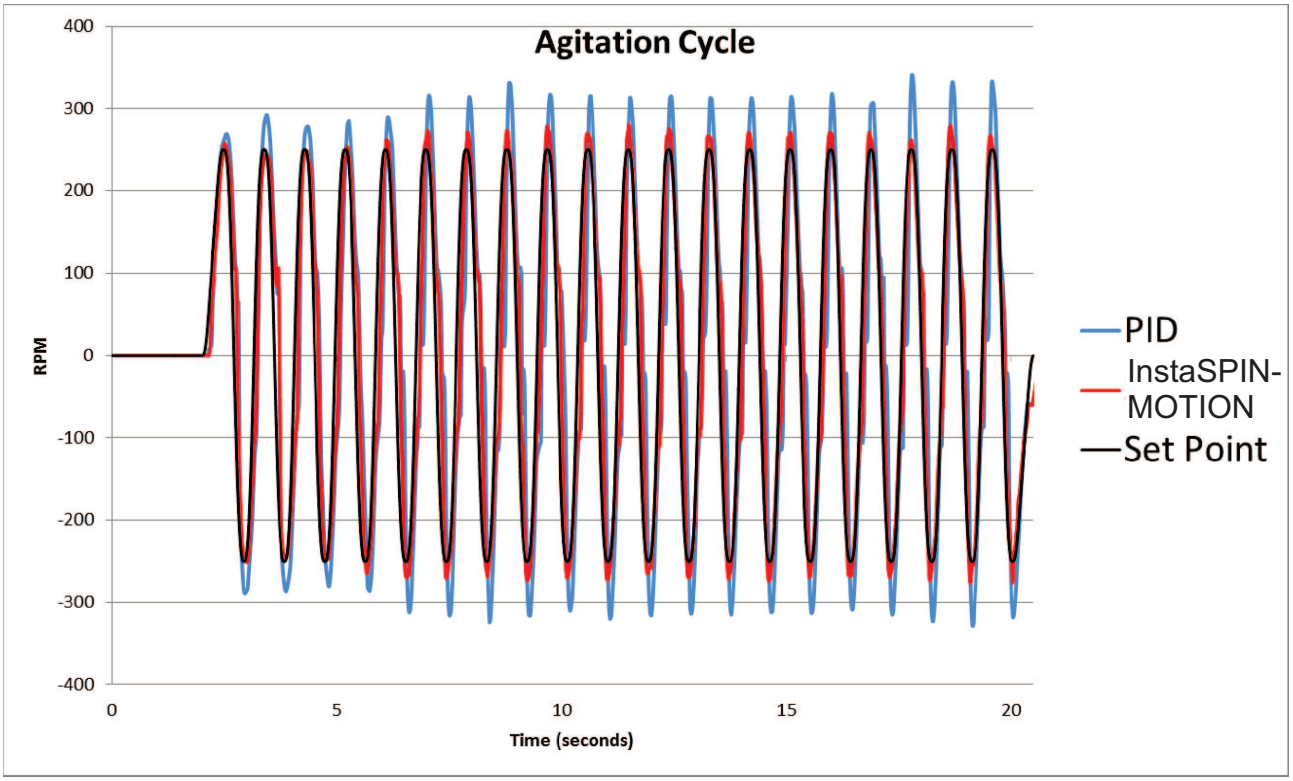


图 **1-16. InstaSPIN-MOTION** 最大限度减少误差

1.2.3.3.2 旋转周期

在第一个旋转周期内，运行目标是保持 500rpm 转速，即使在引入干扰时也是如此。图 7-7 表明，相对于 PI 控制器，InstaSPIN-MOTION 从干扰中恢复的速度更快，振荡更少。 此外，当尝试达到初始 500rpm 设定值时，InstaSPIN-MOTION 不会受到 PI 控制器所表现出的过冲和下冲的影响。

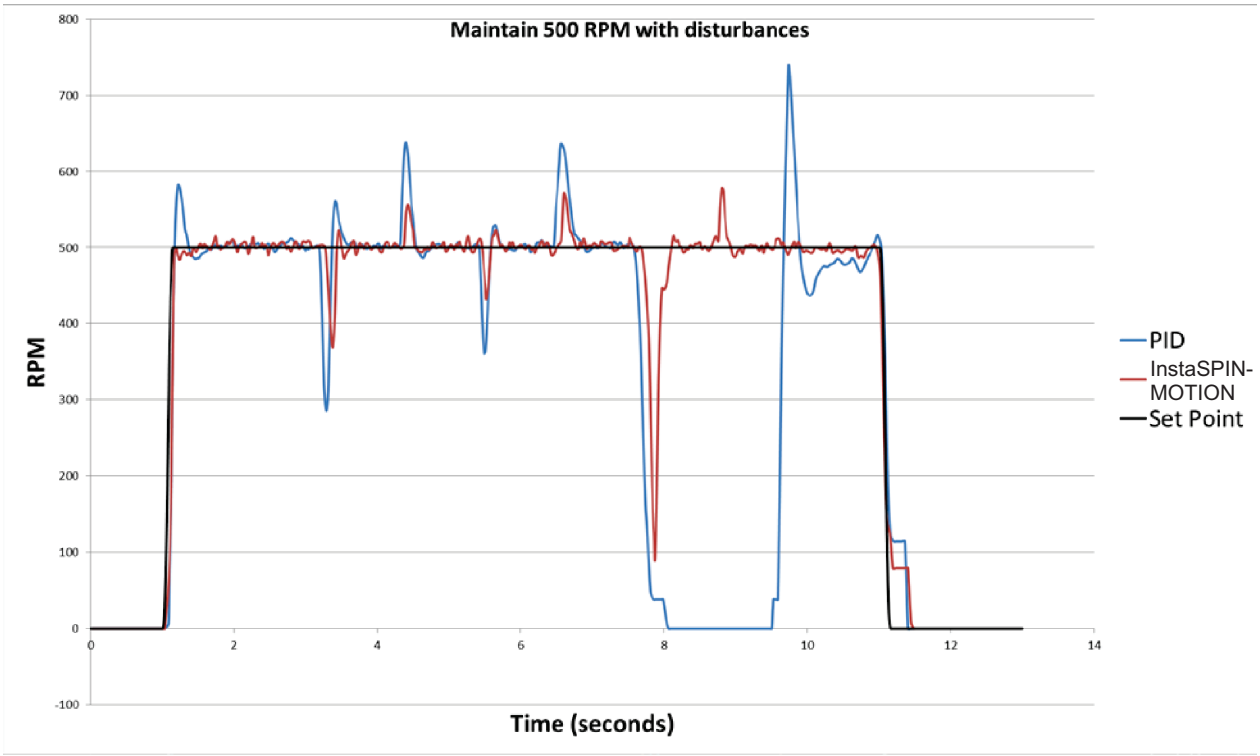


图 **1-17.** 第一个旋转周期 **- 500rpm**

在第二个旋转周期内，如图 1-18 中所示，相对于 PI 控制器，InstaSPIN-MOTION 在 2000rpm 时从干扰中恢复的速度更快，振荡更少。 请注意，当尝试达到初始 2000rpm 设定值时，InstaSPIN-MOTION 不会受到 PI 控制器所表现出的过冲和下冲的影响。

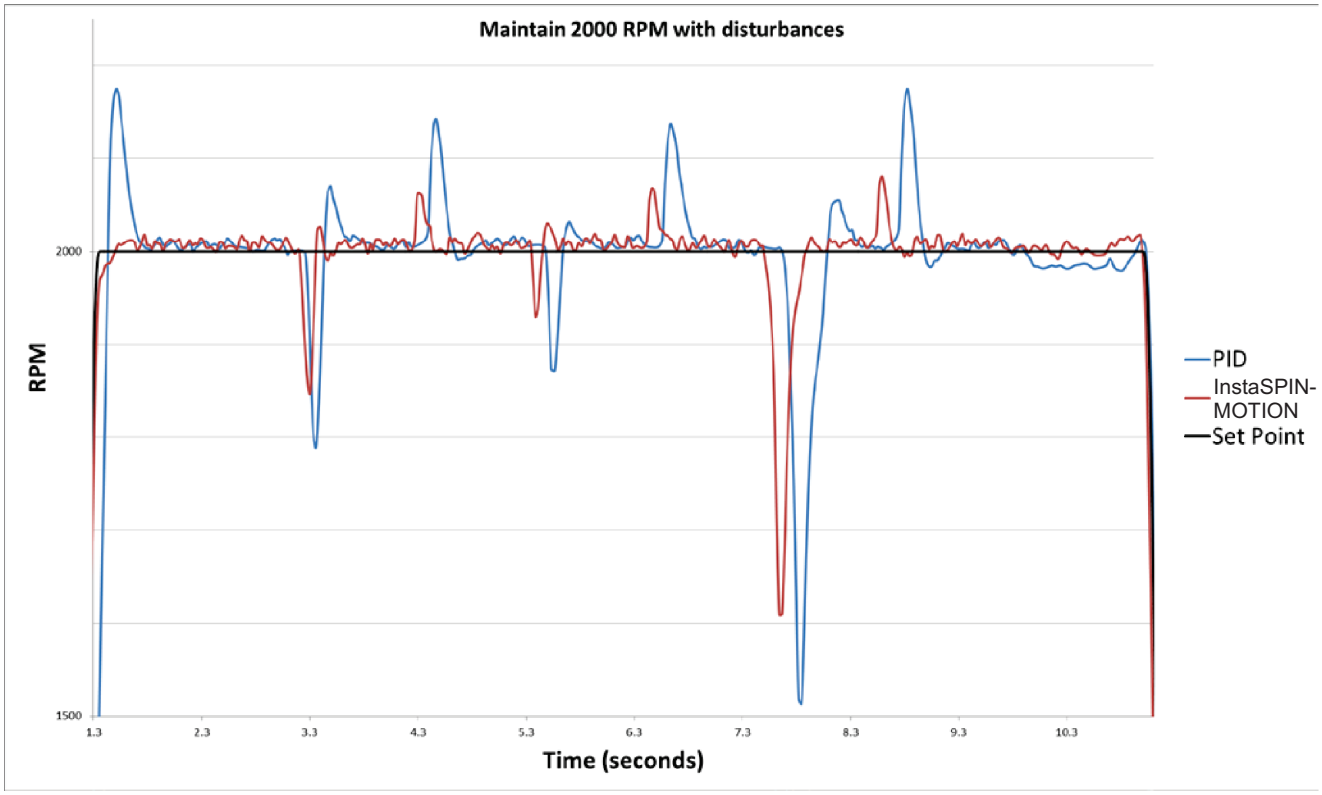


图 **1-18.** 第二旋转周期 **- 2000rpm**

此外，PI 控制器不能从 9.75 秒标记的斜坡干扰中恢复。 相反地，它表现出一个大约为 20rpm 的稳定状态误差。

1.2.3.3.3 InstaSPIN-MOTION 在整个工作范围内的运行

在执行洗衣机应用前，InstaSPIN-MOTION 控制器和 PI 控制器被调整一次。 从示例中可得知，InstaSPIN-MOTION 的调整功能在整个工作范围内均有效。 不论电机是在 250rpm 到 -250rpm 间切换，或者是保持在 500rpm 或 2000rpm 旋转周期，都不需要新的调整集。

快速入门套件 - TI 提供的软件和硬件

TI 提供几个基于 GUI 的应用，方便用户轻松便捷地评估 InstaSPIN-FOC 和 InstaSPIN–MOTION 软件。 使用预配置的图形用户界面 (GUI) 是最快的用户入门方式，其中为 F2806xM 和 F2806xF 器件提供了演示平台。 通用 GUI 所提供的 GUI 选项适用于任何可定制的 MotorWare 项目。 可以在 Code Composer Studio 外或集成开发环境 (IDE) 内使用此 GUI。 *GUI 快速入门指南*将为您介绍评估过程的详细信息。 以下网址提供了预配置 GUI 的简单概述：http://www.ti.com/tool/instaspinfocmotorwaregui

目前提供的套件均使用同一款处理器 TMDSCNCD28069MISO Piccolo F28069M (ROM) controlCARD，与以下任意一种 3 相逆变器配套：

- 低压/低电流：DRV8312
  - PN：DRV8312-69M-KIT http://www.ti.com/tool/DRV8312-69M-KIT
  - DRV8312 三相逆变器集成了电源模块基板，通过 controlCARD 接口支持高达 50V 电压和 3.5A 持续电流
  - 1 个 NEMA17 BLDC/PMSM 55W 电机
  - 若要进行位置控制，需要购买 Anaheim Automation 电机（含编码器）：BLY172D-24V-4000-2000SI
- 低压/中等电流：适用于 LaunchPad 的 BoosterPack
  - 6-24V、14A 持续电流
  - PN：BOOSTXL-DRV8301 BoosterPack http://www.ti.com/tool/BOOSTXL-DRV8301
  - PN：支持 InstaSPIN 的 LaunchPad：http://www.ti.com/launchpad
  - DRV8301 2.3A 灌电流/1.7A 拉电流三相逆变器，具有用于 1.5A 外部负载的集成降压转换器
  - NexFET™ 功率 MOSFET
  - 不包含电机或电源
- 低压/高电流：DRV8301
  - PN：DRV8301-69M-KIT http://www.ti.com/tool/DRV8301-69M-KIT
  - DRV8301 2.3A 灌电流/1.7A 拉电流三相逆变器，具有用于 1.5A 外部负载的集成降压转换器
  - 不包含电机
  - 若要进行位置控制，需要购买 http://www.ti.com/tool/lvservomtr
- 高压：hvmtrkit
  - PN：TMDSHVMTRINSPIN http://www.ti.com/tool/TMDSHVMTRINSPIN
  - 支持交流感应电机、永久磁性同步电机、无刷直流电机
  - 使用 350V 直流总线时电机驱动器级最高为持续 10A
  - 可订购以下高压电机：
    - ACIM http://www.ti.com/tool/hvacimtr
    - BLDC http://www.ti.com/tool/hvbldcmtr

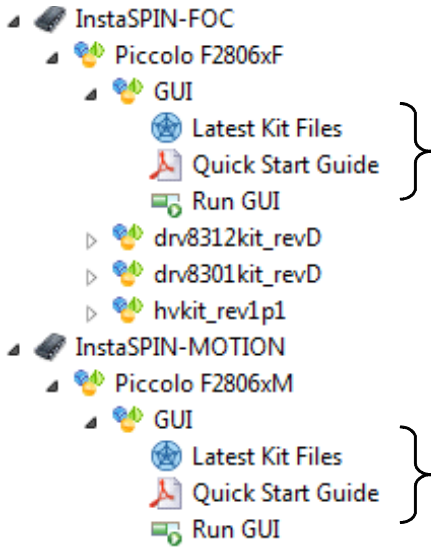
SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

- PMSM http://www.ti.com/tool/hvpmsmmtr

– 若要进行位置控制，需要购买 http://www.ti.com/tool/hvpmsmmtr

MotorWare 软件下载中提供了所有软件和文档，网址为：
http://www.ti.com/tool/motorware。

套件信息也可以通过资源浏览器访问，资源浏览器是适用于 TI MCU 的 Code Composer Studio™ (CCStudio) IDE 中的一个应用。 资源浏览器还将显示首次在新工作区使用的 CCStudio，如果稍后要将其打开，可选择：*帮助 (Help) -> 欢迎使用 CCS (Welcome to CCS)*。 以下是所提供的信息示例。



本文档以及使用 InstaSPIN 开发套件的所有相关文档均可通过资源浏览器访问：

附加硬件信息通过 ControlSUITE™ 库（针对 C2000™ 微处理器）提供，该库包含全套软件基础设施和软件工具，旨在最大限度缩短软件开发时间。 controlSUITE 库可从以下位置下载：ti.com/tool/controlsuite。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

2.1 评估 InstaSPIN-FOC 和 InstaSPIN-MOTION

GUI 提供了一种可以轻松便捷地评估 InstaSPIN-FOC 和 InstaSPIN–MOTION 的方法。 InstaSPIN-FOC 和 InstaSPIN-MOTION 快速入门指南将为您介绍评估过程的详细信息。 此处提供简单的概述：

Step 1. 识别电机参数 (图 2-1)。

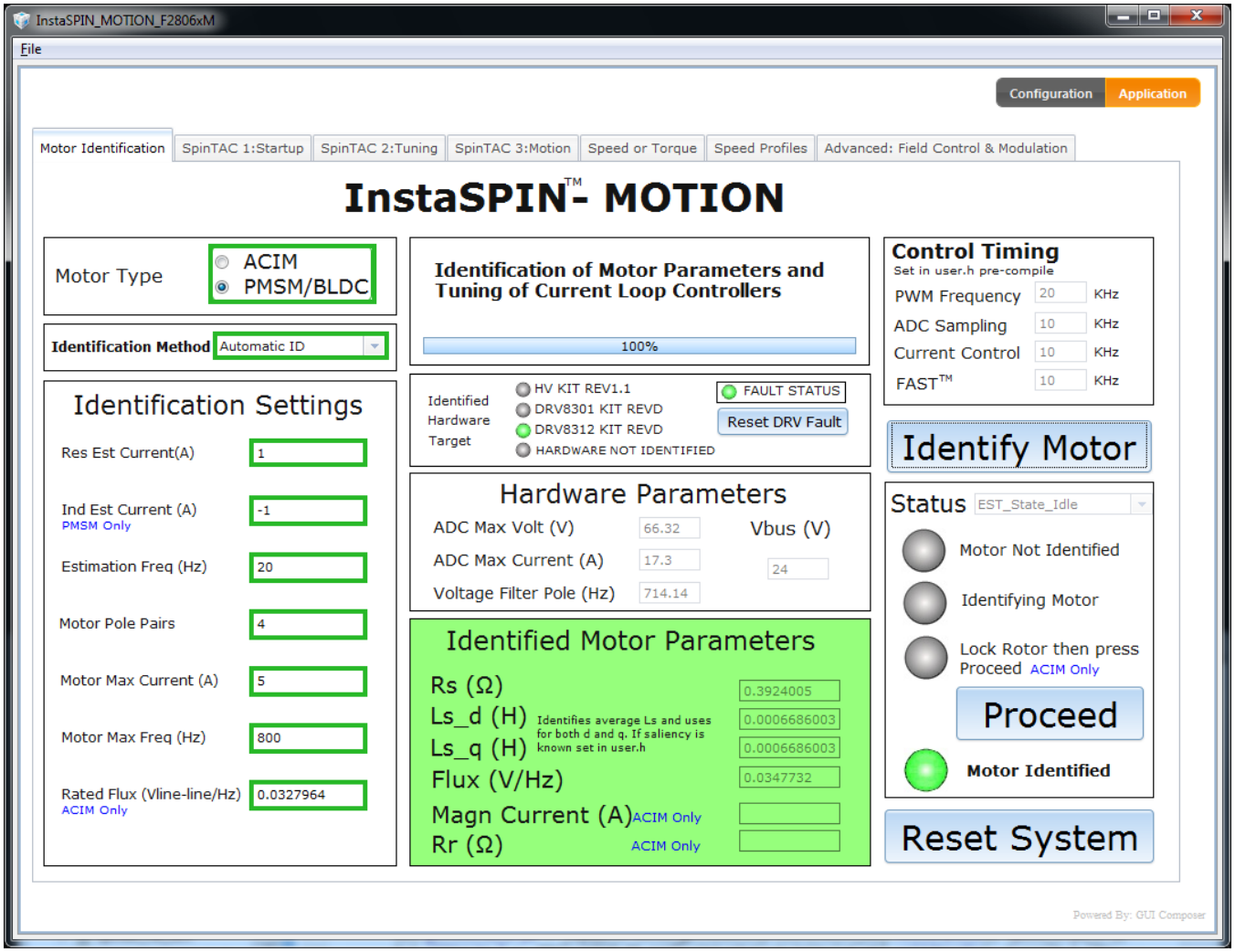


图 2-1. InstaSPIN-MOTION GUI 使用*电机识别 (Motor Identification)* 标签

Step 2. 调整转矩和速度控制 (图 10-5)。

图 **2-2. InstaSPIN-MOTION GUI** 使用*速度或转矩 (Speed or Torque)* 标签

Step 3. 识别系统惯性 (图 2-3)。

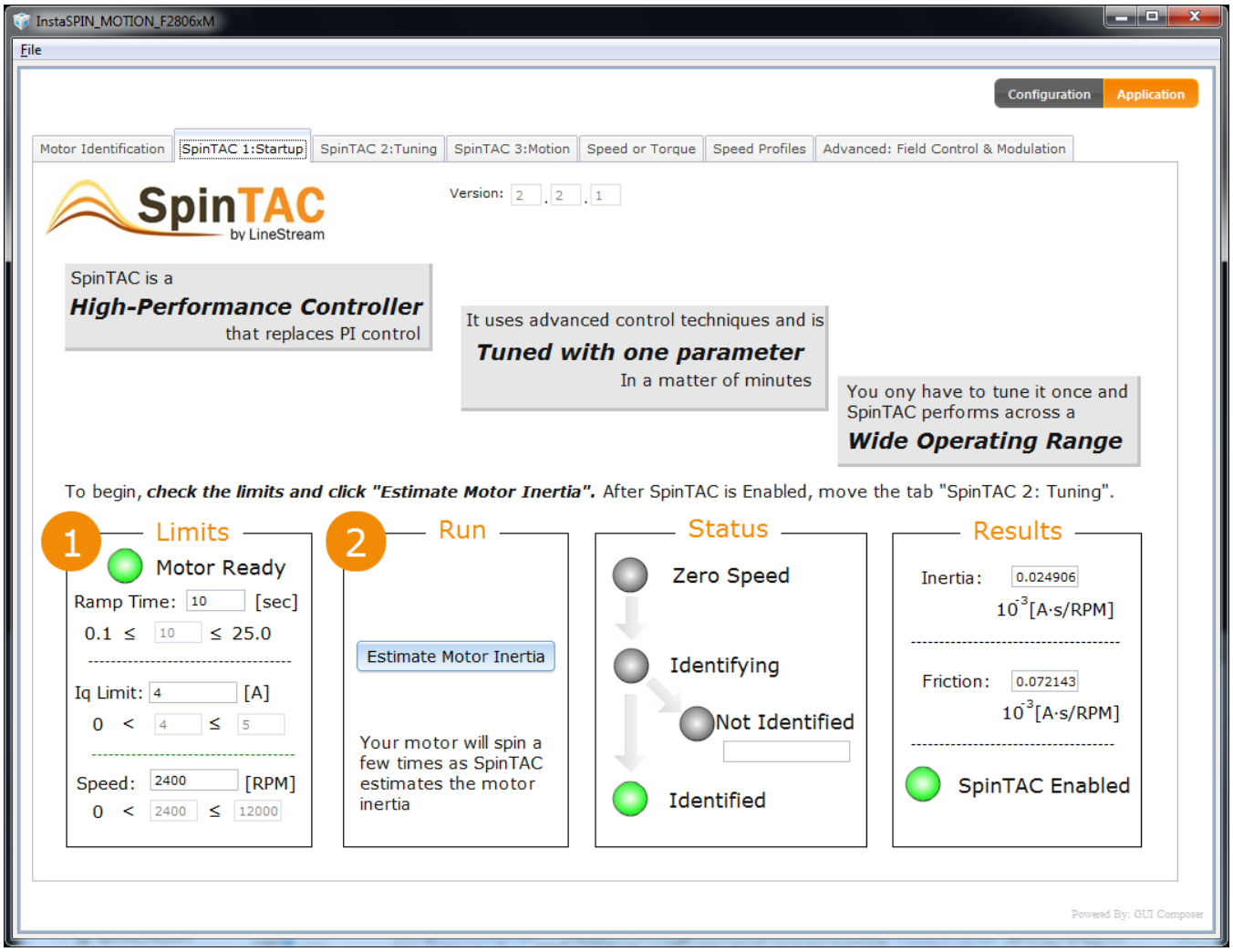


图 **2-3. InstaSPIN-MOTION GUI** 使用 *SpinTAC 1：启动 (pinTAC 1:Startup)* 标签

Step 4. 调整抗扰速度控制器 (图 2-4)。此操作将替代第 2 步中的速度控制器。

图 **2-4. InstaSPIN-MOTION GUI** *SpinTAC 2：调整 (SpinTAC 2:Tuning)* 标签

Step 5. 设置目标速度并选择曲线类型 (图 2-5)。

图 **2-5. InstaSPIN-MOTION GUI** 使用 *SpinTAC 3：运动 (SpinTAC 3:Motion)* 标签

通过浏览 InstaSPIN 快速入门套件和快速入门指南，您可以熟悉软件并能够参考硬件原理图设计自己的电路板。 已配置的代码示例（实验）可在各个套件上运行。 无论是否对 InstaSPIN-FOC 或 InstaSPIN-MOTION 的代码感兴趣，您都可以找到快速开始项目所需的各个示例。 有关示例软件和文档，请参见 MotorWare 软件下载中的 MotorWare *InstaSPIN 项目和实验用户指南*（网址：http://www.ti.com/tool/motorware）。

InstaSPIN 和 MotorWare

MotorWare™ 库是一套全面的软件和技术资源，旨在最大限度地缩短电机控制系统开发时间。

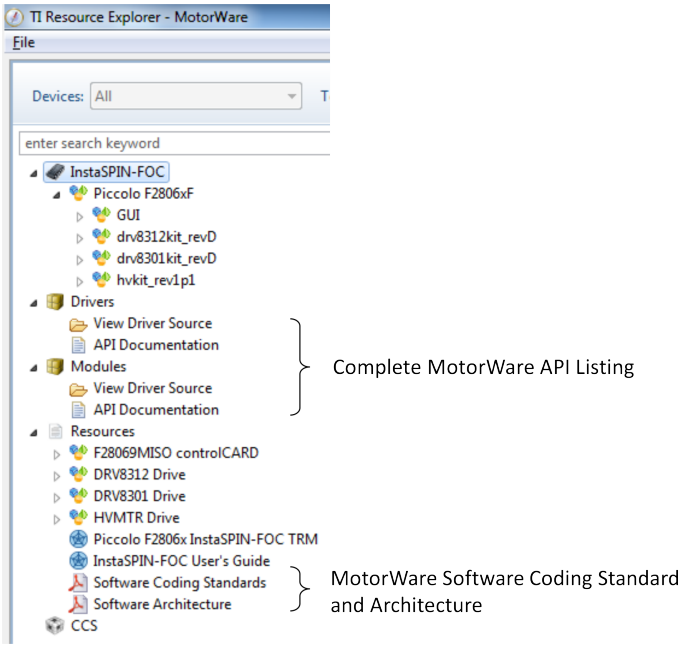
从器件专用驱动程序及支持软件，到完整的系统示例和技术培训，MotorWare 软件可为开发及评估的各个阶段提供支持。

MotorWare 软件设计成可轻松集成一流的电机控制技术。

该软件旨在实现：

- 跨 TI MCU 支持
- 跨 MCU、电力电子设备和控制技术的模块化和便携性
- 面向对象的软件设计
- 基于 API

MotorWare 内提供 InstaSPIN-FOC 和 InstaSPIN-MOTION 电机控制解决方案。 有关最新的完整 API 函数列表、MotorWare 的软件编码标准和架构，请参见 CCStudio 内的 Resource Explorer。



Topic Page

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

3.1 MotorWare 目录结构

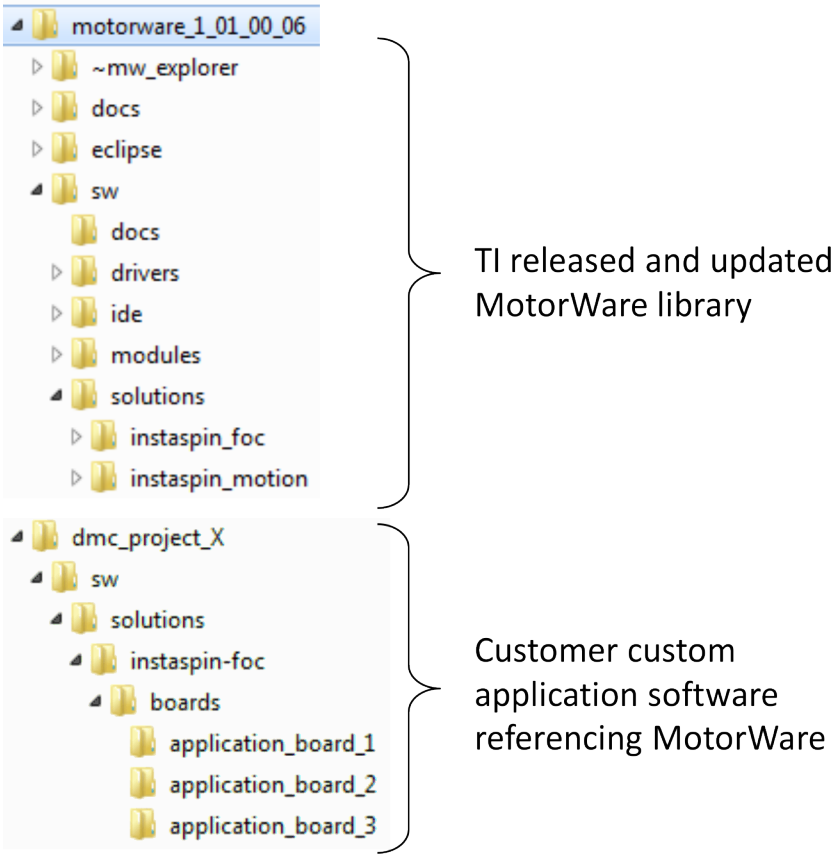
MotorWare 的目录结构包含了运行每个电机控制项目所需的所有代码。

Code Composer Studio (CCStudio) 项目中的文件引用是相对的。 通过相对目录链接能够打开项目并进行首次编译。 MotorWare 目录结构旨在提供查找头文件、库和源代码的简单方式。

MotorWare 目录结构的核心由四个文件夹组成：

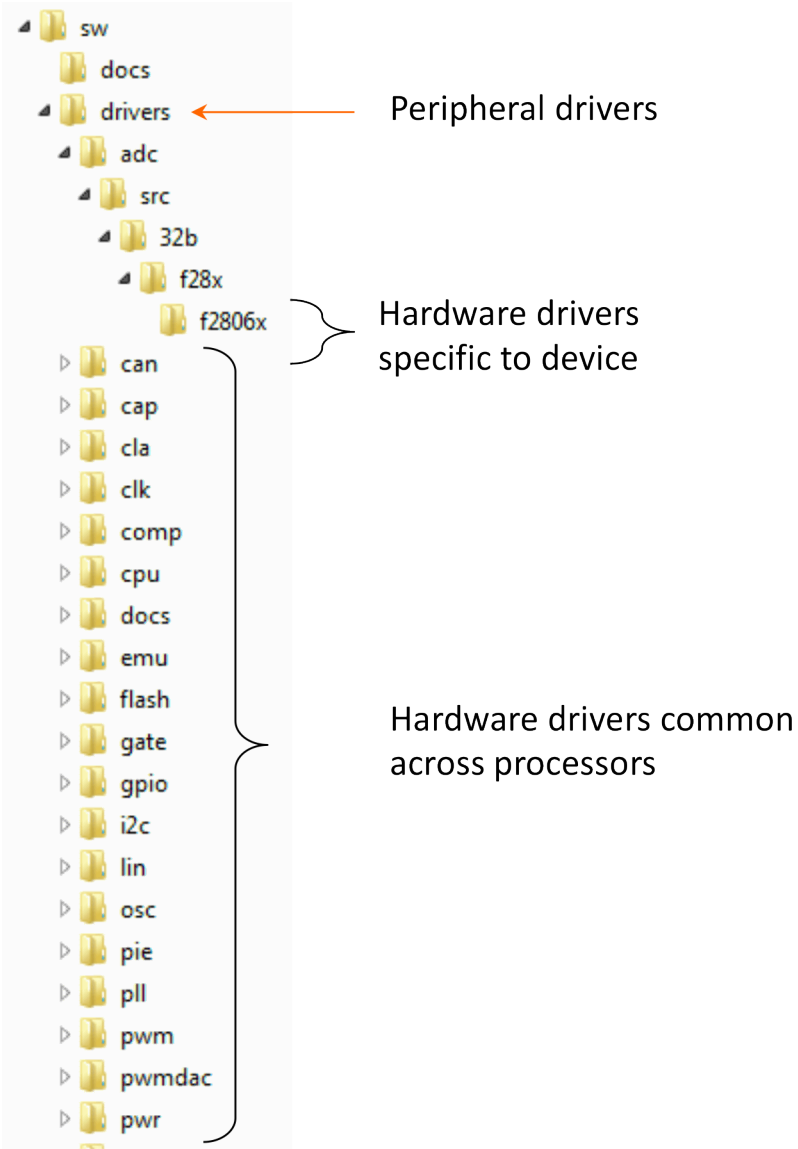
- drivers - 外设驱动程序 API 代码。
- ide - CCStudio 使用的通用连接器文件
- modules - 电机控制使用的函数
- solutions - 包含 CCStudio 项目，这些项目用于运行基于电机示例套件的软件解决方案。

将 TI 的 MotorWare 软件与您的应用程序集成时，建议创建一个单独的 MotorWare 目录结构，在其中包含多个项目的板特定文件。 之后您的软件便可引用 TI 的 MotorWare 目录内的文件。 这是可选的，但建议这样做，以简化 TI 计划的未来更新的使用，请参见下面的屏幕截图。 有关目录结构的实际内容，请参见 MotorWare 软件的最新版本。



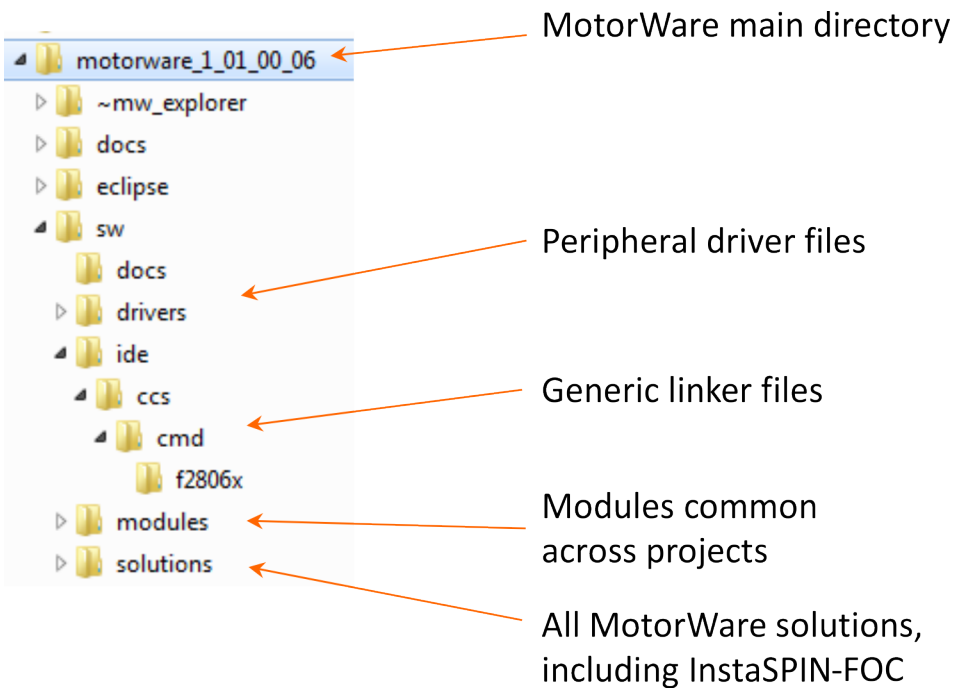
3.1.1 MotorWare – drivers

drivers 目录包含用于配置特定处理器的外设 API。



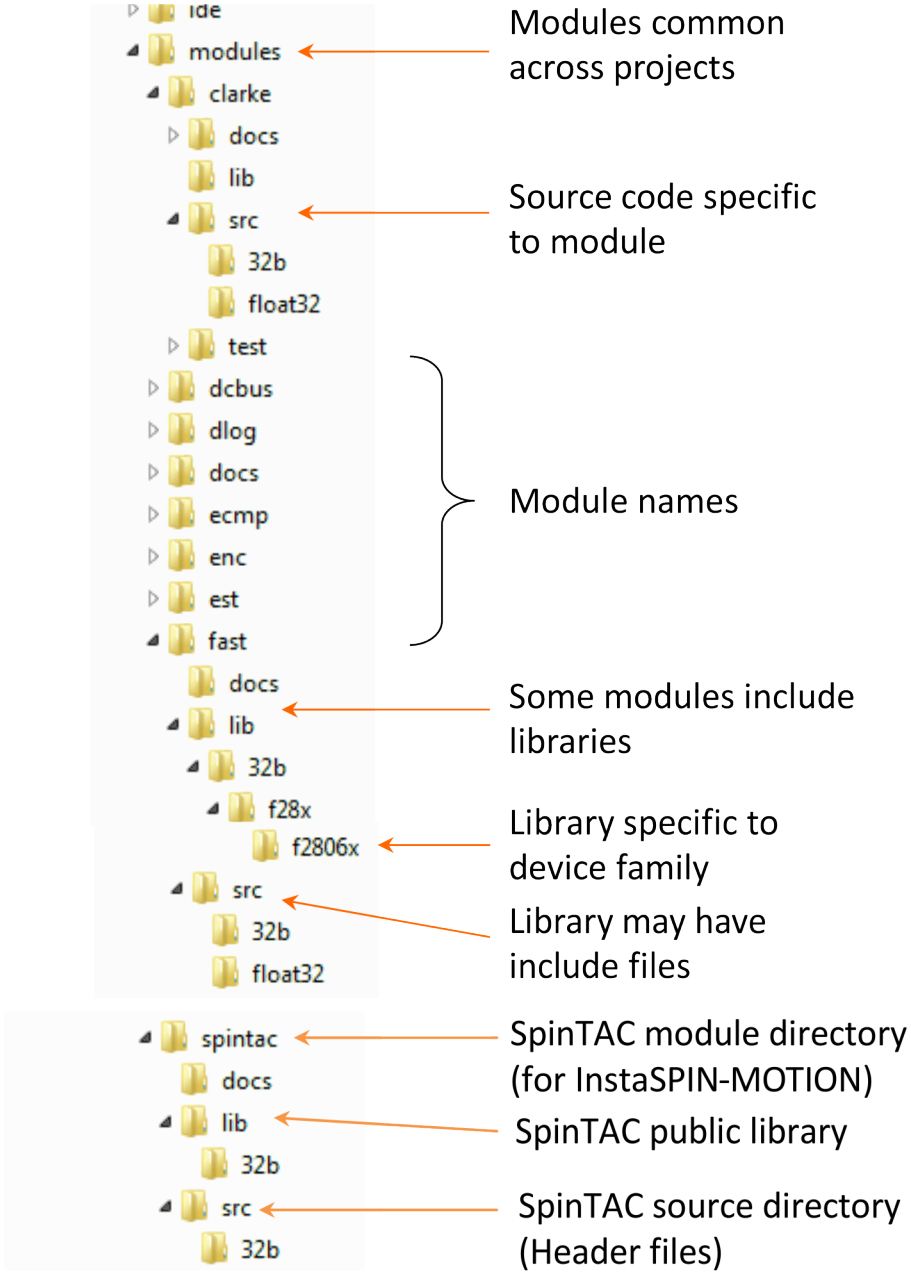
3.1.2 MotorWare – ide

IDE 目录包含编译器工具所需的通用连接器文件。



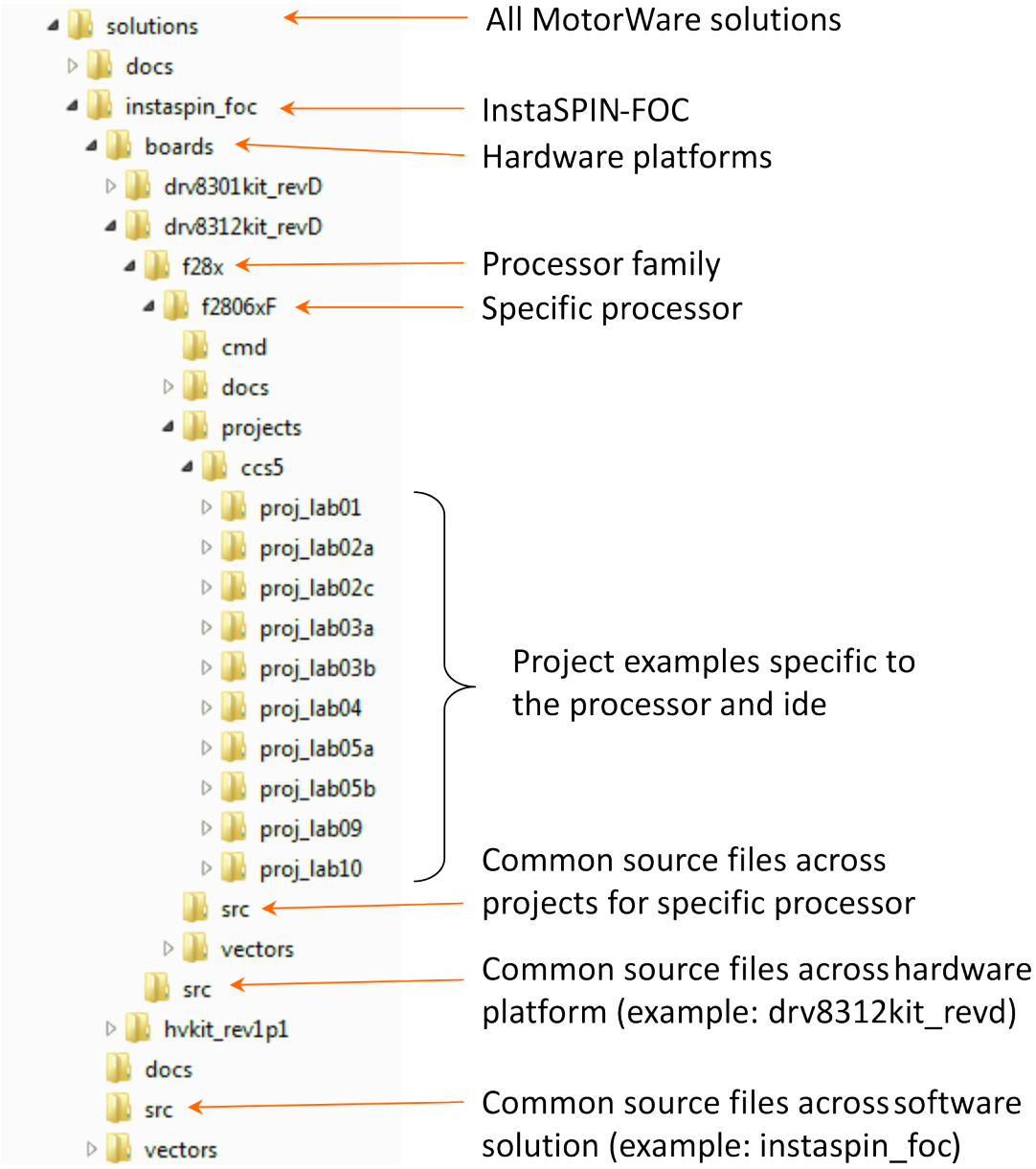
3.1.3 MotorWare – modules

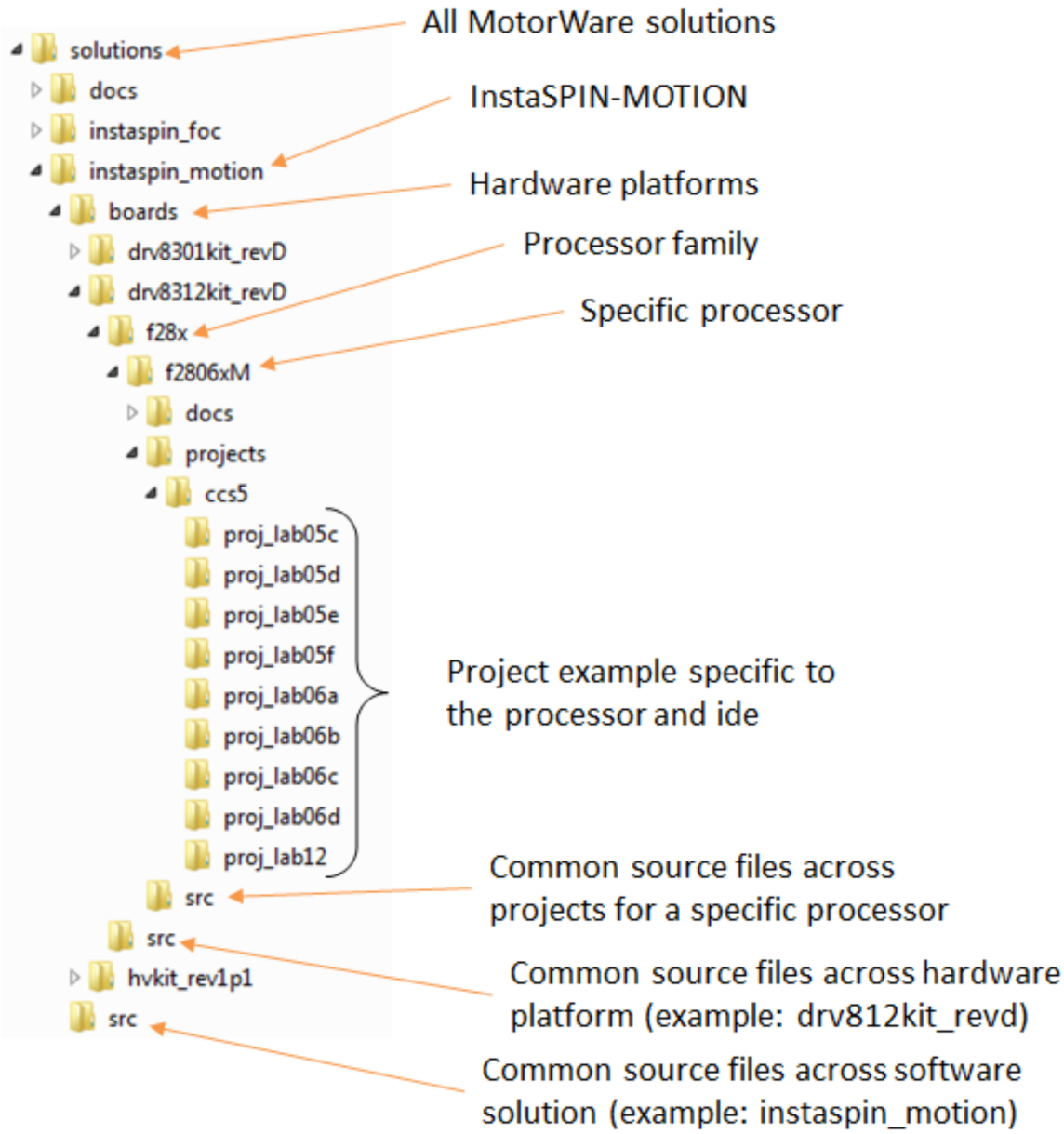
modules 目录包含处理器和项目间通用的算法。 如果某个模块与处理器有特定的依赖关系，则有子目录对应此依赖关系。 模块将具有源代码和/或库函数。



3.1.4 MotorWare – solutions

solutions 目录包含完整的解决方案，其中含有针对特定目标的完整 CCStudio 项目。 InstaSPIN-FOC 和 InstaSPIN-MOTION 示例代码在多个目标器件间通用。 通过分层的目录结构可将与目标无关的通用源代码与不同的处理器和目标板一起使用。 如果存在与目标相关或与处理器相关的源代码，则有特定的源代码目录用于这些用途。 下图说明了此结构。





3.2 MotorWare 面向对象的设计

MotorWare 软件已使用面向对象的方法。 通过使用对象，软件实现了自我文档化，并且减少了在 main.c 文件中使用的空间。 对象实际是一种结构，其中包含对象用来执行其函数的变量。 与对象关联的是方法，它们是用来设置和运行对象计算的函数调用。 当我们讨论面向对象的软件技术的定义时，使用帕克变换对象来演示对象编写方式的示例。

3.2.1 对象

对象是一种结构。 帕克变换结构示例如下所示。 Park.h 是包含结构声明的文件。

```
typedef struct _PARK_Obj_{
 _iq sinTh; //!< the sine of the angle between the d,q and the alpha,beta
            coordinate systems
 _iq cosTh; //!< the cosine of the angle between the d,q and the alpha,beta
            coordinate systems
 } PARK_Obj;
```

每个对象都有一个句柄。 句柄是对象的指针。 在函数间传递对象时，句柄非常有用。 对象的句柄还允许函数仅对该对象起作用，或者称为可重入代码。 下面列出了帕克变换对象的句柄声明。

```
typedef struct PARK_Obj *PARK_Handle;
```

3.2.2 方法

每个对象都必须做一些事情。 目前，对象只是变量的容器。 为了让对象执行计算，甚至在其变量间发送和接收数据，必须使用方法。 方法是特定于对象的函数，作用于对象包含的变量。 MotorWare 中的每个对象有四个主要方法，它们的命名如下。

- Init 方法 - 仅用于创建对象的句柄
- Set 方法 - 设置对象的内部变量
- Get 方法 - 返回对象的内部变量值
- Run 方法 - 执行对象的计算函数

3.2.2.1 Init 方法

init 方法仅用于将句柄指向对象。 帕克变换 **init** 方法的代码如下所示。

```
PARK_Handle PARK_init(void *pMemory,const size_t numBytes)
{
 PARK_Handle parkHandle;
 if(numBytes < sizeof(PARK_Obj))
  return((PARK_Handle)NULL);

 // assign the handle
 parkHandle = (PARK_Handle)pMemory;

 return(parkHandle);
}
```

init 方法仅有两个参数，第一个参数是对象的地址，第二个参数是对象的大小（以字节为单位）。 创建对象后，将使用其它方法。

3.2.2.2 Set 方法

set 方法将一个值放入对象包含的变量中。 在下面的帕克变换示例代码中，**set** 函数将正弦值和余弦值指定给 sinTh 和 cosTh 对象变量。

```
static inline void PARK_setup(PARK_Handle parkHandle,const _iq angle_pu)
{
 PARK_Obj *park = (PARK_Obj *)parkHandle;

 park->sinTh = _IQsinPU(angle_pu);
 park->cosTh = _IQcosPU(angle_pu);

 return;
}
```

在帕克变换示例中，set 方法的参数为对象句柄和角度 θ。 Set 函数不返回任何值。

3.2.2.3 Get 方法

Get 方法返回对象变量。 帕克对象中仅包含两个变量。 由于帕克对象外部需要对象中包含的两个变量，因此有两个 **get** 方法。 其中一个 **get** 方法如下所示。

```
static inline _iq PARK_getSinTh(PARK_Handle parkHandle)
{
 PARK_Obj *park = (PARK_Obj *)parkHandle;

 return(park->sinTh);
}
```

get 方法仅返回该方法名称所对应的变量。 在上面的示例代码中，帕克对象 **get** 方法返回 sinTh 变量。 对象的句柄是传递给 **get** 方法的唯一变量。 **get** 方法仅返回一个变量。

3.2.2.4 Run 方法

Run 方法执行对象变量的计算。 使用嵌入式软件时，**run** 方法可能还操作外设或其它一些硬件。 帕克 **run** 方法计算输入矢量 $\{I_d, I_q\}$ 的帕克变换，然后返回输出矢量 $\{I_\alpha, I_\beta\}$。 帕克 **run** 方法的代码如下所示。

```
static inline void PARK_run(PARK_Handle parkHandle,const MATH_vec2
                *pInVec,MATH_vec2 *pOutVec)
{
 PARK_Obj *park = (PARK_Obj *)parkHandle;
 _iq sinTh = park->sinTh;
 _iq cosTh = park->cosTh;
 _iq value_0 = pInVec->value[0];
 _iq value_1 = pInVec->value[1];
 pOutVec->value[0] = _IQmpy(value_0,cosTh) + _IQmpy(value_1,sinTh);
 pOutVec->value[1] = _IQmpy(value_1,cosTh) - _IQmpy(value_0,sinTh);
 return;
} // end of PARK_run() function
```

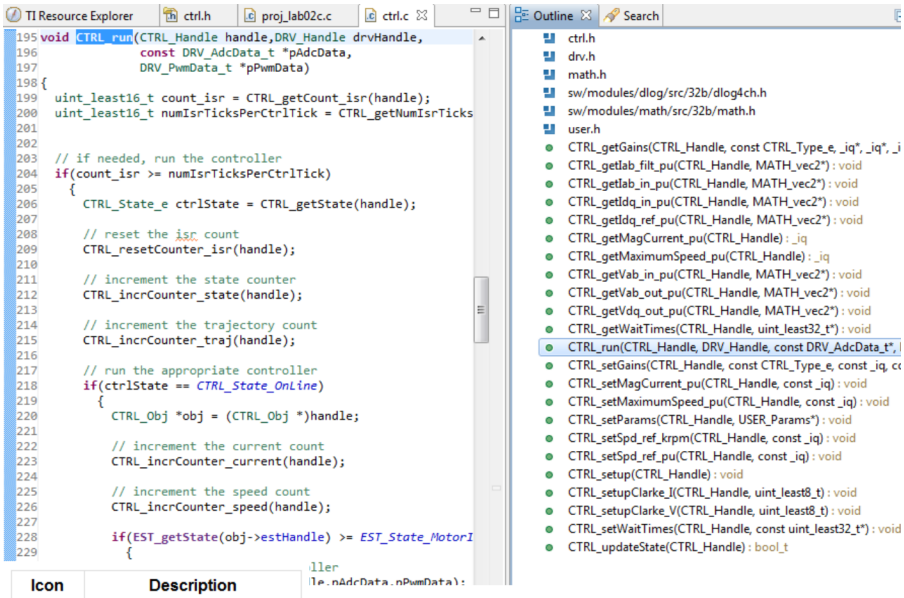
在 **run** 方法中，第一个参数是对象的句柄，后面的参数是单个量时的输入和输出变量或者矢量的指针。 **run** 方法不返回任何值。

3.3 InstaSPIN-FOC API

InstaSPIN-FOC 的所有功能都可通过丰富的 API 访问。 无论 InstaSPIN-FOC 在 ROM 中还是在用户内存中，该 API 都保持不变。 在本节，我们将介绍最常用的提供变量访问和使应用程序实现系统控制的函数。 本指南结尾的实验示例项目中使用了这些函数。 有关最新的完整 API 函数列表，请参见 CCStudio 内的 Resource Explorer。

在软件开发过程中，另一个特别有用的 API 函数资源是 CCStudio 内的 Outline View。 通过此功能可利用一个完整的超链接在多个文件之间导航，该超链接列出了您正在使用的 CCStudio 项目文件内的所有符号。 通过以下 CCStudio 菜单访问此视图：*窗口 (Window)* -> *显示视图 (Show View)* -> *大纲 (Outline)*。

下面是使用此视图显示 ctrl.c 的屏幕截图。 请注意，单击大纲 (Outline) 窗口中的函数名称时，源 (Source) 窗口中的光标会突出显示相关代码。 您可以快速浏览一个文件内的函数和所有符号，这对于 InstaSPIN-FOC 中的大量 API 函数特别有用。



Icon	Description
	Class
	Namespace
	Macro Definition
	Enum
	Enumerator
	Variable
	Field private
	Field protected
	Field public
	Include
	Method private
	Method protected
	Method public
	Struct
	Type definition
	Union
	Function

使用 Outline View 显示 ctrl.h 时，您会注意到一些函数没有列出源代码。 这些函数的文件名旁边有一个白底的绿色圆圈，而不是绿色的实心圆圈。 这表示它是必须保留在 ROM 中的几个文件之一，因为它是直接与 FAST 估算器相连的函数。 下图显示了这种情况的一个示例，其中 ***CTRL_initCtrl()*** 函数没有源代码，但 ***CTRL_isError()*** 函数有源代码。

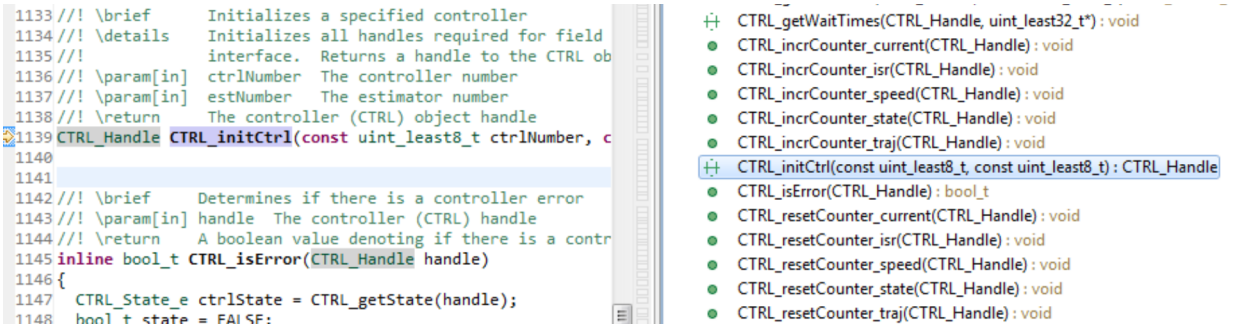


图 3-1 中的方框图很好地概述了 InstaSPIN-FOC 系统的与用户内存和 ROM 有关的函数和变量。 请注意关键函数：CTRL_run、CTRL_setup、EST_run、HAL_run、HAL_acqAdcInt 和 HAL_readAdcData。 另外，图中显示的变量全部可用。 例如，用于 Id 和 Iq 的 PI 的 Ki 增益可使用 CTRL_getKi 函数读取，并使用 CTRL_setKi 设置。 目的是提供对所有函数和变量的完全访问。

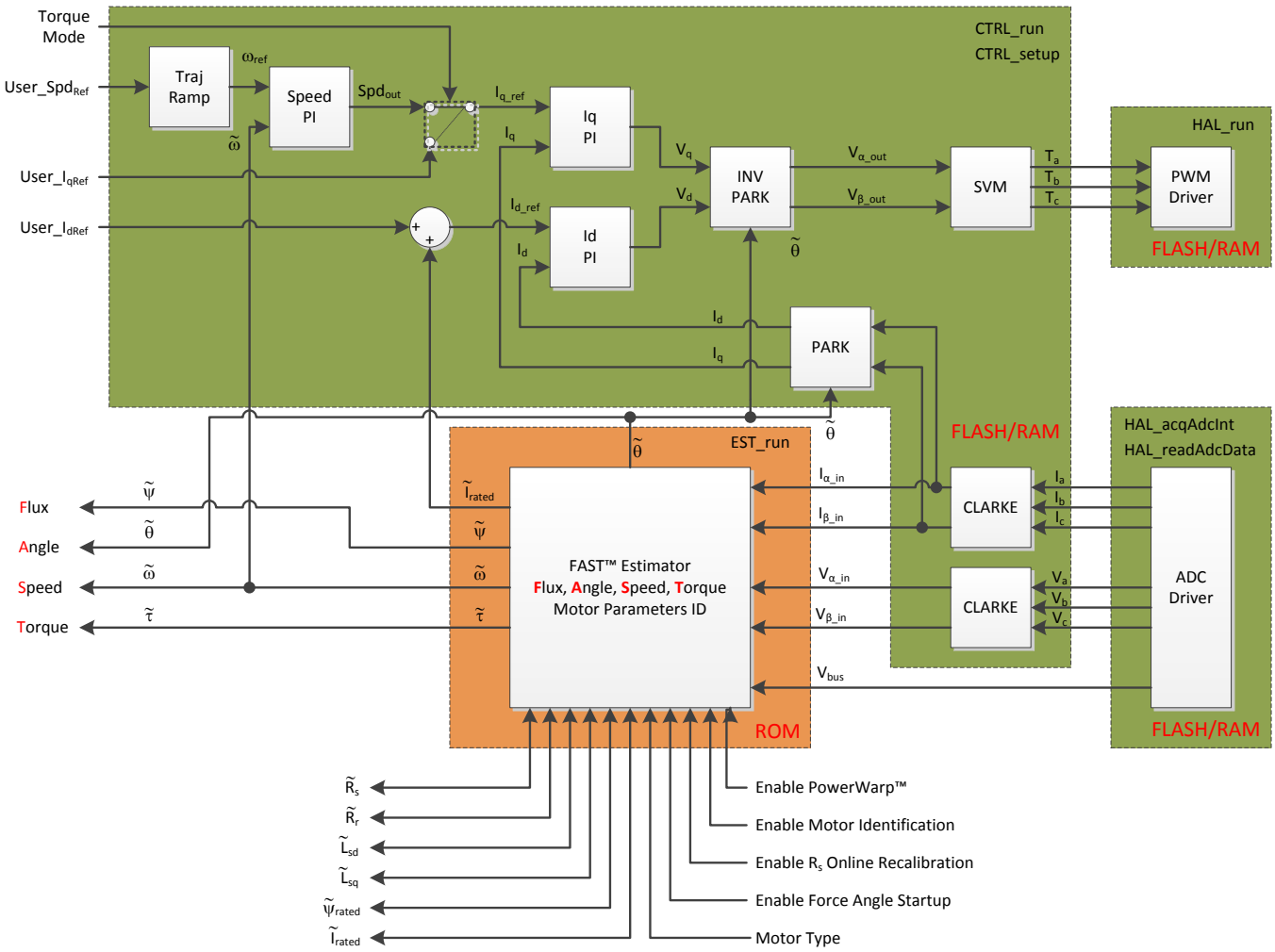


图 **3-1.** 用户内存中的 **InstaSPIN-FOC** 方框图，不包括 **ROM** 中的 **FAST**

API 可分组为以下四个类别：

- 控制器 – ctrl.c（可移动到用户内存的软件）
- 估算器 – FAST 库（ROM 中的 FAST 估算器）
- HAL – hal.c（硬件抽象层）
- 用户 – user.c（用户设置）

接下来的几节列出了每个分组中的常用函数。

对于 F2802xF 器件，API 在用户内存中有一个附加库：

- 公共库 – fast_public.lib（必须在用户内存中加载的软件）

接下来的几节列出了每个分组中的常用函数。

3.3.1 控制器 API 函数 - ctrl.c、ctrl.h、CTRL_obj.h

3.3.1.1 CTRL 枚举和结构

CTRL_Obj

定义控制器 (CTRL) 对象。 控制器对象实现所有 FOC 算法和估算器功能。

```
typedef struct _CTRL_Obj_
{
  CTRL_Version       version;             //!< the controller version
  CTRL_State_e       state;               //!< the current state of the controller
  CTRL_State_e       prevState;           //!< the previous state of the
controller
  CTRL_ErrorCode_e   errorCode;           //!< the error code for the controller
  CLARKE_Handle      clarkeHandle_I; //!< the handle for the current Clarke
transform
  CLARKE_Obj         clarke_I;            //!< the current Clarke transform object
  CLARKE_Handle      clarkeHandle_V; //!< the handle for the voltage Clarke
transform
  CLARKE_Obj         clarke_V;            //!< the voltage Clarke transform object
  EST_Handle         estHandle;           //!< the handle for the parameter
estimator
  PARK_Handle        parkHandle;          //!< the handle for the Park object
  PARK_Obj           park;                //!< the Park transform object
  PID_Handle         pidHandle_Id;        //!< the handle for the Id PID
controller
  PID_Obj            pid_Id;              //!< the Id PID controller object
  PID_Handle         pidHandle_Iq;        //!< the handle for the Iq PID
controller
  PID_Obj            pid_Iq;              //!< the Iq PID controller object
  PID_Handle         pidHandle_spd;   //!< the handle for the speed PID controller
  PID_Obj            pid_spd;             //!< the speed PID controller object
  IPARK_Handle       iparkHandle;     //!< the handle for the inverse Park
transform
  IPARK_Obj          ipark;               //!< the inverse Park transform object
  SVGEN_Handle       svgenHandle;     //!< the handle for the space vector
generator
  SVGEN_Obj          svgen;               //!< the space vector generator object
  TRAJ_Handle        trajHandle_Id;   //!< the handle for the Id trajectory
generator
  TRAJ_Obj           traj_Id;             //!< the Id trajectory generator object
  TRAJ_Handle        trajHandle_spd;  //!< handle for the speed trajectory
generator
  TRAJ_Obj           traj_spd;            //!< the speed trajectory generator
object
  TRAJ_Handle        trajHandle_spdMax;  //!< handle for max speed traj generator
```

```
  TRAJ_Obj           traj_spdMax;     //!< the max speed trajectory generator
object
  MOTOR_Params       motorParams;         //!< the motor parameters
  uint_least32_t     waitTimes[CTRL_numStates];
//!< an array of wait times for each state, estimator clock counts
  uint_least32_t     counter_state;           //!< the state counter
  uint_least16_t     numIsrTicksPerCtrlTick;  //!< # of isr ticks per controller
tick
  uint_least16_t     numCtrlTicksPerCurrentTick;
//!< # of controller ticks per current controller tick
  uint_least16_t     numCtrlTicksPerSpeedTick;
//!< # of controller ticks per speed controller tick
  uint_least16_t     numCtrlTicksPerTrajTick;
//!< # of controller ticks per trajectory tick
  uint_least32_t     ctrlFreq_Hz;         //!< Defines the controller frequency,
Hz
  uint_least32_t     trajFreq_Hz;         //!< Defines the trajectory frequency,
Hz
  _iq                trajPeriod_sec;      //!< Defines the trajectory period, sec
  float_t            ctrlPeriod_sec;      //!< Defines the controller period, sec
  _iq                maxVsMag_pu; //!< the maximum voltage vector that is
allowed, pu
  MATH_vec2          Iab_in;              //!< the Iab input values
  MATH_vec2          Iab_filt;            //!< the Iab filtered values
  MATH_vec2          Idq_in;              //!< the Idq input values
  MATH_vec2          Vab_in;              //!< the Vab input values
  _iq                spd_out;             //!< the speed output value
  MATH_vec2          Vab_out;             //!< the Vab output values
  MATH_vec2          Vdq_out;             //!< the Vdq output values
  float_t            Rhf;                 //!< the Rhf value
  float_t            Lhf;                 //!< the Lhf value
  float_t            RoverL;              //!< the R/L value
  _iq                Kp_Id;               //!< the desired Kp_Id value
  _iq                Kp_Iq;               //!< the desired Kp_Iq value
  _iq                Kp_spd;              //!< the desired Kp_spd value
  _iq                Ki_Id;               //!< the desired Ki_Id value
  _iq                Ki_Iq;               //!< the desired Ki_Iq value
  _iq                Ki_spd;              //!< the desired Ki_spd value
  _iq                Kd_Id;               //!< the desired Kd_Id value
  _iq                Kd_Iq;               //!< the desired Kd_Iq value
  _iq                Kd_spd;              //!< the desired Kd_spd value
  _iq                Ui_Id;               //!< the desired Ui_Id value
  _iq                Ui_Iq;               //!< the desired Ui_Iq value
  _iq                Ui_spd;              //!< the desired Ui_spd value
  MATH_vec2          Idq_ref;             //!< the Idq reference values, pu
  _iq                IdRated;             //!< the Id rated current, pu
  _iq                spd_ref;             //!< the speed reference, pu
  _iq                spd_max;             //!< the maximum speed, pu
  uint_least16_t     counter_current;     //!< the isr counter
  uint_least16_t     counter_isr;         //!< the isr counter
  uint_least16_t     counter_speed;       //!< the speed counter
  uint_least16_t     counter_traj;        //!< the traj counter
  bool             flag_enableCtrl;       //!< a flag to enable the controller
  bool             flag_enableDcBusComp;
//!< a flag to enable the DC bus compensation in the controller
  bool             flag_enablePowerWarp;  //!< a flag to enable PowerWarp
  bool             flag_enableOffset;
//!< a flag to enable offset estimation after idle state
  bool             flag_enableSpeedCtrl;  //!< a flag to enable the speed
controller
  bool             flag_enableUserMotorParams;
//!< a flag to use known motor parameters from user.h file
} CTRL_Obj;
```

CTRL_State_e

控制器状态的枚举

```
typedef enum {
  CTRL_State_Error=0,          //!< the controller error state
  CTRL_State_Idle,             //!< the controller idle state
  CTRL_State_OffLine,          //!< the controller offline state
  CTRL_State_OnLine,           //!< the controller online state
  CTRL_numStates               //!< the number of controller states
} CTRL_State_e;
```

CTRL_ErrorCode_e

错误代码的枚举

```
typedef enum
{
  CTRL_ErrorCode_NoError=0,        //!< no error error code
  CTRL_ErrorCode_IdClip,           //!< Id clip error code
  CTRL_ErrorCode_EstError,         //!< estimator error code
  CTRL_numErrorCodes               //!< the number of controller error codes
} CTRL_ErrorCode_e;
```

CTRL_TargetProc_e

目标处理器的枚举

```
typedef enum
{
  CTRL_TargetProc_2806x=0,   //!< 2806x processor
  CTRL_TargetProc_2805x,     //!< 2805x processor
  CTRL_TargetProc_2803x,     //!< 2803x processor
  CTRL_TargetProc_2802x,     //!< 2802x processor
  CTRL_TargetProc_Unknown    //!< Unknown processor
} CTRL_TargetProc_e;
```

CTRL_Type_e

目标处理器的枚举

```
typedef enum
{
  CTRL_Type_PID_spd=0,             //!< PID Speed controller
  CTRL_Type_PID_Id,                //!< PID Id controller
  CTRL_Type_PID_Iq                 //!< PID Iq controller
} CTRL_Type_e;
```

CTRL_Version

定义控制器 (CTRL) 版本号

```
typedef struct _CTRL_Version_
{
  uint16_t rsvd;                 //!< reserved value
  CTRL_TargetProc_e targetProc;  //!< the target processor
  uint16_t major;                //!< the major release number
  uint16_t minor;                //!< the minor release number
} CTRL_Version;
```

3.3.1.2 CTRL 状态控制和错误处理

CTRL_initCtrl() ***ctrlHandle = CTRL_initCtrl(ctrlNumber,estNumber)***

初始化指定的控制器

ctrlNumber： FOC (CTRL) 控制器编号 – FOC 控制器的编号。

estNumber： 估算器 (EST) 编号 – InstaSPIN 估算器的编号。

返回： 控制器 (CTRL) 对象句柄 - 返回的句柄指向特定的估算器和控制器。

CTRL_updateState () ***bool CTRL_updateState(CTRL_Handle handle)***

反馈控制器状态是否发生变化

句柄： 控制器 (CTRL) 句柄

返回： 指示状态是否变化的布尔值 (true/false)

CTRL_isError () ***inline bool CTRL_isError(CTRL_Handle handle)***

确定是否存在控制器错误

句柄： 控制器 (CTRL) 句柄

返回： 指示是否存在控制器错误的布尔值 (true/false)

CTRL_checkForErrors ()
inline void CTRL_checkForErrors(CTRL_Handle handle)

检查估算器是否存在错误，如果存在，则将控制器状态设置为错误状态

句柄： 控制器 (CTRL) 句柄

错误： CTRL_State_Error、CTRL_ErrorCode_EstError

3.3.1.3 CTRL Get 函数

CTRL_getCount_current ()

inline uint_least16_t CTRL_getCount_current(CTRL_Handle handle)

获取当前循环计数

句柄： 控制器 (CTRL) 句柄

返回： 当前循环计数，obj->counter_current

CTRL_getCount_isr ()

inline uint_least16_t CTRL_getCount_isr(CTRL_Handle handle)

获取 isr 计数

句柄： 控制器 (CTRL) 句柄

返回： isr 计数，obj->counter_isr

CTRL_getCount_speed ()

inline uint_least16_t CTRL_getCount_speed(CTRL_Handle handle)

获取速度环路计数

句柄： 控制器 (CTRL) 句柄

返回： 速度环路计数，obj->counter_speed

CTRL_getCount_state ()

inline uint_least32_t CTRL_getCount_state(CTRL_Handle handle)

获取状态计数

句柄： 控制器 (CTRL) 句柄

返回： 状态计数，obj->counter_state

CTRL_getCount_traj ()

inline uint_least32_t CTRL_getCount_state(CTRL_Handle handle)

获取轨迹环路计数

句柄： 控制器 (CTRL) 句柄

返回： 轨迹环路计数，obj->counter_traj

CTRL_getCtrlFreq () ***inline uint_least32_t CTRL_ getCtrlFreq(CTRL_Handle handle)***

获取控制器执行频率

句柄： 控制器 (CTRL) 句柄

返回： 控制器执行频率，Hz，obj->ctrlFreq_Hz

CTRL_getCtrlPeriod_sec ()

inline float_t CTRL_getCtrlPeriod_sec(CTRL_Handle handle)

获取控制器执行周期

句柄： 控制器 (CTRL) 句柄

返回： 控制器执行周期，秒，obj->ctrlPeriod_sec

CTRL_getErrorCode ()

inline CTRL_ErrorCode_e CTRL_getErrorCode(CTRL_Handle handle)

获取控制器 (CTRL) 对象的错误代码

句柄： 控制器 (CTRL) 句柄

返回： 错误代码，obj->errorCode

CTRL_getEstHandle ()

inline EST_Handle CTRL_getEstHandle(CTRL_Handle handle)

获取给定控制器的估算器句柄

句柄： 控制器 (CTRL) 句柄

返回： 给定控制器的估算器句柄，obj->estHandle

CTRL_getFlag_enableCtrl ()

inline bool CTRL_getFlag_enableCtrl(CTRL_Handle handle)

获取估算器的使能控制器标志值

句柄： 控制器 (CTRL) 句柄

返回： 使能控制器标志值，obj->flag_enableCtrl

CTRL_getFlag_enableDcBusComp ()

inline bool CTRL_getFlag_enableDcBusComp(CTRL_Handle handle)

获取估算器的使能直流总线补偿标志值

句柄： 控制器 (CTRL) 句柄

返回： 使能直流总线补偿标志值，obj-> flag_enableDcBusComp

CTRL_getFlag_enablePowerWarp ()

inline bool CTRL_getFlag_enablePowerWarp(CTRL_Handle handle)

获取估算器的 PowerWarp 使能标志值

句柄： 控制器 (CTRL) 句柄

返回： PowerWarp 使能标志值，obj-> flag_enablePowerWarp

CTR CTRL_getFlag_enableOffset ()

inline bool CTRL_getFlag_enableOffset(CTRL_Handle handle)

获取控制器的使能偏移标志值

句柄： 控制器 (CTRL) 句柄

返回： 使能偏移标志值，obj->flag_enableOffset

CTRL_getFlag_enableSpeedCtrl ()

inline bool CTRL_getFlag_enableSpeedCtrl(CTRL_Handle handle)

获取控制器的使能速度控制标志值

句柄： 控制器 (CTRL) 句柄

返回： 使能速度控制标志值，obj->flag_enableSpeedCtrl

CTRL_getFlag_enableUserMotorParams ()

inline bool CTRL_getFlag_enableSpeedCtrl(CTRL_Handle handle)

获取估算器的使能用户电机参数标志值

句柄： 控制器 (CTRL) 句柄

返回： 使能用户电机参数标志值：

- true = 使用 user.h 中的用户电机参数
- false = 执行电机参数估算

CTRL_getGains ()

void CTRL_getGains(CTRL_Handle handle, const CTRL_Type_e ctrlType, iq *pKp,_iq *pKi,_iq *pKd)

更新控制器对象中的 Kp、Ki 和 Kd

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

pKp： Kp 值的指针，pu

pKi： Ki 值的指针，pu

pKd： Kd 值的指针，pu

CTRL_getlab_filt_pu ()

void CTRL_getlab_filt_pu (CTRL_Handle handle,MATH_vec2 *plab_filt_pu)

更新控制器对象中的 alpha/beta 滤波电流矢量值

句柄： 控制器 (CTRL) 句柄

plab_filt_pu： alpha/beta 滤波电流矢量的矢量值，pu

CTRL_getIab_filt_addr ()

inline MATH_vec2 *CTRL_getIab_filt_addr(CTRL_Handle handle)

获取控制器中的 alpha/beta 滤波电流矢量内存地址

句柄： 控制器 (CTRL) 句柄

返回： alpha/beta 滤波电流矢量内存地址，obj->Iab_filt

CTRL_getIab_in_addr ()

inline MATH_vec2 *CTRL_getIab_in_addr(CTRL_Handle handle)

获取控制器中的 alpha/beta 电流输入矢量内存地址

句柄： 控制器 (CTRL) 句柄

返回： alpha/beta 电流输入矢量内存地址，obj-> Iab_in

CTRL_getIab_in_pu ()

void CTRL_getIab_in_pu(CTRL_Handle handle,MATH_vec2 *pIab_in_pu);

获取控制器中的 alpha/beta 电流输入矢量值

句柄： 控制器 (CTRL) 句柄

pIab_in_pu： alpha/beta 电流输入矢量的矢量值，pu

CTRL_getId_in_pu ()

inline _iq CTRL_getId_in_pu(CTRL_Handle handle)

获取控制器的直流电流输入值

句柄： 控制器 (CTRL) 句柄

返回： 直流电流输入值，pu，obj->Idq_in.value[0]

CTRL_getId_ref_pu ()

inline _iq CTRL_getId_ref_pu(CTRL_Handle handle)

获取控制器的直流电流 (Id) 参考值

句柄： 控制器 (CTRL) 句柄

返回： 直流电流参考值，pu，obj->Idq_ref.value[0]

CTRL_getIdq_in_addr ()

inline MATH_vec2 *CTRL_getIdq_in_addr(CTRL_Handle handle)

获取控制器中的直流/正交电流输入矢量内存地址

句柄： 控制器 (CTRL) 句柄

返回： 直流/正交电流输入矢量内存地址，obj-> Idq_in

CTRL_getIdq_in_pu ()

void CTRL_getIdq_in_pu(CTRL_Handle handle,MATH_vec2 *pIdq_in_pu);

获取控制器中的直流/正交电流输入矢量值

句柄： 控制器 (CTRL) 句柄

pIdq_in_pu： 直流/正交输入电流矢量的矢量值，pu

CTRL_getIdq_ref_pu ()

void CTRL_getIdq_ref_pu(CTRL_Handle handle,MATH_vec2 *pIdq_ref_pu);

获取控制器中的直流/正交电流参考矢量值

句柄： 控制器 (CTRL) 句柄

pIdq_ref_pu： 直流/正交电流参考矢量的矢量值，pu

CTRL_getIdRated_pu ()

inline _iq CTRL_getIdRated_pu(CTRL_Handle handle)

获取控制器的 Id 额定电流值

句柄： 控制器 (CTRL) 句柄

返回： Id 额定电流值，pu，obj-> IdRated

CTRL_getIq_in_pu ()

inline _iq CTRL_getIq_in_pu(CTRL_Handle handle)

获取控制器的正交电流输入值

句柄： 控制器 (CTRL) 句柄

返回： 正交电流输入值，pu，obj-> Idq_in.value[1]

CTRL_getIq_ref_pu ()

inline _iq CTRL_getIq_ref_pu(CTRL_Handle handle)

获取控制器的正交电流 (Id) 参考值

句柄： 控制器 (CTRL) 句柄

返回： 正交电流参考值，pu，obj-> Idq_ref.value [1]

CTRL_getKi ()

_iq CTRL_getKi (CTRL_Handle handle,const CTRL_Type_e ctrlType)

获取控制器状态

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

返回： Ki 值

CTRL_getKd () *_iq CTRL_getKd (CTRL_Handle handle,const CTRL_Type_e ctrlType)*

获取指定控制器的比例增益 (Kd) 值

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

返回： Kd 值

CTRL_getKp () *_iq CTRL_getKp (CTRL_Handle handle,const CTRL_Type_e ctrlType)*

获取控制器状态

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

返回： Kp 值

CTRL_getLhf () *inline float_t CTRL_getLhf(CTRL_Handle handle)*

获取控制器的高频电感 (Lhf) 值

句柄： 控制器 (CTRL) 句柄

返回： 返回：Lhf 值，obj->Lhf

CTRL_getMagCurrent_pu () *_iq CTRL_getMagCurrent_pu(CTRL_Handle handle)*

获取控制器的磁化电流值

句柄： 控制器 (CTRL) 句柄

返回： 磁化电流值

CTRL_getMaxVsMag_pu () *inline _iq CTRL_getMaxVsMag_pu(CTRL_Handle handle)*

获取最大电压矢量

句柄： 控制器 (CTRL) 句柄

返回： 最大电压矢量（值介于 0 和 4/3 之间）

CTRL_getMaximumSpeed_pu ()

_iq CTRL_getMaximumSpeed_pu(CTRL_Handle handle);

获取控制器的最大速度值

句柄： 控制器 (CTRL) 句柄

返回： 最大电压矢量（值介于 0 和 4/3 之间）

CTRL_getMotorRatedFlux ()

inline float_t CTRL_getMotorRatedFlux(CTRL_Handle handle)

获取控制器的电机额定磁通

句柄： 控制器 (CTRL) 句柄

返回： 电机额定磁通，V*sec，obj->motorParams.ratedFlux_VpHz

CTRL_getMotorType ()

inline MOTOR_Type_e CTRL_getMotorType(CTRL_Handle handle)

获取控制器的电机类型

句柄： 控制器 (CTRL) 句柄

返回： 电机类型，obj-> motorParams.type

CTRL_getNumCtrlTicksPerCurrentTick ()

inline uint_least16_t CTRL_getNumCtrlTicksPerCurrentTick(CTRL_Handle handle)

获取每个电流控制器时钟节拍的控制器时钟节拍数

句柄： 控制器 (CTRL) 句柄

返回： 每个估算器时钟节拍的控制器时钟节拍数，obj-> numCtrlTicksPerCurrentTick

CTRL_getNumCtrlTicksPerSpeedTick ()

inline uint_least16_t CTRL_getNumCtrlTicksPerSpeedTick(CTRL_Handle handle)

获取每个速度控制器时钟节拍的控制器时钟节拍数

句柄： 控制器 (CTRL) 句柄

返回： 每个速度时钟节拍的控制器时钟节拍数，obj-> numCtrlTicksPerSpeedTick

CTRL_getNumCtrlTicksPerTrajTick ()

inline uint_least16_t CTRL_getNumCtrlTicksPerTrajTick(CTRL_Handle handle)

获取每个轨迹时钟节拍的控制器时钟节拍数

句柄： 控制器 (CTRL) 句柄

返回： 每个轨迹时钟节拍的控制器时钟节拍数，obj-> numCtrlTicksPerTrajTick

CTRL_getNumIsrTicksPerCtrlTick ()

inline uint_least16_t CTRL_getNumIsrTicksPerCtrlTick(CTRL_Handle handle)

获取每个控制器时钟节拍的中断服务程序 (ISR) 时钟节拍数

句柄： 控制器 (CTRL) 句柄

返回： 每个控制器时钟节拍的中断服务程序 (ISR) 时钟节拍数，obj-> numIsrTicksPerCtrlTick

CTRL_getRefValue_pu ()

inline _iq CTRL_getRefValue_pu(CTRL_Handle handle,const CTRL_Type_e ctrlType)

获取指定控制器的参考值

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

返回： 参考值，pu

CTRL_getRhf ()

inline float_t CTRL_getRhf(CTRL_Handle handle)

获取控制器的高频电阻 (Rhf) 值

句柄： 控制器 (CTRL) 句柄

返回： Rhf 值，obj->Rhf

CTRL_getRoverL ()

inline float_t CTRL_getRoverL(CTRL_Handle handle)

获取控制器的 R/L 值

句柄： 控制器 (CTRL) 句柄

返回： R/L 值，obj-> RoverL

CTRL_getSpd_max_pu () *i*

nline _iq CTRL_getSpd_max_pu(CTRL_Handle handle)

获取控制器的最大速度值

句柄： 控制器 (CTRL) 句柄

返回： 最大速度值，pu，obj-> spd_max

CTRL_getSpd_out_addr ()

inline _iq *CTRL_getSpd_out_addr(CTRL_Handle handle)

获取控制器的输出速度内存地址

句柄： 控制器 (CTRL) 句柄

返回： 输出速度内存地址，obj-> spd_out

CTRL_getSpd_out_pu ()

inline _iq CTRL_getSpd_out_pu(CTRL_Handle handle)

获取控制器的输出速度值

句柄： 控制器 (CTRL) 句柄

返回： 输出速度值，pu，obj-> spd_out

CTRL_getSpd_ref_pu ()

inline _iq CTRL_getSpd_ref_pu(CTRL_Handle handle)

获取控制器的输出速度基准值

句柄： 控制器 (CTRL) 句柄

返回： 输出速度基准值，pu，obj-> spd_ref

CTRL_getSpd_int_ref_pu ()

inline _iq CTRL_getSpd_int_ref_pu(CTRL_Handle handle)

获取控制器的输出速度中间基准值

句柄： 控制器 (CTRL) 句柄

返回： 输出速度中间基准值，pu，obj-> trajHandle_ref

CTRL_getState ()

CTRL_State_e CTRL_getState(CTRL_Handle handle)

获取控制器状态

句柄： 控制器 (CTRL) 句柄

返回： 控制器状态

CTRL_getTrajFreq ()

inline uint_least32_t CTRL_getTrajFreq(CTRL_Handle handle)

获取轨迹执行频率

句柄： 控制器 (CTRL) 句柄

返回： 轨迹执行频率，Hz，obj->trajFreq_Hz

CTRL_getTrajPeriod_sec ()

inline _iq CTRL_getTrajPeriod_sec(CTRL_Handle handle)

获取轨迹执行频率

句柄： 控制器 (CTRL) 句柄

返回： 轨迹执行周期，秒，obj-> trajPeriod_sec

CTRL_getTrajStep ()

void CTRL_getTrajStep(CTRL_Handle handle);

获取轨迹步长

句柄： 控制器 (CTRL) 句柄

返回： 轨迹执行频率，Hz

CTRL_getUi ()

inline _iq CTRL_getUi(CTRL_Handle handle,const CTRL_Type_e ctrlType)

获取指定控制器的积分器 (Ui) 值

句柄： 控制器 (CTRL) 句柄

返回： Ui 值

CTRL_getVab_in_pu ()

void CTRL_getVab_in_pu(CTRL_Handle handle,MATH_vec2 *pVab_in_pu);

获取控制器中的 alpha/beta 电压输入矢量值

句柄： 控制器 (CTRL) 句柄

pVab_in_pu： alpha/beta 电压输入矢量的矢量值，pu

CTRL_getVab_out_addr ()

inline MATH_vec2 *CTRL_getVab_out_addr(CTRL_Handle handle)

获取控制器中的 alpha/beta 电压输出矢量内存地址

句柄： 控制器 (CTRL) 句柄

返回： alpha/beta 电压输出矢量内存地址，&(obj->Vab_out)

CTRL_getVab_out_pu ()

void CTRL_getVab_out_pu(CTRL_Handle handle,MATH_vec2 *pVab_out_pu);

获取控制器中的 alpha/beta 电压输出矢量值

句柄： 控制器 (CTRL) 句柄

返回： alpha/beta 电压输出矢量的矢量值，pu

CTRL_getVd_out_addr ()

inline _iq *CTRL_getVd_out_addr(CTRL_Handle handle)

获取控制器中的直流电压输出值内存地址

句柄： 控制器 (CTRL) 句柄

返回： 直流电压输出值内存地址，&(obj->Vdq_out.value[0])

CTRL_getVd_out_pu ()

inline _iq CTRL_getVd_out_pu(CTRL_Handle handle)

获取控制器的直流电压输出值

句柄： 控制器 (CTRL) 句柄

返回： 直流电压输出值，pu，obj->Vdq_out.value[0])

CTRL_getVdq_out_addr ()

inline MATH_vec2 *CTRL_getVdq_out_addr(CTRL_Handle handle)

获取控制器中的直流/正交电压输出矢量内存地址

句柄： 控制器 (CTRL) 句柄

返回： 直流/正交电压输出矢量内存地址，&(obj->Vdq_out)

CTRL_getVdq_out_pu ()

void CTRL_getVdq_out_pu(CTRL_Handle handle,MATH_vec2 *pVdq_out_pu);

获取控制器中的直流/正交电压输出矢量值

句柄： 控制器 (CTRL) 句柄

pVdq_out_pu： 直流/正交电压输出矢量的矢量值，pu

CTRL_getVersion ()

void CTRL_getVersion(CTRL_Handle handle,CTRL_Version *pVersion);

获取控制器版本号

句柄： 控制器 (CTRL) 句柄

pVersion： 版本的指针

CTRL_getVq_out_addr ()

inline _iq *CTRL_getVq_out_addr(CTRL_Handle handle)

获取控制器中的正交电压输出值内存地址

句柄： 控制器 (CTRL) 句柄

返回： 正交电压输出值内存地址，&(obj->Vdq_out.value[1])

CTRL_getVq_out_pu ()

inline _iq CTRL_getVq_out_pu(CTRL_Handle handle)

获取控制器的正交电压输出值

句柄： 控制器 (CTRL) 句柄

返回： 正交电压输出值，pu，obj->Vdq_out.value[1])

CTRL_getWaitTime ()

inline uint_least32_t CTRL_getWaitTime(CTRL_Handle handle,const CTRL_State_e ctrlState)

获取给定状态的等待时间

句柄： 控制器 (CTRL) 句柄

ctrlState： 控制器状态

返回： 等待时间，控制器时钟计数，waitTimes[ctrlState]

3.3.1.4 CTRL 计数器函数

CTRL_incrCounter_current ()

inline void CTRL_incrCounter_current(CTRL_Handle handle)

递增电流计数器

句柄： 控制器 (CTRL) 句柄

counter_current： 递增的 CTRL 对象成员

CTRL_incrCounter_isr ()

inline void CTRL_incrCounter_isr(CTRL_Handle handle)

递增 ISR 计数器

句柄： 控制器 (CTRL) 句柄

counter_isr： 递增的 CTRL 对象成员

CTRL_incrCounter_speed ()

inline void CTRL_incrCounter_speed(CTRL_Handle handle)

递增速度计数器

句柄： 控制器 (CTRL) 句柄

counter_speed： 递增的 CTRL 对象成员

CTRL_incrCounter_state ()

inline void CTRL_incrCounter_state(CTRL_Handle handle)

递增状态计数器

句柄： 控制器 (CTRL) 句柄

counter_state： 递增的 CTRL 对象成员

CTRL_incrCounter_traj ()

inline void CTRL_incrCounter_traj(CTRL_Handle handle)

递增轨迹计数器

句柄： 控制器 (CTRL) 句柄

counter_traj： 递增的 CTRL 对象成员

CTRL_resetCounter_current ()

inline void CTRL_resetCounter_current(CTRL_Handle handle)

将电流计数器重置为 0

句柄： 控制器 (CTRL) 句柄

counter_current: 重置为 0 的 CTRL 对象成员

CTRL_resetCounter_isr ()

inline void CTRL_resetCounter_isr(CTRL_Handle handle)

将 ISR 计数器重置为 0

句柄: 控制器 (CTRL) 句柄

counter_isr: 重置为 0 的 CTRL 对象成员

CTRL_resetCounter_speed ()

inline void CTRL_resetCounter_speed(CTRL_Handle handle)

将速度计数器重置为 0

句柄: 控制器 (CTRL) 句柄

counter_speed: 重置为 0 的 CTRL 对象成员

CTRL_resetCounter_state ()

inline void CTRL_resetCounter_state(CTRL_Handle handle)

将状态计数器重置为 0

句柄: 控制器 (CTRL) 句柄

counter_state: 重置为 0 的 CTRL 对象成员

CTRL_resetCounter_traj ()

inline void CTRL_resetCounter_traj(CTRL_Handle handle)

将轨迹计数器重置为 0

句柄: 控制器 (CTRL) 句柄

counter_traj: 重置为 0 的 CTRL 对象成员

3.3.1.5 CTRL Set 函数

CTRL_setCtrlFreq_Hz ()

nline void CTRL_setCtrlFreq_Hz(CTRL_Handle handle,const uint_least32_t ctrlFreq_Hz)

设置控制器频率

句柄： 控制器 (CTRL) 句柄

ctrlFreq_Hz： 控制器频率，Hz

CTRL_setCtrlFreq_sec ()

inline void CTRL_setCtrlPeriod_sec(CTRL_Handle handle,const float_t ctrlPeriod_sec)

设置控制器执行周期

句柄： 控制器 (CTRL) 句柄

ctrlPeriod_sec： 控制器执行周期，秒

CTRL_setErrorCode ()

inline void CTRL_setErrorCode(CTRL_Handle handle,const CTRL_ErrorCode_e errorCode)

设置控制器中的错误代码

句柄： 控制器 (CTRL) 句柄

errorCode： 错误代码

CTRL_setEstParams ()

void CTRL_setEstParams(EST_Handle estHandle,USER_Params *pUserParams);

设置默认估算器参数。 将 user.h 文件中定义的和 CTRL 使用的所有换算系数复制到 CTRL 对象中。

estHandle： 估算器 (EST) 句柄

pUserParams： 用户参数的指针

CTRL_setFlag_enableCtrl ()

void CTRL_setFlag_enableCtrl(CTRL_Handle handle,const bool_t state)

使能 FOC 控制器（使能电机控制器）

句柄： 控制器 (CTRL) 句柄

状态： 所需状态：

- True -> 使能控制器
- False -> 禁用控制器

CTRL_setFlag_enableDcBusComp ()

void CTRL_setFlag_enableDcBusComp (CTRL_Handle handle,bool_t state)

设置估算器的使能直流总线补偿标志值。 直流总线补偿算法将补偿 PI 控制器的 Iq 和 Id。

句柄： 控制器 (CTRL) 句柄

状态： 所需状态的布尔值

CTRL_setFlag_enablePowerWarp ()

inline void CTRL_setFlag_enablePowerWarp(CTRL_Handle handle,const bool state)

设置估算器的 PowerWarp 使能标志值。 仅当控制感应电机时才使用 PowerWarp。 PowerWarp 会调整磁场级别，根据电机负载使用最少的功耗。

句柄： 控制器 (CTRL) 句柄

状态： 所需状态的布尔值

CTRL_setFlag_enableOffset ()

void CTRL_setFlag_enableOffset(CTRL_Handle handle,const bool_t state)

启用或禁用电压和电流偏移校准

句柄： 控制器 (CTRL) 句柄

状态： 所需状态：

- True -> 执行偏移校准
- False -> 不执行偏移校准

calibrationCTRL_setFlag_enableSpeedCtrl ()

void CTRL_setFlag_enableSpeedCtrl(CTRL_Handle handle,const bool_t state)

启用速度控制模式或启用转矩控制模式（将速度 PI 输出连接到 Iq）

句柄： 控制器 (CTRL) 句柄

状态： 所需状态：

- True -> 启用速度控制（将速度 PI 输出连接到 Iq 输入）
- False -> 禁用速度控制（断开速度 PI 与 Iq 的连接。 Iq 可用于直流输入）

CTRL_setFlag_enableUserMotorParams ()

void CTRL_setFlag_enableUserMotorParams(CTRL_Handle handle,const bool_t state)

设置估算器的使能用户电机参数标志值

句柄： 控制器 (CTRL) 句柄

状态： 所需状态：

- True -> 使用 user.h 中的用户电机参数
- False -> 执行电机参数估算

CTRL_setGains ()

void CTRL_setGains(CTRL_Handle handle,const CTRL_Type_e ctrlType, const _iq Kp,const _iq Ki,const _iq Kd);

设置指定控制器的增益值

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

Kp： Kp 增益值，pu

Ki： Ki 增益值，pu

Kd： Kd 增益值，pu

CTRL_setlab_in_pu ()

inline void CTRL_setlab_in_pu(CTRL_Handle handle,const MATH_vec2 *plab_in_pu)

设置控制器中的 alpha/beta 电流（实验）输入矢量值

句柄： 控制器 (CTRL) 句柄

plab_in_pu： alpha/beta 电流输入矢量的矢量值，pu

CTRL_setlab_filt_pu ()

void CTRL_setlab_filt_pu(CTRL_Handle handle,const MATH_vec2 *plab_filt_pu);

设置控制器中的 alpha/beta 滤波电流矢量值

句柄： 控制器 (CTRL) 句柄

plab_filt_pu： alpha/beta 滤波电流矢量的矢量值，pu

CTRL_setId_ref_pu ()

inline void CTRL_setId_ref_pu(CTRL_Handle handle,const _iq Id_ref_pu)

设置控制器的直流电流 (Id) 参考值

句柄： 控制器 (CTRL) 句柄

Id_ref_pu： 正交电流参考值，pu，obj-> Idq_ref.value [0]

CTRL_setIdq_in_pu ()

inline void CTRL_setIdq_in_pu(CTRL_Handle handle,const MATH_vec2 *pIdq_in_pu)

设置控制器中的直流/正交电流 (Idq) 输入矢量值

句柄： 控制器 (CTRL) 句柄

pIdq_in_pu： 直流/正交电流输入矢量的矢量值，pu，obj-> Idq_in.value [0,1]

CTRL_setIdq_ref_pu ()

inline void CTRL_setIdq_ref_pu(CTRL_Handle handle,const MATH_vec2 *pIdq_ref_pu)

设置控制器中的直流/正交电流 (Idq) 参考矢量值

句柄： 控制器 (CTRL) 句柄

pIdq_ref_pu： 直流/正交电流参考矢量的矢量值，pu，obj-> Idq_ref.value[0,1]

CTRL_setIdRated_pu ()

inline void CTRL_setIdRated_pu(CTRL_Handle handle,const _iq IdRated_pu)

设置控制器的 Id 额定电流值

句柄： 控制器 (CTRL) 句柄

IdRated_pu： Id 额定电流值，pu，obj-> IdRated

CTRL_setIq_ref_pu ()

void CTRL_setIq_ref_pu (CTRL_Handle handle, const _iq IqRef_pu)

设置控制器的正交电流 (Id) 参考值

句柄： 控制器 (CTRL) 句柄

IqRef_pu： 正交电流参考值，pu，Idq_ref.value[1]

CTRL_setKd ()

void CTRL_setKd (CTRL_Handle handle, const CTRL_Type_e ctrlType,const _iq Kd)

设置指定控制器（速度、Id 或 Iq）的微分增益 (Kd) 值

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型 Kd：Kd 值

CTRL_setKi ()

void CTRL_setKi (CTRL_Handle handle, const CTRL_Type_e ctrlType,const _iq Ki)

设置指定控制器（速度、Id 或 Iq）的积分增益 (Kd) 值

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型 Ki：Ki 值

CTRL_setKp ()

void CTRL_setKp (CTRL_Handle handle, const CTRL_Type_e ctrlType,const _iq Kp)

设置指定控制器（速度、Id 或 Iq）的比例增益 (Kp) 值

句柄： 控制器 (CTRL) 句柄

ctrlType: 控制器类型

Kp: Kp 值

CTRL_setLhf () *inline void CTRL_setLhf(CTRL_Handle handle,const float_t Lhf)*

设置控制器的高频电感 (Lhf) 值

句柄: 控制器 (CTRL) 句柄

ctrlType: 控制器类型

Lhf: Lhf 值

CTRL_setMagCurrent_pu ()

void CTRL_setMagCurrent_pu(CTRL_Handle handle,const _iq magCurrent_pu);

设置控制器的磁化电流值

句柄: 控制器 (CTRL) 句柄

magCurrent_pu: 磁化电流值，pu

CTRL_setMaxVsMag_pu ()

inline void CTRL_setMaxVsMag_pu(CTRL_Handle handle,const _iq maxVsMag)

设置控制器中的最大电压矢量

句柄: 控制器 (CTRL) 句柄

maxVsMag: 最大电压矢量（值介于 0 和 4/3 之间），obj->maxVsMag_pu

CTRL_setMaxAccel_pu ()

inline void CTRL_setMaxAccel_pu(CTRL_Handle handle,const _iq maxAccel_pu)

设置速度控制器的最大加速度 设置速度基准的最大加速率

句柄: 控制器 (CTRL) 句柄

maxAccel_pu: 最大加速度（值介于 0 和 1 之间），pu，obj->traj_spd.maxDelta

CTRL_setMaximumSpeed_pu ()

void CTRL_setMaximumSpeed_pu(CTRL_Handle handle,const _iq maxSpeed_pu);

设置控制器的最大速度值

句柄: 控制器 (CTRL) 句柄

maxSpeed_pu: 最大速度值，pu

CTRL_setParams() *void CTRL_setParams(CTRL_Handle handle,USER_Params *pUserParams)*

设置默认控制器参数。 此函数允许在实时操作控制器期间更新换算系数。

句柄： 控制器 (CTRL) 句柄

pUserParams： 用户参数的指针

CTRL_setNumCtrlTicksPerCurrentTick ()

inline void CTRL_setNumCtrlTicksPerCurrentTick(CTRL_Handle handle, const uint_least16_t numCtrlTicksPerCurrentTick)

设置每个电流控制器时钟节拍的控制器时钟节拍数。

句柄： 控制器 (CTRL) 句柄

numCtrlTicksPerCurrentTick： 每个估算器时钟节拍的控制器时钟节拍数，obj-> numCtrlTicksPerCurrentTick

CTRL_setNumCtrlTicksPerSpeedTick ()

inline void CTRL_setNumCtrlTicksPerSpeedTick(CTRL_Handle handle,,const uint_least16_t numCtrlTicksPerSpeedTick)numCtrlTicksPerCurrentTick)

设置每个速度控制器时钟节拍的控制器时钟节拍数。

句柄： 控制器 (CTRL) 句柄

numCtrlTicksPerSpeedTick： 每个速度时钟节拍的控制器时钟节拍数，obj-> numCtrlTicksPerSpeedTick

CTRL_setNumCtrlTicksPerTrajTick ()

inline void CTRL_setNumCtrlTicksPerTrajTick(CTRL_Handle handle, const uint_least16_t numCtrlTicksPerTrajTick)

设置每个轨迹时钟节拍的控制器时钟节拍数。

句柄： 控制器 (CTRL) 句柄

numCtrlTicksPerTrajTick： 每个轨迹时钟节拍的控制器时钟节拍数，obj-> numCtrlTicksPerTrajTick

CTRL_setNumIsrTicksPerCtrlTick () *inline void CTRL_setNumIsrTicksPerCtrlTick(CTRL_Handle handle, const uint_least16_t numIsrTicksPerCtrlTick)*

设置每个控制器时钟节拍的中断服务程序 (ISR) 时钟节拍数。

句柄： 控制器 (CTRL) 句柄

numIsrTicksPerCtrlTick： 每个控制器时钟节拍的 ISR 时钟节拍数

CTRL_setRhf () *inline void CTRL_setRhf(CTRL_Handle handle,const float_t Rhf)*

设置控制器的高频电阻 (Rhf) 值

句柄： 控制器 (CTRL) 句柄

Rhf： Rhf 值，obj->Rhf

CTRL_setRoverL () *inline void CTRL_setRoverL(CTRL_Handle handle,const float_t RoverL)*

设置控制器的 R/L 值

句柄： 控制器 (CTRL) 句柄

RoverL： R/L 值，obj-> RoverL

CTRL_setSpdMax () *void CTRL_setSpdMax (CTRL_Handle handle, const _iq spdMax)*

设置控制器中的 PI 速度基准值

句柄： 控制器 (CTRL) 句柄

spdMax： 速度控制器的最大允许输出

CTRL_setSpd_max_pu ()

inline void CTRL_setSpd_max_pu(CTRL_Handle handle,const _iq maxSpd_pu)

设置控制器的最大速度值

句柄： 控制器 (CTRL) 句柄

maxSpd_pu： 最大速度值，pu

CTRL_setSpd_out_pu ()

inline void CTRL_setSpd_out_pu(CTRL_Handle handle,const _iq spd_out_pu)

设置控制器的输出速度值

句柄： 控制器 (CTRL) 句柄

spd_out_pu： 输出速度值，pu

CTRL_setSpd_ref_pu ()

void CTRL_setSpd_ref_pu(CTRL_Handle handle,const _iq spd_ref_pu);

设置控制器的输出速度基准值

句柄： 控制器 (CTRL) 句柄

spd_ref_pu： 输出速度基准值，pu

CTRL_setSpd_ref_krpm ()

void CTRL_setSpd_ref_krpm(CTRL_Handle handle,const _iq spd_ref_krpm)

设置控制器中的 PI 速度基准值

句柄： 控制器 (CTRL) 句柄

spd_ref_krpm： 输出速度基准值，krpm

CTRL_setState () *inline void CTRL_setState(CTRL_Handle handle,const CTRL_State_e state)*

设置控制器状态

句柄： 控制器 (CTRL) 句柄

状态： 新状态

CTRL_setTrajFreq_Hz ()

inline void CTRL_setTrajFreq_Hz(CTRL_Handle handle,const uint_least32_t trajFreq_Hz)

设置轨迹执行频率

句柄： 控制器 (CTRL) 句柄

trajFreq_Hz： 轨迹执行频率，Hz，obj->trajFreq_Hz

CTRL_setTrajPeriod_sec ()

inline void CTRL_setTrajPeriod_sec(CTRL_Handle handle,const _iq trajPeriod_sec)

设置轨迹执行周期

句柄： 控制器 (CTRL) 句柄

trajPeriod_sec： 轨迹执行周期，秒，obj-> trajPeriod_sec

CTRL_setUi ()

inline void CTRL_setUi(CTRL_Handle handle,const CTRL_Type_e ctrlType,const _iq Ui)

设置指定控制器（速度、Id 或 Iq）的积分器 (Ui) 值

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

Ui： Ui 值

CTRL_setupClarke_I ()

void CTRL_setupClarke_I(CTRL_Handle handle,uint_least8_t numCurrentSensors);

设置电流传感器数。 当读取不同数量的电流时，可使用不同的算法来计算克拉克变换。

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

numCurrentSensors： 电流传感器数

CTRL_setupClarke_V ()

void CTRL_setupClarke_V(CTRL_Handle handle,uint_least8_t numVoltageSensors);

设置电压传感器数。 当读取不同数量的电压时，可使用不同的算法来计算克拉克变换。

句柄： 控制器 (CTRL) 句柄

ctrlType： 控制器类型

numVoltageSensors： 电压传感器数

CTRL_setupEstIdleState ()

void CTRL_setupEstIdleState(CTRL_Handle handle);

针对估算器空闲状态设置控制器和轨迹生成器

句柄： 控制器 (CTRL) 句柄

句柄： 控制器 (CTRL) 句柄

：

CTRL_setupEstOnLineState ()

void CTRL_setupEstOnLineState(CTRL_Handle handle);

针对估算器空闲状态设置控制器和轨迹生成器

句柄： 控制器 (CTRL) 句柄

CTRL_setUserMotorParams ()

void CTRL_setUserMotorParams(CTRL_Handle handle);

使用 user.h 文件中的电机参数设置控制器和估算器

句柄： 控制器 (CTRL) 句柄

CTRL_setVab_in_pu ()

inline void CTRL_setVab_in_pu(CTRL_Handle handle,const MATH_vec2 *pVab_in_pu)

设置控制器中的 alpha/beta 电压输入矢量值

句柄： 控制器 (CTRL) 句柄

CTRL_setVab_out_pu ()

inline void CTRL_setVab_out_pu(CTRL_Handle handle,const MATH_vec2 *pVab_out_pu)

设置控制器中的 alpha/beta 电压输出矢量值

句柄： 控制器 (CTRL) 句柄

CTRL_setVdq_out_pu ()

inline void CTRL_setVdq_out_pu(CTRL_Handle handle,const MATH_vec2 *pVdq_out_pu)

设置控制器中的直流/正交电压输出矢量值

句柄： 控制器 (CTRL) 句柄

pVdq_out_pu： 直流/正交电流输出矢量的矢量值，pu

CTRL_setWaitTimes () *void CTRL_setWaitTimes(CTRL_Handle handle,const uint_least32_t *pWaitTimes)*

设置控制器状态的等待时间

句柄： 控制器 (CTRL) 句柄

pWaitTimes： 等待时间矢量的指针，控制器时钟计数

CTRL_setup () *void CTRL_setup(CTRL_Handle handle)*

设置控制器

句柄： 控制器 (CTRL) 句柄

CTRL_setupCtrl () *void CTRL_setupCtrl(CTRL_Handle handle);*

设置控制器 (CTRL) 对象及所有从属对象（运行 InstaSPIN 状态机）

句柄： 控制器 (CTRL) 句柄

CTRL_setupEst () *void CTRL_setupEst(CTRL_Handle handle);*

设置控制器 (CTRL) 对象及所有从属对象（运行 InstaSPIN 状态机）

句柄： 控制器 (CTRL) 句柄

CTRL_setupTraj () *void CTRL_setupTraj(CTRL_Handle handle);*

设置轨迹 (TRAJ) 对象

句柄： 控制器 (CTRL) 句柄

3.3.1.6 CTRL Run 和 Compute 函数

CTRL_angleDelayComp ()

inline _iq CTRL_angleDelayComp(CTRL_Handle handle, const _iq angle_pu)

运行角度延迟补偿。 此函数采用抽取率计算相位延迟，并对延迟进行补偿。 这使得电压输出可更好地纠正误差。

句柄： 控制器 (CTRL) 句柄

angle_pu： 延迟的角度

返回： 相位延迟补偿角度，angleComp_pu

CTRL_computePhasor ()

inline void CTRL_computePhasor(const _iq angle_pu,MATH_vec2 *pPhasor)

计算给定角度的相量

angle_pu： 角度，pu

pPhasor： 相量矢量值的指针

CTRL_doCurrentCtrl ()

inline bool CTRL_doCurrentCtrl(CTRL_Handle handle)

确定是否应运行电流控制器

句柄： 控制器 (CTRL) 句柄

返回： 指示是否应运行电流控制器的值 (true/false)，结果

CTRL_doSpeedCtrl ()

inline bool CTRL_doSpeedCtrl(CTRL_Handle handle)

确定是否应执行速度控制器

句柄： 控制器 (CTRL) 句柄

返回： 指示是否应执行速度控制器的布尔值 (true/false)

CTRL_run()

void CTRL_run(CTRL_Handle handle,HAL_Handle halHandle,const HAL_AdcData_t *pAdcData, HAL_PwmData_t *pPwmData)

运行电机控制器计算，必须以 ISR 速率调用

句柄： 控制器 (CTRL) 句柄

halHandle： 驱动程序 (HAL) 句柄

pAdcData： “HAL_AdcData_t”类型格式的 ADC 数据的指针

pPwmData： “HAL_AdcData_t”类型格式的 PWM 数据的指针

CTRL_runTraj () *void CTRL_runTraj(CTRL_Handle handle)*

运行轨迹

句柄： 控制器 (CTRL) 句柄

CTRL_runOffLine () *inline void CTRL_runOffLine(CTRL_Handle handle,HAL_Handle halHandle, const HAL_AdcData_t *pAdcData,HAL_PwmData_t *pPwmData)*

运行离线控制器

句柄： 控制器 (CTRL) 句柄

halHandle： 硬件抽象层 (HAL) 句柄

pAdcData： ADC 数据的指针

pPwmData： PWM 数据的指针

CTRL_runOnLine () *inline void CTRL_runOnLine(CTRL_Handle handle,const HAL_AdcData_t *pAdcData,HAL_PwmData_t *pPwmData)*

运行在线控制器

句柄： 控制器 (CTRL) 句柄

pAdcData： ADC 数据的指针

pPwmData： PWM 数据的指针

CTRL_runOnLine_User () *inline void CTRL_runOnLine_User(CTRL_Handle handle, const HAL_AdcData_t *pAdcData,HAL_PwmData_t *pPwmData)*

运行在线控制器

句柄： 控制器 (CTRL) 句柄

pAdcData： ADC 数据的指针

pPwmData： PWM 数据的指针

CTRL_useZeroIq_ref () *inline bool CTRL_useZeroIq_ref(CTRL_Handle handle)*

确定是否应在控制器中使用零 Iq 电流参考

句柄： 控制器 (CTRL) 句柄

返回： 指示是否应使用零 Iq 电流参考的布尔值 (true/false)

3.3.2 估算器 API 函数 - FAST 库 - est.h、est_states.h

3.3.2.1 EST 枚举和结构

EST_RsOnLineFilterType_e

Rs 在线滤波器类型的枚举

```
typedef enum
{
  EST_RsOnLineFilterType_Current=0,        < Current Filter
  EST_RsOnLineFilterType_Voltage           < Voltage Filter
} EST_RsOnLineFilterType_e;
```

EST_ErrorCode_e

估算器错误代码的枚举

```
typedef enum
{
  EST_ErrorCode_NoError=0,               < no error error code
  EST_ErrorCode_Flux_OL_ShiftOverFlow,   < flux open loop shift overflow error
code
  EST_ErrorCode_FluxError,               < flux estimator error code
  EST_ErrorCode_Dir_ShiftOverFlow,       < direction shift overflow error code
  EST_ErrorCode_Ind_ShiftOverFlow,       < inductance shift overflow error code
  EST_numErrorCodes                      < the number of estimator error codes
} EST_ErrorCode_e;
```

EST_State_e

估算器状态的枚举

```
typedef enum
{
  EST_State_Error=0,           < error
  EST_State_Idle,              < idle
  EST_State_RoverL,            < R/L estimation
  EST_State_Rs,                < Rs estimation state
  EST_State_RampUp,            < ramp up the speed
  EST_State_IdRated,           < control Id and estimate the rated flux
  EST_State_RatedFlux_OL,      < estimate the open loop rated flux
  EST_State_RatedFlux,         < estimate the rated flux
  EST_State_RampDown,          < ramp down the speed
  EST_State_LockRotor,         < lock the rotor
  EST_State_Ls,                < stator inductance estimation state
  EST_State_Rr,                < rotor resistance estimation state
  EST_State_MotorIdentified,   < motor identified state
  EST_State_OnLine,            < online parameter estimation
  EST_numStates                < the number of estimator states
} EST_State_e;
```

3.3.2.2 EST Set 函数

EST_setRsOnLineId_pu ()

extern void EST_setRsOnLineId_pu(EST_Handle handle,const _iq Id_pu);

获取用于在线定子电阻估算的 Id 值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： Id 值，pu

EST_setAngle_pu ()

extern void EST_setAngle_pu(EST_Handle handle,const _iq angle_pu);

设置估算器中的角度值，标么值 (pu)，IQ24

此函数会用用户提供的角度覆盖估算角度。 设置的值应介于 0x00000000 到 0x00FFFFFF 或 _IQ(0.0) 到 _IQ(1.0) 之间。 以下示例说明了如何覆盖估算角度：

```
_iq Overwrite_Flux_Angle_pu = _IQ(0.5);
EST_setAngle_pu(handle, Overwrite_Flux_Angle_pu);
```

一般用途不建议使用此函数，因为此函数会自动在转子磁通轴与驱动电机的控制信号之间产生轴失准。 建议对执行需要绕过估算器的开环启动算法感兴趣的高级用户使用此函数。

句柄： 估算器 (EST) 句柄

angle_pu： 角度值，pu

EST_setDcBus_pu ()

extern void EST_setDcBus_pu(EST_Handle handle,const _iq dcBus_pu);

设置估算器中的直流总线电压，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

dcBus_pu： 直流总线电压，pu

EST_setDir_qFmt ()

extern void EST_setDir_qFmt(EST_Handle handle,const uint_least8_t dir_qFmt);

设置估算器中的方向 Q 格式

句柄： 估算器 (EST) 句柄

dir_qFmt： 方向 Q 格式

EST_setFe_neg_max_pu ()

extern void EST_setFe_neg_max_pu(EST_Handle handle,const _iq fe_neg_max_pu);

设置估算器中的最大负电频率

句柄： 估算器 (EST) 句柄

fe_neg_max_pu： 最大负电频率，Hz

EST_setFe_pos_min_pu ()

extern void EST_setFe_pos_min_pu(EST_Handle handle,const _iq fe_pos_min_pu);

设置估算器中的最小正电频率

句柄： 估算器 (EST) 句柄

fe_pos_min_pu： 最小正电频率，Hz

EST_setFlag_enableFluxControl ()

extern void EST_setFlag_enableFluxControl(EST_Handle handle,const bool state);

设置估算器中的使能磁通控制标志

句柄： 估算器 (EST) 句柄

状态： 所需的标志状态，开 (1) 或关 (0)

EST_setFlag_enableForceAngle ()

void EST_setFlag_enableForceAngle (EST_Handle handle,const bool_t state)

设置估算器中的使能强制角标志

启用或禁用电机启动时的 Rs 直流测量

句柄： 估算器 (EST) 句柄

状态： 所需的标志状态，开 (1) 或关 (0)

- TRUE：启用强制角。 如果磁通频率低于以下定义中的阈值，则绕过估算角度：

```
#define USER_ZEROSPEEDLIMIT (0.001)
```

  该定义在 user.h 中。 此频率的典型值为以下定义中的满量程频率乘以 0.001：

```
#define USER_IQ_FULL_SCALE_FREQ_Hz (500.0)
```

  强制角算法有效时，也就是转子磁通电频率低于阈值时，将强制以下面定义设置的频率旋转一定角度：

```
#define USER_FORCE_ANGLE_FREQ_Hz  (1.0)
```

- FALSE：禁用强制角。 估算器不会被任何强制角算法绕过。

EST_setFlag_enableRsOnLine ()

void EST_setFlag_enableRsOnLine(EST_Handle handle,const bool_t state)

启用或禁用估算器句柄中的 Rs 在线估算

句柄： 估算器 (EST) 句柄

状态： 所需的标志状态，开 (1) 或关 (0)

EST_setFlag_enableRsRecalc ()

void EST_setFlag_enableRsRecalc(EST_Handle handle,const bool_t state)

设置估算器中的使能定子电阻 (Rs) 重新计算标志

启用或禁用电机启动时的 Rs 直流测量

句柄： 估算器 (EST) 句柄

状态： 所需的标志状态，开 (1) 或关 (0)

EST_setFlag_estComplete () *extern void EST_setFlag_estComplete(EST_Handle handle,const bool state);*

设置估算器中的估算完成标志

句柄： 估算器 (EST) 句柄

状态： 所需的标志状态，true (1) 或 false (0)

EST_setFlag_updateRs ()

void EST_setFlag_updateRs(EST_Handle handle,const bool_t state)

设置估算器中的更新定子电阻 (Rs) 标志。 将 Rs 在线估算器中的 Rs 值复制为 InstaSpin 角度估算器使用的 Rs 值

句柄： 估算器 (EST) 句柄

状态： 所需的标志状态，开 (1) 或关 (0)

EST_setForceAngleDelta_pu ()

extern void EST_setForceAngleDelta_pu(EST_Handle handle,const _iq angleDelta_pu);

设置估算器中的强制角增量值，标么值 (pu)，IQ24

此函数设置一个强制角增量，表示强制角要加上或减去的增量。 该值越大，强制角时生成的频率越高（在强制角模式下将绕过估算角度）。 默认情况下，强制角频率在 user.h 中设置。 以下示例说明了如何将强制角频率的单位从赫兹 (Hz) 设为标么值：

```
 #define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_EST_TICK  (1)
 #define USER_PWM_FREQ_kHz         (15.0)
 #define USER_ISR_FREQ_Hz          (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_EST_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_EST_TICK)
 #define USER_FORCE_ANGLE_FREQ_Hz  (1.0)

 _iq delta_hz_to_pu_sf = _IQ(1.0/(float_t)USER_EST_FREQ_Hz);
 _iq Force_Angle_Freq_Hz = _IQ(USER_FORCE_ANGLE_FREQ_Hz);
 _iq Force_Angle_Delta_pu = _IQmpy(Force_Angle_Freq_Hz, delta_hz_to_pu_sf);

 EST_setForceAngleDelta_pu(handle, Force_Angle_Delta_pu);
```

句柄： 估算器 (EST) 句柄

angleDelta_pu： 强制角增量值，pu

EST_setFreqB0_lp_pu ()

extern void EST_setFreqB0_lp_pu(EST_Handle handle,const _iq b0_lp_pu);

设置频率估算器中的低通滤波器分子值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

b0_lp_pu： 低通滤波器分子值，pu

EST_setFreqBeta_lp_pu ()

extern void EST_setFreqBeta_lp_pu(EST_Handle handle,const _iq beta_lp_pu);

设置用于设置频率估算器中的低通极点位置的值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

beta_lp_pu： 用于设置滤波器极点位置的值，pu

EST_setFullScaleCurrent ()

extern void EST_setFullScaleCurrent(EST_Handle handle,const float_t fullScaleCurrent);

设置估算器中的满量程电流，安培 (A)

句柄： 估算器 (EST) 句柄

fullScaleCurrent： 满量程电流，A

EST_setFullScaleFlux ()

extern void EST_setFullScaleFlux(EST_Handle handle,const float_t fullScaleFlux);

设置估算器中使用的满量程磁通值，伏*秒 (V.s)

句柄： 估算器 (EST) 句柄

fullScaleFlux： 满量程磁通值，V*s

EST_setFullScaleFreq ()

extern void EST_setFullScaleFreq(EST_Handle handle,const float_t fullScaleFreq);

设置估算器中的满量程频率，赫兹 (Hz)

句柄： 估算器 (EST) 句柄

fullScaleFreq： 满量程频率，Hz

EST_setFullScaleInductance ()

extern void EST_setFullScaleInductance(EST_Handle handle,const float_t fullScaleInductance);

设置估算器中的满量程电感，亨利 (H)

句柄： 估算器 (EST) 句柄

fullScaleInductance： 满量程电感，亨利

EST_setFullScaleResistance () *extern void EST_setFullScaleResistance(EST_Handle handle,const float_t fullScaleResistance);*

设置估算器中的满量程电阻，欧姆

句柄： 估算器 (EST) 句柄

fullScaleResistance： 满量程电阻，欧姆

EST_setFullScaleVoltage ()

extern void EST_setFullScaleVoltage(EST_Handle handle,const float_t fullScaleVoltage);

设置估算器中的满量程电压，伏特 (V)

句柄： 估算器 (EST) 句柄

fullScaleVoltage： 满量程电压，V

EST_setIdle () *extern void EST_setIdle(EST_Handle handle);*

将估算器设置为空闲

句柄： 估算器 (EST) 句柄

EST_setIdle_all () *extern void EST_setIdle_all(EST_Handle handle);*

将估算器及所有从属估算器设置为空闲

句柄： 估算器 (EST) 句柄

EST_setId_ref_pu () *extern void EST_setId_ref_pu(EST_Handle handle,const _iq Id_ref_pu);*

设置估算器中的直流电流 (Id) 参考值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

Id_ref_pu： Id 参考值，pu

EST_setIdRated_pu ()

extern void EST_setIdRated_pu(EST_Handle handle,const _iq IdRated_pu);

设置估算器中的 Id 额定电流值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

IdRated_pu： Id 额定电流值，pu

EST_setIq_ref_pu () *extern void EST_setIq_ref_pu(EST_Handle handle,const _iq Iq_ref_pu);*

设置估算器中的正交电流 (Iq) 参考值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

Iq_ref_pu： Iq 参考值，pu

EST_setLs_d_pu () *extern void EST_setLs_d_pu(EST_Handle handle,const _iq Ls_d_pu);*

设置估算器中的直流定子电感值，标么值 (pu)，IQ30

调用此函数可实时更改估算器使用的内部直流电感 (Ls_d)。 下面显示了此操作的一个示例：

```
 #define USER_MOTOR_Ls_d  (0.012)

 float_t fullScaleInductance = EST_getFullScaleInductance(handle);
 float_t Ls_coarse_max = _IQ30toF(EST_getLs_coarse_max_pu(handle));
 int_least8_t lShift =
ceil(log(USER_MOTOR_Ls_d/(Ls_coarse_max*fullScaleInductance))/log(2.0));
 uint_least8_t Ls_qFmt = 30 - lShift;
 float_t L_max = fullScaleInductance * pow(2.0,lShift);
 _iq Ls_d_pu = _IQ30(USER_MOTOR_Ls_d / L_max);

 EST_setLs_d_pu(handle, Ls_d_pu);
 EST_setLs_qFmt(handle, Ls_qFmt);
```

句柄： 估算器 (EST) 句柄

Ls_d_pu： 直流定子电感值，pu

EST_setLs_delta_pu () *extern void EST_setLs_delta_pu(EST_Handle handle,const _iq Ls_delta_pu);*

设置精确估算过程中的定子电感增量值

句柄： 估算器 (EST) 句柄

Ls_delta_pu： 定子电感增量值，pu

EST_setLs_dq_pu () *extern void EST_setLs_dq_pu(EST_Handle handle,const MATH_vec2 *pLs_dq_pu);*

设置估算器中的直流/正交定子电感矢量值，标么值 (pu)，IQ30

调用此函数可实时更改估算器使用的内部直流和正交电感 (Ls_d 和 Ls_q)。 下面显示了此操作的一个示例：

```
 #define USER_MOTOR_Ls_d  (0.012)
 #define USER_MOTOR_Ls_q  (0.027)

 float_t fullScaleInductance = EST_getFullScaleInductance(handle);
 float_t Ls_coarse_max = _IQ30toF(EST_getLs_coarse_max_pu(handle));
 int_least8_t lShift =
ceil(log(USER_MOTOR_Ls_d/(Ls_coarse_max*fullScaleInductance))/log(2.0));
 uint_least8_t Ls_qFmt = 30 - lShift;
 float_t L_max = fullScaleInductance * pow(2.0,lShift);
 MATH_vec2 Ls_dq_pu;
```

```
Ls_dq_pu.value[0] = _IQ30(USER_MOTOR_Ls_d / L_max);
Ls_dq_pu.value[1] = _IQ30(USER_MOTOR_Ls_q / L_max);

EST_setLs_dq_pu(handle, &Ls_dq_pu);
EST_setLs_qFmt(handle, Ls_qFmt);
```

句柄： 估算器 (EST) 句柄

pLs_dq_pu： 直流/正交定子电感矢量值的指针，pu

EST_setLs_q_pu () *extern void EST_setLs_q_pu(EST_Handle handle,const _iq Ls_q_pu);*

设置估算器中的正交定子电感值，标么值 (pu)，IQ30

调用此函数可实时更改估算器使用的内部正交电感 (Ls_q)。 下面显示了此操作的一个示例：

```
#define USER_MOTOR_Ls_q  (0.027)

float_t fullScaleInductance = EST_getFullScaleInductance(handle);
float_t Ls_coarse_max = _IQ30toF(EST_getLs_coarse_max_pu(handle));
int_least8_t lShift =
ceil(log(USER_MOTOR_Ls_q/(Ls_coarse_max*fullScaleInductance))/log(2.0));
uint_least8_t Ls_qFmt = 30 - lShift;
float_t L_max = fullScaleInductance * pow(2.0,lShift);
_iq Ls_d_pu = _IQ30(USER_MOTOR_Ls_q / L_max);

EST_setLs_q_pu(handle, Ls_q_pu);
EST_setLs_qFmt(handle, Ls_qFmt);
```

句柄： 估算器 (EST) 句柄

Ls_q_pu： 正交定子电感值，pu

EST_setLs_qFmt () *extern void EST_setLs_qFmt(EST_Handle handle,const uint_least8_t Ls_qFmt);*

设置估算器中的定子电感 Q 格式，8 位无符号整数 (uint_least8_t)

更新内部电感还需要更新 Q 格式变量，该变量用于扩展覆盖范围。 此 qFmt（Q 格式）变量会使用定点数学运算生成浮点。 需要注意的是，通过调用 EST_setLs_qFmt() 设定的电感 Q 格式将被标么电感计算值 Ls_d 和 Ls_q 同时使用。 下面显示了如何设置该 Q 格式的示例：

```
#define USER_MOTOR_Ls_d  (0.012)
#define USER_MOTOR_Ls_q  (0.027)

float_t fullScaleInductance = EST_getFullScaleInductance(handle);
float_t Ls_coarse_max = _IQ30toF(EST_getLs_coarse_max_pu(handle));
int_least8_t lShift =
ceil(log(USER_MOTOR_Ls_d/(Ls_coarse_max*fullScaleInductance))/log(2.0));
uint_least8_t Ls_qFmt = 30 - lShift;
float_t L_max = fullScaleInductance * pow(2.0,lShift);
MATH_vec2 Ls_dq_pu;

Ls_dq_pu.value[0] = _IQ30(USER_MOTOR_Ls_d / L_max);
Ls_dq_pu.value[1] = _IQ30(USER_MOTOR_Ls_q / L_max);

EST_setLs_dq_pu(handle, &Ls_dq_pu);
EST_setLs_qFmt(handle, Ls_qFmt);
```

句柄： 估算器 (EST) 句柄

Ls_qFmt: 定子电感 Q 格式

EST_setMaxAccel_pu ()

extern void EST_setMaxAccel_pu(EST_Handle handle,const _iq maxAccel_pu);

设置估算器中的最大加速度值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

maxAccel_pu: 最大加速度值，pu

EST_setMaxAccel_est_pu ()

extern void EST_setMaxAccel_est_pu(EST_Handle handle,const _iq maxAccel_pu);

设置估算器中的最大估算加速度值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

maxAccel_pu: 最大估算加速度值，pu

EST_setMaxCurrentSlope_pu ()

void EST_setMaxCurrentSlope_pu (EST_Handle handle, const _iq maxCurrentSlope_pu)

确定电机是否已被识别

句柄： 估算器 (EST) 句柄

maxCurrentSlope_pu: 最大电流斜率值，pu

EST_setMaxCurrentSlope_PowerWarp_pu ()

extern void EST_setMaxCurrentSlope_PowerWarp_pu(EST_Handle handle,const _iq maxCurrentSlope_pu);

设置估算器中使用的最大 PowerWarp 电流斜率值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

maxCurrentSlope_pu: 最大电流斜率值，pu

EST_setRr_pu ()

extern void EST_setRr_pu(EST_Handle handle,const _iq Rr_pu);

设置估算器中的转子电阻值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

Rr_pu: 转子电阻值，pu

EST_setRr_qFmt ()

extern void EST_setRr_qFmt(EST_Handle handle,uint_least8_t Rr_qFmt);

设置估算器中的转子电阻 Q 格式，8 位无符号整数 (uint_least8_t)

句柄： 估算器 (EST) 句柄

Rr_qFmt： 转子电阻 Q 格式

EST_setRs_delta_pu ()

extern void EST_setRs_delta_pu(EST_Handle handle,const _iq Rs_delta_pu);

设置定子电阻增量值

句柄： 估算器 (EST) 句柄

Rs_delta_pu： 定子电阻增量值，pu

EST_setRsOnLine_pu ()

extern void EST_setRsOnLine_pu(EST_Handle handle,const _iq Rs_pu);

设置在线定子电阻估算器中的定子电阻值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

Rs_pu： 定子电阻值，pu

EST_setRsOnLine_qFmt ()

extern void EST_setRsOnLine_qFmt(EST_Handle handle,const uint_least8_t Rs_qFmt);

设置在线定子电阻估算器中的定子电阻 Q 格式，8 位无符号整数 (uint_least8_t)

句柄： 估算器 (EST) 句柄

Rs_qFmt： 定子电阻 Q 格式

EST_setRsOnLineFilterParams ()

extern void EST_setRsOnLineFilterParams(EST_Handle handle,const EST_RsOnLineFilterType_e filterType, const _iq filter_0_b0,const _iq filter_0_a1,const _iq filter_0_y1, const _iq filter_1_b0,const _iq filter_1_a1,const _iq filter_1_y1);

设置在线定子电阻滤波器参数，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

filterType： 滤波器类型

filter_0_b0： z^0 的滤波器 0 分子系数值

filter_0_a1： z^(-1) 的滤波器 0 分母系数值

filter_0_y1： 时间采样 n=-1 时的滤波器 0 输出值

filter_1_b0： z^0 的滤波器 1 分子系数值

filter_1_a1： z^(-1) 的滤波器 1 分母系数值

filter_1_y1: 时间采样 n=-1 时的滤波器 1 输出值

EST_setRsOnLineId_mag_pu ()

extern void EST_setRsOnLineId_mag_pu(EST_Handle handle,const _iq Id_mag_pu);

设置用于在线定子电阻估算的 Id 幅值，标么值 (pu)，IQ24

句柄: 估算器 (EST) 句柄

Id_mag_pu: Id 幅值，pu

EST_setRs_pu ()

extern void EST_setRs_pu(EST_Handle handle,const _iq Rs_pu);

设置估算器中使用的定子电阻值，标么值 (pu)，IQ30

句柄: 估算器 (EST) 句柄

Rs_pu: 定子电阻值，pu

EST_setRs_qFmt ()

extern void EST_setRs_qFmt(EST_Handle handle,uint_least8_t Rs_qFmt);

设置估算器中的定子电阻 Q 格式，8 位无符号整数 (uint_least8_t)

句柄: 估算器 (EST) 句柄

Rs_qFmt: 定子电阻 Q 格式

EST_updateId_ref_pu ()

extern void EST_updateId_ref_pu(EST_Handle handle,_iq *pId_ref_pu);

更新用于在线定子电阻估算的 Id 参考值，标么值 (pu)，IQ24

句柄: 估算器 (EST) 句柄

pId_ref_pu: Id 参考值的指针，pu

3.3.2.3 EST Get 函数

EST_get_krpm_to_pu_sf ()

extern _iq EST_get_krpm_to_pu_sf(EST_Handle handle);

获取 krpm 到 pu 的换算系数，标么值 (pu)，IQ24 当用户需要将电机速度值从 krpm（每分钟千转数）值换算为标么值时，需要使用此函数。

此换算系数的计算和使用如下所示：

```
 #define USER_MOTOR_NUM_POLE_PAIRS       (2)
#define USER_IQ_FULL_SCALE_FREQ_Hz      (500.0)

 _iq scale_factor = _IQ(USER_MOTOR_NUM_POLE_PAIRS * 1000.0 / (60.0 *
USER_IQ_FULL_SCALE_FREQ_Hz));

 _iq Speed_krpm = EST_getSpeed_krpm(handle);
 _iq Speed_krpm_to_pu_sf = EST_get_krpm_to_pu_sf(handle);
 _iq Speed_pu = _IQmpy(Speed_krpm,Speed_krpm_to_pu_sf);
```

句柄： 估算器 (EST) 句柄

返回： krpm 到 pu 的换算系数。 该值为 IQ24 格式

EST_get_pu_to_krpm_sf ()

extern _iq EST_get_pu_to_krpm_sf(EST_Handle handle);

获取 pu 到 krpm 的换算系数，标么值 (pu)，IQ24 当用户需要将电机速度值从标么值换算为 krpm（每分钟千转数）值时，需要使用此函数。

此换算系数的计算和使用如下所示：

```
 #define USER_MOTOR_NUM_POLE_PAIRS       (2)
#define USER_IQ_FULL_SCALE_FREQ_Hz      (500.0)

 _iq scale_factor = IQ(60.0 * USER_IQ_FULL_SCALE_FREQ_Hz /
(USER_MOTOR_NUM_POLE_PAIRS * 1000.0));

 _iq Speed_pu = EST_getFm_pu(handle);
 _iq Speed_pu_to_krpm_sf = EST_get_pu_to_krpm_sf(handle);
 _iq Speed_krpm = _IQmpy(Speed_krpm,Speed_krpm_to_pu_sf);
```

句柄： 估算器 (EST) 句柄

返回： krpm 到 pu 的换算系数。 该值为 IQ24 格式

EST_getAngle_pu () *_iq EST_getAngle_pu(EST_Handle handle)*

获取估算器中的角度值，标么值 (pu)，IQ24 此函数返回转子磁通角的标么值。 该值在 1.0 处计满循环，因此返回值介于 0x00000000 到 0x00FFFFFF 或 _IQ(0.0) 到 _IQ(1.0) 之间。 使用此角度的示例如下所示：

```
_iq Rotor_Flux_Angle_pu = EST_getAngle_pu(handle);
```

句柄： 估算器 (EST) 句柄

返回： 磁通角度值，pu

EST_getDcBus_pu () *_iq EST_getDcBus_pu(EST_Handle handle)*

获取估算器中的直流总线值，标么值 (pu)，IQ24 调用函数 EST_run() 时，该值最初作为一个参数传递。 一个类似的函数可以简单地读取 ADC 转换器通过 pAdcData->dcBus 读取和换算的数据。 库在内部使用该值来计算直流总线值的倒数，以补偿电流控制器的比例增益。 以下示例说明了如何使用此函数计算直流总线值（以千伏为单位）：

```
#define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)

 _iq Vbus_pu = EST_getDcBus_pu(handle);
 _iq Vbus_pu_to_kV_sf = _IQ(USER_IQ_FULL_SCALE_VOLT
_iq Vbus_kV = _IQmpy(Vbus_pu,Vbus_pu_to_kV_sf);
```

句柄： 估算器 (EST) 句柄

返回： 直流总线值，pu

EST_ErrorCode_e EST_getErrorCode ()

extern EST_ErrorCode_e EST_getErrorCode(EST_Handle handle);

获取控制器的错误代码

句柄： 估算器 (EST) 句柄

返回： 错误代码

EST_getFe ()

extern int32_t EST_getFe(EST_Handle handle);

获取电机的电频率，赫兹 (Hz)。 此频率 (Hz) 是进入电机的电流和电压的频率。 要获取电机的速度，最好使用 EST_getFm()。

句柄： 估算器 (EST) 句柄

返回： 电频率，Hz

EST_getFe_pu ()

extern _iq EST_getFe_pu(EST_Handle handle);

获取电机的电频率，标么值 (pu)，IQ24 与 EST_getFe() 函数类似，该函数返回电机的电频率（标么值）。 要将电频率从标么值转换为 Hz，用户需要将返回值乘以以下换算系数：

```
_iq Full_Scale_Freq_Elec_Hz = _IQ(USER_IQ_FULL_SCALE_FREQ_Hz);
_iq Freq_Elec_Hz = _IQmpy(EST_getFe_pu(handle),Full_Scale_Freq_Elec_Hz);
```

句柄： 估算器 (EST) 句柄

返回： 电频率，pu

EST_getFlag_enableForceAngle ()

extern bool EST_getFlag_enableForceAngle(EST_Handle handle);

获取估算器的使能强制角标志值。

句柄： 估算器 (EST) 句柄

返回： 标志值，布尔型，bool

- TRUE： 启用强制角，如果磁通频率低于以下定义中的阈值，则将绕过估算角度：

```
#define USER_ZEROSPEEDLIMIT (0.001)
```

此频率的典型值为以下定义中的满量程频率乘以 0.001：

```
#define USER_IQ_FULL_SCALE_FREQ_Hz (500.0)
```

强制角算法有效时，也就是转子磁通电频率低于阈值时，将强制以下面定义设置的频率旋转一定角度：

```
#define USER_FORCE_ANGLE_FREQ_Hz  (1.0)
```

- FALSE：禁用强制角。 估算器不会被任何强制角算法绕过。

EST_getFlag_enableRsOnLine ()

extern bool EST_getFlag_enableRsOnLine(EST_Handle handle);

获取使能在线定子电阻 (Rs) 估算的标志值

句柄： 估算器 (EST) 句柄

返回： 使能在线 Rs 标志值

- true - 使能 Rs 在线再校准算法。 估算器将运行一组与 Rs 在线再校准算法相关的函数，该算法会在电机旋转时重新计算定子电阻。 当电机升温并且定子电阻因此增大时，此算法非常有用。
- false - 禁用 Rs 在线再校准算法，即使电机升温也不对 Rs 进行更新。 如果定子电阻由于电机升温而发生变化，可能会影响低速性能和满转矩时的启动性能。 定子电阻将固定，并等于 EST_getRs_Ohm() 返回的值。

EST_getFlag_enableRsRecalc ()

extern bool EST_getFlag_enableRsRecalc(EST_Handle handle);

获取使能在线定子电阻 (Rs) 估算的标志值

句柄： 估算器 (EST) 句柄

返回： 使能在线 Rs 标志值

- true - 使能 Rs 在线再校准算法。 估算器将运行一组与 Rs 在线再校准算法相关的函数，该算法会在电机旋转时重新计算定子电阻。 当电机升温并且定子电阻因此增大时，此算法非常有用。
- false - 禁用 Rs 在线再校准算法，即使电机升温也不对 Rs 进行更新。 如果定子电阻由于电机升温而发生变化，可能会影响低速性能和满转矩时的启动性能。 定子电阻将固定，并等于 EST_getRs_Ohm() 返回的值。

EST_getFlag_estComplete ()

extern bool EST_getFlag_estComplete(EST_Handle handle);

获取指示估算完成的标志值。 每次运行 EST_run() 函数时，此标志都会设置为 true。 可通过以下示例将此标志重置为 false：

```
bool estComplete_Flag = EST_getFlag_estComplete(handle);
```

句柄： 估算器 (EST) 句柄

返回： 使能在线 Rs 标志值

- true - 自上次调用 EST_setFlag_estComplete(handle, false) 以来，估算器已运行至少一次。
- false - 自上次调用 EST_setFlag_estComplete(handle, false) 以来，估算器尚未运行。

EST_getFlag_updateRs ()

extern bool EST_getFlag_updateRs(EST_Handle handle);

获取使能更新定子电阻 (Rs) 值的标志值。 启用在线电阻估算器时，更新标志允许将在线电阻复制为估算器模型使用的电阻。 如果更新标志未设置为 true，估算器模型将不会使用在线电阻估算，而且如果由于温度升高使电阻变化过大，模型可能无法按预期工作。

```
bool update_Flag = EST_getFlag_updateRs(handle);
```

句柄： 估算器 (EST) 句柄

返回： 更新 Rs 标志值

- true - Rs 在线模块估算的定子电阻将复制为模块使用的定子电阻，因此当电机的温度发生变化时，将根据最新的定子电阻计算估算角度。
- false - Rs 在线模块估算的定子电阻会根据使能标志更新，但不会在用于生成估算速度和角度的电机模型中使用。

EST_getFlux_VpHz ()

int32_t EST_getFlux_VpHz(EST_Handle handle)

获取磁通值 (V/Hz)

估算器会不断计算转子与定子之间的磁链，它是磁通中产生转矩的一部分。 此函数返回转子和定子线圈之间的磁链，并忽略匝数，单位为伏/赫兹 (V/Hz)。 仅当识别电机后，此函数才会返回精确值，这可通过以下代码示例检查：

```
if(EST_isMotorIdentified(handle))
  {
    // once the motor has been identified, get the flux
    float_t Flux_VpHz = EST_getFlux_VpHz(handle);
  }
```

句柄： 估算器 (EST) 句柄

返回： 磁通值，V/Hz

EST_getFlux_Wb ()

int32_t EST_getFlux_Wb(EST_Handle handle)

获取磁通值（韦伯）

估算器会不断计算转子与定子之间的磁链，它是磁通中产生转矩的一部分。 此函数返回转子和定子线圈之间的磁链，并忽略匝数，单位为韦伯 (Wb) 或伏*秒 (V.s)。 仅当识别电机后，此函数才会返回精确值，这可通过以下代码示例检查：

```
if(EST_isMotorIdentified(handle))
  {
    // once the motor has been identified, get the flux
    float_t Flux_Wb = EST_getFlux_Wb(handle);
  }
```

句柄： 估算器 (EST) 句柄

返回： 磁通值，韦伯

EST_getFlux_pu () *extern _iq EST_getFlux_pu(EST_Handle handle);*

获取磁通值，标么值 (pu)，IQ24

估算器会不断计算转子与定子之间的磁链，它是磁通中产生转矩的一部分。 此函数返回转子和定子线圈之间的磁链，并忽略匝数，标么值。 仅当识别电机后，此函数才会返回精确值，这可通过以下代码示例检查：

```
if(EST_isMotorIdentified(handle))
  {
    // once the motor has been identified, get the flux
    _iq Flux_pu = EST_getFlux_pu(handle);
  }
```

对于一些应用，获取此标么值非常重要，因为处理该值的速度要快得多，特别是微控制器的架构没有浮点处理单元时。 为了将此标么值转换为 _iq 换算值，必须考虑一个换算系数将此标么磁通转换为所需单位。 以下示例说明了如何将标么值转换为 IQ 格式的 Wb 和 V/Hz 以进行更快的处理：

```
float_t FullScaleFlux = (USER_IQ_FULL_SCALE_VOLTAGE_V/(float_t)USER_EST_FREQ_Hz);
 float_t maxFlux =
(USER_MOTOR_RATED_FLUX*((USER_MOTOR_TYPE==MOTOR_Type_Induction)?0.05:0.7));
 float_t lShift = -ceil(log(FullScaleFlux/maxFlux)/log(2.0));
 _iq gFlux_pu_to_Wb_sf = _IQ(FullScaleFlux/(2.0*MATH_PI)*pow(2.0,lShift));
 _iq gFlux_pu_to_VpHz_sf = _IQ(FullScaleFlux*pow(2.0,lShift));
 // The value of gFlux_pu_to_Wb_sf and gFlux_pu_to_VpHz_sf can be calculated once
at the beginning of the
 // code and stored as global variables

 _iq Flux_Wb;
 _iq Flux_VpHz;
 _iq Flux_pu = EST_getFlux_pu(handle);

 Flux_Wb = _IQmpy(Flux_pu, gFlux_pu_to_Wb_sf);
 Flux_VpHz = _IQmpy(Flux_pu, gFlux_pu_to_VpHz_sf);
```

句柄： 估算器 (EST) 句柄

返回： 磁通值，pu

EST_getFm () *extern int32_t EST_getFm(EST_Handle handle);*

获取电机的机械频率，赫兹 (Hz)。 此频率 (Hz) 是电机的机械频率。 如果电机是永磁电机，机械频率将等于电频率，因为是同步电机。 如果电机是交流感应电机，机械频率将等于电频率减去转差频率。 以下代码示例说明了如何使用此函数计算每分钟转数 (RPM)（浮点）：

```
#define USER_MOTOR_NUM_POLE_PAIRS  (2)

 float_t Mechanical_Freq_Hz = EST_getFm(handle);
 float_t hz_to_rpm_sf = 60.0/USER_MOTOR_NUM_POLE_PAIRS;
 float_t Speed_RPM = Mechanical_Freq_Hz * hz_to_rpm_sf;
```

句柄： 估算器 (EST) 句柄

返回： 机械频率，Hz

EST_getFm_pu () *extern _iq EST_getFm_pu(EST_Handle handle);*

获取电机的机械频率，标么值 (pu)，IQ24 与 EST_getFe_pu() 函数类似，该函数返回电机的机械频率（标么值）。 要将机械频率从标么值转换为 KHz（以避免 IQ24 饱和），用户需要将返回值乘以以下换算系数：

```
#define USER_IQ_FULL_SCALE_FREQ_Hz  (500.0)

 _iq pu_to_khz_sf = _IQ(USER_IQ_FULL_SCALE_FREQ_Hz/1000.0);
 _iq khz_to_krpm_sf = _IQ(60.0/USER_MOTOR_NUM_POLE_PAIRS);
 _iq Mechanical_Freq_kHz = _IQmpy(EST_getFm_pu(handle),pu_to_khz_sf);
 _iq Speed_kRPM = _IQmpy(Mechanical_Freq_kHz,khz_to_krpm_sf);
```

句柄： 估算器 (EST) 句柄

返回： 机械频率，pu

EST_getForceAngleDelta_pu () *extern _iq EST_getForceAngleDelta_pu(EST_Handle handle);*

获取估算器中的强制角增量值，标么值 (pu)，IQ24 仅当通过调用 CTRL_setParams() 函数初始化控制器对象后，此函数才返回有效值。 强制角增量表示强制角要加上或减去的增量。 该值越大，强制角时生成的频率越高（在强制角模式下将绕过估算角度）。 默认情况下，强制角频率在 user.h 中设置。 以下示例说明了如何将增量的单位从标么值转换为千赫兹 (kHz)。

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_EST_TICK  (1)
 #define USER_PWM_FREQ_kHz         (15.0)
 #define USER_ISR_FREQ_Hz          (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_EST_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_EST_TICK)

 _iq delta_pu_to_kHz_sf = _IQ((float_t)USER_EST_FREQ_Hz/1000.0);
 _iq Force_Angle_Delta_pu = EST_getForceAngleDelta_pu(handle);
 _iq Force_Angle_Freq_kHz = _IQmpy(Force_Angle_Delta_pu, delta_pu_to_kHz_sf);
```

注意首选 kHz，以避免 IQ24 变量溢出。

句柄： 估算器 (EST) 句柄

返回： 强制角增量，pu 最小值 _IQ(0.0) 和最大值 _IQ(1.0)。

EST_getForceAngleStatus () *extern bool EST_getForceAngleStatus(EST_Handle handle);*

获取估算器中的强制角操作的状态。 当通过调用以下函数启用强制角模式后，状态才能变为活动：EST_setFlag_enableForceAngle(handle, true)；当电机的电频率低于 user.h 中 #define USER_ZEROSPEEDLIMIT (0.001) 定义的阈值时，强制角模式将处于活动状态。可使用以下代码示例手动检查强制角状态：

```
_iq fe_pu = EST_getFe_pu(handle);
 bool is_forced_angle_active;
 if(_IQabs(fe_pu) < _IQ(USER_ZEROSPEEDLIMIT))
   {
     is_forced_angle_active = true;
```

```
  }
else
  {
    is_forced_angle_active = false;
  }
```

注意首选 kHz，以避免 IQ24 变量溢出。

句柄： 估算器 (EST) 句柄

返回： 指示是否已强制角的布尔值 (true/false)

- true - 估算器的上次迭代使用强制角来运行帕克变换和帕克逆变换。 估算器也与强制角并行运行，但未使用估算器输出。

\retval

- false - 强制角模式已被禁用，或电频率未低于预先定义的阈值。 估算器输出用于运行帕克变换和帕克逆变换。

EST_getFreqB0_lp_pu ()

extern _iq EST_getFreqB0_lp_pu(EST_Handle handle);

获取频率估算器中的低通滤波器分子值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

返回： 低通滤波器分子值，pu

EST_getFreqBeta_lp_pu ()

extern _iq EST_getFreqBeta_lp_pu(EST_Handle handle);

获取用于设置频率估算器的低通滤波器中的极点位置的值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

返回： 用于设置滤波器极点位置的值，pu

EST_getFslip ()

extern int32_t EST_getFslip(EST_Handle handle);

获取电机的转差频率，赫兹 (Hz)。

句柄： 估算器 (EST) 句柄

返回： 转差频率，Hz

句柄： 估算器 (EST) 句柄

返回： krpm 到 pu 的换算系数。 该值为 IQ24 格式

EST_getFslip_pu ()

extern _iq EST_getFslip_pu(EST_Handle handle);

获取电机的转差频率，标么值 (pu)，IQ24

与 EST_getFe_pu() 函数类似，该函数返回电机的转差频率（标么值）。 要将转差频率从标么值转换为 Hz，用户需要将返回值乘以以下换算系数：

```
#define USER_IQ_FULL_SCALE_FREQ_Hz (500.0)

 _iq Full_Scale_Freq_Elec_Hz = _IQ(USER_IQ_FULL_SCALE_FREQ_Hz);

 _iq Freq_Slip_Hz = _IQmpy(EST_getFslip_pu(handle),Full_Scale_Freq_Elec_Hz);
```

句柄： 估算器 (EST) 句柄

返回： 转差频率，pu

EST_getFullScaleCurrent ()

extern int32_t EST_getFullScaleCurrent(EST_Handle handle);

获取估算器中使用的满量程电流值，安培 (A)

此函数返回的值与 user.h 中定义的值相同。 当用户需要以真实单位（即，安培）显示值时，使用该值将电流标么值转换为安培值。 以下示例说明了执行该转换的两种不同方法，一种使用浮点，另一种使用 IQ 数学运算。 使用浮点的示例：

```
float_t pu_to_amps_sf = EST_getFullScaleCurrent(handle);
 _iq Id_rated_pu = EST_getIdRated_pu(handle);
 float_t Id_rated_A = _IQtoF(Id_rated_pu) * pu_to_amps_sf;

    Example using fixed point:

 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)

 _iq pu_to_amps_sf = _IQ(USER_IQ_FULL_SCALE_CURRENT_A);
 _iq Id_rated_pu = EST_getIdRated_pu(handle);
 _iq Id_rated_A = _IQmpy(Id_rated_pu, pu_to_amps_sf);
```

句柄： 估算器 (EST) 句柄

返回： 满量程电流值，A

EST_getFullScaleFlux ()

extern int32_t EST_getFullScaleFlux(EST_Handle handle);

获取估算器中使用的满量程磁通值，伏/赫兹 (V/Hz)

句柄： 估算器 (EST) 句柄

返回： 满量程磁通值

EST_getFullScaleFreq ()

extern int32_t EST_getFullScaleFreq(EST_Handle handle);

获取估算器中使用的满量程频率值，赫兹 (Hz)

满量程频率可用作将标么值转换为赫兹的换算系数。 以下代码示例说明了如何使用此函数通过浮点数学运算将频率从标么值转换为 Hz：

```
float_t Mechanical_Frequency_pu = _IQtoF(EST_getFm_pu(handle));
 float_t pu_to_hz_sf = EST_getFullScaleFreq(handle);
 float_t Mechanical_Frequency_hz = Mechanical_Frequency_pu * pu_to_hz_sf
```

为了使执行更快，可使用 user.h 中的满量程频率定义来避免调用此函数。 以下示例显示了相同功能，但使用了定点数学运算以加快执行：

```
#define USER_IQ_FULL_SCALE_FREQ_Hz (500.0)

 _iq Mechanical_Frequency_pu = EST_getFm_pu(handle);
 _iq pu_to_khz_sf = _IQ(USER_IQ_FULL_SCALE_FREQ_Hz/1000.0);
 _iq Mechanical_Frequency_khz = _IQmpy(Mechanical_Frequency_pu, pu_to_khz_sf);
```

句柄： 估算器 (EST) 句柄

返回： 满量程频率值，Hz

EST_getFullScaleInductance ()

extern int32_t EST_getFullScaleInductance(EST_Handle handle);

获取估算器中使用的满量程电感值，亨利 (H)。

获取估算器使用的电感有不同的方法。 将电感从标么值转换为 H 时，此函数很有用。但返回值为浮点格式，因此使用该满量程值将标么值转换为 H 并不是最高效的方式。 下面提供了两个示例，说明了为加快执行而进行的浮点标么值到 H 的转换以及定点标么值到 H 的转换。 浮点示例：

```
uint_least8_t Ls_qFmt = EST_getLs_qFmt(handle);
 float_t fullScaleInductance = EST_getFullScaleInductance(handle);
 float_t Ls_d_pu = _IQ30toF(EST_getLs_d_pu(handle));
 float_t pu_to_h_sf = fullScaleInductance * pow(2.0, 30 - Ls_qFmt);
 float_t Ls_d_H = Ls_d_pu * pu_to_h_sf;
```

另一个示例的目的是避免使用浮点数学运算以加快执行。 在此示例中，使用预编译器数学运算根据 user.h 中的用户参数计算满量程电感值：

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-
(nshift))) : ((var) <<(nshift)))

 #define MATH_PI (3.14159265358979323846264338327795)
 #define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)
 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
 #define USER_VOLTAGE_FILTER_POLE_Hz  (335.648)
 #define USER_VOLTAGE_FILTER_POLE_rps (2.0 * MATH_PI *
USER_VOLTAGE_FILTER_POLE_Hz)

 uint_least8_t Ls_qFmt = EST_getLs_qFmt(handle);
 _iq fullScaleInductance =
_IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/(USER_IQ_FULL_SCALE_CURRENT_A *
USER_VOLTAGE_FILTER_POLE_rps));
 _iq Ls_d_pu = _IQ30toIQ(EST_getLs_d_pu(handle));
 _iq pu_to_h_sf = VarShift(fullScaleInductance, 30 - Ls_qFmt);
 _iq Ls_d_H = _IQmpy(Ls_d_pu, pu_to_h_sf);
```

句柄： 估算器 (EST) 句柄

返回： 满量程电感值，亨利

EST_getFullScaleResistance ()

extern int32_t EST_getFullScaleResistance(EST_Handle handle);

获取估算器中使用的满量程电阻值（欧姆）。获取估算器使用的电阻有不同的方法。 将电阻从标么值转换为欧姆时，此函数很有用。 但返回值为浮点格式，因此使用该满量程值将标么值转换为欧姆并不是最高效的方式。 下面提供了两个示例，说明了为加快执行而进行

的浮点标么值到欧姆的转换以及定点标么值到欧姆的转换。 浮点示例：

```
uint_least8_t Rs_qFmt = EST_getRs_qFmt(handle);
 float_t fullScaleResistance = EST_getFullScaleResistance(handle);
 float_t Rs_pu = _IQ30toF(EST_getRs_pu(handle));
 float_t pu_to_ohms_sf = fullScaleResistance * pow(2.0, 30 - Rs_qFmt);
 float_t Rs_Ohms = Rs_pu * pu_to_ohms_sf;
```

另一个示例的目的是避免使用浮点数学运算以加快执行。 在以下示例中，使用预编译器数学运算根据 user.h 中的用户参数计算满量程电阻值：

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-
(nshift))) : ((var)<<(nshift)))

 #define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)
 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)

 uint_least8_t Rs_qFmt = EST_getRs_qFmt(handle);
 _iq fullScaleResistance =
_IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/USER_IQ_FULL_SCALE_CURRENT_A);
 _iq Rs_pu = _IQ30toIQ(EST_getRs_pu(handle));
 _iq pu_to_ohms_sf = VarShift(fullScaleResistance, 30 - Rs_qFmt);
 _iq Rs_Ohms = _IQmpy(Rs_pu, pu_to_ohms_sf);
```

句柄： 估算器 (EST) 句柄

返回： 满量程电阻值，欧姆

EST_getFullScaleVoltage ()

extern int32_t EST_getFullScaleVoltage(EST_Handle handle);

获取估算器中使用的满量程电压值，伏 (V)。

此函数返回的值与 user.h 中定义的值相同。 当用户需要以真实单位（即，伏）显示值时，使用该值将电压标么值转换为伏值。 以下示例说明了执行该转换的两种不同方法，一种使用浮点，另一种使用 IQ 数学运算。

使用浮点的示例：

```
float_t pu_to_v_sf = EST_getFullScaleVoltage(handle);
 _iq DcBus_pu = EST_getDcBus_pu(handle);
 float_t DcBus_V = _IQtoF(DcBus_pu) * pu_to_v_sf;
```

使用定点的示例：

```
#define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)

 _iq pu_to_kv_sf = _IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/1000.0);
 _iq DcBus_pu = EST_getDcBus_pu(handle);
 _iq DcBus_kV = _IQmpy(DcBus_pu, pu_to_kv_sf);
```

句柄： 估算器 (EST) 句柄

返回： 满量程电阻值，欧姆

EST_getIdRated () *float_t EST_getIdRated(EST_Handle handle)*

获取估算器的 Id 额定电流值

句柄： 估算器 (EST) 句柄

返回： Id 额定电流值，A

EST_getIdRated_pu ()

extern _iq EST_getIdRated_pu(EST_Handle handle);

获取估算器中的 Id 额定电流值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： Id 额定电流值，pu

EST_getIdRated_indEst_pu ()

_iq EST_getIdRated_indEst_pu(EST_Handle handle)

获取用于感应估算的 Id 额定电流值

句柄： 估算器 (EST) 句柄

返回： Id 额定值，pu

EST_getIdRated_ratedFlux_pu ()

extern _iq EST_getIdRated_ratedFlux_pu(EST_Handle handle);

获取用于感应电机磁通估算的 Id 电流值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： Id 额定值，pu

EST_getLr_H ()

extern int32_t EST_getLr_H(EST_Handle handle);

获取转子电感值，亨利 (H)。

句柄： 估算器 (EST) 句柄

返回： Id 额定值，pu

EST_getLr_pu ()

extern _iq EST_getLr_pu(EST_Handle handle);

获取转子电感值，标么值 (pu)，IQ30

还可以使用转子电感的标么值通过定点数学运算来计算感应电机的转子电感。 下面显示了此操作的一个示例：

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-
(nshift))) : ((var)<<(nshift)))

 #define MATH_PI (3.14159265358979323846264338327795)
 #define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)
 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
 #define USER_VOLTAGE_FILTER_POLE_Hz  (335.648)
 #define USER_VOLTAGE_FILTER_POLE_rps (2.0 * MATH_PI *
USER_VOLTAGE_FILTER_POLE_Hz)

 uint_least8_t Lr_qFmt = EST_getLr_qFmt(handle);
 _iq fullScaleInductance =
_IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/(USER_IQ_FULL_SCALE_CURRENT_A *
USER_VOLTAGE_FILTER_POLE_rps));
```

```
 _iq Lr_pu = _IQ30toIQ(EST_getLr_pu(handle));
 _iq pu_to_h_sf = VarShift(fullScaleInductance, 30 - Lr_qFmt);
 _iq Lr_H = _IQmpy(Lr_pu, pu_to_h_sf);
```

句柄： 估算器 (EST) 句柄

返回： 转子电感值，pu

EST_getLr_qFmt () *extern uint_least8_t EST_getLr_qFmt(EST_Handle handle);*

获取转子电感 Q 格式，8 位无符号整数 (uint_least8_t)。

估算器识别电机后，各种识别的参数均使用 Q 格式。 该 Q 格式是用于识别的实际 Q 格式与识别电机参数期间内部使用的 IQ30 的区别。 要了解如何在用户代码中使用此 Q 格式，请参见以下将读取自估算器的标么值转换为亨利的示例：

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-
(nshift))) : ((var)<<(nshift)))

 #define MATH_PI (3.14159265358979323846264338327795)
 #define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)
 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
 #define USER_VOLTAGE_FILTER_POLE_Hz  (335.648)
 #define USER_VOLTAGE_FILTER_POLE_rps (2.0 * MATH_PI *
USER_VOLTAGE_FILTER_POLE_Hz)

 uint_least8_t Lr_qFmt = EST_getLr_qFmt(handle);
 _iq fullScaleInductance =
_IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/(USER_IQ_FULL_SCALE_CURRENT_A *
USER_VOLTAGE_FILTER_POLE_rps));
 _iq Lr_pu = _IQ30toIQ(EST_getLr_pu(handle));
 _iq pu_to_h_sf = VarShift(fullScaleInductance, 30 - Lr_qFmt);
 _iq Lr_H = _IQmpy(Lr_pu, pu_to_h_sf);
```

句柄： 估算器 (EST) 句柄

返回： 转子电感值 Q 格式

EST_getLs_d_H () *float_t EST_getLs_d_H(EST_Handle handle)*

获取直流定子电感值（亨利）

句柄： 估算器 (EST) 句柄

返回： 直流定子电感值，亨利

EST_getLs_d_pu () *extern _iq EST_getLs_d_pu(EST_Handle handle);*

获取直流定子电感值，标么值 (pu)，IQ30

还可以使用直流定子电感的标么值通过定点数学运算来计算永磁电机的直流定子电感。 下面显示了此操作的一个示例：

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-
(nshift))) : ((var)<<(nshift)))

 #define MATH_PI (3.14159265358979323846264338327795)
 #define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)
 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
 #define USER_VOLTAGE_FILTER_POLE_Hz  (335.648)
```

```
 #define USER_VOLTAGE_FILTER_POLE_rps (2.0 * MATH_PI *
USER_VOLTAGE_FILTER_POLE_Hz)

 uint_least8_t Ls_qFmt = EST_getLs_qFmt(handle);
 _iq fullScaleInductance =
_IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/(USER_IQ_FULL_SCALE_CURRENT_A *
USER_VOLTAGE_FILTER_POLE_rps));
 _iq Ls_d_pu = _IQ30toIQ(EST_getLs_d_pu(handle));
 _iq pu_to_h_sf = VarShift(fullScaleInductance, 30 - Ls_qFmt);
 _iq Ls_d_H = _IQmpy(Ls_d_pu, pu_to_h_sf);
```

句柄： 估算器 (EST) 句柄

返回： 直流定子电感值，pu

EST_getLs_delta_pu ()

extern _iq EST_getLs_delta_pu(EST_Handle handle);

获取定子电感估算器中的定子电感增量值

句柄： 估算器 (EST) 句柄

返回： 定子电感增量值，pu

EST_getLs_dq_pu () *extern void EST_getLs_dq_pu(EST_Handle handle,MATH_vec2 *pLs_dq_pu);*

获取估算器中的直流/正交定子电感矢量值，标么值 (pu)，IQ30。通过使用此函数调用并向存储直流和正交定子电感值的结构传递一个指针，可以读取估算器中的这两个值。

句柄： 估算器 (EST) 句柄

pLs_dq_pu： 直流/正交定子电感矢量值的指针，pu

EST_getLs_q_H () *float_t EST_getLs_q_H(EST_Handle handle)*

获取正交坐标方向上的定子电感值（亨利）

句柄： 估算器 (EST) 句柄

返回： 定子电感值，亨利

EST_getLs_q_pu () *extern _iq EST_getLs_q_pu(EST_Handle handle);*

获取正交坐标方向上的定子电感值，标么值 (pu)，IQ30

还可以使用正交定子电感的标么值通过定点数学运算来计算永磁电机的正交定子电感。 下面显示了此操作的一个示例：

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-
(nshift))) : ((var)<<(nshift)))

 #define MATH_PI (3.14159265358979323846264338327795)
 #define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)
 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
 #define USER_VOLTAGE_FILTER_POLE_Hz  (335.648)
 #define USER_VOLTAGE_FILTER_POLE_rps (2.0 * MATH_PI *
USER_VOLTAGE_FILTER_POLE_Hz)
```

```
 uint_least8_t Ls_qFmt = EST_getLs_qFmt(handle);
 _iq fullScaleInductance =
_IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/(USER_IQ_FULL_SCALE_CURRENT_A *
USER_VOLTAGE_FILTER_POLE_rps));
 _iq Ls_q_pu = _IQ30toIQ(EST_getLs_q_pu(handle));
 _iq pu_to_h_sf = VarShift(fullScaleInductance, 30 - Ls_qFmt);
 _iq Ls_q_H = _IQmpy(Ls_q_pu, pu_to_h_sf);
```

句柄： 估算器 (EST) 句柄

返回： 定子电感值，pu

EST_getLs_qFmt () *extern uint_least8_t EST_getLs_qFmt(EST_Handle handle);*

获取定子电感 Q 格式，8 位无符号整数 (uint_least8_t)。

估算器识别电机后，各种识别的参数均使用 Q 格式。 该 Q 格式是用于识别的实际 Q 格式与识别电机参数期间内部使用的 IQ30 的区别。 要了解如何在用户代码中使用此 Q 格式，请参见以下将读取自估算器的标么值转换为亨利的示例：

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-
(nshift))) : ((var)<<(nshift)))

 #define MATH_PI (3.14159265358979323846264338327795)
 #define USER_IQ_FULL_SCALE_VOLTAGE_V (300.0)
 #define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
 #define USER_VOLTAGE_FILTER_POLE_Hz  (335.648)
 #define USER_VOLTAGE_FILTER_POLE_rps (2.0 * MATH_PI *
USER_VOLTAGE_FILTER_POLE_Hz)

 uint_least8_t Ls_qFmt = EST_getLs_qFmt(handle);
 _iq fullScaleInductance =
_IQ(USER_IQ_FULL_SCALE_VOLTAGE_V/(USER_IQ_FULL_SCALE_CURRENT_A *
USER_VOLTAGE_FILTER_POLE_rps));
 _iq Ls_q_pu = _IQ30toIQ(EST_getLs_q_pu(handle));
 _iq pu_to_h_sf = VarShift(fullScaleInductance, 30 - Ls_qFmt);
 _iq Ls_q_H = _IQmpy(Ls_q_pu, pu_to_h_sf);
```

句柄： 估算器 (EST) 句柄

返回： 定子电感 Q 格式

EST_getLs_max_pu ()
extern _iq EST_getLs_max_pu(EST_Handle handle);

获取定子电感估算器中的最大定子电感值

句柄： 估算器 (EST) 句柄

返回： 最大定子电感值，pu

EST_getLs_min_pu ()
extern _iq EST_getLs_min_pu(EST_Handle handle);

获取定子电感估算器中的最小定子电感值

句柄： 估算器 (EST) 句柄

返回： 最小定子电感值，pu

EST_getLs_coarse_max_pu ()

extern _iq EST_getLs_coarse_max_pu(EST_Handle handle);

获取粗略估算期间定子电感估算器中的最大定子电感值

句柄： 估算器 (EST) 句柄

返回： 最大定子电感值，pu

EST_getMaxAccel_pu ()

extern _iq EST_getMaxAccel_pu(EST_Handle handle);

获取估算器中使用的最大加速度值，标么值 (pu)，IQ24

最大加速度是轨迹模块的一项设置，用于设置速度基准。 识别电机后，使用此函数调用返回的加速度。 该值表示如何从初始值增大或减小速度基准以达到目标值。 以下示例说明了如何将此函数的返回值转换为千赫兹/秒 (kHz/s) 和千 RPM/秒 (kRPM/s)：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_FREQ_Hz        (500.0)
 #define USER_MOTOR_NUM_POLE_PAIRS         (4)

 _iq pu_to_khzps_sf = _IQ((float_t)USER_TRAJ_FREQ_Hz * USER_IQ_FULL_SCALE_FREQ_Hz
/ 1000.0);
 _iq khzps_to_krpmps_sf = _IQ(60.0 / (float_t)USER_MOTOR_NUM_POLE_PAIRS);

 _iq Accel_pu = EST_getMaxAccel_pu(handle);
 _iq Accel_kilo_hz_per_sec = _IQmpy(Accel_pu, pu_to_khzps_sf);
 _iq Accel_kilo_rpm_per_sec = _IQmpy(Accel_kilo_hz_per_sec, khzps_to_krpmps_sf);
```

默认值由 user.h 中的用户定义值设置，默认标么值的内部计算方式如下：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_FREQ_Hz        (500.0)
 #define USER_MAX_ACCEL_Hzps               (20.0)

 _iq hzps_to_pu_sf = _IQ(1.0 / ((float_t)USER_TRAJ_FREQ_Hz *
USER_IQ_FULL_SCALE_FREQ_Hz));

 _iq Accel_hertz_per_sec = _IQ(USER_MAX_ACCEL_Hzps);
 _iq Accel_pu = _IQmpy(Accel_hertz_per_sec, hzps_to_pu_sf);
```

句柄： 估算器 (EST) 句柄

返回： 最大加速度值，pu

EST_getMaxAccel_est_pu ()

extern _iq EST_getMaxAccel_est_pu(EST_Handle handle);

获取估算器中使用的最大估算加速度值，标么值 (pu)，IQ24

最大加速度是轨迹模块的一项设置，用于设置速度基准。 电机识别过程中使用此函数调用返回的加速度。 该值表示如何从初始值增大或减小速度基准以达到目标值。 以下示例说明了如何将此函数的返回值转换为千赫兹/秒 (kHz/s) 和千 RPM/秒 (kRPM/s)：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_FREQ_Hz        (500.0)
 #define USER_MOTOR_NUM_POLE_PAIRS         (4)

 _iq pu_to_khzps_sf = _IQ((float_t)USER_TRAJ_FREQ_Hz * USER_IQ_FULL_SCALE_FREQ_Hz
/ 1000.0);
 _iq khzps_to_krpmps_sf = _IQ(60.0 / (float_t)USER_MOTOR_NUM_POLE_PAIRS);

 _iq est_Accel_pu = EST_getMaxAccel_est_pu(handle);
 _iq est_Accel_kilo_hz_per_sec = _IQmpy(est_Accel_pu, pu_to_khzps_sf);
 _iq est_Accel_kilo_rpm_per_sec = _IQmpy(est_Accel_kilo_hz_per_sec,
khzps_to_krpmps_sf);
```

默认值由 user.h 中的用户定义值设置，默认标么值的内部计算方式如下：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_FREQ_Hz        (500.0)
 #define USER_MAX_ACCEL_EST_Hzps           (2.0)

 _iq hzps_to_pu_sf = _IQ(1.0 / ((float_t)USER_TRAJ_FREQ_Hz *
USER_IQ_FULL_SCALE_FREQ_Hz));

 _iq est_Accel_hertz_per_sec = _IQ(USER_MAX_ACCEL_EST_Hzps);
 _iq est_Accel_pu = _IQmpy(est_Accel_hertz_per_sec, hzps_to_pu_sf);
```

句柄： 估算器 (EST) 句柄

返回： 最大估算加速度值，pu

EST_getMaxCurrentSlope_pu ()

extern _iq EST_getMaxCurrentSlope_pu(EST_Handle handle);

获取估算器中使用的最大电流斜率值，标么值 (pu)，IQ24

获取 Id 参考的斜率。 以下示例说明了如何将返回值转换为千安培/秒 (kA/s)：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
```

```
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_CURRENT_A      (10.0)

 _iq pu_to_kA_per_sec_sf = _IQ((float_t)USER_TRAJ_FREQ_Hz *
USER_IQ_FULL_SCALE_CURRENT_A / 1000.0);

 _iq currentSlope_pu = EST_getMaxCurrentSlope_pu(handle);
 _iq currentSlope_kAps = _IQmpy(currentSlope_pu, pu_to_kA_per_sec_sf);
```

默认值由 user.h 中的用户定义值设置，默认标么值的内部计算方式如下：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_CURRENT_A      (10.0)
 #define USER_MOTOR_RES_EST_CURRENT        (1.0)

 _iq A_per_sec_to_pu_sf = _IQ(1.0 / ((float_t)USER_TRAJ_FREQ_Hz *
USER_IQ_FULL_SCALE_CURRENT_A));

 _iq currentSlope_Aps = _IQ(USER_MOTOR_RES_EST_CURRENT);
 _iq currentSlope_pu = _IQmpy(currentSlope_Aps, A_per_sec_to_pu_sf);
```

句柄： 估算器 (EST) 句柄

返回： 最大电流斜率值，pu

EST_getMaxCurrentSlope_PowerWarp_pu ()

extern _iq EST_getMaxCurrentSlope_PowerWarp_pu(EST_Handle handle);

获取估算器中使用的最大 PowerWarp 电流斜率值，标么值 (pu)，IQ24

获取启用有效部分负载时的 Id 参考变化斜率。 此模式仅适用于感应电机。 以下示例说明了如何将返回值转换为千安培/秒 (kA/s)：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_CURRENT_A      (10.0)

 _iq pu_to_kA_per_sec_sf = _IQ((float_t)USER_TRAJ_FREQ_Hz *
USER_IQ_FULL_SCALE_CURRENT_A / 1000.0);

 _iq currentSlope_PowerWarp_pu = EST_getMaxCurrentSlope_PowerWarp_pu(handle);
 _iq currentSlope_PowerWarp_kAps = _IQmpy(currentSlope_PowerWarp_pu,
pu_to_kA_per_sec_sf);
```

默认值由 user.h 中的用户定义值设置，默认标么值的内部计算方式如下：

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK  (1)
 #define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK (10)
 #define USER_PWM_FREQ_kHz                 (15.0)
```

```
 #define USER_ISR_FREQ_Hz                  (USER_PWM_FREQ_kHz * 1000.0)
 #define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)
 #define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
 #define USER_IQ_FULL_SCALE_CURRENT_A      (10.0)
 #define USER_MOTOR_RES_EST_CURRENT        (1.0)

 _iq A_per_sec_to_pu_sf = _IQ(1.0 / ((float_t)USER_TRAJ_FREQ_Hz *
USER_IQ_FULL_SCALE_CURRENT_A));

 _iq currentSlope_PowerWarp_Aps = _IQ(0.3 * USER_MOTOR_RES_EST_CURRENT);
 _iq currentSlope_PowerWarp_pu = _IQmpy(currentSlope_PowerWarp_Aps,
A_per_sec_to_pu_sf);
```

句柄： 估算器 (EST) 句柄

返回： 最大 PowerWarp 电流斜率值，pu

EST_getOneOverDcBus_pu ()

extern _iq EST_getOneOverDcBus_pu(EST_Handle handle);

获取直流总线电压的反向电压，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： 直流总线电压的反向电压，pu

EST_getRr_Ohm ()

extern int32_t EST_getRr_Ohm(EST_Handle handle);

获取转子电阻值（欧姆）

句柄： 估算器 (EST) 句柄

返回： 转子电阻值，欧姆

EST_getRr_pu ()

extern _iq EST_getRr_pu(EST_Handle handle);

获取转子电阻值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

返回： 转子电阻值，pu

EST_getRr_qFmt ()

extern uint_least8_t EST_getRr_qFmt(EST_Handle handle);

获取转子电阻 Q 格式，8 位无符号整数 (uint_least8_t)。

句柄： 估算器 (EST) 句柄

返回： 转子电阻 Q 格式

EST_getRs_delta_pu ()

extern _iq EST_getRs_delta_pu(EST_Handle handle);

获取定子电阻估算器中的定子电阻增量值

句柄： 估算器 (EST) 句柄

返回： 定子电阻增量值，pu

EST_getRs_Ohm () *float_t EST_getRs_Ohm(EST_Handle handle)*

获取角度估算器使用的定子电阻值

句柄： 估算器 (EST) 句柄

返回： 定子电阻值，欧姆

EST_getRs_pu () *extern _iq EST_getRs_pu(EST_Handle handle);*

获取定子电阻值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

返回： 定子电阻值，pu

EST_getRs_qFmt () *extern uint_least8_t EST_getRs_qFmt(EST_Handle handle);*

获取定子电阻 Q 格式，8 位无符号整数 (uint_least8_t)

句柄： 估算器 (EST) 句柄

返回： 定子电阻 Q 格式

EST_getRs_qFmt () *extern void EST_getRsOnLineFilterParams(EST_Handle handle,const EST_RsOnLineFilterType_e filterType, _iq *pFilter_0_b0,_iq *pFilter_0_a1,_iq *pFilter_0_y1, _iq *pFilter_1_b0,_iq *pFilter_1_a1,_iq *pFilter_1_y1);*

获取在线定子电阻滤波器参数，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

filterType： 滤波器类型

pFilter_0_b0： z^0 的滤波器 0 分子系数值的指针

pFilter_0_a1： z^(-1) 的滤波器 0 分母系数值的指针

pFilter_0_y1： 时间采样 n=-1 时的滤波器 0 输出值的指针

pFilter_1_b0： z^0 的滤波器 1 分子系数值的指针

pFilter_1_a1： z^(-1) 的滤波器 1 分母系数值的指针

pFilter_1_y1： 时间采样 n=-1 时的滤波器 1 输出值的指针

EST_getRsOnLine_Ohm ()

extern int32_t EST_getRsOnLine_Ohm(EST_Handle handle)

获取在线定子电阻值

句柄： 估算器 (EST) 句柄

返回： 在线定子电阻值，欧姆

EST_getRsOnLine_pu ()

extern _iq EST_getRsOnLine_pu(EST_Handle handle);

获取在线定子电阻值，标么值 (pu)，IQ30

句柄： 估算器 (EST) 句柄

返回： 在线定子电阻 Q 格式

EST_getRsOnLineId_mag_pu ()

extern _iq EST_getRsOnLineId_mag_pu(EST_Handle handle);

获取用于在线定子电阻估算的 Id 幅值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： Id 幅值，pu

EST_getRsOnLineId_pu ()

extern _iq EST_getRsOnLineId_pu(EST_Handle handle);

获取用于在线定子电阻估算的 Id 值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： Id 值，pu

EST_getSpeed_krpm ()

_iq EST_getSpeed_krpm(EST_Handle handle)

获取速度值 (krpm)

句柄： 估算器 (EST) 句柄

返回： 速度值，krpm

EST_getSignOfDirection ()

extern int_least8_t EST_getSignOfDirection(EST_Handle handle);

获取 8 位有符号整数方向值 (int_least8_t) 的符号

句柄： 估算器 (EST) 句柄

返回： 方向值的符号（-1 表示负，1 表示正）

EST_getSpeed_krpm ()

_iq EST_getSpeed_krpm(EST_Handle handle)

获取速度值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： 速度值，krpm

EST_getState () *EST_State_e EST_getState(EST_Handle handle)*

获取估算器的状态

句柄： 估算器 (EST) 句柄

返回： 估算器状态

EST_getTorque_lbin () *_iq EST_getTorque_lbin(EST_Handle handle)*

获取转矩值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： 转矩值，lb*in

EST_getTorque_Nm () *extern _iq EST_getTorque_Nm(EST_Handle handle);*

获取转矩值，标么值 (pu)，IQ24

句柄： 估算器 (EST) 句柄

返回： 转矩值，N*m

EST_getDir_qFmt () *extern uint_least8_t EST_getDir_qFmt(EST_Handle handle);*

获取估算器中的方向 Q 格式

句柄： 估算器 (EST) 句柄

返回： 方向 Q 格式

3.3.2.4 EST Run 和 Compute 函数

EST_computeLr_H ()

extern int32_t EST_computeLr_H(EST_Handle handle,const _iq current);

计算转子电感，亨利 (H)

句柄： 估算器 (EST) 句柄

电流: 转子电流

返回： 转子电感，H

EST_doCurrentCtrl ()

extern bool EST_doCurrentCtrl(EST_Handle handle);

确定是否应在电机识别期间执行电流控制

句柄： 估算器 (EST) 句柄

返回： 指示是否执行电流控制的布尔值 (true/false)

EST_genOutputLimits_Pid_Id ()

extern void EST_genOutputLimits_Pid_Id(EST_Handle handle, const _iq maxDutyCycle, _iq *outMin,_iq *outMax);

生成 PID Id 控制器输出限制

句柄： 估算器 (EST) 句柄

maxDutyCycle： 最大占空比，pu

outMin： 指向最小输出值的指针

outMax： 指向最大输出值的指针

EST_genOutputLimits_Pid_Iq ()

extern void EST_genOutputLimits_Pid_Iq(EST_Handle handle, const _iq maxDutyCycle, const _iq out_Id,_iq *outMin,_iq *outMax);

生成 PID Iq 控制器输出限制

句柄： 估算器 (EST) 句柄

maxDutyCycle： 最大占空比，pu

out_Id： Id 输出值

outMin： 指向最小输出值的指针

outMax： 指向最大输出值的指针

EST_run ()

extern void EST_run(EST_Handle handle, const MATH_vec2 *plab_pu, const MATH_vec2 *pVab_pu, const _iq dcBus_pu, const _iq speed_ref_pu);

运行估算器

句柄： 估算器 (EST) 句柄

plab_pu： 指向 alpha/beta 坐标系中的相电流的指针，pu

IQ24 pVab_pu： 指向 alpha/beta 坐标系中的相电压的指针，pu

IQ24 dcBus_pu： 直流总线电压，pu

IQ24 speed_ref_pu： 控制器的速度基准值，pu IQ24

EST_computeDirection_qFmt ()

extern uint_least8_t EST_computeDirection_qFmt(EST_Handle handle,const int32_t flux_max);

计算估算器的方向 Q 格式

句柄： 估算器 (EST) 句柄

flux_max： 最大磁通值

返回： 方向 Q 格式

3.3.2.5 EST 计数器函数

EST_resetCounter_ctrl ()

extern void EST_resetCounter_ctrl(EST_Handle handle);

重置控制计数器

句柄： 估算器 (EST) 句柄

EST_resetCounter_state ()

extern void EST_resetCounter_state(EST_Handle handle);

重置状态计数器

句柄： 估算器 (EST) 句柄

3.3.2.6 EST 状态控制和错误处理函数

EST_isError () ***extern bool EST_isError(EST_Handle handle);***

确定是否存在估算器错误

句柄： 估算器 (EST) 句柄

返回： 指示是否存在估算器错误的布尔值 (true/false)

EST_isIdle () ***extern bool EST_isIdle(EST_Handle handle);***

确定估算器是否空闲

句柄： 估算器 (EST) 句柄

返回： 指示估算器是否空闲的布尔值 (true/false)

EST_isLockRotor () ***extern bool EST_isLockRotor(EST_Handle handle);***

确定估算器是否正在等待转子锁定

句柄： 估算器 (EST) 句柄

返回： 指示估算器是否正在等待转子锁定的布尔值 (true/false)

EST_isMotorIdentified ()
EST_State_e EST_isMotorIdentified (EST_Handle handle)

确定电机是否已被识别

句柄： 估算器 (EST) 句柄

返回： 估算器状态

EST_isOnLine () ***extern bool EST_isOnLine(EST_Handle handle);***

确定估算器是否准备好在线控制

句柄： 估算器 (EST) 句柄

返回： 指示估算器是否准备好在线控制的布尔值 (true/false)

EST_updateState () ***extern bool EST_updateState(EST_Handle handle,const _iq Id_target_pu);***

更新估算器状态

句柄： 估算器 (EST) 句柄

Id_target_pu： 各个估算器状态期间的目标 Id 电流，pu，IQ24

返回： 指示状态是否变化的布尔值 (true/false)

EST_useZeroIq_ref ()

extern bool EST_useZeroIq_ref(EST_Handle handle);

确定是否应在控制器中使用零 Iq 电流参考

句柄： 估算器 (EST) 句柄

返回： 指示是否应使用零 Iq 电流参考的布尔值 (true/false)

3.3.3 硬件抽象层 (HAL) API 函数 - hal.c、hal.h、hal_obj.h

HAL_Obj 是包含器件外设句柄的结构。 HAL_init() 为 HAL 对象分配内存，内存分配完毕后，HAL_setParams() 根据对象 USER_Params 中的用户设置来配置各个外设。

3.3.3.1 HAL 枚举和结构

HAL_AdcData_t

定义 ADC 数据。该数据结构包含执行 HAL_AdcRead 时使用的电压和电流值，该结构随后将传递给 CTRL 控制器和 FAST 估算器。

```
typedef struct _HAL_AdcData_t_
{
  MATH_vec3 I;          //!< the current values

  MATH_vec3 V;          //!< the voltage values

  _iq       dcBus;      //!< the dcBus value

} HAL_AdcData_t;
```

HAL_DacData_t

定义 DAC 数据。该数据结构包含用于许多硬件套件上的 DAC 输出的 pwm 值，以供调试之用。

```
typedef struct _HAL_DacData_t_
{
  _iq  value[4];      //!< the DAC data

} HAL_DacData_t;
```

HAL_PwmData_t

定义 PWM 数据。该结构包含三相的 pwm 电压值。 HAL_PwmData_t 变量将填充空间矢量调制器之类的值，然后发送到类似 HAL_writePwmData() 的函数以写入 PWM 外设。

```
typedef struct _HAL_PwmData_t_
{
  MATH_vec3  Tabc;      //!< the PWM time-durations for each motor phase

} HAL_PwmData_t;
```

HAL_LedNumber_e

用于定义 ControlCARD 上的 LED 的枚举

```
typedef enum
{
  HAL_Gpio_LED2=GPIO_Number_31,  //!< GPIO pin number for ControlCARD LED 2
  HAL_Gpio_LED3=GPIO_Number_34   //!< GPIO pin number for ControlCARD LED 3
} HAL_LedNumber_e;
```

GPIO_Number_e

用于定义通用 I/O (GPIO) 编号的枚举

```
typedef enum
{
  GPIO_Number_0=0,    //!< Denotes GPIO number 0
```

```
  GPIO_Number_1,      //!< Denotes GPIO number 1
  GPIO_Number_2,      //!< Denotes GPIO number 2
...
  GPIO_Number_57,     //!< Denotes GPIO number 57
  GPIO_Number_58,     //!< Denotes GPIO number 58
  GPIO_numGpios
} GPIO_Number_e;
```

HAL_SensorType_e

```
typedef enum
{
  HAL_SensorType_Current=0,      //!< Enumeration for current sensor
  HAL_SensorType_Voltage         //!< Enumeration for voltage sensor
} HAL_SensorType_e;
```

HAL_Obj

HAL 对象包含所有外设句柄。 访问处理器上的外设时，将 HAL 函数与该处理器的 HAL 句柄一起使用才能访问其外设。

```
typedef struct _HAL_Obj_
{
  ADC_Handle    adcHandle;        //!< the ADC handle

  CLK_Handle    clkHandle;        //!< the clock handle

  CPU_Handle    cpuHandle;        //!< the CPU handle

  FLASH_Handle  flashHandle;      //!< the flash handle

  GPIO_Handle   gpioHandle;       //!< the GPIO handle

  OFFSET_Handle offsetHandle_I[3];  //!< the handles for the current offset
estimators
  OFFSET_Obj    offset_I[3];        //!< the current offset objects

  OFFSET_Handle offsetHandle_V[3];  //!< the handles for the voltage offset
estimators
  OFFSET_Obj    offset_V[3];        //!< the voltage offset objects

  OSC_Handle    oscHandle;        //!< the oscillator handle

  PIE_Handle    pieHandle;        //<! the PIE handle

  PLL_Handle    pllHandle;        //!< the PLL handle

  PWM_Handle    pwmHandle[3];     //<! the PWM handles

  PWMDAC_Handle pwmDacHandle[3];  //<! the PWMDAC handles

  PWR_Handle    pwrHandle;        //<! the power handle

  TIMER_Handle  timerHandle[3];   //<! the timer handles

  WDOG_Handle   wdogHandle;       //!< the watchdog handle

  HAL_AdcData_t adcBias;          //!< the ADC bias

  _iq           current_sf;       //!< the current scale factor, amps_pu/cnt

  _iq           voltage_sf;       //!< the voltage scale factor, volts_pu/cnt

  uint_least8_t numCurrentSensors; //!< the number of current sensors
```

```
  uint_least8_t numVoltageSensors; //!< the number of voltage sensors

  AFE_Handle    afeHandle;        //!< the AFE handle

#ifdef QEP
  QEP_Handle    qepHandle[1];      //!< the QEP handle
#endif

} HAL_Obj;
```

3.3.3.2 HAL – ADC 和 AFE

HAL_setupAdcs () ***void HAL_setupAdcs(HAL_Handle handle)***

设置 ADC（模数转换器）

句柄： 驱动程序 (HAL) 句柄

HAL_setupAfe () ***void HAL_setupAfe(HAL_Handle halHandle)***

设置 AFE（模拟前端）

句柄： 驱动程序 (HAL) 句柄

HAL_acqAdcInt () ***void HAL_acqAdcInt(HAL_Handle handle,const ADC_IntNumber_e intNumber)***

确认 ADC 的中断

句柄： 驱动程序 (HAL) 句柄

intNumber： 中断编号

HAL_readAdcData() ***void HAL_readAdcData(HAL_Handle handle,HAL_AdcData_t *pAdcData)***

将 ADC 数据读入 pAdcData 指向的值

读入 ADC 结果寄存器，调整偏移，并根据 user.h 中的设置换算值。 结构 gAdcData 保存三个相电压、三个线电流和一个直流总线电压。

句柄： 驱动程序 (HAL) 句柄

pAdcData： ADC 数据缓冲区的指针

HAL_updateAdcBias ()
static inline void HAL_updateAdcBias(HAL_Handle handle)

更新 ADC 偏置值。启动电机前调用此函数。 它会设置电压和电流测量偏移。

句柄： 驱动程序 (HAL) 句柄

HAL_setBias () ***void HAL_setBias (HAL_Handle handle,const HAL_SensorType_e sensorType,uint_least8_t sensorNumber,const _iq bias)***

设置 ADC 偏置值

句柄： 驱动程序 (HAL) 句柄

sensorType： 传感器类型

sensorNumber： 传感器编号

bias： ADC 偏置值

HAL_getBias ()

void HAL_getBias (HAL_Handle handle,const HAL_SensorType_e sensorType,uint_least8_t sensorNumber)

获取 ADC 偏置值

ADC 偏置包含反馈电路的偏移和偏置。 偏置是将双极信号读入单极 ADC 时使用的数学偏移。

句柄： 驱动程序 (HAL) 句柄

sensorType： 传感器类型

sensorNumber： 传感器编号

返回： ADC 偏置值

HAL_cal ()

extern void HAL_cal(HAL_Handle handle);

执行校准例程

偏移值和增益值由 TI 厂家编写到 OTP 存储器中。 此函数会调用将偏移值和增益值编写入 ADC 寄存器的内部函数。

句柄： 驱动程序 (HAL) 句柄

HAL_AdcCalConversion ()

uint16_t HAL_AdcCalConversion(HAL_Handle handle);

从所选校准通道读取转换后的值

句柄： 驱动程序 (HAL) 句柄

返回： 转换后的值

HAL_AdcOffsetSelfCal ()

void HAL_AdcOffsetSelfCal(HAL_Handle handle);

执行 ADC 的偏移校准

句柄： 驱动程序 (HAL) 句柄

HAL_getAdcSocSampleDelay ()

static inline ADC_SocSampleDelay_e HAL_getAdcSocSampleDelay(HAL_Handle handle, const ADC_SocNumber_e socNumber)

获取 ADC 延迟值

句柄： 驱动程序 (HAL) 句柄

socNumber： ADC SOC 编号

返回： ADC 延迟值

HAL_setAdcSocSampleDelay ()

static inline ADC_SocSampleDelay_e HAL_setAdcSocSampleDelay(HAL_Handle handle, const ADC_SocNumber_e socNumber)

设置 ADC 延迟值

句柄： 驱动程序 (HAL) 句柄

socNumber： ADC SOC 编号

sampleDelay： ADC 延迟值

HAL_getCurrentScaleFactor ()

static inline _iq HAL_getCurrentScaleFactor(HAL_Handle handle)

获取电流换算系数

电流换算系数定义为 USER_ADC_FULL_SCALE_CURRENT_A/USER_IQ_FULL_SCALE_CURRENT_A。 在 PU 电流和实际电流之间转换时，不使用此换算系数。

句柄： 驱动程序 (HAL) 句柄

返回： 电流换算系数

HAL_setCurrentScaleFactor () *static inline _iq HAL_setCurrentScaleFactor(HAL_Handle handle)*

设置电流换算系数

电流换算系数定义为 USER_ADC_FULL_SCALE_CURRENT_A/USER_IQ_FULL_SCALE_CURRENT_A。 在 PU 电流和实际电流之间转换时，不使用此换算系数。

句柄： 驱动程序 (HAL) 句柄

current_sf： 电流换算系数

HAL_getVoltageScaleFactor ()

static inline _iq HAL_getVoltageScaleFactor(HAL_Handle handle)

获取电压换算系数

电压换算系数定义为 USER_ADC_FULL_SCALE_VOLTAGE_V/USER_IQ_FULL_SCALE_VOLTAGE_V。 在 PU 电压和实际电压之间转换时，不使用此换算系数。

句柄： 驱动程序 (HAL) 句柄

返回： 电压换算系数

HAL_setVoltageScaleFactor ()

static inline _iq HAL_setVoltageScaleFactor(HAL_Handle handle)

设置电压换算系数

句柄： 驱动程序 (HAL) 句柄

voltage_sf： 电压换算系数

HAL_getNumCurrentSensors ()

static inline uint_least8_t HAL_getNumCurrentSensors(HAL_Handle handle)

获取电流传感器数

句柄： 驱动程序 (HAL) 句柄

返回： 电流传感器数

HAL_setNumCurrentSensors ()

static inline uint_least8_t HAL_setNumCurrentSensors(HAL_Handle handle)

设置电流传感器数

句柄： 驱动程序 (HAL) 句柄

返回： 电流传感器数

HAL_getNumVoltageSensors ()

static inline uint_least8_t HAL_getNumVoltageSensors(HAL_Handle handle)

获取电压传感器数

句柄： 驱动程序 (HAL) 句柄

返回： 电压传感器数

HAL_setNumVoltageSensors ()

static inline uint_least8_t HAL_setNumVoltageSensors(HAL_Handle handle)

设置电压传感器数

句柄： 驱动程序 (HAL) 句柄

numVoltageSensors： 电压传感器数

句柄： 驱动程序 (HAL) 句柄

返回：

HAL_getOffsetBeta_lp_pu ()

static inline _iq HAL_getOffsetBeta_lp_pu(HAL_Handle handle, const HAL_SensorType_e sensorType, const uint_least8_t sensorNumber)

获取用于设置偏移估算的低通滤波器极点的值

IIR 单极低通滤波器用于查找反馈电路的偏移。 此函数返回该极点的值。

句柄： 驱动程序 (HAL) 句柄

sensorType： 传感器类型

sensorNumber： 传感器编号

返回： 用于设置低通滤波器极点的值，pu

HAL_setOffsetBeta_lp_pu ()

static inline _iq HAL_setOffsetBeta_lp_pu(HAL_Handle handle, const HAL_SensorType_e sensorType, const uint_least8_t sensorNumber)

设置用于设置偏移估算的低通滤波器极点的值

IIR 单极低通滤波器用于查找反馈电路的偏移。 此函数返回该极点的值。

句柄： 驱动程序 (HAL) 句柄

sensorType： 传感器类型

sensorNumber： 传感器编号

beta_lp_pu： 用于设置低通滤波器极点的值，pu

HAL_setOffsetInitCond ()

static inline void HAL_setOffsetInitCond(HAL_Handle handle, const HAL_SensorType_e sensorType, const uint_least8_t sensorNumber, const _iq initCond)

设置偏移估算的偏移初始状态值

句柄： 驱动程序 (HAL) 句柄

sensorType： 传感器类型

sensorNumber： 传感器编号

initCond： 初始状态值

HAL_getOffsetValue ()

static inline _iq HAL_getOffsetValue(HAL_Handle handle, const HAL_SensorType_e sensorType, const uint_least8_t sensorNumber)

获取偏移值

在从 IIR 滤波器对象返回反馈电路校准期间计算出来的偏移。

句柄： 驱动程序 (HAL) 句柄

sensorType： 传感器类型

sensorNumber： 传感器编号

返回： 偏移值

HAL_setOffsetValue ()

static inline _iq HAL_setOffsetValue(HAL_Handle handle, const HAL_SensorType_e sensorType, const uint_least8_t sensorNumber)

设置偏移值

句柄： 驱动程序 (HAL) 句柄

sensorType： 传感器类型

sensorNumber： 传感器编号

值： 初始偏移值

HAL_runOffsetEst () *inline void HAL_runOffsetEst(HAL_Handle handle,const HAL_AdcData_t *pAdcData)*

运行偏移估算

句柄： 驱动程序 (HAL) 句柄

pAdcData： ADC 数据的指针

3.3.3.3 HAL – PWM 和 PWM-DAC

HAL_setupPwms () ***extern void HAL_setupPwms(HAL_Handle handle, const uint_least16_t systemFreq_MHz, const float_t pwmPeriod_usec, const uint_least16_t numPwmTicksPerIsrTick);***

设置 PWM（脉宽调制器）

句柄： 驱动程序 (HAL) 句柄

systemFreq_MHz： 系统频率，MHz

pwmPeriod_usec： PWM 周期，微秒

numPwmTicksPerIsrTick： 每个 ISR 时钟节拍的 PWM 时钟节拍数

HAL_setupPwmDacs ()

void HAL_setupPwmDacs(HAL_Handle handle)

设置 PWM DAC

句柄： 驱动程序 (HAL) 句柄

HAL_readTimerCnt ()

static inline uint32_t HAL_readTimerCnt(HAL_Handle handle,const uint_least8_t timerNumber)

关闭 EPWM 外设的输出，该外设会将电源开关置于高阻态

句柄： 驱动程序 (HAL) 句柄

HAL_reloadTimer () ***static inline void HAL_reloadTimer(HAL_Handle handle,const uint_least8_t timerNumber)***

关闭 EPWM 外设的输出，该外设会将电源开关置于高阻态

句柄： 驱动程序 (HAL) 句柄

HAL_readPwmPeriod ()

static inline uint16_t HAL_readPwmPeriod(HAL_Handle handle,const PWM_Number_e pwmNumber)

读取 PWM 周期寄存器

句柄： 驱动程序 (HAL) 句柄

pwmNumber： PWM 编号

返回： PWM 周期值

HAL_disablePwm () ***void HAL_disablePwm (HAL_Handle handle);***

关闭 EPWM 外设的输出，该外设会将电源开关置于高阻态

句柄： 驱动程序 (HAL) 句柄

HAL_enablePwm () *void HAL_enablePwm (HAL_Handle handle);*

开启 EPWM 外设的输出，该外设允许控制电源开关

句柄： 驱动程序 (HAL) 句柄

HAL_writeDacData ()

static inline void HAL_writeDacData(HAL_Handle handle,HAL_DacData_t *pDacData)

将 DAC（数模转换）数据写入 PWM 比较器以供 DAC 输出

句柄： 驱动程序 (HAL) 句柄

pDacData： DAC 数据的指针

HAL_writePwmData()

void HAL_writePwmData (HAL_Handle handle, HAL_PwmData_t *pPwmData)

将 PWM 数据写入 PWM 比较器以供电机控制之用

句柄： 驱动程序 (HAL) 句柄

pPwmData： PWM 数据的指针

HAL_readPwmCmpA ()

static inline uint16_t HAL_readPwmCmpA(HAL_Handle handle,const PWM_Number_e pwmNumber)

读取 PWM 比较寄存器 A

句柄： 驱动程序 (HAL) 句柄

pwmNumber： PWM 编号

返回： PWM 比较值

HAL_readPwmCmpB ()

static inline uint16_t HAL_readPwmCmpB(HAL_Handle handle,const PWM_Number_e pwmNumber)

读取 PWM 比较寄存器 B

句柄： 驱动程序 (HAL) 句柄

pwmNumber： PWM 编号

返回： PWM 比较值

HAL_setTrigger () *static inline void HAL_setTrigger(HAL_Handle handle,const int16_t minwidth)*

句柄： 驱动程序 (HAL) 句柄

minwidth：

HAL_acqPwmInt ()

static inline void HAL_acqPwmInt(HAL_Handle handle,const PWM_Number_e pwmNumber)

确认 PWM 的中断

句柄： 驱动程序 (HAL) 句柄

pwmNumber： PWM 编号

HAL_enablePwmInt ()

extern void HAL_enablePwmInt(HAL_Handle handle);

使能 PWM 中断

句柄： 驱动程序 (HAL) 句柄

HAL_hvProtection ()

void HAL_hvProtection(HAL_Handle handle)

运行高压保护逻辑。 设置跳闸寄存器。

句柄： 驱动程序 (HAL) 句柄

3.3.3.4 HAL – CPU 定时器

HAL_setupClks () ***void HAL_setupClks (HAL_Handle halHandle)***

设置时钟

句柄： 驱动程序 (HAL) 句柄

HAL_setupTimers () ***void HAL_setupTimers(HAL_Handle handle,const uint_least16_t systemFreq_MHz);***

设置 CPU 定时器 0 和 1

句柄： 驱动程序 (HAL) 句柄

systemFreq_MHz： 系统频率，MHz

HAL_startTimer () ***static inline void HAL_startTimer(HAL_Handle handle,const uint_least8_t timerNumber)***

启动 CPU 定时器

句柄： 驱动程序 (HAL) 句柄

timerNumber： CPU 定时器编号；0、1 或 2

HAL_stopTimer () ***static inline void HAL_stopTimer(HAL_Handle handle,const uint_least8_t timerNumber)***

停止 CPU 定时器

句柄： 驱动程序 (HAL) 句柄

timerNumber： 定时器编号；0、1 或 2

HAL_setTimerPeriod ()
static inline void HAL_setTimerPeriod(HAL_Handle handle,const uint_least8_t timerNumber, const uint32_t period)

设置 CPU 定时器周期

句柄： 驱动程序 (HAL) 句柄

timerNumber： 定时器编号；0、1 或 2

period： 定时器周期

HAL_getTimerPeriod ()
static inline void HAL_getTimerPeriod(HAL_Handle handle,const uint_least8_t timerNumber, const uint32_t period)

获取 CPU 定时器周期

句柄： 驱动程序 (HAL) 句柄

timerNumber： 定时器编号；0、1 或 2

period： 定时器周期

3.3.3.5 HAL – GPIO 和 LED

HAL_setupGpios () ***void HAL_setupGpios(HAL_Handle handle)***

设置 GPIO

句柄： 驱动程序 (HAL) 句柄

HAL_toggleGpio () ***void HAL_toggleGpio (HAL_Handle handle ,const GPIO_Number_e gpioNumber);***

采用枚举 GPIO_Number_e 并翻转该 GPIO 引脚。

句柄： 驱动程序 (HAL) 句柄

gpioNumber： GPIO 编号

HAL_setGpioHigh () ***static inline void HAL_setGpioHigh(HAL_Handle handle,const GPIO_Number_e gpioNumber)***

将 GPIO 引脚置为高电平。 采用枚举 GPIO_Number_e 并将该 GPIO 引脚置为高电平。

句柄： 驱动程序 (HAL) 句柄

gpioNumber： GPIO 编号

HAL_setGpioLow () ***static inline void HAL_setGpioLowHAL_Handle handle,const GPIO_Number_e gpioNumber)***

将 GPIO 引脚置为低电平。 采用枚举 GPIO_Number_e 并将该 GPIO 引脚置为低电平。

句柄： 驱动程序 (HAL) 句柄

gpioNumber： GPIO 编号

HAL_toggleLed

定义开启 LED 的函数

```
#define HAL_toggleLed            HAL_toggleGpio
```

3.3.3.6 HAL – 其他

HAL_init() ***HAL_Handle HAL_init(void *pMemory,const size_t numBytes)***

初始化驱动程序 (HAL) 对象并返回该 HAL 对象的句柄

pMemory：指向驱动程序对象内存的指针

numBytes：为驱动程序对象分配的字节数，字节

返回：驱动程序 (HAL) 对象句柄

HAL_initIntVectorTable () ***void HAL_initIntVectorTable(HAL_Handle handle)***

初始化中断矢量表

句柄：驱动程序 (HAL) 句柄

HAL_setParams () ***void HAL_setParams(HAL_Handle handle,const USER_Params *pUserParams)***

设置硬件抽象层参数

设置微控制器外设。 创建 ADC 电压和电流转换的所有换算系数。 设置电压和电流测量的初始偏移值。

句柄：驱动程序 (HAL) 句柄

pUserParams：用户参数的指针

HAL_setupFlash () ***void HAL_setupFlash(HAL_Handle handle)***

设置闪存。

句柄：驱动程序 (HAL) 句柄

HAL_setupPie () ***void HAL_setupPie(HAL_Handle handle)***

设置外设中断扩展 (PIE)。

句柄：驱动程序 (HAL) 句柄

HAL_setupPll () ***void HAL_setupPll(HAL_Handle handle,const PLL_ClkFreq_e clkFreq)***

设置锁相环 (PLL)

句柄：驱动程序 (HAL) 句柄

clkFreq：时钟频率

HAL_setupPeripheralClks () ***void HAL_setupPeripheralClks (HAL_Handle handle)***

设置外设时钟。

句柄： 驱动程序 (HAL) 句柄

HAL_getOscTrimValue ()

uint16_t HAL_getOscTrimValue(int16_t coarse, int16_t fine);

将振荡器粗调值和细调值转换为 16 位单字值。

句柄： 驱动程序 (HAL) 句柄

coarse： 振荡器调节的粗调部分

fine： 振荡器调节的细调部分

返回： 合并的调节值

HAL_OscTempComp () *void HAL_OscTempComp(HAL_Handle handle);*

执行振荡器 1 和 2 校准功能。

句柄： 驱动程序 (HAL) 句柄

HAL_osc1Comp () *void HAL_osc1Comp(HAL_Handle handle, const int16_t sensorSample);*

根据输入采样执行振荡器 1 校准。

句柄： 驱动程序 (HAL) 句柄

HAL_osc2Comp ()

void HAL_osc2Comp(HAL_Handle handle, const int16_t sensorSample);

根据输入采样执行振荡器 2 校准。

句柄： 驱动程序 (HAL) 句柄

HAL_setupFaults () *extern void HAL_setupFaults(HAL_Handle handle);*

配置故障保护逻辑

设置触发区输入，以便当微控制器外部的比较器信号触发故障时，EPWM 外设块将强制电源开关进入高阻态。

句柄： 驱动程序 (HAL) 句柄

HAL_setParams () *void HAL_setParams(HAL_Handle handle,const USER_Params *pUserParams)*

设置驱动程序参数

句柄： 驱动程序 (HAL) 句柄

pUserParams： 用户参数的指针

HAL_enableDebugInt ()

void HAL_enableDebugInt (HAL_Handle handle);

使能实时调试全局中断

句柄： 驱动程序 (HAL) 句柄

HAL_enableGlobalInts ()

void HAL_enableGlobalInts(HAL_Handle handle);

使能全局中断

句柄： 驱动程序 (HAL) 句柄

HAL_disableGlobalInts ()

void HAL_disableGlobalInts(HAL_Handle handle);

禁用全局中断

句柄： 驱动程序 (HAL) 句柄

HAL_disableWdog () *extern void HAL_disableWdog(HAL_Handle handle);*

禁用看门狗

句柄： 驱动程序 (HAL) 句柄

3.3.4 用户设置 – user.c、user.h、userParams.h

3.3.4.1 USER 枚举和结构

Struct_USER_Params_

用户参数的结构。

```
typedef struct _USER_Params_
{
  float_t       iqFullScaleCurrent_A;         //!< Defines the full scale current
for the IQ variables, A
  float_t       iqFullScaleVoltage_V;         //!< Defines the full scale voltage
for the IQ variable, V
  float_t       iqFullScaleFreq_Hz;           //!< Defines the full scale
frequency for IQ variable, Hz

  uint_least16_t  numIsrTicksPerCtrlTick;       //!< Defines the number of
Interrupt Service Routine (ISR) clock ticks per controller clock tick
  uint_least16_t  numCtrlTicksPerCurrentTick;   //!< Defines the number of
controller clock ticks per current controller clock tick
  uint_least16_t  numCtrlTicksPerEstTick;       //!< Defines the number of
controller clock ticks per estimator clock tick
  uint_least16_t  numCtrlTicksPerSpeedTick;     //!< Defines the number of
controller clock ticks per speed controller clock tick
  uint_least16_t  numCtrlTicksPerTrajTick;      //!< Defines the number of
controller clock ticks per trajectory clock tick
  uint_least8_t   numCurrentSensors;            //!< Defines the number of
current sensors
  uint_least8_t   numVoltageSensors;            //!< Defines the number of
voltage sensors

  float_t       offsetPole_rps;               //!< Defines the pole location for
the voltage and current offset estimation, rad/s
  float_t       fluxPole_rps;                 //!< Defines the pole location for
the flux estimation, rad/s
  float_t       zeroSpeedLimit;               //!< Defines the low speed limit
for the flux integrator, pu
  float_t       forceAngleFreq_Hz;            //!< Defines the force angle
frequency, Hz
  float_t       maxAccel_Hzps;                //!< Defines the maximum
acceleration for the speed profiles, Hz/s
  float_t       maxAccel_est_Hzps;            //!< Defines the maximum
acceleration for the estimation speed profiles, Hz/s
  float_t       directionPole_rps;            //!< Defines the pole location for
the direction filter, rad/s
  float_t       speedPole_rps;                //!< Defines the pole location for
the speed control filter, rad/s
  float_t       dcBusPole_rps;                //!< Defines the pole location for
the DC bus filter, rad/s
  float_t       fluxFraction;                 //!< Defines the flux fraction for
Id rated current estimation
  float_t       indEst_speedMaxFraction;      //!< Defines the fraction of
SpeedMax to use during inductance estimation
  float_t       powerWarpGain;                //!< Defines the PowerWarp gain for
computing Id reference

  uint_least16_t  systemFreq_MHz;               //!< Defines the system clock
frequency, MHz

  float_t       pwmPeriod_usec;               //!< Defines the Pulse Width
Modulation (PWM) period, usec
  float_t       voltage_sf;                   //!< Defines the voltage scale
factor for the system
  float_t       current_sf;                   //!< Defines the current scale
```

```
factor for the system
  float_t       voltageFilterPole_rps;        //!< Defines the analog voltage
filter pole location, rad/s
  float_t       maxVsMag_pu;                  //!< Defines the maximum voltage
magnitude, pu
  float_t       estKappa;                     //!< Defines the convergence factor
for the estimator

  MOTOR_Type_e    motor_type;                   //!< Defines the motor type
  uint_least16_t  motor_numPolePairs;           //!< Defines the number of pole
pairs for the motor

  float_t       motor_ratedFlux;              //!< Defines the rated flux of the
motor, V/Hz
  float_t       motor_Rr;                     //!< Defines the rotor resistance,
ohm
  float_t       motor_Rs;                     //!< Defines the stator resistance,
ohm
  float_t       motor_Ls_d;                   //!< Defines the direct stator
inductance, H
  float_t       motor_Ls_q;                   //!< Defines the quadrature stator
inductance, H
  float_t       maxCurrent;                   //!< Defines the maximum current
value, A
  float_t       maxCurrent_resEst;            //!< Defines the maximum current
value for resistance estimation, A
  float_t       maxCurrent_indEst;            //!< Defines the maximum current
value for inductance estimation, A
  float_t       maxCurrentSlope;              //!< Defines the maximum current
slope for Id current trajectory
  float_t       maxCurrentSlope_powerWarp;    //!< Defines the maximum current
slope for Id current trajectory during PowerWarp
  float_t       IdRated;                      //!< Defines the Id rated current
value, A
  float_t       IdRatedFraction_indEst;       //!< Defines the fraction of Id
rated current to use during inductance estimation
  float_t       IdRatedFraction_ratedFlux;    //!< Defines the fraction of Id
rated current to use during rated flux estimation
  float_t       IdRated_delta;                //!< Defines the Id rated delta
current value, A
  float_t       fluxEstFreq_Hz;               //!< Defines the flux estimation
frequency, Hz

  uint_least32_t  ctrlWaitTime[CTRL_numStates]; //!< Defines the wait times for
each controller state, estimator ticks
  uint_least32_t  estWaitTime[EST_numStates];   //!< Defines the wait times for
each estimator state, estimator ticks
  uint_least32_t  FluxWaitTime[EST_Flux_numStates]; //!< Defines the wait times
for each Ls estimator, estimator ticks
  uint_least32_t  LsWaitTime[EST_Ls_numStates]; //!< Defines the wait times for
each Ls estimator, estimator ticks
  uint_least32_t  RsWaitTime[EST_Rs_numStates]; //!< Defines the wait times for
each Rs estimator, estimator ticks
  uint_least32_t  ctrlFreq_Hz;                  //!< Defines the controller
frequency, Hz
  uint_least32_t  estFreq_Hz;                   //!< Defines the estimator
frequency, Hz
  uint_least32_t  RoverL_estFreq_Hz;            //!< Defines the R/L estimation
frequency, Hz
  uint_least32_t  trajFreq_Hz;                  //!< Defines the trajectory
frequency, Hz

  float_t       ctrlPeriod_sec;               //!< Defines the controller
execution period, sec

  float_t       maxNegativeIdCurrent_a;       //!< Defines the maximum negative
```

```
current that the Id PID is allowed to go to, A

  USER_ErrorCode_e  errorCode;
} USER_Params;
```

USER_ErrorCode_e

用户错误代码的结构。

```
typedef enum
{
  USER_ErrorCode_NoError=0,                          //!< no error error code
  USER_ErrorCode_iqFullScaleCurrent_A_High=1,        //!< iqFullScaleCurrent_A
too high error code
  USER_ErrorCode_iqFullScaleCurrent_A_Low=2,         //!< iqFullScaleCurrent_A
too low error code
  USER_ErrorCode_iqFullScaleVoltage_V_High=3,        //!< iqFullScaleVoltage_V
too high error code
  USER_ErrorCode_iqFullScaleVoltage_V_Low=4,         //!< iqFullScaleVoltage_V
too low error code
  USER_ErrorCode_iqFullScaleFreq_Hz_High=5,          //!< iqFullScaleFreq_Hz too
high error code
  USER_ErrorCode_iqFullScaleFreq_Hz_Low=6,           //!< iqFullScaleFreq_Hz too
low error code
  USER_ErrorCode_numPwmTicksPerIsrTick_High=7,       //!< numPwmTicksPerIsrTick
too high error code
  USER_ErrorCode_numPwmTicksPerIsrTick_Low=8,        //!< numPwmTicksPerIsrTick
too low error code
  USER_ErrorCode_numIsrTicksPerCtrlTick_High=9,      //!< numIsrTicksPerCtrlTick
too high error code
  USER_ErrorCode_numIsrTicksPerCtrlTick_Low=10,      //!< numIsrTicksPerCtrlTick
too low error code
  USER_ErrorCode_numCtrlTicksPerCurrentTick_High=11, //!<
numCtrlTicksPerCurrentTick too high error code
  USER_ErrorCode_numCtrlTicksPerCurrentTick_Low=12,  //!<
numCtrlTicksPerCurrentTick too low error code
  USER_ErrorCode_numCtrlTicksPerEstTick_High=13,     //!< numCtrlTicksPerEstTick
too high error code
  USER_ErrorCode_numCtrlTicksPerEstTick_Low=14,      //!< numCtrlTicksPerEstTick
too low error code
  USER_ErrorCode_numCtrlTicksPerSpeedTick_High=15,   //!<
numCtrlTicksPerSpeedTick too high error code
  USER_ErrorCode_numCtrlTicksPerSpeedTick_Low=16,    //!<
numCtrlTicksPerSpeedTick too low error code
  USER_ErrorCode_numCtrlTicksPerTrajTick_High=17,    //!<
numCtrlTicksPerTrajTick too high error code
  USER_ErrorCode_numCtrlTicksPerTrajTick_Low=18,     //!<
numCtrlTicksPerTrajTick too low error code
  USER_ErrorCode_numCurrentSensors_High=19,          //!< numCurrentSensors too
high error code
  USER_ErrorCode_numCurrentSensors_Low=20,           //!< numCurrentSensors too
low error code
  USER_ErrorCode_numVoltageSensors_High=21,          //!< numVoltageSensors too
high error code
  USER_ErrorCode_numVoltageSensors_Low=22,           //!< numVoltageSensors too
low error code
  USER_ErrorCode_offsetPole_rps_High=23,             //!< offsetPole_rps too
high error code
  USER_ErrorCode_offsetPole_rps_Low=24,              //!< offsetPole_rps too low
error code
  USER_ErrorCode_fluxPole_rps_High=25,               //!< fluxPole_rps too high
error code
  USER_ErrorCode_fluxPole_rps_Low=26,                //!< fluxPole_rps too low
error code
  USER_ErrorCode_zeroSpeedLimit_High=27,             //!< zeroSpeedLimit too
```

```
high error code
  USER_ErrorCode_zeroSpeedLimit_Low=28,              //!< zeroSpeedLimit too low
error code
  USER_ErrorCode_forceAngleFreq_Hz_High=29,          //!< forceAngleFreq_Hz too
high error code
  USER_ErrorCode_forceAngleFreq_Hz_Low=30,           //!< forceAngleFreq_Hz too
low error code
  USER_ErrorCode_maxAccel_Hzps_High=31,              //!< maxAccel_Hzps too high
error code
  USER_ErrorCode_maxAccel_Hzps_Low=32,               //!< maxAccel_Hzps too low
error code
  USER_ErrorCode_maxAccel_est_Hzps_High=33,          //!< maxAccel_est_Hzps too
high error code
  USER_ErrorCode_maxAccel_est_Hzps_Low=34,           //!< maxAccel_est_Hzps too
low error code
  USER_ErrorCode_directionPole_rps_High=35,          //!< directionPole_rps too
high error code
  USER_ErrorCode_directionPole_rps_Low=36,           //!< directionPole_rps too
low error code
  USER_ErrorCode_speedPole_rps_High=37,              //!< speedPole_rps too high
error code
  USER_ErrorCode_speedPole_rps_Low=38,               //!< speedPole_rps too low
error code
  USER_ErrorCode_dcBusPole_rps_High=39,              //!< dcBusPole_rps too high
error code
  USER_ErrorCode_dcBusPole_rps_Low=40,               //!< dcBusPole_rps too low
error code
  USER_ErrorCode_fluxFraction_High=41,               //!< fluxFraction too high
error code
  USER_ErrorCode_fluxFraction_Low=42,                //!< fluxFraction too low
error code
  USER_ErrorCode_indEst_speedMaxFraction_High=43,    //!<
indEst_speedMaxFraction too high error code
  USER_ErrorCode_indEst_speedMaxFraction_Low=44,     //!<
indEst_speedMaxFraction too low error code
  USER_ErrorCode_powerWarpGain_High=45,              //!< powerWarpGain too high
error code
  USER_ErrorCode_powerWarpGain_Low=46,               //!< powerWarpGain too low
error code
  USER_ErrorCode_systemFreq_MHz_High=47,             //!< systemFreq_MHz too
high error code
  USER_ErrorCode_systemFreq_MHz_Low=48,              //!< systemFreq_MHz too low
error code
  USER_ErrorCode_pwmFreq_kHz_High=49,                //!< pwmFreq_kHz too high
error code
  USER_ErrorCode_pwmFreq_kHz_Low=50,                 //!< pwmFreq_kHz too low
error code
  USER_ErrorCode_voltage_sf_High=51,                 //!< voltage_sf too high
error code
  USER_ErrorCode_voltage_sf_Low=52,                  //!< voltage_sf too low
error code
  USER_ErrorCode_current_sf_High=53,                 //!< current_sf too high
error code
  USER_ErrorCode_current_sf_Low=54,                  //!< current_sf too low
error code
  USER_ErrorCode_voltageFilterPole_Hz_High=55,       //!< voltageFilterPole_Hz
too high error code
  USER_ErrorCode_voltageFilterPole_Hz_Low=56,        //!< voltageFilterPole_Hz
too low error code
  USER_ErrorCode_maxVsMag_pu_High=57,                //!< maxVsMag_pu too high
error code
  USER_ErrorCode_maxVsMag_pu_Low=58,                 //!< maxVsMag_pu too low
error code
  USER_ErrorCode_estKappa_High=59,                   //!< estKappa too high
error code
  USER_ErrorCode_estKappa_Low=60,                    //!< estKappa too low error
```

```
code
  USER_ErrorCode_motor_type_Unknown=61,              //!< motor type unknown
error code
  USER_ErrorCode_motor_numPolePairs_High=62,         //!< motor_numPolePairs too
high error code
  USER_ErrorCode_motor_numPolePairs_Low=63,          //!< motor_numPolePairs too
low error code
  USER_ErrorCode_motor_ratedFlux_High=64,            //!< motor_ratedFlux too
high error code
  USER_ErrorCode_motor_ratedFlux_Low=65,             //!< motor_ratedFlux too
low error code
  USER_ErrorCode_motor_Rr_High=66,                   //!< motor_Rr too high
error code
  USER_ErrorCode_motor_Rr_Low=67,                    //!< motor_Rr too low error
code
  USER_ErrorCode_motor_Rs_High=68,                   //!< motor_Rs too high
error code
  USER_ErrorCode_motor_Rs_Low=69,                    //!< motor_Rs too low error
code
  USER_ErrorCode_motor_Ls_d_High=70,                 //!< motor_Ls_d too high
error code
  USER_ErrorCode_motor_Ls_d_Low=71,                  //!< motor_Ls_d too low
error code
  USER_ErrorCode_motor_Ls_q_High=72,                 //!< motor_Ls_q too high
error code
  USER_ErrorCode_motor_Ls_q_Low=73,                  //!< motor_Ls_q too low
error code
  USER_ErrorCode_maxCurrent_High=74,                 //!< maxCurrent too high
error code
  USER_ErrorCode_maxCurrent_Low=75,                  //!< maxCurrent too low
error code
  USER_ErrorCode_maxCurrent_resEst_High=76,          //!< maxCurrent_resEst too
high error code
  USER_ErrorCode_maxCurrent_resEst_Low=77,           //!< maxCurrent_resEst too
low error code
  USER_ErrorCode_maxCurrent_indEst_High=78,          //!< maxCurrent_indEst too
high error code
  USER_ErrorCode_maxCurrent_indEst_Low=79,           //!< maxCurrent_indEst too
low error code
  USER_ErrorCode_maxCurrentSlope_High=80,            //!< maxCurrentSlope too
high error code
  USER_ErrorCode_maxCurrentSlope_Low=81,             //!< maxCurrentSlope too
low error code
  USER_ErrorCode_maxCurrentSlope_powerWarp_High=82,  //!<
maxCurrentSlope_powerWarp too high error code
  USER_ErrorCode_maxCurrentSlope_powerWarp_Low=83,   //!<
maxCurrentSlope_powerWarp too low error code
  USER_ErrorCode_IdRated_High=84,                    //!< IdRated too high error
code
  USER_ErrorCode_IdRated_Low=85,                     //!< IdRated too low error
code
  USER_ErrorCode_IdRatedFraction_indEst_High=86,     //!< IdRatedFraction_indEst
too high error code
  USER_ErrorCode_IdRatedFraction_indEst_Low=87,      //!< IdRatedFraction_indEst
too low error code
  USER_ErrorCode_IdRatedFraction_ratedFlux_High=88,  //!<
IdRatedFraction_ratedFlux too high error code
  USER_ErrorCode_IdRatedFraction_ratedFlux_Low=89,   //!<
IdRatedFraction_ratedFlux too low error code
  USER_ErrorCode_IdRated_delta_High=90,              //!< IdRated_delta too high
error code
  USER_ErrorCode_IdRated_delta_Low=91,               //!< IdRated_delta too low
error code
  USER_ErrorCode_fluxEstFreq_Hz_High=92,             //!< fluxEstFreq_Hz too
high error code
  USER_ErrorCode_fluxEstFreq_Hz_Low=93,              //!< fluxEstFreq_Hz too low
```

```
error code
  USER_ErrorCode_ctrlFreq_Hz_High=94,                //!< ctrlFreq_Hz too high
error code
  USER_ErrorCode_ctrlFreq_Hz_Low=95,                 //!< ctrlFreq_Hz too low
error code
  USER_ErrorCode_estFreq_Hz_High=96,                 //!< estFreq_Hz too high
error code
  USER_ErrorCode_estFreq_Hz_Low=97,                  //!< estFreq_Hz too low
error code
  USER_ErrorCode_RoverL_estFreq_Hz_High=98,          //!< RoverL_estFreq_Hz too
high error code
  USER_ErrorCode_RoverL_estFreq_Hz_Low=99,           //!< RoverL_estFreq_Hz too
low error code
  USER_ErrorCode_trajFreq_Hz_High=100,               //!< trajFreq_Hz too high
error code
  USER_ErrorCode_trajFreq_Hz_Low=101,                //!< trajFreq_Hz too low
error code
  USER_ErrorCode_ctrlPeriod_sec_High=102,            //!< ctrlPeriod_sec too
high error code
  USER_ErrorCode_ctrlPeriod_sec_Low=103,             //!< ctrlPeriod_sec too low
error code
  USER_ErrorCode_maxNegativeIdCurrent_a_High=104,    //!< maxNegativeIdCurrent_a
too high error code
  USER_ErrorCode_maxNegativeIdCurrent_a_Low=105,     //!< maxNegativeIdCurrent_a
too low error code
  USER_numErrorCodes=106                             //!< the number of user
error codes
} USER_ErrorCode_e;
```

3.3.4.2 USER Set 和 Compute 函数

USER_setParams() ***void USER_setParams(USER_Params *pUserParams)***

设置用户参数值

pUserParams: 用户参数结构的指针

USER_calcPIgains () ***void USER_calcPIgains(CTRL_Handle handle)***

更新 Id 和 Iq PI 增益

句柄: 控制器 (CTRL) 句柄

USER_computeTorque_Ls_Id_Iq_pu_to_Nm_sf ()
_iq USER_computeTorque_Ls_Id_Iq_pu_to_Nm_sf(void);

计算将通过 Ld、Lq、Id 和 Iq 创建的转矩从标么值转换为 Nm 所需的换算系数

返回: 将通过 (Ld - Lq) * Id * Iq 计算的转矩从标么值转换为 Nm 时所使用的换算系数，IQ24 格式

USER_computeTorque_Flux_Iq_pu_to_Nm_sf ()
_iq USER_computeTorque_Flux_Iq_pu_to_Nm_sf(void);

计算将通过磁通和 Iq 创建的转矩从标么值转换为 Nm 所需的换算系数

返回: 将通过磁通 * Iq 计算的转矩从标么值转换为 Nm 时所使用的换算系数，IQ24 格式

USER_computeFlux_pu_to_Wb_sf ()
_iq USER_computeFlux_pu_to_Wb_sf (void);

计算将磁通从标么值转换为 Wb 所需的换算系数

返回: 将磁通从标么值转换为 Wb 时所使用的换算系数，IQ24 格式

USER_computeFlux_pu_to_VpHz_sf () ***_iq USER_computeFlux_pu_to_VpHz_sf (void);***

计算将磁通从标么值转换为 V/Hz 所需的换算系数

返回: 将磁通从标么值转换为 V/Hz 时所使用的换算系数，IQ24 格式

USER_computeFlux () ***_iq USER_computeFlux(CTRL_Handle handle, const _iq sf);***

根据作为参数发送的换算系数计算磁通（Wb 或 V/Hz）

句柄: 控制器 (CTRL) 句柄

sf: 将磁通从标么值转换为 Wb 或 V/Hz 时所使用的换算系数

返回: 根据作为参数发送的换算系数计算出的磁通（Wb 或 V/Hz），IQ24 格式

USER_computeTorque_Nm ()

_iq USER_computeTorque_Nm(CTRL_Handle handle, const _iq torque_Flux_sf, const _iq torque_Ls_sf);

根据作为参数发送的换算系数计算磁通（Wb 或 V/Hz）

句柄： 控制器 (CTRL) 句柄

torque_Flux_sf： 将通过 (Ld - Lq) * Id * Iq 计算的转矩从标么值转换为 Nm 时所使用的换算系数

torque_Ls_sf： 将通过磁通 * Iq 计算的转矩从标么值转换为 Nm 时所使用的换算系数

返回： 转矩 (Nm)，IQ24 格式

USER_computeTorque_Ibin () *_iq USER_computeTorque_Ibin(CTRL_Handle handle, const _iq torque_Flux_sf, const _iq torque_Ls_sf);*

计算转矩 (Ibin)

句柄： 控制器 (CTRL) 句柄

torque_Flux_sf： 将通过 (Ld - Lq) * Id * Iq 计算的转矩从标么值转换为 Nm 时所使用的换算系数

torque_Ls_sf： 将通过磁通 * Iq 计算的转矩从标么值转换为 Nm 时所使用的换算系数

返回： 转矩 (Nm)，IQ24 格式

3.3.4.3 USER 错误处理函数

USER_checkForErrors ()

void USER_checkForErrors(USER_Params *pUserParams);

检查用户参数值中的错误

pUserParams: 用户参数结构的指针

USER_getErrorCode ()

USER_ErrorCode_e USER_getErrorCode(USER_Params *pUserParams);

获取用户参数中的错误代码

pUserParams: 用户参数结构的指针

返回: 错误代码

USER_setErrorCode ()

void USER_setErrorCode(USER_Params *pUserParams,const USER_ErrorCode_e errorCode);

设置用户参数中的错误代码

pUserParams: 用户参数结构的指针

返回: 错误代码

3.3.5 其他函数

softwareUpdate1p6 ()

void softwareUpdate1p6 (CTRL_Handle handle)

使用正确的 Q 格式重新计算电感。 F2806x 器件 ROM 中的软件版本 taSPIN-FOC v1.6 的漏洞修复。 此函数仅适用于 F2806x 器件，因为 F2802x 和 F2805x 含有 v1.7，不需要此修复。

句柄： 控制器 (CTRL) 句柄

3.4 InstaSPIN-MOTION 和 SpinTAC API

InstaSPIN-MOTION 将 InstaSPIN-FOC 与 LineStream Technologies 开发的 SpinTAC 运动控制套件结合在一起。 这些组件为简单的运动系统提供了低维护、高性能和易于使用的解决方案。 SpinTAC 提供两个解决方案：一个用于速度控制应用，一个用于位置控制应用。

速度控制解决方案的组件包括运动序列生成器（SpinTAC 速度规划）、运动系统配置生成器（SpinTAC 速度移动）、闭环抗扰速度控制器（SpinTAC 速度控制）和系统惯性识别（SpinTAC 速度识别）。

通过 SpinTAC 运动控制库可使用每个组件的多个实例。 它允许 SpinTAC Plan 和 SpinTAC Move 的两个实例。 还允许使用 SpinTAC 控制和 SpinTAC 识别的一个实例控制两个运动轴。

类似地，位置控制解决方案的组件包括信号转换器（SpinTAC 位置转换）、位置序列生成器（SpinTAC 位置规划）、运动系统配置生成器（SpinTAC 位置移动）和闭环抗扰级联位置及速度控制器（SpinTAC 位置控制）。

这些组件打包在一起组成 SpinTAC 运动控制库，该库位于 sw/modules/spintac。 SpinTAC 运动控制库设计成模块化的形式。 这允许开发人员在项目中仅包含所选的 SpinTAC 组件，以最大程度地减小代码尺寸。 任何组件都可以与其他 SpinTAC 组件或第三方软件配合使用。

通过 SpinTAC 运动控制库可使用每个组件的多个实例。 它允许 SpinTAC Plan 和 SpinTAC Move 的两个实例。 还允许使用 SpinTAC 控制和 SpinTAC 识别的一个实例控制两个运动轴。

图 3-2 概述了 SpinTAC 运动套件的速度解决方案的组件如何连接在一起以及如何与 InstaSPIN-FOC 进行连接。

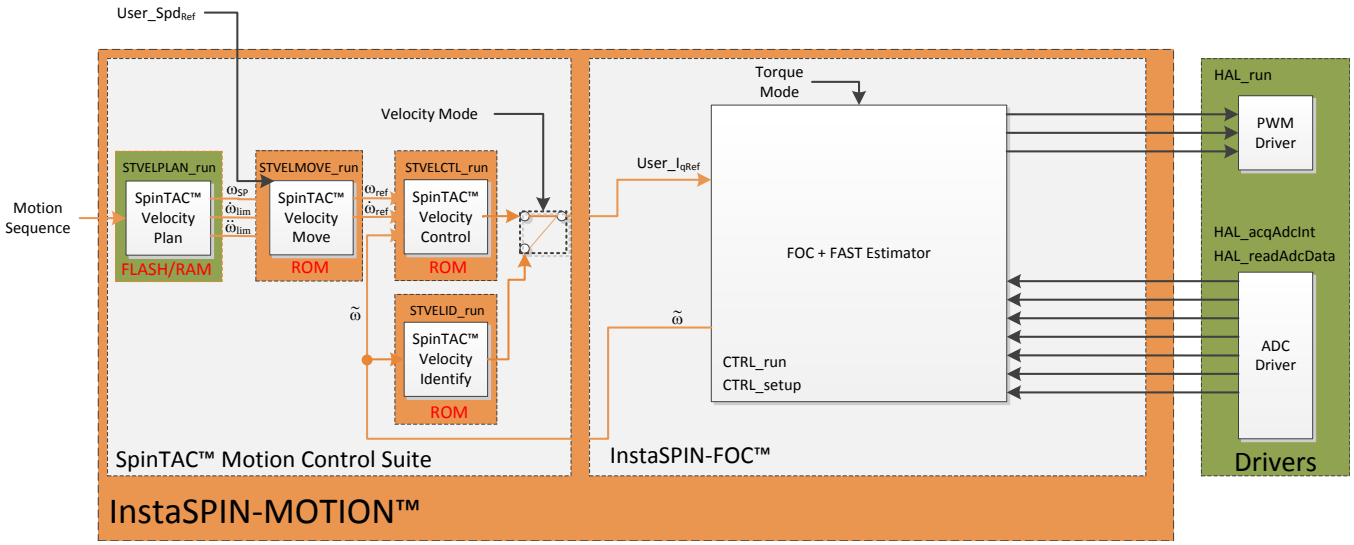


图 **3-2. InstaSPIN-MOTION** 速度控制

图 3-3 概述了 SpinTAC 运动套件的位置解决方案的组件如何连接在一起以及如何与 InstaSPIN-FOC 进行连接。

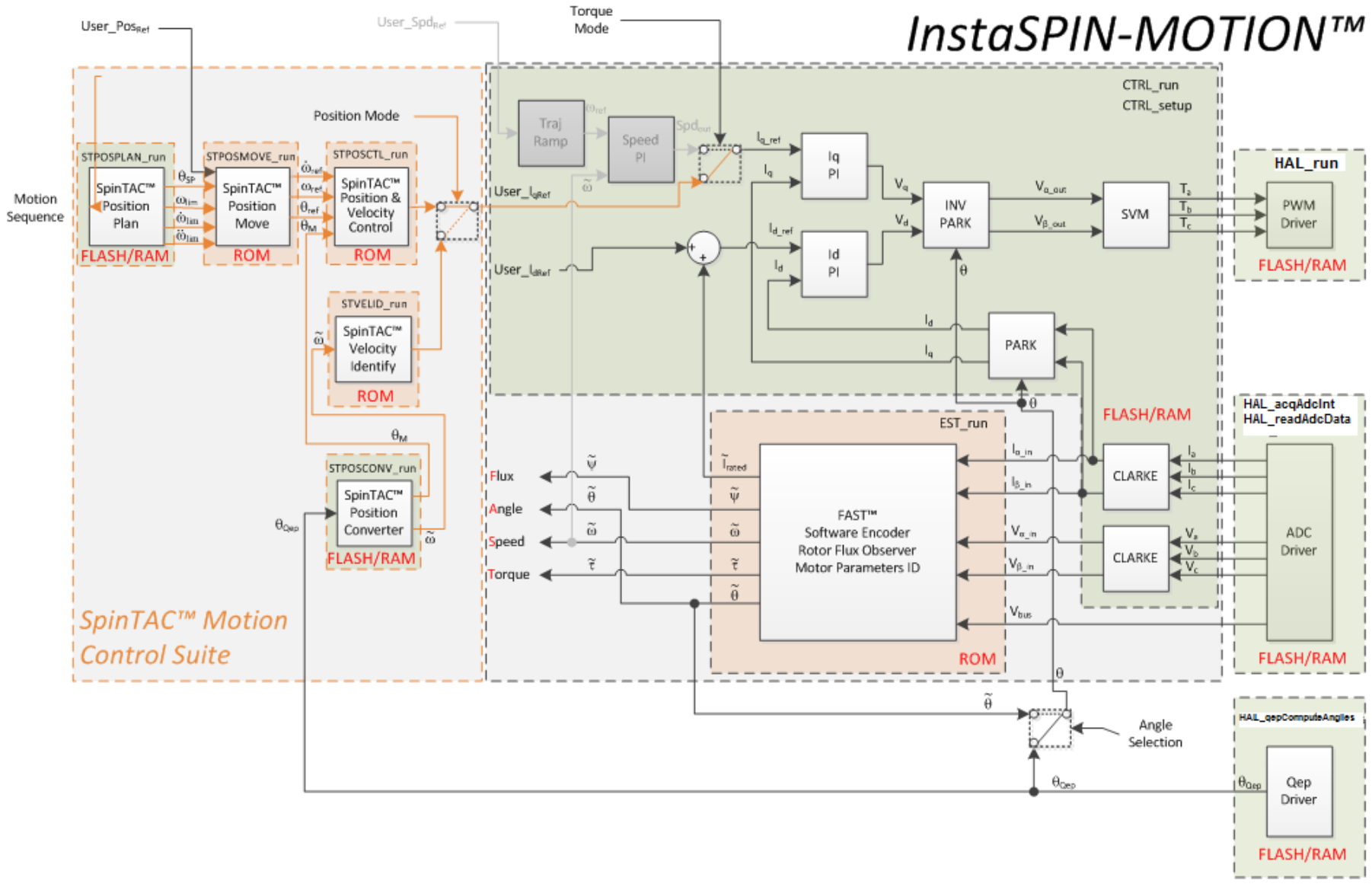


图 **3-3. InstaSPIN-MOTION** 位置控制

SpinTAC 运动控制套件的 API 可细分为下列组件，每个组件的 API 函数都有特定前缀：

组件	前缀
1. SpinTAC 速度控制	STVELCTL
2. SpinTAC 速度移动	STVELMOVE
3. SpinTAC 速度规划	STVELPLAN
4. SpinTAC 速度识别	STVELID
5. SpinTAC 位置转换	STPOSCONV
6. SpinTAC 位置控制	STPOSCTL
7. SpinTAC 位置移动	STPOSMOVE
8. SpinTAC 位置规划	STPOSPLAN

SpinTAC 套件的每个组件都包含一个初始化函数和一个运行函数。 初始化函数旨在建立将用于与 SpinTAC 组件连接的句柄。 运行函数是该组件的主要计算函数。 这些组件中的所有变量都可以通过 get 和 set 函数访问。 3.5.9 节详细介绍了 SpinTAC 组件的常用函数。

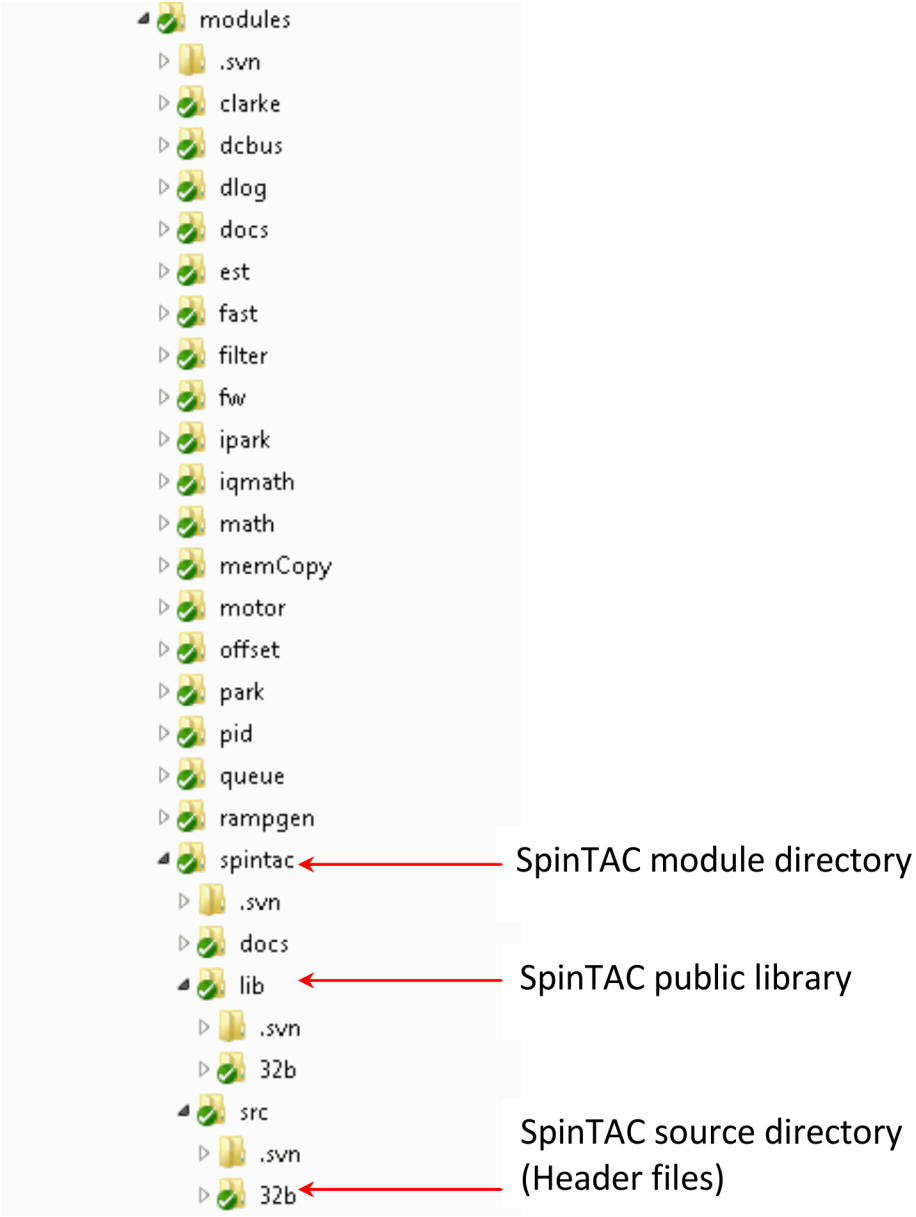


图 **3-4. SpinTAC** 模块目录结构

3.4.1 头文件、公共库和 ROM 库

SpinTAC 套件由一个公共库和一个 ROM 库组成。 公共库也称为 SpinTAC 库文件。 “SpinTAC.lib”位于 /sw/modules/spintac/lib/32b/，用户必须在项目中包含此文件。 如果项目需要 fpu32 支持，应使用库文件“SpinTAC_fpu32.lib”。 请注意，该库仍使用定点 IQMath 库。 表 3-1 列出了需要在用户代码中包含的头文件。 开发人员可包含与所需组件关联的头文件。 该库的源代码未提供。

表 **3-1.** 用户代码头文件

SpinTAC 组件	头文件
SpinTAC 速度控制	spintac_vel_ctl.h
SpinTAC 速度移动	spintac_vel_move.h
SpinTAC 速度规划	spintac_vel_plan.h
SpinTAC 速度识别	spintac_vel_id.h
SpinTAC 位置转换	spintac_pos_conv.h
SpinTAC 位置控制	spintac_pos_ctl.h
SpinTAC 位置移动	spintac_pos_move.h
SpinTAC 位置规划	spintac_pos_plan.h
SpinTAC 版本	spintac_version.h

SpinTAC ROM 库是 TMS320F2805xM 和 TMS320F2806xM 器件的片上只执行 ROM 中嵌入的 C 语言可调用库。 该库的源代码未提供。 该 ROM 库实现了公共库调用的核心 SpinTAC 函数。

3.4.2 版本信息

ST_Ver_t 对象包含了有关 SpinTAC 库的详细版本信息。 表 3-2 详细说明了结构。

表 **3-2. SpinTAC** 版本结构

成员名称	数据类型	说明	**V2.2.7** 示例
Major	uint16_t	库的主要版本	2
Minor	uint16_t	库的次要版本	2
Revision	uint16_t	库的修订版本	7
MathType	ST_MathType_e	编译库时所针对的数学实现。	FIXED_POINT32b
SecureROM	uint32_t	SecureROM 版本	20010008
Date	int32_t	编译库的日期	20140530
Label	uint_least8_t [10]	库的其他信息	TI_C2000

3.4.2.1 返回 **SpinTAC** 版本信息的代码示例

返回 SpinTAC 版本信息有四个步骤。 这些步骤在每个示例实验项目中完成。 下面包括了这些步骤，以展示在项目中包含 SpinTAC 版本信息的简便性。

3.4.2.1.1 包含头文件

这应该通过其余的模块头文件包含来完成。 在实验 8a 示例项目中，此文件包含在 spintac.h 头文件中。 在您的项目中，可通过包含 spintac.h 来完成此步骤。

```
#include "sw/modules/spintac/src/32b/spintac_version.h"
```

3.4.2.1.2 声明全局结构

这应该通过主源文件中的全局变量声明来完成。 在示例实验项目中，此结构包含在 ST_Obj 结构中，后者声明为 spintac.h 头文件的一部分。

```
ST_Obj    st_obj;   // The SpinTAC Object
ST_Handle stHandle; // The SpinTAC Handle
```

如果不希望使用 spintac.h 头文件中声明的 ST_Obj 结构，可参见以下示例。

```
ST_Ver_t      stVersion;       // The SpinTAC Version object
ST_VER_Handle stVersionHandle; // The SpinTAC Version Handle
```

3.4.2.1.3 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这会将所有默认值加载到 SpinTAC 版本结构中。 此步骤可通过运行 spintac.h 头文件中声明的函数 ST_init 来完成。 如果不希望使用此函数，可使用下面的代码示例来设置 SpinTAC 版本结构。

```
// Initialize the SpinTAC Speed Controller Component
stVersionHandle = ST_initVersion (&stVersion, sizeof(ST_Ver_t));
```

3.4.2.1.4 返回版本信息

现在已定义版本句柄，可使用它返回 SpinTAC 库的版本信息。

```
uint16_t      major, minor, revision;      // Variables to return the version numbers
ST_getVersionNumbers(stVersionHandle, &major, &minor, &revision);
```

3.4.3 SpinTAC 结构名称

所有结构数据类型名称都遵守以下模式：

ST_[*对象*] [*功能*] [*子功能*(*可选*)]_**t**

例如，速度控制结构类型的名称为 ST_VelCtl_t。

速度控制的配置子结构类型的名称为 ST_VelCtlCfg_t，包含在 ST_VelCtl_t 结构内。

表 3-3 列出了 SpinTAC 的所有结构和子结构的名称。

表 3-3. SpinTAC 结构名称

组件	结构名称	子结构名称	说明
速度控制	ST_VelCtl_t		主结构
		ST_VelCtlCfg_t	配置子结构
速度移动	ST_VelMove_t		主结构
		ST_VelMoveCfg_t	配置子结构
		ST_VelMoveMsg_t	信息子结构
速度规划	ST_VelPlan_t		主结构

表 3-3. SpinTAC 结构名称 (continued)

组件	结构名称	子结构名称	说明
速度识别	ST_VelId_t		主结构
		ST_VelIdCfg_t	配置子结构
位置转换	ST_PosConv_t		主结构
		ST_PosConvCfg_t	配置子结构
位置控制	ST_PosCtl_t		主结构
		ST_PosCtlCfg_t	配置子结构
位置移动	ST_PosMove_t		主结构
		ST_PosMoveCfg_t	配置子结构
		ST_PosMoveMsg_t	信息子结构
位置规划	ST_PosPlan_t		主结构
版本	ST_Ver_t		主结构

3.4.4 SpinTAC 变量

变量的大致分类如表 3-4 所示。

表 3-4. SpinTAC 变量

变量类别	子类别
配置变量: 用于配置 SpinTAC 组件的变量。	**系统变量:**基于系统参数的已知值。 一些示例: 1. 中断的采样时间 2. 机械旋转与用户单位之间的换算系数。 3. 系统增益是一个特例，其中速度环路或位置-速度环路中的惯性通常通过对恒定惯性系统运行惯性估算功能来获得。 恒定惯性可通过开环中的“惯性识别”估算。 但是，如果系统惯性逐渐变化，则需要在“速度控制”外部计算或估算，并实时发送到“速度控制”。
	保护变量: 用于指示其他变量的上限或下限。 为此，保护变量均带有后缀“Max”或“Min”。 一些示例: 1. OutMax 和 OutMin 是变量 Out 的保护界限。 这些界限来自于定点 IQMath 计算限制、物理限制或通过经验获得的安全限制。 2. VelMax 由具体电机的最大允许速度决定，它是物理限制。 3. BwMax（带宽上限）可在开始阶段设置为较高值，然后在通过测试确定合理上限后降低。
	调节变量: 用于让用户灵活调整 SpinTAC 组件行为的参数。 一些示例: 1. “速度控制”的控制带宽。 2. “速度移动”的速度、加速度/减速度和急动限制。 3. “速度识别”的低通滤波器时间常数。
输入/输出变量: 用于将值传入和传出 SpinTAC 组件的变量	**输入变量:** 要输入 SpinTAC 组件的外部信号的接口。 它们在组件函数被调用前收到外部信号。 示例包括: 1. “速度控制”的参考和反馈信号 2. “惯性识别”的反馈信号。 **控制变量:** 用于控制各个组件的信号。 示例包括: 1. ENB 和 RES 是每个 SpinTAC 组件的控制变量。 通常，当 RES 的值为 false 时，ENB 在上升沿使能组件，当 ENB 值设置为 false 时将禁用组件。 当 RES 的值为 true 时，组件被禁用（处于 RESET 状态），ENB 的值保持为 false。 2. TST 是用于测试“速度移动”组件的控制变量。 如果 TST 的值为 true，“速度移动”只会提供系统配置信息，而不会生成控制器的系统配置；若为 false，将生成系统配置。 设计人员可在将系统配置应用到控制器之前使用此控制位验证计算出的系统配置限制和具体时间。
	输出变量: 包含 SpinTAC 组件的输出。 在调用组件函数之后，这些变量需要设置为适当的外部变量。 示例: 1. “速度控制”和“惯性识别”的控制输出 2. “速度移动”的系统配置参考。 **信息变量:** 只读变量，提供有关 SpinTAC 组件的有用信息。 示例包括: 1. 任意 SpinTAC 组件的状态和错误代码 2. “速度移动”的系统配置时间和实际限制。

表 3-4. SpinTAC 变量 (continued)

变量类别	子类别
内部变量：SpinTAC 组件中的内部变量不应被客户访问到。 修改内部变量非常危险。内部变量存储在声明为大容量存储器的位置，本文档未列出这些变量。	

3.5 SpinTAC API

本部分将介绍每个 SpinTAC 组件的应用程序接口 (API)，包括每个组件的内部状态机、主要函数以及用于控制和配置的数据结构。 先介绍速度解决方案的组件，再介绍位置解决方案的组件。

3.5.1 SpinTAC 速度控制

SpinTAC 速度控制器不同于基于误差的控制设计。 以下示例说明了区别。

速度系统可通过公式 1 描述：

$$J\dot{v}(t) = f\left(v(t), d(t)\right) + u(t) \tag{1}$$

在公式 1 中，v(t)、u(t) 和 d(t) 分别是系统输出（速度）、系统输入（转矩）和外部干扰；f(·) 是未知非线性函数，J 是系统惯性。 在传统控制设计中，使用一个 PI 控制器通过以实验方式确定的比例增益和积分增益来控制这些动态特性。 另一方面，基于模型的控制器根据预定义的线性或非线性模型来响应动态特性。

SpinTAC 速度控制器比较独特，它将非线性项 f(·) 作为可估算和抑制的干扰进行处理。 还通过参数化方法简化了调节过程，该方法允许使用单个调节参数（控制带宽）对动态系统进行高性能控制。

3.5.1.1 SpinTAC 速度控制接口

SpinTAC 速度控制的接口和函数如图 3-5 所示。

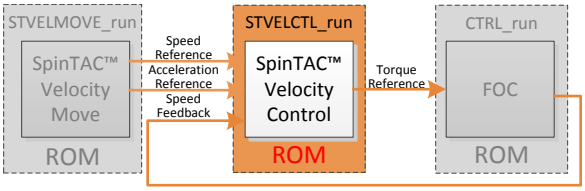


图 3-5. SpinTAC 速度控制接口

注： 在 SpinTAC 速度控制中，惯性值可通过 SpinTAC 速度识别组件获取。 如果反馈的噪声非常大，可启用开关 cfg.FiltEn。 唯一的控制器调节参数是通过 BwScale 设置的带宽。

表 3-5 列出了 SpinTAC 速度控制的接口参数。

表 3-5. SpinTAC 速度控制接口参数

信号类型	结构成员名称	数据类型	说明	值 范围	单位
配置	cfg.Axis	ST_Axis_e	SpinTAC 控制轴 ID	{ST_AXIS0, ST_AXIS1}	
	cfg.T_sec	_iq24	采样时间	(0 , 1]	s
	cfg.InertiaMax	_iq24	速度环路控制的最大系统惯性	(0, 100]	PU · s^2 / pu
	cfg.InertiaMin	_iq24	速度环路控制的最小系统惯性	(0, cfg.InertiaMax]	PU · s^2 / pu
	cfg.OutMax	_iq24	最大控制信号	[-1 , 1]	PU
	cfg.OutMin	_iq24	最小控制信号	[-1, cfg.OutMax]	PU
	cfg.VelMax	_iq24	最大基准信号（系统的最大速度）	[-1 , 1]	pu / s
	cfg.VelMin	_iq24	最小基准信号（系统的最小速度）	[-1 , cfg.VelMax]	pu / s
	cfg.BwScaleMax	_iq24	带宽范围上限	[0.01, min(100, 0.01/ cfg.T_sec)]	
	cfg.BwScaleMin	_iq24	带宽范围下限	[0.01, cfg.BwScaleMax]	
	cfg.FiltEn	bool	启用反馈低通滤波器	false：禁用；true：启用	
输入	VelRef	_iq24	基准信号（速度基准）	[cfg.VelMin, cfg.VelMax]	pu / s
	AccRef	_iq24	前馈信号（加速度基准）		pu / s^2
	VelFdb	_iq24	反馈信号（速度反馈）		pu / s
	Inertia	_iq24	系统惯性	[cfg.InertiaMin, cfg.InertiaMax]	PU · s^2 / pu
	Friction	_iq24	摩擦系数	[0, 5]	PU · s / pu
调节	BwScale	_iq24	带宽范围	[cfg.BwScaleMin, cfg.BwScaleMax]	
控制	ENB	bool	使能位	false：禁用；true：使能	
	RES	bool	复位位	false：不复位；true：复位 ERR_ID，并将 Out 保持为 0	
输出	Out	_iq24	控制输出	[cfg.OutMin, cfg.OutMax]	PU
信息	Bw_radps	_iq20	控制器带宽		rad/s
	STATUS	ST_CtlStatus_e	状态信息	{ST_CTL_IDLE, ST_CTL_INIT, ST_CTL_CONF, ST_CTL_BUSY}	
	ERR_ID	uint16_t	错误代码	请参见表 12-2	

3.5.1.2 SpinTAC 速度控制运行函数

主函数为 STVELCTL_run(ST_VELCTL_Handle handle)，其中 handle 是具体 ST_VelCtl_t 对象的指针。此函数运行 SpinTAC 速度控制。 建议以电流控制器速率的五分之一或四分之一运行此控制器。

void **STVELCTL_run**(ST_VELCTL_Handle handle)

参数：

编号	类型	参数	说明
1	ST_VELCTL_Handle	Handle	ST_VelCtl_t 对象的指针

SpinTAC 速度控制状态转换图如图 3-6 所示。

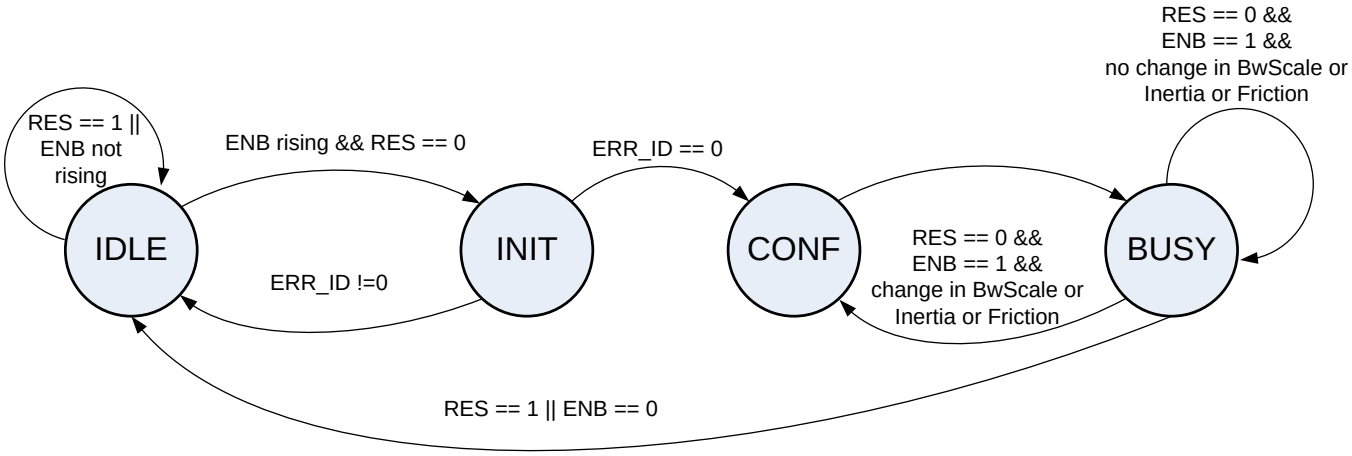


图 **3-6. SpinTAC** 速度控制状态转换图

请注意，在图 3-6 中，IDLE 到 INIT、INIT 到 CONF 以及 CONF 到 BUSY 的状态转换在一个采样时间内发生。 因此，控制器启用后会直接在该采样时间内工作。

表 3-6 说明了 SpinTAC 速度控制状态转换。

表 **3-6. SpinTAC** 速度控制状态转换

起始状态	结束状态	转换条件	操作
IDLE			1. 设置 ENB = false； 2. 将 Out 保持为零 参数验证 1. 验证配置参数，包括 cfg.InertiaMax、cfg.InertiaMin、cfg.BwScaleMax、cfg.BwScaleMin、cfg.Axis、cfg.FiltEn 和 cfg.T_sec。 如果任一检查的变量无效，ERR_ID 都将为非零值。
	INIT	RES == false 并且 ENB 在上升沿并且 ERR_ID == 0	
INIT			参数验证 1. 验证参数，包括 cfg.VelMax、cfg.VelMin、cfg.OutMax、cfg.OutMin。 如果任一检查的变量无效，ERR_ID 都将为非零值。
	IDLE	ERR_ID != 0	设置 ENB = false
	CONF	ERR_ID == 0	
CONF	BUSY		Inertia、BwScale 和 Friction 饱和至指定界限。 如果发生饱和，ERR_ID 将为 1012、1013、1014 或 1016。
BUSY			生成控制信号
	IDLE	RES == true 或 ENB == false	设置 ENB = false
	CONF	RES == false 并且 ENB == true 并且 BwScale、Inertia 或 Friction 发生变化	

3.5.2 SpinTAC 速度移动

SpinTAC 速度移动可生成运动系统配置，满足指定的最大急动、加速度和速度值要求。 生成的基准信号之间的关系为： 速度基准是位置基准的微分；加速度基准是速度基准的微分；急动基准是加速度基准的微分。

3.5.2.1 SpinTAC 速度移动接口

SpinTAC 速度移动的接口和函数如图 3-7 所示。

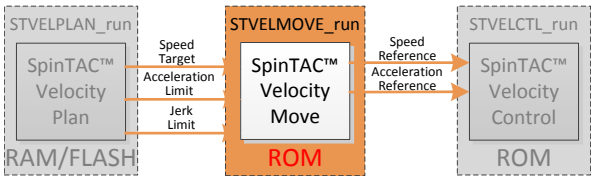


图 3-7. SpinTAC 速度移动接口

表 3-7 列出了 SpinTAC 速度移动的接口参数。

表 3-7. SpinTAC 速度移动接口

信号类型	结构成员名称	数据类型	说明	值范围	单位
配置	cfg.Axis	ST_Axis_e	SpinTAC 移动轴 ID	{ST_AXIS0, ST_AXIS1}	
	cfg.CurveType	ST_MoveCurveType_e	曲线类型	{ST_MOVE_CUR_TRAP, ST_MOVE_CUR_SCRV, ST_MOVE_CUR_STCRV}	
	cfg.T_sec	_iq24	采样时间	(0 , 0.01]	s
	cfg.VelMax	_iq24	系统的最大速度	(0 , 1]	pu / s
	cfg.AccMax	_iq24	系统的最大加速度	[0.001 , 120]	pu / s^2
	cfg.JrkMax	_iq20	系统的最大急动	[0.0005, 2000]	pu / s^3
	cfg.VelStart	_iq24	速度起始值	[-cfg.VelMax , cfg.VelMax]	pu / s
	cfg.IgnoreLimitErrors	bool	如果系统配置界限设置在有效值范围外，则使系统配置限制饱和至有效值范围内	false：提供错误代码，不生成系统配置；true：使系统配置限制饱和，生成系统配置	
消息	msg.ProTime_tick	uint_32	系统配置时间（以采样时间计数为单位）		采样计数
	msg.Acc	_iq24	系统配置的最大加速度		pu / s^2
	msg.Jrk	_iq20	系统配置的最大急动		pu / s^3
输入	VelEnd	_iq24	速度结束值	[-cfg.VelMax , cfg.VelMax]	pu / s
	AccLim	_iq24	加速度限制	[0.001 , cfg.AccMax]	pu / s^2
	JrkLim	_iq20	急动限制	[0.0005, cfg.JrkMax]	pu / s^3
控制	ENB	bool	使能位	false：系统配置已生成或被禁用； true：使能并运行	
	RES	bool	复位位	false：不复位； true：复位 ERR_ID，并将系统配置输出保持为之前的值	
	TST	bool	系统配置测试位	false：不测试； true：测试系统配置	
输出	VelRef	_iq24	速度基准		pu / s
	AccRef	_iq24	加速度基准		pu / s^2
	JrkRef	_iq20	急动基准		pu / s^3

表 3-7. SpinTAC 速度移动接口 (continued)

信号类型	结构成员名称	数据类型	说明	值范围	单位
信息	STATUS	ST_MoveStatus_e	状态信息	{ST_MOVE_IDLE, ST_MOVE_INIT, ST_MOVE_CONF, ST_MOVE_BUSY, ST_MOVE_HALT}	
	DON	bool	系统配置完成指示器	false：正在运行； true：系统配置完成或空闲	
	ERR_ID	uint16_t	错误代码	请参见表 13-1	

为了保持 SpinTAC 速度移动的简便性，所有速度系统配置均使用配置的加速度进行所有移动。 例如，某个系统配置严格上说是减速，但 SpinTAC 速度移动对该系统配置使用加速度限制。

3.5.2.2 SpinTAC 速度移动运行函数

主函数为 STVELMOVE_run(ST_VELMOVE_Handle handle)，其中 handle 是具体 ST_VelMove_t 对象的指针。 此函数运行 SpinTAC 速度移动。 此函数必须以 SpinTAC 速度控制的相同速率运行。

void **STVELMOVE_run**(ST_VELMOVE_Handle handle)

参数：

编号	类型	参数	说明
1	ST_VELMOVE_Handle	Handle	ST_VelMove_t 对象的指针

SpinTAC 速度移动状态转换图如图 3-8 所示。

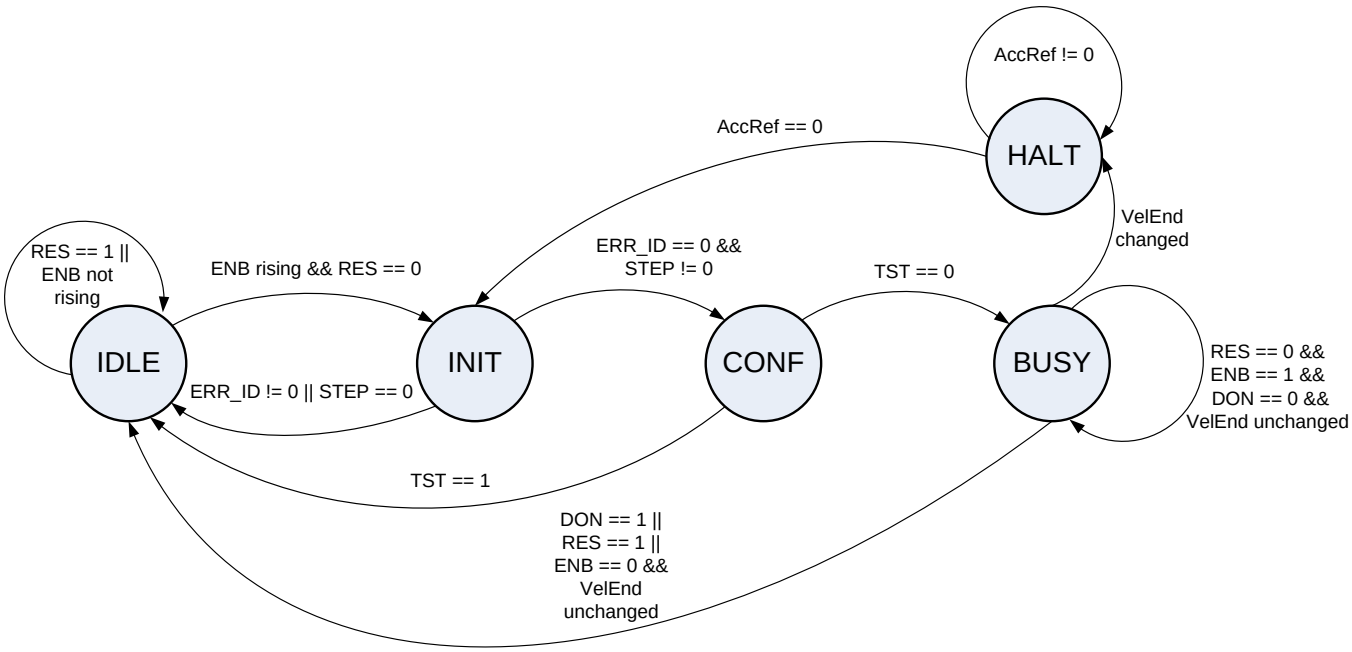


图 3-8. SpinTAC 速度移动状态转换图

请注意，在图 3-8 中，IDLE 到 INIT，再到 CONF 的状态转换在一个采样时间内发生。 因此，在启用系统配置的采样时间内生成系统配置。

表 3-8 说明了 SpinTAC 速度移动的状态。

表 3-8. SpinTAC 速度移动状态转换

起始状态	结束状态	转换条件(1)(2)(3)(4)	操作
IDLE			保持 IDLE 状态 1. 设置 ENB = false； 2. 设置 DON = true； 3. 保持 VelRef、AccRef 和 JrkRef 的值（cfg.VelStart 和 VelEnd 的值只能在 IDLE 状态下设置）
	INIT	RES == false 并且 ENB 在上升沿	
INIT			参数验证 1. 验证配置参数，包括 cfg.VelMax、cfg.VelStart、VelEnd、cfg.JrkMax、cfg.AccMax、JrkLim、AccLim、cfg.CUR_MOD 和 cfg.T_sec。 如果任一检查的变量无效，ERR_ID 都将为非零值。 2. 计算速度阶跃 STEP
	IDLE	ERR_ID != 0 或 STEP == 0	设置 ENB = false
	CONF	ERR_ID == 0 并且 STEP != 0	
CONF			根据配置的参数确定系统配置
	IDLE	TST == true	将 VelEnd 的值设置回 cfg.VelStart 的值
	BUSY	TST == false	
BUSY			生成系统配置 1. 在每个采样时间内更新 VelRef、AccRef、JrkRef 基准值； 2. 如果系统配置完成，将 DON 设置为 true
	IDLE	RES == true 或 ENB == false 或 DON == true	设置 ENB = false
	HALT	VelEnd 在系统配置中发生变化	
HALT			将 AccRef 减至 0。在每个采样时间内更新 AccRef、JrkRef。 准备平滑移动至新 VelEnd。
	INIT	AccRef == 0	

(1) RES 信号提供了复位 SpinTAC 速度移动的功能。

(2) 如果 RES 设置为 true，ENB 将被设置为 false。 当前任何错误都将被丢弃。 当前系统配置将被丢弃，AccRef 设置为 0，VelRef 保持为上一次采样时的值，VelEnd 和 cfg.VelStart 设置为 VelRef 的值。 因此，当 RES 设置回 false 时，可使用保留的速度基准启动新系统配置。

(3) ENB 信号提供了“SpinTAC 速度移动”的启动信号。 仅当 RES 设置为 false 时，ENB 信号才起作用。

(4) TST 位的目的是在不生成轨迹的情况下提供系统配置信息。 该信息包括系统配置时间以及实际的最大加速度和急动。 在 INIT 状态下，ENB 为上升沿时由相关函数接收 TST 信号。 如果 TST 为 true，则处于测试模式。 在测试模式下，系统配置输出 VelRef 将保持 cfg.VelStart 的值；AccRef 将为 0，不受 VelEnd 影响。 测试后，将输出系统配置信息（msg.ProTtime_tick、msg.Acc 和 msg.Jrk），VelEnd 设置回 cfg.VelStart 的值，DON 将设置为 true，ENB 将设置为 false。

3.5.3 SpinTAC 速度规划

SpinTAC 速度规划组件的功能是设置和运行由用户应用确定的运动序列。

3.5.3.1 SpinTAC 速度规划接口

SpinTAC 速度规划的接口和函数如图 3-9 所示，表 3-9 对此进行了说明。

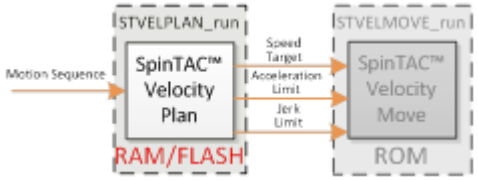


图 3-9. SpinTAC 速度规划接口

表 3-9. SpinTAC 速度规划接口

信号类型	结构成员名称	数据类型	说明	值范围	单位
控制	ENB	bool	使能位	false：禁用； true：使能	
	RES	bool	复位位	false： 不复位； true：复位	
输出	VelEnd	_iq24	当前速度设定点	[-VelMax , VelMax]	pu / s
	AccLim	_iq24	当前加速度限制	[0.002 , AccMax]	pu / s^2
	JrkLim	_iq20	当前急动限制	[0.001, JrkMax]	pu / s^3
信息	Timer_tick	int32_t	当前状态剩余的时间		
	STATUS	ST_PlanStatus_e	状态信息	{ST_PLAN_IDLE, ST_PLAN_INIT, ST_PLAN_CONF, ST_PLAN_BUSY}	
	CurState	uint16_t	当前状态索引	[0, StateNum)	
	CurTran	uint16_t	当前转换索引	[0, TranNum)	
	FsmState	ST_PlanFsmState_e	一种状态，用于指示是处于转换中，还是处于某个状态，还是正在等待转换	{ ST_FSM_STATE_STAY, ST_FSM, _STATE_COND, ST_FSM _STATE_TRAN }	
	DON	bool	规划完成指示器	false：未完成； true：已完成	
	ERR_ID	uint16_t	导致错误的规划函数	请参见表 13-4	
	CfgError.ERR_idx	uint16_t	发生错误的组件索引		
	CfgError.ERR_code	uint16_t	导致错误的条件	请参见表 13-4	

3.5.3.2 SpinTAC 速度规划主函数

主函数为 STVELPLAN_run(ST_VELPLAN_Handle handle)，其中 handle 是具体 ST_VelPlan_t 对象的指针。 此函数运行 SpinTAC 速度规划。 可在程序的主循环中运行此函数。

void **STVELPLAN_run**(ST_VELPLAN_Handle handle)

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 对象的指针

SpinTAC 速度规划中处理状态定时器递减的函数是 STVELPLAN_runTick(ST_VELPLAN_Handle handle)，其中 handle 是具体 ST_VelPlan_t 对象的指针。 此函数递减 SpinTAC 速度规划的状态定时器。 此函数必须以 SpinTAC 速度控制的相同速率运行。

void **STVELPLAN_runTick**(ST_VELPLAN_Handle handle)

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 对象的指针

SpinTAC 速度规划状态转换图如图 3-10 所示。

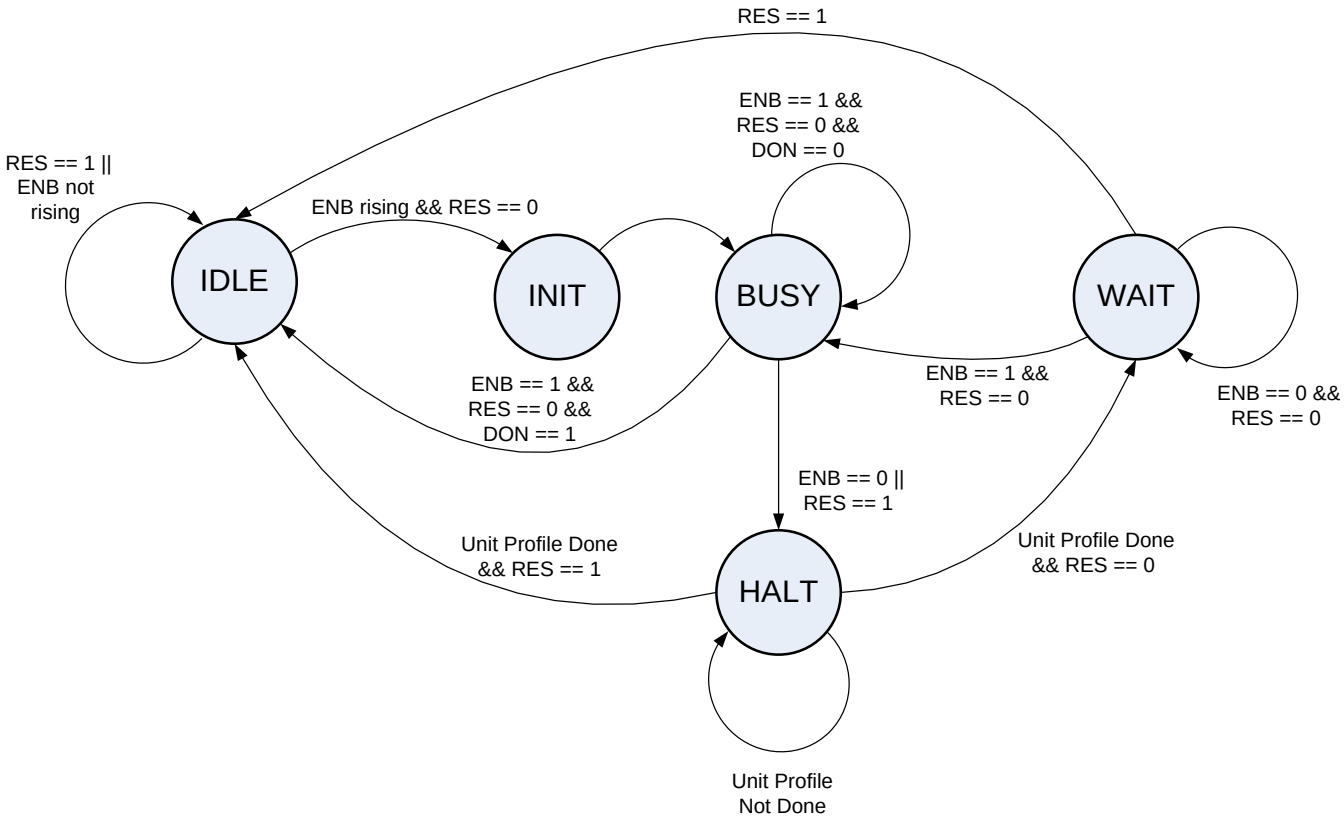


图 3-10. SpinTAC 速度规划状态转换图

请注意，在图 3-10 中，IDLE 到 INIT，再到 BUSY 的状态转换在一个采样时间内发生。

表 3-10 说明了 SpinTAC 速度规划的状态。

表 3-10. SpinTAC 速度规划状态转换

起始状态	结束状态	转换条件(1)(2)(3)(4)(5)	操作
IDLE			保持 IDLE 状态 1. 设置 ENB = false； 2. 保持 VelEnd、AccLim 和 JrkLim 的值
	INIT	RES == false 并且 ENB 在上升沿	
INIT	BUSY		参数验证 1. 复位内部状态； 2. 输入状态 0 并执行为输入状态 0 定义的操作。
BUSY			操作规划
	IDLE	RES == false 并且 ENB == true 并且 DON == true	设置 ENB = false
	HALT	RES == true 或 ENB == false	加载暂停状态系统配置

表 3-10. SpinTAC 速度规划状态转换 (continued)

起始状态	结束状态	转换条件(1)(2)(3)(4)(5)	操作
HALT	IDLE	HALT 状态定时器启动并且 RES == true	加载状态 0 的配置
	WAIT	HALT 状态定时器启动并且 RES == false	
WAIT	IDLE	RES == true	加载状态 0 的配置
	HALT	ENB == true	加载上个状态的配置

(1) RES 信号提供了复位 SpinTAC 速度规划的功能。

(2) 当 RES 为 false 时，ENB 信号控制 SpinTAC 速度规划的运行。

(3) 如果在 SpinTAC 速度规划运行时将 ENB 设置为 false，SpinTAC 速度规划将发出速度设定点和 HALT 状态的限制。 完成单位系统配置后，SpinTAC 速度规划将进入 WAIT 状态，并且只能在 ENB 设置为 true 后继续规划。

(4) 如果 RES 设置为 true，ENB 将被设置为 false。 SpinTAC 速度规划随后将发出速度设定点和状态 0 的限制，并进入 IDLE 状态。

(5) 实际上，ENB 功能起到暂停/停止按钮的作用，而 RES 功能起到停止按钮的作用。

表 3-11 列出了可用于执行操作（如设置、获取、添加和删除 SpinTAC 速度规划的配置和运行时参数）的函数。 3.5.9 节 更详细地介绍了这些函数。

表 3-11. SpinTAC 速度规划附加函数

函数组	函数名称	说明
初始化	STVELPLAN_init	初始化 SpinTAC 速度规划
配置	STVELPLAN_setCfg	设置系统和保护参数
	STVELPLAN_setCfgArray	设置配置数组
	STVELPLAN_setCfgHaltState	设置 HALT 状态的参数
	STVELPLAN_addCfgState	添加新状态
	STVELPLAN_addCfgVar	添加新变量
	STVELPLAN_addCfgCond	添加比较变量和静态值的新条件
	STVELPLAN_addCfgVarCond	添加比较两个变量的新条件
	STVELPLAN_addCfgTran	添加新转换
	STVELPLAN_addCfgAct	添加新操作
运行时	STVELPLAN_run	运行 SpinTAC 速度规划。 可从主循环运行。
	STVELPLAN_runTick	运行 SpinTAC 速度规划定时器功能。 必须从 ISR 运行。
	STVELPLAN_setVar	设置运行期间的变量值
	STVELPLAN_getVar	获取运行期间的变量值
	STVELPLAN_reset	复位 SpinTAC 速度规划和配置
	STVELPLAN_setUnitProfDone	设置当前运行的系统配置是否完成
规划修改和调试函数（提供在运行时修改 SpinTAC 速度规划的功能）	STVELPLAN_getCfgStateNum	获取已配置状态数
	STVELPLAN_getCfgVarNum	获取已配置变量数
	STVELPLAN_getCfgCondNum	获取已配置条件数
	STVELPLAN_getCfgTranNum	获取已配置转换数
	STVELPLAN_getCfgActNum	获取已配置操作数
	STVELPLAN_getCfg	获取系统和保护参数
	STVELPLAN_getCfgHaltState	获取 HALT 状态的参数
	每个带 -Add 后缀的函数有其它三个函数：-Del、-Set 和 -Get，分别用于删除项目、设置项目和获取项目。 这些函数允许在运行时修改 SpinTAC 速度规划。	

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

3.5.4 SpinTAC 速度识别

SpinTAC 速度识别按照所应用的扭矩系统配置和所测量的速度反馈来估算系统惯性。

可以用公式 2 描述忽略干扰的非线性简单运动系统。

$$J\dot{v}(t) = -Bv(t) + u(t) \tag{2}$$

在公式 2 中，v(t) 和 u(t) 分别是速度和控制输入；J 和 B 分别是惯性比和摩擦系数。

SpinTAC 速度识别应用连续的转矩系统配置并相对于速度反馈估算系统惯性比。 估算的惯性比和摩擦系数应提供给 SpinTAC 控制器。 SpinTAC 速度识别应该用于估算速度解决方案和位置解决方案的惯性。

3.5.4.1 SpinTAC 速度识别接口

SpinTAC 速度识别的接口如节 8.5.2.1.1 所示，表 3-12 对此进行了说明。

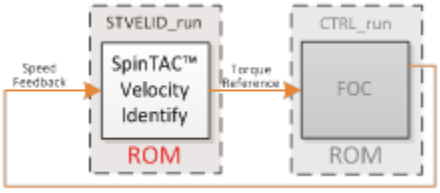


图 3-11. SpinTAC 速度识别接口

表 3-12. SpinTAC 速度识别接口和参数

信号类型	变量名称	数据类型	说明	值范围	单位
配置	cfg.T_sec	_iq24	采样时间	(0 , 0.01]	s
	cfg.VelMax	_iq24	系统的最大速度	(0 , 1]	pu / s
	cfg.OutMax	_iq24	最大速度环路控制信号	(0 , 1]	PU
	cfg.OutMin	_iq24	最小速度环路控制信号	[-1, 0)	PU
	cfg.VelPos	_iq24	速度正值	(0 , cfg.VelMax]	pu / s
	cfg.OutPos	_iq24	控制信号正值	(0 , cfg.OutMax]	PU
	cfg.OutNeg	_iq24	控制信号负值	[cfg.OutMin, 0)	PU
	cfg.LpfTime_tick	int16_t	反馈信号低通滤波器时间常数	[1 , 100]	采样计数
	cfg.TimeOut_sec	_iq24	惯性识别允许的最大时间	[1, 10]	s
	cfg.RampTime_sec	_iq24	在惯性估算过程中控制信号达到 1.0 PU 所允许的时间	[cfg.T, 25 秒]	s
输入	VelFdb	_iq24	速度反馈		pu / s
控制	ENB	bool	使能位	false：禁用；true：使能	
	RES	bool	复位位	false：不复位；true：复位 ERR_ID，并将 Out 保持为 0	
输出	Out	_iq24	转矩信号		PU
结果	InertiaEst	_iq24	估算出的惯性		PU · s^2 / pu
	FrictionEst	_iq24	估算出的摩擦系数		PU · s / pu
信息	STATUS	ST_VelIdStatus_e	状态	{ST_VEL_ID_IDLE, ST_VEL_ID_INIT, ST_VEL_ID_BUSY }	
	DON	bool	识别完成指示器	false：运行或禁用；true：识别完成	
	ERR_ID	uint16_t	错误 ID	请参见表 7-1	

3.5.4.2 SpinTAC 速度识别运行函数

主函数为 STVELID_run(ST_VELID_Handle handle)，其中 handle 是具体 ST_VelId_t 对象的指针。 此函数需要以配置的采样速率调用。

void **STVELID_run**(ST_VELID_Handle handle)

参数：

编号	类型	参数	说明
1	ST_VELID_Handle	handle	ST_VelId_t 对象的指针

状态转换图如图 3-12 所示。

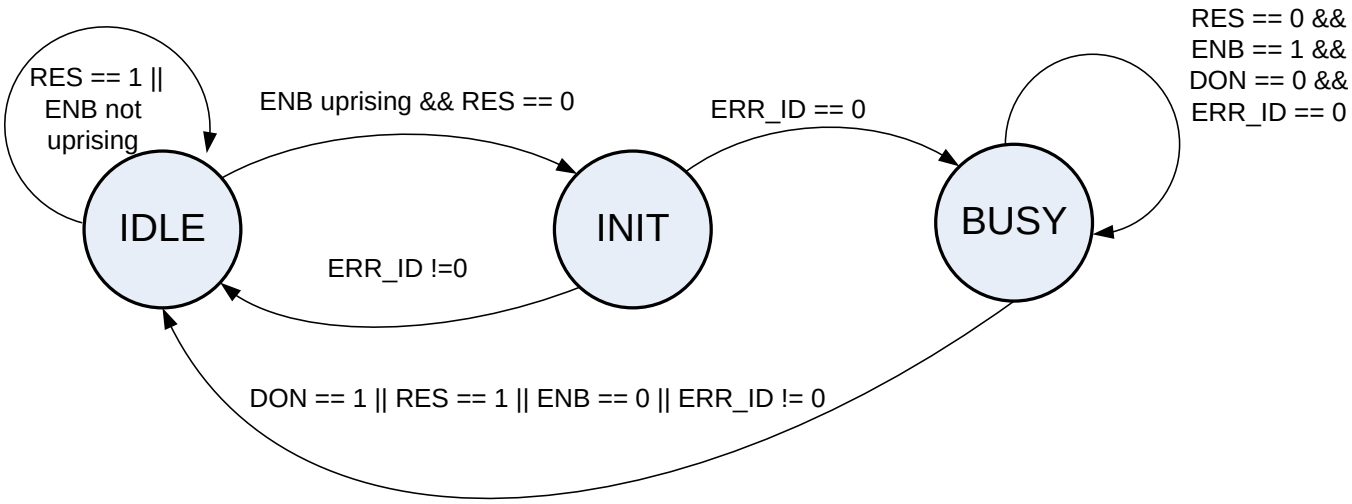


图 **3-12. SpinTAC** 速度识别状态转换图

表 3-13 说明了惯性识别的状态转换。

表 **3-13. SpinTAC** 速度识别状态转换

起始状态	结束状态	转换条件	操作
IDLE			1. 设置 ENB = false； 2. 将 Out 保持为 0 参数验证 1. 验证配置参数，包括 cfg.VelMax、cfg.OutMax、cfg.OutMin、cfg.LpfTime_tick、cfg.TimeOut_sec 和 cfg.T_sec，如果任一检查的变量无效，ERR_ID 都将为非零值；
	INIT	RES == false 并且 ENB 在上升沿并且 ERR_ID == 0	
INIT			参数验证 1. 验证参数，包括 cfg.VelPos、cfg.OutPos、cfg.OutNeg 和 cfg.RampTime_sec，如果任一检查的变量无效，ERR_ID 都将为非零值；
	IDLE	ERR_ID != 0	设置 ENB = false
	BUSY	ERR_ID == 0	

表 3-13. SpinTAC 速度识别状态转换 (continued)

起始状态	结束状态	转换条件	操作
BUSY			生成转矩系统配置并监视速度反馈 1. 在每个采样时间生成转矩系统配置并监视速度反馈； 2. 如果系统配置完成，将 DON 设置为 true；如果系统配置超时，ERR_ID 会设置为 2004； 3. 如果估算出的惯性为非正值，ERR_ID 将设置为 2003； 4. 如果电机在测试期间停止，ERR_ID 将设置为 2006；
	IDLE	RES == true 或 ENB == false 或 DON == true 或 ERR_ID != 0	1. 如果 RES == true 或 ENB == FALSE，Out 将平稳逼近零，等待几秒后稳定。 ERR_ID 将设置为 2005。 设置 ENB = false

3.5.5 SpinTAC 位置转换

使用反馈编码器的系统要求使用 SpinTAC 位置转换。 它根据位置编码器反馈计算位置和速度信号。 当使用编码器向 FOC 提供电角时，使用该组件为 SpinTAC 速度识别和 SpinTAC 速度控制提供速度反馈信号 [pu/s]。 所有位置解决方案也可使用该组件将编码器电角转换为机械角。 使用 FAST 无传感器估算器的系统不需要该组件。 SpinTAC 位置转换还将为交流感应电机提供转差速度的估算值。

3.5.5.1 SpinTAC 位置转换接口

SpinTAC 位置转换的接口和函数如图 3-13 所示，表 3-14 对此进行了说明。

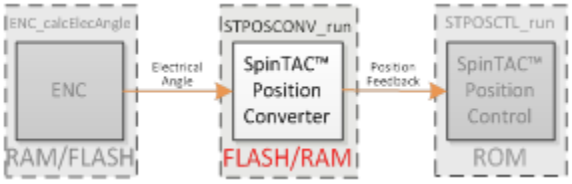


图 3-13. SpinTAC 位置转换接口

表 3-14. SpinTAC 位置转换接口和参数

信号类型	结构成员名称	数据类型	说明	值范围	单位
配置	cfg.T_sec	_iq24	采样时间	(0 , 0.01]	s
	cfg.ROMax_erev	_iq24	电气旋转 [ERev] 的上限	[0 , 16]	ERev
	cfg.ROMin_erev	_iq24	电气旋转 [ERev] 的下限	[-16, 0]	ERev
	cfg.erev_TO_pu_ps	_iq24	电气旋转 [ERev] 至速度用户单位 [pu/s] 的转换比	(0 , 0.01]	pu / s / ERev
	cfg.PolePairs	_iq24	机械旋转至电气旋转 [极对数] 的转换比	[1, 32]	
	cfg.ROMax_mrev	_iq24	位置翻转界限	[2, 100]	MRev
	cfg.LpfTime_tick	int16	低通滤波器时间常数 [ISR 节拍数]	[1 , 100]	采样计数
	cfg.SampleTimeOverTimeConst	_iq24	转差补偿器中使用的标量值（仅限 ACIM）	[0, 128.0)	
	cfg.OneOverFreqTimeConst	_iq24	转差补偿器中使用的标量值（仅限 ACIM）	[0, 128.0)	
输入	Pos_erev	_iq24	锯齿电角信号	[cfg.ROMin_erev, cfg.ROMax_erev)	ERev
	Id	_iq24	Id 电流反馈（仅限 ACIM）	[-1.0, 1.0]	PU
	Iq	_iq24	Iq 电流反馈（仅限 ACIM）	[-1.0, 1.0]	PU

表 3-14. SpinTAC 位置转换接口和参数 (continued)

信号类型	结构成员名称	数据类型	说明	值范围	单位
控制	ENB	bool	使能位	false：禁用；true：使能	
	RES	bool	复位位	false：不复位；true：复位	
输出	Vel	_iq24	计算出的速度（用户单位，未滤波）	[-1, 1]	pu / s
	VelLpf	_iq24	滤波后的速度（用户单位）	[-1, 1]	pu / s
	Pos_mrev	_iq24	锯齿机械角信号	[-cfg.ROMax_mrev, cfg.ROMax_mrev]	MRev
	PosROCounts	int32_t	位置翻转计数		
	SlipVel	_iq24	磁转差速度（仅限 ACIM）	[-1.0, 1.0]	ERev / s
信息	STATUS	ST_PosConvStatus_e	状态信息	{ST_POS_CONV_IDLE, ST_POS_CONV_INIT, ST_POS_CONV_BUSY}	
	ERR_ID	uint16_t	错误代码	请参见表 18-2	

3.5.5.2 SpinTAC 位置转换运行函数

主函数为 STPOSCONV_run(ST_POSCONV_Handle handle)，其中 handle 是具体 ST_PosConv_t 对象的指针，该句柄需要由初始化函数 STPOSCONV_init 建立。

void **STPOSONV_run**(ST_POSCONV_Handle handle)

参数：

编号	类型	参数	说明
1	ST_POSCONV_Handle	handle	ST_PosConv_t 对象的指针

状态转换图如图 3-14 所示。

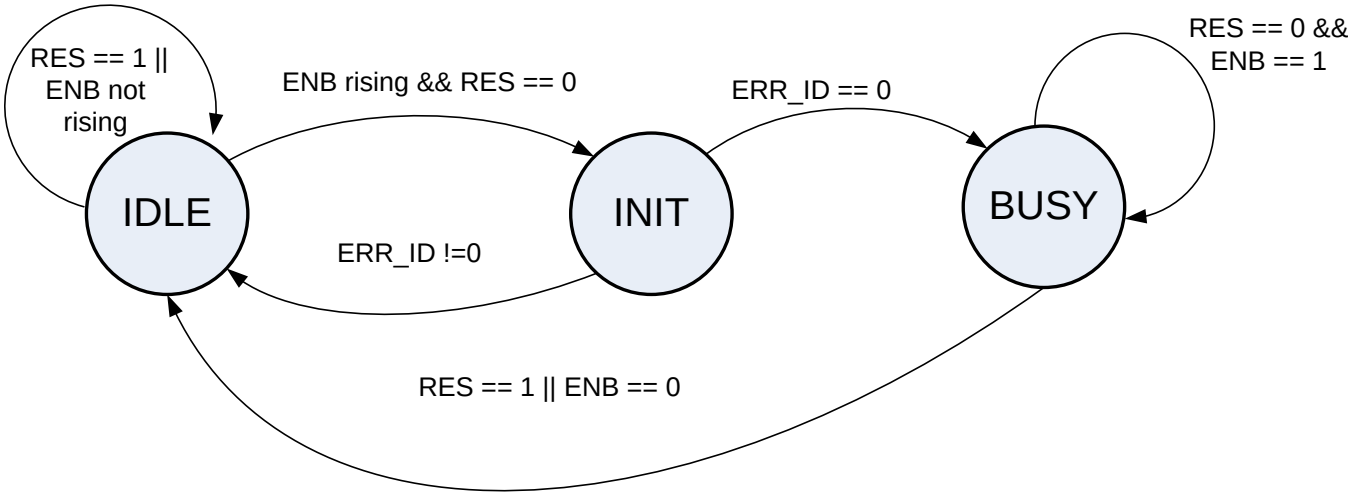


图 3-14. SpinTAC 位置转换状态转换图

表 3-15 说明了 SpinTAC 位置转换的状态转换。

表 3-15. SpinTAC 位置转换状态转换

起始状态	结束状态	转换条件	操作
IDLE			1. 设置 ENB = false； 2. 保持 Vel、VelLpf、Pos_mrev 和 PosROCounts 为 0
	INIT	RES == false 并且 ENB 在上升沿	
INIT			参数验证 1. 验证配置参数，包括 cfg.ROMax_erev、cfg.ROMin_erev、cfg.erev_TO_pu_ps、cfg.PolePairs、cfg.ROMax_mrev、cfg.LpfTime_ticks 和 cfg.T_sec，如果任一检查的变量无效，ERR_ID 都将为非零值；
	IDLE	ERR_ID != 0	设置 ENB = false
	BUSY	ERR_ID == 0	
BUSY			计算锯齿位置信号 ([MRev]) 和速度信号 ([pu/s])
	IDLE	RES == true 或 ENB == false	设置 ENB = false

3.5.6 *SpinTAC 位置控制*

SpinTAC 位置控制是控制运动系统位置环路和速度环路的级联控制器。 开发人员使用一个调节参数就可以调节两个环路。 与 SpinTAC 速度控制类似，SpinTAC 位置控制可抑制外部干扰，外部干扰由控制公式中未知的非线性项表示。 像 SpinTAC 速度控制一样，该控制器也通过参数化方法简化了调节过程，该方法允许使用单个调节参数（带宽）对动态系统进行高性能控制。 该单个参数用于调节 SpinTAC 位置控制的速度环路和位置环路。

3.5.6.1 SpinTAC 位置控制接口

SpinTAC 位置控制的接口和函数如图 3-15 所示。

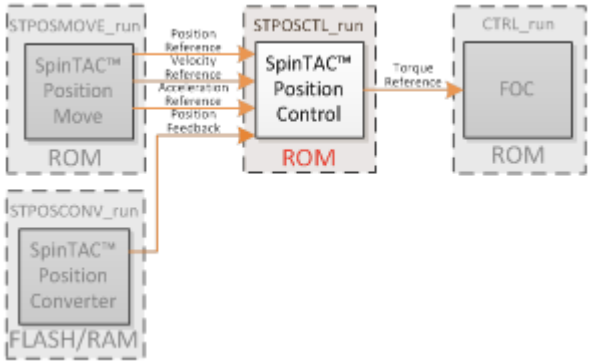


图 3-15. SpinTAC 位置控制接口

注意：在 SpinTAC 位置控制中，惯性 (cfg.Inertia) 可通过 SpinTAC 速度识别获取。 如果主要干扰为斜坡型，可启用开关 cfg.RampDist。 唯一的控制器调节参数是带宽 BwScale。

表 3-16 列出了 SpinTAC 位置控制的接口参数。

表 **3-16. SpinTAC** 位置控制接口参数

信号类型	结构成员名称	数据类型	说明	值范围	单位
配置	cfg.Axis	ST_Axis_e	SpinTAC 控制轴 ID	{ST_AXIS0, ST_AXIS1}	
	cfg.T_sec	_iq24	采样时间	(0 , 0.01]	s
	cfg.InertiaMax	_iq24	最大系统惯性	(0, 100]	PU · s2 / pu
	cfg.InertiaMin	_iq24	最小系统惯性	(0, cfg.InertiaMax]	PU · s2 / pu
	cfg.OutMax	_iq24	最大控制信号	(0 , 1]	PU
	cfg.OutMin	_iq24	最小控制信号	[-1, 0)	PU
	cfg.VelMax	_iq24	最大速度基准信号	(0 , 1]	pu / s
	cfg.ROMax_mrev	_iq24	位置翻转界限	[2, 100]	MRev
	cfg.mrev_TO_pu	_iq24	机械旋转 [MRev] 至 [pu] 的转换比	[0.002, 1]	pu / MRev
	cfg.PosErrMax_mrev	_iq24	最大允许位置误差	(0, cfg.ROMax_mrev /2]	MRev
	cfg.RampDist	bool	抑制斜坡干扰	false：禁用；true：使能	
	cfg.BwScaleMax	_iq24	带宽上限	[0.01, min(50, 0.005/ cfg.T_sec)]	
	cfg.BwScaleMin	_iq24	带宽下限	[0, cfg.BwScaleMax]	
	cfg.FiltEn	bool	启用反馈低通滤波器	false：禁用；true：使能	
输入	PosRef_mrev	_iq24	位置基准信号	[-cfg.ROMax_mrev, cfg.ROMax_mrev]	MRev
	VelRef	_iq24	速度基准信号	[-cfg.VelMax , cfg.VelMax]	pu / s
	AccRef	_iq24	加速度基准信号	[cfg.OutMin, cfg.OutMax]	pu / s2
	PosFdb_mrev	_iq24	位置反馈信号	[-cfg.ROMax_mrev, cfg.ROMax_mrev)	MRev
	Inertia	_iq24	系统惯性	[cfg.InertiaMin, cfg.InertiaMax]	PU · s2 / pu
	Friction	_iq24	系统摩擦	[0, 5]	PU · s2 / pu
调节	BwScale	_iq24	控制器带宽范围	[cfg.BwScaleMin, cfg.BwScaleMax]	
控制	ENB	bool	使能位	false：禁用；true：使能	
	RES	bool	复位位	false：不复位；true：复位 ERR_ID，并将 Out 保持为 0	
输出	Out	_iq24	控制输出	[cfg.OutMin, cfg.OutMax]	PU
信息	Bw_radps	_iq20	控制器带宽		rad/s
	STATUS	ST_CtlStatus_e	状态信息	{ST_CTL_IDLE, ST_CTL_INIT, ST_CTL_CONF, ST_CTL_BUSY}	
	ERR_ID	uint16_t	错误代码	请参见表 12-3	
	PosErr_mrev	_iq24	位置误差		MRev

3.5.6.2 SpinTAC 位置控制运行函数

主函数为 STPOSCTL_run(ST_POSCTL_Handle handle)，其中 handle 是具体 ST_PosCtl_t 对象的指针，该句柄需要由初始化函数 STPOSCTL_init 建立。

void **STPOSCTL_run**(ST_POSCTL_Handle handle)

参数：

编号	类型	参数	说明
1	ST_POSCTL_Handle	Handle	ST_PosCtl_t 对象的指针

SpinTAC 位置控制状态转换图如图 3-16 所示。

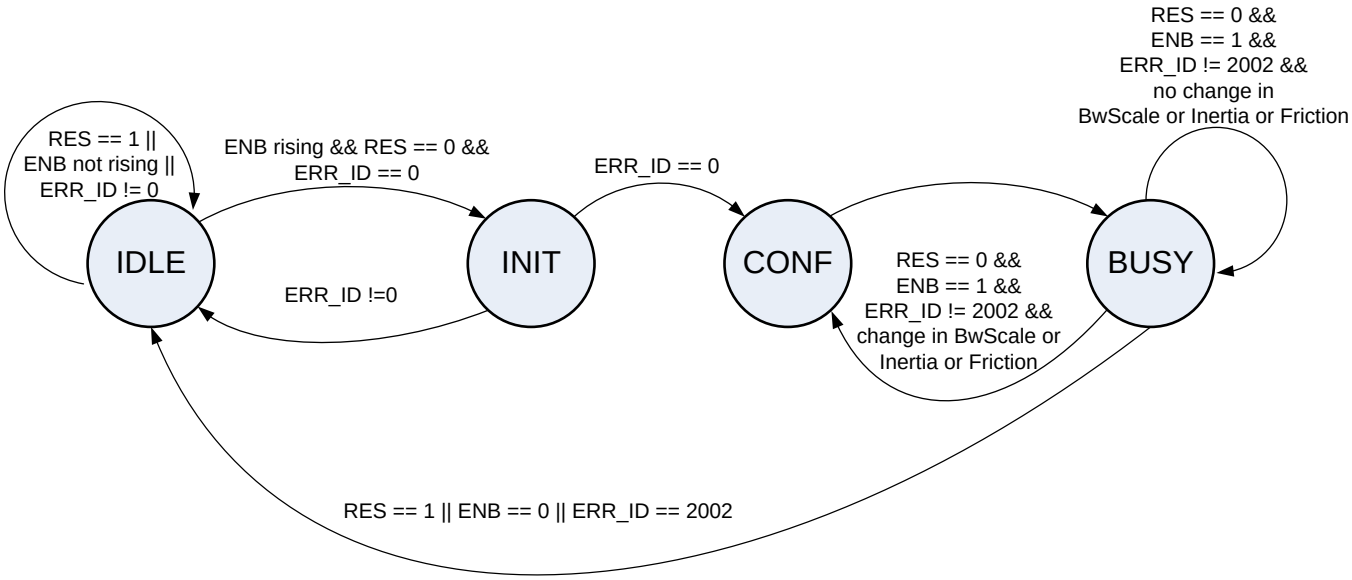


图 3-16. SpinTAC 位置控制状态转换图

请注意，在图 3-16 中，IDLE 到 INIT、INIT 到 CONF 以及 CONF 到 BUSY 的状态转换在一个采样时间内发生。 因此，控制器启用后会直接在该采样时间内工作。

表 3-17 说明了 SpinTAC 位置控制的状态转换。

表 3-17. SpinTAC 位置控制状态转换

起始状态	结束状态	转换条件	操作
IDLE			1. 设置 ENB = false； 2. 将 Out 保持为零
	INIT	RES == false 并且 ENB 在上升沿并且 ERR_ID == 0	
INIT			参数验证 1. 验证配置参数，包括 cfg.VelMax、cfg.OutMax、cfg.OutMin、cfg.ROMax_mrev、cfg.mrev_TO_pu、cfg.PosErrMax_mrev、cfg.InertiaMax、cfg.InertiaMin、cfg.BwScaleMax、cfg.BwScaleMin、cfg.DistType 和 cfg.T_sec。 如果任一检查的变量无效，ERR_ID 都将为非零值
	IDLE	ERR_ID != 0	设置 ENB = false
	CONF	ERR_ID == 0	
CONF	BUSY		Inertia 和 BwScale 饱和至指定界限。 如果发生饱和，ERR_ID 将为 1012、1013、1014 或 1015。
BUSY			1. 生成控制信号。 2. 如果 PosErr_mrev 超过 cfg.PosErrMax_mrev，则 ERR_ID =2002。
	IDLE	RES == true 或 ENB == false 或 ERR_ID == 2002	设置 ENB = false
	CONF	RES == false 并且 ENB == true 并且 BwScale 或 Inertia 发生变化并且 ERR_ID != 2002	

3.5.7 SpinTAC 位置移动

SpinTAC 位置移动提供两种模式：速度控制的位置系统配置和位置控制的位置系统配置。 前一种模式适合在不同速度间切换的加速信号和系统配置。 后一种模式最适合点对点位置移动。 cfg.ProfileType 用于在两种模式之间切换，只能在 ENB 信号的上升沿进行调整。 模式切换需要 cfg.VelStart 设置为零并且 STATUS 处于 IDLE 状态。

3.5.7.1 位置移动接口

SpinTAC 位置移动的接口和函数如图 3-17 所示。

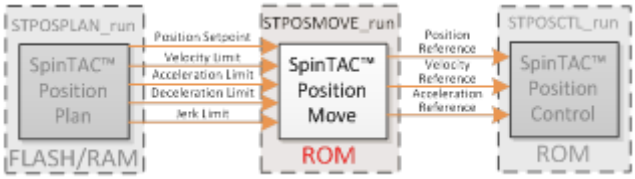


图 3-17. SpinTAC 位置移动接口

表 3-18 列出了 SpinTAC 位置移动的接口参数。

表 3-18. SpinTAC 位置移动接口

信号类型	结构成员名称	数据类型	说明	值范围	单位
配置	cfg.Axis	ST_Axis_e	SpinTAC 移动轴 ID	{ST_AXIS0, ST_AXIS1}	
	cfg.ProfileType	ST_PosMoveProfiletype_e	系统配置模式（速度控制或位置控制）	{ST_POS_MOVE_VEL_TYPE, ST_POS_MOVE_POS_TYPE}	
	cfg.CurveType	ST_MoveCurveType_e	曲线类型	{ST_MOVE_CUR_TRAP, ST_MOVE_CUR_SCRV, ST_MOVE_CUR_STCRV}	
	cfg.T_sec	_iq24	采样时间	(0 , 0.01]	s
	cfg.ROMax_mrev	_iq24	位置翻转界限	[2, 100]	MRev
	cfg.mrev_TO_pu	_iq24	机械旋转至 pu 的转换比	[0.002, 1]	Pu / MRev
	cfg.VelMax	_iq24	系统的最大速度	(0 , 1]	pu / s
	cfg.AccMax	_iq24	系统的最大加速度	[0.001 , 120]	pu / s2
	cfg.DecMax	_iq24	系统的最大减速度	[0.001 , 120]	pu / s2
	cfg.JrkMax	_iq20	系统的最大急动	[0.0005, 2000]	pu / s3
	cfg.VelStart	_iq24	速度起始值	[-cfg.VelMax , cfg.VelMax]	pu / s
	cfg.PosStart_mrev	_iq24	位置起始值	[-cfg.ROMax, cfg.ROMax]	MRev
	cfg.IgnoreLimitErrors	bool	如果系统配置界限设置在有效值范围外，则使系统配置限制饱和至有效值范围内	false：提供错误代码，不生成系统配置；true：使系统配置限制饱和，生成系统配置	
消息	msg.ProTime_tick	uint32_t	系统配置时间（以 1 百万计数为单位）		采样计数
	msg.ProTime_mtick	uint32_t	系统配置时间（以百万计数为单位）		百万采样计数
	msg.Vel	_iq24	系统配置的最大速度		pu / s
	msg.Acc	_iq24	系统配置的最大加速度		pu / s2
	msg.Dec	_iq24	系统配置的最大减速度		pu / s2
	msg.Jrk	_iq20	系统配置的最大急动		pu / s3
	msg.PosStepMax_mrev	_iq24	最大位置阶跃		MRev

表 3-18. SpinTAC 位置移动接口 (continued)

信号类型	结构成员名称	数据类型	说明	值范围	单位
输入	PosStepInt_mrev	int32_t	位置阶跃整数值	[-2147483647, 2147483647]	MRev
	PosStepFrac_mrev	_iq24	位置阶跃分数值	(-1, 1)	MRev
	VelLim	_iq24	速度限制	(0 , cfg.VelMax]	pu / s
	AccLim	_iq24	加速度限制	[0.001 , cfg.AccMax]	pu / s2
	DecLim	_iq24	减速度限制	[0.001 , cfg.DecMax]	pu / s2
	JrkLim	_iq20	急动限制	[0.0005, cfg.JrkMax]	pu / s3
	VelEnd	_iq24	速度结束值	[-cfg.VelMax , cfg.VelMax]	pu / s
控制	ENB	bool	使能位	false：系统配置完成或被禁用；true：使能并运行	
	RES	bool	复位位	false：不复位；true：复位 ERR_ID，并将系统配置输出保持为之前的值	
	TST	bool	系统配置测试位	false：不测试；true：测试系统配置	
信息	STATUS	ST_MoveStatus_e	状态信息	{ST_MOVE_IDLE, ST_MOVE_INIT, ST_MOVE_CONF, ST_MOVE_BUSY, ST_MOVE_HALT}	
	DON	bool	系统配置完成指示器	false：正在运行；true：系统配置完成或空闲	
	ERR_ID	uint16_t	错误代码	请参见表 13-2	
输出	PosRollOver	int32_t	位置翻转计数		
	PosRef_mrev	_iq24	位置基准		MRev
	VelRef	_iq24	速度基准		pu / s
	AccRef	_iq24	加速度基准		pu / s2
	JrkRef	_iq20	急动基准		pu / s3

3.5.7.2 SpinTAC 位置移动运行函数

SpinTAC 位置移动函数为 STPOSMOVE_run(ST_POSMOVE_Handle handle)，其中 handle 是具体 ST_PosMove_t 对象的指针，该句柄需要由初始化函数 STPOSMOVE_init 建立。

void **STPOSMOVE_run**(ST_POSMOVE_Handle handle)

参数：

编号	类型	参数	说明
1	ST_POSMOVE_Handle	Handle	ST_PosMove_t 对象的指针

SpinTAC 位置移动状态转换图如图 3-18 所示。

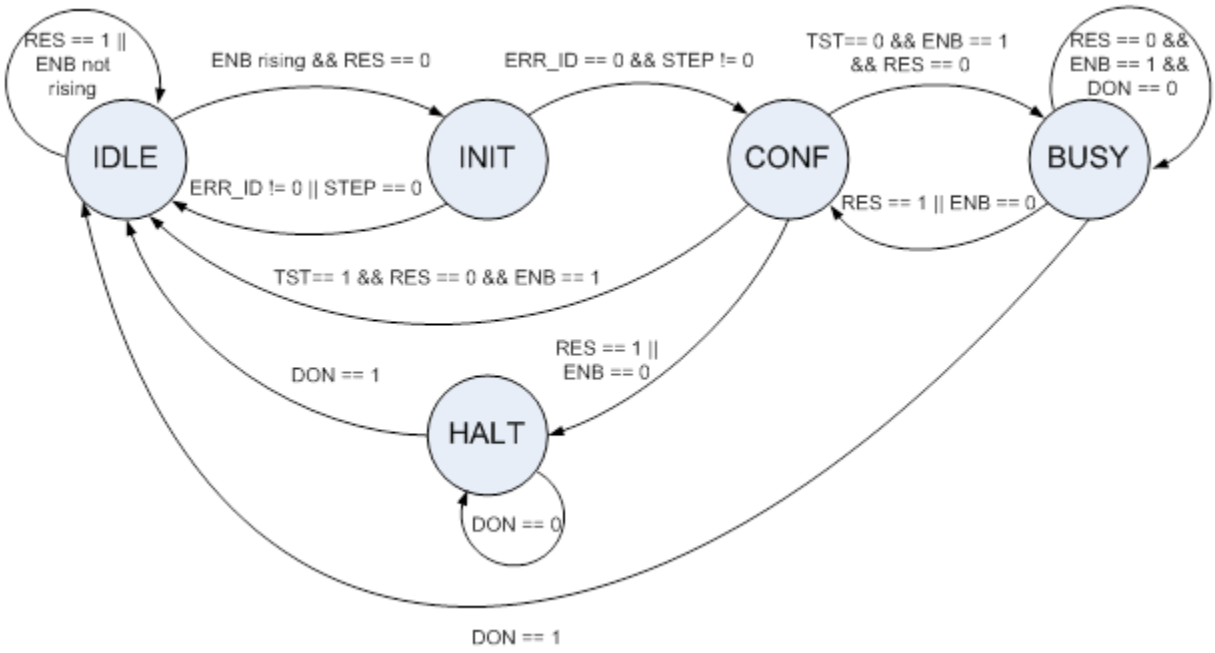


图 **3-18. SpinTAC** 位置移动状态转换图

请注意，在图 3-18 中，IDLE 到 INIT，再到 CONF 的状态转换在一个采样时间内发生。 因此，在启用系统配置的相同采样时间内生成系统配置。

Changed 13.1 节，生成 InstaSPIN-MOTION 系统配置 说明了 SpinTAC 位置移动的状态。

表 **3-19.** 位置移动状态转换

起始状态	结束状态	转换条件(1)(2)(3)(4)	操作
IDLE			保持 IDLE 状态 1. 设置 ENB = false； 2. 设置 DON = false； 3. 保持 PosRef、VelRef、AccRef 和 JrkRef 的值（cfg.PosStart_mrev 的值只能在 IDLE 状态下设置。）
	INIT	RES == false 并且 ENB 在上升沿	
INIT			参数验证 1. 验证配置参数，包括 cfg.PosStart_mrev、PosStepInt_mrev、PosStepFrac_mrev、cfg.VelStart、VelEnd、cfg.VelMax、cfg.AccMax、cfg.DecMax、cfg.JrkMax、AccLim、DecLim、JrkLim、cfg.CurveType、cfg.ProfileType 和 cfg.T_sec。 如果任一检查的变量无效，ERR_ID 都将为非零值。 2. 计算系统配置阶跃 STEP（如果 cfg.ProfileType == ST_POS_MOVE_VEL_TYPE，则为速度阶跃；如果 cfg.ProfileType == ST_POS_MOVE_POS_TYPE，则为位置阶跃）。
	IDLE	ERR_ID != 0 或 STEP == 0	设置 ENB = false
	CONF	ERR_ID == 0 并且 STEP != 0	

表 3-19. 位置移动状态转换 (continued)

起始状态	结束状态	转换条件(1)(2)(3)(4)	操作
CONF			根据配置的参数确定系统配置
	IDLE	TST== true 并且 ENB == true 并且 RES == false	在测试模式下，不生成系统配置 1. 如果 cfg.ProfileType == ST_POS_MOVE_POS_TYPE，将 PosStepInt_mrev 和 PosStepFrac_mrev 设置为 0；如果 cfg.ProfileType == ST_POS_MOVE_VEL_TYPE，将 VelEnd 设置回 cfg.VelStart 的值
	BUSY	TST== false 并且 ENB == true 并且 RES == false	
	HALT	RES == true 或 ENB == false	
BUSY			生成系统配置 1. 在每个采样时间内更新 PosRef_mrev、VelRef、AccRef、JrkRef 基准； 2. 如果系统配置完成，DON = true
	IDLE	RES == false 并且 ENB == true 并且 DON == true	设置 ENB = false
	CONF	RES == true 或 ENB == false	配置减速度系统配置 1. 设置 cfg.ProfileType = ST_POS_MOVE_VEL_TYPE，VelEnd = 0，AccLim = cfg.AccMax，cfg.DecMax，JrkLim = cfg.JrkMax
HALT			生成减速度系统配置 1. 在每个采样时间内更新 PosRef、VelRef、AccRef、JrkRef 基准； 2. 如果减速度系统配置完成，DON = True
	IDLE	DON == true	设置 ENB = false
	HALT	DON == false	

(1) RES 信号提供了复位 SpinTAC 位置移动的功能。

(2) 如果 RES 设置为 true，ENB 将被设置为 false。 当前任何错误都将被丢弃。 SpinTAC 位置移动随后将生成减速度系统配置以停止轴的所有运动。

(3) ENB 信号提供了 SpinTAC 位置移动的启动信号。 仅当 RES 为 false 时，ENB 信号才起作用。

(4) TST 位的目的是在不实际生成轨迹的情况下提供系统配置信息。 该信息包括系统配置时间以及实际的最大速度、加速度和急动。 在 INIT 状态下，ENB 为上升沿时由相关函数接收 TST 信号。 如果 TST 为 true，则运行在测试模式。 在测试模式下，系统配置输出 PosRef_mrev 将保持 cfg.PosStart_mrev 的值；AccRef 将为 0，不受 PosStepInt_mrev 和 PosStepFrac_mrev 影响。 测试后，将输出系统配置信息（msg.ProTime_tick、msg.ProTime_mtick、msg.Vel、msg.Acc 和 msg.Jrk），DON 将设置为 true，ENB 将设置为 false。

3.5.8 SpinTAC 位置规划

SpinTAC 位置规划的功能是设置和运行由用户应用确定的位置序列。

3.5.8.1 位置规划接口

SpinTAC 位置规划的接口和函数如图 3-19 所示。

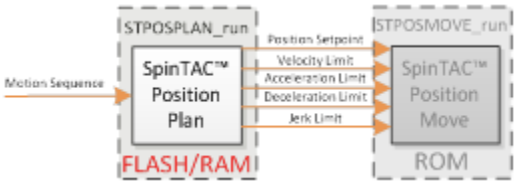


图 3-19. SpinTAC 位置规划接口

表 3-20 列出了 SpinTAC 位置规划的接口参数。

表 3-20. SpinTAC 位置规划接口

信号类型	结构成员名称	数据类型	说明	值范围	单位
控制	ENB	bool	使能位	false：禁用；true：使能	
	RES	bool	复位位	false：不复位；true：复位	
输出	PosStepInt_mrev	uint32_t	当前位置阶跃命令整数部分	[-2147483647, 2147483647]	MRev
	PosStepFrac_mrev	_iq24	当前位置阶跃命令分数部分	(-1, 1)	MRev
	VelLim	_iq24	当前速度限制	(0 , VelMax]	pu / s
	AccLim	_iq24	当前加速度限制	[0.001 , AccMax]	pu / s2
	DecLim	_iq24	当前减速度限制	[0.001 , DecMax]	pu / s2
	JrkLim	_iq20	当前急动限制	[0.0005, JrkMax]	pu / s3
	Timer_tick	uint32_t	当前状态剩余的时间		采样计数
信息	STATUS	ST_PlanStatus_e	状态信息	{ST_PLAN_IDLE, ST_PLAN_INIT, ST_PLAN_BUSY, ST_PLAN_HALT, ST_PLAN_WAIT}	
	CurState	unit16_t	当前状态索引	[0, StateNum)	
	CurTran	unit16_t	当前转换索引	[0, TranNum)	
	FsmState	ST_PlanFsmState_e	一种状态，用于指示是处于转换中，还是处于某个状态，还是正在等待转换	{ST_FSM _STATE_STAY, ST_FSM _STATE_COND, ST_FSM_STATE_TRAN }	
	Timer_tick	uint32_t	当前状态剩余的时间		采样计数
	ERR_ID	uint16_t	错误代码	请参见表 13-6	
	DON	bool	规划完成指示器	false：未完成；true：已完成	
	CfgError.ERR_idx	uint16_t	发生错误的索引		
	CfgError.ERR_code	uint16_t	导致错误的条件	请参见表 13-6	

3.5.8.2 SpinTAC 位置规划主函数

主函数为 STPOSPLAN_run(ST_POSPLAN_Handle handle)，其中 handle 是具体 ST_PosPlan_t 对象的指针，该句柄需要由初始化函数 ST_POSPLAN_init 建立。 可从项目的主循环调用此函数。

void **STPOSPLAN_run**(ST_POSPLAN_Handle handle)

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 对象的指针

ISR 函数为 STPOSPLAN_runTick(ST_POSPLAN_Handle handle)，其中 handle 是具体 ST_PosPlan_t 对象的指针。 此函数处理 ST_PosPlan 的时间关键代码。 必须在项目的主 ISR 中调用此函数。

void **STPOSPLAN_runTick**(ST_POSPLAN_Handle handle)

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 对象的指针

ST_PosPlan 的状态转换图与 ST_VelPlan 的状态转换图相同，如图 3-10 所示。

表 3-21 说明了 ST_PosPlan 的状态。

表 3-21. SpinTAC 位置规划状态转换

起始状态	结束状态	转换条件(1)(2)(3)(4)	操作
IDLE			保持 IDLE 状态 1. 设置 ENB = false； 2. 使 PosStepInt = 0，PosStepFrac=0，并保持 VelLim、AccLim 和 JrkLim 的值
	INIT	RES == false 并且 ENB 在上升沿	
INIT	BUSY		参数验证 1. 复位内部状态； 2. 输入状态 0 并执行为输入状态 0 定义的操作。
BUSY			操作规划
	IDLE	RES == false 并且 ENB == True 并且 DON == true	设置 ENB = false
	HALT	RES == true 或 ENB == false	加载暂停状态系统配置
HALT	IDLE	HALT 状态定时器启动并且 RES == true	加载状态 0 的配置
	WAIT	HALT 状态定时器启动并且 RES == false	
WAIT	IDLE	RES == true	加载状态 0 的配置
	HALT	ENB == true	加载上个状态的配置

(1) RES 信号提供了复位 SpinTAC 位置规划的功能。

(2) 当 RES 为 false 时，ENB 信号控制 SpinTAC 位置规划的运行。

(3) 如果在 SpinTAC 位置规划运行时将 ENB 设置为 false，SpinTAC 位置规划将发出位置阶跃和 HALT 状态的限制。 完成单位系统配置后，SpinTAC 位置规划将进入 WAIT 状态，并且只能在 ENB 设置为 true 后继续规划。
如果 RES 设置为 true，ENB 将被设置为 false。 SpinTAC 位置规划随后将发出位置阶跃和状态 0 的限制，并进入 IDLE 状态。

(4) 实际上，ENB 功能起到暂停/停止按钮的作用，而 RES 功能起到停止按钮的作用。

表 3-22 列出了可用于执行操作（如设置、获取、添加和删除“SpinTAC 位置规划”的配置和运行时参数）的函数。 3.5.9 节 更详细地介绍了这些函数。

表 3-22. SpinTAC 位置规划附加函数

函数组	函数名称	说明
初始化	STPOSPLAN_init	初始化 SpinTAC 位置规划
配置	STPOSPLAN_setCfgArray	设置 SpinTAC 位置规划用来存储配置的数组
	STPOSPLAN_setCfg	设置系统参数和保护参数
	STPOSPLAN_setCfgHaltState	设置 HALT 状态的参数
	STPOSPLAN_addCfgState	添加新状态
	STPOSPLAN_addCfgVar	添加新变量
	STPOSPLAN_addCfgCond	添加比较变量和静态值的新条件
	STPOSPLAN_addCfgVarCond	添加比较两个变量的新条件
	STPOSPLAN_addCfgTran	添加新转换
	STPOSPLAN_addCfgAct	添加新操作
运行时	STPOSPLAN_run	运行 SpinTAC 位置规划。 可从主循环运行。
	STPOSPLAN_runTick	运行 SpinTAC 位置规划定时器。 必须从主 ISR 运行
	STPOSPLAN_setVar	设置运行期间的变量值
	STPOSPLAN_getVar	获取运行期间的变量值
	STPOSPLAN_reset	复位 SpinTAC 位置规划和配置
	STPOSPLAN_setUnitProfDone	设置当前运行的系统配置是否完成

表 3-22. SpinTAC 位置规划附加函数 (continued)

函数组	函数名称	说明
规划修改和调试函数（提供在运行时修改规划的功能）	STPOSPLAN_getCfgStateNum	获取已配置状态数
	STPOSPLAN_getCfgVarNum	获取已配置变量数
	STPOSPLAN_getCfgCondNum	获取已配置条件数
	STPOSPLAN_getCfgTranNum	获取已配置转换数
	STPOSPLAN_getCfgActNum	获取已配置操作数
	STPOSPLAN_getCfg	获取系统和保护参数
	STPOSPLAN_getCfgHaltState	获取 HALT 状态的参数
	每个带 -add 后缀的函数有其它三个函数：-del、-set 和 -get，分别用于删除项目、设置项目和获取项目。 这些函数允许在线修改 SpinTAC 位置规划。	

3.5.9 SpinTAC 函数

下面列出了常用的 SpinTAC 函数。 本节作为参考提供，在实现某个组件之前，请仔细阅读本文档的相应部分。

void **STVELCTL_run**(ST_VELCTL_Handle)

函数： 此函数运行 SpinTAC 速度控制。

参数：

编号	类型	参数	说明
1	ST_VELCTL_Handle	handle	ST_VelCtl_t 数据类型的句柄

void **STVELMOVE_run**(ST_VELMOVE_Handle)

函数： 此函数运行 SpinTAC 速度移动。

参数：

编号	类型	参数	说明
1	ST_VELMOVE_Handle	handle	ST_VelMove_t 数据类型的句柄

void **STVELID_run**(ST_VELID_Handle)

函数： 此函数运行 SpinTAC 速度识别。

参数：

编号	类型	参数	说明
1	ST_VELID_Handle	handle	ST_VelId_t 数据类型的句柄

void **STVELPLAN_run**(ST_VELPLAN_Handle)

函数： 此函数运行 SpinTAC 速度规划。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄

void **STVELPLAN_runTick**(ST_VELPLAN_Handle)

函数： 此函数运行 SpinTAC 速度规划定时器。 必须从 ISR 运行。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄

void **STVELPLAN_addCfgAct**(ST_VELPLAN_Handle, uint16_t, ST_PlanCond_e , uint16_t , uint16_t, uint16_t, ST_PlanActOptn_e, _iq24, ST_PlanActTrgr_e)

函数： 此函数向 SpinTAC 速度规划添加操作。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	uint16_t	StateIdx	将执行操作的状态的索引
3	ST_PlanCond_e	AndOr	ST_COND_NC：无条件执行； ST_COND_FC：满足第一个条件时执行； ST_COND_AND：两个条件都满足时执行； ST_COND_OR：满足任一个条件时执行；
4	uint16_t	CondIdx1	第一个条件的索引
5	uint16_t	CondIdx2	第二个条件的索引
6	uint16_t	VarIdx	要操作的变量的索引
7	ST_PlanActOptn_e	Opt	ST_ACT_EQ：设置变量的值 ST_ACT_QDD：向变量添加值
8	_iq24	Value	要设置到变量或添加到变量的值
9	ST_PlanActTrgr_e	EnterExit	ST_ACT_ENTR：进入状态时执行操作 ST_ACT_EXIT：退出状态时执行操作

void **STVELPLAN_setCfgArray**(ST_VELPLAN_Handle, uint32_t *, const size_t, uint16_t, uint16_t, uint16_t, uint16_t, uint16_t)

函数： 此函数向 SpinTAC 速度规划添加操作。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	uint32_t *	cfgArray	用于规划配置的 cfgArray 的指针。
3	const size_t	numBytes	分配给 cfgArray 数组的字节数。 通过调用 sizeof(cfgArray) 获取字节数
4	uint16_t	MaxActNum	操作数
5	uint16_t	MaxCondNum	条件数
6	uint16_t	MaxVarNum	变量数
7	uint16_t	MaxTranNum	转换数
8	uint16_t	MaxStateNum	状态数

ST_VELPLAN_Handle **STVELPLAN_init** (void *, const size_t)

函数： 此函数初始化 SpinTAC 速度规划。

参数：

编号	类型	参数	说明
1	void *	pMemory	指向 ST_VelPlan_t 对象的内存的指针
2	const size_t	numBytes	分配给 ST_VelPlan_t 对象的字节数

返回：

ST_VelPlan_t 对象句柄

void **STVELPLAN_addCfgCond**(ST_VELPLAN_Handle, uint16_t, ST_PlanComp_e, _iq24, _iq24)

函数： 此函数向 SpinTAC 速度规划添加条件。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	uint16_t	VarIdx	要比较的变量的索引
3	ST_PlanComp_e	Comp	ST _COMP_NA：未定义； ST _COMP_EQ：等于 Value1； ST _COMP_NEQ：不等于 Value1； ST__COMP_GT：大于 Value1； ST_COMP_EGT：等于或大于 Value1； ST_COMP_LW：小于 Value1； ST_COMP_ELW：等于或小于； ST_COMP_In：属于 (Value1, Value2) 的范围； ST_COMP_Ein：属于 [Value1, Value2) 的范围； ST_COMP_InE：属于 (Value1, Value2] 的范围； ST_COMP_EinE：属于 [Value1, Value2] 的范围； ST_COMP_Out：不属于 [Value1, Value2] 的范围； ST_COMP_Eout：不属于 (Value1, Value2] 的范围； ST_COMP_OutE：不属于 [Value1, Value2) 的范围； ST_COMP_EoutE：不属于 (Value1, Value2) 的范围
4	_iq24	Value1	第一个值
5	_iq24	Value2	第二个值

void **STVELPLAN_addCfgVarCond** ST_VELPLAN_Handle, uint16_t, uint16_t, ST_PlanComp_e)

函数： 此函数添加将两个变量与 SpinTAC 速度规划进行比较的条件。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	uint16_t	VarIdx1	要比较的第一个变量的索引
3	uint16_t	VarIdx2	要比较的第二个变量的索引
4	ST_PlanComp_e	Comp	ST _COMP_NA：未定义； ST _COMP_EQ：等于 Value1； ST _COMP_NEQ：不等于 Value1； ST__COMP_GT：大于 Value1； ST_COMP_EGT：等于或大于 Value1； ST_COMP_LW：小于 Value1； ST_COMP_ELW：等于或小于；

void **STVELPLAN_setCfgHaltState**(ST_VELPLAN_Handle, _iq24, _iq24, _iq20, int32_t)

函数： 此函数向 SpinTAC 速度规划添加状态。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	_iq24	VelEnd	HALT 状态的速度设定点
3	_iq24	AccLim	HALT 状态的加速度限制
4	_iq20	JrkLim	HALT 状态的急动限制
5	int32_t	Timer	用于指示在进一步操作前 HALT 状态保持时间的定时器 ([计数])

void **STVELPLAN_setCfg**(ST_VELPLAN_Handle, _iq24, _iq24, _iq24, _iq20, bool)

函数： 此函数设置 SpinTAC 速度规划系统配置。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	_iq24	T_sec	采样时间 ([s])
3	_iq24	VelMax	系统速度上限
4	_iq24	AccMax	系统加速度上限
5	_iq20	JrkMax	系统急动上限
6	bool	LoopENB	false：规划仅运行一次，然后回到 IDLE 状态 true：规划每次进入 IDLE 状态时都再次运行

void **STVELPLAN_addCfgState**(ST_VELPLAN_Handle, _iq24, int32_t)

函数： 此函数向 SpinTAC 速度规划添加状态。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	_iq24	VelEnd	状态的速度设定点
3	int32_t	Timer_tick	用于指示状态保持时间的定时器 ([计数])

void **STVELPLAN_addCfgTran**(ST_VELPLAN_Handle, uint16_t, uint16_t, ST_PlanCond_e, uint16_t, uint16_t, _iq24, _iq20)

函数： 此函数向 SpinTAC 速度规划添加事务。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	uint16_t	FromState	转换的起始状态的索引
3	uint16_t	ToState	转换的结束状态的索引
4	ST_PlanCond_e	AndOr	ST_COND_NC：无条件转换； ST_COND_FC：满足第一个条件时转换； ST_COND_AND：两个条件都满足时转换； ST_COND_OR：满足任一个条件时转换；
5	uint16_t	CondIdx1	第一个条件的索引

编号	类型	参数	说明
6	uint16_t	CondIdx2	第二个条件的索引
7	_iq24	AccLim	用于此转换的加速度限制
8	_iq20	JrkLim	用于此转换的急动限制

void **STVELPLAN_addCfgVar**(ST_VELPLAN_Handle, ST_PlanVar_e, _iq24)

函数： 此函数向 SpinTAC 速度规划添加变量。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	ST_PlanVar_e	Type	ST_VAR_INOUT：定时器类型，可在条件中使用 ST_VAR_IN：传感器类型，仅接收值，可在条件中使用 ST_VAR_OUT：传动器类型，仅发送值，不能在条件中使用
3	_iq24	Value	变量的初始值

void **STVELPLAN_setUnitProfDone**(ST_VELPLAN_Handle, bool)

函数： 此函数在当前运行的系统配置完成后通知 SpinTAC 速度规划。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	bool	ProDON	false：未完成；true：已完成

void **STVELPLAN_getVar**(ST_VELPLAN_Handle, uint16_t, _iq24 *)

函数： 此函数返回 SpinTAC 速度规划的变量值。 通常用于从 SpinTAC 速度规划向外部组件发送数据

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	uint16_t	VarIdx	要接收值的变量的索引
3	_iq24 *	Value	要接收值的外部变量的指针

void **STVELPLAN_setVar**(ST_VELPLAN_Handle, uint16_t, _iq24)

函数： 此函数设置 SpinTAC 速度规划中的变量值。 通常用于向 SpinTAC 速度规划传递传感器数据。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	uint16_t	VarIdx	要接收值的变量的索引
3	_iq24	Value	要设置到变量的值

void **STPOSCTL_run**(ST_POSCTL_Handle)

函数： 此函数运行 SpinTAC 位置控制。

参数：

编号	类型	参数	说明
1	ST_POSCTL_Handle	handle	ST_PosCtl_t 数据类型的句柄

void **STPOSMOVE_run**(ST_POSMOVE_Handle)

函数： 此函数运行 SpinTAC 位置移动。

参数：

编号	类型	参数	说明
1	ST_POSMOVE_Handle	handle	ST_PosMove_t 数据类型的句柄

void **STPOSCONV_run**(ST_POSCONV_Handle)

函数： 此函数运行 SpinTAC 位置转换。

参数：

参数：

编号	类型	参数	说明
1	ST_POSCONV_Handle	handle	ST_PosConv_t 数据类型的句柄

void **STPOSPLAN_run**(ST_POSPLAN_Handle)

函数： 此函数运行 SpinTAC 位置规划。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄

void **STPOSPLAN_runTick**(ST_POSPLAN_Handle)

函数： 此函数运行 SpinTAC 位置规划定时器。 必须从 ISR 运行。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄

void **STPOSPLAN_addCfgAct**(ST_POSPLAN_Handle, uint16_t, ST_PlanCond_e , uint16_t , uint16_t, uint16_t, ST_PlanActOptn_e, _iq24, ST_PlanActTrgr_e)

函数： 此函数向 SpinTAC 位置规划添加操作。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	uint16_t	StateIdx	将执行操作的状态的索引
3	uint16_t	VarIdx	要操作的变量的索引
4	ST_PlanCond_e	AndOr	ST_COND_NC：无条件执行； ST_COND_FC：满足第一个条件时执行； ST_COND_AND：两个条件都满足时执行； ST_COND_OR：满足任一个条件时执行；
5	uint16_t	CondIdx1	第一个条件的索引
6	uint16_t	CondIdx2	第二个条件的索引
7	ST_PlanActOptn_e	Opt	ST_ACT_EQ：设置变量的值 ST_ACT_QDD：向变量添加值
8	_iq24	Value	要设置到变量或添加到变量的值
9	ST_PlanActTrgr_e	EnterExit	ST_ACT_ENTR：进入状态时执行操作 ST_ACT_EXIT：离开状态时执行操作

void **STPOSPLAN_setCfgArray**(ST_POSPLAN_Handle, uint32_t *, const size_t, uint16_t, uint16_t, uint16_t, uint16_t, uint16_t)

函数： 此函数向 SpinTAC 位置规划添加操作。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	uint32_t *	cfgArray	用于规划配置的 cfgArray 的指针。
3	const size_t	numBytes	分配给 cfgArray 数组的字节数。 通过调用 sizeof(cfgArray) 获取字节数
4	uint16_t	MaxActNum	操作数
5	uint16_t	MaxCondNum	条件数
6	uint16_t	MaxVarNum	变量数
7	uint16_t	MaxTranNum	转换数
8	uint16_t	MaxStateNum	状态数

ST_POSPLAN_Handle **STPOSPLAN_init**(void *, const size_t)

函数： 此函数初始化 SpinTAC 位置规划。

参数：

编号	类型	参数	说明
1	void *	pMemory	指向 ST_PosPlan_t 对象的内存的指针
	const size_t	numBytes	分配给 ST_PosPlan_t 对象的字节数

返回： ST_PosPlan_t 对象句柄

void **STPOSPLAN_addCfgCond**(ST_POSPLAN_Handle, uint16_t, ST_PlanComp_e, _iq24, _iq24)

函数： 此函数向 SpinTAC 位置规划添加条件。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	uint16_t	VarIdx	要比较的变量的索引
3	ST_PlanComp_e	Comp	ST _COMP_NA：未定义； ST _COMP_EQ：等于 Value1； ST _COMP_NEQ：不等于 Value1； ST__COMP_GT：大于 Value1； ST_ COMP_EGT：等于或大于 Value1； ST_ COMP_LW：小于 Value1； ST_ COMP_ELW：等于或小于； ST_ COMP_In：属于 (Value1, Value2) 的范围； ST_ COMP_Ein：属于 [Value1, Value2) 的范围； ST_ COMP_InE：属于 (Value1, Value2] 的范围； ST_ COMP_EinE：属于 [Value1, Value2] 的范围； ST_ COMP_Out：不属于 [Value1, Value2] 的范围； ST_ COMP_Eout：不属于 (Value1, Value2] 的范围； ST_ COMP_OutE：不属于 [Value1, Value2) 的范围； ST_ COMP_EoutE：不属于 (Value1, Value2) 的范围
4	_iq24	Value1	第一个值
5	_iq24	Value2	第二个值

void **STPOSPLAN_addCfgVarCond**(ST_POSPLAN_Handle, uint16_t, uint16_t, ST_PlanComp_e)

函数： 此函数添加将两个变量与 SpinTAC 位置规划进行比较的条件。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	uint16_t	VarIdx1	要比较的第一个变量的索引
3	uint16_t	VarIdx2	要比较的第二个变量的索引
4	ST_PlanComp_e	Comp	ST _COMP_NA： 未定义； ST _COMP_EQ： 等于 Value1； ST _COMP_NEQ： 不等于 Value1； ST__COMP_GT： 大于 Value1； ST_ COMP_EGT： 等于或大于 Value1； ST_ COMP_LW： 小于 Value1； ST_ COMP_ELW：等于或小于；

void **STPOSPLAN_setCfgHaltState**(ST_POSPLAN_Handle, int32_t, _iq24,_iq24, _iq24, , int32_t)

函数： 此函数向 SpinTAC 位置规划添加状态。

参数：

编号	类型	参数	说明
1	ST_VELPLAN_Handle	handle	ST_VelPlan_t 数据类型的句柄
2	int32_t	PosStepInt_mrev	HALT 状态的位置阶跃整数部分
3	_iq24	PosStepFrac_mrev	HALT 状态的位置阶跃分数部分
4	_iq24	VelLim	HALT 状态的速度限制
5	_iq24	AccLim	HALT 状态的加速度限制
6	_iq20	JrkLim	HALT 状态的急动限制

编号	类型	参数	说明
7	Int32_t	Timer	用于指示在进一步操作前 HALT 状态保持时间的定时器 ([计数])

void **STPOSPLAN_setCfg**(ST_POSPLAN_Handle, _iq24, _iq24, _iq24, _iq20, bool)

函数： 此函数设置 SpinTAC 位置规划系统配置。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	_iq24	T_sec	采样时间 ([s])
3	_iq24	VelMax	系统速度上限
4	_iq24	AccMax	系统加速度上限
5	_iq20	JrkMax	系统急动上限
6	bool	LoopENB	false：规划仅运行一次，然后回到 IDLE 状态 true：规划每次进入 IDLE 状态时都再次运行

void **STPOSPLAN_addCfgState**(ST_POSPLAN_Handle, int32_t, _iq24, int32_t)

函数： 此函数向 SpinTAC 位置规划添加状态。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	int32_t	PosStepInt_mrev	位置阶跃命令整数部分值
3	_iq24	PosStepFrac_mrev	位置阶跃命令分数部分值
4	int32_t	Timer_tick	用于指示状态保持时间的定时器 ([计数])

void **STPOSPLAN_addCfgTran**(ST_POSPLAN_Handle, uint16_t, uint16_t, ST_PlanCond_e, uint16_t, uint16_t, _iq24, _iq24, _iq20)

函数： 此函数向 SpinTAC 位置规划添加事务。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	uint16_t	FromState	转换的起始状态的索引
3	uint16_t	ToState	转换的结束状态的索引
4	ST_PlanCond_e	AndOr	ST_COND_NC：无条件转换； ST_COND_FC：满足第一个条件时转换； ST_COND_AND：两个条件都满足时转换； ST_COND_OR：满足任一个条件时转换；
5	uint16_t	CondIdx1	第一个条件的索引
6	uint16_t	CondIdx2	第二个条件的索引
7	_iq24	VelLim	用于此转换的速度限制
8	_iq24	AccLim	用于此转换的加速度限制
9	_iq24	DecLim	用于此转换的减速度限制
10	_iq20	JrkLim	用于此转换的急动限制

void **STPOSPLAN_addCfgVar**(ST_POSPLAN_Handle, ST_PlanVar_e, _iq24)

函数： 此函数向 SpinTAC 位置规划添加变量。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	ST_PlanVar_e	Type	ST_VAR_INOUT：定时器类型，可在条件中使用 ST_VAR_IN：传感器类型，仅接收值，可在条件中使用 ST_VAR_OUT：传动器类型，仅发送值，不能在条件中使用
3	_iq24	Value	变量的初始值

void **STPOSPLAN_setUnitProfDone**(ST_POSPLAN_Handle, bool)

函数： 此函数在当前运行的系统配置完成后通知 SpinTAC 位置规划。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	bool	ProDON	false：未完成；true：已完成

void **STPOSPLAN_getVar**(ST_POSPLAN_Handle, uint16_t, _iq24 *)

函数： 此函数返回 SpinTAC 位置规划的变量值。 通常用于从 SpinTAC 位置规划向外部组件发送数据。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	uint16_t	VarIdx	要接收值的变量的索引
3	_iq24 *	Value	要接收值的外部变量的指针

void **STPOSPLAN_setVar**(ST_POSPLAN_Handle, uint16_t, _iq24)

函数： 此函数返回 SpinTAC 位置规划的变量值。 通常用于从 SpinTAC 位置规划向外部组件发送数据。

参数：

编号	类型	参数	说明
1	ST_POSPLAN_Handle	handle	ST_PosPlan_t 数据类型的句柄
2	uint16_t	VarIdx	要接收值的变量的索引
3	_iq24	Value	要设置到变量的值

void **ST_getVersionNumber**(ST_VER_Handle, uint16_t *, uint16_t *, uint16_t *)

函数： 此函数返回 SpinTAC 库的版本号。

参数：

编号	类型	参数	说明
1	ST_VER_Handle	handle	ST_Ver_t 数据类型的句柄
2	uint16_t *	major	库的主要版本
3	uint16_t *	minor	库的次要版本
4	uint16_t *	revision	库的修订版本

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

用户参数 (user.h)

User.h 中存储了所有的用户参数。 其中的某些值可在运行时通过 GUI 或 CCStudio 更改，但必须在 user.h 中更新后才能永久保存。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

4.1 电流和电压

User.h 包含针对 CTRL、HAL 和 EST 模块的用户初始化数据的公共接口。

4.1.1 USER_IQ_FULL_SCALE_FREQ_Hz

```
#define USER_IQ_FULL_SCALE_FREQ_Hz   (800.0)
```

此模块定义 IQ 变量的满量程频率（单位为 Hz）。 所有频率均根据对此值的比率转换为 (pu)。 此值必须大于预期的电机最大速度。

4.1.2 USER_IQ_FULL_SCALE_VOLTAGE_V

```
#define USER_IQ_FULL_SCALE_VOLTAGE_V   (24.0)
```

定义系统内 IQ30 电压变量的满量程值。 所有电压均根据对此值的比率转换为 (pu)。

CAUTION

- 此值必须大于控制系统中计算得出的任何电压的最大值，否则此值会饱和并翻转，从而产生错误值。
- 此值通常大于测得的最大 ADC 值，尤其是在高反电势电机的运行速度大于额定速度的情况下。
- 如果已知反电势常数并且知道运行速度因场强减弱导致超过额定速度，请确保将此值设置为大于预期反电势电压。
- 此值还可用于计算可通过以下公式识别的最小磁通：
  USER_IQ_FULL_SCALE_VOLTAGE_V/USER_EST_FREQ_Hz/0.7

对于高磁通电机（即洗衣机电机），建议开始使用的值大于 USER_ADC_FULL_SCALE_VOLTAGE_V 约 3 倍，在反电势计算可能超出这些限制的情况下可增加到 4-5 倍。

对于低磁通电机（即低电感、高速小型低成本电机），建议设置的值允许按以下公式识别磁通：

USER_IQ_FULL_SCALE_VOLTAGE_V/USER_EST_FREQ_Hz/0.7

4.1.3 USER_ADC_FULL_SCALE_VOLTAGE_V

```
#define USER_ADC_FULL_SCALE_VOLTAGE_V   (66.32)
```

此模块定义模数 (AD) 转换器的最大输入电压。 此值由最大 ADC 输入 (3.3V) 和转换 (0FFFh) 表示。 由于与硬件相关，此值应基于 ADC 输入的电压传感和换算系数。

4.1.4 USER_VOLTAGE_SF

```
#define USER_VOLTAGE_SF
((float_t)((USER_ADC_FULL_SCALE_VOLTAGE_V)/(USER_IQ_FULL_SCALE_VOLTAGE_V)))
```

此模块定义系统的电压换算系数。

编译时计算系统中使用的换算系数（比率）。

4.1.5 USER_IQ_FULL_SCALE_CURRENT_A

```
#define USER_IQ_FULL_SCALE_CURRENT_A   (10.0)
```

此模块定义 IQ 变量的满量程电流（单位为 A）。

所有电流均根据对此值的比率转换为 (pu)。

CAUTION

此值必须大于预期的电机最大电流读数，否则读数会翻转为零并且出现控制问题。

4.1.6 USER_ADC_FULL_SCALE_CURRENT_A

```
#define USER_ADC_FULL_SCALE_CURRENT_A   (17.30)
```

此模块定义 AD 转换器的最大电流。

此值由最大 ADC 输入 (3.3V) 和转换 (0FFFh) 表示。

由于与硬件相关，此值应基于 ADC 输入的电流传感和换算系数。

4.1.7 USER_CURRENT_SF

```
#define USER_CURRENT_SF
((float_t)((USER_ADC_FULL_SCALE_CURRENT_A)/(USER_IQ_FULL_SCALE_CURRENT_A)))
```

此模块定义系统的换算系数。

编译时计算系统中使用的换算系数（比率）。

4.1.8 USER_NUM_CURRENT_SENSORS

```
#define USER_NUM_CURRENT_SENSORS   (3)
```

此模块定义使用的电流传感器的数量。

由硬件功能确定。

可以是 (2) 个或 (3) 个。

4.1.9 USER_NUM_VOLTAGE_SENSORS

```
#define USER_NUM_VOLTAGE_SENSORS   (3)
```

此模块定义电压（相位）传感器的数量。

必须是 (3) 个。

4.1.10 I_A_offset、I_B_offset、I_C_offset

```
#define I_A_offset (0. 8661925197)
#define I_B_offset (0.8679816127)
#define I_C_offset (0.8638074994)
```

此模块定义 A、B 和 C 相的 ADC 电流偏移。

与硬件一度相关，也可在运行时执行校准。

在电路板上进行初始校准后，应针对特定硬件更新这些值，使其可在编译为二进制后加载到控制器中。

4.1.11 V_A_offset、V_B_offset、V_C_offset

```
#define V_A_offset   (0.1776982546)
#define V_B_offset   (0.1776063442)
#define V_C_offset   (0.1771019101)
```

此模块定义 A、B 和 C 相的 ADC 电压偏移。

与硬件一度相关，也可在运行时执行校准。

在电路板上进行初始校准后，应针对特定硬件更新这些值，使其可在编译为二进制后加载到控制器中。

4.2 时钟与定时器

4.2.1 USER_SYSTEM_FREQ_MHz

```
#define USER_SYSTEM_FREQ_MHz   (90.0) // Maximum frequency for F2805xF/M and F2806xF/M devices
#define USER_SYSTEM_FREQ_MHz   (60.0) // Maximum frequency for F2802xF devices
```

此模块定义系统时钟频率（单位为 MHz）。

4.2.2 USER_PWM_FREQ_kHz

```
#define USER_PWM_FREQ_kHz   (20.0)
```

此模块定义脉宽调制 (PWM) 频率（单位为 kHz）。

为保证稳定运行，可直接在此处将 PWM 频率设置为不超过 30kHz（某些情况下最大值为 60kHz）。

对于更高的 PWM 频率（对于低电感、高电流的脉动电机，典型值超过 60kHz），建议使用 ePWM 硬件和可调 ADC SOC 来抽取控制系统的 ADC 转换结束中断。 使用硬件抽取 USER_NUM_PWM_TICKS_PER_ISR_TICK 即可完成上述操作。 如果未对高 PWM 频率使用硬件抽取，则可能会丢失中断并扰乱控制状态机的时序。

4.2.3 USER_MAX_VS_MAG_PU

```
#define USER_MAX_VS_MAG_PU   (1.0)
```

如果未使用电流重构技术，则设置为 1.0。 有关详细信息，请参见实验 10a-x 中的 svgen_current 模块。

确定允许的最大电压矢量 (Vs) 幅度。 此值用于设置 Id 和 Iq PI 电流控制器输出的最大幅值。 Id 和 Iq PI 电流控制器的输出为 Vd 和 Vq。

Vs、Vd 与 Vq 之间的关系为：

Vs = sqrt(Vd^2 + Vq^2)。

在此 FOC 控制器中，将 Vd 值设置为等于 USER_MAX_VS_MAG*USER_VD_MAG_FACTOR。

Vq = sqrt(USER_MAX_VS_MAG^2 - Vd^2)。

- 对于波峰占空比 SQRT(3)/2 = 86.6% 的纯正弦波，设置 USER_MAX_VS_MAG = 1.0。 这种情况下，不需要使用电流重构技术。
- 对于波峰占空比为 100% 的纯正弦波，设置 USER_MAX_VS_MAG = 2/SQRT(3) = 1.1547。 这种情况下（实验 10a-x），需要使用电流重构技术。
- 为生成梯形电压波形，设置 USER_MAX_VS_MAG = 4/3 = 1.3333。 这种情况下（实验 10a-x），需要使用电流重构技术。
- 对于空间矢量过调制，有关 SVM 生成器生成梯形的系统要求的详细信息，请参见实验 10。

4.2.4 USER_PWM_PERIOD_usec

```
#define USER_PWM_PERIOD_usec   (1000.0/USER_PWM_FREQ_kHz)
```

此模块定义脉宽调制 (PWM) 期间（单位为微秒）。

编译时计算。

4.2.5 USER_ISR_FREQ_Hz

```
#define USER_ISR_FREQ_Hz
```

```
((float_t)USER_PWM_FREQ_kHz * 1000.0 / (float_t)USER_NUM_PWM_TICKS_PER_ISR_TICK)
```

此模块定义中断服务程序 (ISR) 频率（单位为 Hz）。

编译时计算。

4.2.6 USER_ISR_PERIOD_usec

```
#define USER_ISR_PERIOD_usec
  (USER_PWM_PERIOD_usec * (float_t)USER_NUM_PWM_TICKS_PER_ISR_TICK)
```

此模块定义中断服务程序 (ISR) 期间（单位为微秒）。

4.3 抽取率

抽取率定义模块执行间的节拍数。

控制器时钟节拍 (CTRL) 是用于软件中所有时序的主时钟。

通常，PWM 频率用于触发（可由 ePWM 硬件抽取以减少开销）ADC SOC。

ADC SOC 用于触发 ADC 转换结束。

ADC 转换结束用于触发 ISR。

这会将硬件 ISR 速率与软件控制速率关联起来。

通常，需要对 16kHz ISR 进行某种形式的抽取（ePWM 硬件，CURRENT 或 EST），以确保中断完成并为后台任务预留一些时间。

4.3.1 USER_NUM_PWM_TICKS_PER_ISR_TICK

```
#define USER_NUM_PWM_TICKS_PER_ISR_TICK    (1)
```

此模块定义每次中断的 PWM 周期数。

PWM 频率与中断频率之间的关系。

4.3.2 USER_NUM_ISR_TICKS_PER_CTRL_TICK

```
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK   (1)
```

此模块定义每个电流控制器时钟节拍的控制器时钟节拍数。

控制器时钟速率与电流控制器 (FOC) 速率之间的关系。

4.3.3 USER_NUM_CTRL_TICKS_PER_CURRENT_TICK

```
#define USER_NUM_CTRL_TICKS_PER_CURRENT_TICK   (1)
```

此模块定义每个估算器时钟节拍的控制器时钟节拍数。

控制器时钟速率与估算器 (FAST) 速率之间的关系。

4.3.4 USER_NUM_CTRL_TICKS_PER_EST_TICK

```
#define USER_NUM_CTRL_TICKS_PER_EST_TICK   (1)
```

此模块取决于所需的动态性能，FAST 提供的最佳结果可低至 1kHz，而更加动态或高速应用可能需要高达 15kHz 的频率。

4.3.5 USER_NUM_CTRL_TICKS_PER_SPEED_TICK

```
#define USER_NUM_CTRL_TICKS_PER_SPEED_TICK   (20)
```

此模块定义每个速度控制器时钟节拍的控制器时钟节拍数。

控制器时钟速率与速度环路速率之间的关系。

4.3.6 USER_NUM_CTRL_TICKS_PER_TRAJ_TICK

```
#define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK   (20)
```

此模块定义每个轨迹时钟节拍的控制器时钟节拍数。

控制器时钟速率与轨迹环路速率之间的关系。

通常与速率相同。

4.3.7 USER_CTRL_FREQ_Hz

#define USER_CTRL_FREQ_Hz
(uint_least32_t)(USER_ISR_FREQ_Hz/USER_NUM_ISR_TICKS_PER_CTRL_TICK)

此模块定义控制器频率（单位为 Hz）。

编译时计算。

4.3.8 USER_EST_FREQ_Hz

```
#define USER_EST_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_EST_TICK)
```

此模块定义估算器频率（单位为 Hz）。

编译时计算。

4.3.9 USER_TRAJ_FREQ_Hz

```
#define USER_TRAJ_FREQ_Hz
(uint_least32_t)(USER_CTRL_FREQ_Hz/USER_NUM_CTRL_TICKS_PER_TRAJ_TICK)
```

此模块定义轨迹频率（单位为 Hz）。

编译时计算。

4.3.10 USER_CTRL_PERIOD_usec

```
#define USER_CTRL_PERIOD_usec   (USER_ISR_PERIOD_usec *
USER_NUM_ISR_TICKS_PER_CTRL_TICK)
```

此模块定义控制器执行周期（单位为微妙）。

编译时计算。

4.3.11 USER_CTRL_PERIOD_sec

```
#define USER_CTRL_PERIOD_sec
((float_t)USER_CTRL_PERIOD_usec/(float_t)1000000.0)
```

此模块定义控制器执行周期（单位为秒）。

编译时计算。

4.4 限制

4.4.1 USER_MAX_NEGATIVE_ID_REF_CURRENT_A

```
#define USER_MAX_NEGATIVE_ID_REF_CURRENT_A   (-0.5 * USER_MOTOR_MAX_CURRENT)
```

示例，可对其进行调整以满足电机安全需求：

-0.5 * USER_MOTOR_MAX_CURRENT

此模块定义要在 Id 参考值中应用的最大负电流。

仅用于场强减弱，这是安全设置（例如，用于防止消磁）。

用户必须注意总电流强度 [$\sqrt{(Id^2 + Iq^2)}$] 应始终低于所有机械设计规格。

4.4.2 USER_ZEROSPEEDLIMIT

```
#define USER_ZEROSPEEDLIMIT   (1.0 / USER_IQ_FULL_SCALE_FREQ_Hz)
```

典型值 = 0.002pu，1-5Hz

Hz = USER_ZEROSPEEDLIMIT * USER_IQ_FULL_SCALE_FREQ_Hz

此模块定义磁通积分器的速度下限（单位为 pu）。

这是速度范围 (CW/CCW)，强制角对象在此范围内有效（仅限于该对象启用后）。

如果强制角对象超出此速度范围或被禁用，则绝对不会生效，这种情况下，只能由 FAST 提供角度。

4.4.3 USER_FORCE_ANGLE_FREQ_Hz

```
#define USER_FORCE_ANGLE_FREQ_Hz   (USER_ZEROSPEEDLIMIT * USER_IQ_FULL_SCALE_FREQ_Hz)
```

典型强制角启动速度 = 1.0

此模块定义强制角频率（单位为 Hz）。

强制角对象使用的定子矢量旋转频率。

可以为正，也可以为负

4.4.4 USER_MAX_CURRENT_SLOPE_POWERWARP

```
#define USER_MAX_CURRENT_SLOPE_POWERWARP
 (0.3*USER_MOTOR_RES_EST_CURRENT/USER_IQ_FULL_SCALE_CURRENT_A/USER_TRAJ_FREQ_Hz)
```

此模块定义 PowerWarp 模式下 Id 轨迹的最大电流斜率。

仅用于感应电机，可控制在 PowerWarp 控制下 Id 输入的变化速度。

4.4.5 USER_MAX_ACCEL_Hzps

```
#define USER_MAX_ACCEL_Hzps   (20.0)
```

此模块定义速度曲线的最大启动加速度*和*减速度（单位为 Hz/秒）。

可通过用户函数在运行时更新。

逆变器、电机、惯性和负载将限制实际加速能力。

4.4.6 USER_MAX_ACCEL_EST_Hzps

```
#define USER_MAX_ACCEL_EST_Hzps   (2.0)
```

此模块定义估算速度曲线的最大加速度（单位为 Hz/秒）。

仅在电机识别期间使用（确认）。

4.4.7 USER_MAX_CURRENT_SLOPE

```
#define USER_MAX_CURRENT_SLOPE
```

```
(USER_MOTOR_RES_EST_CURRENT/USER_IQ_FULL_SCALE_CURRENT_A/USER_TRAJ_FREQ_Hz)
```

此模块定义估算期间 Id 轨迹的最大电流斜率。

4.4.8 USER_IDRATED_FRACTION_FOR_RATED_FLUX

```
#define USER_IDRATED_FRACTION_FOR_RATED_FLUX   (1.0)
```

此模块定义要在额定磁通估算期间使用的 Id 额定电流（小数表示形式）。

默认值为 1.0；请勿更改。

4.4.9 USER_IDRATED_FRACTION_FOR_L_IDENT

```
#define USER_IDRATED_FRACTION_FOR_L_IDENT   (1.0)
```

此模块定义要在电感估算期间使用的 Id 额定电流（小数表示形式）。

默认值为 1.0；请勿更改。

4.4.10 USER_IDRATED_DELTA

```
#define USER_IDRATED_DELTA   (0.00002)
```

此模块定义要在估算期间使用的 Id 额定电流增量。

4.4.11 USER_SPEEDMAX_FRACTION_FOR_L_IDENT

```
#define USER_SPEEDMAX_FRACTION_FOR_L_IDENT   (1.0)
```

此模块定义要在电感估算期间使用的最大速度（小数表示形式）。

4.4.12 USER_FLUX_FRACTION

```
#define USER_FLUX_FRACTION   (1.0)
```

此模块定义要在电感识别期间使用的磁通（小数表示形式）。

4.4.13 USER_POWERWARP_GAIN

```
#define USER_POWERWARP_GAIN   (1.0)
```

此模块定义用于计算 Id 参考值的 PowerWarp 增益。

仅用于感应电机。 默认值为 1.0；请勿更改。

4.4.14 USER_R_OVER_L_EST_FREQ_Hz

```
#define USER_R_OVER_L_EST_FREQ_Hz   (300)
```

此模块定义 R/L 估算频率（单位为 Hz）。

4.5 极

4.5.1 USER_VOLTAGE_FILTER_POLE_Hz

```
#define USER_VOLTAGE_FILTER_POLE_Hz   (714.14)
```

此模块定义模拟电压滤波器极点位置（单位为 Hz）。

必须与 Vph 硬件滤波器匹配。

4.5.2 USER_VOLTAGE_FILTER_POLE_rps

#define USER_VOLTAGE_FILTER_POLE_rps (2.0 * MATH_PI * USER_VOLTAGE_FILTER_POLE_Hz)

此模块定义模拟电压滤波器极位置（单位为 rad/s）。

编译时计算单位从 Hz 转换为 rad/s。

4.5.3 USER_OFFSET_POLE_rps

```
#define USER_OFFSET_POLE_rps   (20.0)
```

此模块定义电压和电流偏移估算的软件极点位置（单位为 rad/s）。

不应更改默认值 (20.0)。

4.5.4 USER_FLUX_POLE_rps

```
#define USER_FLUX_POLE_rps   (100.0)
```

此模块定义磁通估算的软件极点位置（单位为 rad/s）。

不应更改默认值 (100.0)。

4.5.5 USER_DIRECTION_POLE_rps

```
#define USER_DIRECTION_POLE_rps   (6.0)
```

此模块定义方向滤波器的软件极点位置（单位为 rad/s）。

4.5.6 USER_SPEED_POLE_rps

```
#define USER_SPEED_POLE_rps   (100.0)
```

此模块定义速度控制滤波器的软件极点位置（单位为 rad/s）。

4.5.7 USER_DCBUS_POLE_rps

```
#define USER_DCBUS_POLE_rps   (100.0)
```

此模块定义直流总线滤波器的软件极点位置（单位为 rad/s）。

4.5.8 USER_EST_KAPPAQ

```
#defineUSER_EST_KAPPAQ   (1.5)
```

此模块定义估算器的收敛因子。

请勿更改针对 FAST 的默认值。

4.6 使用电机和识别设置

4.6.1 USER_MOTOR_TYPE

```
#define USER_MOTOR_TYPE   MOTOR_Type_Pm
```

Motor_Type_Pm（所有同步电机：BLDC、PMSM、SMPM、IPM）或 Motor_Type_Induction（ACI 异步电机）。

4.6.2 USER_MOTOR_NUM_POLE_PAIRS

```
#define USER_MOTOR_NUM_POLE_PAIRS   (4)
```

PAIRS，不是总极数。 仅用于根据转子频率 (Hz) 计算用户 RPM。

4.6.3 USER_MOTOR_Rr

```
#define USER_MOTOR_Rr   (NULL)
```

仅用于感应电机，否则为 NULL。

4.6.4 USER_MOTOR_Rs

```
#define USER_MOTOR_Rs   (2.303403)
```

识别的相位 Y 等效电路中的中性电阻（Ω，浮点数）。

4.6.5 USER_MOTOR_Ls_d

```
#define USER_MOTOR_Ls_d   (0.008464367)
```

对于 PM 电机，识别的平均定子电感（H，浮点数）。

4.6.6 USER_MOTOR_Ls_q

```
#define USER_MOTOR_Ls_q   (0.008464367)
```

对于 PM 电机，识别的平均定子电感（H，浮点数）。

4.6.7 USER_MOTOR_RATED_FLUX

```
#define USER_MOTOR_RATED_FLUX   (0.38)
```

识别的转子和定子间总磁链（韦伯 = 伏*秒）。

4.6.8 USER_VOLTAGE_FILTER_POLE_Hz

```
#define USER_MOTOR_MAGNETIZING_CURRENT   (NULL)
```

仅用于感应电机，否则为 NULL。

4.6.9 USER_MOTOR_RES_EST_CURRENT

```
#define USER_MOTOR_RES_EST_CURRENT   (1.0)
```

在电机识别期间用于 Rs 估算的最大电流（A，浮点数），留 10-20% 余量。

4.6.10 USER_MOTOR_IND_EST_CURRENT

```
#define USER_MOTOR_IND_EST_CURRENT   (-1.0)
```

在电机识别期间用于 Ls 估算的最大电流（A，负浮点数），刚好能够使电机转动。

4.6.11 USER_MOTOR_MAX_CURRENT

```
#define USER_MOTOR_MAX_CURRENT   (3.82)
```

注意事项：在电机识别和运行期间使用，可限定所提供的速度 PI 控制器向 Iq 控制器输出的最大电流。

4.6.12 USER_MOTOR_FLUX_EST_FREQ_Hz

```
#define USER_MOTOR_FLUX_EST_FREQ_Hz   (20.0)
```

电机识别期间的最大指令速度（Hz，浮点值），约为额定速度的 10%。

4.6.13 USER_MOTOR_ENCODER_LINES（仅限 InstaSPIN-MOTION）

```
#define USER_MOTOR_ENCODER_LINES      (2500.0)
```

用于设置编码器，可指定编码轮上的行数。

4.6.14 USER_MOTOR_MAX_SPEED_KRPM（仅限 InstaSPIN-MOTION）

```
#define USER_MOTOR_MAX_SPEED_KRPM      (3.0)
```

用于设置位置控制应用的速度参考值的上限。

4.6.15 USER_SYSTEM_INERTIA（仅限 InstaSPIN-MOTION）

```
#define USER_SYSTEM_INERTIA      (0.02)
```

惯性用于描述与电机刚性耦合的质量。 InstaSPIN-MOTION 控制器将其用作输入。 应通过 InstaSPIN-MOTION 惯性识别对其进行识别。

4.6.16 USER_SYSTEM_FRICTION（仅限 InstaSPIN-MOTION）

```
#define USER_SYSTEM_FRICTION      (0.02)
```

摩擦力用于描述电机所接触到的运动阻力。 InstaSPIN-MOTION 控制器将其用作输入。 应通过 InstaSPIN-MOTION 惯性识别对其进行识别。

4.6.17 USER_SYSTEM_BANDWIDTH_SCALE（仅限 InstaSPIN-MOTION）

```
#define USER_SYSTEM_BANDWIDTH_SCALE      (1.0)
```

带宽范围用于设置 InstaSPIN-MOTION 控制器所使用的默认带宽。 在完成对 SPIN-MOTION 控制器的调整过程后应对其进行更新。

4.7 SpinTAC 参数（spintac_velocity.h 和 spintac_position.h）

需要针对特定应用配置用于组成 InstaSPIN-MOTION 的 SpinTAC 组件。 此过程很简单。 InstaSPIN-MOTION 中 SpinTAC 组件的特定配置详情将在本文档后续章节中介绍。 利用 spintac_velocity.h 和 spintac_position.h 头文件可以将 SpinTAC 组件包含在项目中。 这些文件包含设置 SpinTAC 组件所需的宏定义、类型定义和函数定义。 为了使用组成 InstaSPIN-MOTION 的 SpinTAC 组件，需要首先在项目中包含这些文件。 应包含的文件取决于控制类型。 如果应用为速度应用，则应包含 spintac_velocity.h。 如果应用需要进行位置控制，则应包含 spintac_position.h。

4.7.1 宏定义

宏定义 (#define) 提供一种简单的方法，可在 Motorware 项目中的多个位置更新配置。 #define 指令用于指定宏标识符和替换项。 替换项将在编译时对随后遇到的每个宏标识符进行替换。 SpinTAC 组件所使用的宏定义在 spintac_velocity.h 和 spintac_position.h 文件中有具体描述。

4.7.1.1 ST_MREV_ROLLOVER（仅限 spintac_position.h）

定义用于表示 SpinTAC 位置控制中机械旋转的最大和最小值。 可用于保持高精度。 当机械旋转的值达到 ST_MREV_ROLLOVER 中定义的值时，翻转计数器会递增，而机械旋转的值将设置为负 ST_MREV_ROLLOVER。

4.7.1.2 ST_EREV_MAXIMUM（仅限 spintac_position.h）

用于定义电气旋转的最大值。 此为编码器或其它电角源产生的最大电角值。

4.7.1.3 ST_POS_ERROR_MAXIMUM_MREV（仅限 spintac_position.h）

定义应用允许的最大位置误差。 如果检测到位置误差超过此阈值，则控制器输出将一直为零直到位置误差降至此阈值以下为止。

4.7.1.4 ST_ISR_TICKS_PER_SPINTAC_TICK

用于识别 InstaSPIN-MOTION 的 SpinTAC 组件的抽取因数。 此值表示每次执行 SpinTAC 组件时完成的 ISR 数量。 此值可根据 user.h 中定义的参数进行计算。 抽取因数在9.2 节中有详细介绍。

4.7.1.5 ST_SPEED_SAMPLE_TIME

用于确定执行 SpinTAC 组件的频率。 此值可根据 user.h 中定义的参数进行计算。

4.7.1.6 ST_SPEED_PU_PER_KRPM

用于确定 krpm 和 pu/s 速度间的换算系数。 用于将用户变量（通常为 krpm）转换为标度速度变量。 这样可确保 InstaSPIN-MOTION 中的所有计算不会导致溢出。 此值可根据 user.h 中定义的参数进行计算

4.7.1.7 ST_SPEED_KRPM_PER_PU

用于确定 pu/s 和 krpm 速度间的换算系数。 用于将标度速度变量转换为用户单位变量。 这样可确保 InstSPIN-MOTION 中的所有计算不会导致溢出。 此值可根据 user.h 中定义的参数进行计算

4.7.1.8 ST_MOTOR_INERTIA_PU

用于识别以标度单位表示的系统惯性。 此值根据 ST_MOTOR_INERTIA_A_PER_KRPM 进行计算并将提供给 SpinTAC 速度控制或 SpinTAC 位置控制。

4.7.1.9 ST_MOTOR_FRICTION_PU

用于识别以标度单位表示的系统惯性。 此值根据 ST_MOTOR_FRICTION_A_PER_KRPM 进行计算并将提供给 SpinTAC 速度控制或 SpinTAC 位置控制。

4.7.1.10 ST_MIN_ID_SPEED_RPM

用于识别运行 SpinTAC 速度识别之前的电机最小速度。 执行此操作可确保在电机旋转过快的情况下不会启动惯性识别过程。 该值单位为 rpm。 如果不能启动惯性识别过程，电机也无法保持零速度，则应通过增大此值来增加带宽，确保能够启动惯性识别过程。

4.7.1.11 ST_MIN_ID_SPEED_PU

用于识别运行 SpinTAC 速度识别之前的电机最小速度。 此值根据 ST_MIN_ID_SPEED_RPM 进行计算。 此值将与用户项目中的值进行比较。

4.7.1.12 ST_ID_INCOMPLETE_ERROR

只要 SpinTAC 速度识别失败且系统惯性未知就会触发此错误。 此值即为 SpinTAC 速度识别组件生成的特定错误代码。 请勿修改此值。

4.7.1.13 ST_VARS_DEFAULTS

用于识别应加载到节 4.7.2.5中描述的 ST_Vars_t 结构的默认值。 这些默认值用于初始化结构并在启动时提供正确的值。 请勿修改这些值。

4.7.2 类型定义

类型定义 (typedefs) 用于组织代码以调用 SpinTAC 和简化 API 用户体验。

4.7.2.1 VEL_Params_t/POS_Params_t

此结构用于识别项目中要使用的 SpinTAC 组件。 此结构将作为节 4.7.2.2中介绍的 ST_Obj 的一部分包含在程序中。 不应在项目中声明此结构。

4.7.2.2 ST_Obj

这是 SpinTAC 组件的主结构。 此结构旨在绕单电机轴对准 SpinTAC 组件。 此结构包含若干个子结构，可根据 SpinTAC 组件使用的电机轴部分对其进行划分。 此结构应在项目的主源文件中声明。

4.7.2.3 ST_Handle

此句柄用于表示主 SpinTAC 结构的地址。 应在用户项目中使用此句柄将主 SpinTAC 结构的地址传递到用户函数中，这样便可不必定义全局变量进行操作。 MotorWare 所含的实验项目中举例说明了如何设置用户函数以使用接口句柄。

4.7.2.4 ST_PlanButton_e

此枚举类型用于确定为控制 SpinTAC Plan 运行可以设置的状态。 执行此操作可构造小状态机以处理 SpinTAC Plan 的启用、禁用和重置。 应在主源文件中使用此类型构建一个按钮，该按钮可用于控制 SpinTAC Plan 运行。

4.7.2.5 ST_Vars_t

此结构包含操作 SpinTAC 组件应使用的用户接口变量。 不应在主源文件中声明此结构。 已在 main.h 和 main_position.h 中将其声明为 MOTOR_Vars_t 的一部分。 有关 MOTOR_Vars_t 的详细信息，请参见实验文档。

4.7.3 函数

以下函数用于初始化和配置 SpinTAC 组件。

4.7.3.1 ST_init

此函数用于初始化 SpinTAC 结构。 应在主源文件中进入死循环之前调用此函数。 此函数会传递 SpinTAC 组件的内存位置并返回一个用于连接这些组件的句柄。

4.7.3.2 ST_setupPosConv

此函数用于设置 SpinTAC 位置转换器的默认值。 应在主源文件中调用 ST_init 之后，进入死循环之前调用此函数。 此函数可从 user.h 和 spintac.h 中的宏定义中提取配置值。 应对这些值进行修改使其符合系统要求。

4.7.3.3 ST_setupVelCtl（仅限速度控制）

此函数用于设置 SpinTAC 速度控制的默认值。 应在主源文件中调用 ST_init 之后，进入死循环之前调用此函数。 此函数可从 user.h 和 spintac_velocity.h 中的宏定义中提取配置值。 应对这些值进行修改使其符合系统要求。

4.7.3.4 ST_setupPosCtl（仅限位置控制）

此函数用于设置 SpinTAC 位置控制的默认值。 应在主源文件中调用 ST_init 之后，进入死循环之前调用此函数。 此函数可从 user.h 和 spintac_position.h 中的宏定义中提取配置值。 应对这些值进行修改使其符合系统要求。

4.7.3.5 ST_setupVelMove（仅限速度控制）

此函数用于设置 SpinTAC 速度移动的默认值。 应在主源文件中调用 ST_init 之后，进入死循环之前调用此函数。 此函数可从 user.h 和 spintac_velocity.h 中的宏定义中提取配置值。 应对这些值进行修改使其符合系统要求。

4.7.3.6 ST_setupPosMove（仅限位置控制）

此函数用于设置 SpinTAC 位置移动的默认值。 应在主源文件中调用 ST_init 之后，进入死循环之前调用此函数。 此函数可从 user.h 和 spintac_position.h 中的宏定义中提取配置值。 应对这些值进行修改使其符合系统要求。

4.7.3.7 ST_setupVelPlan（仅限速度控制）

此函数用于设置 SpinTAC 速度规划。 应在调用 ST_init 函数之后调用此函数，并在项目的主源文件中进行声明后将其写入。 有关如何配置 SpinTAC Plan 的详细信息，请参见13.5 节。

4.7.3.8 ST_setupPosPlan（仅限位置控制）

此函数用于设置 SpinTAC 位置规划。 应在调用 ST_init 函数之后调用此函数，并在项目的主源文件中进行声明后将其写入。 有关如何配置 SpinTAC Plan 的详细信息，请参见13.5 节。

4.7.3.9 ST_setupVelId（仅限速度控制）

此函数用于设置 SpinTAC 速度识别的默认值。 应在主源文件中调用 ST_init 之后，进入死循环之前调用此函数。 此函数可从 user.h 和 spintac_velocity.h 中的宏定义中提取配置值。 应对这些值进行修改使其符合系统要求。

4.7.3.10 ST_runPosConv

此函数用于在 ISR 中运行 SpinTAC 位置转换器。 此函数应按照在 ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 此函数应在项目的主源文件中声明后写入该文件中。 InstaSPIN-MOTION 实验项目 12 至 13e 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 位置转换器。

4.7.3.11 ST_runVelCtl（仅限速度控制）

此函数用于在 ISR 中运行 SpinTAC 速度控制。 此函数应按照在 ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 此函数应在项目的主源文件中声明后写入该文件中。 InstaSPIN-MOTION 实验项目 05d 至 06d 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 速度控制。

4.7.3.12 ST_runPosCtl（仅限位置控制）

此函数用于在 ISR 中运行 SpinTAC 位置控制。 此函数应按照在 ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 此函数应在项目的主源文件中声明后写入该文件中。 InstaSPIN-MOTION 实验项目 13a 至 13e 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 位置控制。

4.7.3.13 ST_runVelMove（仅限速度控制）

此函数用于在 ISR 中运行 SpinTAC 速度移动。 此函数应按照在 ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 此函数应在项目的主源文件中声明后写入该文件中。 InstaSPIN-MOTION 实验项目 06a 至 06d 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 速度移动。

4.7.3.14 ST_runPosMove（仅限位置控制）

此函数用于在 ISR 中运行 SpinTAC 位置移动。 此函数应按照在 ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 此函数应在项目的主源文件中声明后写入该文件中。 InstaSPIN-MOTION 实验项目 13b 至 13e 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 位置移动。

4.7.3.15 ST_runVelPlan（仅限速度控制）

此函数用于运行 SpinTAC 速度规划的主要组件。 可以在 ISR 中调用此函数，也可以在项目的主循环中调用此函数。 此函数应在项目的主源文件中声明后写入该文件中。 InstaSPIN-MOTION 实验项目 06b 和 06c 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 速度规划。 InstaSPIN-MOTION 实验项目 06d 提供了一个示例，阐释如何在主循环中调用 SpinTAC 速度规划。

4.7.3.16 ST_runVelPlanTick（仅限速度控制）

此函数用于在 ISR 中运行 SpinTAC 速度规划的定时器组件。 此函数应按照在 ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 InstaSPIN-MOTION 实验项目 06b 至 06d 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 速度规划的这一组件。

4.7.3.17 ST_runPosPlan（仅限位置控制）

此函数用于运行 SpinTAC 位置规划的主要组件。 可以在 ISR 中调用此函数，也可以在项目的主循环中调用此函数。 此函数应在项目的主源文件中声明后写入该文件中。 InstaSPIN-MOTION 实验项目 13c 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 位置规划。 InstaSPIN-MOTION 实验项目 13d 提供了一个示例，阐释如何在主循环中调用 SpinTAC 位置规划。

4.7.3.18 ST_runPosPlanTick（仅限位置控制）

此函数用于在 ISR 中运行 SpinTAC 位置规划的 ISR 组件。 此函数应按照在 ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 InstaSPIN-MOTION 实验项目 13c 至 13d 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 位置规划的这一组件。

4.7.3.19 ST_runVelId（仅限速度控制）

此函数用于在 ISR 中运行 SpinTAC 速度识别。 此函数应按照在 ST_ISR_TICKS_PER_SPINTAC_TICK 中定义的抽取率进行抽取。 此函数应在项目的主源文件中声明后写入该文件中。 包含在 MotorWare 中的 InstaSPIN-MOTION 实验项目 05c 提供了一个示例，阐释如何在 ISR 中调用 SpinTAC 速度识别。

4.8 在 user.h 中设置 ACIM 电机参数

下面列出了 user.h 中为 ACIM 电机提供的参数：

```
#if (USER_MOTOR == User_ACIM)
#define USER_MOTOR_TYPE                 MOTOR_Type_Induction
#define USER_MOTOR_NUM_POLE_PAIRS       (2)
#define USER_MOTOR_Rr                   (5.054793)
#define USER_MOTOR_Rs                   (7.801885)
#define USER_MOTOR_Ls_d                 (0.03334743)
#define USER_MOTOR_Ls_q                 (USER_MOTOR_Ls_d)
#define USER_MOTOR_RATED_FLUX           (0.8165*230.0/60.0)
#define USER_MOTOR_MAGNETIZING_CURRENT  (1.134086)
#define USER_MOTOR_MAX_CURRENT          (5.0)
```

表 4-1 汇总了绕过 ACIM 电机识别时所需的 user.h 头文件中的全部参数。

表 4-1. user.h 中的 ACIM 电机参数

user.h 中的 ACIM 电机参数	ACIM 电机参数和单位	ACIM 电机模型符号
USER_MOTOR_Rr	转子电阻 (Ω)	R_R
USER_MOTOR_Rs	定子电阻 (Ω)	R_s
USER_MOTOR_Ls_d	定子串联电感 (H)	$L_{\sigma S}$
USER_MOTOR_RATED_FLUX	额定转子磁通 (V/Hz)	Ψ_R
USER_MOTOR_MAGNETIZING_CURRENT	额定磁化电流 (A)	i_{sd}

下一节将具体介绍上述所有参数以及如何从典型电机制造商数据表中获取这些参数。

4.8.1 从 ACIM 数据表中获取这些参数

图 4-1 即为用作示例的 ACIM 电机数据表。 电机部件号：56H17T2011A，制造商：马拉松电气公司 (www.marathonelectric.com)。

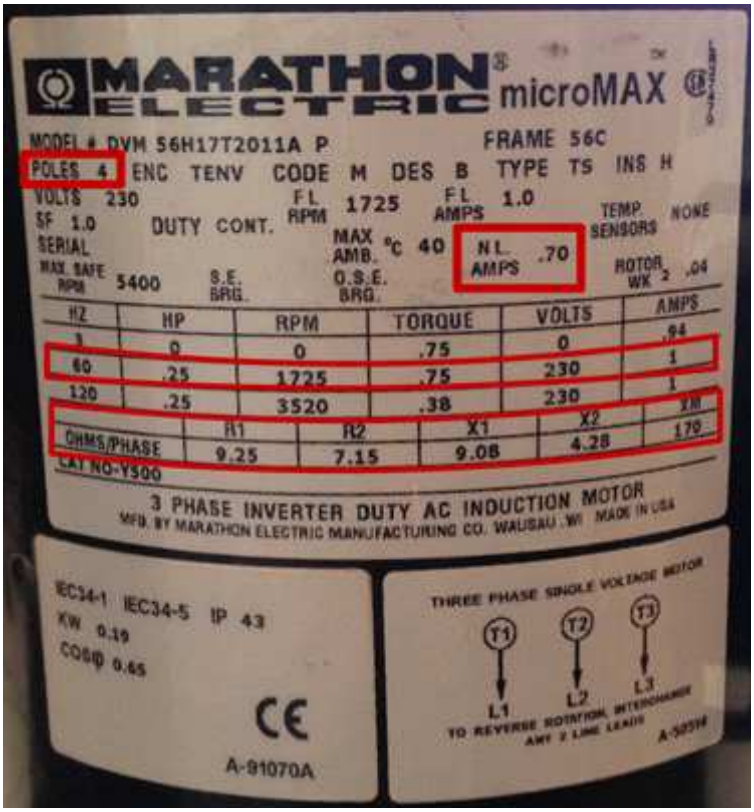


图 4-1. 示例 ACIM 电机数据表

4.8.1.1 极对数

极对数用于计算每分钟转数 (RPM) 等速度，也可用于某些磁通计算，如额定磁通计算示例所示。 只能在 user.h 中使用电机数据表中的极对数，如下所示。 请记住，有时候提供的是极数，因此我们只需将极数除以 2 即可得到极对数：

```
#define USER_MOTOR_NUM_POLE_PAIRS                (2)
```

4.8.1.2 将数据表中的 ACIM 电机参数输入 user.h 中

根据在电机铭牌上获得的信息，我们可以进行以下参数计算（假定阻抗计算的额定频率为 60Hz），然后将计算得出的值输入到 user.h 中。

$$R_R = \left(\frac{XM}{XM + X2}\right)^2 R2 = 6.8031\Omega$$

$$R_S = R1 = 9.25\Omega$$

$$L_{\sigma S} = \frac{XM}{(2\pi)f}\left(\frac{XM + X1}{XM} - \frac{XM}{XM + X2}\right) = 0.0352\ H$$

其中：

R1：定子电阻 (Ω)

R2：参考定子的转子电阻 (Ω)

X1：定子漏电抗 (Ω)

X2：参考定子的转子漏电抗 (Ω)

XM：磁化电抗 (Ω)

f：电机额定频率 (Hz)

R_R：参考定子的额定转子电阻 (Ω)

R_s：定子电阻 (Ω)

$L_{\sigma S}$：定子串联电感 (H)

请注意，为了将电抗 (X) 转换为电感 (L)，用户使用的频率必须与定义电抗值时所用频率相同，通常为电机的额定频率。 在此示例中，电机额定频率为 60Hz，因此 *f* = 60Hz。

得到这些值后，我们可以将各个电机参数输入到 user.h 中：

```
#define USER_MOTOR_Rr                   (6.8031)
#define USER_MOTOR_Rs                   (9.25)
#define USER_MOTOR_Ls_d                 (0.0352)
#define USER_MOTOR_Ls_q                 (USER_MOTOR_Ls_d)
```

在某些情况下，提供的 ACIM 电机参数为电感值而不是电抗值。 例如，如果将电机数据表中的值转化为电感，则会得到以下值：

$$L_{\sigma s} = \frac{X1}{(2\pi)f} = \frac{9.08}{(2\pi)60} = 0.0241\ H$$

$$L_{\sigma r} = \frac{X2}{(2\pi)f} = \frac{4.28}{(2\pi)60} = 0.0114\ H$$

$$L_m = \frac{XM}{(2\pi)f} = \frac{170}{(2\pi)60} = 0.4509\ H$$

现在可以将这一组值转换为 user.h 中所需的值，如下所示：

$$R_R = \left(\frac{L_m}{L_m + L_{\sigma r}}\right)^2 R2 = 6.8031\,\Omega$$

$$R_s = R1 = 9.25\,\Omega$$

$$L_{\sigma S} = L_m\left(\frac{L_m + L_{\sigma s}}{L_m} - \frac{L_m}{L_m + L_{\sigma r}}\right) = 0.0352\,H$$

其中：

$L_{\sigma s}$：定子漏电感 (H)

$L_{\sigma r}$：参考定子的转子漏电感 (H)

L_m：磁化电感 (H)

4.8.1.3 获取 ACIM 的额定磁化电流

为获取 ACIM 电机的额定磁化电流，我们可以根据电机铭牌中指定的空载电流进行计算。 在此特定示例中，空载电流 (i^{noload}) 为 0.7A，该值通常以均方根 (RMS) 的形式提供。 通常，此空载电流与额定磁化电流 (i_d^{rated}) ($i_q \cong 0$) 大致相同。我们需要在 user.h 中定义的值为最大振幅，可以按以下公式进行计算：

$$i_d^{rated} = i^{noload}\sqrt{2} = 0.7\sqrt{2} = 0.9899\ A$$

现在可将此值输入到 user.h 中：

```
#define USER_MOTOR_MAGNETIZING_CURRENT                  (0.9899)
```

4.8.1.4 获取 ACIM 的额定磁通

为了获取 ACIM 的额定磁通，我们需要根据铭牌上的值计算额定定子磁通，将计算得出的磁通减去电感产生的磁通即可得到额定转子磁通。 以下公式用于计算额定转子磁通：

$$\psi_s^{rated} = \left(\frac{\sqrt{2}}{\sqrt{3}}\right)\frac{v_{line-to-line}^{rated}}{(2\pi)f^{rated}} = \left(\frac{\sqrt{2}}{\sqrt{3}}\right)\frac{230}{(2\pi)60} = 0.4981\,Wb$$

$$\psi_R^{rated} = \psi_s^{rated} - L_{\sigma S} i_d^{rated} = 0.4981 - (0.0352)(0.9899) = 0.4633\ Wb$$

$$USER\_\psi_R^{rated} = (2\pi)\psi_R^{rated} = 2.9107\ V/Hz$$

现在可以在 user.h 中设置此值：

```
#define USER_MOTOR_RATED_FLUX                   (2.9107)
```

管理电机信号

对于 InstaSPIN FAST 观测器（磁通、转子磁通角、转轴转速和转矩），需要通过 ADC 来测量电机的电压和电流信号。 这些信号的精度可以直接影响观测器的性能。 本部分介绍所需的各种信号，InstaSPIN 软件中的参数用于对这些信号及其相关电路进行配置。 以下部分为“必要条件”，即成功进行电机识别和运行 InstaSPIN 所需的软件和硬件配置。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

5.1 软件必要条件

以下参数需要在用户软件中进行配置，以便管理 InstaSPIN FAST 观测器所需的电机信号。 本节将介绍各个参数。

- IQ 满量程频率 - 设置为电机的最大电频率，留 20-30% 余量
- IQ 满量程电压 - 设置为电机的最大电压，留 20-30% 余量
- IQ 满量程电流 - 设置为电机的最大可测量电流，留 20-30% 余量
- 最大电流 - 设置为电机制造商建议的最大电流（峰值），余量为 0%
- 抽取率 - 多个循环速率和设置
- 系统频率 - 设置为 MCU 的最大 CPU 时钟速度
- PWM 频率 - 默认值为 20KHz，随电机电感减小而逐渐增加
- 最大占空比 - 100% 占空比使用 3 分流电流测量

5.1.1 IQ 满量程频率

IQ 满量程频率代表以标么值表示的电机电频率。 也就是说，电机电频率按照此定义中的值进行标准化。 建议将 IQ 满量程频率值设置为与电机运行应用的最大绝对电频率相等。

*将 **IQ** 满量程频率设置为与最大绝对电频率相等。*

为说明此值的典型示例，以极对数为 4 的 PMSM 电机在最大绝对速度 15,000RPM 下运行为例。 这种情况下，电机的最大绝对频率为 15000/60*4 = 1000Hz。 建议在此例中使用以下 IQ 满量程频率设置：

```
//! \brief Defines the full-scale frequency for IQ variable, Hz
#define USER_IQ_FULL_SCALE_FREQ_Hz (1000.0)
```

请注意，此值必须大于电机的任何允许频率，因此建议为此值添加 20-30% 的余量，使其大于电机的最大预期频率。

5.1.2 IQ 满量程电压

与 IQ 满量程频率类似，IQ 满量程电压值用于将库中的所有电压术语按照标么值进行标准化。 出于这种原因，此定义值必须大于可为电机绕组提供的任何电压，包括电机内部存在的电压。 电机在场强减弱区域运行时，其运行速度将超过额定速度，此时电机内部的电压可能会大于输入电压本身。

*电机内部的电压可能会大于输入电压。 将 **IQ** 满量程频率设置为大于电机内部的任何电压。*

为了进行说明，以 PMSM 电机在额定速度 4000RPM 下运行为例。 如果使用 24V 电源驱动电机且未使用场强减弱，则电机内部和外部的所有电压均等于或小于 24V。 但是，如果使用场强减弱将电机速度加倍，使其达到最大值 8000RPM，那么电机内部的反电势电压的最大值可能会达到 24V 输入电压的两倍（即 48V）。 这种情况下，将 IQ 满量程电压定义设置为 48V，如以下代码示例所示：

```
//! \brief Defines the full-scale voltage for the IQ variable, V
#define USER_IQ_FULL_SCALE_VOLTAGE_V (48.0)
```

CAUTION

在以下 ψ_d^{max} and ψ_q^{max} 磁通计算中，如果这两个值中的任何一个等于或大于 2.0，则满足数值溢出条件，因为此值以数值形式表示，其最大整数范围为 2（IQ30 的最大值实际上非常接近于 2，即 $(2 - 2^{-30})$，有关此格式的详细信息，请参见 IQmath 库）。

$$V_{IQ}^{max} = USER\_IQ\_FULL\_SCALE\_VOLTAGE\_V = ?$$

$$\omega_{filter\_pole} = 2 \times \pi \times USER\_IQ\_FULL\_SCALE\_FREQ\_Hz = 2 \times \pi \times 714.15Hz = 4487.1rad/s$$

$$I_{motor}^{max} = USER\_MOTOR\_MAX\_CURRENT = 4.2A$$

$$L_{s\_d} = USER\_MOTOR\_Ls\_d = 0.0006H$$

$$L_{s\_q} = USER\_MOTOR\_Ls\_q = 0.0006H$$

$$\psi_d^{max} = I_{motor}^{max} \times L_{s\_d} \times \frac{\omega_{filter\_pole}}{V_{IQ}^{max}} < 2.0 \overset{yields}{\rightarrow} V_{IQ}^{max} > \frac{1}{2} \times I_{motor}^{max} \times L_{s\_d} \times \omega_{filter\_pole}$$

$$\psi_q^{max} = I_{motor}^{max} \times L_{s\_q} \times \frac{\omega_{filter\_pole}}{V_{IQ}^{max}} < 2.0 \overset{yields}{\rightarrow} V_{IQ}^{max} > \frac{1}{2} \times I_{motor}^{max} \times L_{s\_q} \times \omega_{filter\_pole}$$

在以上示例中，两个电感相同 ($L_{s\_d} = L_{s\_q}$)，因此 V_{IQ}^{max} 电压必须大于：

$$V_{IQ}^{max} > \frac{1}{2} \times I_{motor}^{max} \times L_{s\_d} \times \omega_{filter\_pole}$$

$$V_{IQ}^{max} > \frac{1}{2} \times 4.2A \times 0.0006\ H \times 4487.1\ rad/s$$

$$V_{IQ}^{max} > 5.65\ V$$

建议在此最小值的基础上留 20-30% 余量。 在以上示例中，电机的最大运行电压可为 48 V，由于此电压大于 V_{IQ}^{max}（含余量），因此可以将满量程电压设置为 48 V：

```
//! \brief Defines the full-scale voltage for the IQ variable, V
#define USER_IQ_FULL_SCALE_VOLTAGE_V (48.0)
```

用户必须选择相关参数以防止溢出。 如果电感未知，则必须在上述计算中使用粗略估算以便了解是否存在溢出条件。

除在 USER_IQ_FULL_SCALE_VOLTAGE_V 中设置最小值外，还需要在此设置一个最大值。 最大值与 InstaSPIN 可识别的最小磁通相关。 可识别的最小磁通按如下公式进行计算：

最小磁通 (V/Hz) = USER_IQ_FULL_SCALE_VOLTAGE_V/USER_EST_FREQ_Hz/0.7

例如，如果电机的磁通为 0.001V/Hz（如果使用磁通值极低的小型低成本电机，此值为常规值）且估算器的工作频率为 20kHz，则可用于识别此电机的 USER_IQ_FULL_SCALE_VOLTAGE_V 最大值为：

USER_IQ_FULL_SCALE_VOLTAGE_V < 0.001 * 20000 / 0.7 = 28.57V

建议留出 30% 余量，以保证可以稳定识别。 因此，在以上示例中，建议 USER_IQ_FULL_SCALE_VOLTAGE_V 取值 20.0V。

5.1.3 IQ 满量程电流

IQ 满量程电流的作用与上述频率和电压的 IQ 满量程值相同，不过它还可用于电流反馈。 IQ 满量程电流用于将电流反馈按照标么值进行标准化。 此值必须大于任何可测量电流。

IQ 满量程电流必须大于任何可测量电流

例如，如果峰值电流为每相 8A，则 IQ 满量程电流值应设置为含 20-30% 余量的更大值，在本例中为 10A。

```
//! \brief Defines the full-scale current for the IQ variables, A
#define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
```

CAUTION

如果测量电流在任意时刻均大于 IQ 满量程电流，则可能软件中存在数值溢出条件。 请确保可测量电流小于 IQ 满量程电流值，以避免软件出现异常行为。 为了避免出现此类问题，用户必须确保 (USER_IQ_FULL_SCALE_CURRENT_A * 2) 始终大于通过 ADC 测量的电流。 此处使用“乘 2”因数是因为 USER_IQ_FULL_SCALE_CURRENT_A 参数的范围为零至最大振幅（峰值），而 USER_ADC_FULL_SCALE_ CURRENT_A 则为峰峰值。

按照以下指南操作可防止电流测量出现数值溢出：

```
(USER_IQ_FULL_SCALE_CURRENT_A * 2) >= USER_ADC_FULL_SCALE_CURRENT_A
```

5.1.4 最大电流

最大电流用于定义速度控制器的最大输出，这与 IQ 满量程电流不同，后者用于定义通过 ADC 转换器测得电流的标准化因数。 最大电流必须始终小于 IQ 满量程电流，因为最大电流只是软件限制，而 IQ 满量程电流代表最大硬件输入的最大软件表示。

*最大电流必须始终小于 **IQ** 满量程电流。*

最大电流的定义用于设置最大软件限制。 这表示可将速度控制器所需的最大电流钳制到最大电流定义。 例如，如果将最大电流定义设置为 4.2A，而速度控制器要求通过电流控制器提高转矩，则最大所需电流为 4.2A 或此定义中设置的任意值。 建议将最大电流设置为小于或等于电机制造商建议的最大电流以避免损坏电机。

最大电流小于或等于电机制造商建议的最大电流。

例如，Anaheim 电机（与 DRV8312 版本 D 开发板配套提供）的额定转矩为 21oz-in，转矩常数为 5oz-in/A，这会产生 4.2A 的额定电流，从而产生额定转矩。将 4.2A 设置为电机最大电流（峰值电流振幅），如以下代码所示：

```
#define USER_MOTOR_MAX_CURRENT (4.2)
```

请注意，在 user.h 中定义的最大电流不会提供硬件限制或过流保护。 也就是说，这不是硬件电流限制，而是软件限制，仅用于限制电流控制器的最大输入而不限制其输出。

图 5-1 显示了此 USER_MOTOR_MAX_CURRENT 在 InstaSPIN 中使用位置的图示。 如下图中所示，最大电流不是逐周期限制电流，而是在为电流控制器提供基准电流之前提供速度控制器积分部分输出饱和以及总体速度控制器输出饱和。

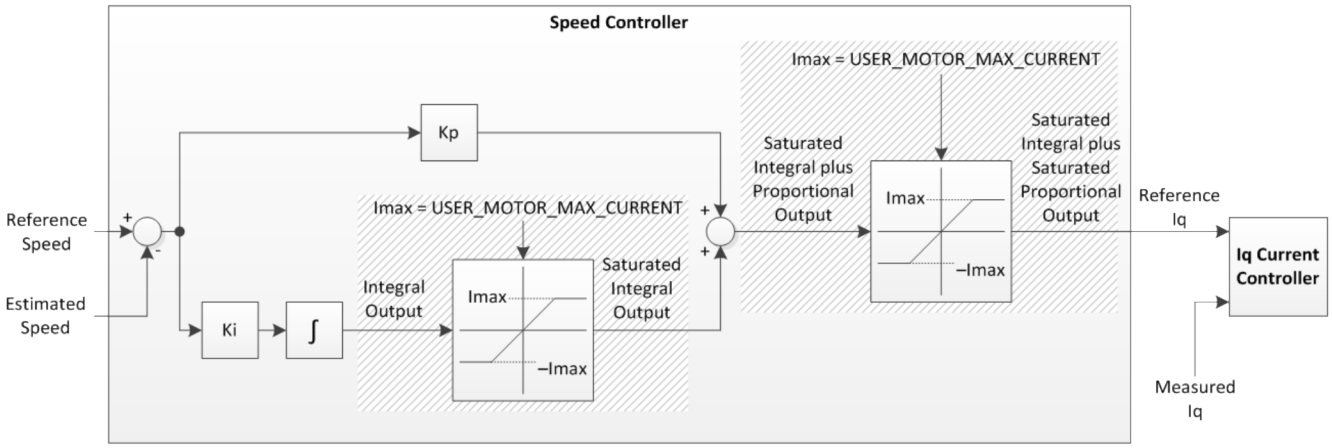


图 5-1. InstaSPIN 中的 USER_MOTOR_MAX_CURRENT

5.1.5 抽取率

利用抽取率，用户可以配置各个循环速率以满足相应的代码执行要求。 建议使用默认抽取率作为起始点。 用户必须验证是否满足实时调度，从而验证单次中断期间是否允许在 ISR 中执行所有软件。 只需在 ISR 中两 (2) 次切换 GPIO 引脚（一次在 ISR 开始执行时，一次在 ISR 结束执行时）并观察示波器即可完成上述操作。 如果不满足实时调度，则不能预测 InstaSPIN 性能。

*为使 **InstaSPIN** 性能一致，需要满足实时调度。*

以下是默认抽取率：

```
// Defines the number of pwm clock ticks per isr clock tick
// Note: Valid values are 1, 2 or 3 only
#define USER_NUM_PWM_TICKS_PER_ISR_TICK       (1)
// Defines the number of isr ticks (hardware) per controller clock tick (software)
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK      (1)
// Defines the number of controller clock ticks per estimator clock tick
#define USER_NUM_CTRL_TICKS_PER_EST_TICK      (1)
// Defines the number of controller clock ticks per current controller clock tick
#define USER_NUM_CTRL_TICKS_PER_CURRENT_TICK  (1)
// Defines the number of controller clock ticks per speed controller clock tick
#define USER_NUM_CTRL_TICKS_PER_SPEED_TICK    (10)
// Defines the number of controller clock ticks per trajectory clock tick
#define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK     (10)
```

如果中断期间不允许完成这些抽取率，请参见 9.1 节了解有关更改抽取率的更多信息。

5.1.6 系统频率

系统频率为 MCU 的时钟速率。 建议以可能的最高频率运行，以便以最快的速度执行代码。 有两个部分需要用户将系统频率设置为最大值。

*将 **MCU** 配置为最快 **CPU** 时钟以实现最佳实时性能。*

第一个部分在 user.h 文件中。 此 #define 语句可确保相应地计算所有定时模块的相关计算。

第二个部分在 hal.c 文件中使用 HAL_setParams 函数进行配置，请参见表 5-1 中的代码示例，针对 2806x 器件配置 PLL 以最大频率 90MHz 运行，针对 2805x 和 2802x 器件配置 PLL 以最大频率 60MHz 运行。

表 5-1. hal.c 配置 PLL

器件	最快系统频率	user.h	hal.c
2806x	90MHz	`#define USER_SYSTEM_FREQ_MHz (90)`	`HAL_setupPll(handle,PLL_ClkFreq_90_MHz)`
2805x	60MHz	`#define USER_SYSTEM_FREQ_MHz (60)`	`HAL_setupPll(handle,PLL_ClkFreq_60_MHz)`
2802x	60MHz	`#define USER_SYSTEM_FREQ_MHz (60)`	`HAL_setupPll(handle,PLL_ClkFreq_60_MHz)`

5.1.7 *PWM 频率*

PWM 频率在 **user.h** 文件中设置。 根据电机电感，某些电机需要比其它电机更高的 PWM 频率。 通常，为避免过多电流纹波，电机的电感越低，所需的 PWM 频率越高。 对于大多数电机，通常建议使用 20kHz，对于需要使用更高 PWM 频率的特殊情况将在本文档的后续章节中进行介绍。

*电机的电感越低，所需的 **PWM** 频率越高。*

以下代码示例显示如何在软件中设置 20kHz PWM 频率：

```
//! \brief Defines the Pulse Width Modulation (PWM) frequency, kHz
#define USER_PWM_FREQ_kHz (20.0)
```

5.1.8 *最大电压矢量*

最大电压矢量在 user.h 文件中设置，可用于设置 Id 和 Iq PI 电流控制器输出的最大幅度。 Id 和 Iq PI 电流控制器的输出为 Vd 和 Vq。 Vs、Vd 与 Vq 之间的关系为：

Vs = sqrt(Vd^2 + Vq^2)

Vq = sqrt(USER_MAX_VS_MAG^2 - Vd^2)

在此 FOC 控制器中，将 Vd 值设置为等于：

USER_MAX_VS_MAG*USER_VD_MAG_FACTOR。

USER_MAX_VS_MAG_PU 最大可达 1.0（全局 IQ 格式），如果未使用电流重构，则可达 _IQ(1.0)。 有关更多讨论和示例，请参见实验 10a-x。

```
//! \brief Defines the voltage vector magnitude
#define USER_MAX_VS_MAG_PU (1.0)
```

除此最大电压矢量幅度定义外，还可以更改控制器对象的成员，从而更改电流控制器的输出（即空间矢量调制 (SVM) 的输入）。 需要注意的是，虽然最大电压矢量幅度在 user.h 中定义的最大值为 1.0（或 100%），但是 SVM 的输入可达 4.0/3.0 = 1.3333（允许过调制）。 SVM 的输入大于 1.0 即为过调制区域。 输入为 2/SQRT(3) = 1.1547 时，正弦波波峰的占空比接近 100%。 输入为 1.3333 时，SVM 生成器将生成梯形波形。 以下代码示例将电流控制器的输出更改为 1.3333，从而允许最大过调制：

```
// Set the maximum current controller output for the Iq and Id current
// controllers to enable overmodulation.
CTRL_setMaxVsMag_pu(ctrlHandle, _IQ(pUserParams->maxVsMag_pu));
```

表 5-2 描述了最大 SVM 输入的不同范围以及对于空间矢量调制 (SVM) 的意义。

表 5-2. 最大 SVM 输入范围

#define USER_MAX_VS_MAG_PU (value)	CTRL_setMaxVsMag_pu(handle, value)	峰值时 EPWM 的占空比	波形类型	电流重构
(1.0)	IQ(1.0)	86.6%	理想正弦波	不需要
$\frac{2}{\sqrt{3}} = 1.1547$	_IQ(2/SQRT(3))	100.0%	准正弦波	需要
$\frac{4}{3} = 1.3333$	_IQ(1.3333)	100.0%	梯形波	需要

在过调制区运行时，电压波形根据应用过调制的程度开始从正弦波向梯形波转换。 随着电机运行逐渐深入过调制区，预期会出现电机振动和转矩纹波。 SVM 模块在表 8-1 中有详细介绍。

*可产生 **100%** 峰值占空比的最大电压幅值要求采用三分流电流测量。*

由最大电压幅值产生的实际占空比范围取决于电流采样所用的分流电阻器数量。 如果应用要求占空比为 100%，则用户必须使用三个分流电阻器对电机的相电流进行采样。 如果只有两个分流电阻器，则会将占空比限制为小于 100%。 使用两个分流电阻器时的最大占空比取决于 OPAMP 参数和布局本身。 有关选择正确的电流反馈组件的详细信息，请参见Chapter 17。

5.2 硬件必要条件

为了正确识别电机以及使用 InstaSPIN 高效运行电机，需要正确设置几个硬件相关参数。 以下参数即为硬件相关参数，下文会对各参数进行详细介绍：

- 电流反馈增益 - 最大 ADC 输入范围
- 电流反馈极性 - 匹配软件与硬件极性
- 电压反馈
- 电压滤波器极
- 分流电阻器数量
- 死区时间配置
- 模拟输入配置
- PWM 输出配置

以下章节将分别介绍各个参数。

5.2.1 电流反馈增益

为了测量双向电流（即正电流和负电流），以下电路需要使用 1.65V 基准电压。此电压在 3.3V 系统中通常不会提供，但可通过电压跟随器轻松生成。图 5-2 为通过 3.3V 系统中提供的 3.3V 输入生成 1.65V 基准电压的电路示例。 对于连接 1.65V 的后续电路，假定使用此电路。

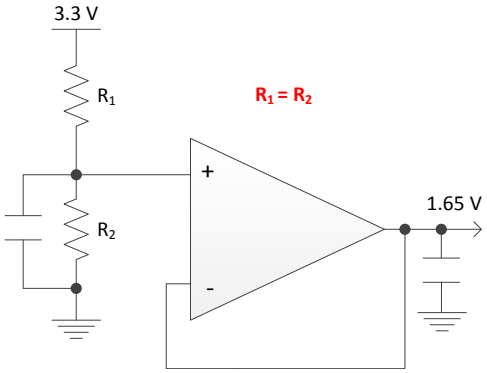


图 **5-2.** 通过 **3.3V** 输入生成 **1.65V** 基准电压电路示例

图 5-3 显示用于电流测量的典型差分放大器配置。

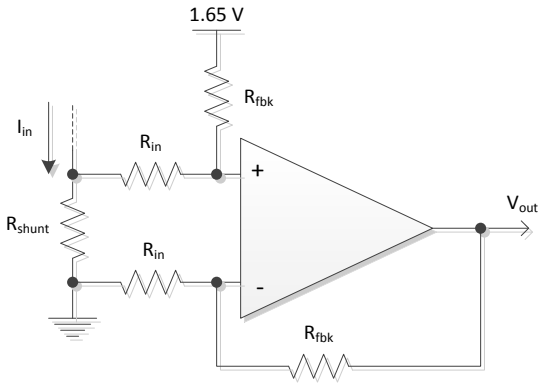


图 **5-3.** 典型差分放大器电路

此电路的传输函数由 公式 3 给定。

$$V_{out} = 1.65 + I_{in} \times R_{shunt} \times \frac{R_{fbk}}{R_{in}} \tag{3}$$

为了演示此电路中的某些值，下面以定义的最大相电流为 10A 的电机为例。在本例中，微处理器要测量的最大电流为 ±10A，使用此电路可生成最大电压 1.65V (±1.65V) 来支持 3.3V ADC 输入范围。

对于 10A 电流的最差情况，可考虑使用 0.01Ω 分流电阻器。

$$\frac{R_{fbk}}{R_{in}} = \frac{V_{out} - 1.65}{I_{in} \times R_{shunt}} = \frac{3.3 - 1.65}{10 \times 0.01} = 16.5 \tag{4}$$

现在，如果使用 1.0kΩ 的输入电阻，则可根据输入电阻和所需比值计算反馈电阻。

$$R_{fbk} = 16.5 \times R_{in} = 16.5 k\Omega \tag{5}$$

根据计算的电阻值可生成如图 5-4 所示电路，其中提供电压范围 0-3.3V 表示测量的 ±10A 相电流。

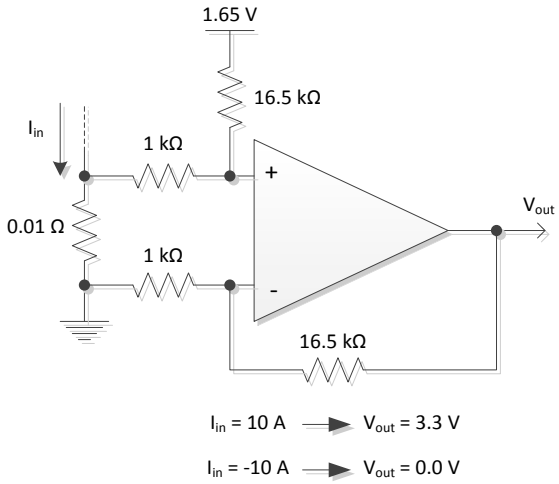


图 **5-4.** 计算电阻值电路

如本例所示，微处理器可测量的最大峰峰值电流为 20A，峰峰值为 ±10A。以下代码段显示如何在 user.h 中该值：

```
//! \brief Defines the maximum current at the AD converter
#define USER_ADC_FULL_SCALE_CURRENT_A (20.0)
```

OPAMP 转换率对于电流测量质量起重要作用。 有关详细信息，请参见Chapter 17。

5.2.2 电流反馈极性

正确的电流反馈极性也很重要，因为这样才能确保微处理器精确测量电流。

5.2.2.1 正反馈

在此硬件配置中，分流电阻器的负引脚接地，同时与运算放大器的**反向**引脚连接。图 5-5 显示硬件中的正极性及其软件配置。 为了在软件中获得电流反馈正极性，需要对突出显示的符号进行配置。 函数 HAL_updateAdcBias 位于 **hal.h** 文件中。

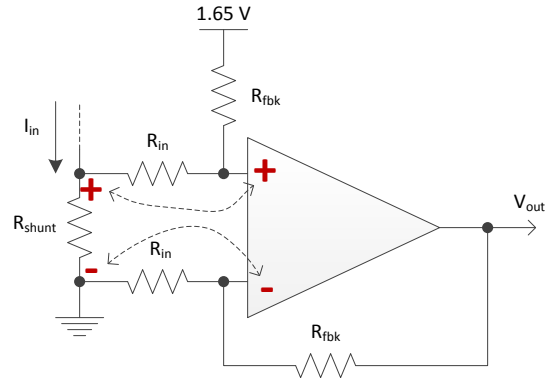


图 **5-5.** 正反馈

```
//! \brief     Updates the ADC bias values
```

```
//! \param[in] handle  The driver (HAL) handle
inline void HAL_updateAdcBias(HAL_Handle handle)
{
  uint_least8_t cnt;
  HAL_Obj *obj = (HAL_Obj *)handle;
  _iq bias;


  // update the current bias
  for(cnt=0;cnt<HAL_getNumCurrentSensors(handle);cnt++)
    {
     bias = HAL_getBias(handle,HAL_SensorType_Current,cnt);

     bias -= OFFSET_getOffset(obj->offsetHandle_I[cnt]);

     HAL_setBias(handle,HAL_SensorType_Current,cnt,bias);
    }

  // update the voltage bias
  for(cnt=0;cnt<HAL_getNumVoltageSensors(handle);cnt++)
    {
     bias = HAL_getBias(handle,HAL_SensorType_Voltage,cnt);

     bias += OFFSET_getOffset(obj->offsetHandle_V[cnt]);

     HAL_setBias(handle,HAL_SensorType_Voltage,cnt,bias);
    }

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

5.2.2.2 负反馈

另一方面，图 5-6 表示负反馈。 在此硬件配置中，分流电阻器的负引脚接地，同时与运算放大器的**非反向**引脚连接。 配置负反馈所需的代码中显示了突出显示的符号，为了在软件中获得负反馈，需要对该符号进行配置。 函数 HAL_updateAdcBias 位于 **hal.h** 文件中。

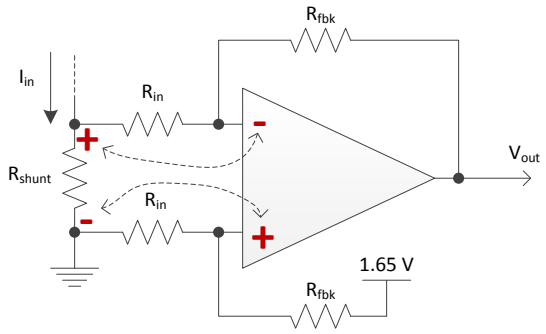


图 **5-6.** 负反馈

```
//! \brief Updates the ADC bias values
//! \param[in] handle The driver (HAL) handle
inline void HAL_updateAdcBias(HAL_Handle handle)
{
  uint_least8_t cnt;
  HAL_Obj *obj = (HAL_Obj *)handle;
  _iq bias;

  // update the current bias
```

```
  for(cnt=0;cnt<HAL_getNumCurrentSensors(handle);cnt++)
    {
     bias = HAL_getBias(handle,HAL_SensorType_Current,cnt);

     bias += OFFSET_getOffset(obj->offsetHandle_I[cnt]);

     HAL_setBias(handle,HAL_SensorType_Current,cnt,bias);
    }

  // update the voltage bias
  for(cnt=0;cnt<HAL_getNumVoltageSensors(handle);cnt++)
    {
     bias = HAL_getBias(handle,HAL_SensorType_Voltage,cnt);

     bias += OFFSET_getOffset(obj->offsetHandle_V[cnt]);

     HAL_setBias(handle,HAL_SensorType_Voltage,cnt,bias);
    }

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

5.2.3 电压反馈

FAST 估算器中需要使用电压反馈，以便在更宽的速度范围内实现最佳性能。 其它算法无法准确表示电压相位，需要依赖软件变量。 在 FAST 中，相电压直接从电机相位测量而不需通过软件估算。 正因如此，电压反馈的硬件设置是 InstaSPIN 运行和电机识别的另一个必要条件。 此软件值 (USER_ADC_FULL_SCALE_VOLTAGE_V) 取决于检测电机相位电压反馈的电路。图 5-7 是基于电阻分压器的电压反馈电路的示例。

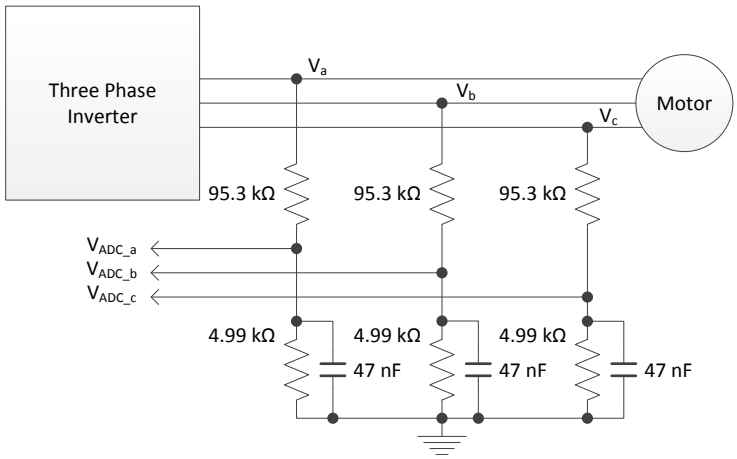


图 **5-7.** 电压反馈电路

在本例中，微处理器可测量的最大相电压反馈可按以下公式进行计算，ADC 输入的最大电压为 3.3V：

$$V_a^{max} = V_{ADC\_a}^{max} \times \frac{(4.99k\Omega + 95.3k\Omega)}{4.99k\Omega} = 3.3\,V \times \frac{(4.99k\Omega + 95.3k\Omega)}{4.99k\Omega} = 66.3\,V \quad (6)$$

使用上述电压反馈电路时，需要在 user.h 中完成以下设置：

```
//! \brief Defines the maximum voltage at the input to the AD converter
#define USER_ADC_FULL_SCALE_VOLTAGE_V (66.3)
```

如果考虑针对此值留 20-30% 余量，则建议系统最大输入电压介于 66.3*0.7 = 46.4V 与 66.3*0.8 = 53V 之间，因此，此电压反馈电阻分压器是 48V 电机的理想选择。

下面介绍不同标称电压的示例。 如果要驱动的电机的标称电压为 24V，则需要修改电压反馈电路，使 ADC 分辨率针对测量电压达到最大值。 遵照关于余量的相同建议，考虑标称电压 24V 和余量值 30%。 这样可得到 USER_ADC_FULL_SCALE_VOLTAGE_V（24*1.3 = 31.2V），如公式 7 中所示，其中一个电阻器固定，仅剩下一个可变电阻器。

$$V_a^{max} = 31.2\,V = V_{ADC_a}^{max} \times \frac{(4.99k\Omega + R)}{4.99k\Omega} = 3.3\,V \times \frac{(4.99k\Omega + R)}{4.99k\Omega}$$

$$R = \frac{31.2\,V \times 4.99k\Omega}{3.3\,V} - 4.99k\Omega = 42.2k\Omega \quad (7)$$

我们可据此得到最大电压 31.2V，配置如下：

```
//! \brief Defines the maximum voltage at the input to the AD converter
#define USER_ADC_FULL_SCALE_VOLTAGE_V (31.2)
```

图 5-8 中显示了含上述值的电压反馈电路。

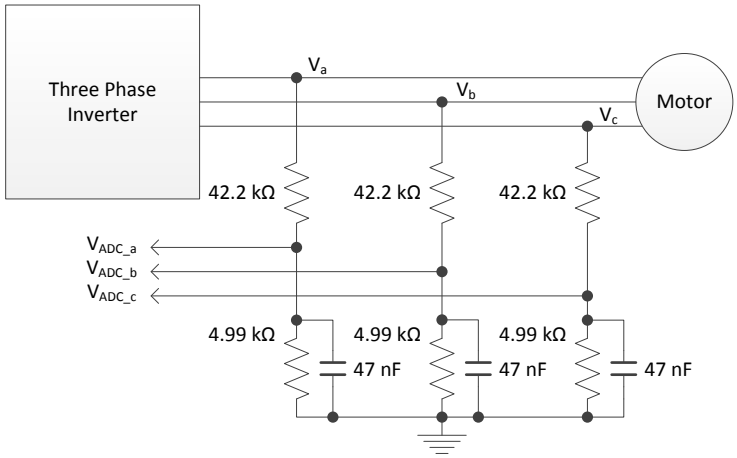


图 **5-8.** 电压反馈电路

5.2.4 电压滤波器极

FAST 估算器中需要使用电压滤波器极，以便准确检测电压反馈。 滤波器的电压应足够低，以便能够过滤 PWM 信号，同时允许高速电压反馈信号通过滤波器。

通常，使用几百 Hz 的截止频率便足以过滤掉 10 至 20kHz 的 PWM 频率。 只有在运行超高速电机时生成几 kHz 相电压频率的情况下，才需更改硬件滤波器。

本例中以 DRV8312 版本 D 开发板配套提供的电机（即 Anaheim PMSM 电机）为例，最大速度约为 8000RPM，极对数为 4。 提供的电压频率为 8000/60*4 = 533.3Hz。 对于此电机和速度，采用上述频率近似值 533Hz 的电压滤波器的截止频率应足够大。

根据 DRV8312 版本 D 硬件，以图 5-8 中所示的同一电压反馈电路为例。

滤波器极设置可按如下公式计算：

$$F_{filter\_pole} = \frac{1}{2 \times \pi \times R_{parallel} \times C} = \frac{1}{2 \times \pi \times \left(\frac{95.3k\Omega \times 4.99k\Omega}{95.3k\Omega + 4.99k\Omega} \right) \times 47nF} = 714.15Hz \quad (8)$$

以下代码示例显示如何在 user.h 中定义此参数：

```
//! \brief Defines the analog voltage filter pole location, Hz
#define USER_VOLTAGE_FILTER_POLE_Hz (714.15)
```

Tips and Tricks:

- 为获得最佳结果，需保持滤波器极 > 200Hz
- 滤波器极必须 > IQ_FULLSCALE_FREQUENCY_Hz / 4.0，以避免出现数值饱和。

5.2.5 分流电阻器数量

选择硬件配置的一大重点是要使用的分流电阻器数量。 此数字最终由 Clarke 变换使用，可将三相系统转换为两相系统。 如果所有相位均通过一个分流电阻器从底部的晶体管接地，则使用三个分流电阻器，如图 5-9 所示。

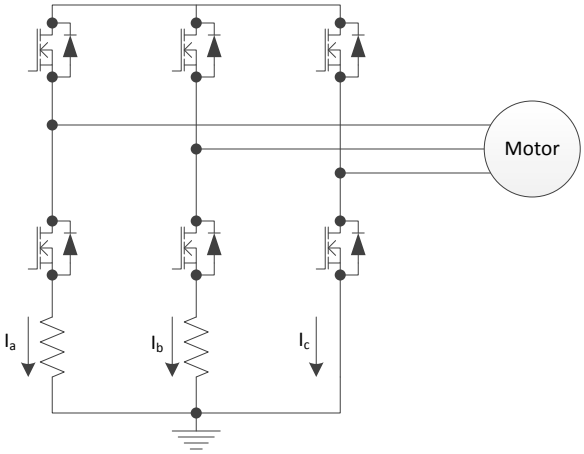


图 **5-9.** 分流电阻器

如果此配置存在，则用户应在软件中定义三个分流电阻器以获得最佳结果。 以下代码示例显示如何配置软件以使用三个分流电阻器：

```
//! \brief Defines the number of current sensors used
#define USER_NUM_CURRENT_SENSORS (3)
```

如果硬件使用两个分流电阻器测量电流，则必须将软件配置为只使用两个分流电阻器，如下所示：

```
//! \brief Defines the number of current sensors used
#define USER_NUM_CURRENT_SENSORS (2)
```

有关 InstaSPIN 的分流电阻器测量要求的详细信息，请参见Chapter 17。

5.2.6 死区时间配置

必须根据使用的硬件正确配置死区时间，以避免击穿逆变器高侧和低侧晶体管（请参见图 5-10）。 有关 EPWM 模块和死区时间配置的详细信息，请参见 MCU 技术参考手册。

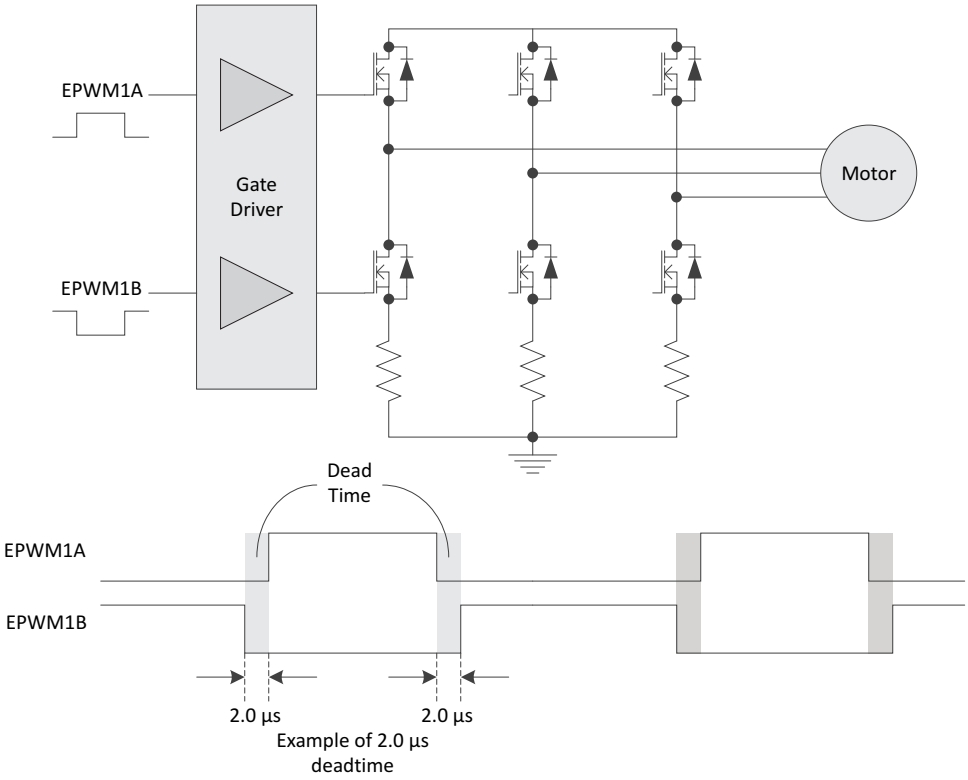


图 **5-10.** 死区时间配置

死区时间取决于使用的晶体管和栅极驱动器，可按照以下代码示例所示进行配置，假定需要的死区时间为 2.0µs，以 90MHz 的系统频率在 F2805xF 和 F2806xF 器件上运行。

```
//! \brief Defines the system clock frequency, MHz
//!
#define USER_SYSTEM_FREQ_MHz (90)

//! \brief Defines the PWM dead band in terms of the number of system clocks
//!
#define USER_PWM_DBCNT_INIT_STATE \
        ((uint16_t)(2.0 * (float_t)USER_SYSTEM_FREQ_MHz))
```

以 60MHz 系统频率在 F2802xF 器件上运行：

```
//! \brief Defines the system clock frequency, MHz
//!
#define USER_SYSTEM_FREQ_MHz (60)
//! \brief Defines the PWM dead band in terms of the number of system clocks
//!
#define USER_PWM_DBCNT_INIT_STATE \
        ((uint16_t)(2.0 * (float_t)USER_SYSTEM_FREQ_MHz))
```

触发区和比较器可用于保护硬件，防止出现过流或过压条件，具体取决于使用的特定硬件配置，而最终用户负责使用 EPWM 和 ADC 模块的所有可用功能来保护硬件。 此外，也可以借助 EPWM 模块的灵活性来实现死区时间的备用实施方案，但本文档的范围仅限于 InstaSPIN 软件的功能，并不涵盖所有 EPWM 实施方案。

5.2.7 模拟输入配置

模拟引脚必须在软件中配置。 为了演示如何进行配置，下面以图 5-11 为例，表示在使用 F2806xF 器件和 DRV8312 版本 D 开发板时如何连接模拟引脚。

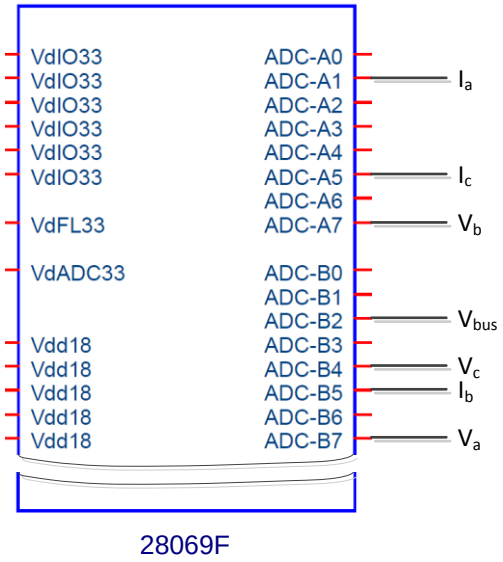


图 **5-11.** 模拟连接

有关 ADC 配置的详细信息，请参见《*TMS320x2806x Piccolo 技术参考指南*》（文献编号 SPRUH18）、《*TMS320x2805x Piccolo 技术参考指南*》（文献编号 SPRUHE5）以及《*TMS320F2802x，TMS320F2802xx Piccolo 系统控制和中断参考指南*》（文献编号 SPRUFN3）。 有关特定封装相关引脚的详细信息，请参见数据手册《*TMS320F2806x Piccolo 微控制器*》（文献编号 SPRS698）、《*TMS320F2805x Piccolo 微控制器*》（文献编号 SPRS797）以及《*TMS320F2802x，TMS320F2802xx Piccolo 微控制器*》（文献编号 SPRS523）。

以下代码示例显示了如何配置 ADC 输入引脚以表示上图。 用于执行实际配置的代码行在代码示例中会突出显示。 此代码是 **hal.c** 文件中 HAL_setupAdcs 函数的一部分。

```
ADC_setSocChanNumber (obj->adcHandle,ADC_SocNumber_0,ADC_SocChanNumber_A1);
ADC_setSocTrigSrc(obj->adcHandle,ADC_SocNumber_0,ADC_SocTrigSrc_EPWM1_ADCSOCA);
ADC_setSocSampleDelay(obj->adcHandle,ADC_SocNumber_0,ADC_SocSampleDelay_9_cycles);

ADC_setSocChanNumber(obj->adcHandle,ADC_SocNumber_1,ADC_SocChanNumber_B5);
ADC_setSocTrigSrc(obj->adcHandle,ADC_SocNumber_1,ADC_SocTrigSrc_EPWM1_ADCSOCA);
ADC_setSocSampleDelay(obj->adcHandle,ADC_SocNumber_1,ADC_SocSampleDelay_9_cycles);

ADC_setSocChanNumber(obj->adcHandle,ADC_SocNumber_2,ADC_SocChanNumber_A5);
ADC_setSocTrigSrc(obj->adcHandle,ADC_SocNumber_2,ADC_SocTrigSrc_EPWM1_ADCSOCA);
ADC_setSocSampleDelay(obj->adcHandle,ADC_SocNumber_2,ADC_SocSampleDelay_9_cycles);

ADC_setSocChanNumber(obj->adcHandle,ADC_SocNumber_3,ADC_SocChanNumber_B7);
ADC_setSocTrigSrc(obj->adcHandle,ADC_SocNumber_3,ADC_SocTrigSrc_EPWM1_ADCSOCA);
ADC_setSocSampleDelay(obj->adcHandle,ADC_SocNumber_3,ADC_SocSampleDelay_9_cycles);
```

```
ADC_setSocChanNumber(obj->adcHandle,ADC_SocNumber_4,ADC_SocChanNumber_A7);
ADC_setSocTrigSrc(obj->adcHandle,ADC_SocNumber_4,ADC_SocTrigSrc_EPWM1_ADCSOCA);
ADC_setSocSampleDelay(obj->adcHandle,ADC_SocNumber_4,ADC_SocSampleDelay_9_cycles);

ADC_setSocChanNumber(obj->adcHandle,ADC_SocNumber_5,ADC_SocChanNumber_B4);
ADC_setSocTrigSrc(obj->adcHandle,ADC_SocNumber_5,ADC_SocTrigSrc_EPWM1_ADCSOCA);
ADC_setSocSampleDelay(obj->adcHandle,ADC_SocNumber_5,ADC_SocSampleDelay_9_cycles);

ADC_setSocChanNumber(obj->adcHandle,ADC_SocNumber_6,ADC_SocChanNumber_B2);
ADC_setSocTrigSrc(obj->adcHandle,ADC_SocNumber_6,ADC_SocTrigSrc_EPWM1_ADCSOCA);
ADC_setSocSampleDelay(obj->adcHandle,ADC_SocNumber_6,ADC_SocSampleDelay_9_cycles);
```

5.2.8 PWM 输出配置

图 5-12 描述了 PWM 引脚的配置，其中电机相位 A 由 EPWM1A/EPWM1B 驱动，电机相位 B 由 EPWM2A/EPWM2B 驱动，电机相位 C 由 EPWM3A/EPWM3B 驱动。 此硬件配置与 DRV8312 版本 D 开发板相同，可作为本文档中的示例。

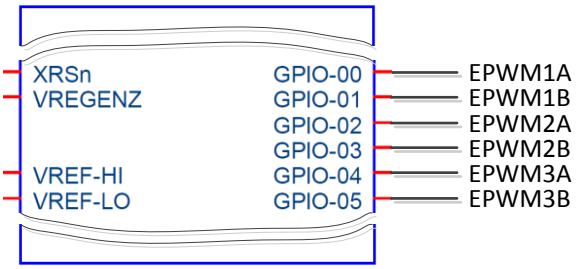


图 **5-12. PWM** 引脚配置

有关 EPWM 配置的详细信息，请参见《*TMS320x2806x Piccolo 技术参考指南*》（文献编号 SPRUH18）、《*TMS320x2805x Piccolo 技术参考指南*》（文献编号 SPRUHE5）以及《*TMS320F2802x，TMS320F2802xx Piccolo 系统控制和中断参考指南*》（文献编号 SPRUFN3）。 有关特定封装相关引脚的详细信息，请参见数据手册《*TMS320F2806x Piccolo 微控制器*》（文献编号 SPRS698）、《*TMS320F2805x Piccolo 微控制器*》（文献编号 SPRS797）以及《*TMS320F2802x，TMS320F2802xx Piccolo 微控制器*》（文献编号 SPRS523）。

以下代码示例显示如何针对上图设置 PWM 引脚。 此代码示例是驱动程序对象 (HAL_init) 的初始化部分，包含在 **hal.c** 文件中。 为使用前三对 EPWM 而初始化句柄后，可针对初始化的句柄完成其它 PWM 配置：

```
// initialize PWM handle
obj->pwmHandle[0] = PWM_init((void *)PWM_ePWM1_BASE_ADDR,sizeof(PWM_Obj));
obj->pwmHandle[1] = PWM_init((void *)PWM_ePWM2_BASE_ADDR,sizeof(PWM_Obj));
obj->pwmHandle[2] = PWM_init((void *)PWM_ePWM3_BASE_ADDR,sizeof(PWM_Obj));
```

突出显示的文本指示以下对象之间的对应关系：

- pwmHandle[0] = EPWM1A/EPWM1B，用于驱动电机相位 A
- pwmHandle[1] = EPWM2A/EPWM2B，用于驱动电机相位 B
- pwmHandle[2] = EPWM3A/EPWM3B，用于驱动电机相位 C。

电机识别和状态图

本节将讨论电机识别功能：比较 ACIM 和 PMSM 识别过程的差异、了解每个过程的详细信息并探讨需要非典型步骤的特殊情况。

有些无传感器的电机控制应用需要依靠电机型号。 此电机型号要求您了解某些参数才能精确地表示电机。 此型号随后用于运行估算器，进而将提供未知变量，如转子磁通角度或速度。 在电机参数未知的情况下，或如果参数随时间变化，将发生问题。 通常，定义明确的电机用于最终应用，但在有些情况下，电机尚未定义，或者产品生命周期中使用了多个电机。

确定并跟踪电机参数的一种方法是使用测量电机参数的软件。 尽管识别电机不是所有应用的强制要求，但是这样可以提供简单、更好的开箱即用体验 - 在没有传感器的情况下轻松运行任何给定的电机。 市面上的其它算法要求提前采用密集调整过程，甚至是在电机闭环运行之前。

以下各节详细介绍了采用 InstaSPIN 解决方案的电机识别过程。 为确保介绍的算法和相关步骤能够成功识别大量的电机类型，我们投入了大量精力。 但不能确保所介绍的算法和步骤始终能够成功识别所有电机或电机类型。 故障排除部分提到了几个需要特别注意的特定电机类型。

Topic **Page**

6.1 InstaSPIN 电机识别

图 6-1 中突出显示的块与 InstaSPIN 电机识别功能相关。

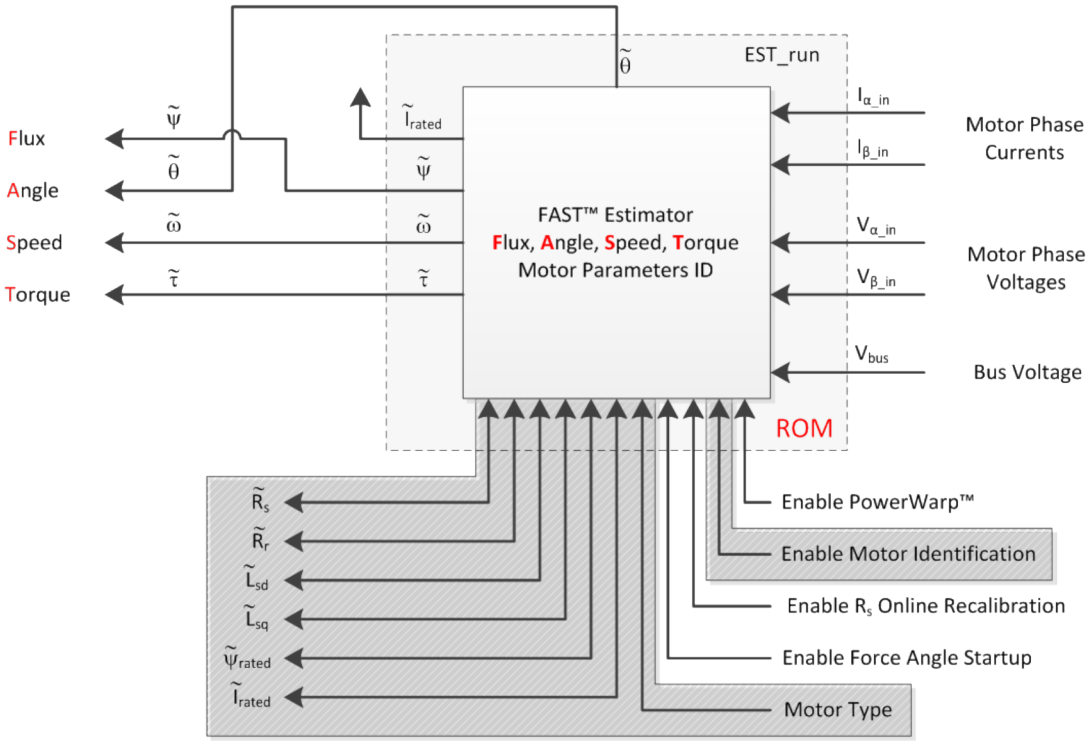


图 6-1. InstaSPIN 电机识别组成部分

电机识别是 InstaSPIN-FOC 的一项附加功能，借此功能可以识别估算器在无传感器条件下闭环运行所需的参数。 如果用户已熟知电机参数，则可以选择是否进行电机识别。 凭借 InstaSPIN 的电机识别功能，即使电机参数未知，用户也可使电机达到最佳运行性能。 如果电机已知或之前已经过识别，则所需电机参数会记录到头文件中，因此可以选择是否运行 InstaSPIN 的电机识别功能。 user.h 即为类似的头文件。 以下示例显示了绕过电机识别时所需的 PMSM 电机参数：

```
#if (USER_MOTOR == User_PMSM)
#define USER_MOTOR_Rs (2.83)
#define USER_MOTOR_Ls_d (0.0115)
#define USER_MOTOR_Ls_q (0.0135)
#define USER_MOTOR_RATED_FLUX (0.502)
```

以下示例显示了绕过电机识别时所需的 ACIM 电机参数：

```
#elif (USER_MOTOR == User_ACIM)
#define USER_MOTOR_Rr (5.5)
#define USER_MOTOR_Rs (10.7)
#define USER_MOTOR_Ls_d (0.053)
#define USER_MOTOR_Ls_q USER_MOTOR_Ls_d
#define USER_MOTOR_MAGNETIZING_CURRENT (1.4)
```

有关所需 PMSM 和 ACIM 电机参数的详细信息，请参见Chapter 4。

InstaSPIN 完整执行和最小执行时均可运行电机识别功能，请参见图 6-2 和图 6-3。 如果在 InstaSPIN 最小执行时运行电机识别，用户必须将磁场定向控制 (FOC) 块包含在 InstaSPIN 开源库内。

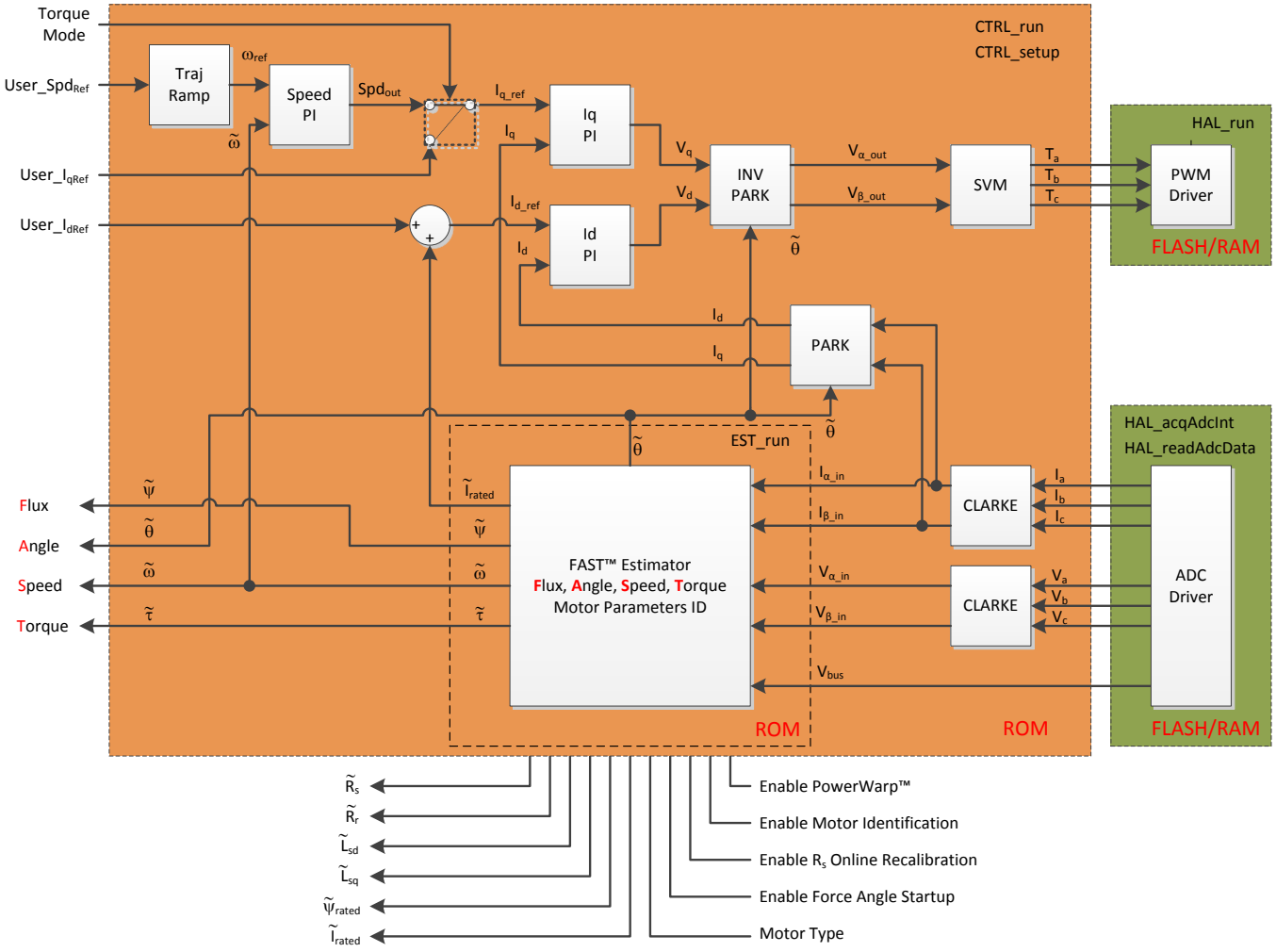


图 6-2. InstaSPIN-FOC 完整执行（仅限 F2805xF、F2805xM、F2806xF 和 F2806xM 器件）

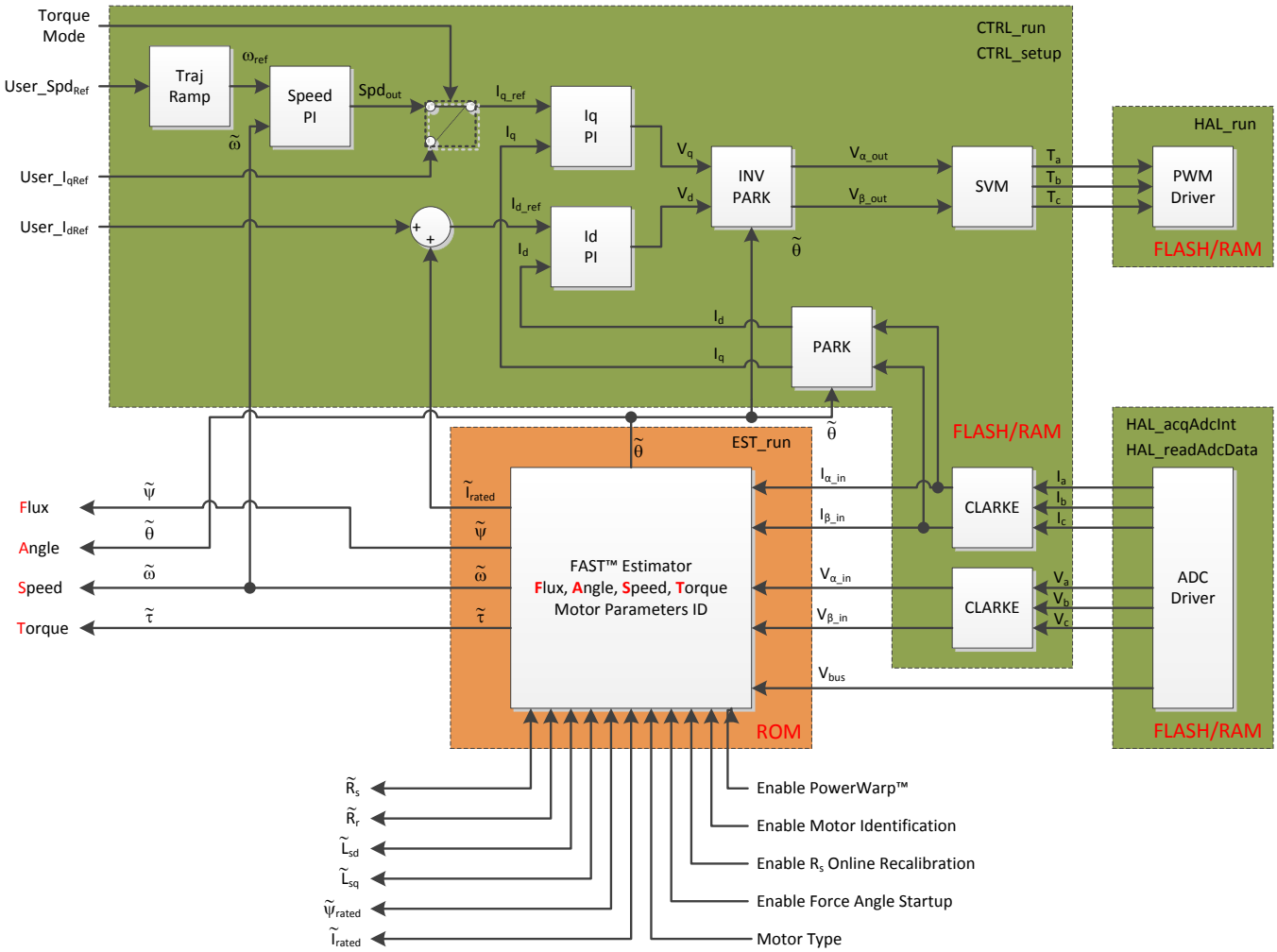


图 **6-3. InstaSPIN-FOC** 最小执行（**F2802xF**、**F2805xF**、**F2805xM**、**F2806xF** 和 **F2806xM** 器件）

6.2 电机识别过程概述

6.2.1 控制器 (CTRL) 状态机

表 6-1 汇总了图 6-4 中显示的所有状态并对各个状态进行了简要说明。 本文档中会在说明详细的电机识别过程时对此给出更加详细的介绍。

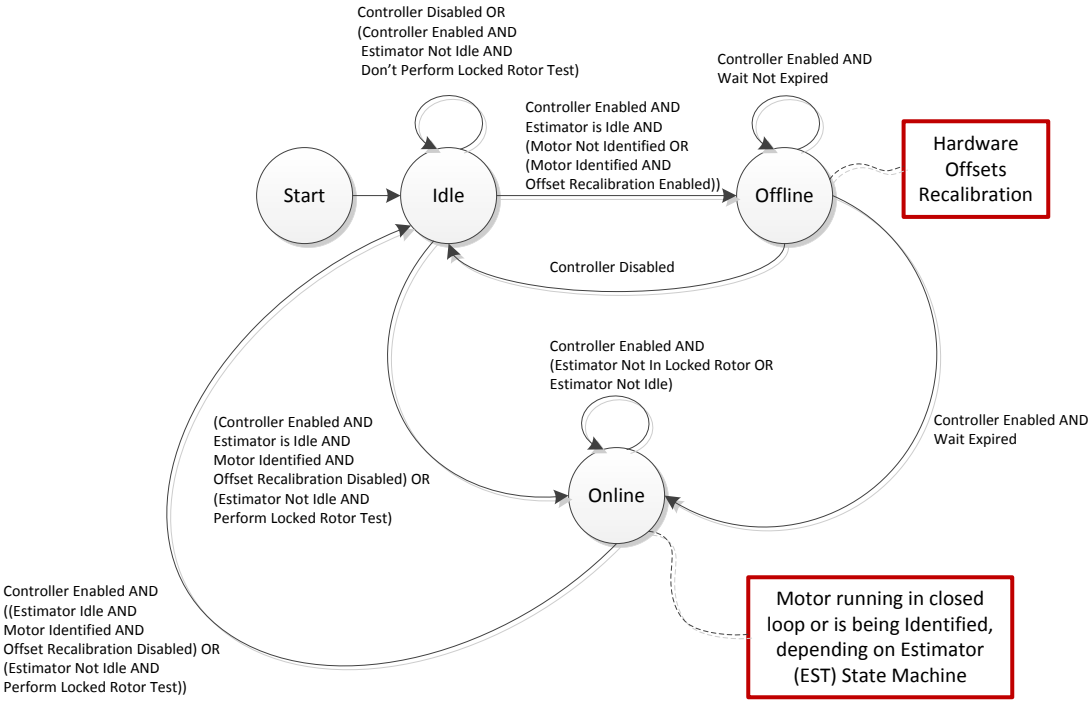


图 **6-4.** 控制器 **(CTRL)** 状态图

表 **6-1.** 控制器 **(CTRL)** 状态

控制器状态	简要说明
启动	启动状态仅显示为整个状态机的起点。 但它并不实际存在于控制器状态机中。
空闲 **CTRL_State_Idle**	当控制器等待用户输入以启动时会出现该状态。 如果识别交流感应电机，当系统等待用户输入以启动锁定转子测试时也会出现此状态。
离线 **CTRL_State_OffLine**	硬件偏移校准在控制器的这一状态下完成
在线 **CTRL_State_OnLine**	电机正在闭环运行或正在被识别。 当控制器 (CTRL) 状态机处于在线状态时，整个估算器 (EST) 状态机处于运行状态

表 6-2 汇总了图 6-4 中显示的所有状态转换条件。

表 **6-2.** 控制器 **(CTRL)** 状态图的状态转换

条件	简要说明

控制器已禁用	此条件在控制器尚未启用时出现，可通过以下指令进行检查：
	`if(CTRL_getFlag_enableCtrl(ctrlHandle)== FALSE)`
	另外，也可随时使用用户代码调用以下含有指定参数的函数，以此来禁用控制器：
	`CTRL_setFlag_enableCtrl(ctrlHandle,FALSE);`

控制器已启用	此条件在控制器已启用时出现，可通过以下指令进行检查：
	`if(CTRL_getFlag_enableCtrl(ctrlHandle)== TRUE)`
	可通过以下指令启用控制器：
	`CTRL_setFlag_enableCtrl(ctrlHandle,TRUE);`

估算器不空闲	此条件在估算器处于非空闲状态时出现。 可通过以下指令进行检查： `if(EST_getState(obj->estHandle)!=EST_State_Idle)`
估算器空闲	此条件在估算器处于空闲状态时出现。 可通过以下指令进行检查： `if(EST_getState(obj->estHandle)==EST_State_Idle)`
不执行锁定转子测试	此条件在估算器状态机内部进行检查，不可通过用户代码公开访问。 此条件仅在电机类型设置为感应电机时进行内部检查。
执行锁定转子测试	与上述“不执行锁定转子测试”的说明相同

表 6-2. 控制器 (CTRL) 状态图的状态转换 (continued)

条件	简要说明
电机未识别	当电机尚未被识别或用户电机的参数尚未加载到控制器对象中时，此条件为真。 “电机未识别”条件可通过以下示例进行检查： `if(EST_isMotorIdentified(obj->estHandle)== FALSE)`
电机已识别	电机识别完成后或用户电机的参数加载到控制器后，电机已识别条件为真。 此条件也可通过以下示例进行检查： `if(EST_isMotorIdentified(obj->estHandle)==TRUE)`
偏移重校准已启用	当硬件偏移重校准已启用时，此条件为真。 偏移重校准默认为启用状态。 此条件可通过以下指令示例进行检查： `if(CTRL_getFlag_enableOffset(ctrlHandle)==TRUE)`
偏移重校准已禁用	当硬件偏移重校准已禁用时，此条件为真。 该条件可通过以下示例进行检查： `if(CTRL_getFlag_enableOffset(ctrlHandle)==FALSE)`

等待未超时	此为内部条件，在执行偏移重校准时进行检查。 偏移重校准所耗时间通过 user.c 文件中的以下指令定义：
	`pUserParams->ctrlWaitTime[CTRL_State_OffLine] =`
	`(uint_least32_t)( 5.0 * USER_CTRL_FREQ_Hz);`
	其中，USER_CTRL_FREQ_Hz 在 user.h. 中定义。

估算器未处于锁定转子测试状态	此条件在估算器状态机未处于锁定转子测试状态时出现。 用户可通过使用以下指令示例检查此条件： `if(EST_getState(obj->estHandle) !=EST_State_LockRotor);`
等待已超时	与时间有关的状态转换基于内部计数器，而不是基于 user.c 中对应等待时间中存储的值： `uint_least32_t ctrlWaitTime[CTRL_numStates];`

6.2.2 估算器 (EST) 状态机

表 6-3 汇总了图 6-5 中显示的所有状态并对各个状态进行了简要说明。 本文档中会在说明详细的电机识别过程时对此给出更加详细的介绍。

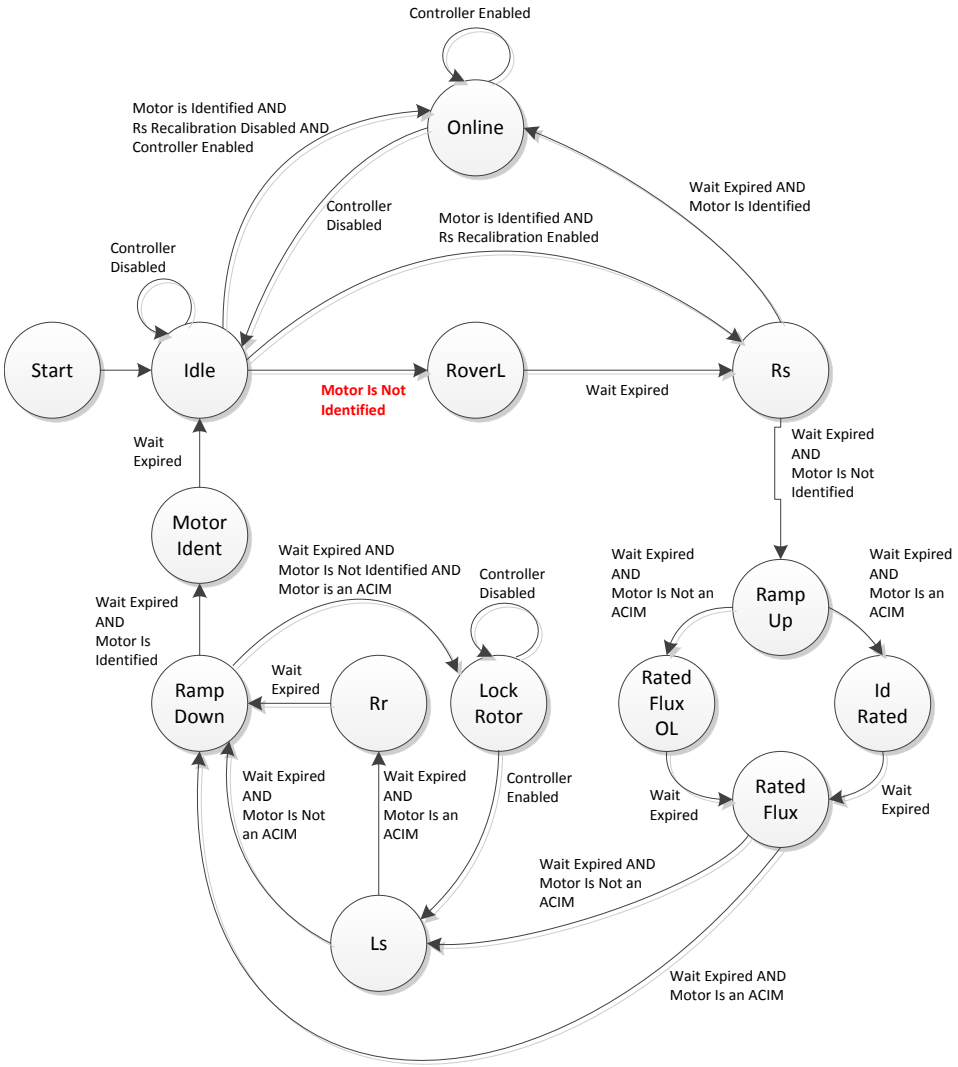


图 **6-5.** 估算器 **(EST)** 状态图

表 **6-3.** 估算器 **(EST)** 状态

估算器状态	简要说明
启动	启动状态仅显示为整个状态机的起点。 但它并不实际存在于估算器状态机中。
空闲 **EST_State_Idle**	在空闲状态期间，估算器状态机不执行任何代码。 只是等待控制器状态机更改估算器的状态。
RoverL **EST_State_RoverL**	估算器的 R/L 状态在电机识别过程期间执行，用于测量电机的电气常量。 此状态结束时，可使用产生的 R/L 比来计算 ID 和 IQ 电流控制器增益。
Rs **EST_State_Rs**	首次识别电机定子电阻时或在完全识别电机后重新校准定子电阻时，估算器处于 Rs 状态。
在线 **EST_State_OnLine**	估算器在线状态在电机闭环运行时出现。 要进入此状态，必须完全识别电机或将电机参数记录到头文件 user.h 中，并且控制器已运行。 在此状态下，可更改速度参考值并且电机转轴可满载运行。
斜升 **EST_State_RampUp**	在估算器斜升状态下，电机可上升至已配置的频率以执行其它识别任务，例如磁通和电感识别。 在此状态期间，不会估算任何参数，仅仅使电机上升至特定频率。
额定磁通 OL **EST_State_RatedFluxOL**	这是识别电机额定磁通之前估算器的一个过渡状态。
额定磁通 **EST_State_RatedFlux**	在此状态期间，将识别电机定子到转子的磁链。

表 6-3. 估算器 (EST) 状态 (continued)

估算器状态	简要说明
额定 Id **EST_State_IdRated**	仅适用于 ACIM 电机，该状态在识别 user.h 中定义的所需磁通对应的电机磁化电流时出现。
Ls **EST_State_Ls**	估算器的这一状态期间将识别定子电感。
Rr **EST_State_Rr**	仅适用于 ACIM 电机，在此状态期间，将识别转子电阻。 必须锁定转子后才能识别转子电阻。
锁定转子 **EST_State_LockRotor**	此状态用于告知用户转子必须锁定，并且转子锁定后必须重新启用控制器才能继续余下的识别过程。 估算器处于锁定转子状态时，控制器处于空闲状态。
斜降 **EST_State_RampDown**	识别电机的所有参数后将出现斜降状态，在此状态期间可移除流经电机绕组的电流。 此状态仅为识别过程结束前的过渡状态，因此，期间不会识别任何参数。
电机已识别 **EST_State_MotorIdentified**	电机已识别状态也是一个过渡状态，用于告知估算器状态机电机已完全识别，此状态结束后，状态机将返回空闲状态。 电机已识别状态结束后，控制器状态机也会进入空闲状态。

表 6-4 汇总了图 6-5 中显示的所有状态转换条件。

表 6-4. 估算器 (EST) 状态图的状态转换

条件	简要说明
电机未识别	已在控制器状态机中进行说明
电机已识别	已在控制器状态机中进行说明

Rs 重校准已禁用	InstaSPIN 定子电阻重校准功能也称为 Rs 离线重校准，当电机静止后，在已完全识别电机或用户已将电机参数记录到 user.h 文件后闭环运行电机之前可使用该功能。 此功能禁用后将出现 Rs 重校准被禁用的条件，该条件可通过以下代码示例进行检查：
	`if(EST_getFlag_enableRsRecalc(obj->estHandle)==FALSE)`
	启用控制器前，可通过调用以下包含指定参数的函数禁用 Rs 重校准功能：
	`EST_setFlag_enableRsRecalc(obj->estHandle, FALSE);`
	Rs 重校准功能默认为启用状态。

Rs 重校准已启用	此功能启用后将出现 Rs 重校准已启用的条件，该条件可通过以下代码示例进行检查：
	`if(EST_getFlag_enableRsRecalc(obj->estHandle)==TRUE)`
	启用控制器前，可通过调用以下包含指定参数的函数启用 Rs 重校准功能：
	`EST_setFlag_enableRsRecalc(obj->estHandle, TRUE);`
	Rs 重校准功能默认为启用状态。

控制器已禁用	已在控制器状态机中进行说明
控制器已启用	已在控制器状态机中进行说明
电机不是 ACIM	当电机类型不是 ACIM 时此条件为真，可通过以下示例进行检查： `if(CTRL_getMotorType(ctrlHandle)!=MOTOR_Type_Induction)`
电机为 ACIM	当电机类型为 ACIM 时此条件为真，可通过以下示例进行检查： `if(CTRL_getMotorType(ctrlHandle)==MOTOR_Type_Induction)`

等待已超时	与时间有关的状态转换基于内部计数器，而不是基于 user.c 中对应等待时间中存储的值：
	`uint_least32_t estWaitTime[EST_numStates];`
	`uint_least32_t FluxWaitTime[EST_Flux_numStates];`
	`uint_least32_t LsWaitTime[EST_Ls_numStates];`
	`uint_least32_t RsWaitTime[EST_Rs_numStates];`

6.2.3 控制器 (CTRL) 和估算器 (EST) 状态机的相关性

控制器状态机管理估算器状态机。 事实上，所有估算器状态转换仅在控制器处于在线状态时发生。 为对此进行说明，现以下面的简化控制器状态机（图 6-6）为例，图中以放大的在线状态来显示控制器状态机内的整个估算器状态机。

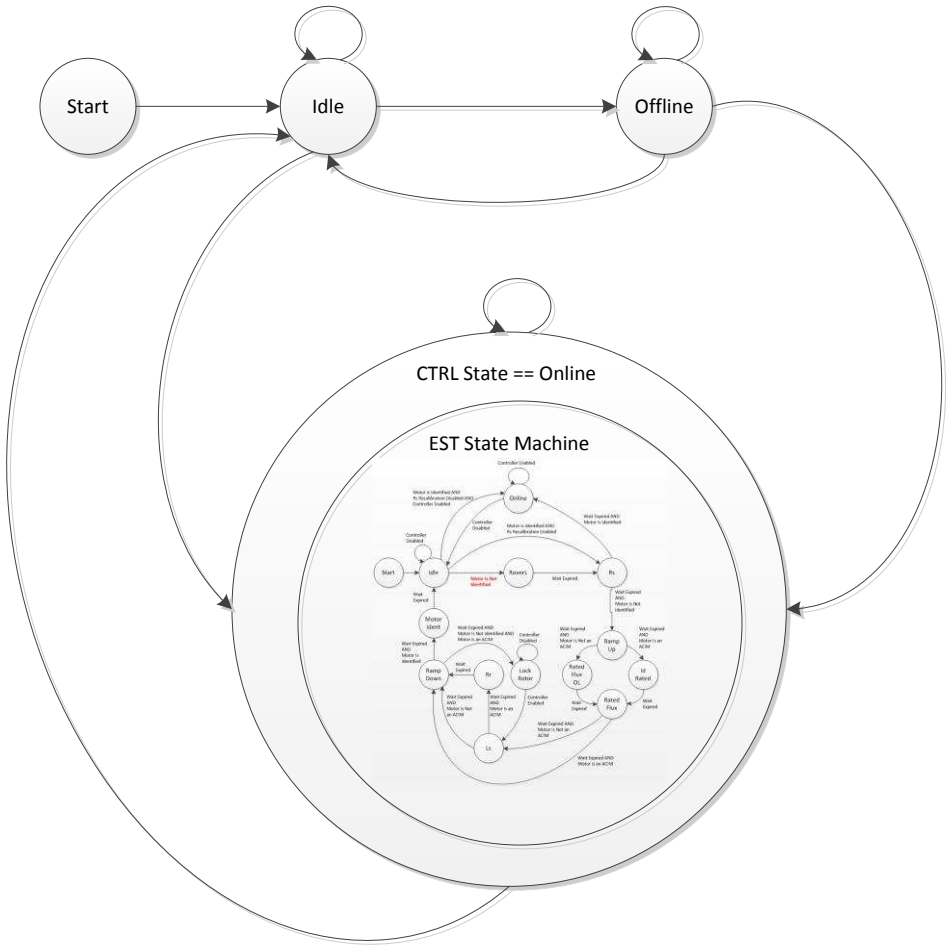


图 **6-6.** 控制器和估算器状态图 **-** 关系图

6.3 PMSM 和 ACIM 识别过程的差别

PMSM 和 ACIM 电机识别过程中的某些状态存在差异。图 6-7 突出显示了与 PMSM 电机相关的状态、与 ACIM 电机相关的状态以及与二者均相关的状态。

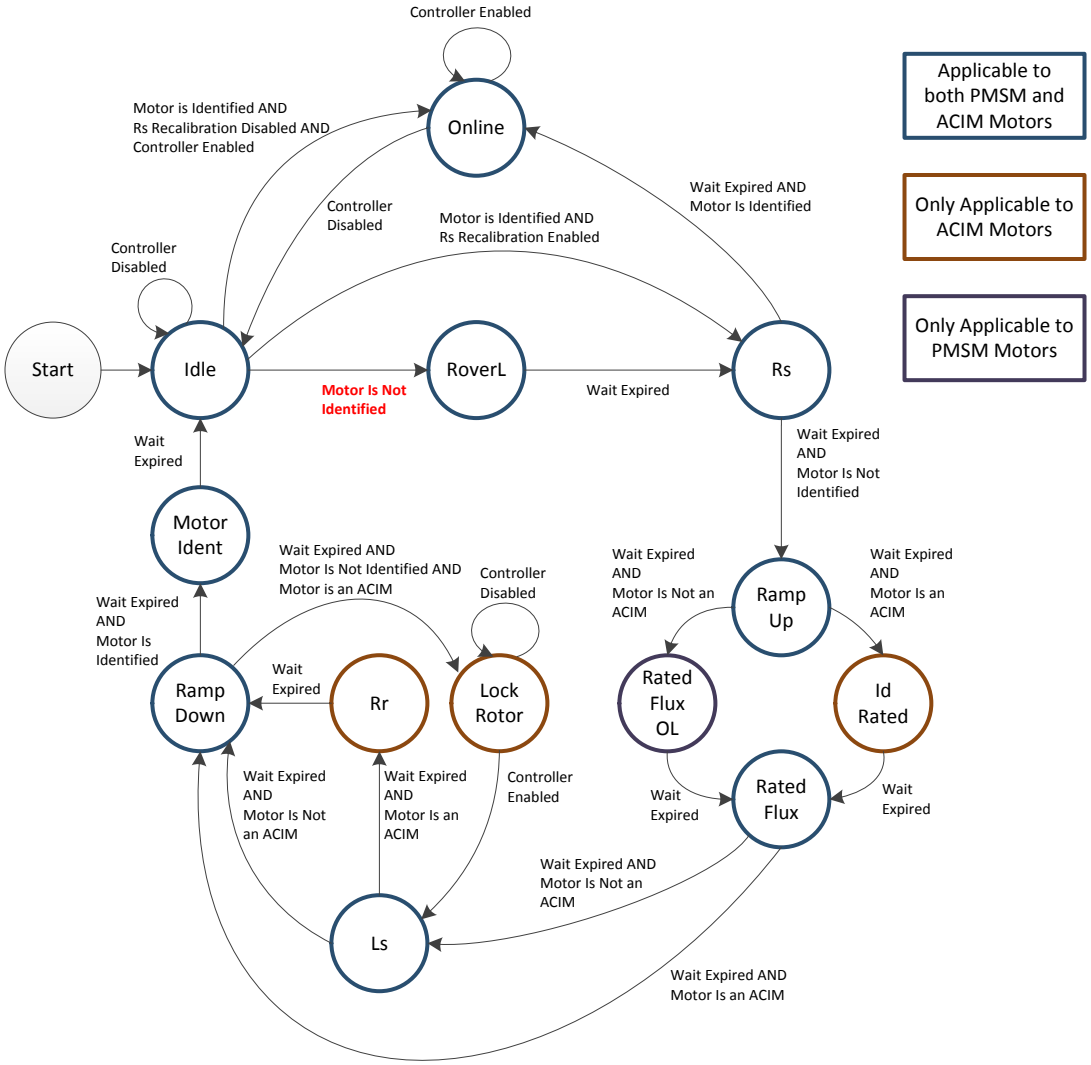


图 6-7. EST 状态图中的 PMSM 和 ACIM 状态

表 6-5 为图 6-7 的补充，其中列出了所有状态以及这些状态相关的电机。

表 6-5. PMSM 和 ACIM EST 状态列表

状态适用的电机	估算器状态
PMSM 和 ACIM	空闲、在线、R/L、Rs、斜升、额定磁通、Ls、斜降和电机已识别
仅 ACIM	额定 Id、锁定转子和 Rr
仅 PMSM	额定磁通 OL

6.4 必要条件

要想成功识别 PMSM 电机，需满足几个必要条件。 以下各个小节将从机械、硬件和软件这三个主要方面介绍这些必要条件：

6.4.1 机械必要条件

6.4.1.1 电机连接

在运行电机识别功能之前，电机必须连接到驱动板。 此驱动板可以是开发板或用户板，不论是哪种板，均需对软件进行相应配置使其与指定板配合工作。

6.4.1.2 相序

板上连接的相序不会影响对电机动态性能的控制（方向除外）。 如果方向很关键，建议在开始识别之前按所需方向连接电机。 如果电机旋转方向不是所需方向，请交换电机的其中两相，电机将反向旋转。 之后再次尝试识别。

6.4.1.3 最小机械负载

实现最小机械负载同样非常重要。 这是因为识别过程会运行一系列电机测试，其中一些测试在开环条件下进行。 这些开环测试无法在电机转轴上产生最大转矩，因此，要求电机转轴上的负载尽可能小，空载即为理想条件。节 6.9.2.1介绍了在无法移除机械负载（例如在压缩机或直驱洗衣机中）的情况下识别电机时所要遵循的一些准则。

6.4.2 硬件必要条件

有关详细信息，请参见Chapter 5。

6.4.3 软件必要条件

有关详细信息，请参见Chapter 5。

6.4.4 PMSM 电机识别的软件配置

- 电机类型
- 极对数
- Rhf 和 Lhf 频率
- Rs 电流
- Ls 电流
- Ls 和磁通频率

6.4.4.1 电机类型

用户必须已知电机类型才能运行电机识别。 对于 PMSM 电机识别，请为 PMSM 电机类型 (MOTOR_Type_Pm) 设置如下所示定义。 此定义位于 user.h 中：

```
#define USER_MOTOR_TYPE MOTOR_Type_Pm
```

如果选择了错误的电机（例如，连接 ACIM 时选择了 PMSM），估算器将无法识别正确的参数。 电机识别功能无法识别所连电机的类型，而是会识别电机参数。 要求指定正确的电机定义，否则，电机识别将不起作用。

6.4.4.2 极对数

极对数并非关键因素，但应正确设置，以便正确测量转速（每分转数 (RPM)）和转矩。 以具有四对电极的电机为例。 以下代码示例显示如何在 user.h 中设置此极对数：

```
#define USER_MOTOR_NUM_POLE_PAIRS (4)
```

6.4.4.3 Rhf 和 Lhf 频率

电机识别算法使用此频率来估算定子电阻和定子电感的初始值，以便计算电流控制器增益。 此估算过程在 100 Hz 的默认频率下完成，如 user.h 中的以下代码示例所示：

```
//! \brief Defines the R/L estimation frequency, Hz
//!
#define USER_R_OVER_L_EST_FREQ_Hz (100)
```

本文档后续部分在介绍识别步骤时将更详细地介绍此频率。

6.4.4.4 Rs 电流

定子电阻通过注入恒定电流来估算，该电流通过此参数进行设置。 尽管本文档将会详细说明此部分识别过程，但设置该值的一个通用准则是将其设为电机额定相电流的 10% 到 20%。 例如，如果某个电机需要 4A 电流来产生额定转矩，则用于估算定子电阻 (Rs) 的电流应为 0.5A 左右。

```
#define USER_MOTOR_RES_EST_CURRENT (0.5)
```

在电机识别过程的后续阶段，该电流还用于使电机开环旋转。 如果电机在整个斜升过程中不旋转，则以额定电流的 10% 为增量增大该电流，直至转轴在整个斜升过程中旋转。

6.4.4.5 Ls 电流

通常，估算定子电感 (Ls) 所需的电流应为额定相电流（负电流）的 10% 到 20%。 仍以电阻评估示例中的同一电机为例，其额定电流为 4A，因此建议的电流约为 -0.5A，如下所示。

```
#define USER_MOTOR_IND_EST_CURRENT (-0.5)
```

请记住，用于估算 Ls 的电流与用于识别 Rs 的电流无关。 尽管这两种电流的通用准则均为电机额定电流的 10-20%，但如果电机需要更大的 Rs 电流来通过开环斜坡，由于电机已处于旋转状态，因此 Ls 电流可能无需过高。

6.4.4.6 Ls 和磁通频率

该频率用于斜升正在识别的电机，以估算定子电感 (Ls) 和磁通。 通常，对于 PMSM 电机而言，使用 20Hz 频率便足以估算几十 µH 及更大的定子电感。 如果已知电感为几 µH 甚至更小，建议使用较高频率（高达 60Hz）来估算超低电感。 以 DRV8312 套件（版本 D 开发板）中的 Anaheim 电机为例：

```
#define USER_MOTOR_FLUX_EST_FREQ_Hz (20.0)
```

如果典型值 20Hz 超出电机额定速度，请降低设定的频率，使其保持低于电机额定转速。

6.4.5 ACIM 电机识别的软件配置

- 电机类型
- 极数
- Rhf 和 Lhf 频率
- 额定磁通
- Rs 电流
- IdRated、Ls 和 Rr 频率

6.4.5.1 电机类型

如前文所述，用户必须已知电机类型才能运行电机识别。 对于 ACIM 电机识别，请为 ACIM 电机类型 (MOTOR_Type_Induction) 设置如下所示定义。 此定义位于 user.h 中：

```
#define USER_MOTOR_TYPE MOTOR_Type_Induction
```

6.4.5.2 极对数

需遵循节 6.4.4.2中介绍的相同条件。

6.4.5.3 Rhf 和 Lhf 频率

需遵循节 6.4.4.3中介绍的相同条件。

6.4.5.4 额定磁通

进行完全识别时需要 ACIM 电机的额定磁通。 此额定磁通在 user.h 中按如下设置：

```
#define USER_MOTOR_RATED_FLUX (0.8165*220.0/60.0)
```

此磁通的计算方法是借助电机铭牌。 例如，如果电机的额定值为单相 220VAC 和 60Hz，则磁通值的计算方法如下：

$$\text{RatedFlux} = \sqrt{2} \times \frac{1}{\sqrt{3}} \times \frac{220\text{VAC}}{60\text{Hz}} = 0.8165 \times \frac{220\text{VAC}}{60\text{Hz}} = 2.9938 \quad (9)$$

又如，某个电机的输入电压同为 220VAC 但额定频率为 50Hz。 此时的额定磁通为：

$$\text{RatedFlux} = \sqrt{2} \times \frac{1}{\sqrt{3}} \times \frac{220\text{VAC}}{50\text{Hz}} = 0.8165 \times \frac{220\text{VAC}}{50\text{Hz}} = 3.5926\text{V / Hz} \quad (10)$$

√2 项用于将单相 RMS 电压值转换为峰值电压，1/√3 项用于将电机线电压转换为电机相电压。 基于给定电机的额定磁通来识别一半额定磁通对应的 Id 额定电流。 在这种情况下，只需输入所需磁通或所需磁通的一部分。 请注意，计算额定磁通所需的电压为电机相电压，因此需要从 220VAC（线电压）转换为 VDC（线电压），再从 VDC（线电压）转换为 VDC（相电压）。

6.4.5.5 Rs 电流

需遵循节 6.4.4.4中介绍的相同条件。

6.4.5.6 IdRated、Ls 和 Rr 频率

此频率用于斜升正在识别的电机，以估算 ACIM 电机的 Id 额定电流、定子电感和转子电阻。 感应电机所使用的典型频率为 5 Hz：

```
#define USER_MOTOR_FLUX_EST_FREQ_Hz (5.0)
```

6.5 PMSM 电机完全识别

对 PMSM 电机进行完全识别时，图 6-8 显示了控制器和估算器状态机内部发生事件的顺序。

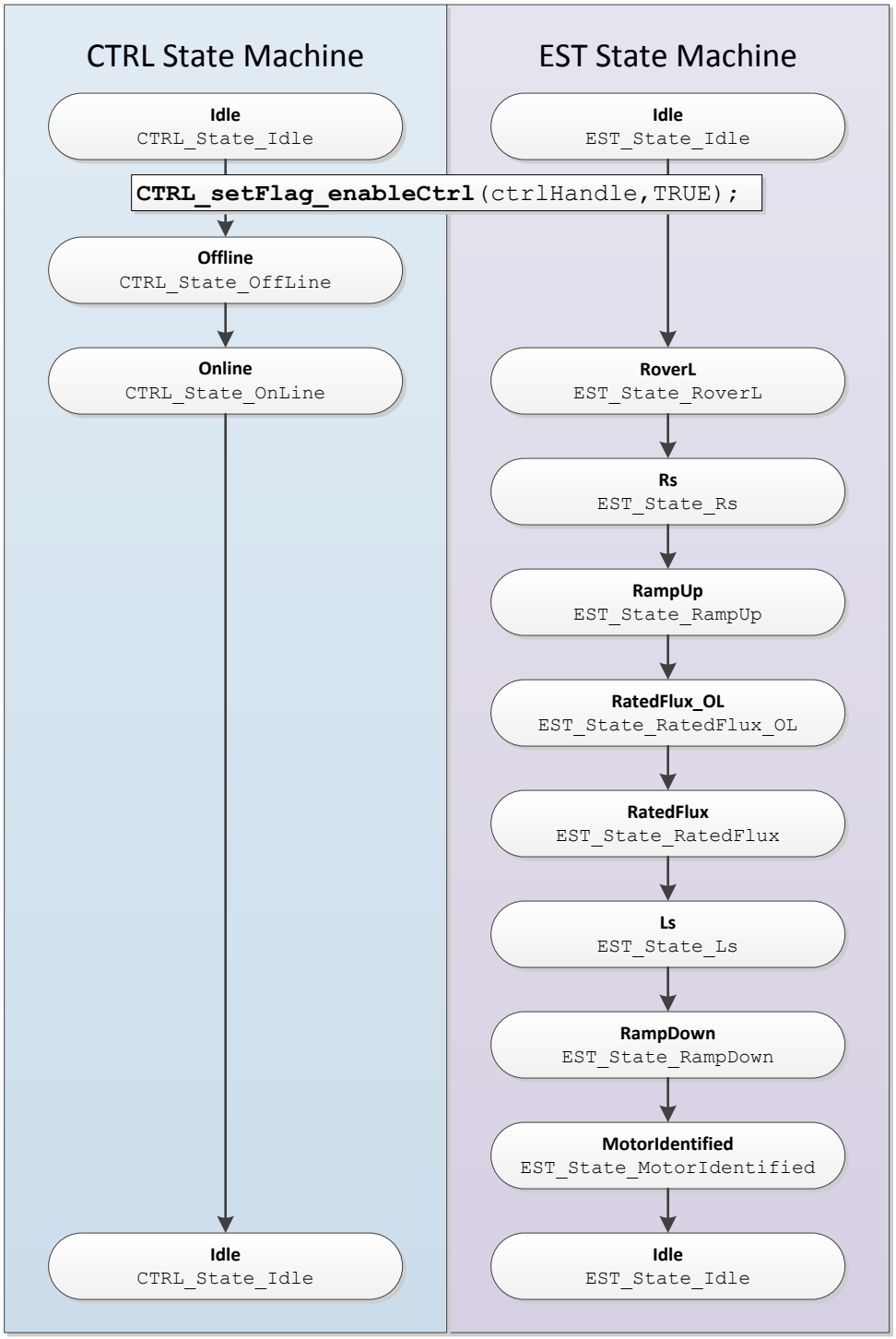


图 **6-8. PMSM** 完全识别 **- CTRL** 和 **EST** 状态顺序

启用控制器之前，代码得知以下两个条件为真时将执行电机完全识别：

1. 电机尚未识别
2. 未使用任何来自 user.h 的参数。

```
if( (EST_isMotorIdentified(obj->estHandle) == FALSE) &&
    (CTRL_getFlag_enableUserMotorParams(ctrlHandle) == FALSE))
```

本文档接下来将详细介绍识别期间的各个状态。

6.5.1 CTRL_State_Idle 和 EST_State_Idle

在启用控制器之前，控制器和估算器状态机均处于空闲状态，分别表示为 CTRL_State_Idle 和 EST_State_Idle。 这也称作这两个状态机的非活动状态。

6.5.2 CTRL_State_OffLine 和 EST_State_Idle（硬件偏移已校准）

启用控制器并且开始完全识别后，控制器状态机执行的首个任务为偏移计算。 此过程通过控制器状态机的以下状态表示：CTRL_State_OffLine。 估算器在控制器离线状态期间保持空闲状态 (EST_State_Idle)。

执行偏移计算的目的在于为电流测量和电压测量设置零点。 为了计算偏移，在 EPWM 引脚上设定 50% 占空比并持续预配置的一段时间。 用户可更改计算这些偏移的时间，该时间在 user.c 文件中按如下所示进行配置：

```
pUserParams->ctrlWaitTime[CTRL_State_OffLine]=(uint_least32_t)(5.0*USER_CTRL_FREQ_Hz);
```

在上述示例中，偏移校准的持续时间为 5s。 尽管 5s 的偏移校准时间足以应对大多数硬件，但如果用户需要更短或更长的时间来满足其特定需求，则只需更改上述代码行中的值 5.0，随后执行偏移校准的时间将根据新设置发生变化。

执行偏移校准后，最终结果将存储在驱动程序对象 (HAL_Obj) 中。 有关 HAL_Obj 的更多详细信息，请参见表 8-1。图 6-9 显示了 DRV8312 版本 D 开发板执行偏移校准后的最终结果。

drv.adcBias	struct _DRV_AdcDat...	{...}
I	struct _MATH_vec3_	{...}
value	long[3]	0x0000BB46@Data
[0]	long	0.8795804977 (Q-Value(24))
[1]	long	0.8918017149 (Q-Value(24))
[2]	long	0.8876305819 (Q-Value(24))
V	struct _MATH_vec3_	{...}
value	long[3]	0x0000BB4C@Data
[0]	long	0.2504473329 (Q-Value(24))
[1]	long	0.2495527864 (Q-Value(24))
[2]	long	0.2473165989 (Q-Value(24))

图 6-9. 偏移校准后的 CCStudio 观察窗口

理想条件下，电流偏移（也称为偏置值）应为：

$$\text{bias} = 0.5 \times \text{Current\_sf} \tag{11}$$

注意：有关以下公式中所用变量的定义，请参见4.1 节。

在以下示例中，此电流换算系数（*Current_sf* 或 USER_CURRENT_SF）通过 DRV8312 版本 D 开发板的值计算得出：

$$USER\_CURRENT\_SF = \frac{USER\_ADC\_FULL\_SCALE\_CURRENT\_A}{USER\_IQ\_FULL\_SCALE\_CURRENT\_A} = \frac{17.69}{10.0} = 1.769$$

$$Current\_Bias^{Ideal} = \frac{1.65}{3.3} \times USER\_CURRENT\_SF = 0.5 \times 1.769 = 0.8845 \quad (12)$$

电流反馈电路为双向，理想零点位于 $V_{DD}/2$（即 1.65V），得出的换算系数为 0.5。

电压偏置按如下公式计算。 首先按如下公式计算电压换算系数：

$$USER\_VOLTAGE\_SF = \frac{USER\_ADC\_FULL\_SCALE\_VOLTAGE\_V}{USER\_IQ\_FULL\_SCALE\_VOLTAGE\_V} = \frac{66.32}{48.0} = 1.3817$$

$$Voltage\_Bias^{Ideal} = \frac{\frac{V_{BUS}}{2}}{USER\_ADC\_FULL\_SCALE\_VOLTAGE\_V} \times USER\_VOLTAGE\_SF = 0.25 \quad (13)$$

理想电压偏置基于以下事实：引入 50% 占空比来测量这些偏移时，相电压将呈现接近 VBUS * 50% 的电压，随后该电压将根据 ADC 测量的最大电压按比例缩小。 以 VBUS 为 24V 的 DRV8312 版本 D 开发板为例，理想电压偏置为 0.25，如下所示：

$$Voltage\_Bias^{Ideal} = \frac{\frac{24}{2}}{66.32} \times 1.3817 = 0.25 \quad (14)$$

在图 6-10 的示波器图中，所示占空比为 50%，同时通过游标测得偏移校准执行时间为 5s。 在左图中，由于水平标尺分辨率的原因，无法观察到任何 PWM 曲线。 左侧显示了 1.65V 幅值，表示 3.3V 信号占空比为 50%。 右侧显示了实际的 PWM 信号，该信号放大为每分度 50µs。

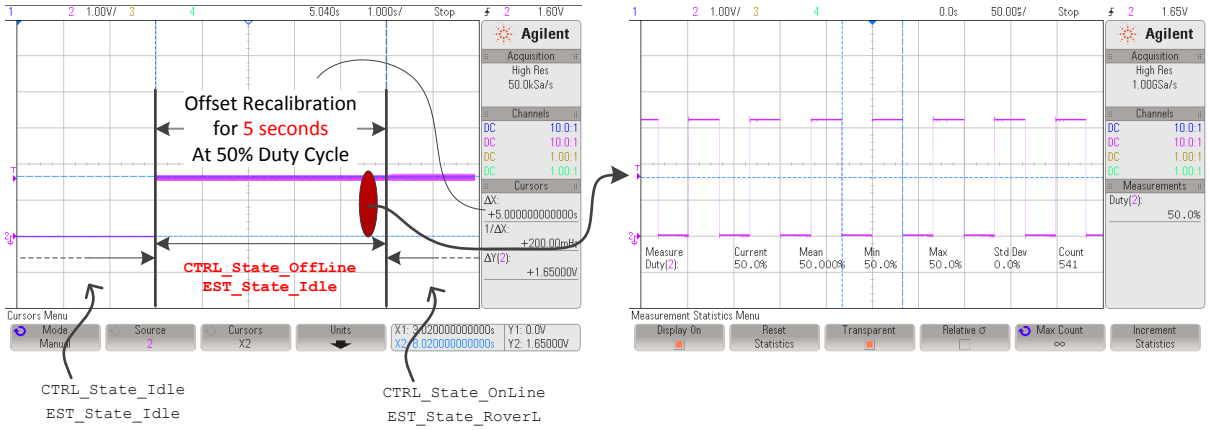


图 **6-10.** 采用 **50% PWM** 占空比进行偏移计算

6.5.3 *CTRL_State_OnLine* 和 *EST_State_RoverL*

偏移（也称为偏置）校准完成后，将启用估算器并开始进入识别过程的以下状态。 估算器状态机进入空闲状态后要执行的首个状态称为 R/L 状态，即 RoverL（图 6-11）。

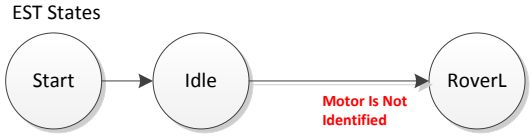


图 **6-11. RoverL EST** 状态

估算器的这一状态通过除以所测的电阻和电感来测量定子电路的电子时间常量。 控制器对象使用 RoverL 时间常量来设置电流控制器 IQ 和 ID 的电流控制器增益 KP 和 KI。 如果绕过电机识别，则将使用 user.h 中提供的参数来设置电流控制器增益。

RoverL 时间常量的具体测量过程为：在固定频率下向定子绕组注入固定幅值的电流。 注入电流的各个参数如下：幅值、频率和测量时间。

6.5.3.1 RoverL 状态下注入电流的幅值

要确定定子注入电流的幅值，需要将 user.h 中的以下参数 (USER_MOTOR_RES_EST_CURRENT) 除以 2。

例如，如果 user.h 中的以下参数 (USER_MOTOR_RES_EST_CURRENT) 值为 1.0，则注入电流的大小为该值的 ½，即 0.5A。要达到目标电流幅值，此注入电流的斜坡速率为 0.5s。

```
#define USER_MOTOR_RES_EST_CURRENT (1.0)
```

通常，此电流需要足够高才能在 ADC 测量中产生有效位数，但也不可过高，因为过高电流会导致电机运动或电机过热。 按照这一通用规则，电流约为电机额定相电流的 10% 到 20%。

6.5.3.2 RoverL 状态下注入电流的频率

用于计算 RoverL 的注入电流频率通过 user.h 中的以下参数设置（规定频率单位为 Hz）。

```
//! \brief Defines the R/L estimation frequency, Hz
//!
#define USER_R_OVER_L_EST_FREQ_Hz (100)
```

对于高速电机而言，默认频率值 100Hz 可能会导致电机旋转或往复运动。 如果正在测试的电机出现这种情况，请以 50Hz 增量将该频率增加到更高值，直至电机不再在 R/L 状态下发生移动。

6.5.3.3 RoverL 状态下的测量时间

执行 R/L 测量时要考虑的第三个参数就是执行此测量所花费的时间。 该参数在 user.c 中配置如下，尽管默认设置适用于大多数情况，但仍可按需对该参数进行更改：

```
pUserParams->estWaitTime[EST_State_RoverL] = (uint_least32_t)(5.0 * USER_EST_FREQ_Hz);
```

图 6-12 显示了如何注入该电流。 配置的参数用红色突出显示：5s 持续时间、1.0 A / 2 = 0.5 A 电流幅值和 100Hz 频率。

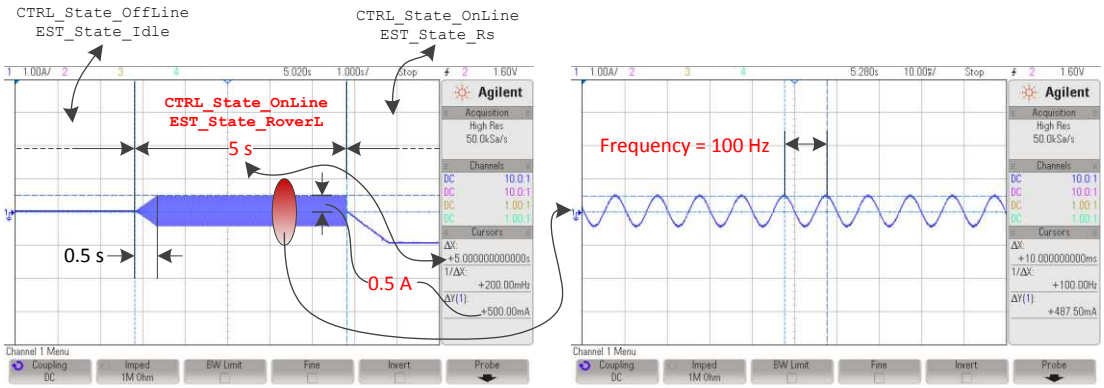


图 **6-12.** 用于测量 **RoverL EST** 状态的注入电流

可通过调用以下函数返回 RoverL 比的方式从估算器中读取 R/L 状态生成的值。

```
// Code example to get RoverL into a variable
float_t RoverL = CTRL_getRoverL(ctrlHandle);
```

检查 RoverL 状态生成的估算值的另一种方法是调用两个函数，一个函数用于高频电阻估算值 (Rhf)，另一个函数用于高频电感估算值 (Lhf)。 以下代码示例使用这两个函数获取 RoverL 状态期间生成值的本地副本：

```
// Code example to get high frequency R (Rhf) and high frequency inductance
// (Lhf) to variables
float_t Rhf = CTRL_getRhf(ctrlHandle);
float_t Lhf = CTRL_getLhf(ctrlHandle);
float_t RoverL = Rhf/Lhf;
```

上述代码示例中计算得出的 RoverL 生成值与函数 CTRL_getRoverL() 返回的值完全相同。

如果不进行电机识别，用户可使用以下代码示例基于 user.h 文件中提供的参数来计算 RoverL 常量：

```
// Code example to get RoverL into a variable based on user.h parameters
#define USER_MOTOR_Rs (4.0)
#define USER_MOTOR_Ls_d (0.03)
float_t RoverL = USER_MOTOR_Rs/USER_MOTOR_Ls_d;
```

最后，控制器对象 (CTRL_Obj) 将根据 RoverL 比来初始化电流控制器增益。图 6-13 显示了控制器对象 (CTRL_Obj) 如何在内部设置电流控制器增益。 此处列出的代码仅用于说明电流控制器初始设置背后的运算过程。 由于代码在控制器内部执行，因此用户无需执行代码。

```
// get the full scale current and voltage values from #defines in user.h
#define USER_IQ_FULL_SCALE_CURRENT_A (10.0)
#define USER_IQ_FULL_SCALE_VOLTAGE_V (48.0)

// deriving controller period in seconds from #defines in user.h
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK (1)
#define USER_PWM_FREQ_kHz      (15.0)
#define USER_PWM_PERIOD_usec   (1000.0/USER_PWM_FREQ_kHz)
#define USER_ISR_PERIOD_usec   USER_PWM_PERIOD_usec
#define USER_CTRL_PERIOD_usec  (USER_ISR_PERIOD_usec*USER_NUM_ISR_TICKS_PER_CTRL_TICK)
#define USER_CTRL_PERIOD_sec      ((float_t)USER_CTRL_PERIOD_usec/(float_t)1000000.0)

// get Lhf and RoverL from the controller object
float_t RoverL = CTRL_getRoverL(ctrlHandle);
float_t Lhf = CTRL_getLhf(ctrlHandle);

// get full scale current and voltage values in local variables
float_t fullScaleCurrent = USER_IQ_FULL_SCALE_CURRENT_A;
float_t fullScaleVoltage = USER_IQ_FULL_SCALE_VOLTAGE_V;

// get the controller period in seconds in a local variable
float_t ctrlPeriod_sec = USER_CTRL_PERIOD_sec;

// get the controller object handle
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;

// set the Id controller gains
Kp = _IQ((0.25*Lhf*fullScaleCurrent)/(ctrlPeriod_sec*fullScaleVoltage));
Ki = _IQ(RoverL*ctrlPeriod_sec);
Kd = _IQ(0.0);
PID_setGains(obj->pidHandle_Id,Kp,Ki,Kd);
PID_setUi(obj->pidHandle_Id,_IQ(0.0));
CTRL_setGains(ctrlHandle,CTRL_Type_PID_Id,Kp,Ki,Kd);

// set the Iq controller gains
Kp = _IQ((0.25*Lhf*fullScaleCurrent)/(ctrlPeriod_sec*fullScaleVoltage));
Ki = _IQ(RoverL*ctrlPeriod_sec);
Kd = _IQ(0.0);
PID_setGains(obj->pidHandle_Iq,Kp,Ki,Kd);
PID_setUi(obj->pidHandle_Iq,_IQ(0.0));
CTRL_setGains(ctrlHandle,CTRL_Type_PID_Iq,Kp,Ki,Kd);
```

图 **6-13.** 用于设置电流控制器 **Kp** 和 **Ki** 初始增益的内部代码

6.5.3.4 RoverL 识别期间的电流控制器稳定性故障排除

请参见6.9 节。

6.5.3.5 调节高速电机产生的电流控制器增益

从上述代码可以看出，ID 和 IQ 电流控制器均初始化为由 RoverL 和 Lhf 计算得出的相同增益。 这两个控制器的比例增益中还引入了系数 0.25。 该系数用于将比例增益设置为理论限值的 ¼。 在电机运行速度需要远高于额定速度的应用中（即场强减弱），Id 和 Iq 电流控制器的比例增益需要增大到 4 倍，从而使增益达到理论限值。 使用以下的代码示例便可轻松将这些增益增大到 4 倍：

```
_iq Kp_Id = CTRL_getKp(handle,CTRL_Type_PID_Id);
_iq Kp_Iq = CTRL_getKp(handle,CTRL_Type_PID_Iq);
CTRL_setKp(handle,CTRL_Type_PID_Id, _IQmpy(Kp_Id, _IQ(4.0)));
CTRL_setKp(handle,CTRL_Type_PID_Iq, _IQmpy(Kp_Iq, _IQ(4.0)));
```

如果用户想要确认电流控制器增益是在 RoverL 时间常量设置电流控制器后设置，可使用以下代码示例：

```
// declare global variables for the Id controller gains
_iq gKp_Id, gKi_Id, gKd_Id;
// declare global variables for the Iq controller gains
_iq gKp_Iq, gKi_Iq, gKd_Iq;

// get the current controller gains for the Id controller
CTRL_g etGains(ctrlHandle,CTRL_Type_PID_Id,&gKp_Id,&gKi_Id,&gKd_Id);
// get the current controller gains for the Iq controller
CTRL_g etGains(ctrlHandle,CTRL_Type_PID_Iq,&gKp_Iq,&gKi_Iq,&gKd_Iq);
```

如果用户选择忽略 RoverL 常量设置的增益并决定使用自己设定的增益，只需使用以下函数来设置电流控制器增益，该过程在 **ctrl.h** 中实现：

```
void CTRL_setKi(CTRL_Handle handle,const CTRL_Type_e ctrlType,const _iq Ki);
void CTRL_setKp(CTRL_Handle handle,const CTRL_Type_e ctrlType,const _iq Kp);
void CTRL_setGains(CTRL_Handle handle,const CTRL_Type_e ctrlType,
                   const _iq Kp,const _iq Ki,const _iq Kd);
```

6.5.4 *CTRL_State_OnLine 和 EST_State_Rs*

识别过程的此状态执行定子电阻识别（图 6-14）。

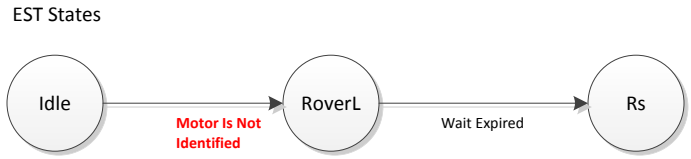


图 **6-14. Rs EST** 状态

D 轴中注入直流电流，电流幅值在 user.h 中按如下定义：

```
#define USER_MOTOR_RES_EST_CURRENT (1.0)
```

请注意，该电流的定义与 RoverL 状态所使用的定义相同，但 RoverL 使用该值的一半，而 Rs 状态则使用定义中的完整值。 注入的电流应足够高，能够在 ADC 转换器中产生有效的测量值；同时又要足够低，以避免电机过热。 通常，电机额定电流的 10% 至 20% 便足以产生精确的定子电阻估算值。

此状态的时间间隔由 user.c 中的三个时间值设置，如下所示：

```
pUserParams->RsWaitTime[EST_Rs_State_RampUp] = (uint_least32_t)(1.0*USER_EST_FREQ_Hz);
pUserParams->RsWaitTime[EST_Rs_State_Coarse] = (uint_least32_t)(2.0*USER_EST_FREQ_Hz);
pUserParams->RsWaitTime[EST_Rs_State_Fine] = (uint_least32_t)(4.0*USER_EST_FREQ_Hz);
```

默认情况下，定子电阻 Rs 的整个识别过程耗费 7s。 Rs 识别过程的第一部分是 1s 的斜升阶段。 在此阶段，定义的直流电流注入到 D 轴。 斜升阶段结束后，Rs 识别过程将开始粗调所识别的 Rs。 粗调过程所耗费的时间为 RsWaitTime [*EST_Rs_State_Coarse*] 存储的时间之前定义的时间。 该时间默认设置为 2s，对于经过 InstaSPIN 库版本测试的所有电机而言，2s 足以执行粗调校准。 不过，该时间设置比较灵活，用户可以按需调整，尽管无法预测是否有此需要。

粗调过程结束后，将开始精确 Rs 重校准。 识别过程完成精确 Rs 重校准所耗费的时间默认设置为 4s，用户可通过修改 RsWaitTime [*EST_Rs_State_Fine*] 中存储的值灵活更改该时间值。

图 6-15 显示了整个 Rs 识别过程，图中突出显示了斜升时间、幅值以及过程的持续时间。

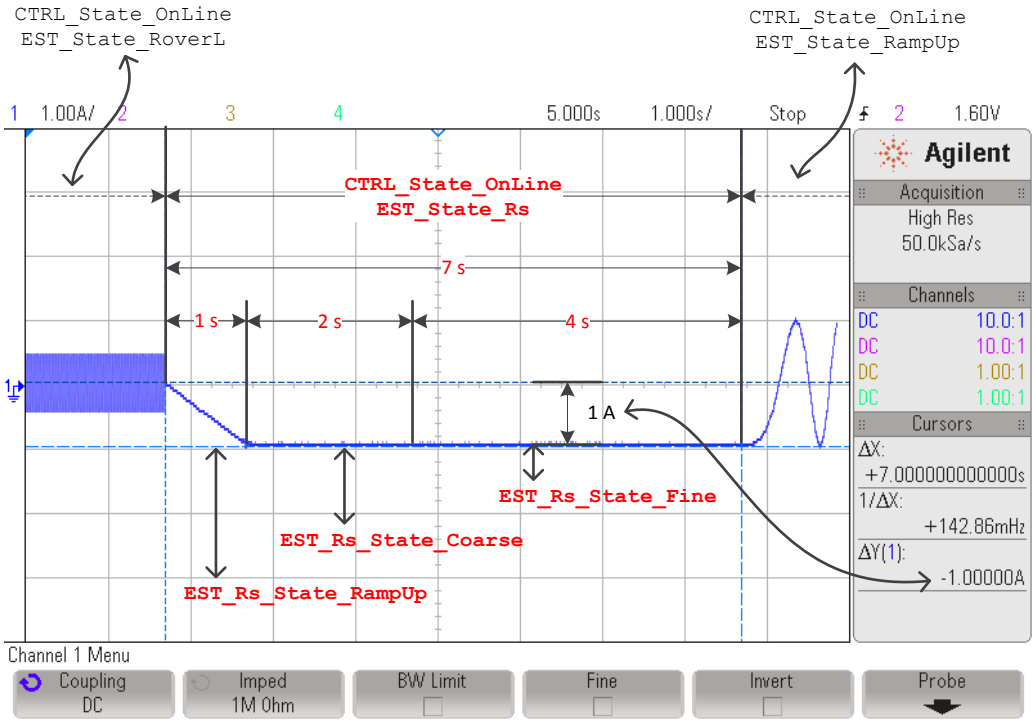


图 6-15. Rs 识别 EST 状态期间的相电流

用户可使用以下代码示例监视估算电阻的过程。 这在调整识别电阻所耗时间时尤其有用。

```
// get the stator resistance
gMotorVars.Rs_Ohm = EST_getRs_Ohm(obj->estHandle);
```

例如，监视正在识别的电阻值能够向用户反馈识别电阻达到稳定状态所需的时间。 识别电阻达到稳定状态所需的时间可以在 user.c 文件中配置，下次识别电机时便可加快电阻识别过程。

6.5.4.1 Rs 识别期间的电流控制器稳定性故障排除

请参见6.9 节。

6.5.5 CTRL_State_OnLine 和 EST_State_RampUp

定子电阻识别完成后，将执行新的估算器状态。 接下来的这一状态称为斜升状态，即 EST_State_RampUp。 在识别过程的这一状态期间，电机会加速到某一速度，从而开始识别其它参数。 该状态期间不会识别任何参数，而是启动相应条件。 此状态的影响因素有多种。

6.5.5.1 斜升电流幅值

第一个因素就是斜升过程使用的电流的幅值。 该电流幅值同时也是识别定子电阻时使用的幅值。图 6-16 给出了该电流的定义，在本例中，斜升过程使用 1A 电流：

```
#define USER_MOTOR_RES_EST_CURRENT (1.0)
```

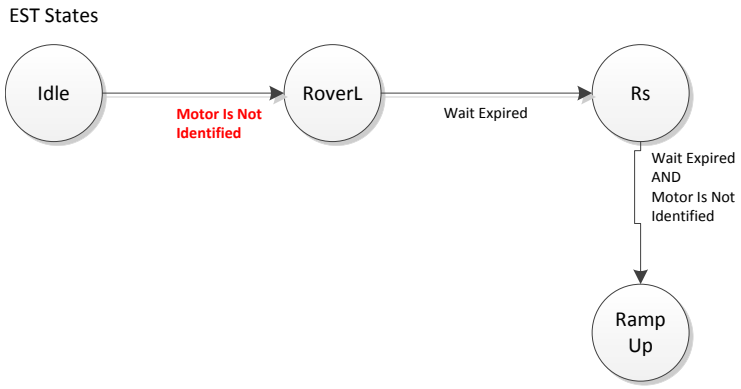


图 6-16. 斜升 EST 状态

6.5.5.1.1 斜升期间的电机转轴停转故障排除

请参见6.9 节。

6.5.5.2 斜升时间和加速度

斜升状态期间的下一个参数为电机斜升的时间（图 6-17）。 该值默认设置为 20s，如 user.c 文件中的以下代码示例所示：

```
pUserParams->estWaitTime[EST_State_RampUp] = (uint_least32_t)(20.0*USER_EST_FREQ_Hz);
```

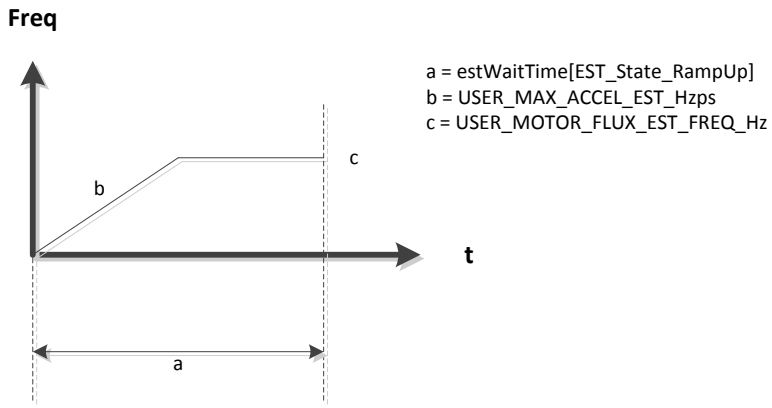


图 6-17. 斜升时间

该时间可更改为任意所需值。图 6-18 是通过电流探针测量相电流得出的，其中显示了斜升状态时间。

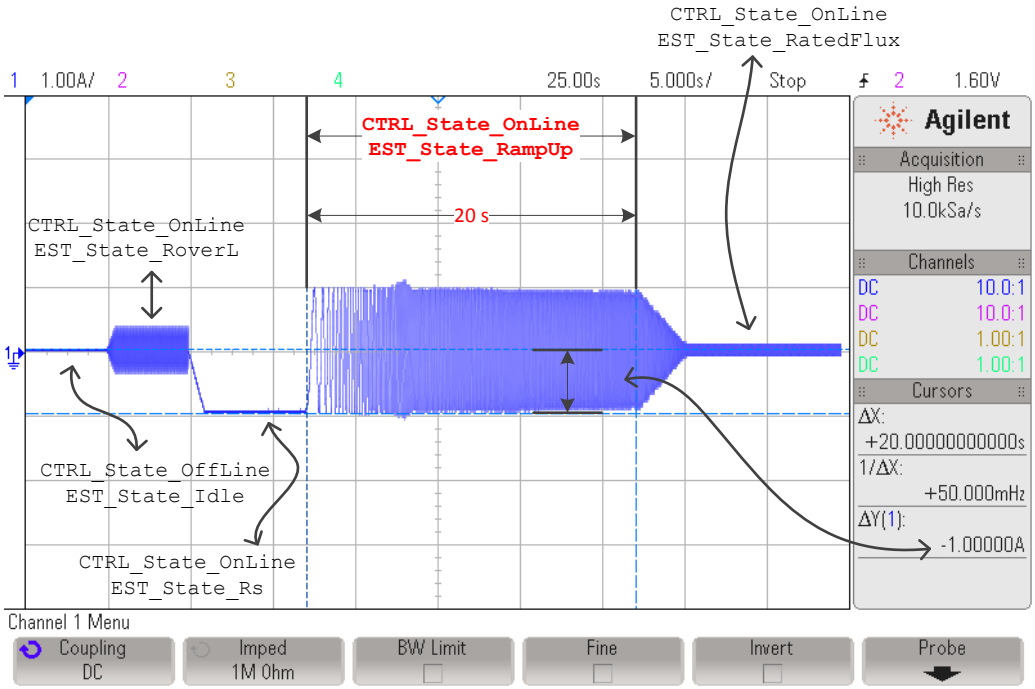


图 6-18. 斜升 EST 状态期间的相电流

此斜坡的加速度为 user.c 中设置的另一个参数，可根据用户要求进行更改。 在高惯性负载的同一示例中，该加速度可更改。 斜升加速度默认设置为 2.0Hz/s，如下所示：

```
//! \brief Defines maximum acceleration for the estimation speed profiles, Hz/s
#define USER_MAX_ACCEL_EST_Hzps (2.0)
```

6.5.5.2.1 平滑斜坡的电机转轴故障排除

请参见6.9 节。

6.5.5.3 PMSM 最终斜升速度

斜升后的最终速度作为电机参数在 user.h 中设置。 该速度单位为 Hz，应根据相位电感范围进行设置。 对于几 µH 电感而言，该值应为 50Hz 左右。 对于几十到几百 µH 电感而言，20Hz 的值便足以准确识别电感。

```
// During Motor ID, maximum commanded speed in Hz
#define USER_MOTOR_FLUX_EST_FREQ_Hz (20.0)
```

请记住，增大该频率可能需要增加斜升时间，从而在足够长时间的斜升状态下以指定加速度达到最终频率。之前的图中也显示了 20Hz 最终频率。

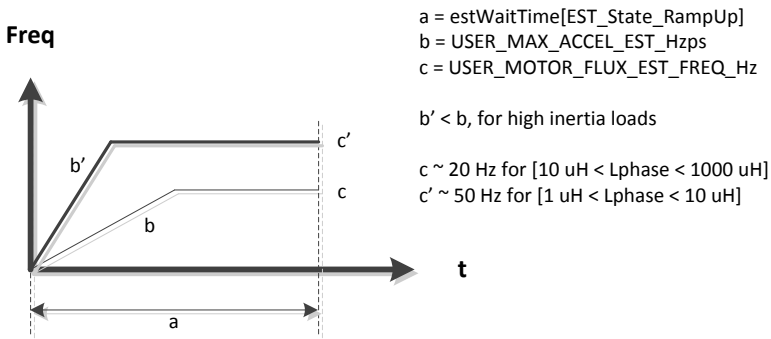


图 6-19. 不同加速度和最终速度下的斜升时间

6.5.6 CTRL_State_OnLine 和 EST_State_RatedFlux

一旦电机以 user.h 中设置的指定频率运行，便会立即开始额定磁通识别过程（图 6-20）。

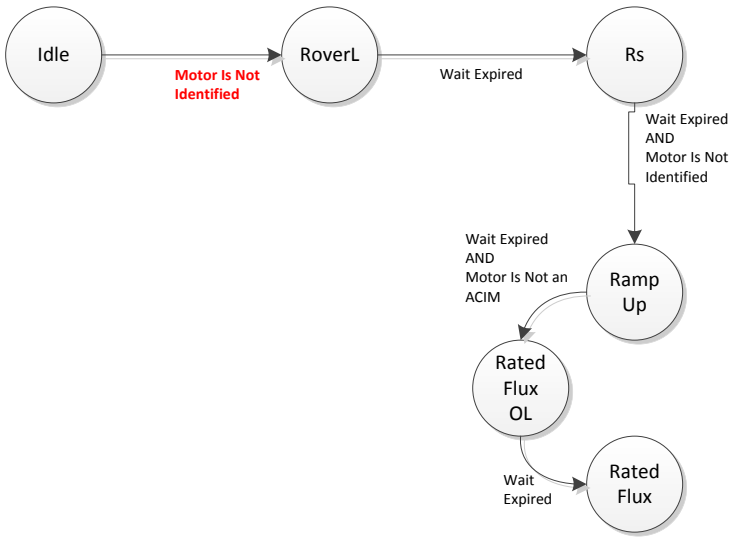


图 6-20. PMSM 额定磁通 EST 状态

6.5.6.1 电流斜降

识别磁通时，将首先由电机识别状态机内部启用闭环。 该闭环不是由用户启用。 启用该闭环时，电流消耗降至机械负载在相同频率下保持旋转状态所需的最小电流值。 电流降低期间的斜率为一个固定值，即每秒的 Rs 估算值除以 3。除数 3 是在电机识别过程的设计阶段选择的，旨在提供一个更小的斜率。

用户可使用以下公式计算该斜率： 本示例中使用 1A 电流来进行电阻识别：

$$\text{RatedFlux\_CurrentSlope} = \frac{\text{USER\_MOTOR\_RES\_EST\_CURRENT}}{1\text{s}} \times \frac{1}{3} = 0.33\,\text{A/s} \quad (15)$$

图 6-21 中斜率为 0.33 A/s，显示了出现额定磁通状态时电流如何减小。 该图还突出显示了识别额定磁通所耗费的时间。

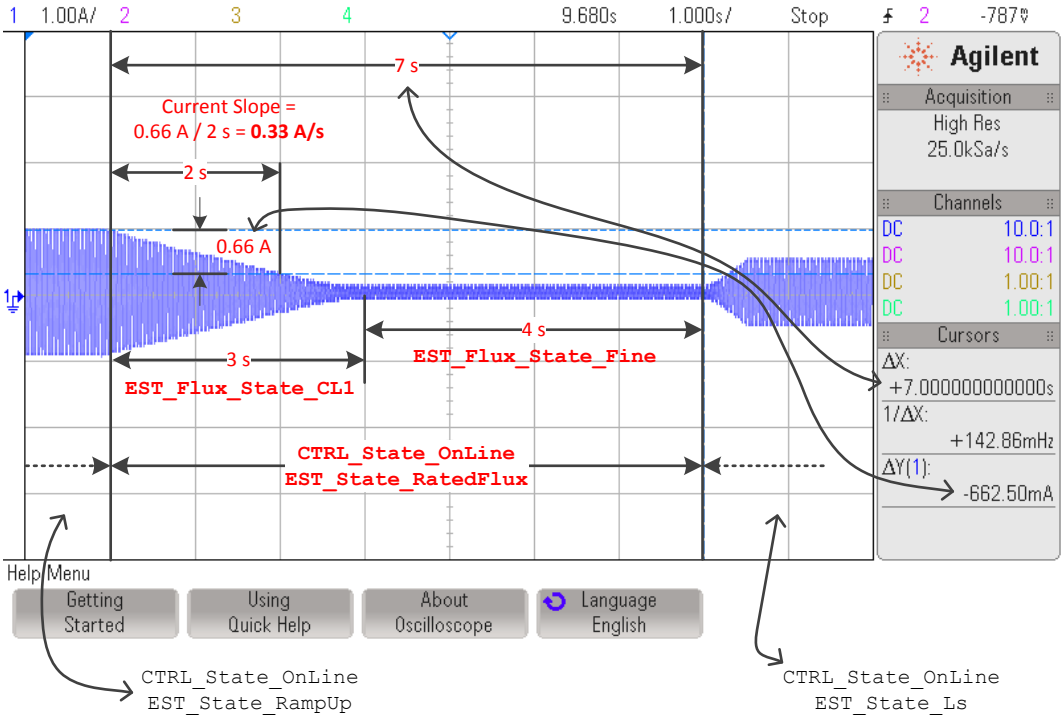


图 6-21. 额定磁通 EST 状态期间的相电流

6.5.6.2 总测量时间

此图中显示的时间基于默认设置：3s 电流斜降时间（EST_Flux_State_CL1，CL1 代表闭环 1）和 4s 额定磁通微调时间 (EST_Flux_State_Fine)，共计 7s。 这两个值均通过函数调用在 user.c 文件中设置，如下所示：

```
pUserParams->FluxWaitTime[EST_Flux_State_CL1] =(uint_least32_t)(3.0*USER_EST_FREQ_Hz);
pUserParams->FluxWaitTime[EST_Flux_State_Fine]=(uint_least32_t)(4.0*USER_EST_FREQ_Hz);
```

这两个默认值被认为适用于算法验证期间测试的所有电机。 用户可通过监视由算法识别的额定磁通来确认识别时间是否足够，并确保当估算器处于 EST_State_RatedFlux 状态时所识别的值处于稳定状态。

以下代码示例显示了如何监视所识别的磁通值，如果观察窗口中该值的变化幅度不超过典型变化幅度（约 5%），则可认为所识别的磁通处于稳定状态。

```
// get the flux
gMotorVars.Flux_VpHz = EST_getFlux_VpHz(obj->estHandle);
```

6.5.6.3 磁通测量故障排除

请参见6.9 节。

6.5.7 CTRL_State_OnLine 和 EST_State_Ls

额定磁通测量结束后，将立即开始定子电感识别过程（图 6-22）。

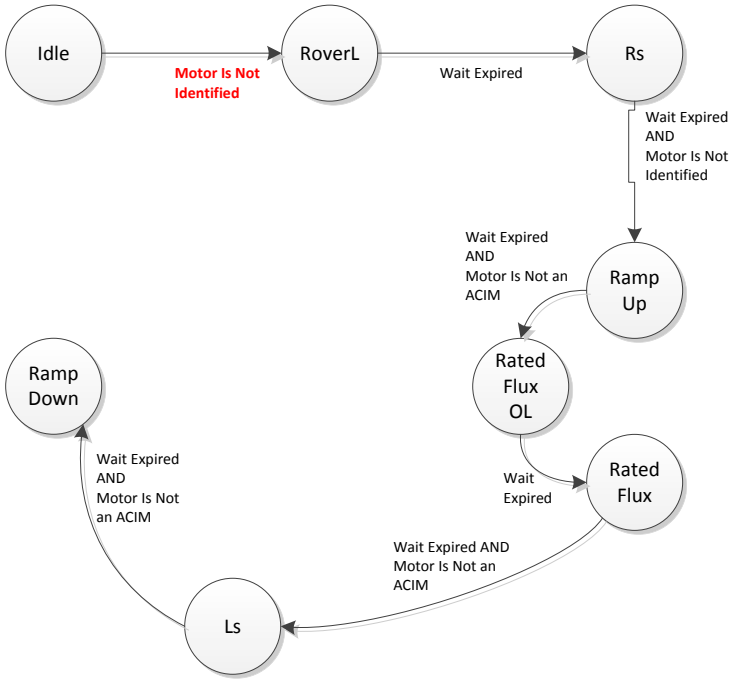


图 6-22. 定子电感 EST 状态

要识别 PMSM 电机的定子电感，相关算法会向 D 轴注入电流（也称为 ID）。 该电流在 user.h 中设置，必须为负值。 通常，该电流应为电机额定电流的 10% 到 20%，符号为负。 以下值针对 4A 电机设置，因此，用于识别定子电感的电流设置为 -0.5A，如下所示：

```
#define USER_MOTOR_IND_EST_CURRENT (-0.5)
```

定子电感识别所耗费的时间在 user.c 中按如下设置：

```
pUserParams->LsWaitTime[EST_Ls_State_Init] = (uint_least32_t)( 3.0*USER_EST_FREQ_Hz);
pUserParams->LsWaitTime[EST_Ls_State_Fine] = (uint_least32_t)(30.0*USER_EST_FREQ_Hz);
```

图 6-23 显示了运行电感识别状态所耗费的时间。 图中还显示了注入电流的幅值。 尽管我们将电流设置为 -0.5A，该电流注入 D 轴，这样即可将其视为相电流波形中幅值为 0.5A、使负载保持移动所需的电流。 由于需要从转轴移动所有机械负载，因此电流幅值将接近所示的 0.5A。

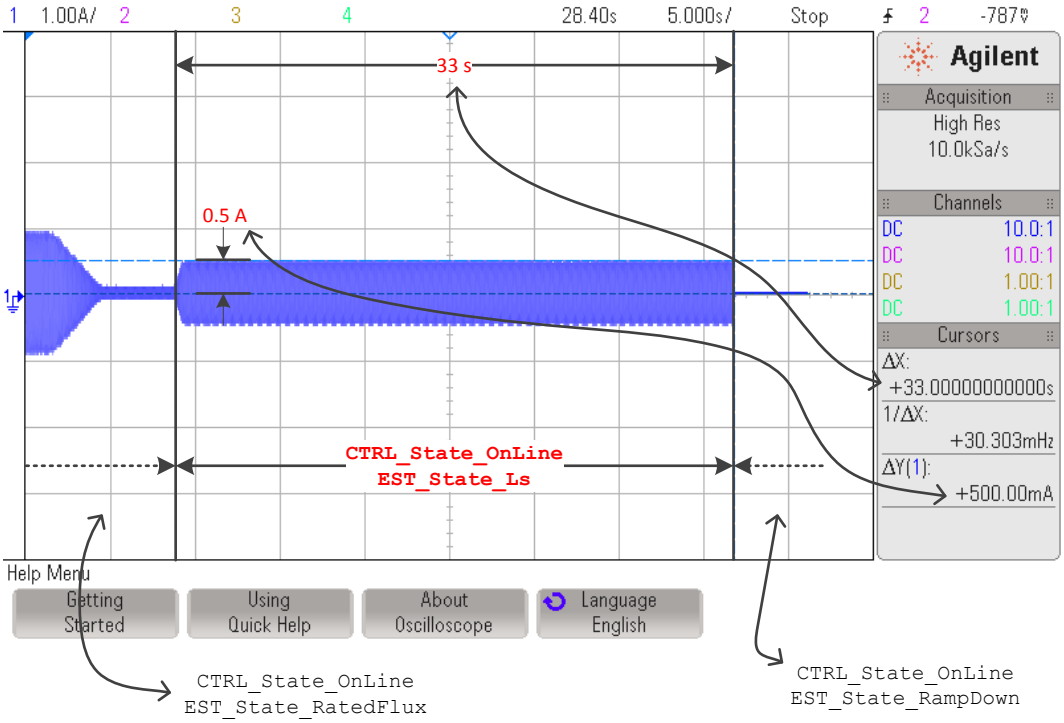


图 6-23. Ls 识别时的注入电流

6.5.7.1 斜升电流

电流增加到指定电流时的初始斜率由每秒电阻估算所使用的电流来设置。 例如，以采用以下配置的 1A 电阻估算电流为例：

```
#define USER_MOTOR_RES_EST_CURRENT (1.0)
```

-0.5A 电感估算电流配置如下：

```
#define USER_MOTOR_IND_EST_CURRENT (-0.5)
```

在这两种配置中，从增加 0.5A 电流到将电流注入 D 轴共需耗费 0.5s，如图 6-24 所示。

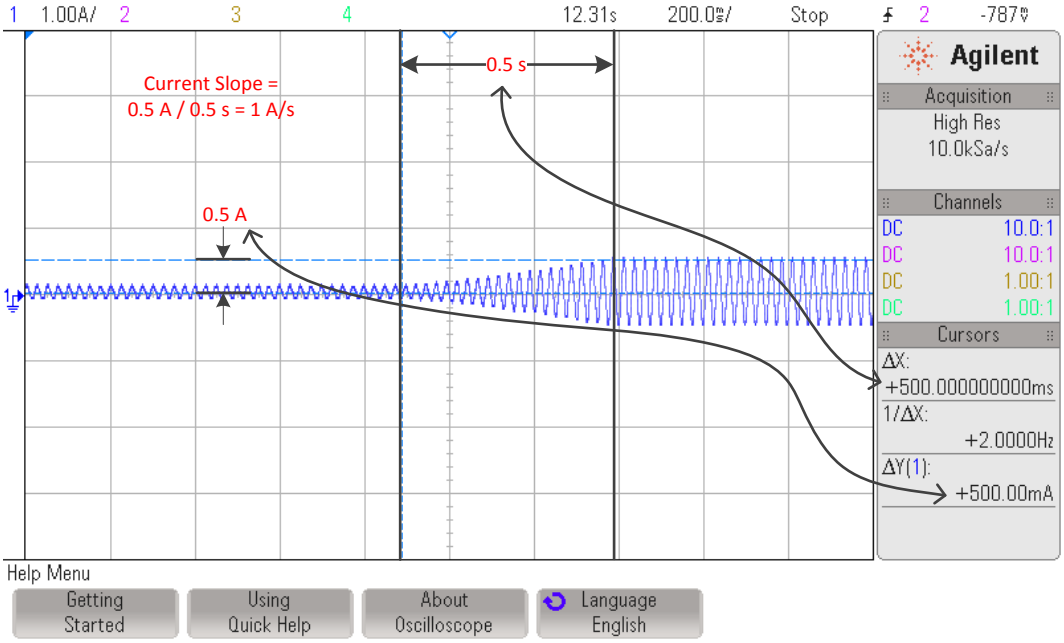


图 **6-24. Ls** 识别时的电流斜坡

6.5.7.2 Ls 识别故障排除

请参见6.9 节。

6.5.7.3 Ls_d 和 Ls_q（定子直流电感和定子正交电感）

请记住，即使给定电机的 Ls_d 值和 Ls_q 值不同，所估算的电感将同时存储到 Ls_d 和 Ls_q 中。 换言之，该版本 InstaSPIN 不单独识别 Ls_d 和 Ls_q，而是识别平均 Ls，然后将同一平均值同时存储到 Ls_d 和 Ls_q 中。 当 InstaSPIN 今后单独识别 Ls_d 和 Ls_q 时，这两个函数将返回不同的值。 如果电机参数在 user.h 中设置为不同的 Ls_d 和 Ls_q，且忽略电机识别，则当前版本 InstaSPIN 在随后调用函数 EST_getLs_d_H 和 EST_getLs_q_H 时也将返回不同的值。

6.5.8 CTRL_State_OnLine 和 EST_State_RampDown

该状态下不执行任何特定操作，直至估算过程开始。 可将其视作状态机的过渡阶段。 如下所示，尽管有一段时间与此状态相关，但更改该时间不会影响识别变量。

```
pUserParams->estWaitTime[EST_State_RampDown] = (uint_least32_t)(2.0*USER_EST_FREQ_Hz);
```

6.5.9 CTRL_State_OnLine 和 EST_State_MotorIdentified

识别过程的最终状态也是一个过渡状态，用于告知内部状态机电机已被识别。 在 EST_State_MotorIdentified 过渡状态结束后，CTRL 和 EST 这两个状态机均会返回空闲状态。 用户可通过调用以下函数来检查电机是否已被识别。 如果该函数返回 TRUE，则表示电机已被识别（经过上述所有状态或使用头文件中的电机参数）：

```
gMotorVars.Flag_MotorIdentified = EST_isMotorIdentified(obj->estHandle);
```

电机完全识别后，如果用户需要再次运行电机识别过程，控制器必须重新初始化以将状态机设置为初始状态，同时将电机识别标志恢复为 FALSE。 以下函数调用会将控制器重新初始化为初始状态，并将电机识别标志恢复为 FALSE：

```
// set the default controller parameters
CTRL_setParams(ctrlHandle,&gUserParams);
```

有关识别过程完整状态机汇总，请参见图 6-25。

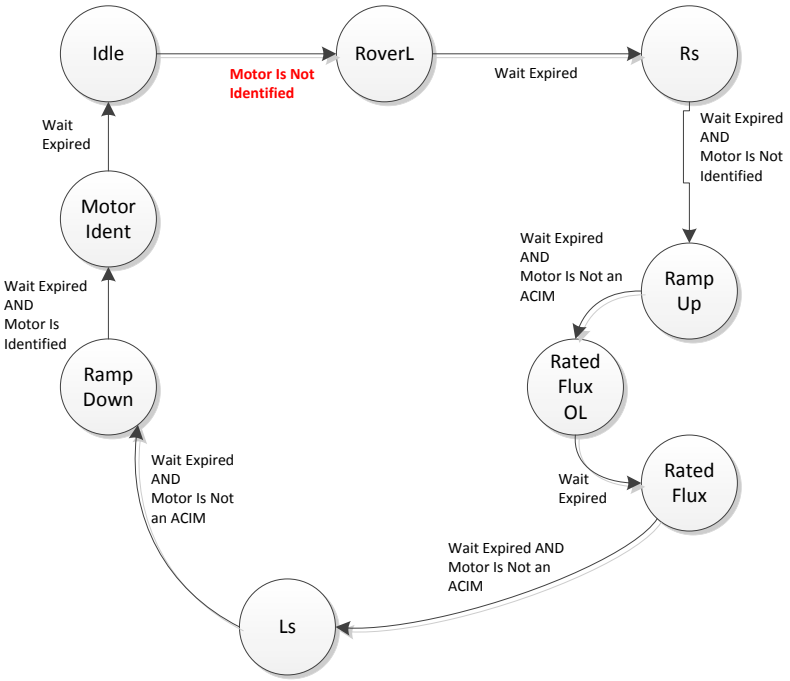


图 **6-25. EST** 状态图中的完整 **PMSM** 电机识别过程

图 6-26 也显示了完整的 PMSM 电机识别过程，并绘制了一个相电流。

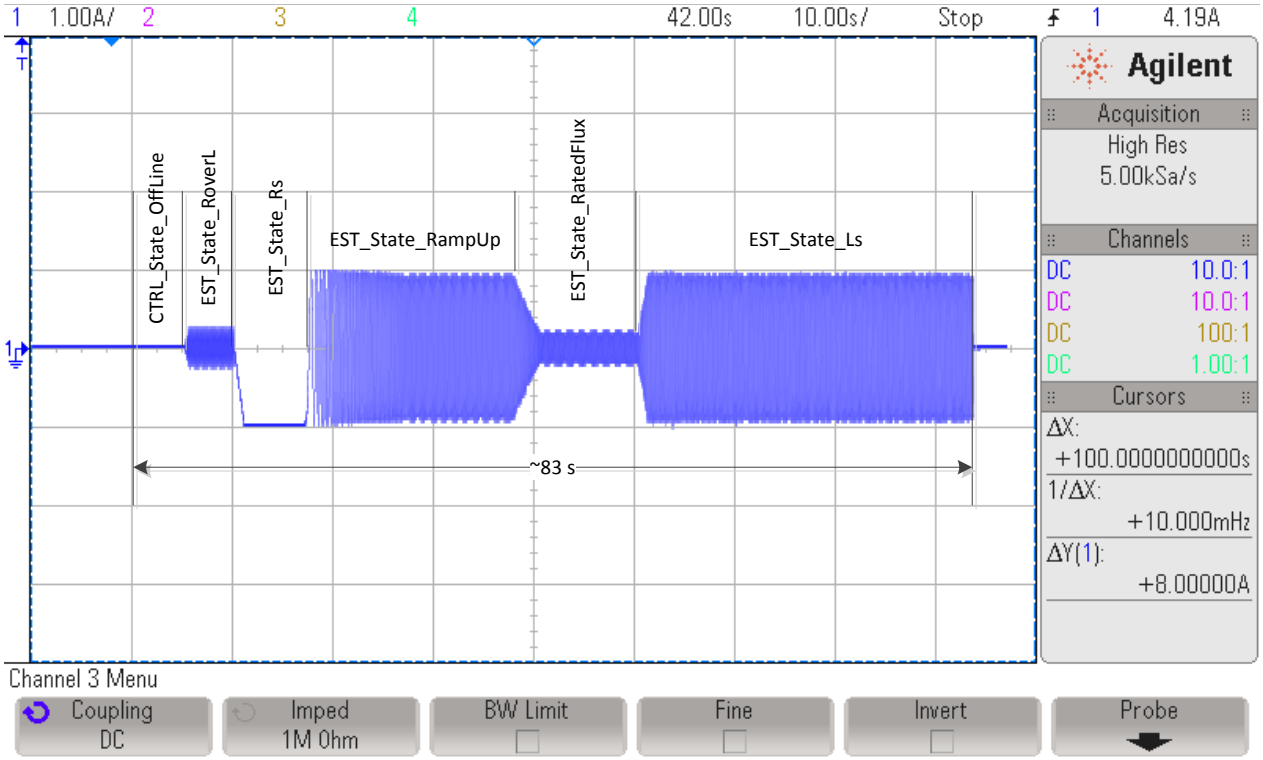


图 6-26. 完整 PMSM 电机识别过程的相电流测量

6.5.10 CTRL_State_Idle 和 EST_State_Idle

电机完全识别后，这两个状态机均设置为空闲状态。

6.6 ACIM 电机完全识别

对 ACIM 电机进行完全识别时，图 6-27 显示了控制器和估算器状态机内部发生事件的顺序。

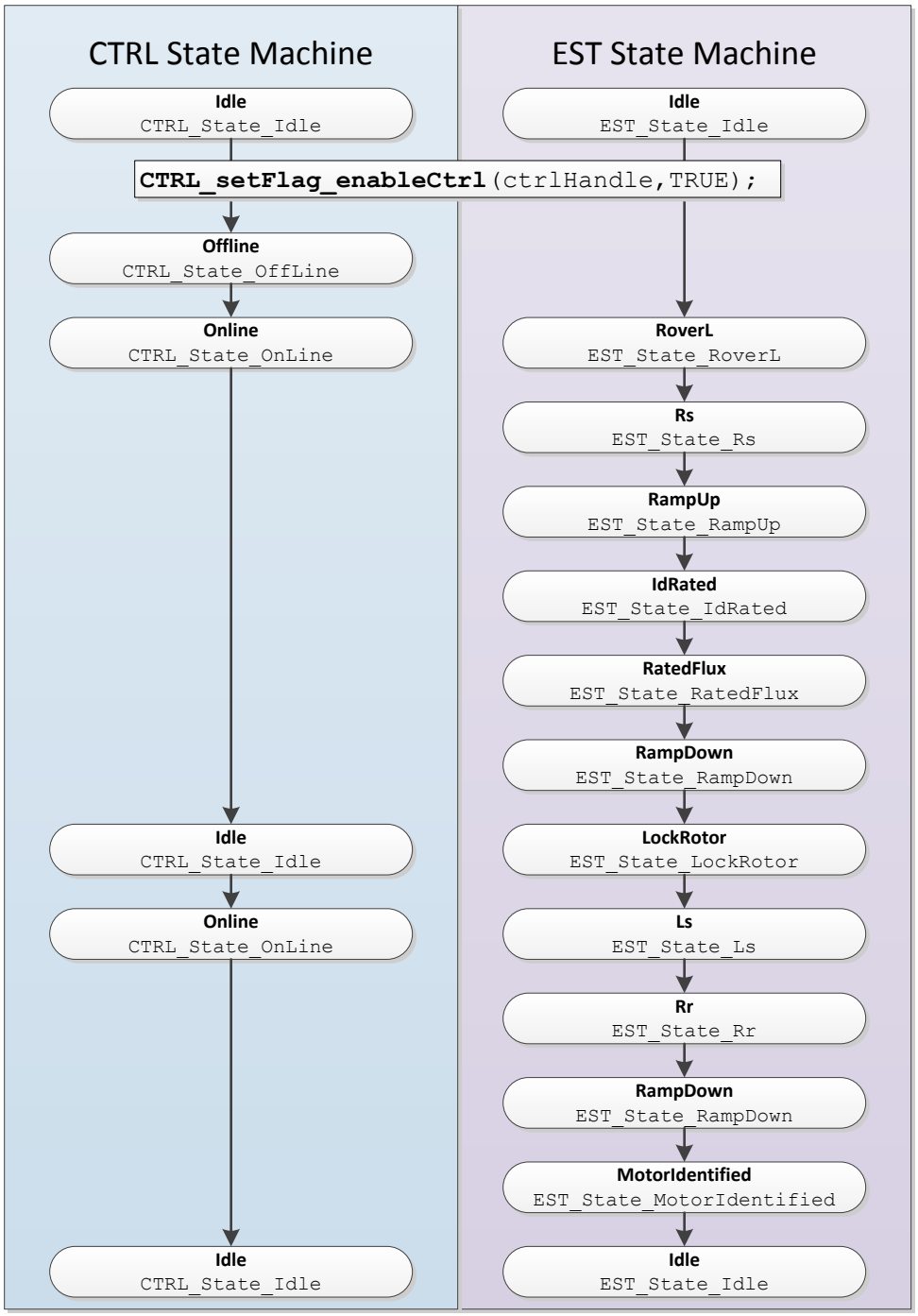


图 **6-27. ACIM** 完全识别 **- CTRL** 和 **EST** 状态顺序

启用控制器之前，代码得知以下两个条件为真时将执行电机完全识别：

1. 电机尚未识别
2. 未使用任何来自 user.h 的参数。

```
if( (EST_isMotorIdentified(obj->estHandle) == FALSE) &&
    (CTRL_getFlag_enableUserMotorParams(ctrlHandle) == FALSE))
```

本文档接下来将详细介绍识别期间的各个状态。

6.6.1 CTRL_State_Idle 和 EST_State_Idle

两个电机的控制器和估算器状态机的空闲状态均相同。 有关该状态的更多信息，请参见6.5.1 节。

6.6.2 CTRL_State_OffLine 和 EST_State_Idle

当控制器处于离线状态时，将进行偏移校准。 有关该状态的更多信息，请参见6.5.2 节。

6.6.3 CTRL_State_OnLine 和 EST_State_RoverL

要计算电流控制器增益，需执行与 PMSM 电机相同的过程和操作。 有关该状态的更多信息，请参见6.5.3 节。

6.6.4 CTRL_State_OnLine 和 EST_State_Rs

当估算器处于 Rs 状态时，将进行定子电阻校准。 有关该状态的更多信息，请参见6.5.4 节。

6.6.5 CTRL_State_OnLine 和 EST_State_RampUp

定子电阻识别完成后，将执行新的估算器状态。 接下来的这一状态称为斜升状态，即 EST_State_RampUp（图 6-28）。 在识别过程的这一状态期间，电机会加速到某一速度，从而开始识别其它参数。 该状态期间不会识别任何参数，而是启动相应条件。 此状态的影响因素有多种。

就功能而言，有关 EST_State_RampUp 状态期间相关情况的详细说明，请参见6.5.5 节。

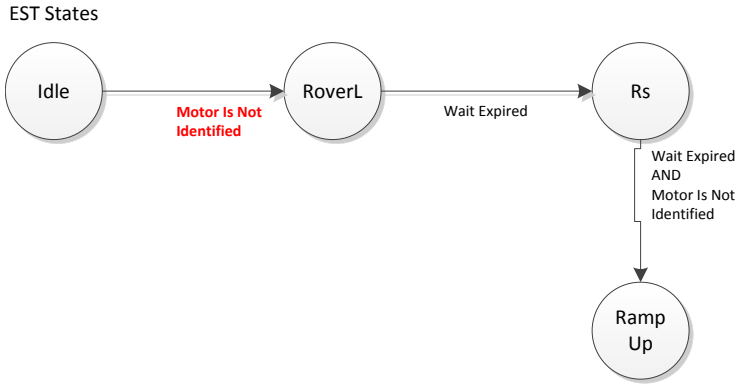


图 6-28. 斜升 EST 状态

6.6.5.1 ACIM 最终斜升速度

PMSM 和 ACIM 电机识别过程在此状态下的唯一差别为：ACIM 电机采用 5Hz 典型频率，而 PMSM 电机采用 20Hz 典型频率。

```
// During Motor ID, maximum commanded speed in Hz
#define USER_MOTOR_FLUX_EST_FREQ_Hz (5.0)
```

以下示波器图显示的是经过放大的相电流图，其中的频率斜升到 5Hz。 请注意，6.5.5 节之前介绍 PMSM 的默认斜升加速度为 2.0Hz/s，该值同样用于图 6-7 所示的 ACIM 示例。

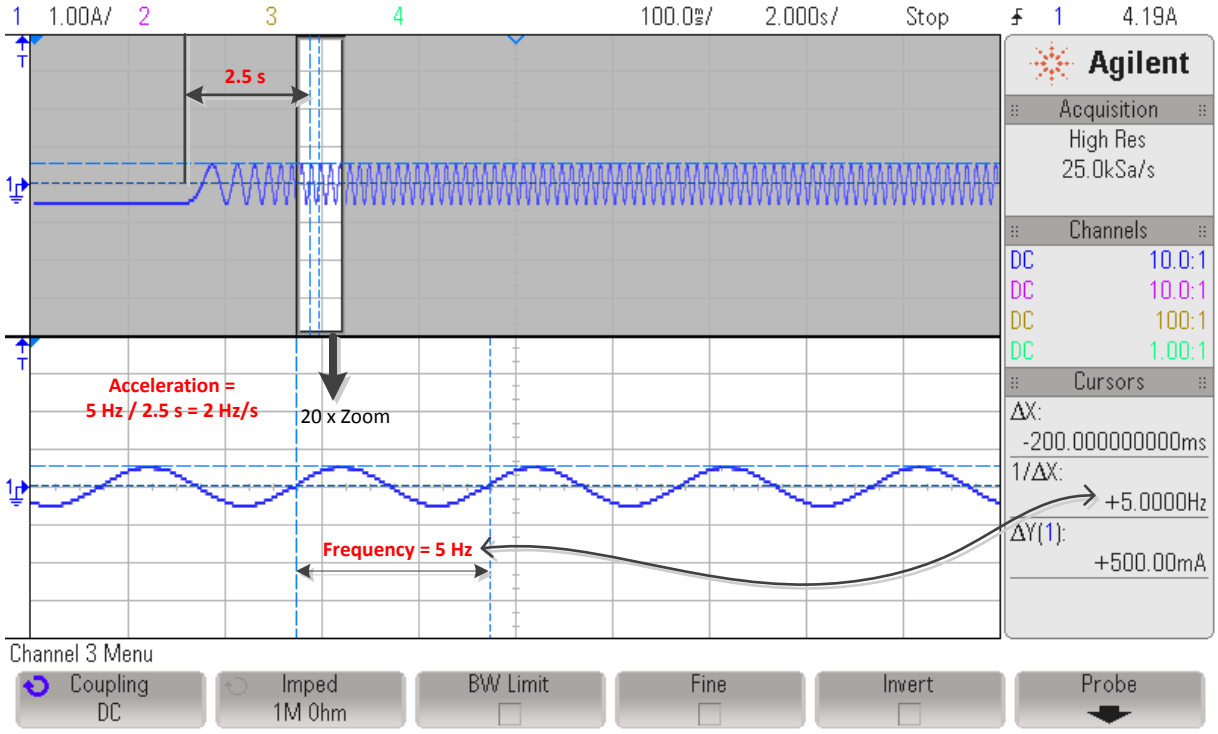


图 **6-29. ACIM** 斜升加速度示波器图

6.6.6 *CTRL_State_OnLine* 和 *EST_State_IdRated*

一旦电机以 user.h 中设置的指定频率运行，便会立即开始 Id 额定电流识别过程（图 6-30）。

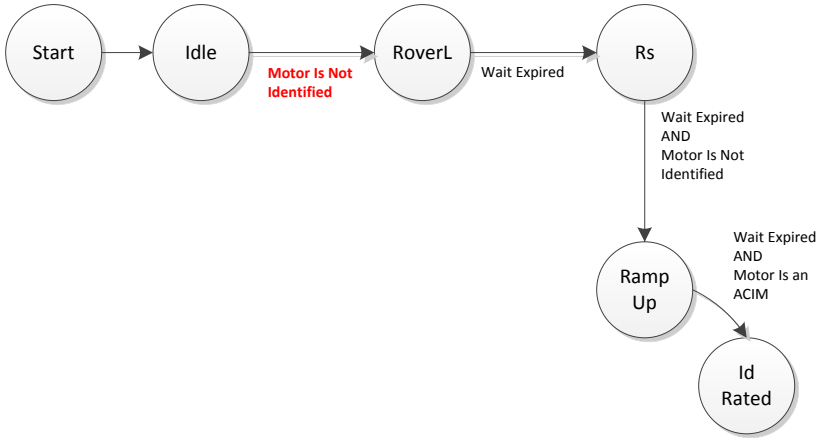


图 **6-30. ACIM Id** 额定电流 **EST** 状态

在 EST_State_IdRated 状态下，估算器用于计算产生特定磁通所需的电流。

在此状态期间，需考虑 user.h 和 user.c 中的多个参数。 第一个参数是此状态的持续时间，该参数在 user.c 文件中按如下配置：

```
pUserParams->estWaitTime[EST_State_IdRated]=(uint_least32_t)(20.0*USER_EST_FREQ_Hz);
```

在此期间，注入 Id 将以 **user.h** 中的 USER_IDRATED_DELTA 定义的增量开始增加：

```
//! \brief Defines the IdRated delta to use during estimation
//!
#define USER_IDRATED_DELTA  (0.00002)
```

通过上述增量方法，注入 d 轴的电流将不断增加，直至产生的磁通达到 user.h 中的 USER_MOTOR_RATED_FLUX 所指定的值：

```
#define USER_MOTOR_RATED_FLUX  (0.8165*220.0/60.0)
```

有关设置 ACIM 电机额定磁通的详细信息，请参见节 6.4.5.4。

当 IdRated 状态开始时，电流将不断增加，直至电机中出现所需磁通。图 6-31 显示电流不断增加并且随后稳定下来的过程。

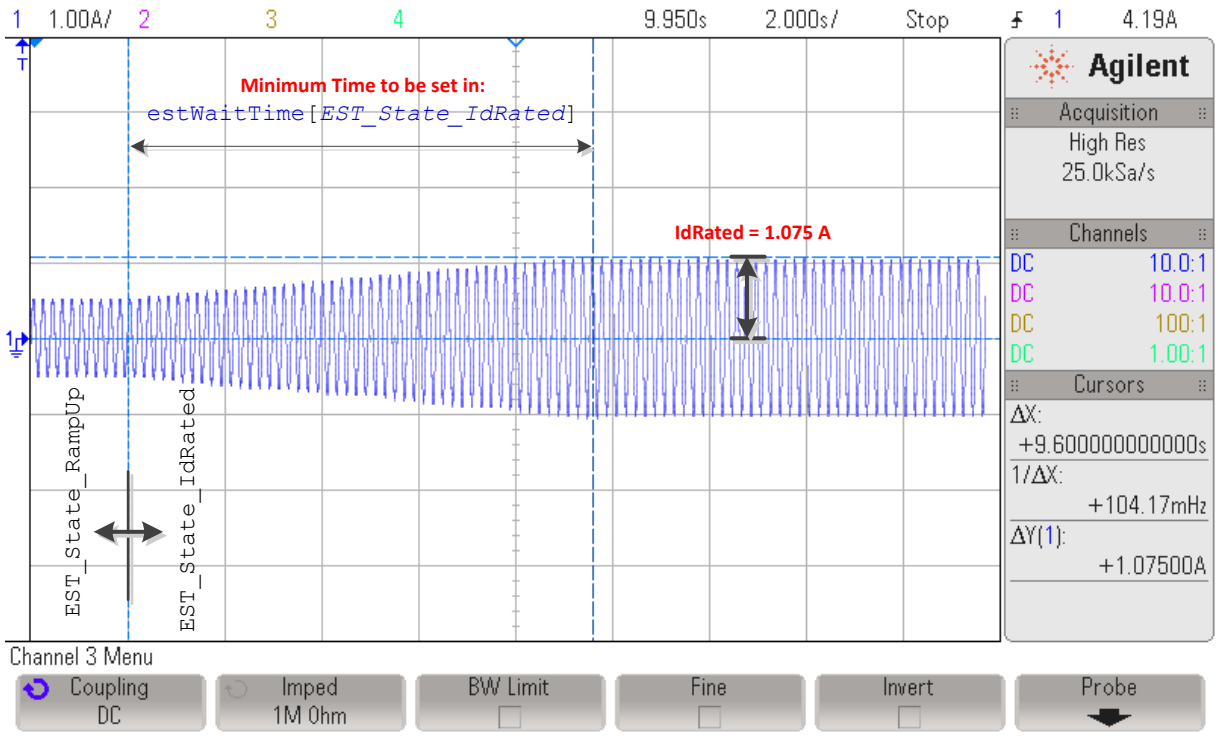


图 6-31. Id 额定电流 EST 状态期间的相电流示波器图

Id 电流达到稳定的时间取决于电机，建议调整 pUserParams->estWaitTime[*EST_State_IdRated*] 中配置的时间，使电流能够在没有过度振荡的情况下稳定到某个值。

6.6.6.1 减少振荡以改善 Id 额定电流测量

IdRated 识别期间要调节的另一个参数是该电流的递增增量。 如果该值对于特定电机而言过高，即使在执行 IdRated 识别时，余下的振荡也会持续很长时间。 例如，图 6-32 显示了使用以下参数时的电流情况。

```
//! \brief Defines the IdRated delta to use during estimation
//!
#define USER_IDRATED_DELTA  (0.0001)
```

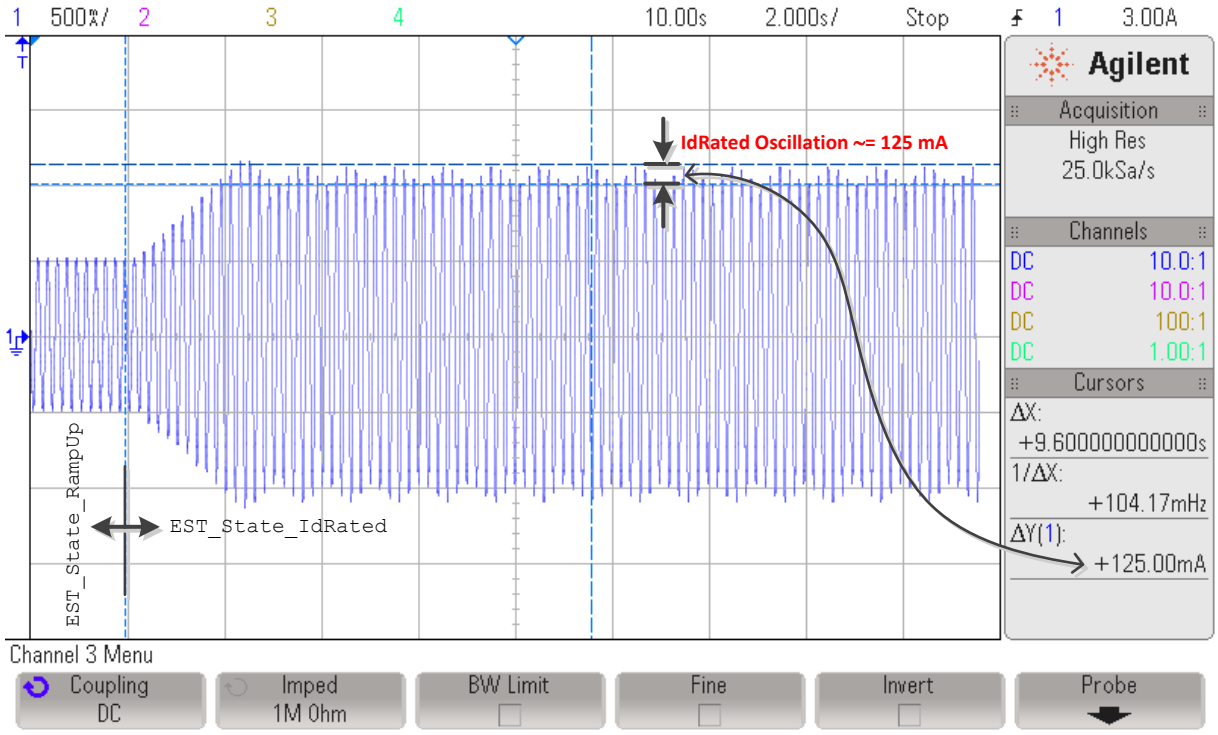


图 6-32. Id 额定电流测量期间的相电流振荡

如示波器图中所示，即使电流增长速度加快，余下的振荡也不会使 Id 额定电流达到稳定值。 尝试用一个较小的值 0.00002 来代替 0.0001 可提高稳态电流的稳定性，如图 6-33 所示。

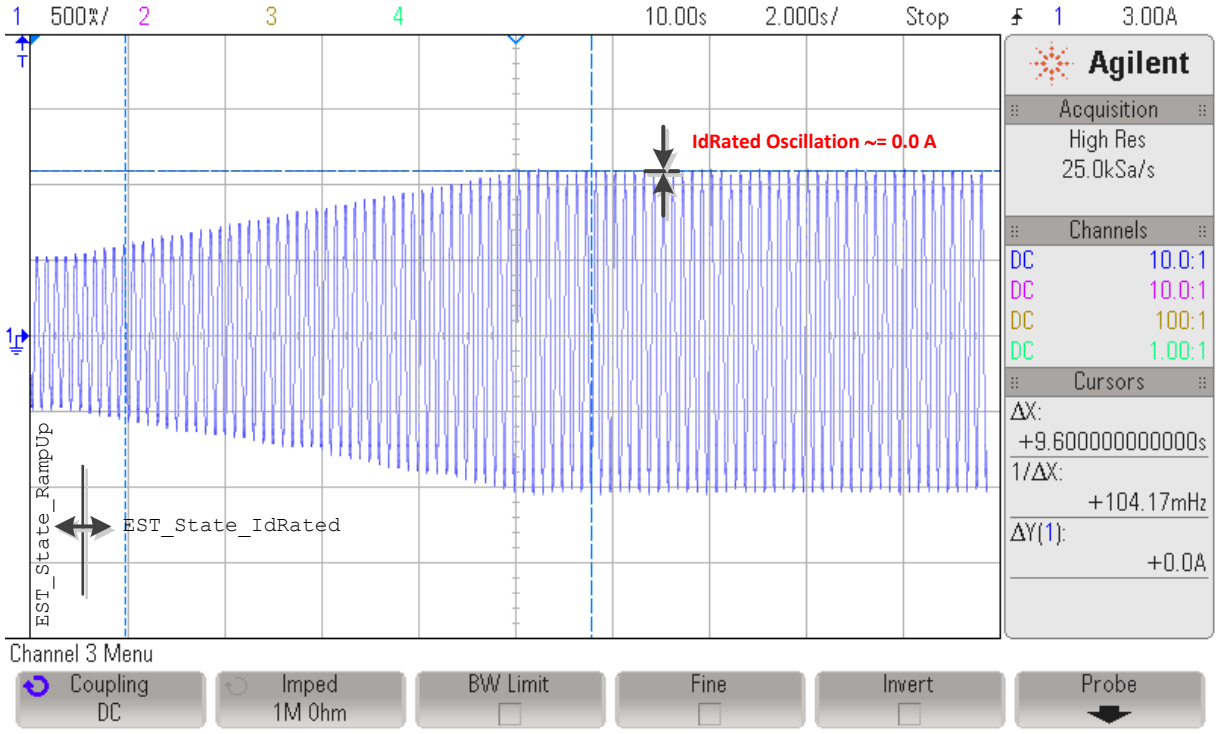


图 6-33. Id 额定电流测量期间减少相电流振荡

6.6.6.2 读取 Id 额定电流最终值

该状态结束时，用户可以通过以下函数读取所识别的 Id 额定电流，也称为 ACIM 电机的额定磁化电流。

```
// get the Id Rated, or rated magnetizing current
IdRated = EST_getIdRated(obj->estHandle);
```

6.6.7 CTRL_State_OnLine 和 EST_State_RatedFlux

Id 额定电流识别完成后，接下来的状态为额定磁通状态（图 6-34）。

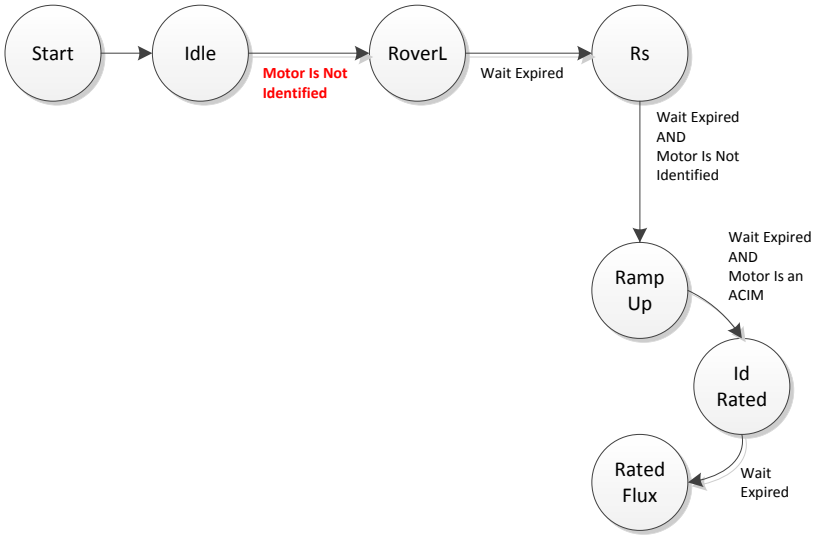


图 6-34. ACIM 额定磁通 EST 状态

在此状态期间，估算器内的闭环系统使用之前识别的 Id 额定电流来计算转子和定子间的磁链。 仅使用 50% 的 Id 额定电流。 该状态结束时，通过 Id 额定电流计算得出的额定磁通保存为机器的额定磁通。

对于额定磁通状态而言，两种电机的电流斜降时间和总测量时间均相同。 有关更多信息，请参见节 6.5.6.1 和节 6.5.6.2。

图 6-35 显示了如何实现闭环，其中的电流比 IdRated 状态的电流更加稳定，用户可在观察窗口中复查所估算的磁通，确认磁通估算值是否为 user.h 中最初设定的值。

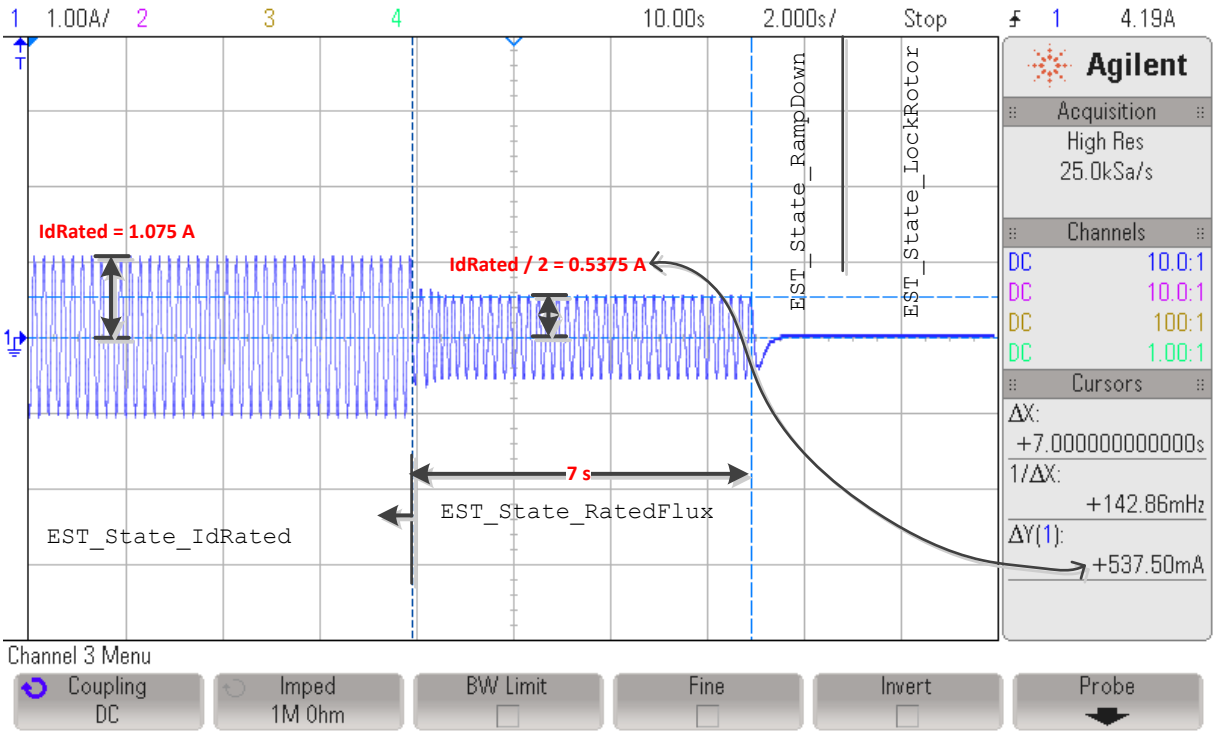


图 6-35. Id 额定电流 EST 状态期间的相电流

6.6.7.1 磁通测量故障排除

请参见6.9 节。

6.6.8 CTRL_State_OnLine 和 EST_State_RampDown

要移除电机绕组中的电流，需要运行中间状态来斜降流经电机的电流（图 6-36）。 此状态即为斜降状态，其持续时间在 user.c 文件中按如下设置：

```
pUserParams->estWaitTime[EST_State_RampDown]=(uint_least32_t)(2.0*USER_EST_FREQ_Hz);
```

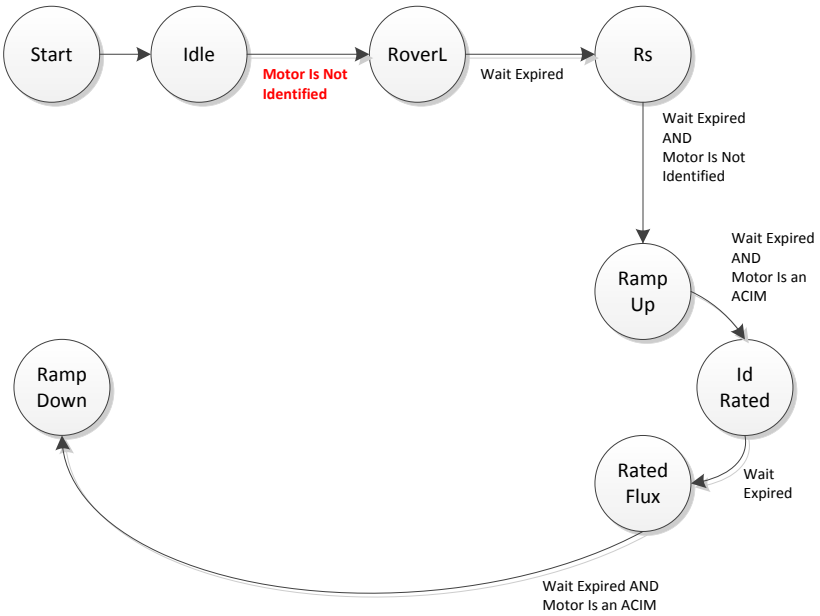


图 6-36. ACIM 斜降 EST 状态

图 6-37 由前一状态得来，显示了逐渐从电机中移除相电流，从而使电机平稳停机的过程。

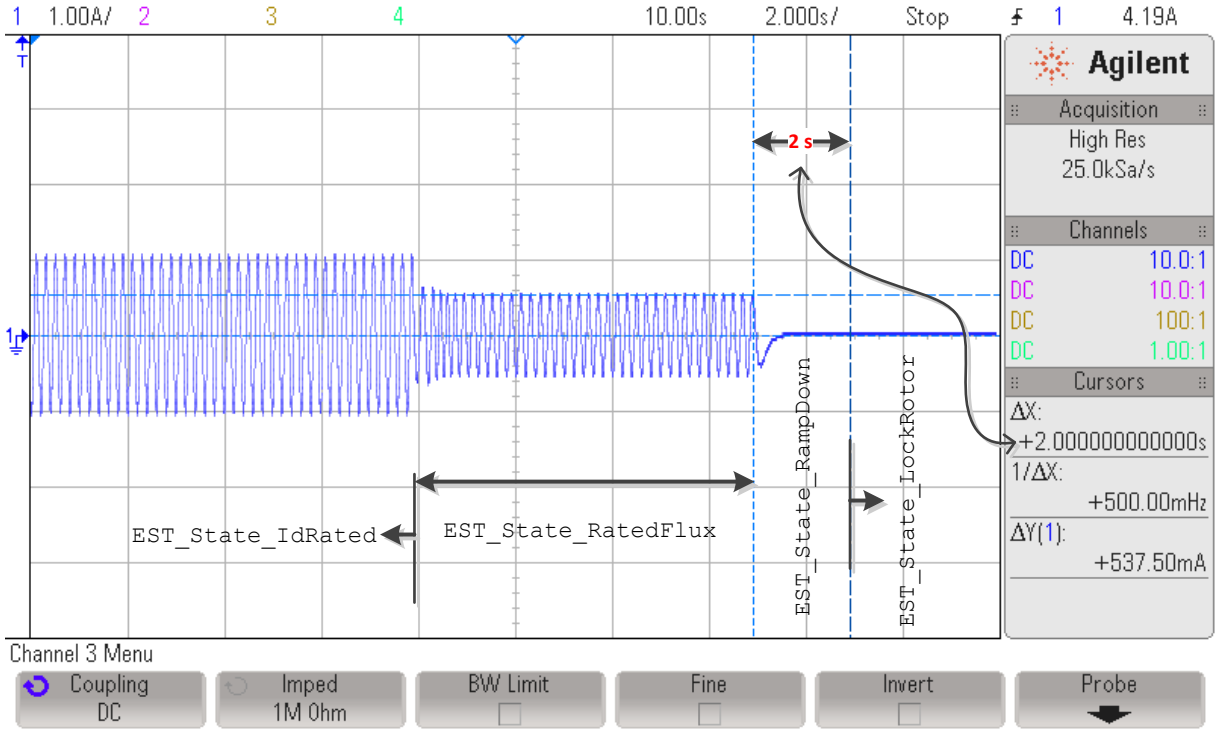


图 6-37. 进入锁定转子状态前斜降 ACIM 相电流

6.6.9 CTRL_State_Idle 和 EST_State_LockRotor

在锁定转子状态期间，一旦电机转轴锁定，状态机将等待用户重新启用控制器（图 6-38）。 要识别其余的 ACIM 电机参数：串联电感 (Ls) 和转子电阻 (Rr)，则需要锁定转子。

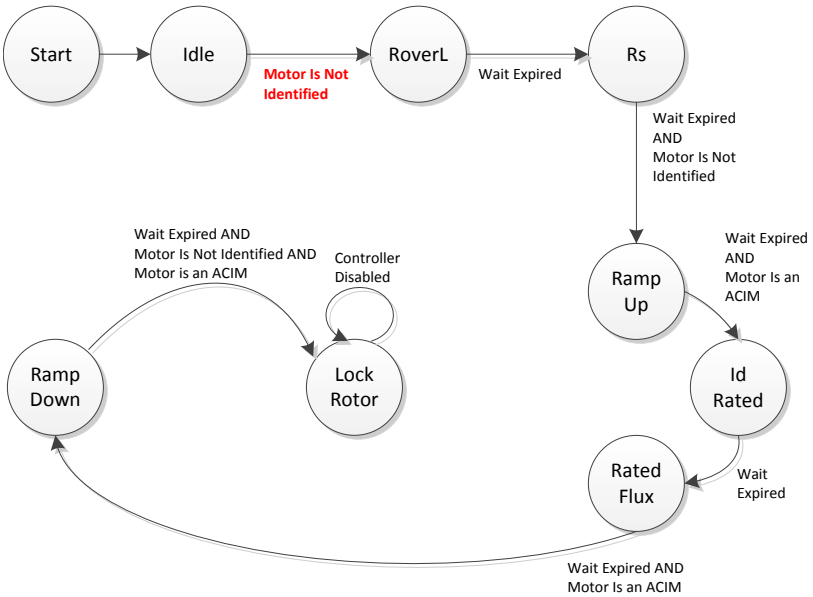


图 6-38. ACIM 锁定转子 EST 状态

状态机将一直保持此状态，直到用户通过调用以下指令重新启用控制器：

```
// enable or disable the control
CTRL_setFlag_enableCtrl(ctrlHandle, TRUE);
```

6.6.9.1 锁定转子测试故障排除

请参见6.9 节。

6.6.10 CTRL_State_OnLine 和 EST_State_Ls

锁定转子参数测量完成后，将立即开始定子电感过程（图 6-39）。

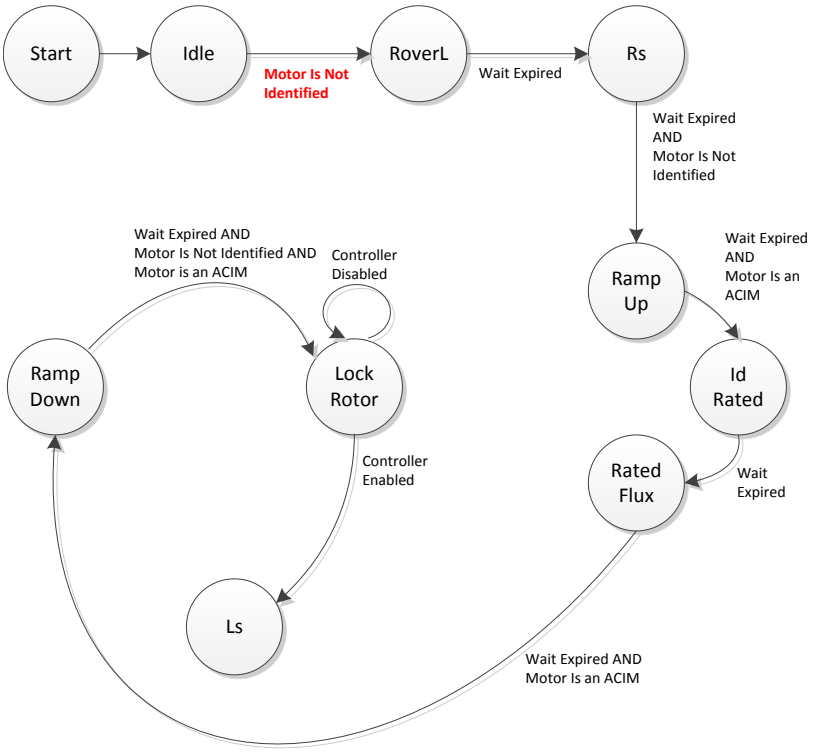


图 6-39. ACIM 定子电感 EST 状态

对于定子电感状态而言，两种电机的步骤是相同的。 有关详细信息，请参见6.5.7 节。

图 6-40 显示的是 ACIM 电机电流流经 Ls 状态并进入后续状态的过程。

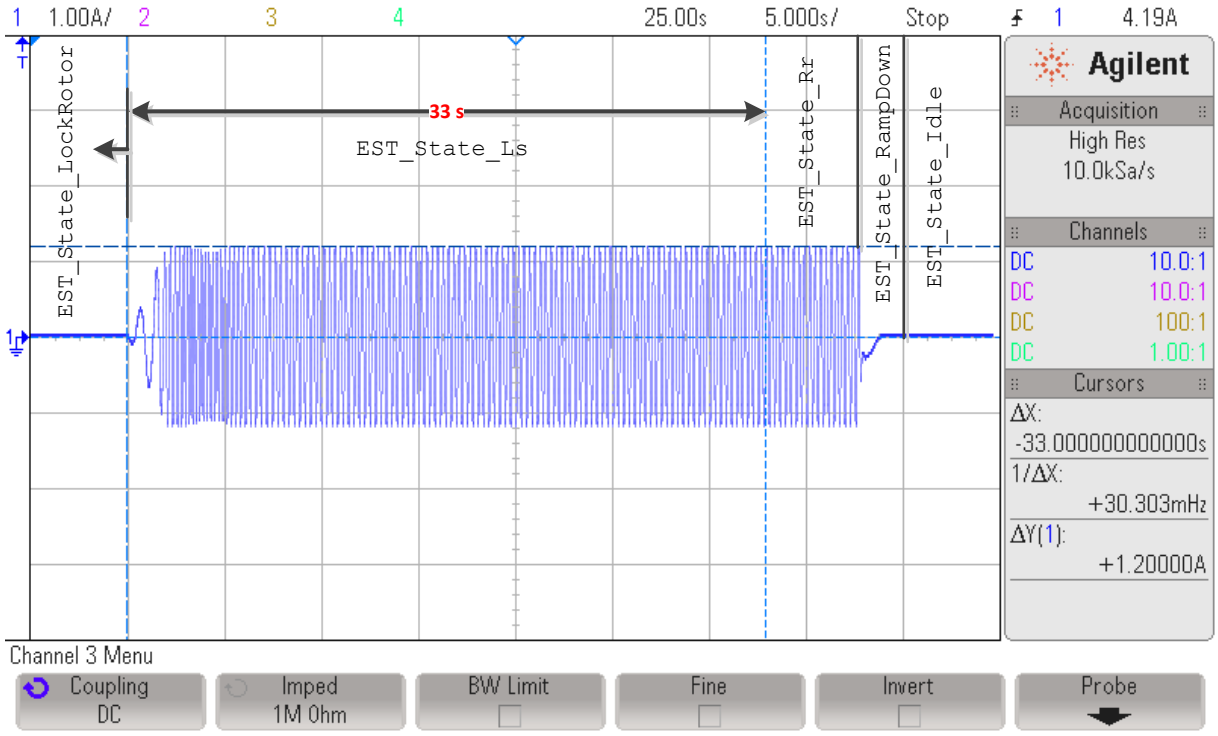


图 **6-40.** 定子电感 **EST** 状态下的 **ACIM** 电流

6.6.11 CTRL_State_OnLine 和 EST_State_Rr

定子电感测量完成后，将立即开始转子电阻过程（图 6-41）。

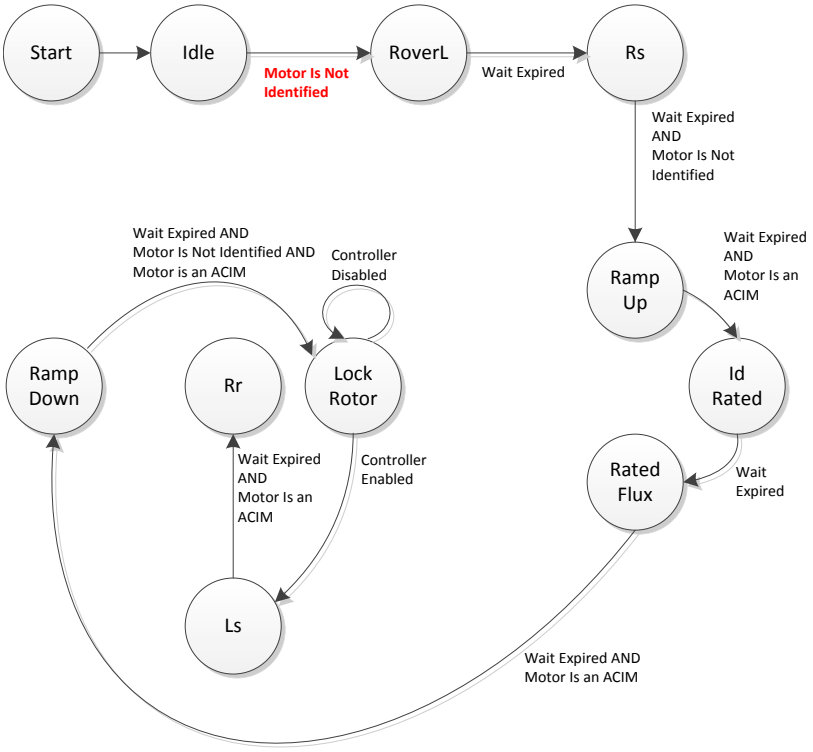


图 **6-41.** 转子电阻 **EST** 状态

ACIM 电机转子电阻识别过程的持续时间由 user.c 中配置的以下数组成员所配置：

```
pUserParams->estWaitTime[EST_State_Rr]=(uint_least32_t)(5.0*USER_EST_FREQ_Hz);
```

图 6-42 显示了 Rr 识别过程注入的电流。 事实上，Ls 和 Rr 识别过程的电流波形并无差异。

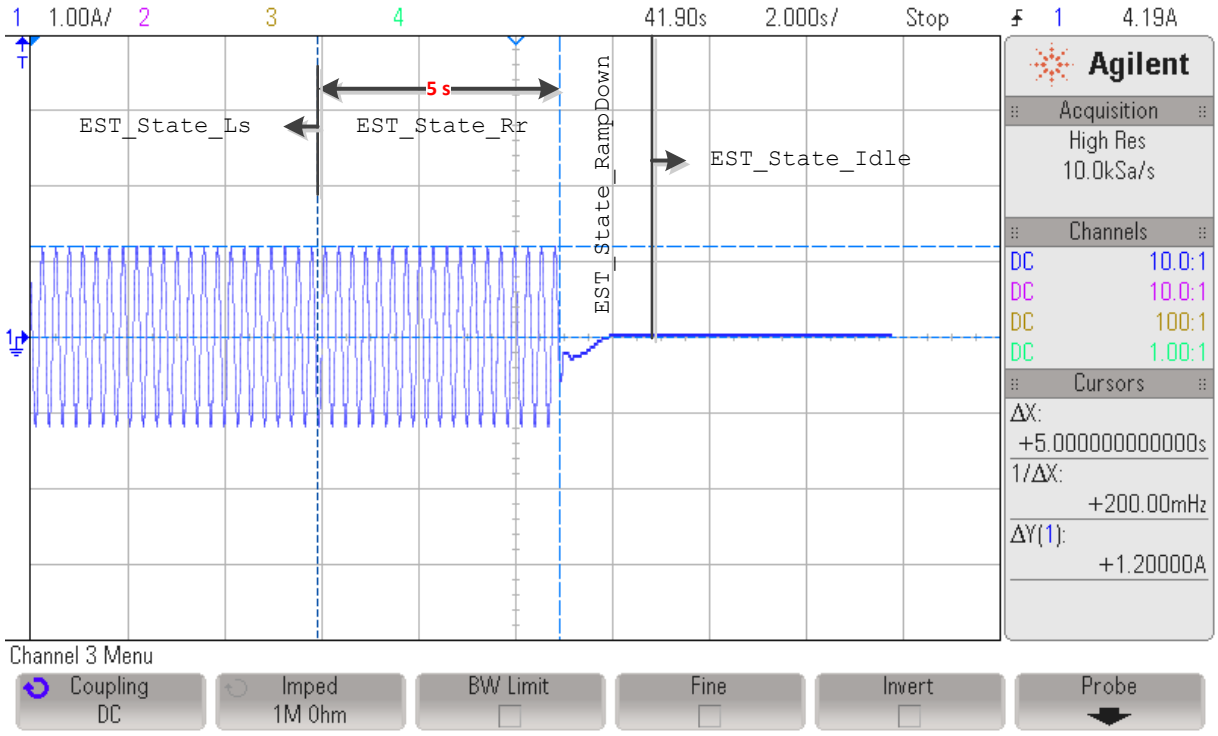


图 6-42. Rr 识别时的注入电流

6.6.12 CTRL_State_OnLine 和 EST_State_RampDown

转子电阻测量完成后，将立即开始斜降过程（图 6-43）。

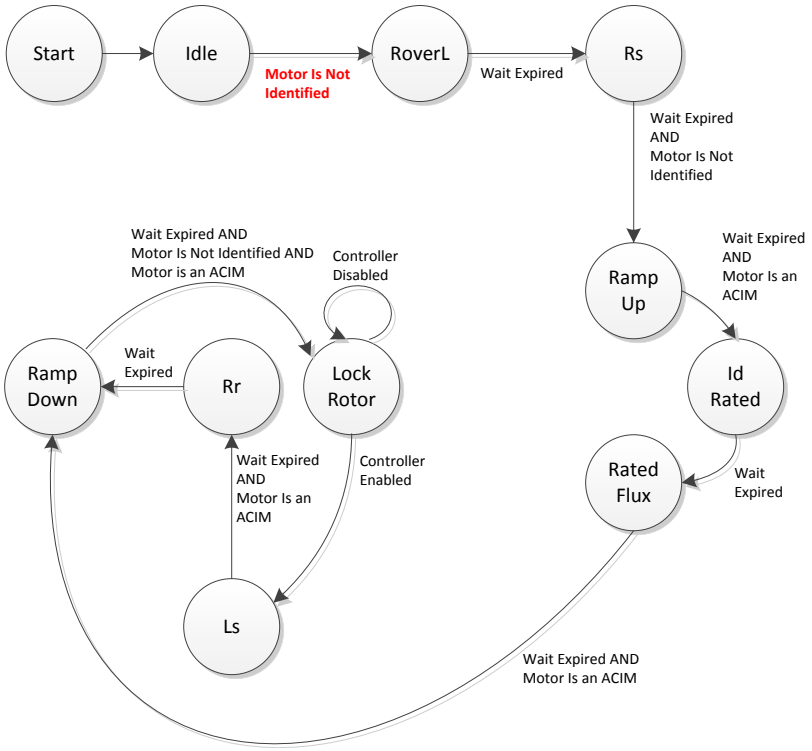


图 **6-43. Rr** 完成后的 **ACIM** 斜降 **EST** 状态

图 6-44 显示了完全识别后的 ACIM 电流波形，请注意，电流以斜降方式从电机中移除。 有关该状态的更多信息，请参见6.5.8 节。

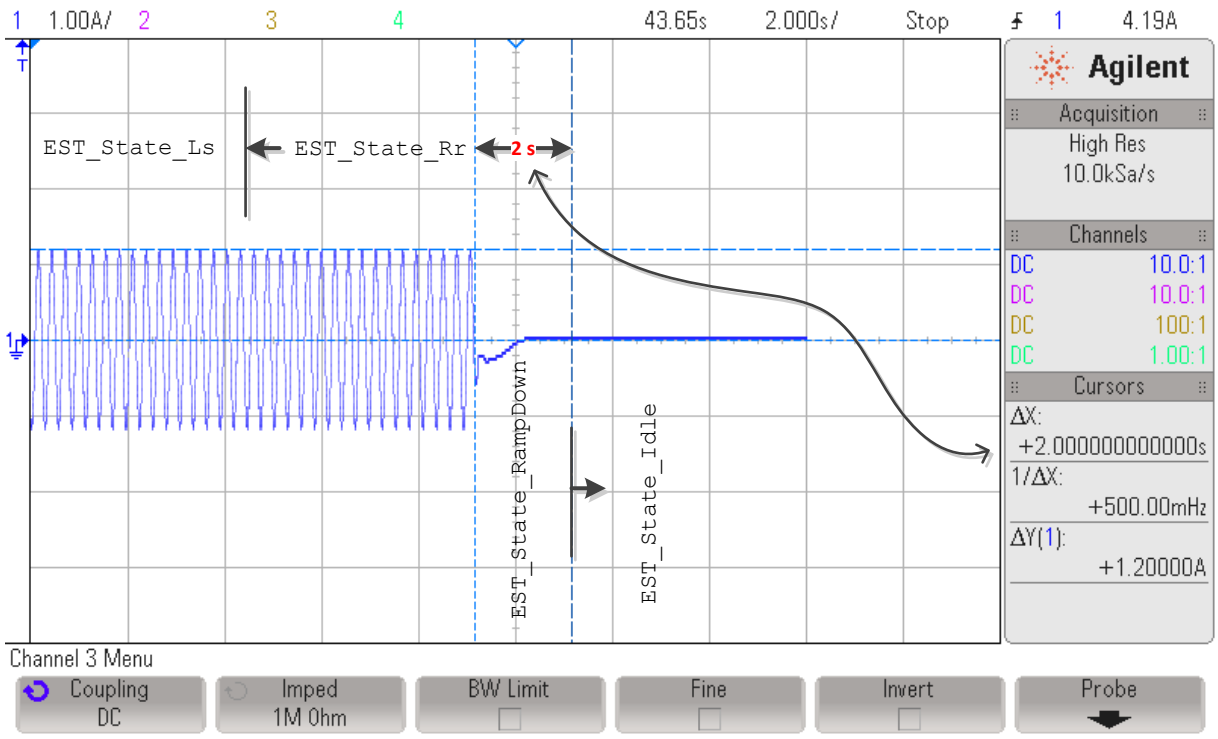


图 **6-44. Rr** 和斜降期间的 **ACIM** 电流

6.6.13 CTRL_State_OnLine 和 EST_State_MotorIdentified

当估算器处于电机已识别状态时，ACIM 电机的完全识别状态即完成（图 6-45）。 有关该状态的更多信息，请参见6.5.9 节。

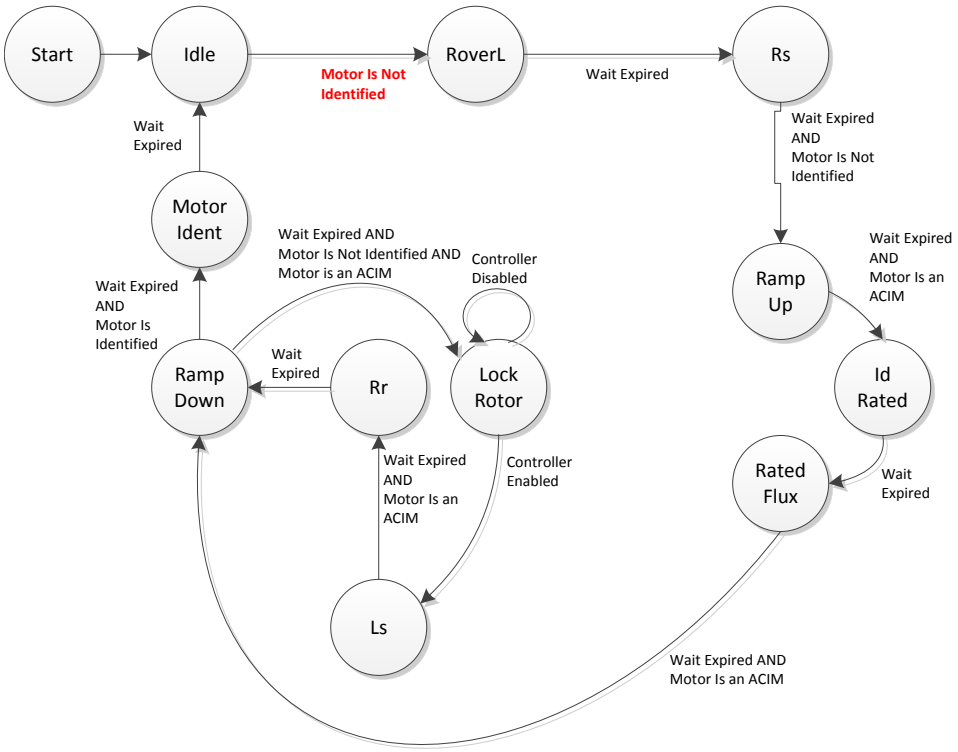


图 6-45. EST 状态图中的完整 ACIM 电机识别过程

图 6-46 也显示了完整的 ACIM 电机识别过程，并绘制了一个相电流。

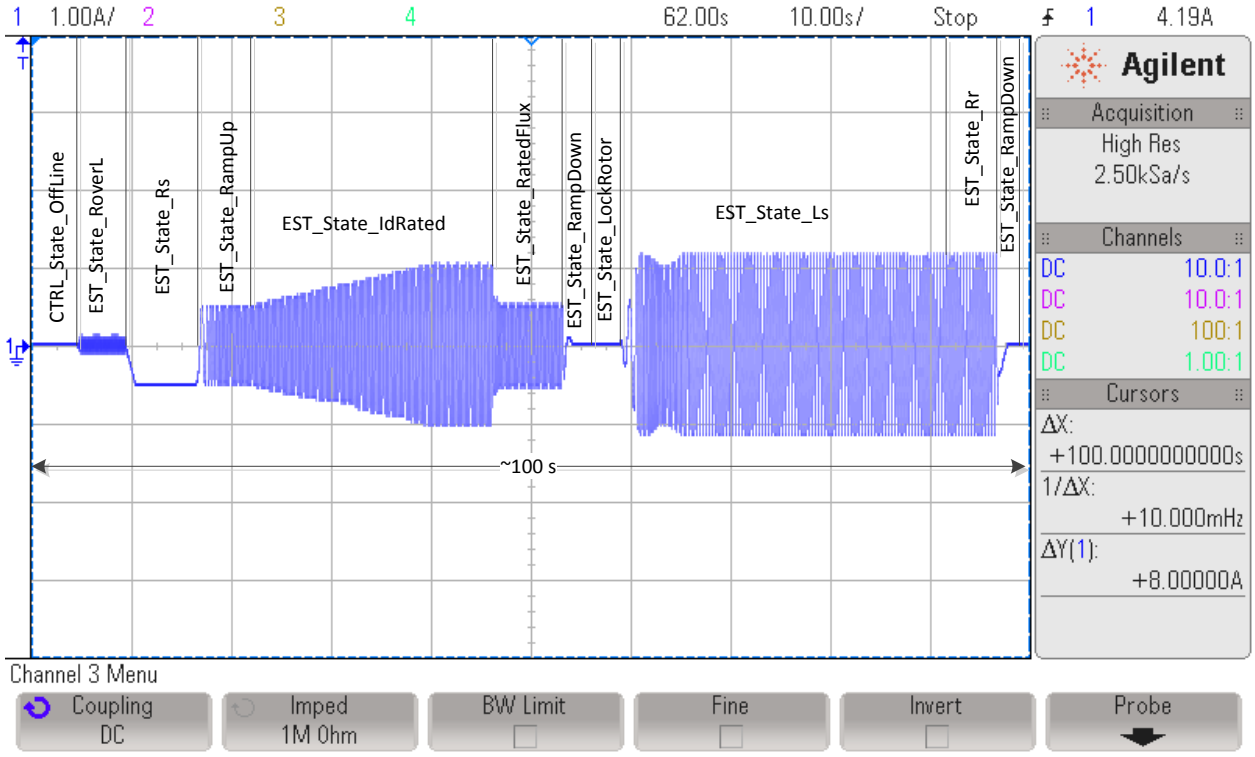


图 **6-46.** 完整 **ACIM** 电机识别过程的相电流

6.6.14 CTRL_State_Idle 和 EST_State_Idle

电机完全识别后，这两个状态机均设置为空闲状态。

6.7 PMSM 和 ACIM 电机识别重校准

重校准是电机识别的一部分，因为该过程将识别和调整两个参数：电机的偏移和定子电阻。

6.7.1 完全识别后的 PMSM 和 ACIM 电机重校准

本小节介绍 PMSM 和 ACIM 电机重校准。 电机重校准用于微调或重校准硬件偏移和定子电阻。 与 PMSM 和 ACIM 电机完全校准相比，重校准仅涵盖估算器状态机的三个状态，如公式 49 所示。 PMSM 电机和 ACIM 电机的重校准完全相同。 如本文档后续内容所述，开发板偏移和定子电阻的重校准可单独启用或禁用。

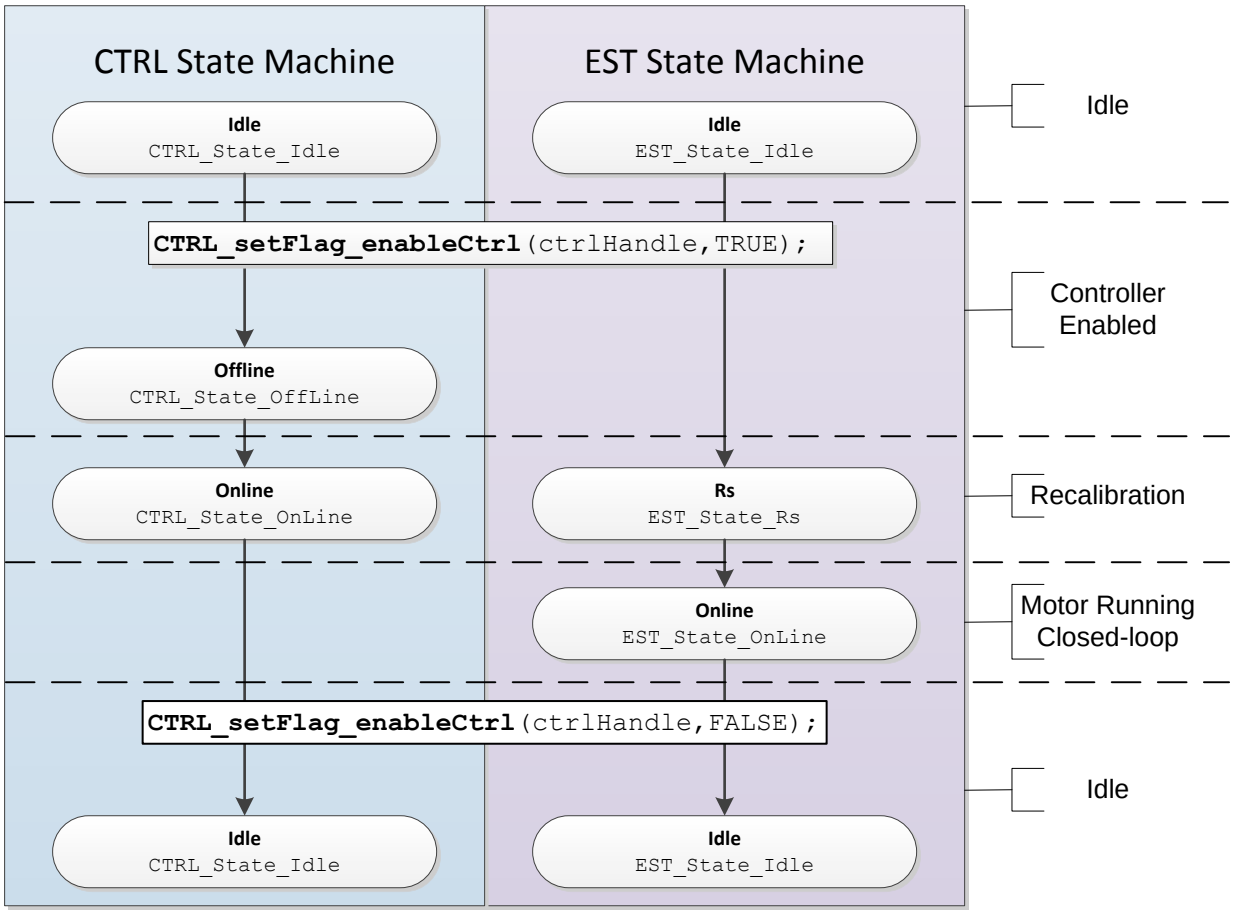


图 6-47. PMSM 和 ACIM 重校准 - CTRL 和 EST 状态顺序

当两个状态机均在线且在线状态将持续到控制器禁用时，电机将闭环运行（如图 6-48 所示）。

电机重校准将在以下两种情况下执行：电机已完成完全识别过程时以及电机参数已通过头文件 user.h 提供时。 这两种情况下执行的状态完全相同，下文将对此详细说明。

启用控制器之前，代码得知以下两个条件为真时将执行电机重校准：电机已识别以及未使用任何来自 user.h 的参数。

```
if( (EST_isMotorIdentified(obj->estHandle) == TRUE) &&
(CTRL_getFlag_enableUserMotorParams(ctrlHandle) == FALSE))
```

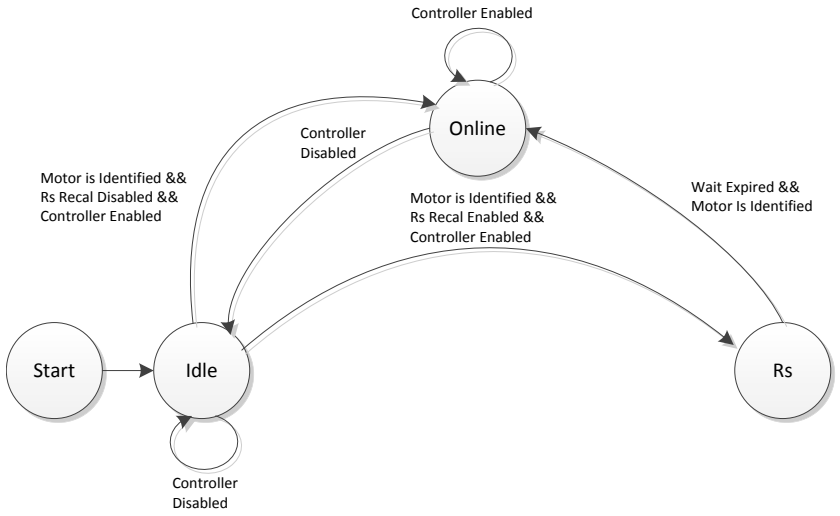


图 **6-48.** 电机重校准 **EST** 状态

6.7.1.1 启动阶段注意事项

尽管执行重校准需耗时，但仍建议始终启用偏移重校准和 Rs 重校准以确保各个参数精准，然后再运行电机，尤其是在电机长时间空闲后。 如果电机已长时间未通过代码运行，则环境温度可能会影响电机定子电阻，因此建议进行 Rs 重校准。 开发板偏移取决于硬件，所以此类偏移重校准并非关键步骤，不过，无源元件的寿命以及温度变化可能会影响硬件偏移，因此建议定期调整这些偏移。

6.7.1.2 CTRL_State_Idle 和 EST_State_Idle

在启用控制器前，控制器状态机和估算器状态机均处于空闲状态，分别表示为 CTRL_State_Idle 和 EST_State_Idle。 这也称作这两个状态机的非活动状态。

6.7.1.3 CTRL_State_OffLine 和 EST_State_Idle

通过以下函数启用控制器可使其退出空闲状态：

```
CTRL_s etFlag_enable Ctrl(ctrlHandle, TRUE);
```

启用控制器并且电机已识别后，控制器状态机执行的首个任务就是偏移重校准。 这种情况仅发生在启用了偏移计算时。 要检查是否启用了偏移重校准标志，用户可使用以下代码示例：

```
if(CTRL_getFlag_enableOffset(ctrlHandle) == TRUE)
```

电机已识别标志在内部检查，也可由用户通过以下代码示例检查：

```
if(EST_isMotorIdentified(obj->estHandle) == TRUE)
```

偏移重校准默认启用，但也可在启用控制器前通过以下代码示例将其启用：

```
CTRL_s etFlag_enableOffset(ctrlHandle, TRUE);
```

以下代码示例用于禁用偏移重校准。

```
CTRL_s etFlag_enableOffset(ctrlHandle, FALSE);
```

6.2 节 中说明的状态 (CTRL_State_OffLine and EST_State_Idle) 通过控制器状态机的状态 CTRL_State_OffLine 表示。 估算器在控制器离线状态期间保持空闲状态 (EST_State_Idle)。 电机完全识别期间，可绕过偏移重校准，但需要进行偏移校准。 执行电机完全识别时，不可绕过偏移校准。

有关电机完全识别过程中偏移校准的详细信息，请参见6.5.2 节。 从图 6-48 中可以看出，“RoverL”状态并不是重校准过程的一部分。

图 6-49 显示了相电流示波器图的相应状态。

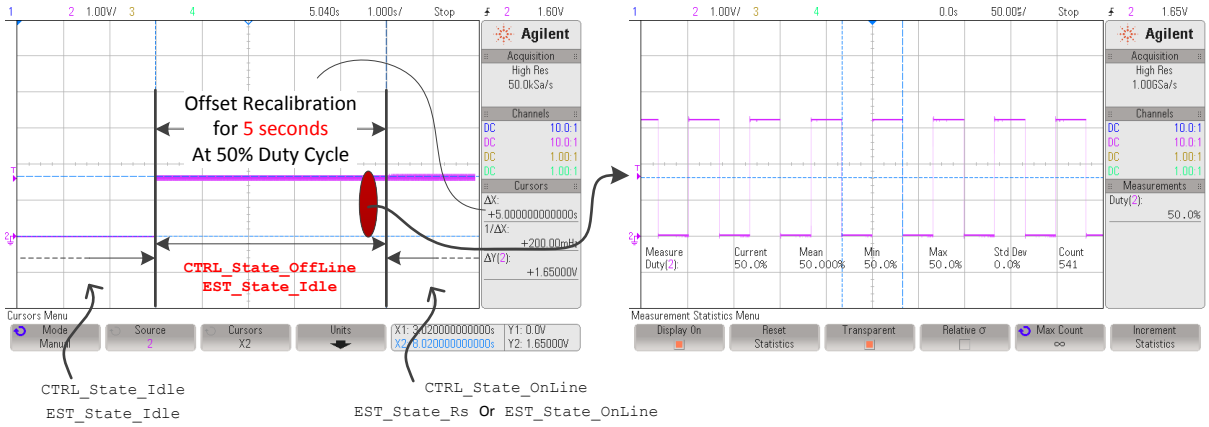


图 6-49. 偏移重校准期间的相电流

6.7.1.4 CTRL_State_OnLine 和 EST_State_Rs

完成电机识别且启用 Rs 重校准后，将退出 EST 空闲状态，进入 Rs 状态（请参见图 6-48）。 请注意，当控制器离线（电机静止）且启用了偏移重校准时，将在进入 Rs 状态前进行偏移重校准。 当估算器状态机进入 Rs 状态 (EST_State_Rs) 后，便会立即开始定子电阻重校准。

电机静止时执行的 Rs 重校准也称为离线定子电阻重校准。 Chapter 15将介绍另一种在电机旋转时执行的 Rs 重校准。

尽管 Rs 启用标志由状态机在内部进行检查，但用户也可使用以下代码示例来检查该标志：

```
if(EST_getFlag_enableRsRecalc(obj->estHandle) == TRUE)
```

尽管 Rs 重校准默认启用，但也可在启用控制器前使用以下代码示例将其启用：

```
EST_ setFlag_enableRsRecalc(obj->estHandle, TRUE);
```

以下代码示例可用于在启用控制器前禁用 Rs 重校准：

```
EST_ setFlag_enableRsRecalc(obj->estHandle, FALSE);
```

6.7.1.4.1 管理 Rs 重校准所需的时间

当估算器进入 Rs 状态 (EST_State_Rs) 后，便会立即开始定子电阻重校准。 EST_State_Rs 包括以下三个在电机完全识别期间执行的状态：

- EST_Rs_State_RampUp
- EST_Rs_State_Coarse
- EST_Rs_State_Fine

定子电阻重校准期间仅执行两个状态：EST_Rs_State_RampUp 和 EST_Rs_State_Fine，因此，重校准时间将短于电机识别期间的完全校准时间。 要计算电机重校准期间的 EST_State_Rs 执行时间，需考虑在 user.c 文件中配置的以下两个等待时间：

```
pUserParams->RsWaitTime[EST_Rs_State_RampUp] = (uint_least32_t)(1.0*USER_EST_FREQ_Hz);
pUserParams->RsWaitTime[EST_Rs_State_Fine]   = (uint_least32_t)(3.0*USER_EST_FREQ_Hz);
```

图 6-50 显示了定子电阻重校准情况。 图中显示该过程耗费的时间以及 Rs 重校准前后的状态。

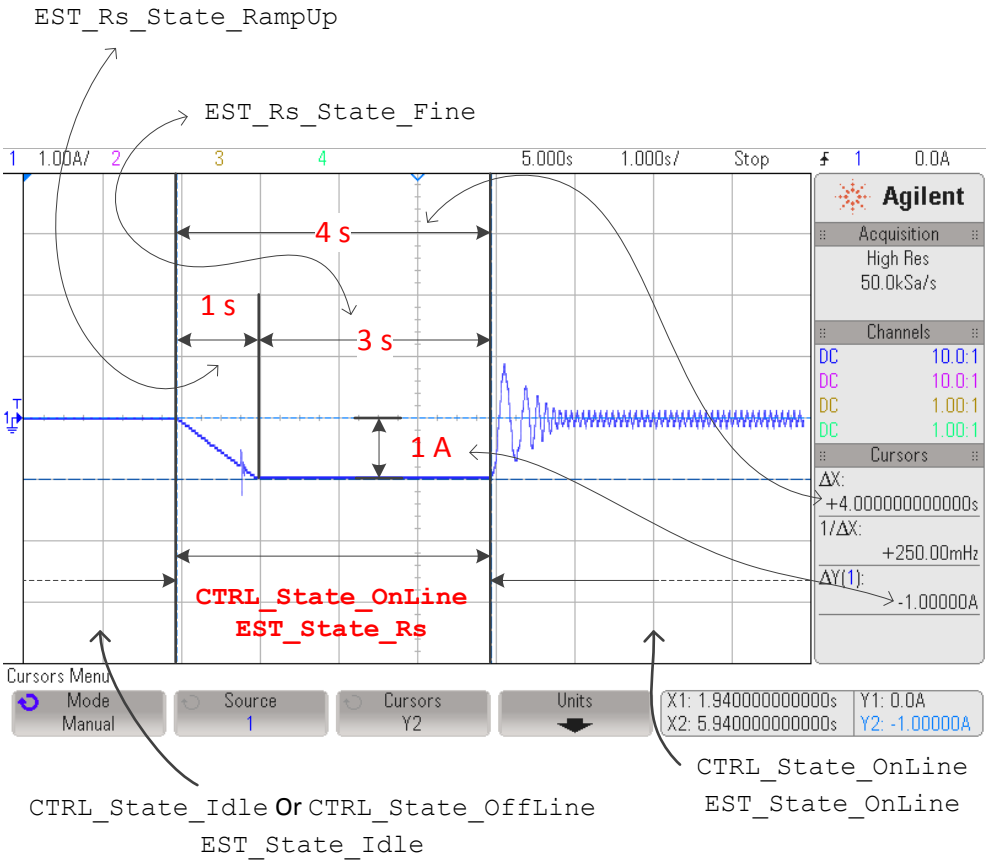


图 **6-50. Rs** 重校准期间的相电流时序图

6.7.1.4.2 Rs 重校准的软件配置

成功执行定子电阻重校准所用的配置步骤与电机识别的 Rs 测量期间所用的步骤相同，有关详细信息，请参见6.5.4 节。 Rs 重校准（电机识别后）和 Rs 校准（电机识别期间）使用相同的软件配置。

6.7.1.5 CTRL_State_OnLine 和 EST_State_OnLine

偏移重校准（如果启用）后，控制器状态设置为在线；Rs 重校准（如果启用）后，估算器状态设置为在线。 当控制器状态机和估算器状态机均处于在线状态时，电机将闭环运行，同时将使用估算角度来运行磁场定向控制 (FOC) 功能块并将估算的速度用于速度控制器。

6.7.1.5.1 从 CTRL 在线状态和 EST Rs 状态转换到在线状态

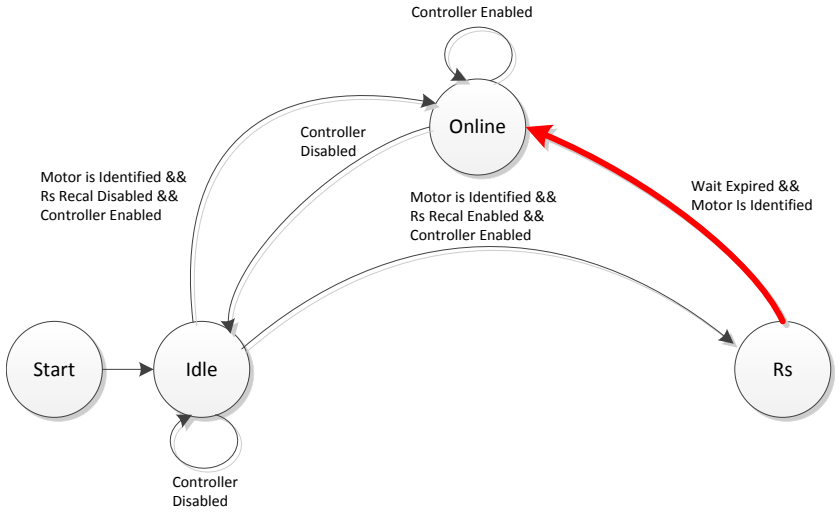


图 6-51. 从 EST Rs 状态转换到在线状态

图 6-52 显示了从 Rs 重校准状态转换到在线状态时的一相电流。

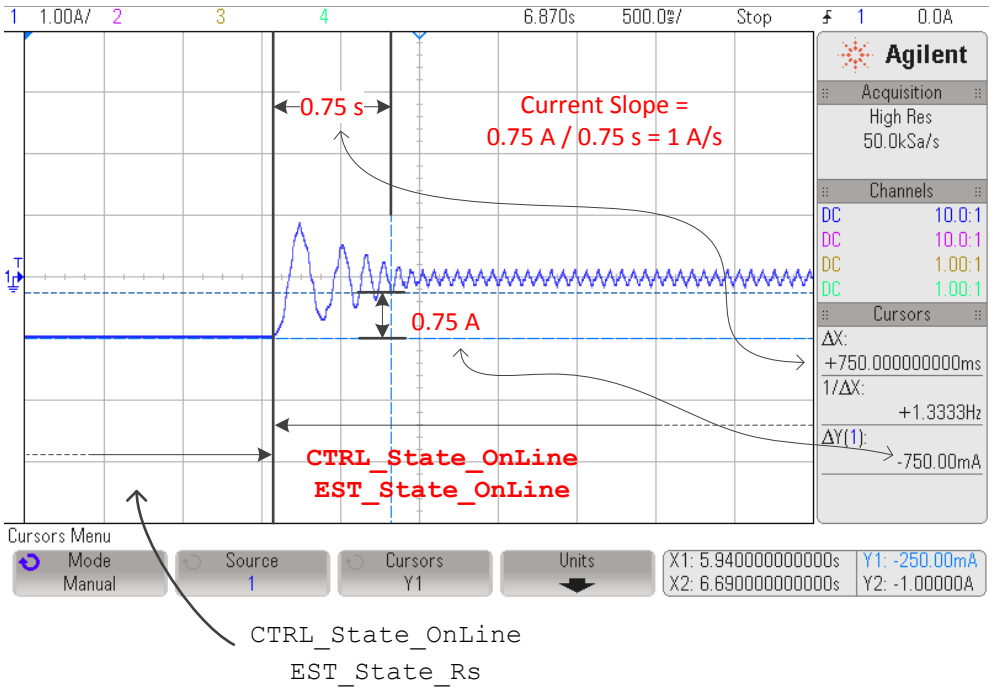


图 6-52. 从 EST Rs 状态转换到在线状态时的相电流

从图中可以看出，该相电流从 Rs 重校准的注入电流逐渐升至负载所需的电流。 图中显示此过程耗费 0.75s。

我们可以清楚地看到，在电机转轴无机械负载的情况下，电流斜率即为 Rs 重校准每秒的注入电流。 例如，如果使用 1A 电流来重校准定子电阻，控制器需耗费 1s 时间来完全移除 D 轴 (ID) 的注入电流。 电机剩余的相电流将为 IQ 电流，该电流将取决于电机的机械负载。

6.7.1.5.2 从（CTRL 空闲状态或 CTRL 离线状态）和 EST 空闲状态转换到在线状态

另一个以在线状态为目标的转换过程发生在电阻重校准禁用时。 在这种情况下，不会向电机注入之前的任何电流。 之前的状态为离线状态（如果偏移重校准已启用）或空闲状态，如图 6-53 和图 6-54 所示。

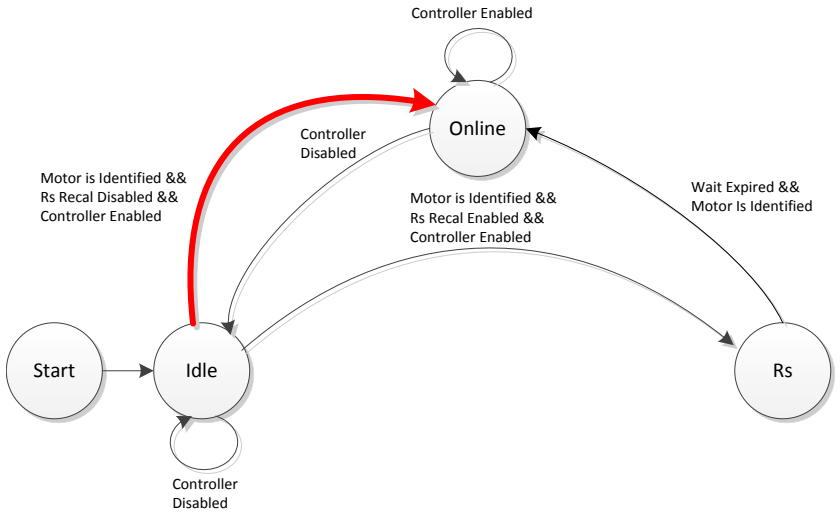


图 6-53. 从 EST 空闲状态转换到在线状态

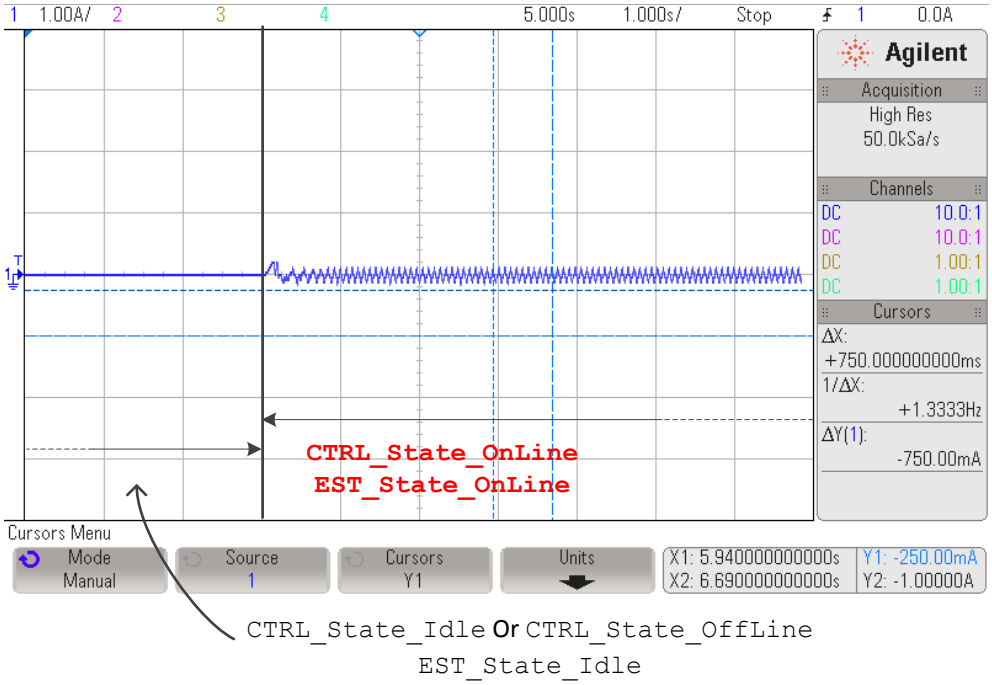


图 6-54. 从 EST 空闲状态转换到在线时的相电流

6.7.1.6 CTRL_State_Idle 和 EST_State_Idle

在通过调用以下函数禁用控制器之前，将一直保持在线状态：

```
CTRL_s etFlag_enable Ctrl(ctrlHandle, FALSE);
```

调用以上函数会禁用控制器，并将控制器状态机和估算器状态机均设为空闲状态。

图 6-55 显示了每个状态的电流和输出电压。 第一个状态是偏移重校准状态，第二个状态是 Rs 重校准状态。 第三个状态是在线状态，此时电机指令速度或转矩在闭环运行中跟随。

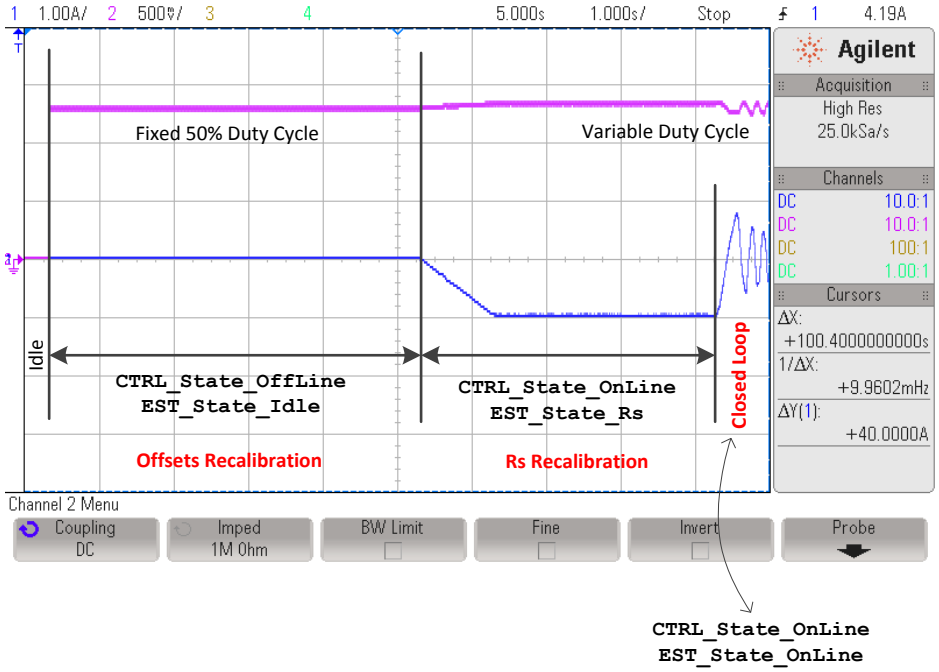


图 6-55. 完整重校准时序

6.7.2 使用 user.h 中的参数后的 PMSM 和 ACIM 电机重校准

本小节介绍 PMSM 和 ACIM 电机重校准，其中相关电机参数通过头文件提供。 6.7.1 节中使用的状态机同样适用于本小节。

启用控制器之前，代码得知以下条件为真时将通过 user.h 中的电机参数执行电机重校准：

```
if(CTRL_getFlag_enableUserMotorParams(ctrlHandle) == TRUE)
```

6.7.1 节中介绍的完全识别后重校准所对应的状态、状态转换条件和功能同样适用于通过 user.h 中的参数执行重校准的情况。

6.8 在 user.h 中设置 PMSM 电机参数

下面列出了 user.h 中为 PMSM 电机提供的参数：

```
#if (USER_MOTOR == User_PMSM)
#define USER_MOTOR_TYPE           MOTOR_Type_Pm
#define USER_MOTOR_NUM_POLE_PAIRS (4)
#define USER_MOTOR_Rs             (2.83)
#define USER_MOTOR_Ls_d           (0.0115)
#define USER_MOTOR_Ls_q           (0.0135)
#define USER_MOTOR_RATED_FLUX     (0.502)
#define USER_MOTOR_MAX_CURRENT    (4.0)
```

表 6-6 汇总了绕过 PMSM 电机识别时所需的 user.h 头文件中的全部参数。

表 6-6. user.h 中的 PMSM 电机参数

user.h 中的 PMSM 电机参数	PMSM 电机参数和单位	PMSM 电机模型符号
USER_Motor_Rs	定子电阻 (Ω)	R_s
USER_Motor_Ls_d	定子直接电感 (H)	L_{sd}
USER_Motor_Ls_q	定子正交电感 (H)	L_{sq}
USER_Motor_RATED_FLUX	额定磁通 (V/Hz)	ψ

从电机数据表中提取电机参数时，可参考以下 PMSM 模型：

$$V_{sd} = R_s i_{sd} - \omega_m L_{sq} i_{sq} + L_{sd} \frac{di_{sd}}{dt}$$
$$V_{sq} = R_s i_{sq} + \omega_m L_{sd} i_{sd} + L_{sq} \frac{di_{sq}}{dt} + \omega_m \psi \quad (16)$$

其中：

R_s：定子电阻

L_{sd}：D 轴定子电感

L_{sq}：Q 轴定子电感

ψ：转子磁通

v_{sd}：D 轴电压分量

v_{sq}：Q 轴电压分量

i_{sd}：D 轴电流分量

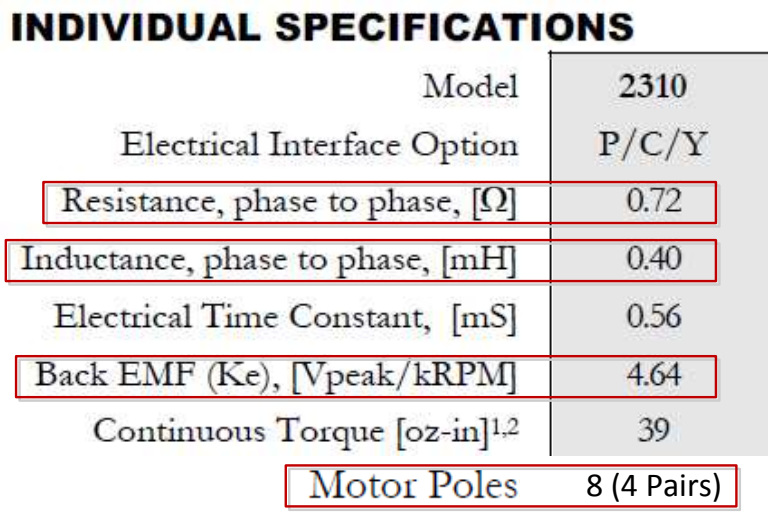
i_{sq}：Q 轴电流分量

ω_m：磁场角频率

下一节将具体介绍上述所有参数以及如何从典型电机制造商数据表中获取这些参数。

6.8.1 从 PMSM 数据表中获取参数

图 6-56 即为用作示例的 PMSM 电机数据表。 电机部件号：2310P，制造商：Teknic, Inc. (www.teknic.com)



INDIVIDUAL SPECIFICATIONS

Model	2310
Electrical Interface Option	P/C/Y
Resistance, phase to phase, [Ω]	0.72
Inductance, phase to phase, [mH]	0.40
Electrical Time Constant, [mS]	0.56
Back EMF (Ke), [Vpeak/kRPM]	4.64
Continuous Torque [oz-in][1,2]	39
Motor Poles	8 (4 Pairs)

图 **6-56.** 示例 **PMSM** 电机数据表

6.8.1.1 极对数

极对数用于计算每分钟转数 (RPM) 等速度，也可用于某些磁通计算，如额定磁通计算示例所示。 只能在 user.h 中使用电机数据表中的极对数，如下所示：

```
#define USER_MOTOR_NUM_POLE_PAIRS (4)
```

6.8.1.2 定子电阻 (Rs)

上述电机数据表中显示的定子相间电阻为 0.72Ω。 我们需要采用 user.h 文件中的 Y 形连接电机的相电阻。 在这种情况下，由于已知电机采用 Y 形配置连接，只需将相间电阻除以 2 便可转换为相电阻。 Y 形连接电机的线间电阻转换为线电阻需要执行以下运算：

$$R_s^{user.h} = R_s^{phase\,to\,phase} \times \frac{1\Omega^{phase\,to\,phase}}{2\Omega^{phase\,to\,neutral}} = 0.72\Omega \times 0.5 = 0.36\Omega \quad (17)$$

所得的值随后按如下方式写入 user.h 中：

```
#define USER_MOTOR_Rs (0.36)
```

在电机中，如果采用 Δ 形连接而非 Y 形连接，则需要将 Δ 形连接转换为 Y 形连接，以便设置电阻值。 例如，如果已知 Δ 形连接的 Rs (Δ) 值为 3Ω，则 Rs (Y) = Rs (Δ) / 3 = 1Ω。

6.8.1.3 定子电感（Ls_d 和 Ls_q）

对于非凸极 PMSM 电机而言，Ls_d 和 Ls_q 相等。 本示例中显示的相间定子电感值为 0.40mH。 我们需要将该值转换为 Y 形连接电机的相电感，具体步骤与之前一个参数采用的步骤相同，即只需除以 2（如下所示）：

$$L_{s\_d}^{user.h} = L_{s\_q}^{user.h} = L_s^{phase\,to\,phase} \times \frac{1H^{phase\,to\,phase}}{2H^{phase\,to\,neutral}} = 0.40mH \times 0.5 = 0.20mH \quad (18)$$

所得的值随后按如下方式写入 user.h 中：

```
#define USER_MOTOR_Ls_d (0.0002)
#define USER_MOTOR_Ls_q (0.0002)
```

当 Ls_d 和 Ls_q 不同时，只需根据 user.h 中的正确定义设置相应值。 在电机中，如果采用 Δ 形连接而非 Y 形连接，则需要将 Δ 形连接转换为 Y 形连接，以便设置电阻值。 例如，如果已知 Δ 形连接的 Ls_d (Δ) 值为 0.3mH，则 Ls_d (Y) = Ls_d (Δ) / 3 = 0.1mH。

6.8.1.4 额定磁通 (ψ)

需要从电机数据表中获取的最后一个参数为额定磁通。 这一特定参数可通过所提供的参数反电动势常数 Ke 计算，该常数以 Vpeak/kRPM 形式提供。 若要在头文件中设置正确的值，需要进行以下单位转换：

$$\psi_s^{user.h} = K_e^{\frac{Vpeak}{kRPM}} \times \frac{60\,s}{1\,min} \times \frac{1k\,Rev}{1000\ Rev} \times \frac{1\,Rev}{PolePairsCycles} \times \frac{1V_{line\,to\,neutral}}{\sqrt{3}\,V_{line\,to\,line}}$$

$$\psi_s^{user.h} = 4.64 \times 60 \times \frac{1}{1000} \times \frac{1}{4} \times \frac{1}{\sqrt{3}} = 0.0402\,V/Hz \quad (19)$$

所得的值随后按如下方式写入 user.h 中：

```
#define USER_MOTOR_RATED_FLUX (0.0402)
```

电机制造商数据表不提供电机额定磁通的情况也很常见，此时，可选用一种测量额定磁通的方法：在待测电机通过另一电机驱动旋转时测量相间电压。 换言之，就是使电机在发电机模式下运行并将示波器连接到电机各相，以查看给定速度下产生的电压。 为了测量磁通，电机在发电机模式下的运行速度应足够高，以克服齿槽转矩干扰。 例如，以同一电机为例，我们通过与其转轴相连的另一电机来运行该电机并绘制相间电压。 在本例中，电机运行速度大约为 1000RPM，但只要电机达到恒定速度时便可以任意速度运行。图 6-57 显示了结果。

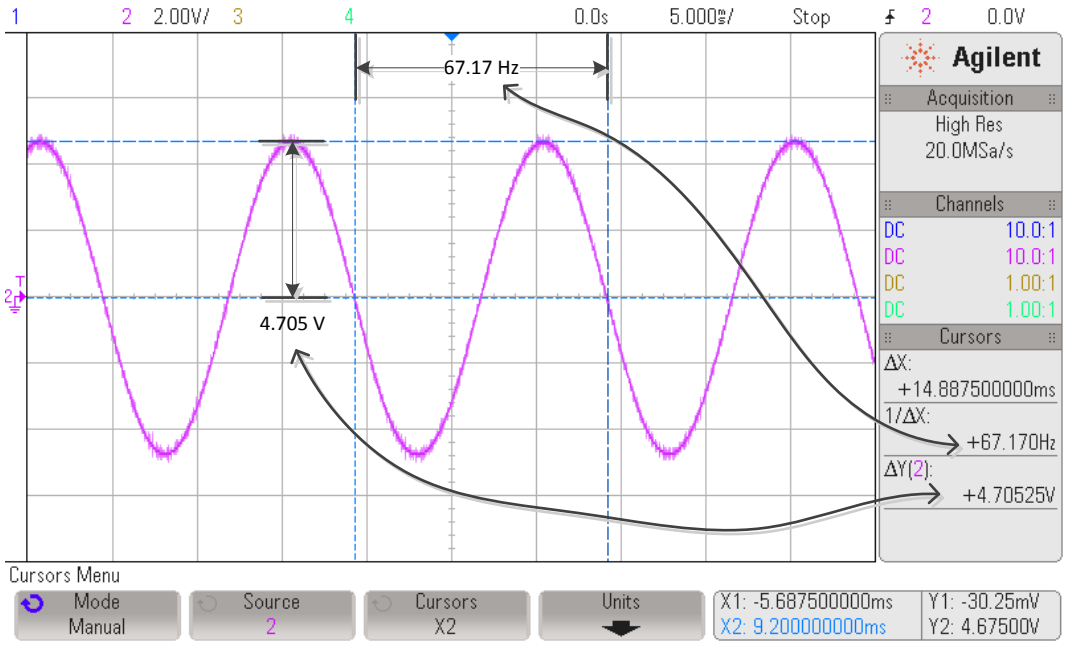


图 **6-57.** 根据发电机模式下的电机相电压确定电机磁通

以 4 对电极为例，Ke 以 Vpeak/kRPM 形式表示，通过示波器测得的 Ke 为：

$$K_e^{\frac{Vpeak}{kRPM}} = \frac{4.705\,V}{67.17\,Hz} \times \frac{1\,min}{60\,s} \times \frac{1000\ Rev}{1k\,Rev} \times \frac{PolePairsCycles}{1\,Rev} = 4.67\,Vpeak\,/\,kRPM \tag{20}$$

基于测量值计算 user.h 中的配置，如下所示：

$$\psi_s^{user.h} = 4.67 \times 60 \times \frac{1}{1000} \times \frac{1}{4} \times \frac{1}{\sqrt{3}} = 0.0404\,V\,/\,H \tag{21}$$

该值也可按如下方式通过示波器测量值直接计算：

$$\psi_s^{user.h} = \frac{4.705\,V}{67.17\,Hz} \times \frac{1}{\sqrt{3}} = 0.0404\,V\,/\,Hz \tag{22}$$

```
#define USER_MOTOR_RATED_FLUX (0.0404)
```

6.9 电机识别故障排除

6.9.1 通用清单

[下一版用户指南中提供]

6.9.2 PMSM 电机识别故障排除

6.9.2.1 在负载无法断开时识别 PMSM 电机

InstaSPIN-FOC 需要获得一些电机参数来实现最佳运行性能。 在绝大多数时间，该过程仅执行一次并且要求电机断开所有机械负载，以便能够自由旋转。 识别过程的一些步骤要求电机开环旋转。 电机在开环控制下运行时容易停转，这就是我们要求电机不连接任何机械负载的原因。 如果电机可以断开机械负载，则 InstaSPIN-FOC 识别过程不会出现问题，结束识别时，观察窗口中会显示供未来使用的电机参数。

另一方面，某些应用规定要连接机械负载。 这些应用的示例为压缩机、某些直驱式洗衣机和带密封外壳的齿轮传动电机等。 对于这些应用而言，用户需要以不同方式来运行电机识别，以便在电机不旋转或旋转最少的情况下获得电机参数。

6.9.2.2 电机能否在连接负载的情况下旋转？

第一步是确保电机没有带负载运行。 在某些情况下，开环测试实际允许电机在连接一定负载的条件下旋转。 如果电机在识别过程期间的任意时刻停转，则继续下一步骤。

6.9.2.3 运行电机识别的前三个步骤

在电机识别期间，有三个初始步骤不需要转轴旋转。 第一个步骤是计算硬件偏移。 第二个步骤是注入高频电流正弦波，以识别称之为高频电阻 (Rhf) 和高频电感 (Lhf) 的参数。 第三个步骤是通过注入直流电流来识别定子电阻 (Rs)。 在此步骤期间，请记录高频电感 (Lhf) 和定子电阻 (Rs)。 可通过库中的以下函数调用来获得此步骤所需的两个变量：

```
// get Lhf
gLhf = CTRL_getLhf(ctrlHandle);
// get the stator resistance
gRs = EST_getRs_Ohm(obj->estHandle);
```

6.9.2.4 通过 user.h 中的电机参数运行

尽管可使用多种方法在不进行电机识别的情况下计算电机磁通（见节 6.8.1.4），但该过程介绍的是完全未知的电机。 目前为止，我们有两个可以使用的参数：粗略估算的电感 (Lhf) 和精确的定子电阻 (Rs)。 还缺少一个 PMSM 电机参数：电机磁通。 在识别过程的这一步骤中，我们将使用两个已知参数和一个任意的磁通值。 请记住，一旦电机闭环运行，估算磁通将趋近于电机实际磁通，因此我们只需使第一次估算足够接近实际值。

请注意 user.h 中的特定定义。

```
#define USER_MOTOR_NUM_POLE_PAIRS (3)
```

极数仅对于从库中获取正确的 RPM 读数非常重要，而库所关心的是电频率，该参数并不受极对数的影响。 尽量准确地估算该参数。 电机运行后可以更改为正确的极数，方法是设定 60RPM 的参考速度，确保电机每秒旋转一圈。

```
#define USER_MOTOR_Rs (0.8)
```

该参数应在步骤 2 中从变量 gRs 中获得。

```
#define USER_MOTOR_Ls_d (0.01)
```

```
#define USER_MOTOR_Ls_q (0.01)
```

对于电感而言，我们需要使用在步骤 2 中获得的高频电感。该电感是电机电感的粗略近似值。 若电感与实际电感不同，则会限制高动态运行期间的性能。 例如，如果电机需要运行转矩阶跃或速度阶跃，电感不正确会引发问题，但通过高频电感代替实际电感则可在低动态运行阶段实现满转矩运行。 因此对于该步骤，请复制从 Ls_d 和 Ls_q 的 gLhf 中获得的值。

```
#define USER_MOTOR_RATED_FLUX (0.5)
```

磁通通过库进行估算，因此用户需要一个足够接近实际值的值，以便基于估算器输出值得出电机的实际磁通。 使用任意数值可能会导致估算磁通出现一定的饱和，因为该值在内部受限。 通过用以下函数查看估算磁通可轻松解决该问题：

```
// get the flux
gFlux_VpHz = EST_getFlux_VpHz(obj->estHandle);
```

如果将 gFlux_VpHz 钳位到小于 user.h 中设置的原始值的值，则将该值减半，如果钳位到大于原始值的值，则将该值加倍。 一旦电机闭环运行且 gFlux_VpHz 值发生略微变化，请记录该值并将其返回 USER_MOTOR_RATED_FLUX。

6.9.2.5 RoverL 识别期间的电流控制器稳定性故障排除

[TBD]

6.9.2.6 斜升期间的电机转轴停转故障排除

电机转轴需要在斜升过程期间始终保持运动。 这意味着，在电机转轴带有一些负载以及电机具有高齿槽转矩的情况下，用于斜升电机的电流必须增大，从而达到保持转轴运动的要求。 从电机额定电流的 10% 开始，以额定电流的 10% 为增量不断增加，直至电机转轴在整个斜升过程期间持续运动。

```
#define USER_MOTOR_RES_EST_CURRENT (1.0) //increase in 10% steps as needed
```

6.9.2.7 平滑斜坡的电机转轴故障排除

如果延长了斜升所需的时间，用户还应考虑设置一个较小的斜升加速度来实现更平滑的斜坡。 例如，对于直驱式洗衣机等高惯性负载，可通过延长斜升时间来实现更平滑的斜升过程，如图 6-58 所示。

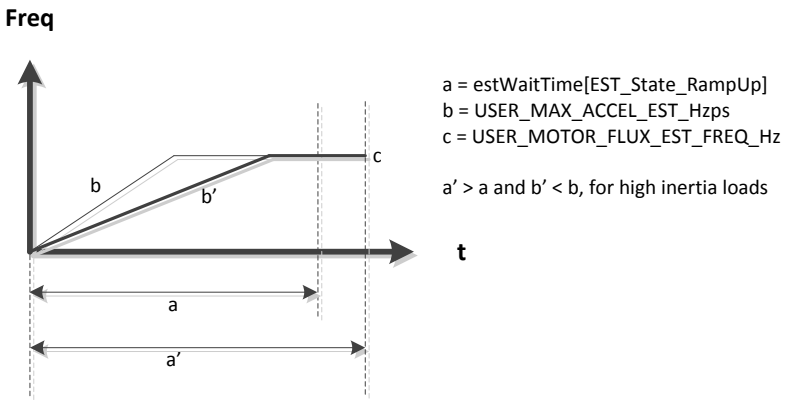


图 **6-58.** 不同加速度下的斜升时间

图 6-59 根据斜升开始期间的状态绘制，其中显示了加速度 (Hz/s) 的情况。

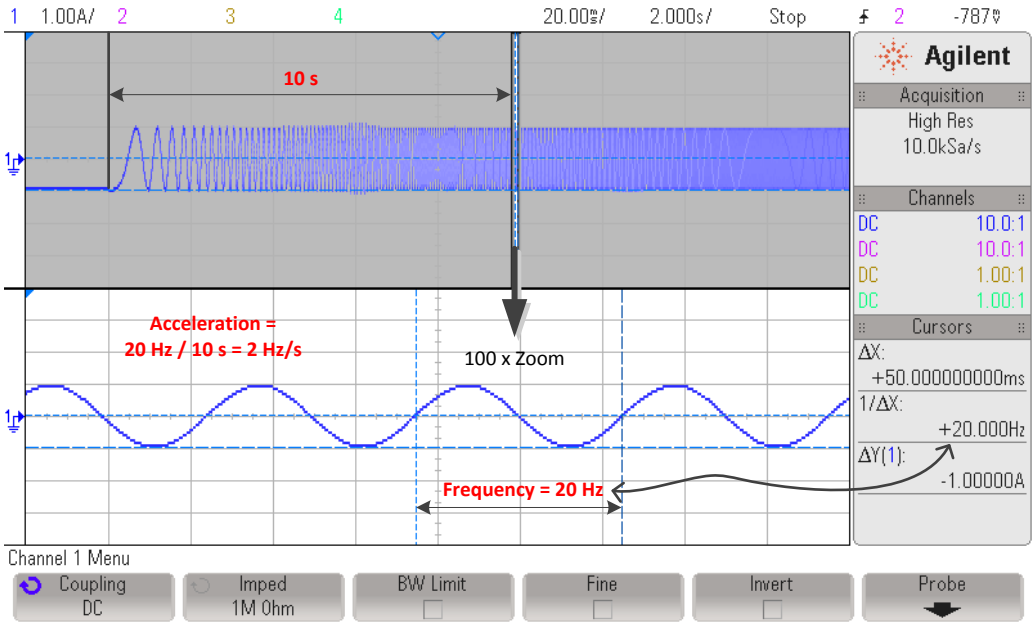


图 **6-59. PMSM** 斜升加速度

6.9.2.8 磁通测量故障排除

如果观察到以下任意情况，则所监测的磁通值将被视作不稳定：

- 饱和且不变化
- 任何负值
- 估算器状态机移至下一状态 EST_State_Ls 时，变化幅度超过 10%

如果该值不稳定，建议增加 user.c 中的额定磁通精调时间 (pUserParams->FluxWaitTime[*EST_Flux_State_Fine*])。

如果电机在磁通识别期间的任意时刻停止旋转，则暂停识别过程，更改 user.h 中设置的以下值后以更高的频率重试：

```
// During Motor ID, maximum commanded speed in Hz
#define USER_MOTOR_FLUX_EST_FREQ_Hz (20.0) (40.0)
```

默认频率设置为 20Hz，该频率适用于大多数 PMSM 电机。 以 10Hz 的增量增大频率，直至电机在磁通识别期间不会停止旋转。

6.9.2.9 Ls 识别故障排除

在整个电感识别过程中，确保电机转动非常重要。 如果电机在电感识别期间的任意时刻停止旋转，则增大磁通和电感估算所使用的频率，例如，如果原来使用 20Hz，则尝试使用 40Hz。 同样，已知电机具有较低电感时，必须增大磁通估算频率。 这种情况下，在使用几 µH 电机时会考虑低电感。 磁通识别和电感识别均使用该频率。

```
// During Motor ID, maximum commanded speed in Hz
#define USER_MOTOR_FLUX_EST_FREQ_Hz (20.0) (40.0)
```

如果用户要求缩短识别时间，则可通过监视电感来获知电感识别何时稳定。 通常，当观察窗口中的值变化幅度不超过 5% 时，即表示所识别电感处于稳定状态。 利用以下代码示例，用户可测量从 EST_State_Ls 状态开始时到电感识别过程显示稳定数值所经过的时间：

```
// get the stator inductance in the direct coordinate direction
gMotorVars.gLsd_H = EST_getLs_d_H(obj->estHandle);

// get the stator inductance in the quadrature coordinate direction
gMotorVars.gLsq_H = EST_getLs_q_H(obj->estHandle);
```

已知 Ls 达到稳定所需的时间后，即将该时间输入到以下时间中，以便缩短整个电机识别过程：pUserParams->LsWaitTime[*EST_Ls_State_Fine*]。

6.9.2.9.1 识别低电感 PMSM 电机

对于低电感 PMSM 电机而言，需要考虑多个注意事项。 一些注意事项涉及电机识别所使用的硬件，另一些注意事项则涉及软件配置。

6.9.2.9.1.1 硬件注意事项

识别低电感 PMSM 电机时，建议使用分压器来尽可能降低反馈电压。 例如，如果电机运行电压为 24V，则电阻分压器的电压应为 26V 左右。 这样在测量反馈电压时，ADC 转换器便可提供最大位数。

一旦硬件发生更改，请在 user.h 中更新 USER_ADC_FULL_SCALE_VOLTAGE_V 定义。

通常，低电感电机为高速电机，因此磁通很小。 用户需要更新 USER_IQ_FULL_SCALE_VOLTAGE_V 值，使识别过程能够识别电机磁通。 以下公式可用于确定 USER_IQ_FULL_SCALE_VOLTAGE_V 定义的值：

可以识别的最小磁通 (V/Hz) = USER_IQ_FULL_SCALE_VOLTAGE_V/ USER_EST_FREQ_Hz/0.7 (23)

6.9.2.9.1.2 软件注意事项

针对所涉及的软件配置，建议采用更高的频率来识别 R/L 常量。 对于经过测试的大多数低电感电机，300Hz 频率足以用于识别 R/L。 R/L 频率更新需要更新到以下定义中：

```
#define USER_R_OVER_L_EST_FREQ_Hz (300)
```

第二个注意事项是采用更高的频率来识别电机电感。 电机电感属于电机参数。 以下示例用于识别电感为几十 μH 的电机。

```
#define USER_MOTOR_FLUX_EST_FREQ_Hz (60.0)
```

第三个注意事项是调用函数来覆盖电流控制环路上的限值。 该函数需要在 ISR 外部调用。 该函数的名称为：CTRL_recalcKpKi()。

第四个注意事项是调用函数来基于 R/L 信息计算特定电机的初始估算电感。 该函数也需要在 ISR 外部调用，函数名称为：CTRL_calcMax_Ls_qFmt()。

第五个注意事项是调用函数来从 CTRL_calcMax_Ls_qFmt() 函数调用中获取计算得出的电感，并在执行电感识别时初始化估算电感。 新的函数调用需要在 ISR 结束时进行，函数名称为：CTRL_resetLs_qFmt()。

6.9.2.9.2 识别凸极 PMSM 电机的电感

对于低电感 PMSM 电机而言，需要考虑多个注意事项。 一些注意事项涉及电机识别所使用的硬件，另一些注意事项则涉及软件配置。

6.9.2.9.2.1 硬件注意事项

识别低电感 PMSM 电机时，建议使用分压器来尽可能降低反馈电压。 例如，如果电机运行电压为 24V，则电阻分压器的电压应为 26V 左右。 这样在测量反馈电压时，ADC 转换器便可提供最大位数。

一旦硬件发生更改，请在 user.h 中更新 USER_ADC_FULL_SCALE_VOLTAGE_V 定义。

通常，低电感电机为高速电机，因此磁通很小。 用户需要更新 USER_IQ_FULL_SCALE_VOLTAGE_V 值，使识别过程能够识别电机磁通。 以下公式可用于确定 USER_IQ_FULL_SCALE_VOLTAGE_V 定义的值：

可以识别的最小磁通 (V/Hz) = USER_IQ_FULL_SCALE_VOLTAGE_V/ USER_EST_FREQ_Hz/0.7

6.9.2.9.2.2 软件注意事项

针对所涉及的软件配置，建议采用更高的频率来识别 R/L 常量。 对于经过测试的大多数低电感电机，300Hz 频率足以用于识别 R/L。 R/L 频率更新需要更新到以下定义中：

```
#define USER_R_OVER_L_EST_FREQ_Hz (300)
```

第二个注意事项是采用更高的频率来识别电机电感。 电机电感属于电机参数。 以下示例用于识别电感为几十 μH 的电机。

```
#define USER_MOTOR_FLUX_EST_FREQ_Hz     (60.0)
```

第三个注意事项是调用函数来覆盖电流控制环路上的限值。 该函数需要在 ISR 外部调用。 该函数的名称为：CTRL_recalcKpKi()。

第四个注意事项是调用函数来基于 R/L 信息计算特定电机的初始估算电感。 该函数也需要在 ISR 外部调用，函数名称为：CTRL_calcMax_Ls_qFmt()。

第五个注意事项是调用函数来从 CTRL_calcMax_Ls_qFmt() 函数调用中获取计算得出的电感，并在执行电感识别时初始化估算电感。 新的函数调用需要在 ISR 结束时进行，函数名称为：CTRL_resetLs_qFmt()。

6.9.2.9.3 识别凸极 PMSM 电机的电感

[TBD]

6.9.2.10 识别高速 PMSM 电机

调节高速电机产生的电流控制器增益。 [TBD]

6.9.2.11 识别高齿槽转矩 PMSM 电机

有关该主题的讨论，请参见节 6.9.2.6。

6.9.3 ACIM 电机识别故障排除

6.9.3.1 识别低压高电流 ACIM 电机

[TBD]

6.9.3.2 识别低电感 ACIM 电机

[TBD]

6.9.3.3 磁通测量故障排除

两种电机的磁通测量故障排除方法相同。 有关详细信息，请参见6.5.3 节。 默认频率设置为 20Hz，该频率也适用于大多数 ACIM 电机。 对于 PMSM 电机，以 10Hz 的增量增大频率，直至电机在磁通识别期间不会停止旋转。

6.9.3.4 锁定转子测试故障排除

务必要完全锁定转子，避免转轴运动。 这是确保内部变量在估算器状态机的后续状态产生精确电机参数的唯一方法。

惯性识别

在经典力学中，惯性矩也称为质量惯性矩或转动惯量，是物体绕轴旋转加速的阻力。 通常，此值为电机力矩与电机质量刚性耦合加速度之间的比值。

常见的误区是认为惯性等效于负载。 负载通常以负载惯性和负载转矩的形式表示，这里的负载惯性是指同电机转子一同旋转的质量，而负载转矩则是施加到电机转子转轴上的外部转矩。 区分这两者的简单方法是考虑当转子转轴改变旋转方向时，负载是否会与转子转轴一同旋转。 直接耦合器和负载转轴上安装的质量刚性皮带轮即为负载惯性示例。 负载惯性和电机转子惯性组成系统惯性。 负载转矩示例包括：施加到电机转子转轴一端的质量重力、洗衣机滚筒转动时分散的衣物以及水泵的流体粘度。 估算系统负载惯性时应尽量减少或消除负载转矩。

图 7-1 显示了一个简单的运动系统示例。 在此系统中，旋转质量与电机刚性耦合。 这意味着旋转质量与电机一同旋转，并且被视为惯性的一部分。 非旋转质量没有与电机刚性耦合，被视为负载的一部分。 在惯性识别过程期间，非旋转质量不应连接到电机上。

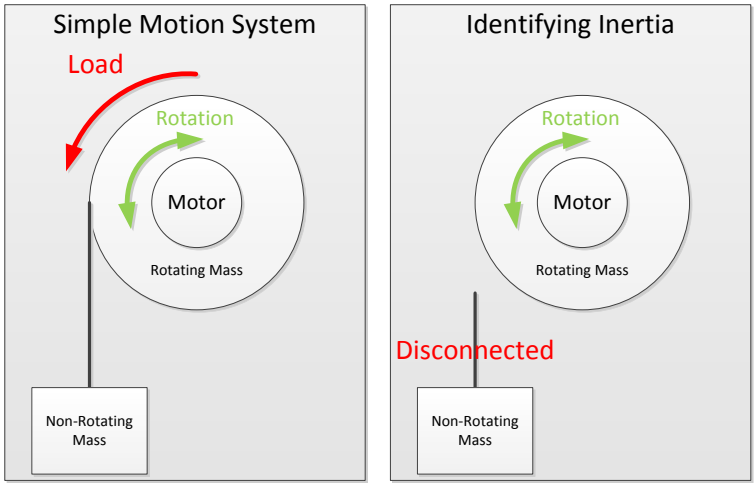


图 **7-1.** 简单运动系统中的惯性识别示例

Topic **Page**

7.1 InstaSPIN-MOTION 惯性识别

惯性是精确控制机械系统时所需的一项重要信息。 InstaSPIN-MOTION 通过 SpinTAC 速度识别实现准确的惯性估算，同时考虑无传感器应用中的摩擦力影响，从而提供稳健的惯性识别特性。 目前，SpinTAC 速度识别并不主动考虑负载转矩。 为了得到正确的值，需要尽量减少或消除负载转矩，例如起重机应用重力或者压缩机中的压缩液体。

SpinTAC 速度识别用单位 A/[krpm/s] 来估算惯性。 这不同于传统的惯性单位 Kg * m^2。 单位 A/[krpm/s] 表示加速系统所需的转矩量。 它与国际单位 Kg * m^2 成比例。 具体关系取决于电机可生成的转矩量。 SpinTAC 控制器需要了解加速系统所需的转矩量并使用该非传统单位来表示惯性。

SpinTAC 速度识别产生了一个非常准确的惯性结果。图 7-2 为同一电机 100 次惯性识别过程测试结果图。 如图所示，惯性识别的重复性极高。 这些试验的最大值和最小值均在 100 次测试的平均值的 0.5% 范围内。 SpinTAC 速度识别正确估算系统惯性后，会立即产生一个包含重复性近似等级的结果。

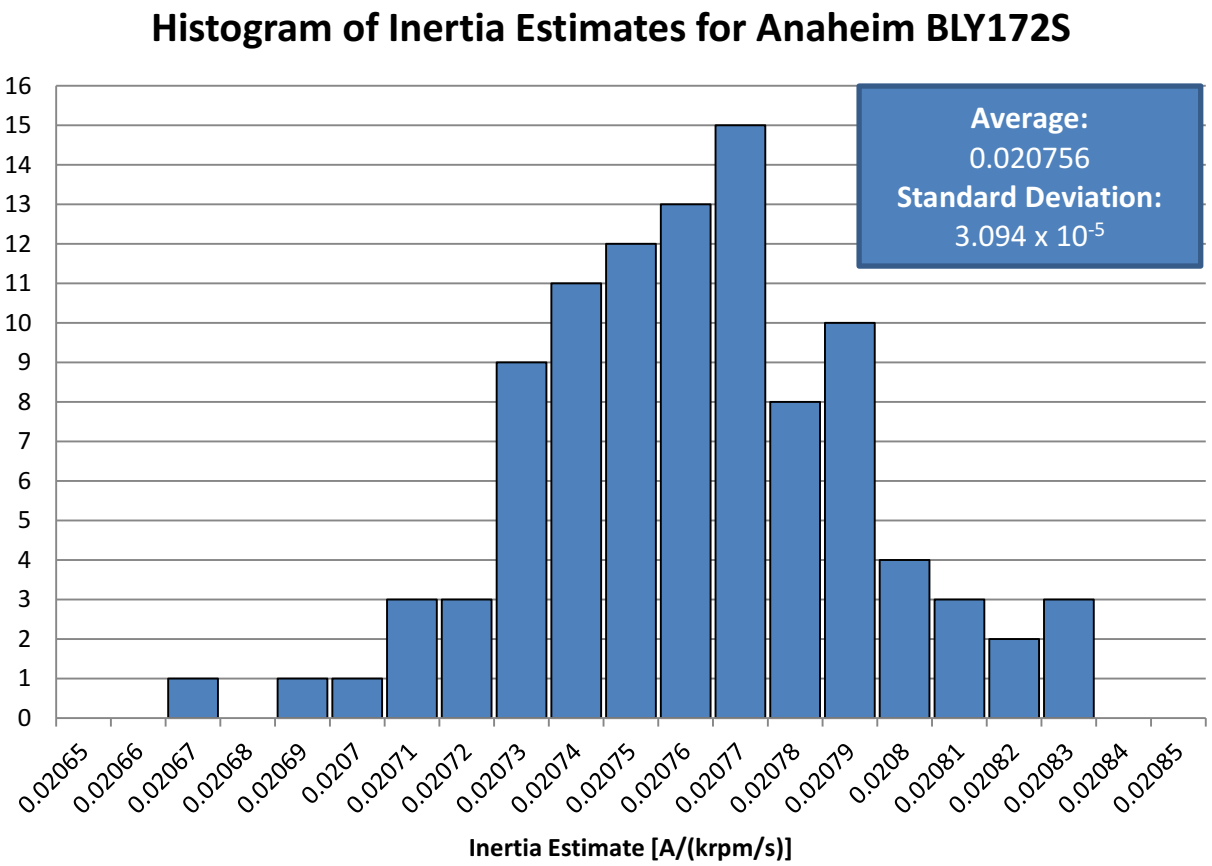


图 **7-2. 100** 次惯性识别试验柱状图

估算的惯性作为 SpinTAC 速度和位置控制器的输入。 不过，SpinTAC 控制器相当稳健，可以承受大范围惯性变动。 此特性对于系统惯性随时间变化的应用非常有用。

图 7-3 比较了 SpinTAC 速度控制器使用多种错误惯性设置时的性能。 这些数据是通过将转矩干扰施加到电机系统而测得的。 提供给 SpinTAC 速度控制器的惯性值被设置为不同的值，以便突出控制器能够承受的惯性误差范围。 从图中可以看出，该 SpinTAC 速度控制器能够承受的惯性不匹配误差高达八倍。 当惯性值与应用匹配时可实现最佳性能，但如果系统惯性发生变化，SpinTAC 速度控制器将保持稳定。

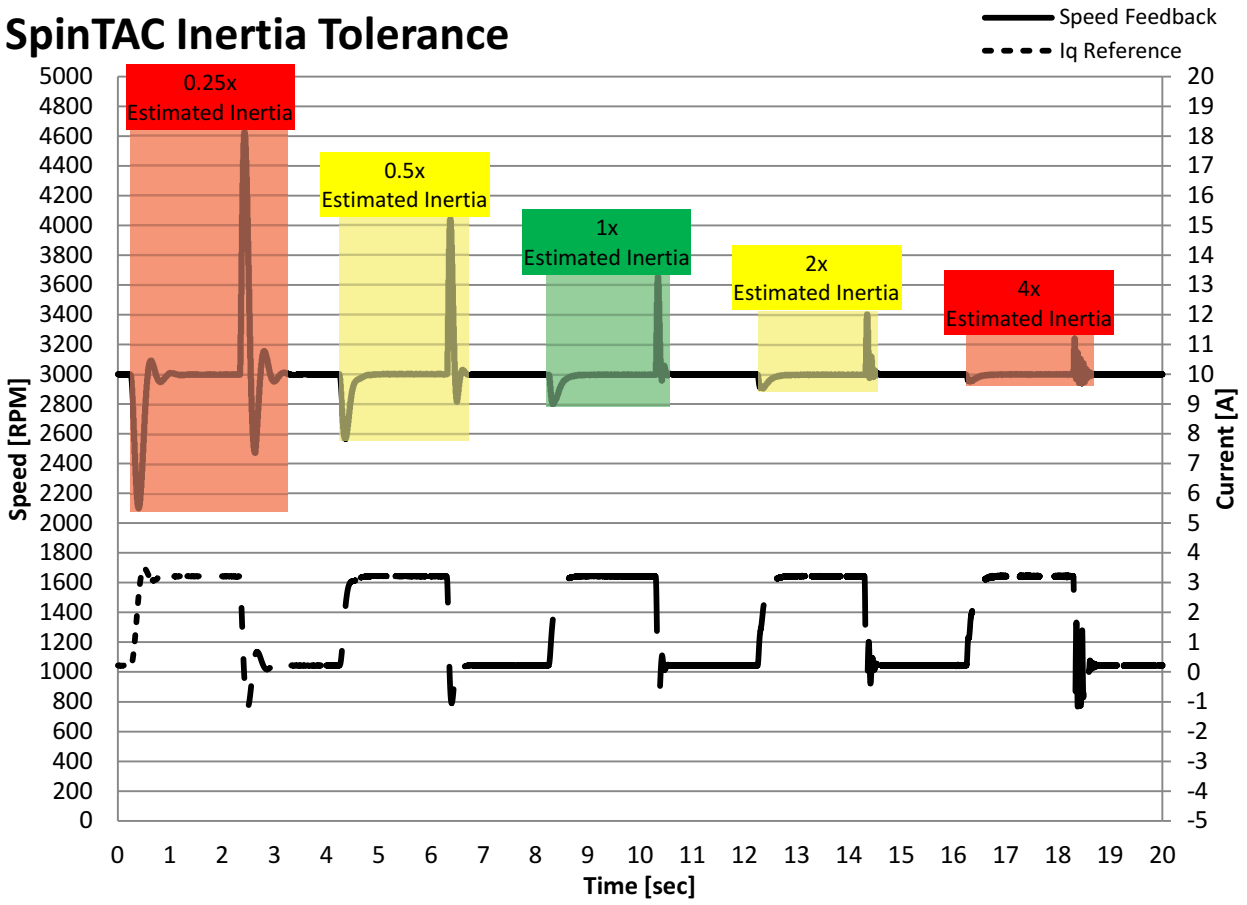


图 7-3. SpinTAC 速度控制器惯性容差

SpinTAC 速度识别提供了一种估算系统惯性的便捷方法。 该方法将连续转矩系统配置应用到电机并使用速度反馈来计算电机惯性。 这是一个开环测试，设计作为开发过程的一部分来运行。 识别惯性后，可立即将其设为默认值，并且在系统发生改变前不需要再次进行估算。

惯性识别期间，电机正向旋转，然后短暂地反向旋转。 如果系统中无法实现这一点，则需要考虑一些特殊注意事项。 关于这些注意事项说明，请参见7.5 节。

7.2 惯性识别过程概述

SpinTAC 惯性识别过程非常快。 它需要通过加速和减速电机来估算系统惯性。 在惯性识别过程之前，需要满足一些条件。

- 电机不应旋转，或者不应缓慢旋转。

  如果在电机已经转动时开始执行转矩配置，惯性的估算结果会不准确。

- InstaSPIN-FOC PI 速度控制器必须禁用。

  为测试惯性，SpinTAC 速度识别需要提供 Iq 参考值。 这只有在 InstaSPIN-FOC PI 速度控制器禁用时才能实现。

- 正向速度基准值必须在 FAST 中设置。

  FAST 估算器需要通过速度基准获得电机旋转方向，从而正确地估算速度。 该值可以是速度基准设置的任意正值。

- 强制角必须启用。

强制角可实现从零速度成功启动，并产生更准确的惯性估算结果。

图 7-4 是启用 SpinTAC 速度识别之前所需步骤的流程图。

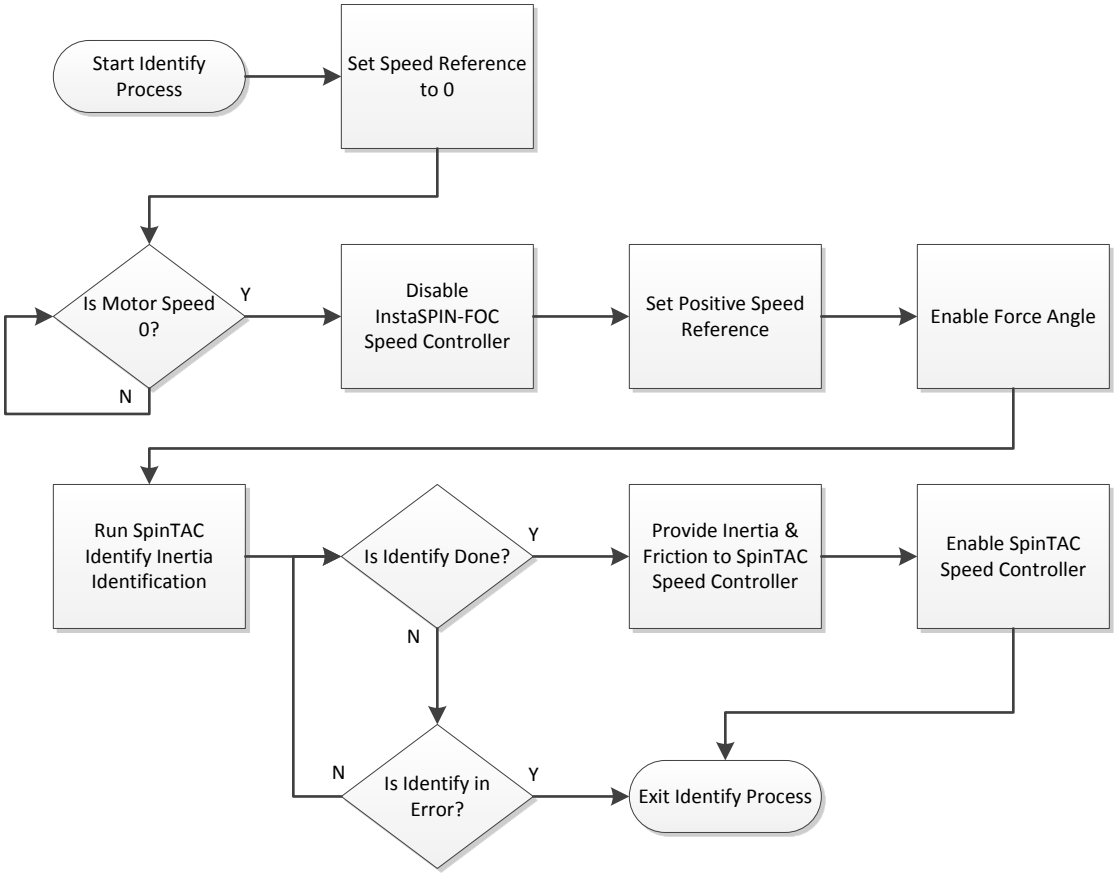


图 7-4. SpinTAC 速度识别过程流程图

图 7-5 为电机连续转矩曲线图。 SpinTAC 速度识别过程同时应用了正向转矩和反向转矩。 转矩最初施加到电机的目的是使电机转子在惯性识别过程之前正确对准。

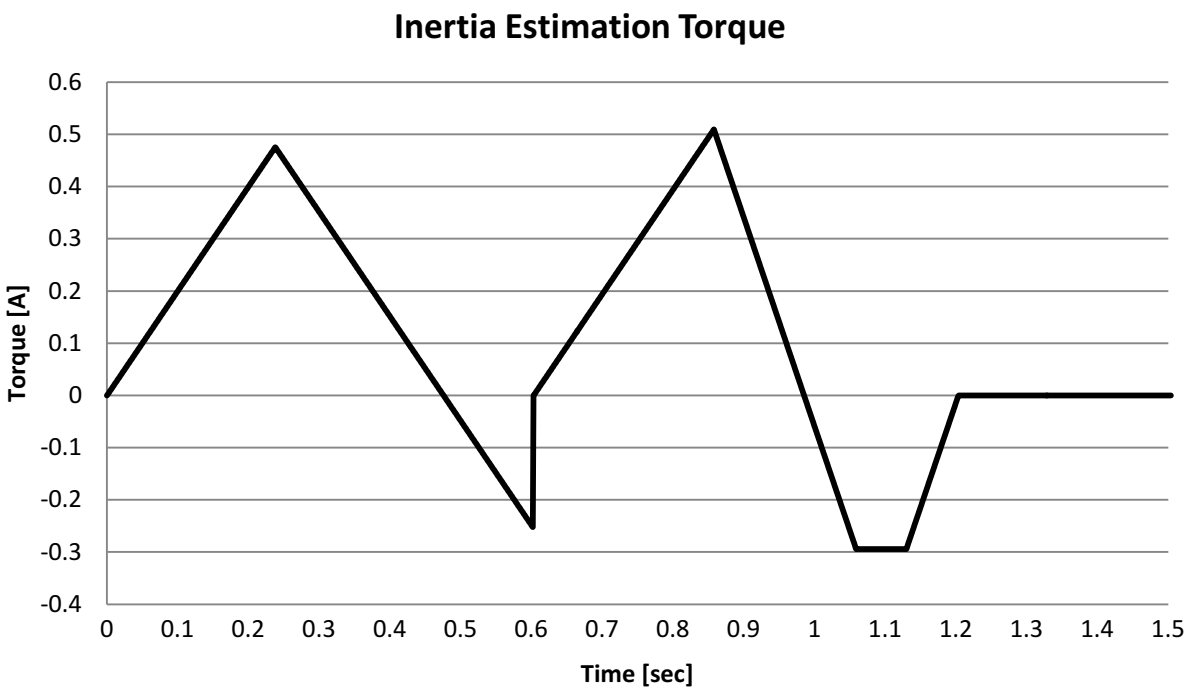


图 7-5. SpinTAC 速度识别转矩基准

电机转动的相关结果在图 14-35 中说明。 在惯性识别过程中，确保电机持续转动非常重要。 如果电机在惯性识别测试过程中停止，应调整配置参数后重新进行惯性识别过程。 有关如何修改惯性识别过程中通用配置错误的更多信息，请参见7.4 节。

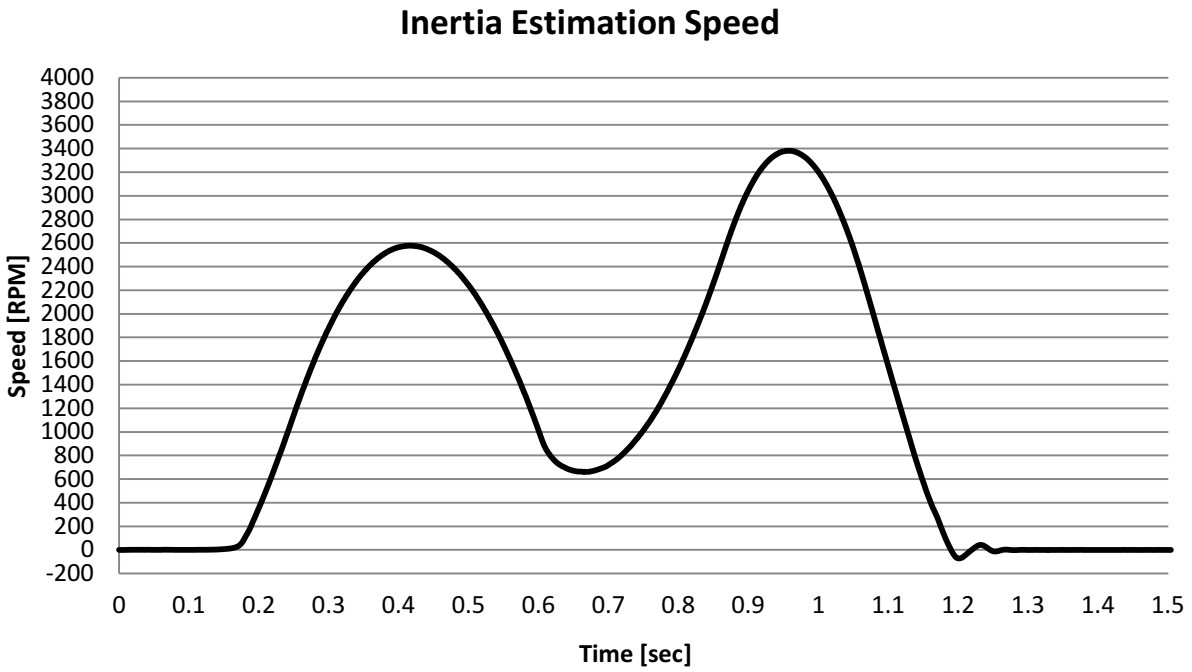


图 7-6. SpinTAC 速度识别速度反馈

这表示典型情况。 有关如何为齿槽力或摩擦力较大的电机识别系统惯性的更多信息，请参见7.5 节。

7.3 SpinTAC 速度识别的软件配置

配置 SpinTAC 速度识别需要四个步骤。 实验 5c — InstaSPIN-MOTION 惯性识别 — 本示例项目执行了使用 SpinTAC 速度识别来估算系统惯性所需的各个步骤。 利用包含在 MotorWare 中的头文件 spintac_velocity.h，您可以快速将 SpinTAC 组件包括在项目中。

7.3.1 包括头文件

这应该通过其余的模块头文件包含来完成。 在实验 5c 示例项目中，该文件包括在 spintac_velocity.h 头文件中。 针对相应项目，可通过包括 spintac_velocity.h 完成此步骤。

```
#include "sw/modules/spintac/src/32b/spintac_vel_id.h"
```

7.3.2 声明全局变量

这应该通过主源文件中的全局变量声明来完成。 在实验 5c 项目中，此结构包括在已声明为 spintac_velocity.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj st_obj;      // The SpinTAC Object
ST_Handle stHandle; // The SpinTAC    Handle
```

如果不希望使用已在 spintac_velocity.h 头文件中声明的 ST_Obj 结构，可使用以下示例。

```
ST_VelId_t stVelId;            // The SpinTAC Velocity Identify object
ST_VELID_Handle stVelIdHandle; // The SpinTAC Inertia Identify handle
```

7.3.3 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这样会将所有的默认值加载到 SpinTAC 速度识别中。 该步骤可通过运行已在 spintac_velocity.h 头文件中声明的函数 ST_init 和 ST_setupVelId 来完成。 如果不希望使用这两个函数，可使用下方的代码示例配置 SpinTAC 速度识别组件。 SpinTAC 速度识别配置是典型配置的代表，适用于大部分电机。

```
// Initialize the SpinTAC Velocity Identify Component
stVelIdHandle = STVELID_init(&stVelId, sizeof(stVelId));

// Setup SpinTAC Velocity Identify Component
// Sample time [s]
STVELID_setSampleTime_sec(stVelIdHandle, _IQ(ST_SPEED_SAMPLE_TIME));
// System speed limit [pu/s], (0, 1]
STVELID_setVelocityMax(stVelIdHandle, _IQ(1.0));
// System maximum (0,1] & minimum [-1,0) control signal [PU]
STVELID_setOutputMaximums(stVelIdHandle, maxCurrent_PU, -maxCurrent_PU);
// Goal Speed of the inertia identification process [pu/s], (0, 1]
STVELID_setVelocityPositive(stVelIdHandle, _IQmpy(_IQ(0.4), _IQ(1.0)));
// System control signal high (0, OutMax] & low [OutMin, 0) limit [PU]
STVELID_setOutputLimits(stVelIdHandle, maxCurrent_PU, -maxCurrent_PU);
// Low pass filter constant to smooth the speed feedback signal [tick], [1, 100]
STVELID_setLowPassFilterTime_tick(stVelIdHandle, 1);
// Configure the time out for inertia identification process [s], [100*T, 10.0]
STVELID_setTimeOut_sec(stVelIdHandle, _IQ(10.0));// Rate at which torque is applied to the
motor [s], [T, 25.0]
```

```
STVELID_setTorqueRampTime_sec(stVelIdHandle, _IQ(5.0));
// Initially ST_VelId is not in reset
STVELID_setReset(stVelIdHandle, false);
// Initially ST_VelId is not enabled
STVELID_setEnable(stVelIdHandle, false);
```

7.3.4 调用 SpinTAC 速度识别

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此功能。 抽取率由 spintac_velocity.h 头文件中声明的 ST_ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。 调用 SpinTAC 速度识别函数之前，必须更新速度反馈。 另外需要注意，本示例按照图 7-4 的流程图执行步骤，确保系统已准备好进行惯性识别。

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle; // Get pointer to CTRL object
_iq speedFeedback = EST_getFm_pu(obj->estHandle); // Get the mechanical speed in pu/s
_iq iqReference = 0;
if(gMotorVars.SpinTAC.VelIdRun != false) {
  // if beginning the SpinTAC Velocity Identify process
  // set the speed reference to zero
  gMotorVars.SpeedRef_krpm = 0;
  // wait until the actual speed is zero
  if((_IQabs(speedFeedback) < _IQ(ST_MIN_ID_SPEED_PU))
     && (STVELID_getEnable(stVelIdHandle) == false)) {
     gMotorVars.Flag_enableForceAngle = true;
     EST_setFlag_enableForceAngle(obj->estHandle, gMotorVars.Flag_enableForceAngle);
     // set the GoalSpeed
     STVELID_setVelocityPositive(stVelIdHandle, gMotorVars.VelIdGoalSpeed);
     // set the Torque Ramp Time
     STVELID_setTorqueRampTime_sec(stVelIdHandle, gMotorVars.VelIdTorqueRampTime);
     // Enable SpinTAC Velocity Identify
     STVELID_setEnable(stVelIdHandle, true);
     // Set a positive speed reference to FAST to provide direction information
     gMotorVars.SpeedRef_krpm = _IQ(0.001);
     CTRL_setSpd_ref_krpm(ctrlHandle, gMotorVars.SpeedRef_krpm);
  }
}

// Run SpinTAC Velocity Identify
STVELID_setVelocityFeedback(stVelIdHandle, speedFeedback);
STVELID_run(stVelIdHandle);

if(STVELID_getDone(stVelIdHandle) != false) {
  // If inertia identification is successful
  // update the inertia setting of SpinTAC Velocity Controller
  // EXAMPLE:
  // STVELCTL_setInertia(stVelCtlHandle, STVELID_getInertiaEstimate(stVelIdHandle));
  gMotorVars.VelIdRun = false;
  // return the speed reference to zero
  gMotorVars.SpeedRef_krpm = _IQ(0.0);
  CTRL_setSpd_ref_krpm(ctrlHandle, gMotorVars.SpeedRef_krpm);
}
else if((STVELID_getErrorID(stVelIdHandle) != false)
        && (STVELID_getErrorID(stVelIdHandle) != ST_ID_INCOMPLETE_ERROR)) {
  // if not done & in error, wait until speed is less than 1RPM to exit
  if(_IQabs(speedFeedback) < _IQ(ST_MIN_ID_SPEED_PU)) {
    gMotorVars.VelIdRun = false;
    // return the speed reference to zero
    gMotorVars.SpeedRef_krpm = _IQ(0.0);
    CTRL_setSpd_ref_krpm(ctrlHandle, gMotorVars.SpeedRef_krpm);
  }
}
```

```
// Set the Iq reference that came out of SpinTAC Identify
iqReference = STVELID_getTorqueReference(stVelIdHandle);
CTRL_setIq_ref_pu(ctrlHandle, iqReference);
```

7.4 惯性识别故障排除

SpinTAC 速度识别已在多种电机上进行测试。 非典型电机可能会在执行惯性识别过程时遇到困难，从而导致错误。 此错误由 SpinTAC 速度识别全局结构的 ERR_ID 字段中的值表示。 下文将讨论常见错误及其纠正方法。

7.4.1 ERR_ID

ERR_ID 可向用户提供错误代码。 表 7-1 中列出了 SpinTAC 速度识别中定义的错误及相应解决方法。

表 7-1. SpinTAC 速度识别错误代码

ERR_ID	问题	解决方法
1	采样时间值无效	将 cfg.T_sec 设置为 (0, 0.01] 范围内
2	系统最大速度值无效	将 cfg.VelMax 设置为 (0, 1] 范围内
4	速度环路控制信号最大值无效	将 cfg.OutMax 设置为 (0, 1] 范围内
5	速度环路控制信号最小值无效	将 cfg.OutMin 设置为 [-1, 0) 范围内
22	速度值无效	将 cfg.VelPos 设置为 (0, cfg.VelMax] 范围内
23	速度环路控制信号正值无效	将 cfg.OutPos 设置为 (0, cfg.OutMax] 范围内
24	速度环路控制信号负值无效	将 cfg.OutNeg 设置为 [cfg.OutMin, 0) 范围内
34	加速度斜坡时间值无效	将 cfg.RampTime_sec 设置为 [cfg.T_sec, 25] 范围内
36	反馈类型值无效	将 cfg.Sensorless 设置为 {false, true} 范围内
1010	速度反馈低通滤波时间常量无效	将 cfg.LptTime_tick 设置为 [1, 100] 范围内
1011	超时值无效	将 cfg.TimeOut_sec 设置为 [1, 10] 范围内
2003	惯性估算值无效	调整配置参数并且重复
2004	惯性识别过程超时	调整配置参数并且重复
2005	通过设置 RES = 1 或 ENB = 0 丢弃识别过程	无操作
2006	识别过程中电机停止	调整配置参数并且重复
4001	SpinTAC 许可证无效	使用具有有效许可证的芯片
4003	ROM 版本无效	使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。

7.4.2 2003 错误

此错误表示惯性估算值不正确。 这通常是由摩擦力或齿槽力较大的电机引起的。 要修正此错误，需减小全局结构配置部分的 RampTime_sec 参数。 减小此参数会增大惯性识别过程中施加到电机的转矩率。

如果识别出的惯性是有效值，并且摩擦系数为很小的负值，则此错误的原因可能是摩擦系数非常小导致计算精度不良。 在这种情况下，识别的惯性仍可能有效。

7.4.3 2004 错误

此错误表示惯性识别过程完成超时。 出现这种情况的原因有多种。

7.4.3.1 电机持续转动

如果电机持续转动并且惯性识别过程产生此错误，这表示识别过程的目标速度过高。 这通常是由额定速度较低的电机引起的。 修正此错误的方法是减小全局结构配置部分的 VelPos 参数。 此参数表示识别过程的目标速度。 如果 VelPos 值设置过低，则可能导致惯性识别不准确。

7.4.3.2 电机初始未转动

如果电机初始未转动且惯性识别过程产生此错误，这表示施加到电机上的转矩过低。 这通常是所需启动电流很大的电机引起的。 修正此错误的方法是增大全局结构配置部分的 OutPos 参数。 此参数表示识别过程中施加的最大转矩。

7.4.4 2006 错误

此错误表示电机在惯性识别过程中未能持续转动。 在惯性识别过程完成前，确保电机持续转动非常重要。 此错误通常是由于 RampTime_sec 参数设置过高引起的。 当电机在惯性识别过程中停止转动时，可能导致估算惯性值高于实际惯性值。 减小 RampTime_sec 参数会增大惯性识别过程中施加到电机的转矩率。

7.5 不易识别惯性的应用

某些应用的特性使其很难识别系统惯性。 SpinTAC 速度识别的默认配置适用于使用典型电机的应用。 针对包含以下任一条件的电机应用，需要在 SpinTAC 速度识别配置中进行一些修改。

- 齿槽力大
- 摩擦力大
- 额定速度低
- 反电势大
- 启动电流大

7.5.1 自动泵（齿槽力大/摩擦力大）

很多自动泵的齿槽力或摩擦力都很大。 对于这些应用，减小 RampTime_sec 参数非常重要。 此参数位于 SpinTAC 速度识别全局结构的配置结构中。 RampTime_sec 值表示 Iq 参考电流值从 0 斜升到 1.0 PU 所用的时间（以秒为单位）。 减小此值意味着 Iq 参考电流在惯性识别期间会增大得更快。 这样可确保电机适当地减速。 如果电机没有适当减速，惯性估算过程会产生不良结果。

图 7-7 为摩擦力较大的自动泵惯性识别过程中的速度反馈。 当 RampTime_sec 设置为 10.0 时，惯性识别过程不会成功完成。 值得注意的是，较大的 RampTime_sec 会导致此过程延迟启动并且电机在测试中停止。 这两种情况都会导致惯性识别失败，并且 ERR_ID 设为 2003。 当 RampTime_sec 设置为 3.0 时，惯性识别不会延迟启动，并且电机也不会在测试中停止。 若要成功完成惯性识别，需要同时满足这两个条件。

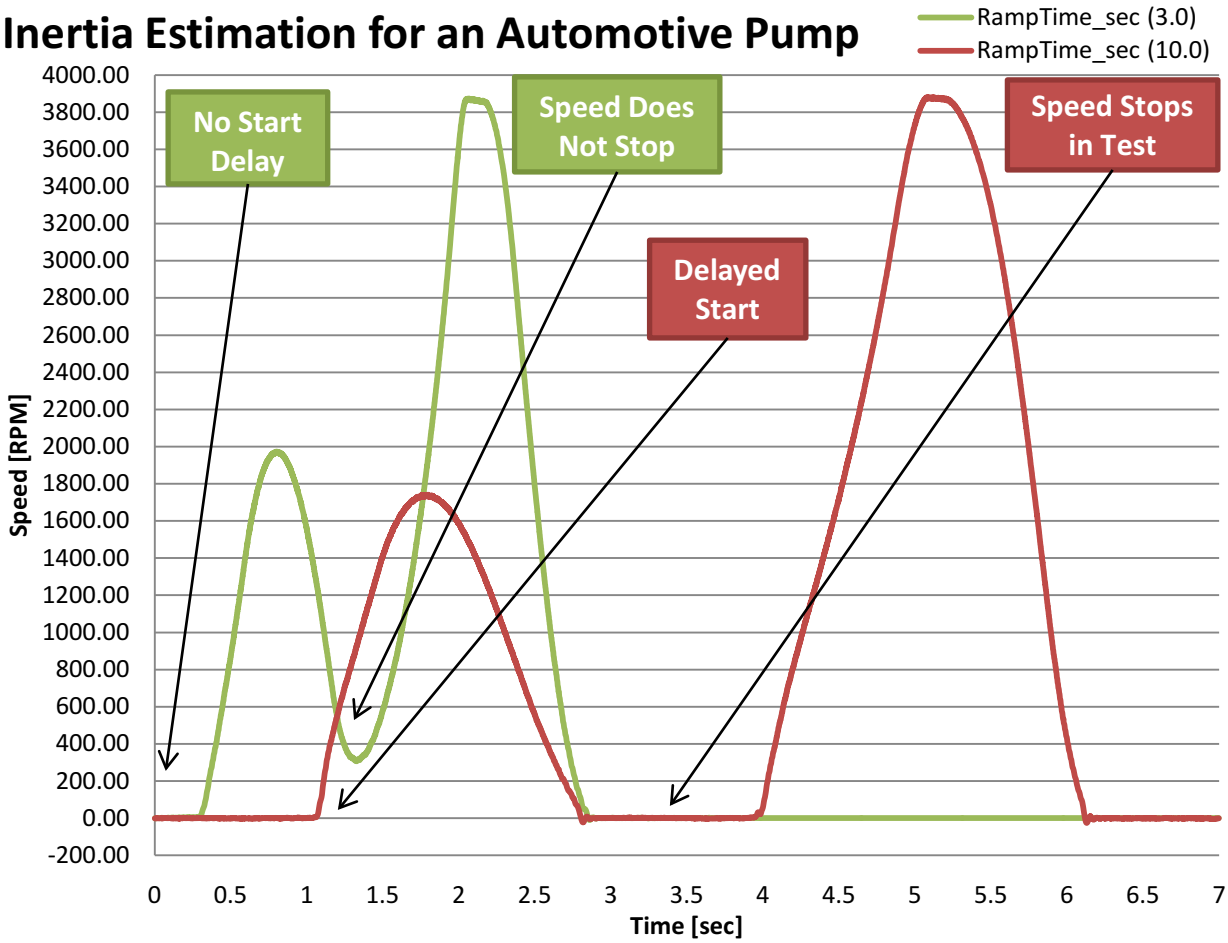


图 **7-7.** 自动泵惯性识别的速度反馈

7.5.2 直接驱动型洗衣机（额定转速低且反电势大）

大多数直接驱动型洗衣机使用额定转速低的电机。 在这些应用中，减小 VelPos 参数非常重要。 此参数位于 SpinTAC 速度识别全局结构的配置结构中。 VelPos 值表示惯性识别过程中的目标速度，单位为 pu/s。 如果目标速度大于电机额定速度的两倍，惯性识别将失败，因为电机无法达到足够的速度。 减小 VelPos 意味着惯性识别过程的目标速度会降低，但该过程会成功完成。 不建议在惯性识别过程中使用场强减弱来增大电机速度。 场强减弱会影响速度和转矩之间的关系。

图 7-8 是直接驱动型洗衣机惯性识别过程中的速度反馈。 注意，电机在约为 360rpm 的额定速度下转动时间超过 5 秒钟。 这意味着电机永远无法达到 VelPos 配置参数指定的目标速度，并且 SpinTAC 速度识别超时后会结束惯性识别过程。

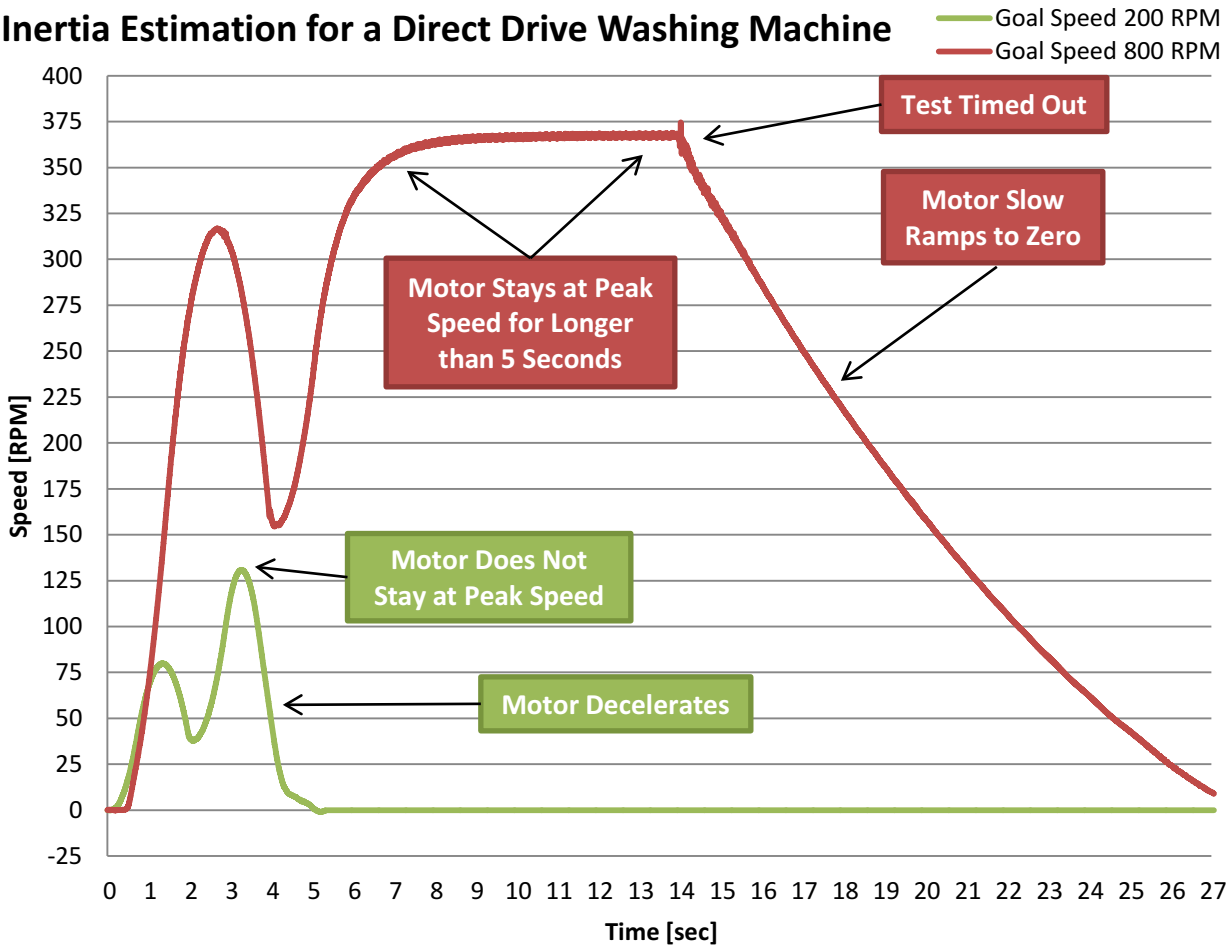


图 **7-8.** 直接驱动型洗衣机惯性识别的速度反馈

直接驱动型洗衣机同样具有较大的反电势，并且快速减速会导致直流总线过压。 降低目标速度会引起电机减速度下降，导致在直流总线上生成的电压较低。 目标速度由 VelPos 参数设定。

图 7-9 为惯性识别过程中的直流总线电压图。 速度反馈供参考。 注意，当目标速度设置为 400RPM 时，惯性识别过程中的直流总线电压升至 400 伏特。 为消除这种电压大幅升高，需减小 VelPos 配置参数。 当目标速度设置为 200RPM 时，惯性识别过程中的直流总线电压低于 350 伏特。

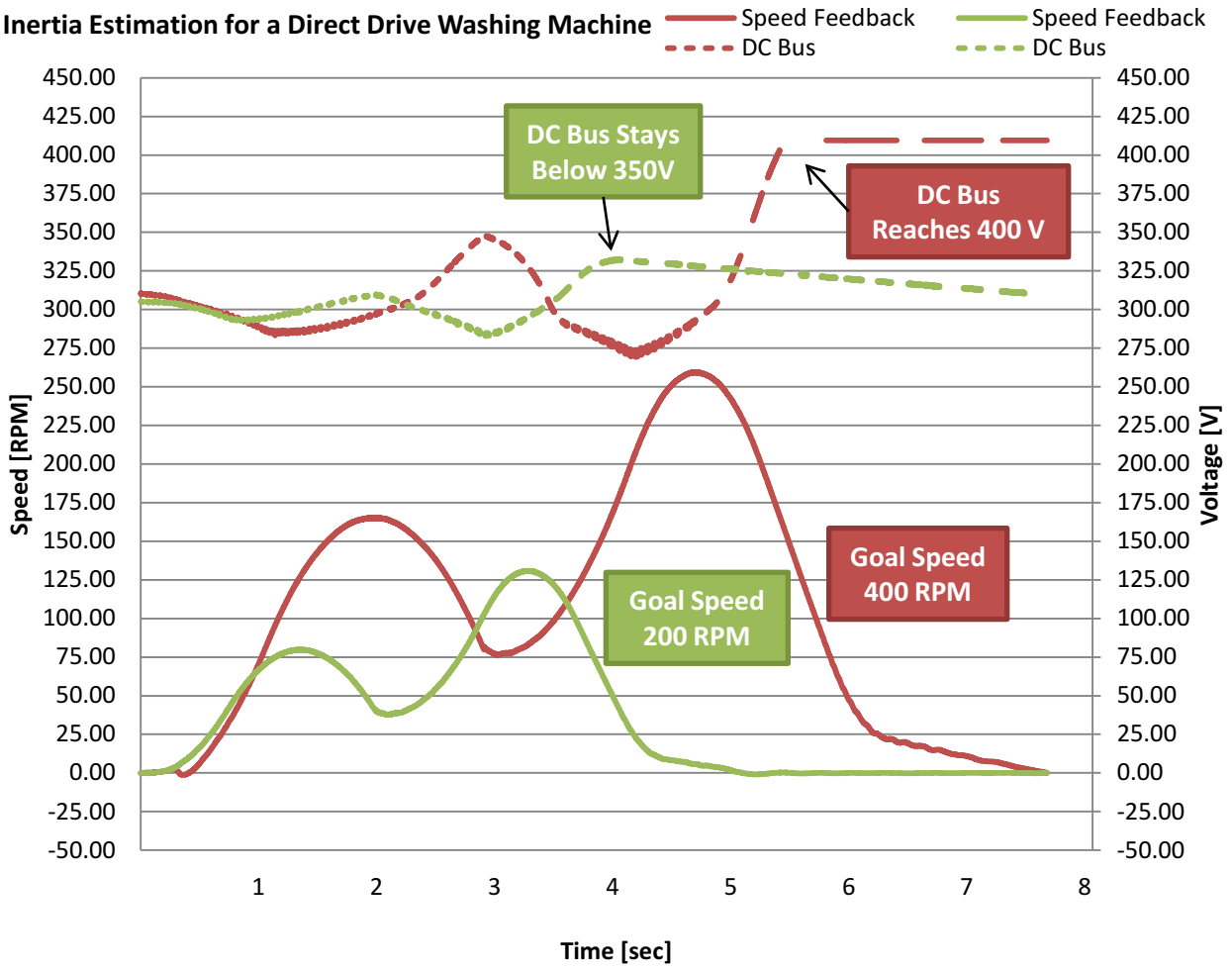


图 **7-9.** 直接驱动型洗衣机惯性识别的直流总线电压

7.5.3 压缩机（启动电流大）

压缩机通常不能完全消除负载转矩。 这会导致电机需要很大的 Iq 参考电流才能开始转动。 对于这些应用，增大 PosOut 参数非常重要。 此参数位于 SpinTAC 速度识别全局结构的配置结构中。 PosOut 值表示 Iq 参考电流（单位为 PU），它将作为惯性识别过程的一部分。 增大此值会在识别过程中为电机提供更多电流。 减小 RampTime_sec 参数也非常重要。 这会增加 Iq 参考电流施加到系统的速率。

图 7-10 是压缩机惯性识别过程中的速度反馈。 注意，速度没有快速加速至目标速度。 增至目标速度耗费了很长时间。 这说明电机要达到目标速度需要额外的转矩。 所需调整的配置参数为 PosOut。 增大此参数即可在惯性识别过程中为电机提供更大的转矩。 惯性识别过程仅会使用达到目标速度所需的转矩。 PosOut 参数最好大于所需的值。

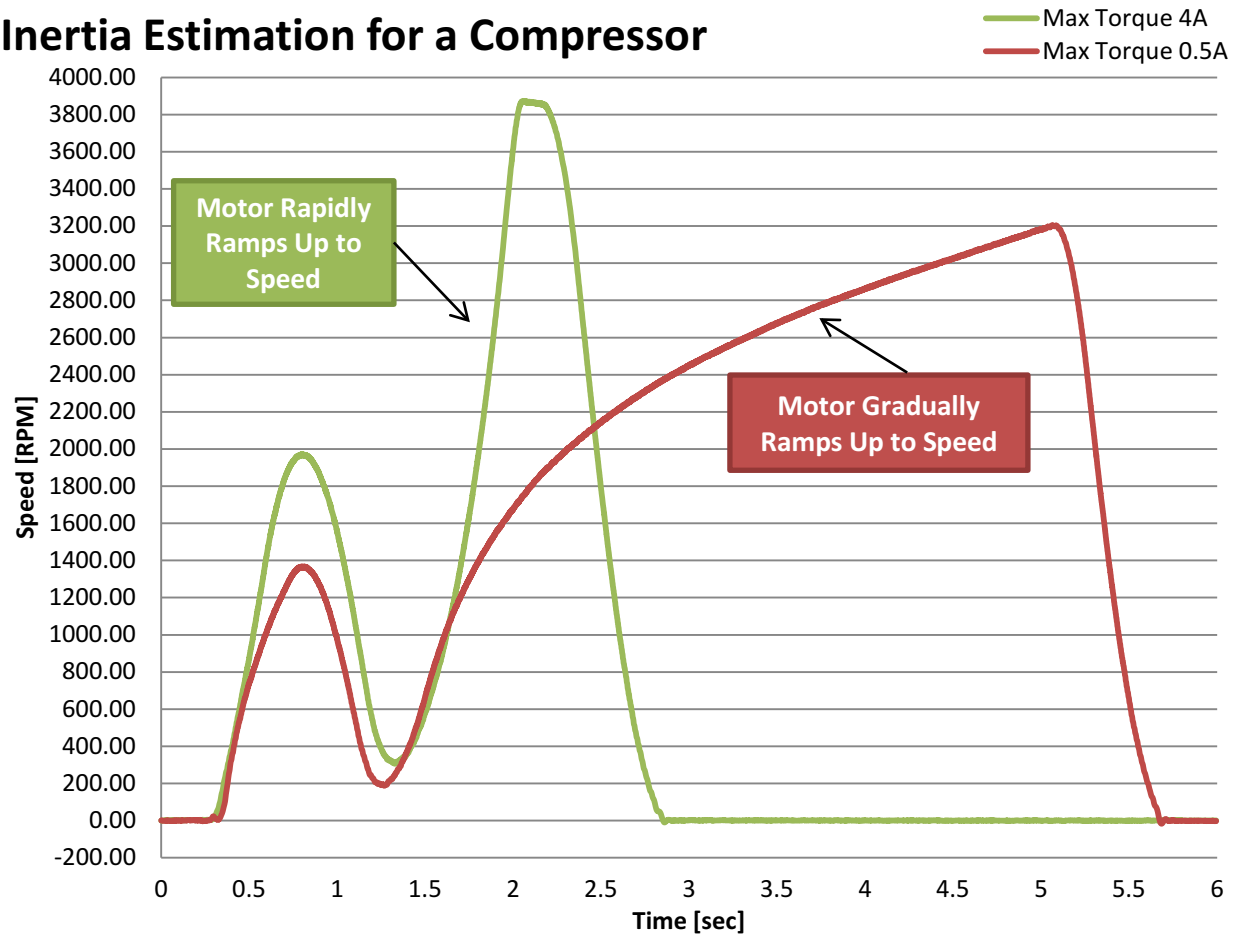


图 **7-10.** 压缩机惯性识别的速度反馈

很多压缩机无法反向运转。 惯性识别过程向系统施加反向 Iq 参考电流即可实现电机减速。 即使施加反向 Iq 参考电流，电机仍不会停止反向运转，因为惯性识别过程在电机开始反向运转前便会结束。

MCU 注意事项

本节将介绍实现成功执行 InstaSPIN-FOC 和 InstaSPIN-MOTION 所需注意的 MCU 相关事项：

- 列出支持 InstaSPIN 的器件并说明其具体需求
- 内存映射注意事项
- 时钟速率

  本节详细介绍 InstaSPIN 库所需的微控制器资源。 本文档中介绍了两种执行 InstaSPIN 的方法，具体取决于从安全 ROM 运行多少功能，或者从用户内存（RAM 或 闪存）中运行哪些功能：

  – 从 ROM 中执行 InstaSPIN-FOC 的全部功能，又称完全执行。

  – 仅从 ROM 中执行 FAST™ 功能，又称最小执行。

  另外，必须根据在用户内存中放置和执行代码的具体位置加以区分。 分两类讨论：

  – 从 ROM 中执行库，从 RAM 中加载和执行用户代码

  – 从 ROM 中执行库，从闪存中加载和执行用户代码
- 专门针对库执行以及此代码的加载与执行位置，在这份文档中讨论了以下资源类别：

  – CPU 利用率

  – 内存分配

  – 堆栈利用率
- 本节末尾的通用部分列出了数字和模拟引脚利用率，适用于 InstaSPIN 的各个运行模式。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

8.1 支持 InstaSPIN 的器件

目前包含 ROM 中的 InstaSPIN-FOC 或 InstaSPIN-MOTION 的器件如表 8-1 所示。

表 8-1. 支持 InstaSPIN 的器件

器件	ROM 中的 InstaSPIN-FOC	ROM 中的 InstaSPIN-MOTION	InstaSPIN TRM	器件数据表	器件 TRM
TMS320F28069M	V1.6	V2.1.8	SPRUHJ0	SPRS698	SPRUH18
TMS320F28069F	V1.6	不包含	SPRUHI9	SPRS698	SPRUH18
TMS320F28068M	V1.6	V2.1.8	SPRUHJ0	SPRS698	SPRUH18
TMS320F28068F	V1.6	不包含	SPRUHI9	SPRS698	SPRUH18
TMS320F28062F	V1.6	不包含	SPRUHI9	SPRS698	SPRUH18
TMS320F28054M	V1.7	V2.1.8	SPRUHW1	SPRS797	SPRUHE5
TMS320F28054F	V1.7	不包含	SPRUHW0	SPRS797	SPRUHE5
TMS320F28052M	V1.7	V2.1.8	SPRUHW1	SPRS797	SPRUHE5
TMS320F28052F	V1.7	不包含	SPRUHW0	SPRS797	SPRUHE5
TMS320F28027F	V1.7	不包含	SPRUHP4	SPRS523	SPRUFN3
TMS320F28026F	V1.7	不包含	SPRUHP4	SPRS523	SPRUFN3

除了 InstaSPIN 技术已添加到 ROM 的指定区域外，器件保持不变。 有关所使用器件的全部详情，请参见器件专用数据表和勘误表。

8.1.1 *softwareUpdate1p6() - 用户代码中所需的函数*

函数 softwareUpdate1p6() 是针对 InstaSPIN-FOC v1.6 中的漏洞的应急措施，当使用 user.h 中的电感时，可用于校正电感从亨利转化为标么值的操作。 当使用 InstaSPIN-FOC v1.6 时，每当从 user.h 加载电机参数时，都需要调用此函数。

此补丁中包含以下修复措施：

- 已添加电感最大值。 因此，我们希望根据此最大值标定电感标么值。 这也会影响电感值的 Q 格式。
- 根据这些新的电感标么值设置电流控制器增益值（Id/Iq 电流控制器）。

下面是该补丁的源代码，在每个 InstaSPIN-FOC v1.6 和 InstaSPIN-MOTION 实验示例中均有使用。

```
void softwareUpdate1p6(CTRL_Handle handle)
{
 CTRL_Obj *obj = (CTRL_Obj *)handle;

 float_t fullScaleInductance = EST_getFullScaleInductance(obj->estHandle);
 float_t Ls_coarse_max = _IQ30toF(EST_getLs_coarse_max_pu(obj->estHandle));
 int_least8_t lShift = ceil(log(obj-
>motorParams.Ls_d/(Ls_coarse_max*fullScaleInductance))/log(2.0));
 uint_least8_t Ls_qFmt = 30 - lShift;
 float_t L_max = fullScaleInductance * pow(2.0,lShift);
 _iq Ls_d_pu = _IQ30(obj->motorParams.Ls_d / L_max);
 _iq Ls_q_pu = _IQ30(obj->motorParams.Ls_q / L_max);

 float_t RoverL = obj->motorParams.Rs/obj->motorParams.Ls_d;
 float_t fullScaleCurrent = EST_getFullScaleCurrent(obj->estHandle);
 float_t fullScaleVoltage = EST_getFullScaleVoltage(obj->estHandle);
 float_t ctrlPeriod_sec = CTRL_getCtrlPeriod_sec(ctrlHandle);
 _iq Kp = _IQ((0.25*obj->motorParams.Ls_d*fullScaleCurrent)/(ctrlPeriod_sec*fullScaleVoltage));
 _iq Ki = _IQ(RoverL*ctrlPeriod_sec);
 _iq Kd = _IQ(0.0);
```

```
 // store the results
 EST_setLs_d_pu(obj->estHandle,Ls_d_pu);
 EST_setLs_q_pu(obj->estHandle,Ls_q_pu);
 EST_setLs_qFmt(obj->estHandle,Ls_qFmt);

// set the Id controller gains
 PID_setKi(obj->pidHandle_Id,Ki);
 CTRL_setGains(ctrlHandle,CTRL_Type_PID_Id,Kp,Ki,Kd);

// set the Iq controller gains
 PID_setKi(obj->pidHandle_Iq,Ki);
 CTRL_setGains(ctrlHandle,CTRL_Type_PID_Iq,Kp,Ki,Kd);

 return;
} // end of softwareUpdate1p6() function
```

8.2 ROM 和用户内存概述

8.2.1 ROM 中的 InstaSPIN-FOC 完全执行

当应用要求允许从 ROM 中运行所有磁场定向控制 (FOC) 块，并且没有其它功能需求时（比如，专门的电流控制算法或者克拉克变换等），建议使用完全执行。 这种执行方式会使用 ROM 中的所有库内容，并且执行全套函数和函数块，称为 InstaSPIN-FOC。 完全执行不仅包括 FAST 算法，还包括其余 FOC 块。 下列框图说明了采用完全执行时包含的几个块如何实现从 ROM 中运行全部 FOC 代码、释放更多内存资源以及利用 ROM 中的 0 等待状态执行。 ROM 也是只执行 ROM，由于不能够写入或读取，仅能执行，因此可提供更高的安全等级。

对于 F2802xF 器件，由于 ROM 容量减小，某些功能块会加载到用户内存中。

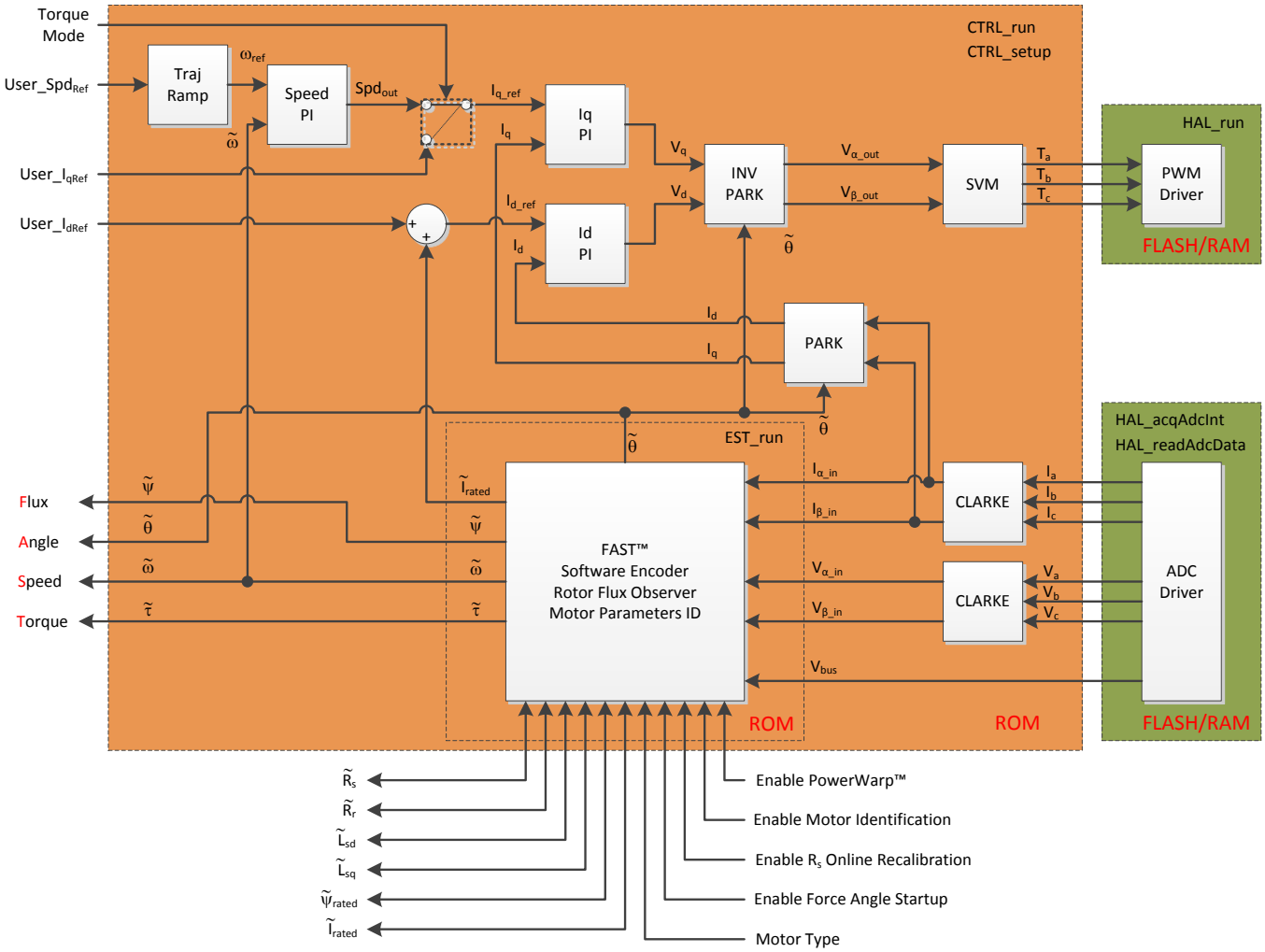


图 8-1. ROM 中的 InstaSPIN-FOC 完全执行

8.2.1.1 从 ROM 和 RAM 中执行

即使整个 InstaSPIN 库从 ROM 中执行，仍然有一些函数需要从用户内存中加载和运行。 这些函数是库连接硬件外设的接口，如图 8-1 所示。 与驱动器对象 (HAL_Obj) 相关的所有函数均连接到硬件并且需要放置在用户内存中。 性能数据将取决于执行这些用户函数的位置。 本节说明了所有用户函数在 RAM 外放置和运行时的性能数据。

从 CPU 性能角度来看，由于 RAM 不需要等待状态，因此从 RAM 加载和执行用户函数有显著优势。 另一方面，将用户函数加载到 RAM 中会消耗易失性存储空间，所以用户必须考虑变量可用的 RAM 总量。 堆栈利用率和库所使用的引脚与用户代码驻留的位置（RAM 或闪存）无关。

8.2.1.2 从 ROM 和闪存中执行

尽管变量和堆栈仍需要使用部分可用的 RAM，但是将用户函数加载至闪存可减少 RAM 的消耗。 由于闪存需要等待状态，当从闪存加载用户代码时的另一点考量是 CPU 执行时间。

由于 F2802xF 器件的 ROM 容量减小，因此不可能从 ROM 中完全执行 InstaSPIN-FOC。 关于运行 InstaSPIN-FOC 最小执行的详细信息，请参见8.2.2 节。

堆栈使用量以及使用的引脚与加载用户代码至 RAM 时所用的相同。 不过，此处仍列出了这些参数，以便为特殊执行提供完整的资源使用量列表。

8.2.2 ROM 中的 InstaSPIN-FOC 最小执行

一些应用需要对磁场定向控制功能进行更多控制。 有此需求的应用可以使用 InstaSPIN 的最小执行，仅从 ROM 中运行 FAST 估算器并将其它软件块移至用户内存中。 由于估算器的源代码为 TI 专有，因此它必须保留在 ROM 中。

图 8-2 用不同颜色突出显示了从 ROM 和用户内存中执行的功能。

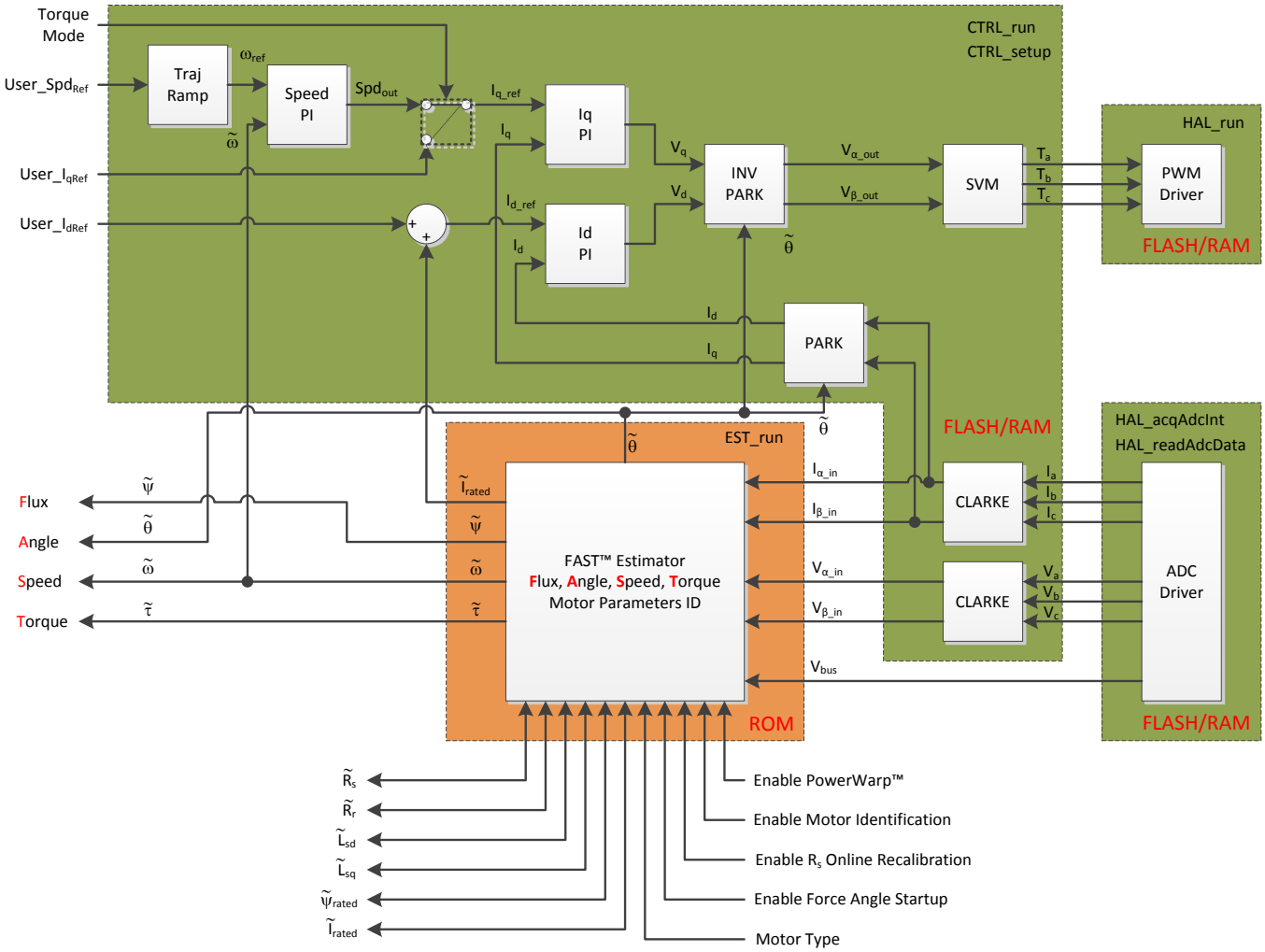


图 8-2. ROM 中的 InstaSPIN-FOC 最小执行

请注意，只有运行估算器的函数 (Est_run) 从 ROM 中执行。 其余函数均从用户内存（RAM 或闪存）中执行。 在以下小节中，描述了使用最小执行时的 InstaSPIN 性能。

8.2.2.1 从 ROM 和 RAM 中执行

当从 RAM 中加载和执行用户函数时，代码执行速度比闪存快，但此时代码需要占用一部分可用 RAM。 如果某个特殊应用需要最大执行速度，且可用 RAM 满足非易失性存储器要求，这是最佳选择。

8.2.2.2 从 ROM 和闪存中执行

如之前章节所述，将用户函数加载至闪存可以减少 RAM 的消耗。 不过，闪存执行的速度较慢，因为闪存需要等待状态才能正确操作；因而影响了执行速度。

8.2.3 ROM 中的 InstaSPIN-MOTION

InstaSPIN-MOTION 采用模块化设计，当前在 F2806xM 器件上可用。 消费者可以在部署系统时决定闪存中包含哪些函数。

InstaSPIN-MOTION 控制、识别和移动组件在 ROM 中可用，InstaSPIN-MOTION 规划和公共库在 RAM 中可用。

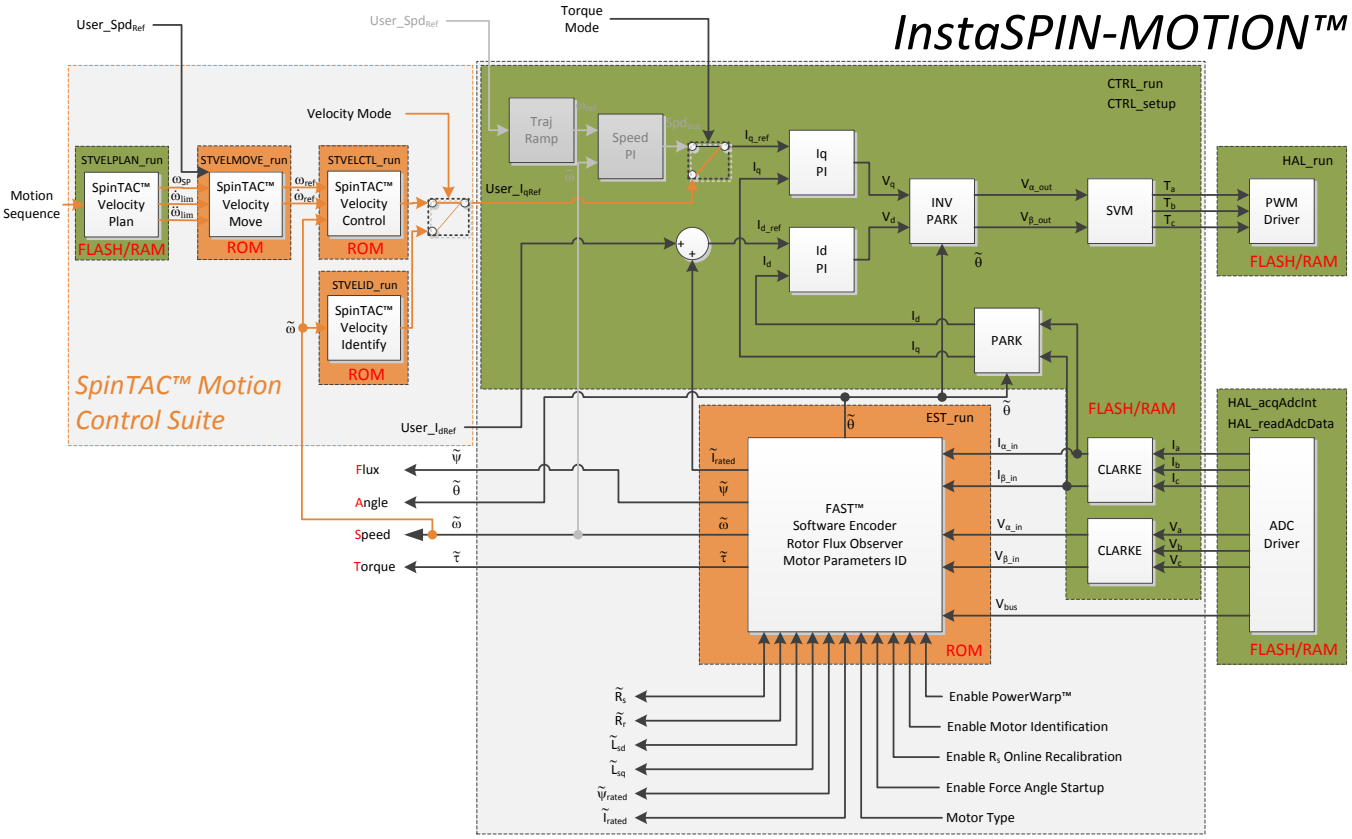


图 8-3. ROM 中的 InstaSPIN-MOTION

8.2.3.1 在主中断内执行 InstaSPIN-MOTION

各个 InstaSPIN-MOTION 组件可在主中断服务程序中单独执行。 应以固定的抽取率从主 ISR 中调用每个 InstaSPIN-MOTION 组件。 建议调用 InstaSPIN-MOTION 组件的速率至少慢于 PWM 中断 ISR 或者主 ISR 的 10 倍。

包括控制、识别和移动在内的 InstaSPIN-MOTION 核心功能仅可从 ROM 中执行。 包含用户的库和规划可从 RAM 或者闪存中执行。

8.2.3.2 从 RAM 中执行库

从 RAM 中加载和执行公共库体现了 CPU 方面的性能优势，因为 RAM 不需要等待状态。 不过，对于需要大量 RAM 的应用，从闪存中执行库可能是更好的选择。

8.2.3.3 从闪存中执行库

尽管某些 RAM 会用于变量和堆栈，但是从闪存中加载和执行公共库可以减少 RAM 的消耗。 此时的一个重要考量是闪存等待状态。

8.3 关于 CPU 负载和内存占用量测量的详细信息

8.3.1 CPU 利用率测量详情

为了尽可能精确地测量 CPU 周期，需使用三个可用 CPU 定时器中的一个。 定时器计时应尽可能快，这样才能为每次执行提供最大数量的计数。 所以，将计时器的输入时钟设置为与 CPU 相同的时钟，并且不使用分频器。 以下代码示例说明了在执行目标函数后如何重新加载并读取计时器计数。

```
// reload the CPU timer
HAL_reloadTimer0(halHandle);

// run the controller
CTRL_run(ctrlHandle,halHandle,&gAdcData,&gPwmData);

// get the CPU timer count
timercount = HAL_getCountTimer0(halHandle);
```

即使重新加载并读取计时器计数的函数非常高效，当使用后续章节中提供的数据时，需要考虑大约 5 个 CPU 周期的溢出。

CPU 利用率表包含一个通过运行成千上万个中断计算得出的最小值列 (Min)，将每个中断周期的时间与最小值相比较，如果小于最小值，则更新最小值。 用同样的方法可计算出周期的最大值，或者最大值列。 通过计算累计周期数，用累计中断数量计数除以该值即可得出平均值列。 平均值列类似于最小值和最大值列，根据成千上万个中断计算得出一个稳定的平均值。

CPU 利用率表列出了几种可选配置，主要更改三个参数：

- 中断与 控制器 (ISR vs CTRL) 抽取率或节拍率
- 控制器与 估算器 (CTRL vs EST) 抽取率或节拍率
- Rs 在线重校准特性

对于关于抽取率的前两项，图 8-4 显示了 InstaSPIN 的完整软件执行时钟树。 此图表说明了从 CPU 时钟到估算器的全部分频过程。 这里仅可改变突出显示的节拍率，因为这两部分是影响 CPU 使用率的主要因素。 更改速度控制器、电流控制器或者跟踪生成节拍率后，CPU 使用率不会显著变化，因此在 CPU 利用率测量期间这些参数始终保持不变。

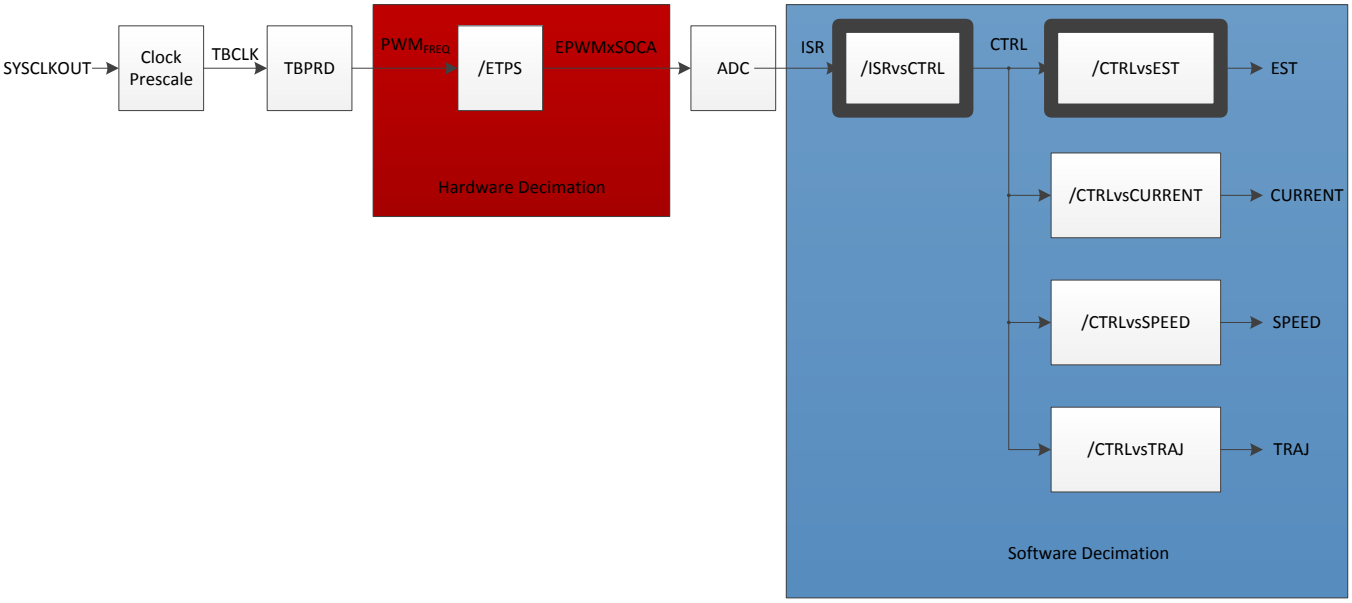


图 **8-4. InstaSPIN** 软件执行时钟树

关于 InstaSPIN 软件执行时钟树的更多信息，请参见9.1 节。

第三个参数为 Rs 在线重校准特性，可针对 CPU 利用率测量启用和禁用。

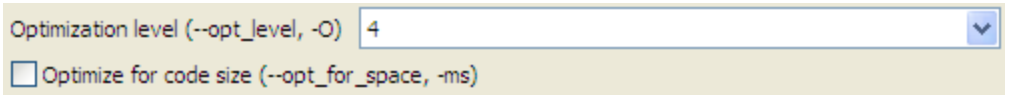
由于此参数会显著影响 CPU 利用率，所以同样需要考虑。 关于 Rs 在线重校准的更多信息，请参见Chapter 15。

8.3.2 内存分配测量详情

内存分配取决于几个因素。 以下给出影响内存分配的一些因素以及各自的配置：

- 编译器版本：6.1.0

- 优化设置：4 级



- 其它用户代码：无。 使用最小代码连接 InstaSPIN 库。

此处为显示编译器选项的完整命令行：

-v28 -ml -mt -O4 -g

要使用最小变量集连接 InstaSPIN 库，代码中必须含有下列全局变量：

```
CTRL_Handle ctrlHandle;

HAL_Handle halHandle;

USER_Params gUserParams;

HAL_PwmData_t gPwmData;
```

```
HAL_AdcData_t gAdcData;
```

关于这些变量及数据类型的完整说明，请参见各个实验。 某些其它变量在控制软件流程时非常有用，比如用于启用或禁用系统的标志，以及用于显示电机参数的其它全局变量。 由于库功能不需要使用这些变量，因此它们不会包含在用于测量内存分配的项目中。

内存分配表中包含五个不同部分：

- 库接口 (.ebss)。 表格的此部分指示用于连接 InstaSPIN 库的各个变量（在未初始化的内存区域或 .ebss 中），这些库位于用户内存或 ROM 中。
- 库 (.ebss)。 这部分是指用于 InstaSPIN 库本身的变量，并且此区域的内存消耗不会改变。
- 代码 (.text)。 这部分是指在用户内存中执行的实际代码。 此代码区域会根据代码的加载位置（RAM 或闪存）进行更改。 如果编译器优化设置改变，它也会发生更改。 另外，如果用户执行的代码大部分来自 ROM，则此代码部分会被最小化。
- IQmath (.text)。 为 IQmath 相关代码分配的内存取决于 InstaSPIN 库的位置。 采用 InstaSPIN 完全执行时，大部分代码位于 ROM 中，IQmath 代码会被最小化，因为 ROM 代码本身包含数学代码，不需要占用用户内存来执行 IQmath 操作。 采用 InstaSPIN 最小执行时需要在用户内存中执行运算量更大的代码，因此，用户内存中需要使用某些额外的 IQmath 函数增加分配给 IQmath 代码的内存，从而完成 InstaSPIN 最小执行。
- 使用的最大堆栈 (.ebss)。 此内存区域将在本文档的后续章节中进行说明。

8.3.3 在 ROM 中构建 IQmath

ROM 中的库由 IQmath 库版本 1.5c 构建。 ROM 中的所有可执行代码使用 ROM 自身执行的函数，所以 ROM 代码不需要依赖外部添加的 IQmath 功能。

不过，使用 IQmath 操作从用户内存中执行的代码需要添加 IQmath 库。 这个外部添加的 IQmath 库可以是任何已发布的库版本，不一定是 1.5c。 用户可以结合自己的 IQmath 库版本，并且仍然从使用 IQmath 库版本 1.5c 的 ROM 中执行代码。 所有代码示例都包括 CCStudio 项目中的完整 IQmath 库。

8.3.4 堆栈利用率测量详情

堆栈利用率可通过以下步骤测得：

- 器件复位。
- 连接器放置堆栈的整个内存区域始化为已知值（0x5555AAAA、0x12345678、0x0BADF00D、0xCAFEBABE、0xFEEDFACE 等）。
- 运行几分钟代码，直到运行完所有分支程序。
- 在初始值出现前，分析堆栈所在的内存区域并查找最后一个更改的值。
- 计算修改后的内存区域。

尽管堆栈利用率方法不能保证所需的绝对堆栈字数，但它针对所需堆栈区域给出了一个良好建议。 不过，建议堆栈部分大于最低空间要求，从而使整个项目更加稳健。 关于此主题的更多详细信息，请参见《*TMS320C28x DSP 在线堆栈溢出检测*》应用报告（文献编号 SPRA820）。

后续章节表格中列出的数字表示使用的最大堆栈值，而不是构建选项所保留的堆栈区域。 如本节所述，建议保留略大一些的堆栈区域以避免可能出现的堆栈溢出情况，尤其是在添加更多代码，其它中断或者更多变量时。

8.3.5 InstaSPIN 主中断

InstaSPIN 库以固定的频率从单个中断服务程序中执行。 该主 ISR 默认由 ADC 转换结束中断触发。 此转换首先由 PWM 模块以固定速率启动。 一旦在 ISR 中（本例中为 mainISR()）执行，则需要一系列函数调用从 ADC 中获取数据并调用 ROM 中的函数。 以下代码为相关示例：

```
interrupt void mainISR(void)
{

  // acknowledge the ADC interrupt
  HAL_acqAdcInt(halHandle,ADC_IntNumber_1);


  // convert the ADC data
  HAL_readAdcData(halHandle,&gAdcData);


  // run the controller
  CTRL_run(ctrlHandle,halHandle,&gAdcData,&gPwmData);


  // run the driver -- set the pwm compare values
  HAL_writePwmData(halHandle,&gPwmData);


  // setup the controller
  CTRL_setup(ctrlHandle);


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

为了说明 InstaSPIN 的性能，我们会首先考虑顶层方法，包括主 ISR 中的五个函数调用（请参见图 8-5）。

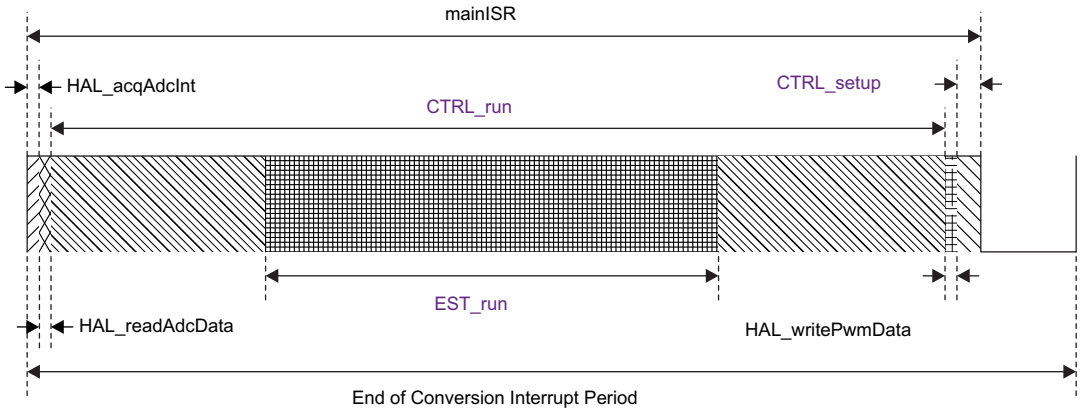


图 **8-5.** 主 **ISR** 中的函数调用

8.3.6 时钟速率

InstaSPIN-FOC 和 InstaSPIN-MOTION 为实时控制系统，因此它们的性能与所用处理器的 CPU 时钟速率直接相关。 可以通过降低 CPU 时钟速率来测试 InstaSPIN-FOC 和 InstaSPIN-MOTION 的性能是否满足应用要求。 CPU 负载方面的内容将在8.5 节 中介绍，InstSPIN-FOC 和 InstaSPIN-MOTION 的软件时钟树将在Chapter 9 中介绍。

8.4 内存占用量

InstaSPIN-FOC 存储在器件 ROM 中，ROM 为只执行 (EXE-only) 内存，不可被软件或者 IDE 读取。

表 **8-2.** 为 **InstaSPIN-FOC** 库分配的内存

特性	2806xF	2806xM	2805xF	2805xM	2802xF
FAST	是	是	是	是	是
SpinTAC	否	是	否	是	否
可被控制的最多电机数量	2	2	2	2	1
可重定位控制器结构	否	否	是	是	是
FAST 版本	1.6	1.6	1.7	1.7	1.7
需要向项目添加公用库	否	否	否	否	是
ROM 库起始位置 [地址、十六进制]	3F 8000	3F 8000	3F 8808	3F 8808	3F C000
库需要的 RAM [大小、十六进制、字]	800	800	800	800	200
库起始 RAM [地址、十六进制]	01 3800	01 3800	00 8000	00 8000	00 0600

8.4.1 器件内存映射

8.4.1.1 F2806xF 和 F2806xM 器件

对于 F2806xF 和 F2806xM 器件，InstaSPIN-FOC v1.6 和 SpinTAC v2.1.8 存储在 0x3F8000 至 0x3BFFF 地址范围内，并且 L8-RAM 的最后部分保留用于 InstaSPIN 变量，地址范围在 0x013800 至 0x013FFF 之间。 请注意，InstaSPIN-FOC 和 InstaSPIN-MOTION 的变量范围是固定的，不可使用。 L8-RAM 的其余部分可供用户使用 (0x012000 至 0x0137FF)；请参见图 8-6。

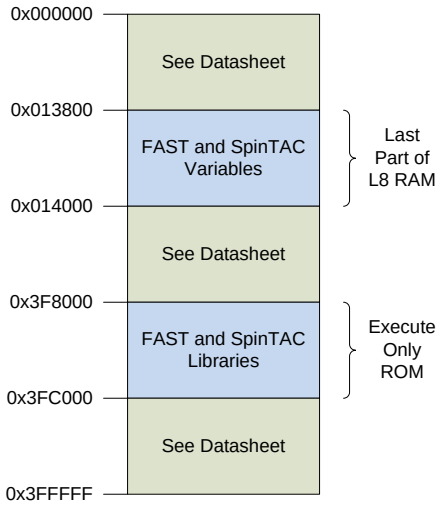


图 **8-6.** 为 **InstaSPIN-FOC** 和 **SpinTAC** 库分配的 **F2806x** 和 **F2806xM** 内存

除了存储在 ROM 中的 InstaSPIN-FOC v1.6 和 SpinTAC v2.1.8 外，ROM 中的几个表格已移动到新地址。如果要移植引用 ROM 中的 IQmath 表的现有代码，连接器命令文件将需要更新的地址，如表 8-3 所示。

表 **8-3. ROM** 表地址

ROM 中的起始地址	F2806x	F2806xF 和 F2806xM
FPUTABLES	0x03fd860	0x03fd590
IQTABLES	0x03fdf00	0x03fdc30
IQTABLES2	0x03fea50	0x03fe780
IQTABLES3	0x03feadc	0x03fe8b6

8.4.1.2 F2805xF 和 F2805xM 器件

对于 2805xF 和 2805xM 器件，InstaSPIN-FOC v1.7 和 SpinTAC v2.1.8 存储在 0x3F8808 至 0x3FC52F 地址范围内，并且 L0-RAM 保留用于 InstaSPIN 变量，地址范围在 0x008000 至 0x0087FF 之间，请参见图 8-7。

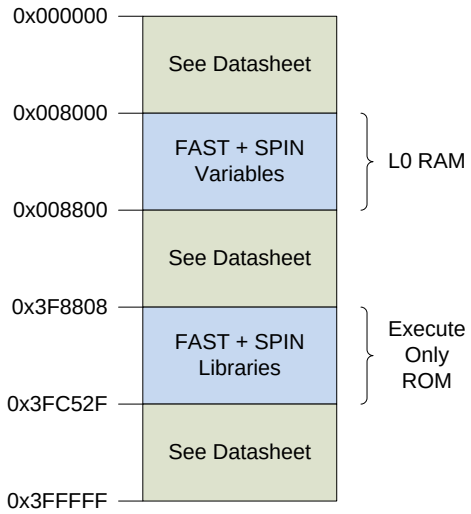


图 **8-7.** 为 **InstaSPIN-FOC** 和 **SpinTAC** 库分配的 **F2805x** 和 **F2805xM** 内存

8.4.1.3 F2802xF 器件

对于 2802xF 器件，InstaSPIN-FOC v1.7 存储在 0x3FC000 至 0x3FDFFF 地址范围内，并且 M1-RAM 的最后部分保留用于 InstaSPIN 变量，地址范围在 0x000600 至 0x0007FF 之间。 请注意，InstaSPIN-FOC 的变量范围是固定的，不可使用。 M1-RAM 的其余部分可供用户使用（0x000400 至 0x0005FF）；请参见图 8-8。

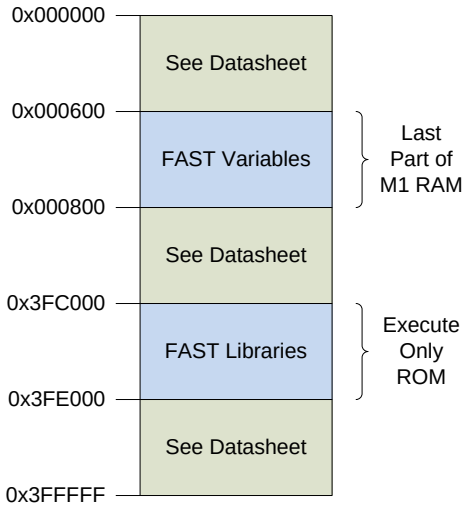


图 **8-8.** 为 **InstaSPIN-FOC** 库分配的 **F2802xF** 内存

8.4.2 InstaSPIN 内存占用量

8.4.2.1 F2806xF 和 F2805xF 器件

表 8-4 汇总了本文档中介绍的四种配置所用的内存。 请注意，当所使用 ROM 中的函数减少时，代码尺寸将增加。

表 8-4. InstaSPIN-FOC 占用的 F2806xF 和 F2805xF 器件内存总量

代码配置		内存大小（16 位字）			使用的最大堆栈（16 位字）
ROM 代码	用户代码	RAM	闪存	总计	
完全执行	RAM	0x1870	0x0000	**0x1870**	0x0120
完全执行	闪存	0x001E	0x186C	**0x188A**	0x0120
最小执行	RAM	0x1F31	0x0000	**0x1F31**	0x0120
最小执行	闪存	0x001E	0x1F2D	**0x1F4B**	0x0120

8.4.2.2 F2802xF 器件

表 8-5 汇总了仅可用于 F2802xF 器件配置的内存。

表 8-5. InstaSPIN-FOC 占用的 F2802xF 器件内存总量

代码配置		内存大小（16 位字）			使用的最大堆栈（16 位字）
ROM 代码	用户代码	RAM	闪存	总计	
最小执行	闪存	0x06B2	0x2DD8	**0x348A**	0x0120

8.4.2.3 F2806xM and F2805xM 器件

要计算 InstaSPIN-MOTION 的内存用量，将 InstaSPIN-FOC 内存用量与表 8-6 中的 SpinTAC 内存用量相加。 SpinTAC 速度规划和 SpinTAC 位置规划的不同内存要求表示项目中所使用的配置函数数量。 RAM 大小可从连接器“.ebss”区域获取，闪存大小则从“.text”获取。

表 8-6. 针对 SpinTAC 组成部分的代码尺寸和 RAM 用量

组成部分	代码 (.text)（16 位字）	RAM (.ebss)（16 位字）
速度控制	0X2E6	0x4C
速度移动	0x488	0x5C
速度规划（最小值）	0x666	0x4E
速度规划（最大值）	0x14BA	0x4E
速度识别	0x392	0x3C
位置转换器	0x21C	0x4C
位置控制	0x416	0x62
位置移动	0x13A4	0xCC
位置规划（最小值）	0x7AE	0x60
位置规划（最大值）	0x16F4	0x60

节 14.1.1.5 在 SpinTAC 组成部分单个运行时，分解了 SpinTAC 组成部分的最大堆栈利用率。 包括了 InstaSPIN-FOC 的堆栈使用量。

表 8-7. SpinTAC 组成部分 + InstaSPIN-FOC 的堆栈利用率

配置（InstaSPIN-FOC 所有运行情况）	使用的最大堆栈（16 位字）
速度控制	0x0120
速度移动	0x0120
速度规划 + 移动 + 控制	0x0120
速度识别	0x0120
位置转换器	0x0120
位置控制	0x0120
位置移动	0x0120
位置规划 + 移动 + 控制	0x0120

8.4.3 内存等待状态

关于其它等待状态选项，请参见所用器件的器件专用数据表。

8.4.3.1 F2806xF/M 器件

当 F2806xF/M 器件以 90MHz 速率在闪存中执行代码时，为 CPU 执行时间测量设置的等待状态如表 8-8 所示。

表 8-8. CPU 执行时间等待状态（F2806xF 和 F2806xM 器件）

页等待状态	随机等待状态	OTP 等待状态
3	3	5

8.4.3.2 F2805xF/M 器件

当 F2805xF/M 器件以 60MHz 速率在闪存中执行代码时，为 CPU 执行时间测量设置的等待状态如表 8-9 所示。

表 8-9. CPU 执行时间等待状态（F2805xF 和 F2805xM 器件）

页等待状态	随机等待状态	OTP 等待状态
2	2	3

8.4.3.3 F2802xF 器件

当 F2802xF 器件以 60MHz 速率在闪存中执行代码时，为 CPU 执行时间测量设置的等待状态如表 8-10 所示。

表 8-10. CPU 执行时间等待状态（F2802xF 器件）

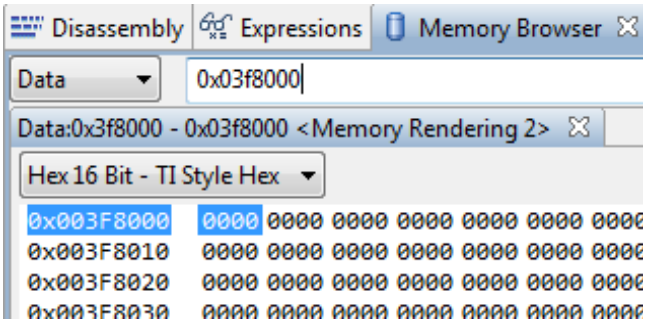
页等待状态	随机等待状态	OTP 等待状态
2	2	3

8.4.4 用于仅 RAM 执行的所需闪存配置

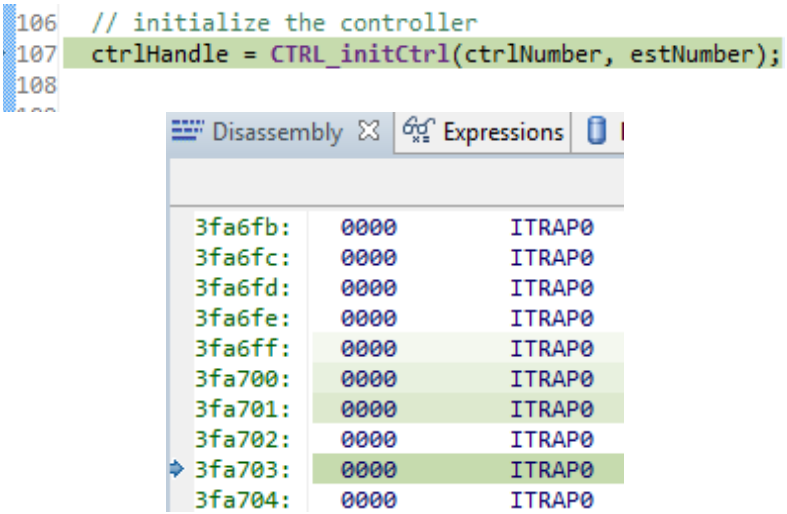
InstaSPIN-FOC 和 InstaSPIN-MOTION 从 ROM 中执行，但也要访问 F2806xF 和 F2806xM 上的 OTP，OTP 是一项基于闪存的技术，要求配置闪存。 因此，运行本指南中的实验时，您会注意到文件 Flash.c 包含在项目中。 实验中的函数 FLASH_init() 从 HAL_init() 中调用，HAL_init() 从 main() 中调用。 MotorWare 为器件上包括闪存在内的所有外设提供驱动器。

8.4.5 只执行内存的调试 (IDE)

尽管 nstaSPIN-FOC 和 InstaSPIN-MOTION 存储在只执行 ROM 中，除了无法看到内存内容外，调试过程非常类似。 如果使用内存浏览器从 CCStudio 中查看内存，只执行内存的内容全为 0。



如果在反汇编窗口打开时通过只执行内存单步执行，它将显示读取所有 0 内容的操作码 (ITRAP0)。 以下是相关示例，单步执行 Lab02A 中的 函数 CTRL_initCtrl()。



8.5 CPU 负载

8.5.1 F2806xF 器件

8.5.1.1 CPU 周期数

下列表格汇总了每个函数的所有性能数据，其中的用户代码均以 InstaSPIN 库的最小执行方式从闪存中加载并执行。 请注意，采用不同执行方式时周期数不会发生显著变化，因为这些配置中的 FAST 估算器块均保留在 ROM 中。 此估算器模块占用所有 InstaSPIN-FOC 块的大部分周期。 关于管理 ISR 中的执行时间的详细信息，请参见9.1 节。

8.5.1.1.1 RAM 中的用户代码

8.5.1.1.1.1 完全执行

表 **8-11.** 在 **RAM** 中进行完全执行的内存用量

区域	内存用量（**16** 位字）	
	RAM	闪存
库接口 (.ebss)	0x018C	×
库 (.ebss)	0x0800	×
代码 (.text)	0x1870	×
IQmath (.text)	0x0014	×

表 8-12 汇总了每个函数的所有性能数据，其中的用户代码均以 InstaSPIN 库的完全执行方式从 RAM 中加载并执行。

表 **8-12.** 在 **RAM** 中进行完全执行

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
HAL_acqAdcInt	23	23	23	×	✓	×
HAL_readAdcData	106	106	106	×	✓	×
Ctrl_run						
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	2345	2355	2425			
CTRL vs EST = 2	1154	1760	2425			
CTRL vs EST = 3	1154	1562	2425			
ISR vs CTRL = 2，CTRL vs EST = 1	58	1207	2425			
CTRL vs EST = 2	58	909	2425			
CTRL vs EST = 3	58	810	2425			
ISR vs CTRL = 3，CTRL vs EST = 1	58	824	2425			
CTRL vs EST = 2	58	626	2425			
CTRL vs EST = 3	58	560	2425	✓	×	×
Rs 在线启用，ISR vs CTRL = 1，CTRL vs EST = 1	2807	2821	2894			
CTRL vs EST = 2	1154	1993	2894			
CTRL vs EST = 3	1154	1717	2894			
ISR vs CTRL = 2，CTRL vs EST = 1	58	1439	2894			
CTRL vs EST = 2	58	1025	2894			
CTRL vs EST = 3	58	887	2894			
ISR vs CTRL = 3，CTRL vs EST = 1	58	979	2894			
CTRL vs EST = 2	58	702	2894			
CTRL vs EST = 3	58	610	2894			
HAL_writePwmData	62	62	62	×	✓	×
CTRL_setup	37	51	178	✓	×	×

8.5.1.1.1.2 最小执行

表 **8-13.** 在 **RAM** 中进行最小执行的内存用量

区域	内存用量（**16** 位字）	
	RAM	闪存
库接口 (.ebss)	0x018C	×
库 (.ebss)	0x0800	×
代码 (.text)	0x1F31	×
IQmath (.text)	0x0064	×

表 8-14 汇总了每个函数的所有性能数据，其中的用户代码均以 InstaSPIN 库的最小执行方式从 RAM 中加载并执行。 请注意，CTRL_run 从 ROM 和 RAM 中均可执行。 这是因为 CTRL_run 的某些函数调用需要调用估算器。 例如，EST_run 函数调用从 CTRL_run 中执行，因此将从 ROM 中执行。 同样，CTRL_setup 的某些代码会调用某些 InstaSPIN 状态机代码，由于其中包含与 FAST 估算器的一些交互，因此需要从 ROM 中执行。 与 RAM 中的完全执行相比二者间的代码差异在于，最小执行的代码中添加了额外的偏移对象并内联了全部 FOC 代码。

表 **8-14.** 在 **RAM** 中进行最小执行

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
HAL_acqAdcInt	23	23	23	×	✓	×

表 **8-14.** 在 **RAM** 中进行最小执行 **(continued)**

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
HAL_readAdcData	106	106	106	×	✓	×
Ctrl_run						
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	2361	2372	2454			
CTRL vs EST = 2	1171	1777	2454			
CTRL vs EST = 3	1171	1579	2454			
ISR vs CTRL = 2，CTRL vs EST = 1	59	1215	2454			
CTRL vs EST = 2	59	918	2454			
CTRL vs EST = 3	59	819	2454			
ISR vs CTRL = 3，CTRL vs EST = 1	59	830	2454			
CTRL vs EST = 2	59	631	2454			
CTRL vs EST = 3	59	565	2454	✓	×	×
Rs 在线启用，ISR vs CTRL = 1，CTRL vs EST = 1	2825	2840	2925			
CTRL vs EST = 2	1171	2012	2925			
CTRL vs EST = 3	1171	1736	2925			
ISR vs CTRL = 2，CTRL vs EST = 1	59	1450	2925			
CTRL vs EST = 2	59	1035	2925			
CTRL vs EST = 3	59	897	2925			
ISR vs CTRL = 3，CTRL vs EST = 1	59	986	2925			
CTRL vs EST = 2	59	710	2925			
CTRL vs EST = 3	59	618	2925			
HAL_writePwmData	62	62	62	×	✓	×
CTRL_setup	36	50	178	✓	✓	×

8.5.1.1.2 闪存中的用户代码

8.5.1.1.2.1 完全执行

表 **8-15.** 在闪存中进行完全执行的内存用量

区域	内存用量（**16** 位字）	
	RAM	闪存
库接口 (.ebss)	0x018C	×
库 (.ebss)	0x0800	×
代码 (.text)	0x001E	0x186C
IQmath (.text)	×	0x0014

表 8-16 显示了采用 InstaSPIN 完全执行且用户代码加载到闪存中时的资源利用率。 代码区域现添加了几个用于初始化闪存的函数，它们需要从 RAM 中运行（从 RAM 中加载）。 因此，新代码会从 RAM 中执行，而不是全部从闪存中执行。 代码从闪存中执行时，其中添加了一个 memCopy 函数，这也会增加代码区域。

表 **8-16.** 在闪存中进行完全执行

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
HAL_acqAdcInt	25	25	25	×	×	✓
HAL_readAdcData	108	108	108	×	×	✓

表 **8-16.** 在闪存中进行完全执行 **(continued)**

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
Ctrl_run						
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	2345	2355	2425			
CTRL vs EST = 2	1154	1760	2425			
CTRL vs EST = 3	1154	1562	2425			
ISR vs CTRL = 2，CTRL vs EST = 1	58	1207	2425			
CTRL vs EST = 2	58	909	2425			
CTRL vs EST = 3	58	810	2425			
ISR vs CTRL = 3，CTRL vs EST = 1	58	824	2425			
CTRL vs EST = 2	58	626	2425			
CTRL vs EST = 3	58	560	2425	✓	×	×
Rs 在线启用，ISR vs CTRL = 1，CTRL vs EST = 1	2807	2821	2894			
CTRL vs EST = 2	1154	1993	2894			
CTRL vs EST = 3	1154	1717	2894			
ISR vs CTRL = 2，CTRL vs EST = 1	58	1439	2894			
CTRL vs EST = 2	58	1025	2894			
CTRL vs EST = 3	58	887	2894			
ISR vs CTRL = 3，CTRL vs EST = 1	58	979	2894			
CTRL vs EST = 2	58	702	2894			
CTRL vs EST = 3	58	610	2894			
HAL_writePwmData	64	64	64	×	×	✓
CTRL_setup	37	51	178	✓	×	×

8.5.1.1.2.2 最小执行

表 **8-17.** 在闪存中进行最小执行的内存用量

区域	内存用量（**16** 位字）	
	RAM	闪存
库接口 (.ebss)	0x018C	×
库 (.ebss)	0x0800	×
代码 (.text)	0x001E	0x1F2D
IQmath (.text)	×	0x0064

表 8-18 显示了采用 InstaSPIN 最小执行且用户代码加载到闪存中时的资源利用率。

表 **8-18.** 在闪存中进行最小执行

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
HAL_acqAdcInt	25	25	25	×	×	✓
HAL_readAdcData	108	108	108	×	×	✓

表 **8-18.** 在闪存中进行最小执行 **(continued)**

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
Ctrl_run						
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	2447	2459	2544			
CTRL vs EST = 2	1257	1864	2544			
CTRL vs EST = 3	1257	1665	2544			
ISR vs CTRL = 2，CTRL vs EST = 1	71	1265	2544			
CTRL vs EST = 2	71	967	2544			
CTRL vs EST = 3	71	868	2544			
ISR vs CTRL = 3，CTRL vs EST = 1	71	867	2544			
CTRL vs EST = 2	71	668	2544			
CTRL vs EST = 3	71	602	2544	✓	×	✓
Rs 在线启用，ISR vs CTRL = 1，CTRL vs EST = 1	2911	2927	3015			
CTRL vs EST = 2	1258	2098	3015			
CTRL vs EST = 3	1258	1822	3015			
ISR vs CTRL = 2，CTRL vs EST = 1	71	1499	3015			
CTRL vs EST = 2	71	1084	3015			
CTRL vs EST = 3	71	946	3015			
ISR vs CTRL = 3，CTRL vs EST = 1	71	1022	3015			
CTRL vs EST = 2	71	746	3015			
CTRL vs EST = 3	71	654	3015			
HAL_writePwmData	64	64	64	×	×	✓
CTRL_setup	46	60	188	✓	×	✓

8.5.1.2 PWM = 10kHz 时的 CPU 负载

表 **8-19.** 在 **ROM** 和闪存中进行完全执行

F2806xF CPU = 90MHz 可用 **MIPs = 90MIPs** **PWM = 10kHz**	**CPU** 利用率 **[%]**	使用的 **MIPS [MIPS]**	可用的 **MIPS [MIPS]**
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	28.86	25.97	64.03
CTRL vs EST = 2	22.24	20.02	69.98
CTRL vs EST = 3	20.04	18.04	71.96
ISR vs CTRL = 2，CTRL vs EST = 1	16.1	14.49	75.51
CTRL vs EST = 2	12.79	11.51	78.49
CTRL vs EST = 3	11.69	10.52	79.48
ISR vs CTRL = 3，CTRL vs EST = 1	11.84	10.66	79.34
CTRL vs EST = 2	9.64	8.68	81.32
CTRL vs EST = 3	8.91	8.02	81.98
Rs 在线启用，ISR vs CTRL = 1，CTRL vs EST = 1	34.03	30.63	59.37
CTRL vs EST = 2	24.83	22.35	67.65
CTRL vs EST = 3	21.77	19.59	70.41
ISR vs CTRL = 2，CTRL vs EST = 1	18.68	16.81	73.19
CTRL vs EST = 2	14.08	12.67	77.33
CTRL vs EST = 3	12.54	11.29	78.71
ISR vs CTRL = 3，CTRL vs EST = 1	13.57	12.21	77.79
CTRL vs EST = 2	10.49	9.44	80.56
CTRL vs EST = 3	9.47	8.52	81.48

表 **8-20.** 在 **ROM** 和闪存中进行最小执行

F2806xF CPU = 90MHz 可用 **MIPs = 90MIPs** **PWM = 10kHz**		**CPU** 利用率 **[%]**	使用的 **MIPS [MIPS]**	可用的 **MIPS [MIPS]**
Rs 在线禁用，	ISR vs CTRL = 1，CTRL vs EST = 1	30.01	27.01	62.99
	CTRL vs EST = 2	23.4	21.06	68.94
	CTRL vs EST = 3	21.19	19.07	70.93
	ISR vs CTRL = 2，CTRL vs EST = 1	16.74	15.07	74.93
	CTRL vs EST = 2	13.43	12.09	77.91
	CTRL vs EST = 3	12.33	11.1	78.9
	ISR vs CTRL = 3，CTRL vs EST = 1	12.32	11.09	78.91
	CTRL vs EST = 2	10.11	9.1	80.9
	CTRL vs EST = 3	9.38	8.44	81.56
Rs 在线启用，	ISR vs CTRL = 1，CTRL vs EST = 1	35.21	31.69	58.31
	CTRL vs EST = 2	26	23.4	66.6
	CTRL vs EST = 3	22.93	20.64	69.36
	ISR vs CTRL = 2，CTRL vs EST = 1	19.34	17.41	72.59
	CTRL vs EST = 2	14.73	13.26	76.74
	CTRL vs EST = 3	13.2	11.88	78.12
	ISR vs CTRL = 3，CTRL vs EST = 1	14.04	12.64	77.36
	CTRL vs EST = 2	10.98	9.88	80.12
	CTRL vs EST = 3	9.96	8.96	81.04

8.5.1.3 PWM = 20kHz 时的 CPU 负载

表 **8-21.** 在 **ROM** 和闪存中进行完全执行

F2806xF CPU = 90MHz 可用 **MIPs = 90MIPs** **PWM = 20kHz**		**CPU** 利用率 **[%]**	使用的 **MIPS [MIPS]**	可用的 **MIPS [MIPS]**
Rs 在线禁用，	ISR vs CTRL = 1，CTRL vs EST = 1	57.71	51.94	38.06
	CTRL vs EST = 2	44.49	40.04	49.96
	CTRL vs EST = 3	40.09	36.08	53.92
	ISR vs CTRL = 2，CTRL vs EST = 1	32.2	28.98	61.02
	CTRL vs EST = 2	25.58	23.02	66.98
	CTRL vs EST = 3	23.38	21.04	68.96
	ISR vs CTRL = 3，CTRL vs EST = 1	23.69	21.32	68.68
	CTRL vs EST = 2	19.29	17.36	72.64
	CTRL vs EST = 3	17.82	16.04	73.96
Rs 在线启用，	ISR vs CTRL = 1，CTRL vs EST = 1	68.07	61.26	28.74
	CTRL vs EST = 2	49.67	44.7	45.3
	CTRL vs EST = 3	43.53	39.18	50.82
	ISR vs CTRL = 2，CTRL vs EST = 1	37.36	33.62	56.38
	CTRL vs EST = 2	28.16	25.34	64.66
	CTRL vs EST = 3	25.09	22.58	67.42
	ISR vs CTRL = 3，CTRL vs EST = 1	27.13	24.42	65.58
	CTRL vs EST = 2	20.98	18.88	71.12
	CTRL vs EST = 3	18.93	17.04	72.96

表 **8-22.** 在 **ROM** 和闪存中进行最小执行

F2806xF CPU = 90MHz 可用 MIPs = 90MIPs PWM = 20kHz		CPU 利用率 [%]	使用的 MIPS [MIPS]	可用的 MIPS [MIPS]
Rs 在线禁用，ISR vs CTRL = 1，	CTRL vs EST = 1	60.02	54.02	35.98
	CTRL vs EST = 2	46.8	42.12	47.88
	CTRL vs EST = 3	42.38	38.14	51.86
ISR vs CTRL = 2，	CTRL vs EST = 1	33.49	30.14	59.86
	CTRL vs EST = 2	26.87	24.18	65.82
	CTRL vs EST = 3	24.67	22.2	67.8
ISR vs CTRL = 3，	CTRL vs EST = 1	24.64	22.18	67.82
	CTRL vs EST = 2	20.22	18.2	71.8
	CTRL vs EST = 3	18.76	16.88	73.12
Rs 在线启用，ISR vs CTRL = 1，	CTRL vs EST = 1	70.42	63.38	26.62
	CTRL vs EST = 2	52	46.8	43.2
	CTRL vs EST = 3	45.87	41.28	48.72
ISR vs CTRL = 2，	CTRL vs EST = 1	38.69	34.82	55.18
	CTRL vs EST = 2	29.47	26.52	63.48
	CTRL vs EST = 3	26.4	23.76	66.24
ISR vs CTRL = 3，	CTRL vs EST = 1	28.09	25.28	64.72
	CTRL vs EST = 2	21.96	19.76	70.24
	CTRL vs EST = 3	19.91	17.92	72.08

8.5.1.4 CPU 负载示例

从上述章节介绍的表格可以看出，通过累加周期数并计算使用百分比可以得出 CPU 使用率。

8.5.1.4.1 示例 1

考虑以下情况：

- CPU 时钟 = 90MHz
- 可用 MIPS = 90MIPS
- PWM 频率 = 10kHz
- InstaSPIN 执行：
  - 完全执行，ROM 中的库和 RAM 中的用户代码 (8.2.1 节)
  - Rs 在线禁用
  - ISR vs CTRL = 1
  - CTRL vs EST = 1

计算中断所用 CPU 的百分比，其中：

最大周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup
InstaSPIN 所用 CPU 的最大百分比 =
100% * ((最大周期数) / 90MHz) * PWM 频率 =
100% * ((23 + 106 + 2425 + 62 + 178) / 90MHz) * 10kHz =
31.04%

InstaSPIN 所用 CPU 的平均百分比 =
100% * ((平均周期数) / 90MHz) * PWM 频率 =

100% * ((23 + 106 + 2355 + 62 + 51) / 90MHz) * 10kHz =
28.86%

另一个实用的计算是应用程序所用的 MIPS 数。 该值的过程计算如下：

InstaSPIN 所用的平均 MIPS =
（InstaSPIN 所用 CPU 的平均百分比/100%） * 可用 MIPS =
(28.86% / 100%) * 90MIPS =
25.97MIPS

随后，可以计算出其它任务的平均可用 MIPS：

用户平均可用 MIPS =
可用的总 MIPS - InstaSPIN 所用的平均 MIPS =
90MIPS – 25.97MIPS =
64.03MIPS 可用于其它任务

8.5.1.4.2 示例 2

通常需要启用 Rs 在线同时增大 PWM 频率。 要释放 CPU 带宽，ISR vs CTRL 设置为 2。

考虑以下情况：

- CPU 时钟 = 90MHz
- 可用 MIPS = 90MIPS
- PWM 频率 = 20kHz
- InstaSPIN 执行：
  - 最小执行，ROM 中的库和 RAM 中的用户代码 (8.2.2 节)
  - Rs 在线启用
  - ISR vs CTRL = 2
  - CTRL vs EST = 1

首先，需要计算 InstaSPIN 在给定条件下所用 CPU 的平均百分比：

InstaSPIN 所用 CPU 的平均百分比 =
100% * ((平均周期数) / 90MHz) * PWM 频率 =
100% * ((23 + 106 + 1450 + 62 + 50) / 90MHz) * 20kHz =
37.58%

其次，计算 InstaSPIN 在给定条件下使用的平均 MIPS 数：

InstaSPIN 所用的平均 MIPS =
（InstaSPIN 所用 CPU 的平均百分比/100%） * 可用 MIPS =
(37.58% / 100 %) * 90MIPS =
33.82MIPS

随后，可以计算出其它任务的平均可用 MIPS：

用户平均可用 MIPS =
可用的总 MIPS - InstaSPIN 所用的平均 MIPS =
90MIPS – 33.82MIPS =
56.18MIPS 可用于其它任务

8.5.2 F2806xM 器件

8.5.2.1 CPU 周期数

InstaSPIN-MOTION 将 InstaSPIN-FOC 与 LineStream Technologies 开发的 SpinTAC 运动控制套件的功能结合在一起。 通过 InstaSPIN-FOC 周期数加上后续章节提供的 SpinTAC 周期数并计算使用百分比可以得出 CPU 使用率。 后续章节将提供具体示例。

8.5.2.1.1 RAM 执行 - SpinTAC 库和用户代码

表 8-23 汇总了每个函数的所有性能数据，其中的 SpinTAC 库从 RAM 中加载并执行。 请注意，各个函数均可调用 ROM 存储器来运行核心 SpinTAC 函数。 各个组件的典型情况用**粗体**突出显示。

表 **8-23. F2806xM 器件在 RAM 中执行库时 SpinTAC 的 CPU 周期利用率**

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	ROM	RAM	闪存
STVELCTL_run（速度控制）						
RES = 0, ENB = 0	158	158	158			
RES = 0，ENB = 1	**573**	**573**	**573**			
ENB = 1 后的首次调用	1010	1010	1010			
	786	786	786	✓	✓	×
改变惯性参数	786	786	786			
RES = 1，ENB = 1	289	289	289			
STVELMOVE_run（速度移动）						
RES = 0，ENB = 0	202	202	202			
stcurve RES = 0，ENB = 1	**675**	**704**	**1346**			
scurve RES = 0，ENB = 1	**638**	**669**	**1312**	✓	✓	×
trap RES = 0，ENB = 1	**509**	**576**	**1039**			
RES = 1，ENB = 1	421	421	421			
STVELPLAN_run（速度规划）						
RES = 1，ENB = 0	159	159	159			
RES = 0，ENB = 1	**169**	**169**	**169**			
ENB = 1 后首次调用	285	285	285			
STAY FSM 状态	194	194	194			
条件 FSM 状态 必须对每个状态完成计算	374（固定）+ 274 * 转换数 + 334 * EXIT（退出）操作数			✓	✓	×
转换 FSM 状态 必须对每个状态完成计算	229（固定）+ 378 * ENTER（进入）操作数					
STVELPLAN_runTick（速度规划）	58	78	78			
STVELID_run（速度识别）						
RES = 0，ENB = 0	142	142	142			
RES = 0，ENB = 1	**332**	**341**	**658**			
ENB = 1 后的首次调用	1063	1063	1063	✓	✓	×
RES = 1，ENB = 1	249	249	249			
STPOSCONV_run（位置转换器）						
RES = 0，ENB = 0	110	110	110			
RES = 0，ENB = 1	**322**	**341**	**343**			
ENB = 1 后的首次调用	1060	1060	1060	✓	✓	×
RES = 1，ENB = 1	118	118	118			

表 8-23. F2806xM 器件在 RAM 中执行库时 SpinTAC 的 CPU 周期利用率 (continued)

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	ROM	RAM	闪存
STPOSCTL_run（位置控制）						
RES = 0，ENB =0	166	166	166			
RES = 0，ENB = 1	**1120**	**1125**	**1140**			
ENB = 1 后的首次调用	1903	1903	1903			
改变带宽参数	1611	1611	1611	✓	✓	×
改变惯性参数	1611	1611	1611			
RES = 1，ENB = 1	385	385	385			
STPOSMOVE_run（位置移动）						
RES = 0，ENB = 0	406	406	406			
stcurve RES = 0，ENB = 1	**616**	**1383**	**2733**			
ENB = 1 后的首次调用	1270	1377	2368			
scurve RES = 0，ENB = 1	**616**	**1333**	**2561**			
ENB = 1 后的首次调用	1219	1337	2324	✓	✓	×
trap RES = 0，ENB = 1	**616**	**1253**	**2501**			
ENB = 1 后的首次调用	1319	1608	2049			
RES = 1，ENB = 1	877	877	877			
STPOSPLAN_run（位置规划）						
RES = 1，ENB = 0	166	166	166			
RES = 0，ENB = 1	201	201	201			
ENB = 1 后首次调用	325	325	325			
STAY FSM 状态	209	209	209			
条件 FSM 状态 必须对每个状态完成计算	436（固定）+ 276 * 转换数 + 334 * EXIT（退出）操作数			✓	✓	×
转换 FSM 状态 必须对每个状态完成计算	245（固定）+ 378 * ENTER（进入）操作数					
STPOSPLAN_runTick（位置规划）	58	78	78			

8.5.2.1.2 闪存执行 - SpinTAC 库和用户代码

表 8-24 汇总了每个函数的所有性能数据，其中的 SpinTAC 库从闪存中加载并执行。 InstaSPIN-FOC 处于完全执行模式。

表 8-24. F2806xM 器件在闪存中执行库时 SpinTAC 的 CPU 周期利用率

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	ROM	RAM	闪存
STVELCTL_run（速度控制）						
RES = ， ENB = 0	216	216	216			
RES = 0，ENB = 1	**672**	**672**	**672**			
ENB = 1 后的首次调用	1183	1183	1183			
改变带宽	925	925	925	✓	×	✓
改变惯性参数	925	925	925			
RES = 1，ENB = 1	396	396	396			

表 8-24. F2806xM 器件在闪存中执行库时 SpinTAC 的 CPU 周期利用率 (continued)

函数名称		CPU 周期数			执行位置		
		最小值	平均值	最大值	ROM	RAM	闪存
STVELMOVE_run（速度移动）							
	RES = 0，ENB = 0	258	258	258			
stcurve	**RES = 0，ENB = 1**	**780**	**816**	**1625**			
scurve	**RES = 0，ENB = 1**	**743**	**781**	**1589**	✓	×	✓
trap	**RES = 0，ENB = 1**	**616**	**700**	**1309**			
	RES = 1，ENB = 1	554	554	554			
STVELPLAN_run（速度规划）							
	RES = 1，ENB = 0	219	219	219			
	RES = 0，ENB = 1	**266**	**266**	**266**			
	ENB = 1 后的首次调用	393	393	393			
	STAY FSM 状态	280	280	280			
	转换 FSM 状态 必须对每个状态完成计算	488（固定）+ 368 * 转换数 + 440 * EXIT（退出）操作数			✓	×	✓
	条件 FSM 状态 必须对每个状态完成计算	337（固定）+ 503 * ENTER（进入）操作数					
STVELPLAN_runTick（速度规划）		91	119	119			
STVELID_run（速度识别）							
	RES = 1，ENB = 0	198	198	198			
	RES = 0，ENB = 1	**311**	**332**	**822**			
	ENB = 1 后的首次调用	1366	1366	1366	✓	×	✓
	RES = 1，ENB = 1	338	338	338			
STPOSCONV_run（位置转换器）							
	RES = 0，ENB = 0	145	145	145			
	RES = 0，ENB = 1	**443**	**448**	**450**			
	ENB = 1 后的首次调用	1372	1372	1372	✓	×	✓
	RES = 1，ENB = 1	170	170	170			
STPOSCTL_run（位置控制）							
	RES = 0，ENB = 0	246	246	246			
	RES = 0，ENB = 1	**1311**	**1316**	**1326**			
	ENB = 1 后的首次调用	2236	2236	2236			
	改变带宽参数	1909	1909	1909	✓	×	✓
	改变惯性参数	1909	1909	1909			
	RES = 1，ENB = 1	509	509	509			
STPOSMOVE_run（位置移动）							
	RES = 0，ENB = 0	520	520	520			
stcurve	**RES = 0，ENB = 1**	**790**	**1611**	**3630**			
	ENB = 1 后的首次调用	1467	1588	2778			
scurve	**RES = 0，ENB = 1**	**790**	**1564**	**3205**			
	ENB = 1 后的首次调用	1415	1551	2734	✓	×	✓
trap	**RES = 0，ENB = 1**	**790**	**1501**	**3130**			
	ENB = 1 后的首次调用	1540	1903	2438			
	RES = 1，ENB = 1	1100	1100	1100			

表 8-24. F2806xM 器件在闪存中执行库时 SpinTAC 的 CPU 周期利用率 (continued)

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	ROM	RAM	闪存
STPOSPLAN_run（位置规划）						
RES = 1，ENB = 0	229	229	229			
RES = 0，ENB = 1	297	297	297			
ENB = 1 后的首次调用	450	450	450			
STAY FSM 状态	297	297	297	✓	×	✓
条件 FSM 状态 必须对每个状态完成计算	548（固定） + 363 * 转换数 + 450 * EXIT（退出）操作数					
转换 FSM 状态 必须对每个状态完成计算	345（固定）+ 508 * ENTER（进入）操作数					
STPOSPLAN_runTick（位置规划）	86	115	115	✓	×	✓

8.5.2.2 CPU 负载示例

8.5.2.2.1 示例 1

考虑以下情况：

- CPU 时钟 = 90MHz
- 可用 MIPS = 90MIPS
- PWM 频率 = 10kHz
- InstaSPIN 执行：
  - InstaSPIN-FOC：完全执行，ROM 中的库和 RAM 中的用户代码 (8.2.1 节)
  - SpinTAC 库：速度控制。 ROM 中的库和 RAM 中的用户库代码。
  - Rs 在线禁用
  - ISR vs CTRL = 1
  - CTRL vs SPEED = 10

计算中断所用 CPU 的百分比，其中：

最小周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup

最大周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup + STVELCTL_run

最小周期数 = 23 + 106 + 2355 + 62 + 51 = 2597 个周期

最大周期数 = 23 + 106 + 2355 + 62 + 51 + 573 = 3170 个周期

在每一毫秒内，最小周期数出现 9 次，最大周期数出现一次。

1 毫秒内的周期数 = 2597 * 9 + 3170 * 1 = 26543 个周期

当前 CPU 使用率为

100% * (26543 / 90MHz) * (10kHz / 10) = **29.49%**

另一个实用的计算是 InstaSPIN 所用的 MIPS 数。 该值的过程计算如下：

InstaSPIN 所用的平均 MIPS =

（InstaSPIN 所用 CPU 的平均百分比/100%） * 可用 MIPS =

(29.49 % / 100 %) * 90MIPS =

26.54MIPS

随后，可以计算出其它任务的平均可用 MIPS：

用户平均可用 MIPS =

可用的总 MIPS - InstaSPIN(FOC + MOTION) 所用的平均 MIPS =
90MIPS – 26.54MIPS =
63.46MIPS 可用于其它任务

8.5.2.2.2 示例 2

例如，考虑以下情况：

- CPU 时钟 = 90MHz
- 可用 MIPS = 90MIPS
- PWM 频率 = 10kHz
- InstaSPIN 执行：
  - InstaSPIN-FOC：完全执行，ROM 中的库和 RAM 中的用户代码 (8.2.1 节)
  - SpinTAC 库：速度控制 + 速度移动 (stcurve)。 ROM 中的库和 RAM 中的用户库代码。
  - Rs 在线禁用
  - ISR vs CTRL = 1
  - CTRL vs SPEED = 10

计算中断所用 CPU 的百分比，其中：

最小周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup
最大周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup + STVELCTL_run
最小周期数 = 23 + 106 + 2355 + 62 + 51 = 2597 个周期
最大周期数 = 23 + 106 + 2355 + 62 + 51 + 573 + 704 = 3874 个周期
在每一毫秒内，最小周期数出现 9 次，最大周期数出现一次。
1 毫秒内的周期数 = 2597 * 9 + 3874 * 1 = 27247 个周期
当前 CPU 使用率为
100% * (27247 / 90MHz) * (10kHz / 10) = **30.27%**

另一个实用的计算是应用程序所用的 MIPS 数。 该值的过程计算如下：

InstaSPIN 所用的平均 MIPS =
（InstaSPIN 所用 CPU 的平均百分比/100%） * 可用 MIPS =
(30.27 % / 100 %) * 90MIPS =
27.24MIPS

随后，可以计算出其它任务的平均可用 MIPS：

用户平均可用 MIPS =
可用的总 MIPS - InstaSPIN(FOC + MOTION) 所用的平均 MIPS =
90MIPS – 27.24MIPS =
62.76MIPS 可用于其它任务

8.5.2.2.3 示例 3

例如，考虑以下情况：

- CPU 时钟 = 90MHz
- 可用 MIPS = 90MIPS
- PWM 频率 = 10kHz
- InstaSPIN 执行：
  - InstaSPIN-FOC：完全执行，ROM 中的库和 RAM 中的用户代码 (8.2.1 节)

– SpinTAC 库：速度控制 + 速度移动 (stcurve) + 速度规划。 ROM 中的库和 RAM 中的用户库代码。
– Rs 在线禁用
– ISR vs CTRL = 1
– CTRL vs SPEED = 10

SpinTAC 速度规划可用于生成一个含 4 种状态的机械运动序列（请参见图 8-9）。 请注意，此示例中的规划函数 STVELPLAN_run 和 STVELPLAN_runTick 均从 ISR 运行。 在最后的执行中，STVELPLAN_run 可以从较慢的 ISR 或者后台循环中运行。

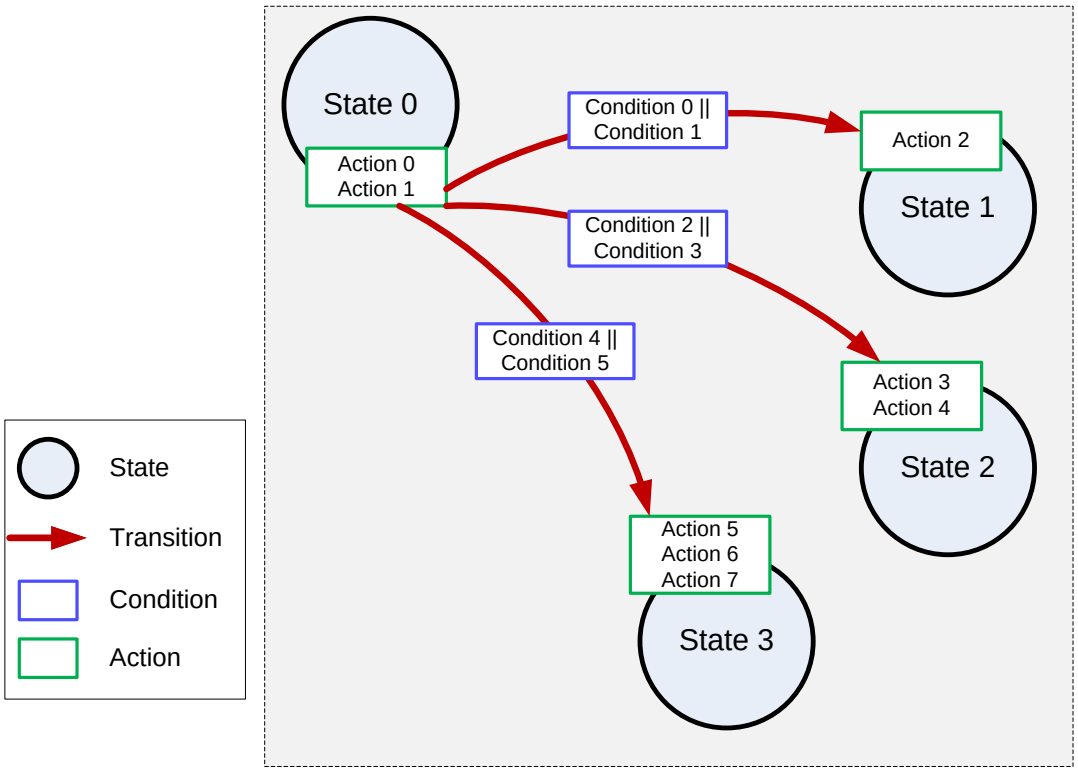


图 8-9. SpinTAC 速度规划示例

STVELPLAN_run 的最大周期数可能受条件 FSM 状态或转换 FSM 状态影响，但不可能同时受两者影响。

1. 条件 FSM 状态中的最大周期数取决于：
   - 从给定状态开始的转换数。
   - 针对转换检查的条件数量（最差情况为两个条件）
   - 这些条件会对比一个变量和一个值或者对比一个变量和另一个变量（最差情况）
   - 为此状态配置的 EXIT（退出）操作数。

   如果代码在 RAM 中运行，则最大周期数 = 374（条件 FSM 状态的固定周期数）+（转换数 * 274）+（EXIT（退出）操作数 * 334）。

   图 8-9 中的示例规划配置了从状态 0 开始的三次转换。所有转换均会检查对比变量的各个条件（最差情况），同时为此状态配置了 2 个 EXIT（退出）操作。 因此，当执行最后一次转换时，最大周期数 = 374 + (3 * 274) + (2 * 334) = **1864**。
2. 转换 FSM 状态中的最大周期数取决于为此状态配置的 ENTER（进入）操作数。

   如果代码在 RAM 中执行，则最大周期数 = 229（转换 FSM 状态的固定周期数） + ENTER（进入）操作数 * 378。

以上示例显示状态 3 具有 3 个 ENTER（进入）操作。 因此，转换 FSM 状态的最大周期数 = 229 + (3 * 378) = **1368**。

对比这两种可以产生 STVELPLAN_run 最大周期数的情况。 在此示例中，由条件 FSM 状态所决定的最大周期数为 1330。 当计算中断所用 CPU 的百分比时，使用此值作为最差情况。

计算中断所用 CPU 的百分比，其中：

最小周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup

最大周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup + STVELCTL_run + STVELPLAN_runTick + STVELPLAN_run

最小周期数 = 23 + 106 + 2355 + 62 + 51 = 2597 个周期

最大周期数 = 23 + 106 + 2355 + 62 + 51 + 573 + 704 + 78 + 1864 = 5738 个周期

在每一毫秒内，最小周期数出现 9 次，最大周期数出现一次。

1 毫秒内的周期数 = 2597 * 9 + 5738 * 1 = 29111 个周期

当前 CPU 使用率为

100% * (29111 / 90MHz) * (10kHz / 10) = **32.35%**

在以上计算中，SpinTAC 速度规划的主要组件在 ISR 运行。 当从后台调用 SpinTAC 速度规划时，中断所用 CPU 的百分比计算方法如下：

最小周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup + STVELCTL_run + STVELMOVE_run + STVELPLAN_runTick

最大周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup + STVELCTL_run + STVELPLAN_runTick + STVELPLAN_run

最小周期数 = 23 + 106 + 2355 + 62 + 51 = 2597 个周期

最大周期数 = 23 + 106 + 2355 + 62 + 51 + 573 + 704 + 78 = 3952 个周期

在每一毫秒内，最小周期数出现 9 次，最大周期数出现一次。

1 毫秒内的周期数 = 2597 * 9 + 5738 * 1 = 27325 个周期

当前 CPU 使用率为

100% * (27325 / 90MHz) * (10kHz / 10) = **30.36%**

8.5.2.2.4 示例 4（SpinTAC 位置）

例如，考虑以下情况：

- CPU 时钟 = 90MHz
- 可用 MIPS = 90MIPS
- PWM 频率 = 10kHz
- InstaSPIN 执行：
  - InstaSPIN-FOC：完全执行，ROM 中的库和 RAM 中的用户代码 (8.2.1 节)
  - SpinTAC 库：位置转换器 + 位置控制 + 位置移动 (stcurve)。 ROM 中的库和 RAM 中的用户库代码。
  - Rs 在线禁用
  - ISR vs CTRL = 1
  - CTRL vs SPEED = 10

计算中断所用 CPU 的百分比，其中：

最小周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup

最大周期数 = HAL_acqAdcInt + HAL_readAdcData + Ctrl_run + HAL_writePwmData + Ctrl_setup + STPOSCONV_run + STPOSCTL_run + STPOSMOVE_run

最小周期数 = 23 + 106 + 2355 + 62 + 51 = 2597 个周期

最大周期数 = 23 + 106 + 2355 + 62 + 51 + 341 + 1125 + 1383 = 5446 个周期

在每一毫秒内，最小周期数出现 9 次，最大周期数出现一次。

1 毫秒内的周期数 = 2597 * 9 + 5446 * 1 = 28819 个周期
当前 CPU 使用率为
100% * (28819 / 90MHz) * (10kHz / 10) = **32.02%**

另一个实用的计算是应用程序所用的 MIPS 数。 该值的过程计算如下：
InstaSPIN 所用的平均 MIPS =
（InstaSPIN 所用 CPU 的平均百分比/100%） * 可用 MIPS =
(32.02% / 100%) * 90MIPS =
28.82MIPS

随后，可以计算出其它任务的平均可用 MIPS：
用户平均可用 MIPS =
可用的总 MIPS - InstaSPIN(FOC + MOTION) 所用的平均 MIPS =
90MIPS – 28.82MIPS =
61.18MIPS 可用于其它任务

8.5.3 F805xF 器件

8.5.3.1 CPU 周期数

F2805xF 周期数与 F2806xF 周期数基本相同。

8.5.3.2 PWM = 10kHz 时的 CPU 负载

表 **8-25.** 在 **ROM** 和闪存中进行完全执行

F2805xF CPU = 60MHz **可用 MIPs = 60MIPs** **PWM = 10kHz**		**CPU** 利用率 **[%]**	使用的 **MIPS [MIPS]**	可用的 **MIPS [MIPS]**
R_s 在线被禁用，ISR vs CTRL = 1，	CTRL vs EST = 1	43.28	25.97	34.03
	CTRL vs EST = 2	33.37	20.02	39.98
	CTRL vs EST = 3	30.07	18.04	41.96
ISR vs CTRL = 2，	CTRL vs EST = 1	24.15	14.49	45.51
	CTRL vs EST = 2	19.18	11.51	48.49
	CTRL vs EST = 3	17.53	10.52	49.48
ISR vs CTRL = 3，	CTRL vs EST = 1	17.77	10.66	49.34
	CTRL vs EST = 2	14.47	8.68	51.32
	CTRL vs EST = 3	13.37	8.02	51.98
R_s 在线启用，ISR vs CTRL = 1，	CTRL vs EST = 1	51.05	30.63	29.37
	CTRL vs EST = 2	37.25	22.35	37.65
	CTRL vs EST = 3	32.65	19.59	40.41
ISR vs CTRL = 2，	CTRL vs EST = 1	28.02	16.81	43.19
	CTRL vs EST = 2	21.12	12.67	47.33
	CTRL vs EST = 3	18.82	11.29	48.71
ISR vs CTRL = 3，	CTRL vs EST = 1	20.35	12.21	47.79
	CTRL vs EST = 2	15.73	9.44	50.56
	CTRL vs EST = 3	14.20	8.52	51.48

8.5.3.3 PWM = 20kHz 时的 CPU 负载

表 **8-26.** 在 **ROM** 和闪存中进行完全执行

F2805xF CPU = 60MHz 可用 **MIPs = 60MIPs** **PWM = 20kHz**	**CPU** 利用率 **[%]**	使用的 **MIPS [MIPS]**	可用的 **MIPS [MIPS]**
R_s 在线被禁用，ISR vs CTRL = 1，CTRL vs EST = 1	86.57	51.94	8.06
CTRL vs EST = 2	66.73	40.04	19.96
CTRL vs EST = 3	60.13	36.08	23.92
ISR vs CTRL = 2，CTRL vs EST = 1	48.3	28.98	31.02
CTRL vs EST = 2	38.37	23.02	36.98
CTRL vs EST = 3	35.07	21.04	38.96
ISR vs CTRL = 3，CTRL vs EST = 1	35.53	21.32	38.68
CTRL vs EST = 2	28.93	17.36	42.64
CTRL vs EST = 3	26.73	16.04	43.96
R_s 在线启用，ISR vs CTRL = 1			
CTRL vs EST = 2	74.5	44.7	15.3
CTRL vs EST = 3	65.3	39.18	20.82
ISR vs CTRL = 2，CTRL vs EST = 1	56.03	33.62	26.38
CTRL vs EST = 2	42.23	25.34	34.66
CTRL vs EST = 3	37.63	22.58	37.42
ISR vs CTRL = 3，CTRL vs EST = 1	40.7	24.42	35.58
CTRL vs EST = 2	31.47	18.88	41.12
CTRL vs EST = 3	28.4	17.04	42.96

8.5.4 F2805xM 器件

8.5.4.1 CPU 周期数

8.5.4.1.1 闪存执行 - SpinTAC 库和用户代码

表 **8-27. F2805xM** 器件在闪存中执行库时 **SpinTAC** 的 **CPU** 周期利用率

函数名称		**CPU** 周期数			执行位置		
		最小值	平均值	最大值	**ROM**	**RAM**	闪存
STVELCTL_run（速度控制）							
	RES = 1，ENB = 0	189	189	189			
	RES = 0，ENB = 1	**614**	**614**	**614**			
	ENB = 1 后的首次调用	1077	1077	1077	✓	×	✓
	改变带宽	842	842	842			
	改变惯性参数	842	842	842			
	RES = 1，ENB = 1	347	347	347			
STVELMOVE_run（速度移动）							
	RES = 1，ENB = 0	220	220	220			
stcurve	**RES = 0，ENB = 1**	**724**	**759**	**1468**			
scurve	**RES = 0，ENB = 1**	**687**	**724**	**1435**	✓	×	✓
trap	**RES = 0，ENB = 1**	**561**	**636**	**1167**			
	RES = 1，ENB = 1	494	494	494			

表 8-27. F2805xM 器件在闪存中执行库时 SpinTAC 的 CPU 周期利用率 (continued)

函数名称	CPU 周期数			执行位置		
	最小值	平均值	最大值	ROM	RAM	闪存
STVELPLAN_run（速度规划）						
RES = 1，ENB = 0	183	183	183			
RES = 0，ENB = 1	**238**	**238**	**238**			
ENB = 1 后的首次调用	333	333	333			
STAY FSM 状态	238	238	238			
转换 FSM 状态 必须对每个状态完成计算	436（固定）+ 320 * 转换数 + 388 * EXIT（退出）操作数			✓	×	✓
条件 FSM 状态 必须对每个状态完成计算	283（固定）+ 438 * ENTER（进入）操作数					
STVELPLAN_runTick（ISR 函数）	76	100	100			
STVELID_run（速度识别）						
RES = 1，ENB = 0	198	198	198			
RES = 0，ENB = 1	**256**	**278**	**723**	✓	×	✓
ENB = 1 后的首次调用	1196	1196	1196			
RES = 1，ENB = 1	292	292	292			
STPOSCOV_run（位置转换器）						
RES = 1，ENB = 0	127	127	127			
RES = 0，ENB = 1	**391**	**398**	**400**	✓	×	✓
ENB = 1 后的首次调用	1209	1209	1209			
RES = 1，ENB = 1	140	140	140			
STPOSCTL_run（位置控制）						
RES = 0，ENB =0	201	201	201			
RES = 0，ENB = 1	**1207**	**1212**	**1225**			
ENB = 1 后的首次调用	2043	2043	2043	✓	×	✓
改变带宽参数	1729	1729	1729			
改变惯性参数	1729	1729	1729			
RES = 1，ENB = 1	449	449	449			
STPOSMOVE_run（位置移动）						
RES = 0，ENB = 0	520	520	520			
stcurve RES = 0，ENB = 1	**790**	**1611**	**3630**			
速度受控系统配置	1467	1588	2778			
scurve RES = 0，ENB = 1	**790**	**1564**	**3205**	✓	×	✓
速度受控系统配置	1415	1551	2734			
trap RES = 0，ENB = 1	**790**	**1501**	**3130**			
速度受控系统配置	1540	1903	2438			
RES = 1，ENB = 1	996	996	996			
STPOSPLAN_run（位置规划）						
RES = 1，ENB = 0	202	202	20			
RES = 0，ENB = 1	255	255	255			
ENB = 1 后的首次调用	373	373	373			
STAY FSM 状态	255	255	255			
条件 FSM 状态 必须对每个状态完成计算	501（固定）+ 323 * 转换数 + 382 * EXIT（退出）操作数			✓	×	✓
转换 FSM 状态 必须对每个状态完成计算	301（固定）+ 432 * ENTER（进入）操作数					
STPOSPLAN_runTick（ISR 函数）	86	115	115			

注意：CPU 周期数存在差异的原因是 F2805xM 器件比 F2806xM 器件的闪存等待状态更低。

8.5.5 F2802xF 器件

8.5.5.1 CPU 周期数

表 **8-28.** 在闪存中进行最小执行的内存用量

区域	内存用量（**16** 位字）	
	RAM	闪存
库接口 (.ebss)	0x0326	×
库 (.ebss)	0x0200	×
代码 (.text)	6x06B0	0x2ED7
.cinit	×	0x007A
常数 (.econst)	×	0x0080
IQmath (.text)	×	0x00C9

节 12.3.5.3 汇总了每个函数的所有性能数据，其中的用户代码均以 InstaSPIN 库的最小执行方式从闪存中加载并执行。 请注意，采用不同执行方式时周期数不会发生显著变化，因为这些配置中的 FAST 估算器块均保留在 ROM 中。 此估算器模块占用所有 InstaSPIN-FOC 块的大部分周期。 关于管理 ISR 中的执行时间的详细信息，请参见9.1 节。

表 **8-29.** 在闪存中进行最小执行

函数名称	**CPU** 周期数			执行位置		
	最小值	平均值	最大值	**ROM**	**RAM**	闪存
HAL_acqAdcInt	17	17	17	×	✓	×
HAL_readAdcData	94	94	94	×	✓	×
Ctrl_run						
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	2320	2331	2413			
CTRL vs EST = 2	1131	1735	2413			
CTRL vs EST = 3	1131	1536	2413			
ISR vs CTRL = 2，CTRL vs EST = 1	51	1191	2413			
CTRL vs EST = 2	51	893	2413			
CTRL vs EST = 3	51	793	2413			
ISR vs CTRL = 3，CTRL vs EST = 1	51	811	2413			
CTRL vs EST = 2	51	612	2413			
CTRL vs EST = 3	51	544	2413	✓	✓	✓
Rs 在线启用，ISR vs CTRL = 1，CTRL vs EST = 1	2766	2781	2882			
CTRL vs EST = 2	1129	1969	2882			
CTRL vs EST = 3	1129	1692	2882			
ISR vs CTRL = 2，CTRL vs EST = 1	51	1424	2882			
CTRL vs EST = 2	51	1010	2882			
CTRL vs EST = 3	51	871	2882			
ISR vs CTRL = 3，CTRL vs EST = 1	51	966	2882			
CTRL vs EST = 2	51	689	2882			
CTRL vs EST = 3	51	596	2882			
HAL_writePwmData	110	110	110	×	✓	×
CTRL_setup	26	36	188	×	✓	✓

8.5.5.2 PWM = 10kHz 时的 CPU 负载

表 8-30. 在 ROM、RAM 和闪存中进行最小执行

F2802xF CPU = 60MHz 可用的 MIPs = 60MIPs PWM = 10kHz	CPU 利用率 [%]	使用的 MIPS [MIPS]	可用的 MIPS [MIPS]
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	43.13	25.88	34.12
CTRL vs EST = 2	33.2	19.92	40.08
CTRL vs EST = 3	29.88	17.93	42.07
ISR vs CTRL = 2，CTRL vs EST = 1	24.13	14.48	45.52
CTRL vs EST = 2	19.17	11.5	48.5
CTRL vs EST = 3	17.5	10.5	49.5
ISR vs CTRL = 3，CTRL vs EST = 1	17.8	10.68	49.32
CTRL vs EST = 2	14.48	8.69	51.31
CTRL vs EST = 3	13.35	8.01	51.99
Rs 在线启用，ISR vs CTRL = 1，CTRL vs EST = 1	50.63	30.38	29.62
CTRL vs EST = 2	37.1	22.26	37.74
CTRL vs EST = 3	32.48	19.49	40.51
ISR vs CTRL = 2，CTRL vs EST = 1	28.02	16.81	43.19
CTRL vs EST = 2	21.12	12.67	47.33
CTRL vs EST = 3	18.8	11.28	48.72
ISR vs CTRL = 3，CTRL vs EST = 1	20.38	12.23	47.77
CTRL vs EST = 2	15.77	9.46	50.54
CTRL vs EST = 3	14.22	8.53	51.47

8.5.5.3 PWM = 20kHz 时的 CPU 负载

表 8-31. 在 ROM、RAM 和闪存中进行最小执行

F2802xF CPU = 60MHz 可用的 MIPs = 60MIPs PWM = 20kHz	CPU 利用率 [%]	使用的 MIPS [MIPS]	可用的 MIPS [MIPS]
Rs 在线禁用，ISR vs CTRL = 1，CTRL vs EST = 1	86.27	51.76	8.24
CTRL vs EST = 2	66.4	39.84	20.16
CTRL vs EST = 3	59.77	35.86	24.14
ISR vs CTRL = 2，CTRL vs EST = 1	48.27	28.96	31.04
CTRL vs EST = 2	38.33	23	37
CTRL vs EST = 3	35	21	39
ISR vs CTRL = 3，CTRL vs EST = 1	35.6	21.36	38.64
CTRL vs EST = 2	28.97	17.38	42.62
CTRL vs EST = 3	26.7	16.02	43.98
Rs 在线启用，ISR vs CTRL = 1，			
CTRL vs EST = 2	74.2	44.52	15.48
CTRL vs EST = 3	64.97	38.98	21.02
ISR vs CTRL = 2，CTRL vs EST = 1	56.03	33.62	26.38
CTRL vs EST = 2	42.23	25.34	34.66
CTRL vs EST = 3	37.6	22.56	37.44
ISR vs CTRL = 3，CTRL vs EST = 1	40.77	24.46	35.54
CTRL vs EST = 2	31.53	18.92	41.08
CTRL vs EST = 3	28.43	17.06	42.94

8.6 数字和模拟引脚

8.6.1 引脚利用率

表 8-32 列出了 InstaSPIN 所使用的引脚。

表 8-32. 每个电机的引脚利用率

引脚类型	引脚名	每个电机的引脚用量	
		最小值	最大值
数字	PWM1A	3 （需要具有死区时间的外部故障和外部互补模式）	7
	PWM1B（可选）		
	PWM2A		
	PWM2B（可选）		
	PWM3A		
	PWM3B（可选）		
	TZ1（可选）		
模拟	IA	5 （只有两个电流，且无 VBUS 纹波补偿）	7
	IB		
	IC（可选）		
	VA		
	VB		
	VC		
	VBUS（可选）		

8.6.2 F2805x 模拟前端 (AFE)

8.6.2.1 AFE 模块注意事项

在 InstaSPIN 应用中，相关算法需要电机线电流和相电压。 在模拟电路处理完所有信号之后，处理器才能对这些模拟信号采样。 外部模拟电路会增加组件成本并增大电路板尺寸。 2805x 系列处理器可通过添加内部模拟调节组件（称为模拟前端 (AFE)）用于电机反馈信号来解决此类问题。

有关 2805x 器件的更多详细信息，请参见《*TMS320x2805x Piccolo 技术参考手册*》（文献编号 SPRUHE5）。

8.6.2.2 电流信号传输

在解决有关实施以及使用可编程增益放大器 (PGA) 和比较器的问题前，建议考虑如何将电流反馈信号从分流器传送到 PGA 的输入端。 使用分流电阻器测量线路电流时，电流值必须很小，以降低分流器中的功耗量。 由于电流值很小，因此分流器上会出现压降。 这样便会有大量电流通过分流电阻器。 从电源器件连接分流器到接地端的铜走线成为与分流器并联的电阻器。 使用分流电阻器测量电机线路电流时，必须考虑铜走线上形成的寄生电阻。

AFE 最多可具有三个不同接地端。 2805x 器件可具有多组放大器模块。 每组放大器具有不同的接地端。 在本文档中，M1 接地用于三个 PGA 构成的组，它们将分别反馈三相电机电流。 对于具有功率因数校正的系统，存在另一个接地端为 PFC 接地的单一 PGA。 固定增益放大器块使用 M2 接地端作为参考，在本文档中用于电机三相电压反馈。

此处介绍了两个用于为 AFE 的 M1 PGA 块反馈电机分流信号的选项。 第一个选项仅将内部运算放大器用于电流反馈，如 图 8-10 所示。 三个运算放大器针对反向输入共用同一接地端，因此不会产生分流差分信号。 对于单端信号，将分流器接地时必须谨慎布线，以减小分流器间不同的走线电阻。 建议将分流器接地端尽可能靠近在一起。 走线必须从分流器的汇集点连接到集成电路的 M1gnd 引脚。 由于放大器中可能加入共模噪声，因此 M1gnd 引脚和 PGA 输入必须尽可能短。 三相电流的走线必须与 M1gnd 走线尽可能靠近，以便减小 Faraday 环路的大小。 Faraday 环路在相电流走线周围形成，相电流走线从分流器顶部连接到 IC，然后背靠 M1gnd 走线连接到分流器底部，通过分流器后返回到分流器顶部。

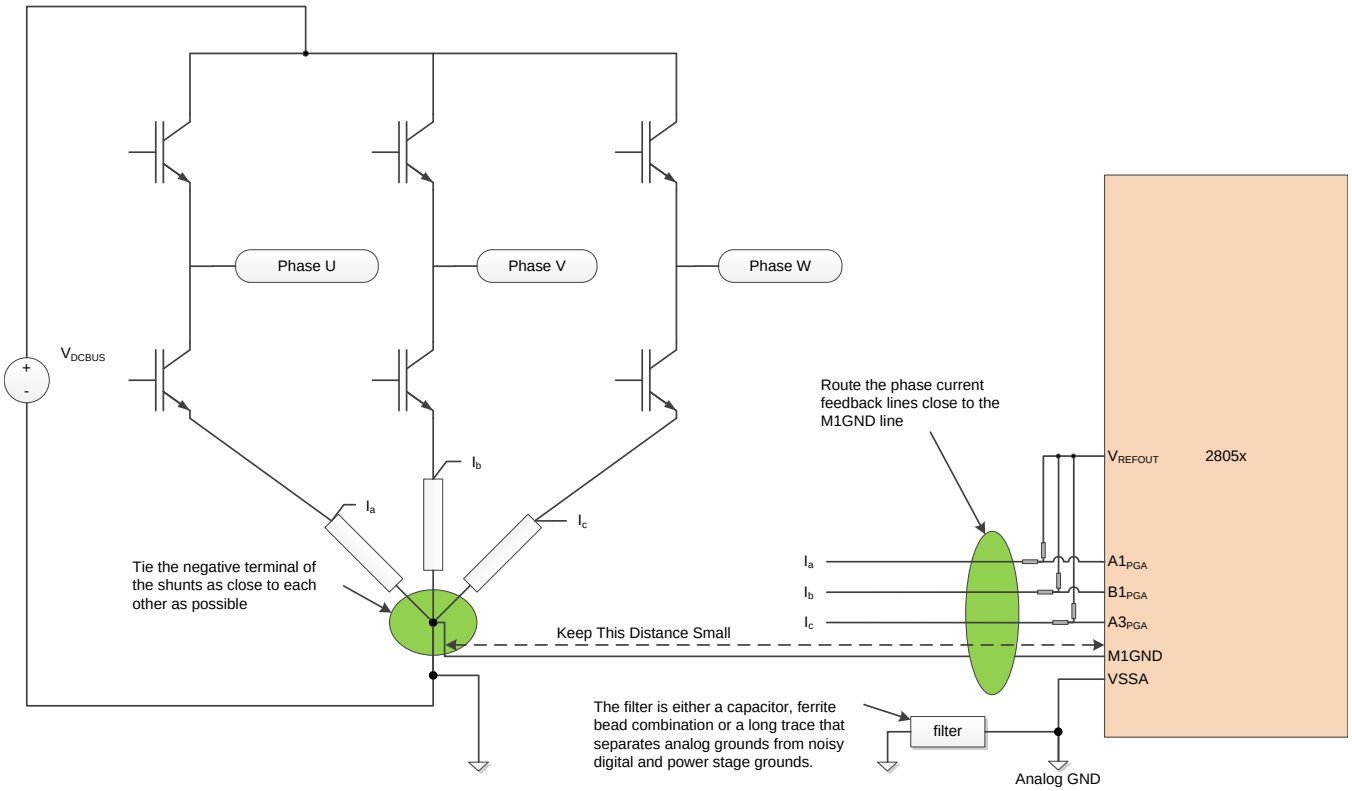


图 8-10. 电流信号通过单端连接直接传输到 PGA

第二个选项是在差分放大器配置中使用外部运算放大器，这种方法的抗扰能力最强。 有效的 Kelvin 连接可直接反馈到差分放大器，然后将差分放大器的输出传送到 PGA 输入端。图 8-11 显示了使用外部差分运算放大器时的典型布线。 由于 Kelvin 连接具有低阻抗特性并可产生真正的差分信号，因此具备极佳的抗扰度。 外部运算放大器电路可将差分电路转换为单端输出。 单端输出更容易受噪声干扰，因此最好将运算放大器的输出端靠近处理器的 AFE 输入端。

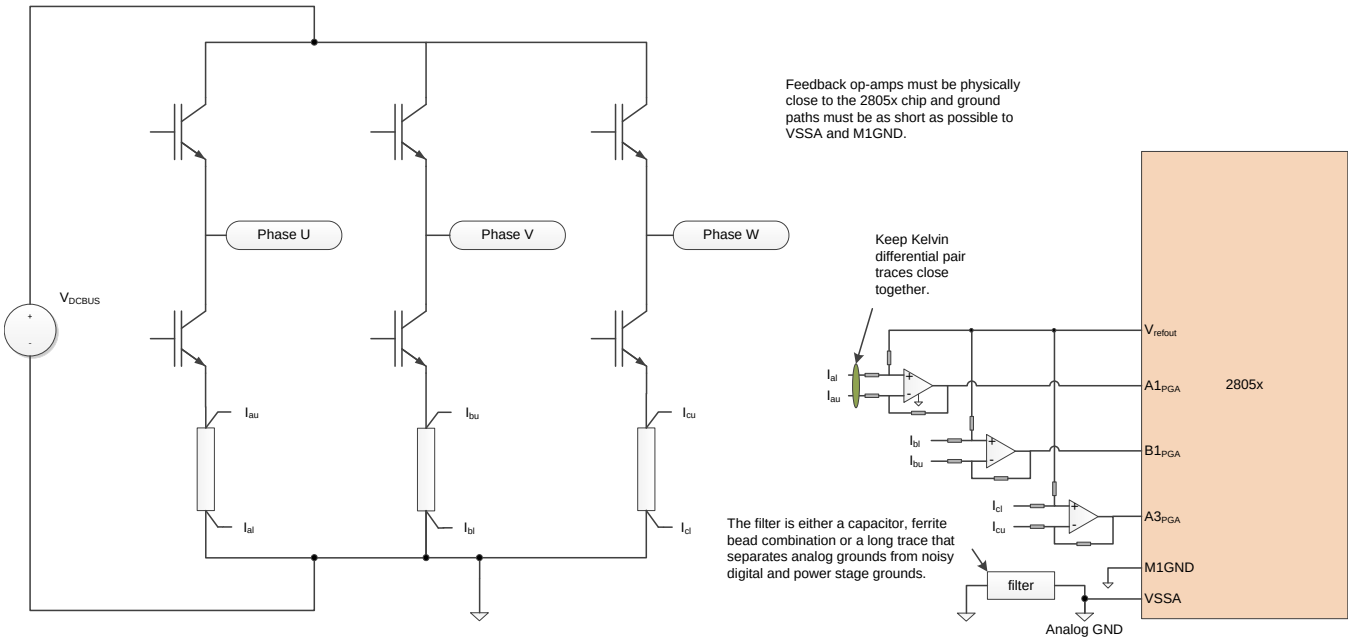


图 **8-11.** 使用外部差分放大器反馈相电流

为什么在使用外部放大器的同时使用 PGA？ 如果多个额定电流不同的电机通过同一个逆变器供电，则符合此情况。 可对电流信号放大率进行调整，以便最适合受控电机的规模。 PGA 块的输出即为比较器窗口的输入。 仍需要连接 PGA 后才能使用故障检测电路。

8.6.2.3 基准电压连接

电流可以正向或反向流经分流器，相应生成的正向电压和反向电压可反馈回分流放大器电路。 最经济高效的电机逆变器不会同时配备可处理这种双极信号的正向和反向电源。 双极电流信号引入仅对从零到正电源电压有效的放大器。 为了能够利用单极运算放大器电路测量双极信号，需将基准电压加入电流反馈运算放大器的同向侧。 为此，2805x 器件的 AFE 包含一个具有电压跟随器的 6 位 DAC，可提供输出基准电压。 图 8-12 中显示的电路配置可使用基准电压测量双极电流信号。

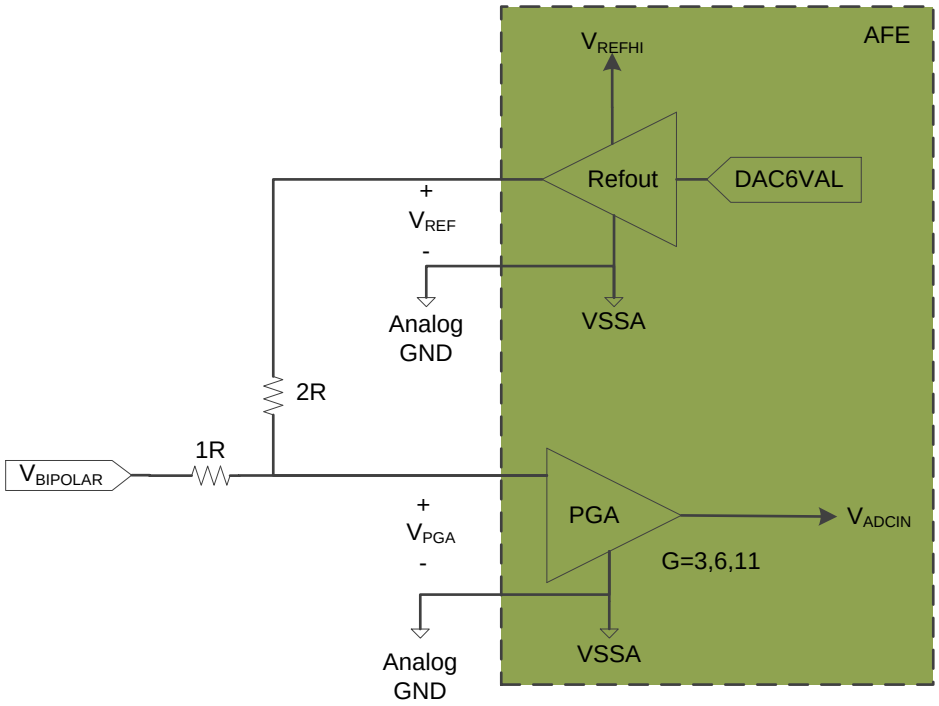


图 **8-12.** 使用 **AFE** 的内置基准电压测量双极信号

公式 24 显示了如何在 V_{PGA} 条件下计算电压。 例如，设置 PGA 增益 = 3。$V_{ADCIN} = 2V_{BIPOLAR} + V_{REF}$。 假设系统的 V_{REFHI} = 3.3V。为实现双向最大电压摆幅，将 V_{REF} 设为 1.65V。此时，测得最大峰值 $V_{BIPOLAR}$ 电压为 ±0.825V。

$$V_{PGA}=\frac{2R\cdot(V_{BIPOLAR}-V_{REF})}{(1R+2R)}+V_{REF}=\frac{2}{3}V_{BIPOLAR}+\frac{1}{3}V_{REF} \quad (24)$$

假设使用相同的硬件并且需要更高的分辨率。 PGA 增益 = 6。$V_{ADCIN} = 4V_{BIPOLAR} + 2V_{REF}$。 V_{REF} 必须调整为 0.825V。可测得最大峰值 $V_{BIPOLAR}$ 电压为 ±0.4125V。

基准电压输出由 6 位 DAC 调节。 VREFOUTCTL 电阻按照下面的公式 25 控制 DAC 的电压输出。

$$V_{REF}=\frac{V_{REFHI}\cdot(VREFOUTCTL_{DACVAL}+1)}{64} \quad (25)$$

8.6.2.4 电压信号传输

与电流信号相比，正弦电机控制驱动器中的电压信号会缓慢变化。 因此，可将更大的硬件滤波器应用于电压反馈信号，这有助于增强抗噪能力。 电压信号为单极信号，因此不需要使用特殊电路和基准电压。 电压较低的电机（400 V_{DCBUS} 以下）通常只需要带电容式低通滤波器的电阻分压器。 对于无刷直流电机控制，电压需要尽可能少地出现相移，因此，低通滤波性能取决于电机达到的最大速度。 电压反馈信号布线的唯一关键在于：低通滤波器必须尽可能靠近 AFE 或 A/D 输入引脚。

实时结构

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

9.1 InstaSPIN 软件执行时钟树

使用 InstaSPIN 时，会发生几次时钟抽取。 第一个需要考虑的时钟为中断时钟，它由使用 CPU 时钟计时的外设产生。 通常，中断服务程序 (ISR) 由 ADC 转换结束 (EOC) 触发。 此转换由 PWM 模块触发。

首先，让我们来回顾如何根据 user.h 文件中的用户参数配置 PWM 频率。 从 CPU 时钟开始，用户定义的 CPU 时钟速率（单位为 MHz）为：

```
//! \brief Defines the system clock frequency, MHz (6xF and 6xM devices)
//!
#define USER_SYSTEM_FREQ_MHz       (90)

//! \brief Defines the system clock frequency, MHz (2xF devices)
//!
#define USER_SYSTEM_FREQ_MHz       (60)
```

然后定义 PWM 频率（单位为 MHz），据此可得出中断频率。

```
//! \brief Defines the Pulse Width Modulation (PWM) frequency, kHz
//!
#define USER_PWM_FREQ_kHz          (15.0)

//! \brief Defines the Pulse Width Modulation (PWM) period, usec
//!
#define USER_PWM_PERIOD_usec          (1000.0/USER_PWM_FREQ_kHz)

//! \brief Defines the Interrupt Service Routine (ISR) frequency, Hz
//!
#define USER_ISR_FREQ_Hz           (USER_PWM_FREQ_kHz *    1000.0)

//! \brief Defines the Interrupt Service Routine (ISR) period, usec
//!
#define USER_ISR_PERIOD_usec       USER_PWM_PERIOD_usec
```

目前为止，CPU 时钟设置了 PWM 频率，同时也设置了 ISR 的频率。 此时 ISR 实际并非由 PWM 定时器本身触发，而是由 ADC 转换结束指令触发，而 ADC 则是由 PWM 定时器启动。

图 9-1 是从 CPU 到生成 ISR 的全过程时钟时序图。

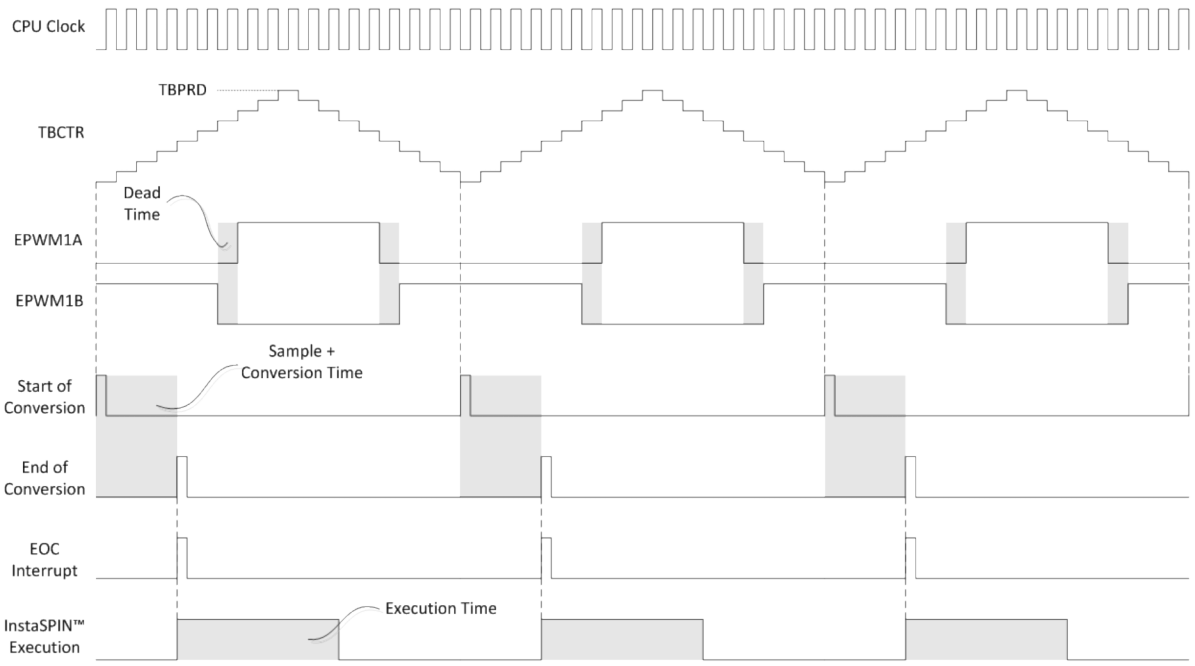


图 **9-1.** 时钟时序 **-** 从 **CPU** 到生成 **ISR**

这个时序图表示德州仪器 (TI) InstaSPIN 软件包交付时的中断触发情况，因为这是在取中断时确保转换就绪的最安全方法。 用户可能还会考虑其它情况，例如 ADC 提前中断、PWM 中断或者 CPU 定时器中断。 唯一要求是那些中断可定期生成。

注意，执行时间可通过多种不同方法测量。 以下示例介绍如何测量执行时间：

- *GPIO 切换*。 这是一种简单但不够准确的执行时间测量方法，即，在测量代码开始执行时设置一个 GPIO，然后在代码执行后立刻清除此 GPIO。 这种方法极具图形化，可以呈现在一定的图形范围内，而中断是周期性发生，这为测量执行时间的范围提供了良好的触发机制。 如有必要，可将时间转换为 CPU 周期。 使用这种方法时需要考虑特定架构设置和清除 GPIO 所耗费的时间以及中断获取和返回时间。
- *CPU 定时器捕捉*。 更加精确的执行时间测量方法是使用 CPU 定时器。 具体过程是在 CPU 时钟开始计时时运行 CPU 定时器，不使用预分频器或后分频器，然后在代码执行后读取定时器。 这样可得出执行相关代码的 CPU 周期数。

在 InstaSPIN 执行时序中有几个抽取值，也称为节拍率，该速率允许 InstaSPIN 中控制代码的不同部分采用不同的执行时钟速率。 以下节拍率可用于 InstaSPIN：

```
//! \brief Defines the number of pwm clock ticks per isr clock tick
//!        Note: Valid values are 1, 2 or 3 only
#define USER_NUM_PWM_TICKS_PER_ISR_TICK       (1)

//! \brief Defines the number of isr ticks per controller clock tick
//!
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK      (1)

//! \brief Defines the number of controller clock ticks per current controller clock tick
//!
#define USER_NUM_CTRL_TICKS_PER_CURRENT_TICK  (1)

//! \brief Defines the number of controller clock ticks per estimator clock tick
//!
#define USER_NUM_CTRL_TICKS_PER_EST_TICK      (1)
```

```
//! \brief Defines the number of controller clock ticks per speed controller clock tick
//!
#define USER_NUM_CTRL_TICKS_PER_SPEED_TICK  (10)

//! \brief Defines the number of controller clock ticks per trajectory clock tick
//!
#define USER_NUM_CTRL_TICKS_PER_TRAJ_TICK   (10)
```

为了显示所有这些节拍率，请参见图 9-2。 定义的以下缩写便于在软件执行时钟树图中引用：

USER_NUM_PWM_TICKS_PER_ISR_TICK -> /ETPS

USER_NUM_ISR_TICKS_PER_CTRL_TICK -> /ISRvsCTRL

USER_NUM_CTRL_TICKS_PER_CURRENT_TICK -> /CTRLvsCURRENT

USER_NUM_CTRL_TICKS_PER_EST_TICK -> /CTRLvsEST

USER_NUM_CTRL_TICKS_PER_SPEED_TICK -> /CTRLvsSPEED

USER_NUM_CTRL_TICKS_PER_TRAJ_TICK -> /CTRLvsTRAJ

对于 F2806x 器件，软件执行时钟树从 90MHz 的 SYSCLKOUT 开始，而所有其它值都从此时钟抽取。 对于 F2805x 和 F2802x 器件，最大频率为 60MHz，而不是 90MHz。

在默认设置为 1 的时钟预分频器后，为得到 PWM 发生器的最佳分辨率，我们使用了 TBPRD 寄存器（请参见图 9-2）。 此寄存器含有一个周期值，这样输出端便能生成 PWM 频率。

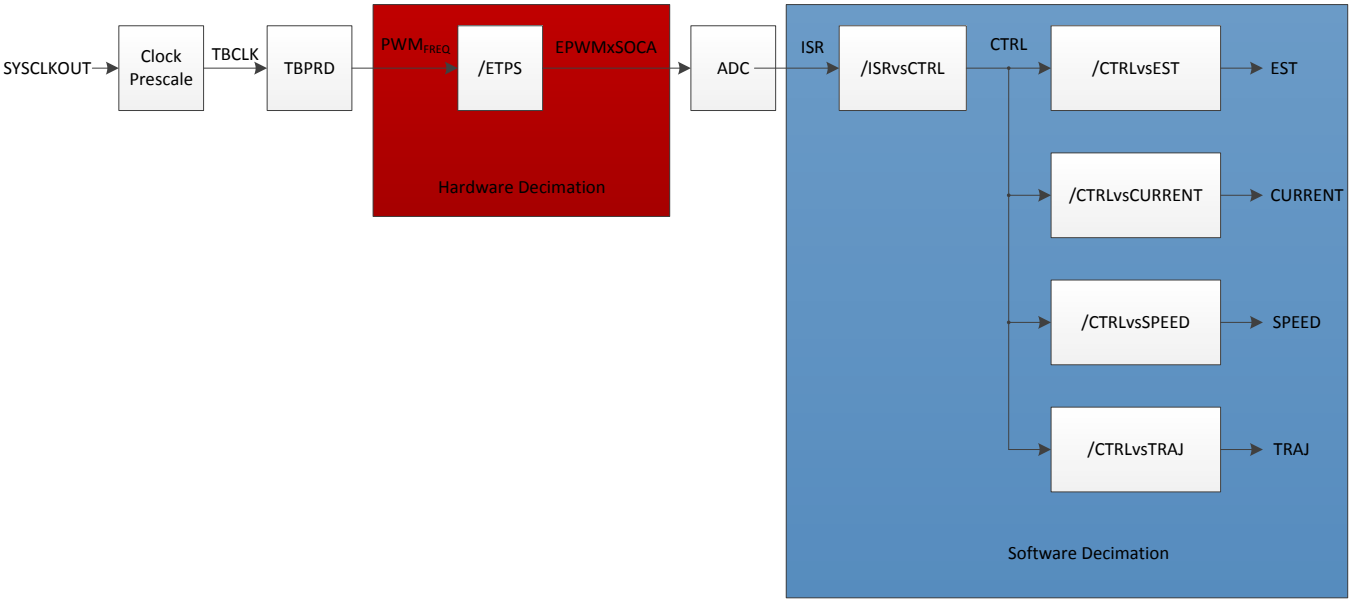


图 **9-2.** 软件执行时钟树

软件执行时钟树的第一次抽取发生在硬件中。 根据 ETPS 寄存器（事件触发预分频寄存器）的值，PWM 频率可实现 1 分频、2 分频或 3 分频。 当 ADC 开始转换信号需要在每个 PWM 周期、每 2 个或每 3 个 PWM 周期触发一次时，这一特性非常实用。 这种硬件抽取过程由 user.h 文件中的 USER_NUM_PWM_TICKS_PER_ISR_TICK 定义控制。

第二次抽取在软件中完成，下文将对此进行详细介绍。

9.2 用于实时调度的软件抽取

在图 9-3 中突出显示的软件节拍率用于抽取软件中 InstaSPIN 的执行，也称为实时调度节拍率。

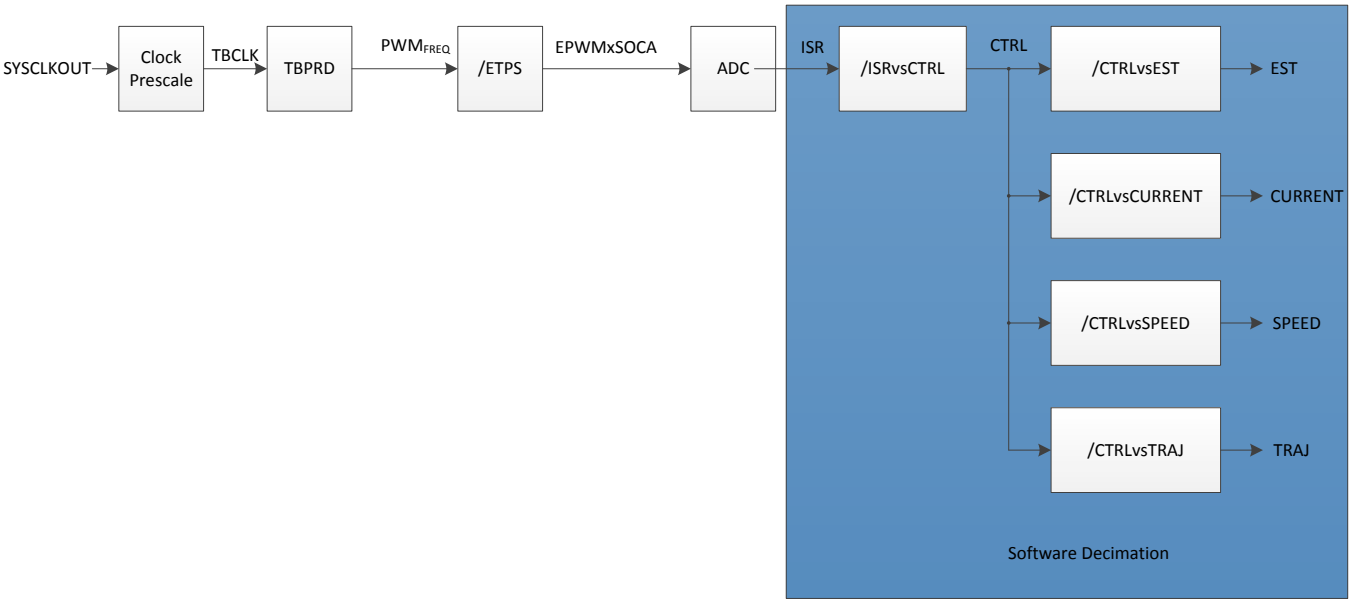


图 **9-3.** 实时调度节拍率

9.2.1 USER_NUM_ISR_TICKS_PER_CTRL_TICK

第一个节拍率定义 InstaSPIN 作为一个整体在 ISR 转换结束后开始执行的速率。 如果此节拍率大于 1，每次执行 InstaSPIN 时都会检查内部计数器，如果该计数器未达到节拍率值，则将从 InstaSPIN 执行过程中返回。 除了检查此计数器，InstaSPIN 库中不会执行任何代码。图 9-4 显示如何在 ISR 中检查节拍计数器。

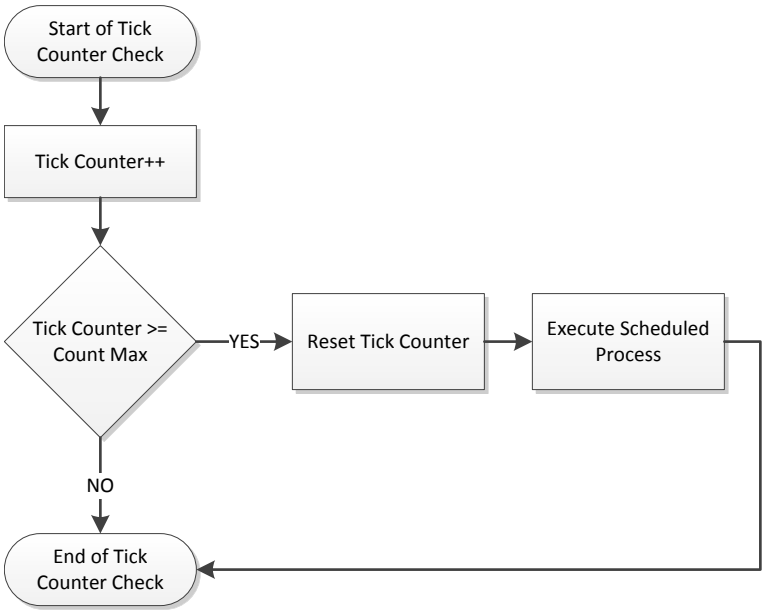


图 **9-4.** 节拍计数器流程图

图 9-5 显示节拍率为 2 时的 InstaSPIN 执行过程。

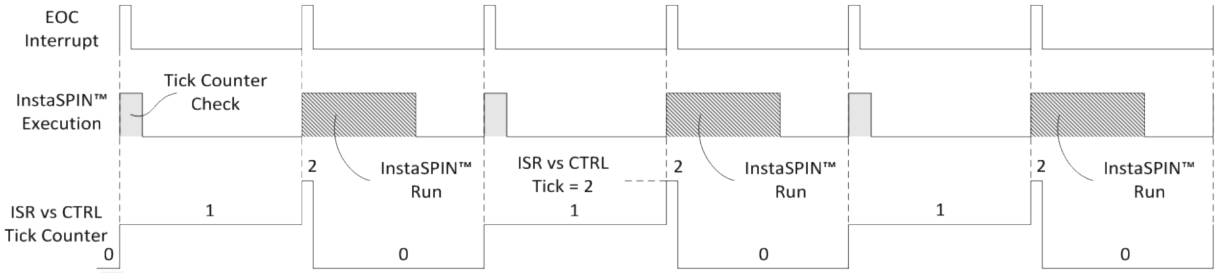


图 **9-5. InstaSPIN** 时序

图 9-6 也可表示为如下所示的软件执行时钟树。

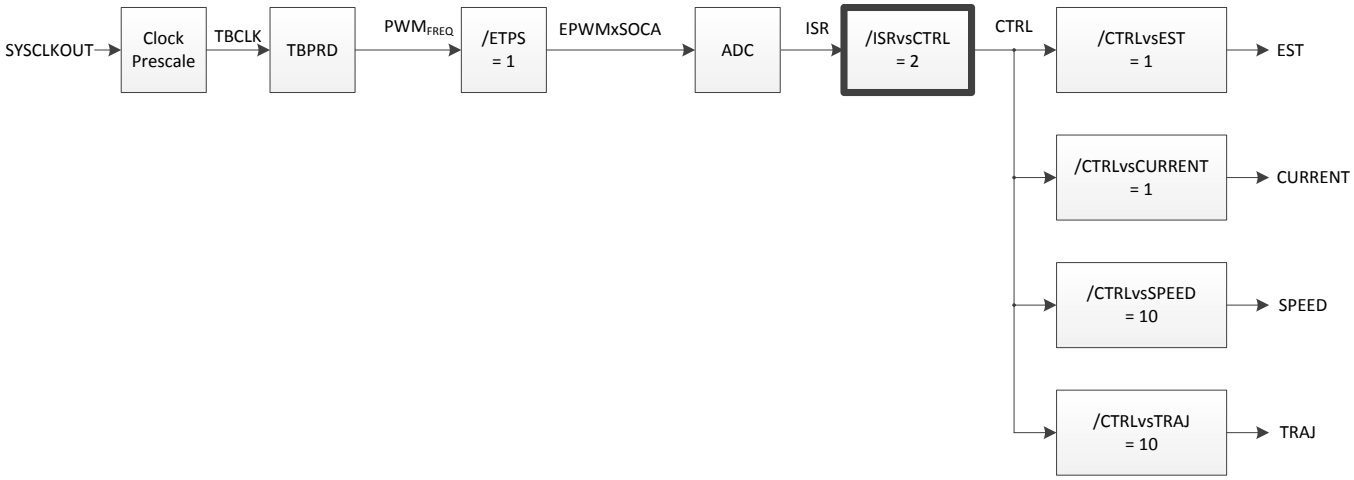


图 **9-6. InstaSPIN** 时序软件执行时钟树

请注意当突出显示的块值为 2 时，如何在软件执行时钟树中实现 2 分频。

从中断到控制器的节拍率大于 1 的原因主要为以下两点：

- 第一个原因是减少 CPU 的使用率。
- 第二个原因是允许更高的 PWM 频率并减小节拍率，以便 InstaSPIN 在更高的频率下仍可执行。

例如，如果 PWM 频率为 50kHz 并且没有使用硬件抽取（本文档稍后将讨论相关主题），则 ISR 转换结束时将为相同速率，50kHz。 所有功能需要在 1/50kHz = 20µs 内完成执行。 如果 ISR 内的功能需要 30µs，则：

执行时间 > ISR 周期 → 30µs > 20µs

这将产生中断溢出，导致 ADC 采样样本被覆写，并且控制时序也将受到影响。

当中断小于执行时间时，如果考虑以下准则，使用不同的 ISR vs CTRL 节拍率可确保安全。

确认中断期间有足够的时间来执行 InstaSPIN。

这是因为在中断服务程序中执行 InstaSPIN 时必须有足够的时间，这样才能避免转换溢出。 例如，如果一个 ISR 程序尚未执行完便进入第二个程序，则第一个程序将完全丢失，并且时序将会受到影响。 图 9-7 显示了一个良好的示例，当 ISR 拥有充足的时间时，InstaSPIN 完成执行且没有发生 ISR 溢出。

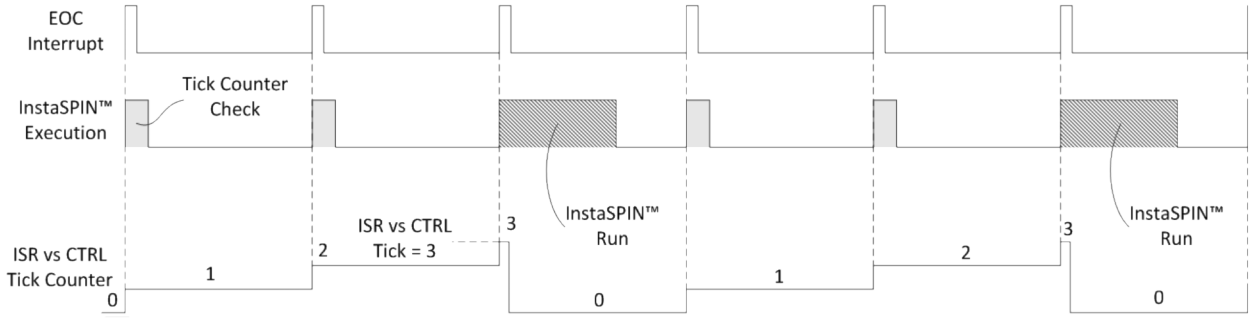


图 **9-7. InstaSPIN** 完成执行且没有发生 **ISR** 溢出

图 9-8 显示了此时序图的软件执行时钟树表示。

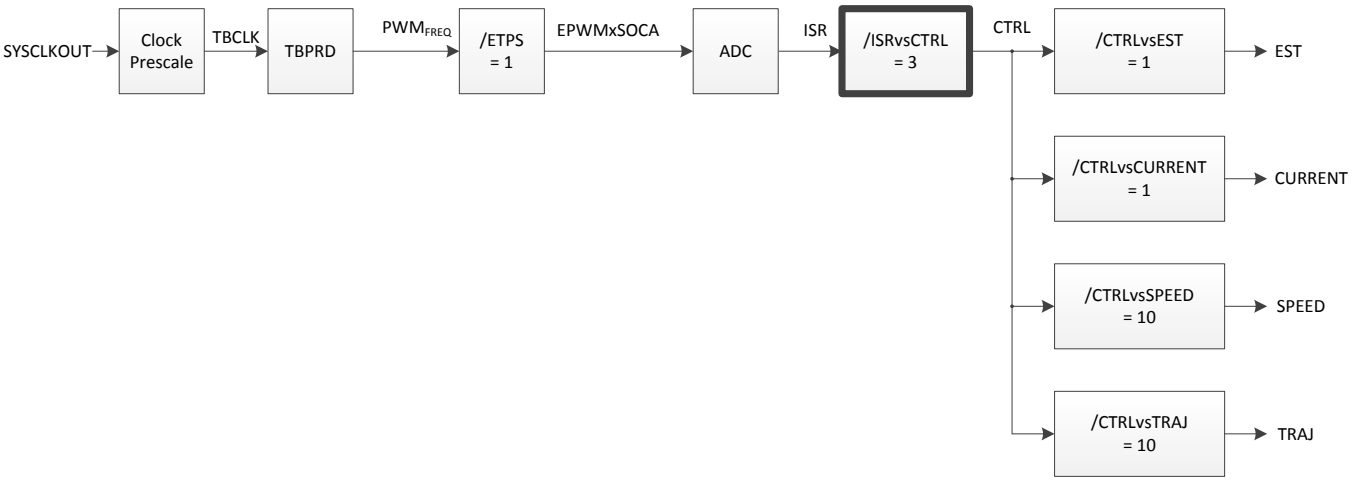


图 **9-8. InstaSPIN** 时序软件执行时钟树 **-** 无 **ISR** 溢出

以下是另一个示例，本例中使用的抽取率具有较高的 PWM 频率，导致 ISR 时间较短，不过，即使某个 ISR 没有立即执行，它最终也会完成执行并且不会发生 ISR 溢出 (图 9-9)。 这种方式也可以正常操作，并不会影响性能。

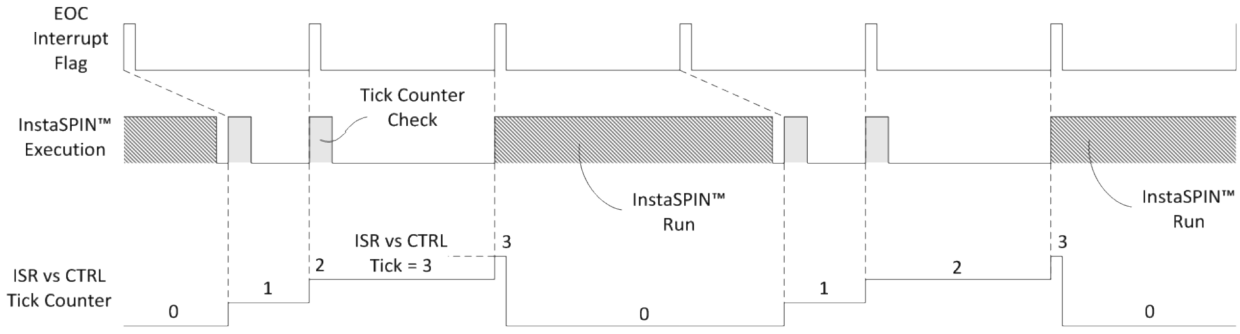


图 **9-9.** 较高 **PWM** 频率时的 **InstaSPIN** 时序

另一方面，如果 PWM 频率设置过高，转换结束中断可能被另一个中断溢出。 由于溢出会导致整组 ADC 采样样本丢失，因此需避免这种情况。 除此之外，中断溢出会导致丢失一个完整中断，因而造成 InstaSPIN 状态机时序错误，而状态机时序正是取决于中断的周期性。 InstaSPIN 库中的时序调度不当会引起诸多问题，例如，角度估算不准确、速度估算关闭、速度和电流控制器未正常运转等等。 以下示例显示中断溢出情况。 注意，InstaSPIN 的执行过程不变，改变的是 PWM 频率（进而改变了转换结束中断频率）和抽取次数。 中断溢出的第一个示例没有执行抽取，因此全部 InstaSPIN 在每个中断中均执行 (图 9-10)。

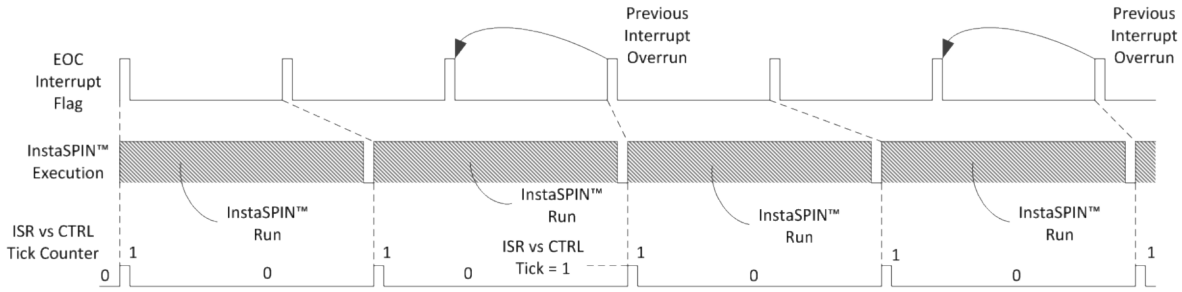


图 9-10. 未执行抽取时的中断溢出时序

图 9-11 显示了此时序图的软件执行时钟数值。

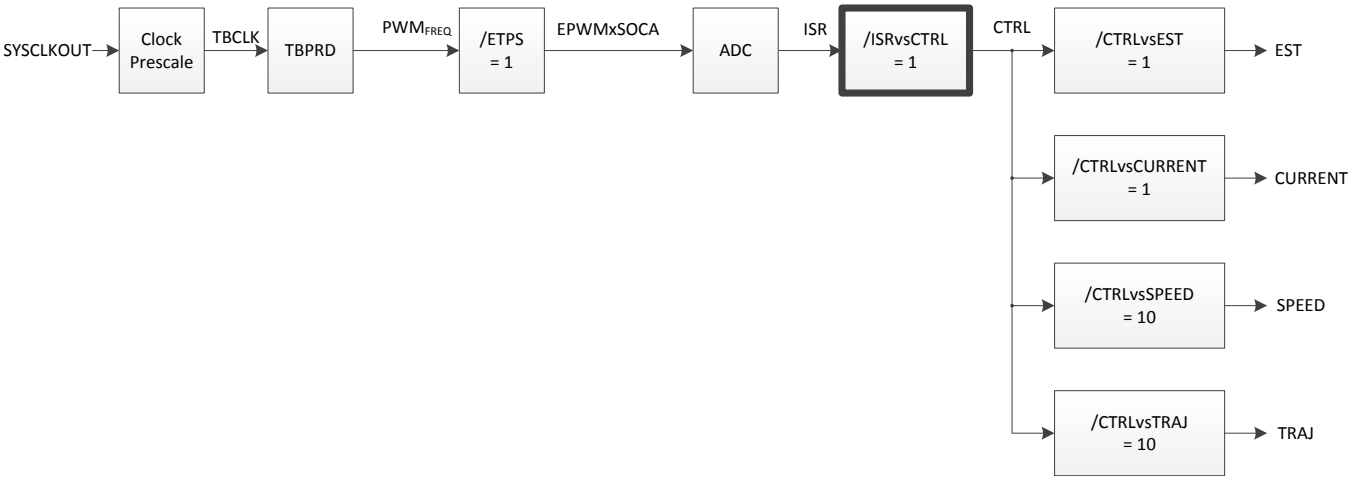


图 9-11. 未执行抽取时的中断溢出软件执行时钟树

请注意在几个中断后有一个完整中断是如何溢出的。 这种方法的问题在于存在时序偏移，所以 InstaSPIN 状态机和控制器内部时序比实际慢。

另一个示例是使用抽取率。 图 9-12 中使用的抽取率值为 2，如图所示，即使在使用抽取率时仍有中断溢出限制。

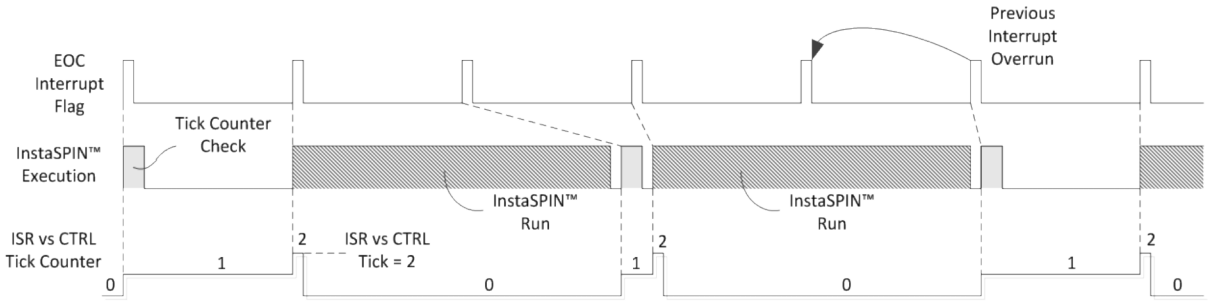


图 **9-12.** 执行抽取时的中断溢出时序

图 9-13 显示了此时序图的软件执行时钟树值。

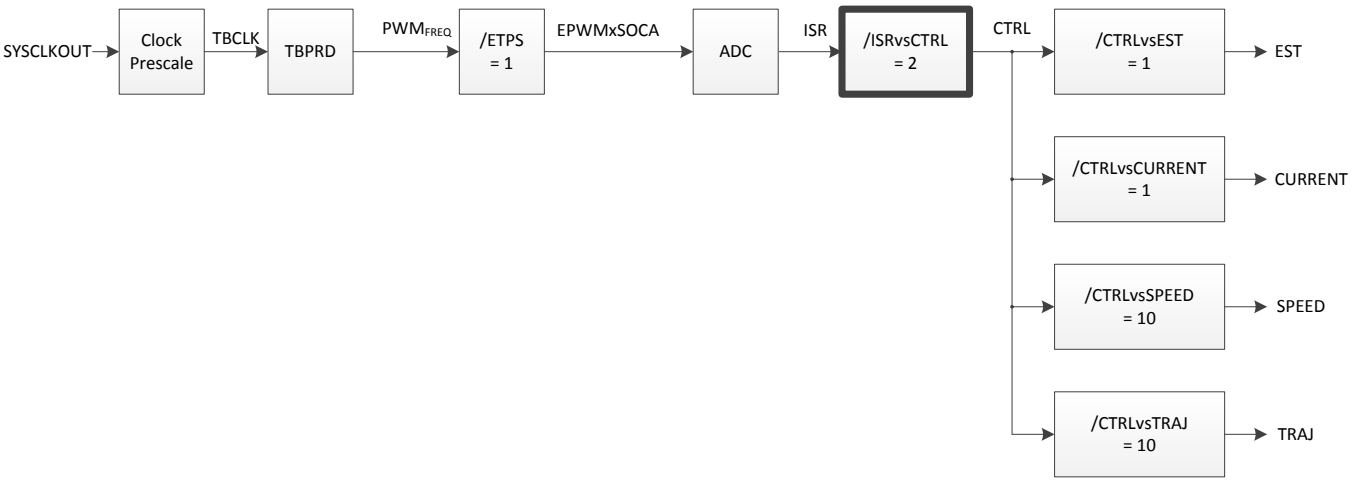


图 **9-13.** 执行抽取时的中断溢出软件执行时钟树

将 InstaSPIN 运行速率至少设为电机电频率的 10 倍。

需要考虑的第二个方面是当设置 PWM 频率且 ISR vs. CTRL 节拍率为 InstaSPIN 频率对电机电频率的倍数。 这是因为在闭环系统中运行时，磁场定向控制取决于电角度，每个电周期应该有充分的估算角度更新来保证磁场正确定向。 与此要求类似的是当交流信号需要进行数字采样时。 此过程与奈奎斯特频率有关，当此频率刚好大于采样频率时便足以避免出现混叠。 在磁场定向控制系统中，奈奎斯特频率不足以提供有效的电机控制。 建议 InstaSPIN 运行速率至少是电机电频率的 10 倍。 要了解电机（即永磁同步电机 (PMSM)）的电频率，我们需要知道电机转速和电极数量。 例如：

极对数：4

转速：7500RPM

电频率：以 RPM 为单位的转速 * 电极对数 / 60 = 7500 * 4 / 60 = 500Hz

推荐的最小 InstaSPIN 运行速率 = 10 * 电频率 = 5000Hz

在此示例中，选择的 ISR vs. CTRL 节拍率为 3，得出 ISR 频率为 5000 * 3 = 15000Hz。

图 9-14 显示了此示例中生成的波形。

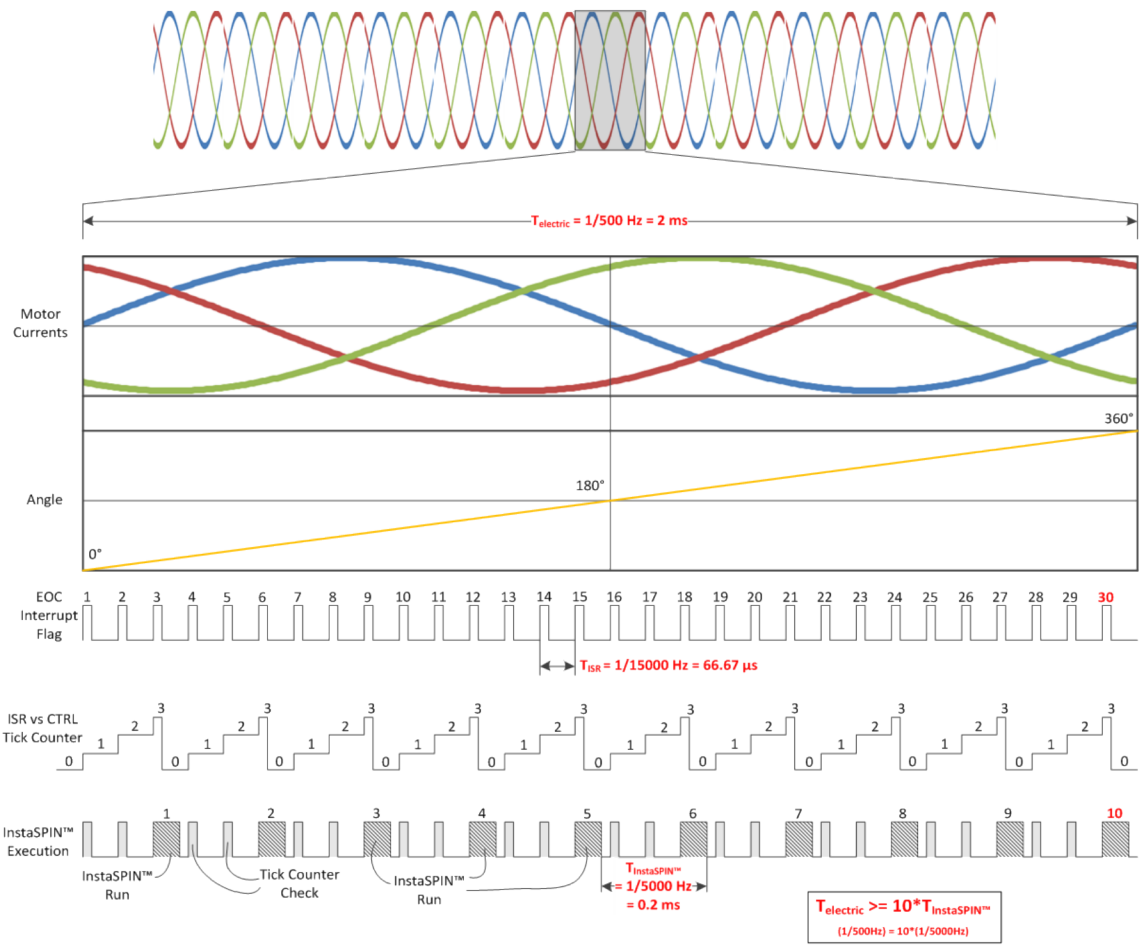


图 **9-14. ISR** 频率波形

图 9-15 显示了此示例中软件执行时钟树的数值。

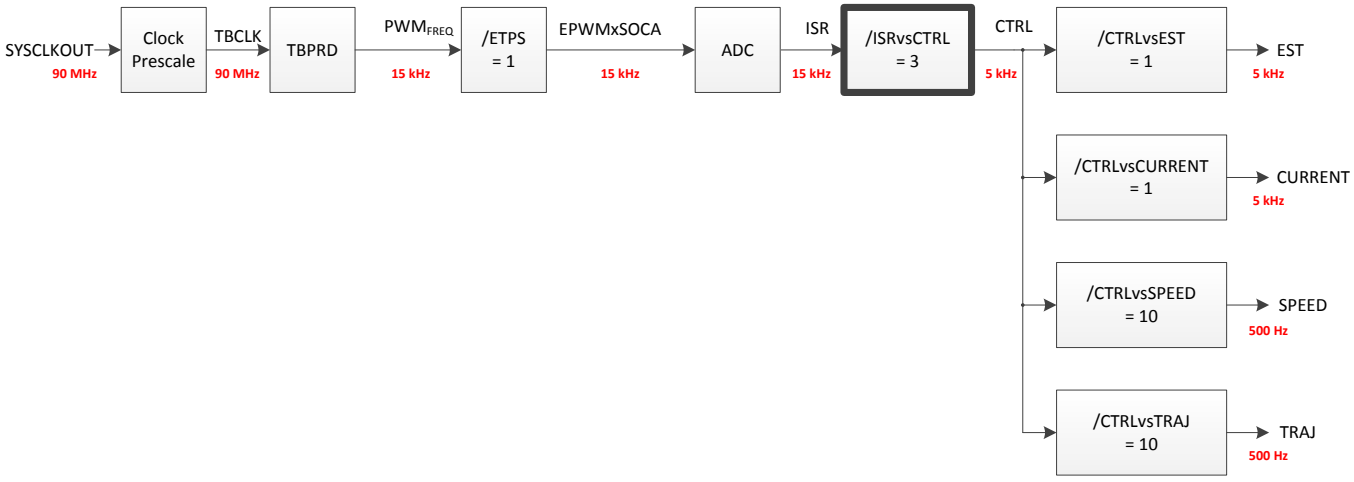


图 **9-15. ISR** 波形的软件执行时钟树

9.2.2 USER_NUM_CTRL_TICKS_PER_CURRENT_TICK

下面要介绍的第二个节拍率是 CTRL vs CURRENT 节拍率。 此节拍率用于减缓电流控制器相对于 InstaSPIN 执行速率的速度。 由于只有两个 PI 控制器，此节拍率仅可减缓电流控制器的执行速度，对于减轻 CPU 负载并没有实际帮助。 但它确实降低了电流控制性能，所以建议保持节拍率为 1，这意味着电流控制器的执行速度将与 InstaSPIN 执行速率相同。 关于如何使用此节拍率的示例，请参见图 9-16。 在本示例中，同样选择 ISR vs CTRL 节拍率为 3，以显示 CTRL vs CURRENT 节拍是如何从第一个节拍率开始级联的。

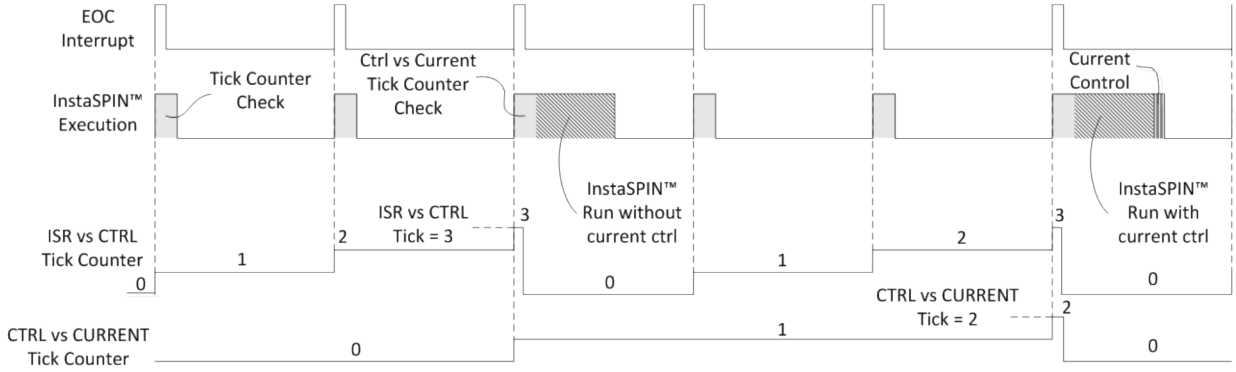


图 **9-16.** 节拍率时序

图 9-17 在突出显示框中显示此示例的相应值。

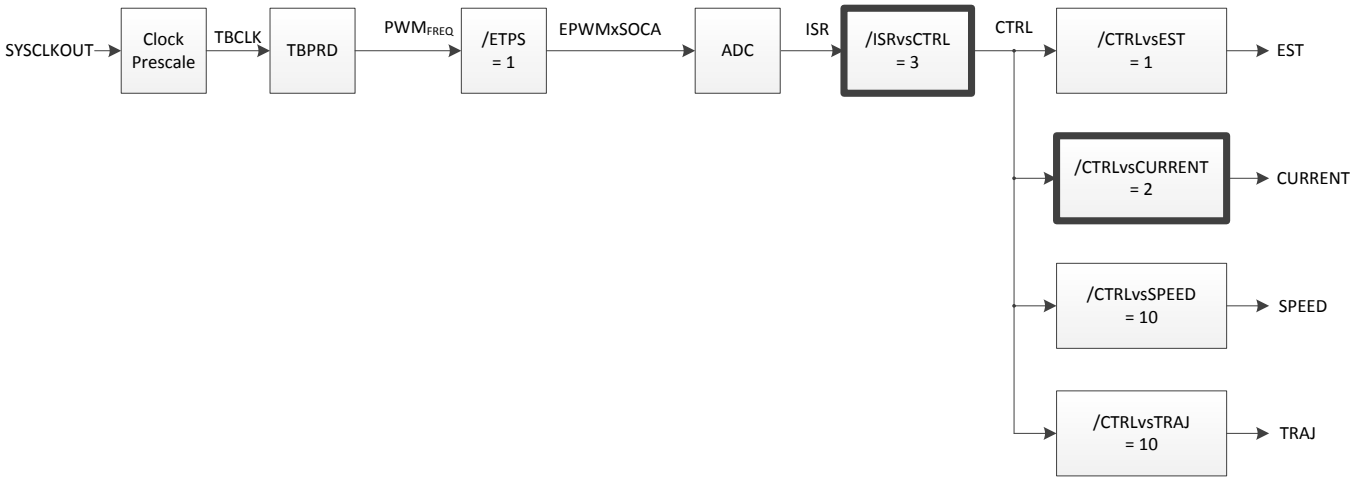


图 **9-17.** 节拍率软件执行时钟树

9.2.3 USER_NUM_CTRL_TICKS_PER_EST_TICK

第三个抽取率是执行 InstaSPIN 内部的估算器，也称为 FAST™ 算法。 这是 InstaSPIN 中最常用的节拍率之一，因为它可抽取 InstaSPIN 的最耗时部分，也就是 FAST 估算器。 如之前抽取电流控制器的节拍率所示，此节拍率抽取估算器执行过程。 关于如何从 InstaSPIN 执行时钟级联此估算器的示例，请参见图 9-18。 此图同时显示值为 1 的 ISR vs CTRL 节拍率、值为 2 的 CTRL vs CURRENT 节拍率和值为 2 的 CTRL vs EST 节拍率。 此示例说明了如何结合几个节拍率实现所需的 CPU 带宽，同时也显示了 InstaSPIN 中其它时钟的相关性。

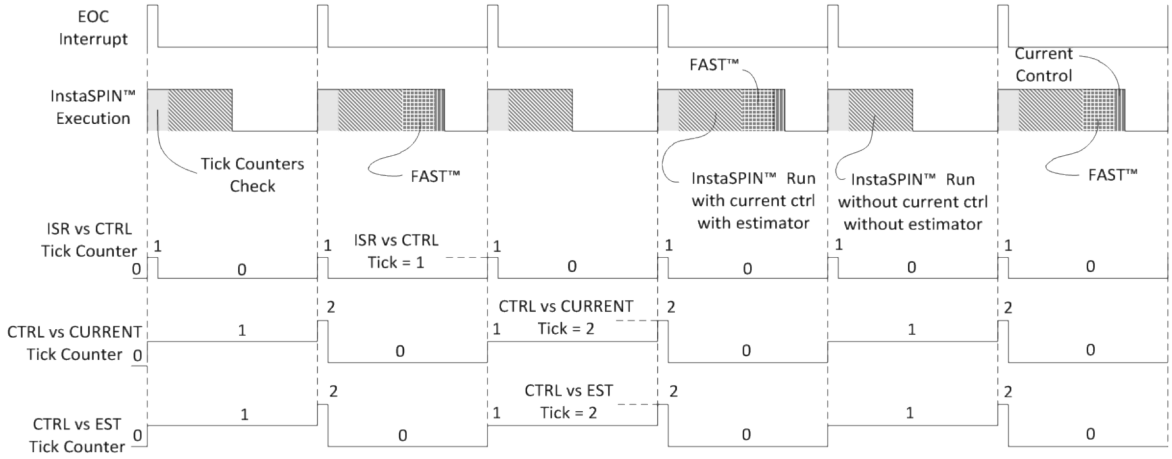


图 **9-18. FAST** 估算器节拍率时序

图 9-19 在突出显示框中显示此时序图的相应值。

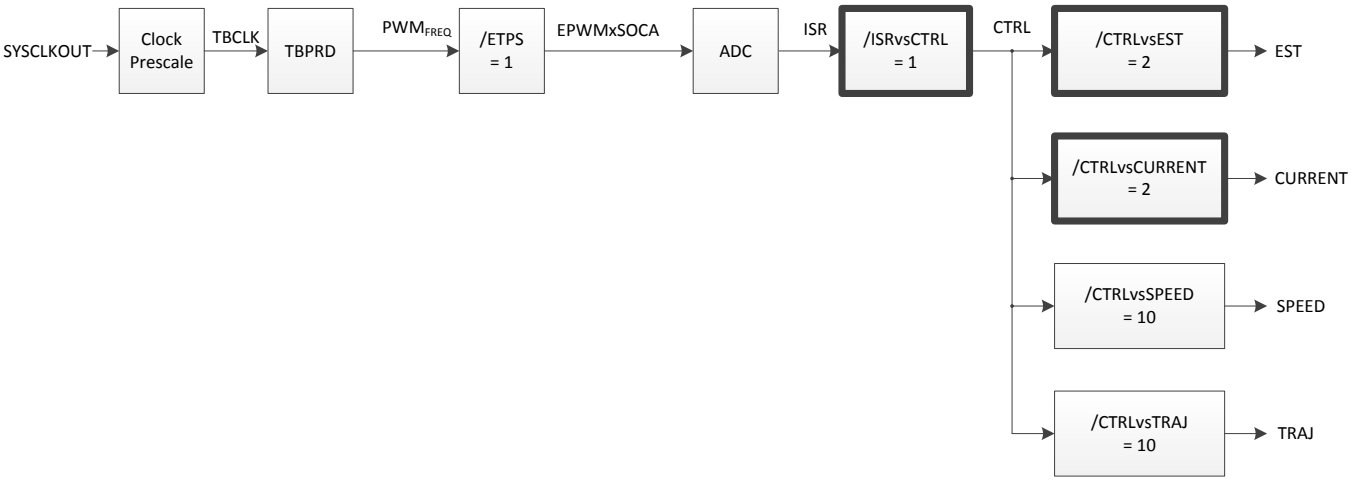


图 **9-19. FAST** 估算器节拍率软件执行时钟树

9.2.4 实例

示例如下，此例中 CPU 负载受应用程序限制并且 PWM 频率要求固定。 例如，在案例研究中考虑以下参数：

仅检查节拍率计数器的 InstaSPIN 执行时间：2.7µs

FAST 估算器执行时间：12.9µs

不使用 FAST 估算器的 InstaSPIN 执行时间：14.2µs

使用 FAST 的 InstaSPIN 总执行时间：27.1µs = 12.9µs + 14.2µs

PWM 频率要求：50kHz (TISR = 20µs)

需要如此高 PWM 频率的典型情况就是当电机电感很低时。 低 PWM 频率会因低电感生成不必要的电流波纹。 这些应用的解决方案是采用更高的 PWM 频率。 在此例中，需要 50kHz PWM 频率。

首先尝试的第一组配置是将 ISR vs CTRL、CTRL vs CURRENT 和 CTRL vs EST 全部设置为 1，这样可获得最佳性能。 如果尝试将这些节拍率设置为 1，可得到图 9-20。

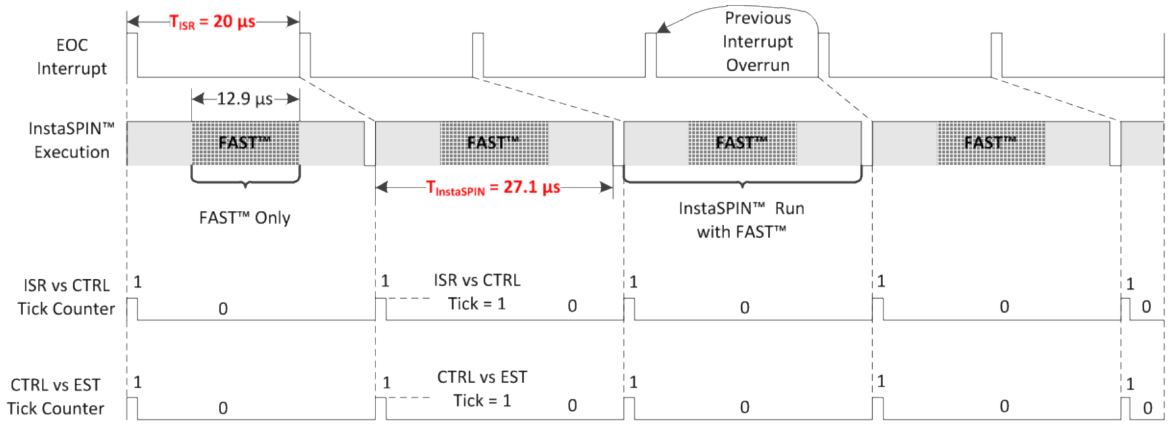


图 9-20. 节拍率时序

如时序图所示，中断时间比所需的执行时间短，这将导致中断溢出，需要避免这种情况发生。

$T_{ISR} < T_{InstaSPIN}$ → 20µs < 27.1µs → 这导致 ISR 溢出，造成 InstaSPIN 出现意外结果。

图 9-21 在突出显示框中显示此时序图的相应值。

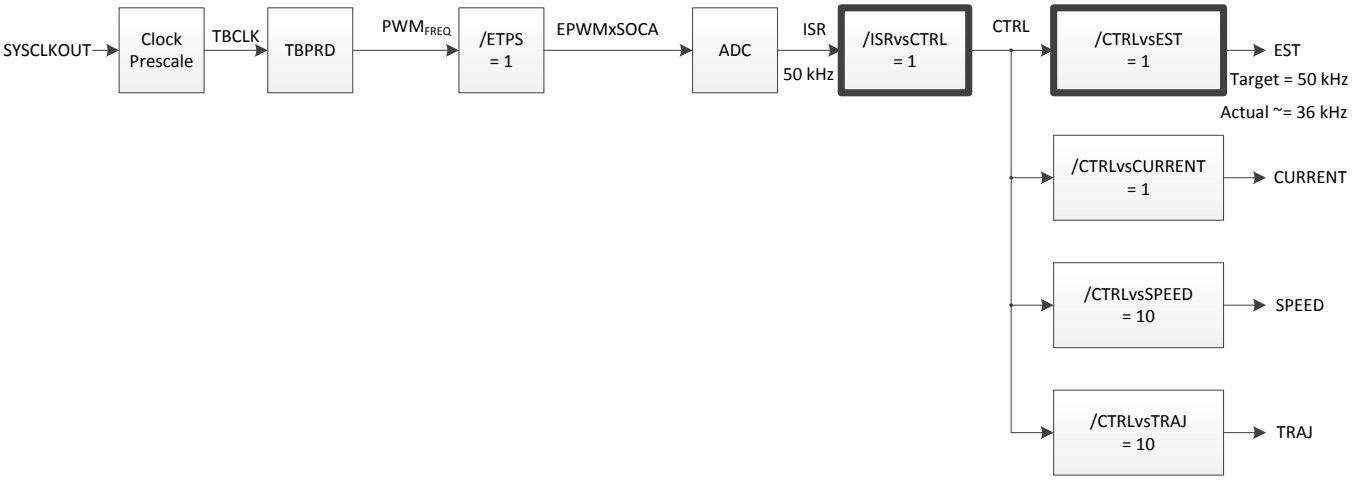


图 9-21. 节拍率软件执行时钟树

请注意中断时间过短如何导致无法在同一节拍率下执行 InstaSPIN，并且永远跟不上执行速度。 事实上，在几个中断后就会丢失中断，这将导致意外结果。 解决此问题的方法是使用节拍率，以便 FAST 估算器以相对低于 InstaSPIN 其余部分的速率运行。 在图 9-22 中，看看是否能使用值为 2 的 CTRL vs. EST 节拍率来解决溢出问题。

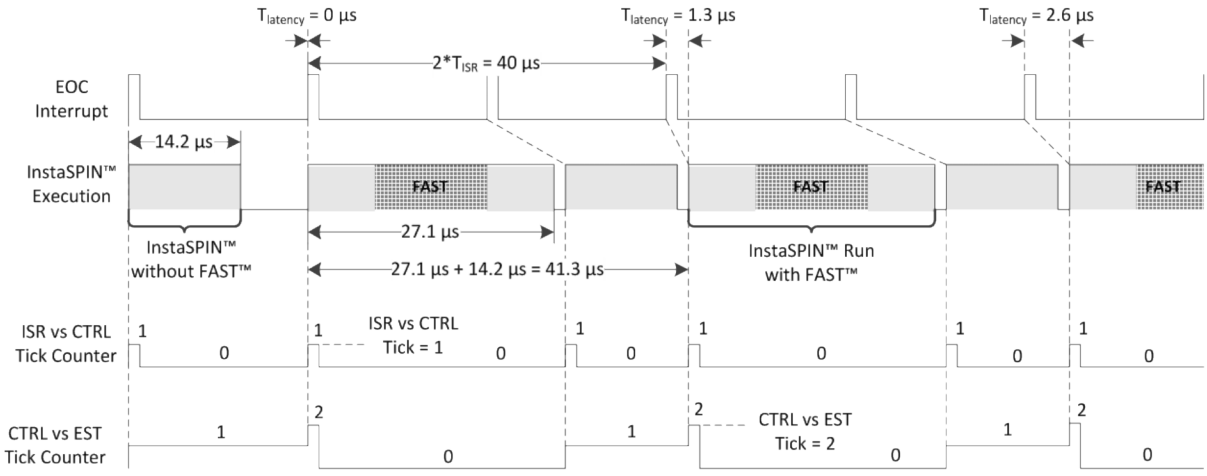


图 **9-22. CTRL vs. EST** 时序 **- 节拍率 = 2**

在这种情况下，由于估算器节拍率值为 2，因此测量时序时必须考虑两个中断，如图所示，2 个中断的运行时间短于不使用 FAST 加使用 FAST 时的执行时间，这将导致中断溢出，因而出现异常行为。

$2 * T_{ISR} < (T_{不使用\ FAST\ 的\ InstaSPIN} + T_{使用\ FAST\ 的\ InstaSPIN}) \rightarrow 40\mu s < 41.3\mu s \rightarrow$ 意外的 InstaSPIN 结果

图 9-23 在突出显示框中显示此时序图的相应值。

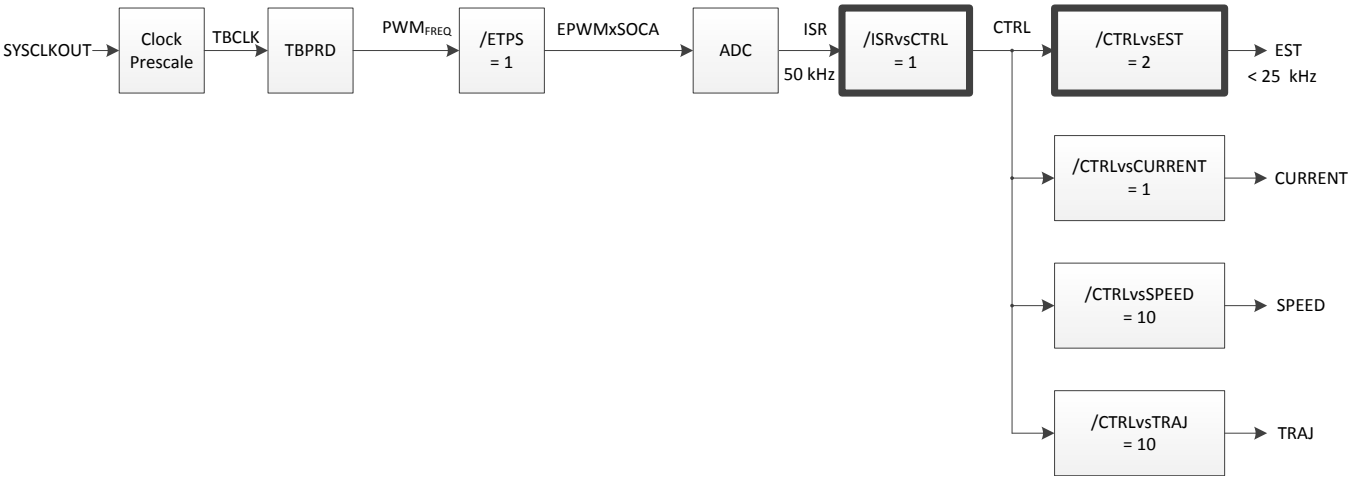


图 **9-23. CTRL vs. EST** 软件执行时钟树 **- 节拍率 = 2**

如图所示，每次执行使用 FAST 的 InstaSPIN 时，都会增加延迟。 在这三个周期内，可以看到此延迟如何从 0µs 增加到 1.3µs 再增加到 2.6µs。 由于执行时间跟不上中断速率，我们可以预测，几个周期后将出现中断溢出。

这种情况的解决方法是将 CTRL vs. EST 节拍率增至 3，以便每个 InstaSPIN 周期的延迟降为 0，如图图 9-24 所示。

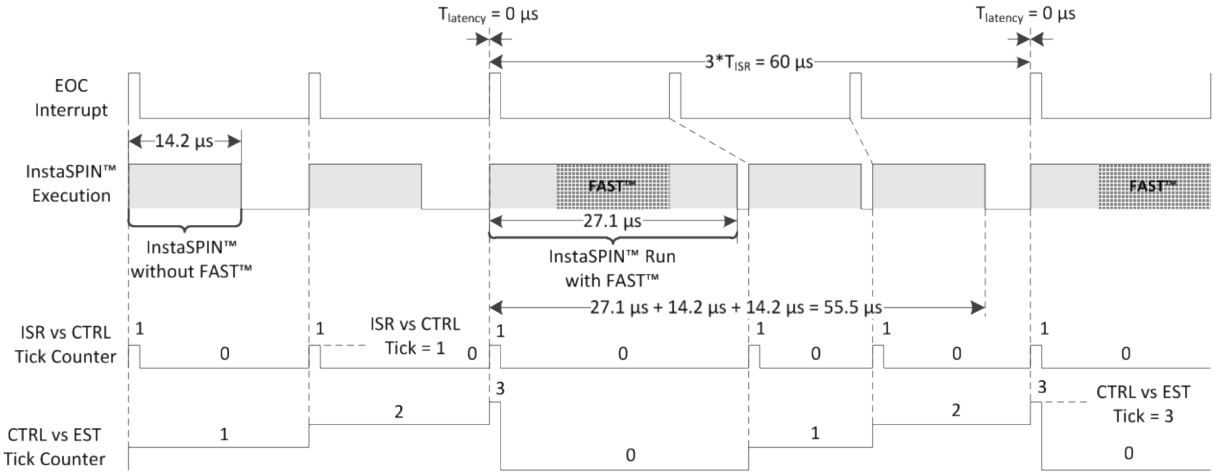


图 **9-24. CTRL vs. EST** 时序 **-** 节拍率 **= 3**

在这种情况下，由于估算器节拍率值为 3，因此测量时序时必须考虑三个中断，如图所示，3 个中断的运行时间长于不使用 FAST 加使用 FAST 时的执行时间，这样即可避免中断溢出。

$3 * T_{ISR} <$（$2 * T_{不使用\ FAST\ 的\ InstaSPIN} + T_{使用\ FAST\ 的\ InstaSPIN}$）→ 60µs > 55.5µs → 预期的 InstaSPIN 结果

除 ISR 之外，其它任务或者其它较低优先级中断的可用 CPU 计算如下：

$3 * T_{ISR} - (2 * T_{不使用\ FAST\ 的\ InstaSPIN} + T_{使用\ FAST\ 的\ InstaSPIN}) = 4.5µs$

CPU % 剩余 = 100% * 4.5µs / 60µs = 7.5%

2806x MIPS 剩余 = CPU % 剩余 * 最大 MIPS = 7.5% * 90MIPS = 6.75MIPS

图 9-25 在突出显示框中显示此时序图的相应值。

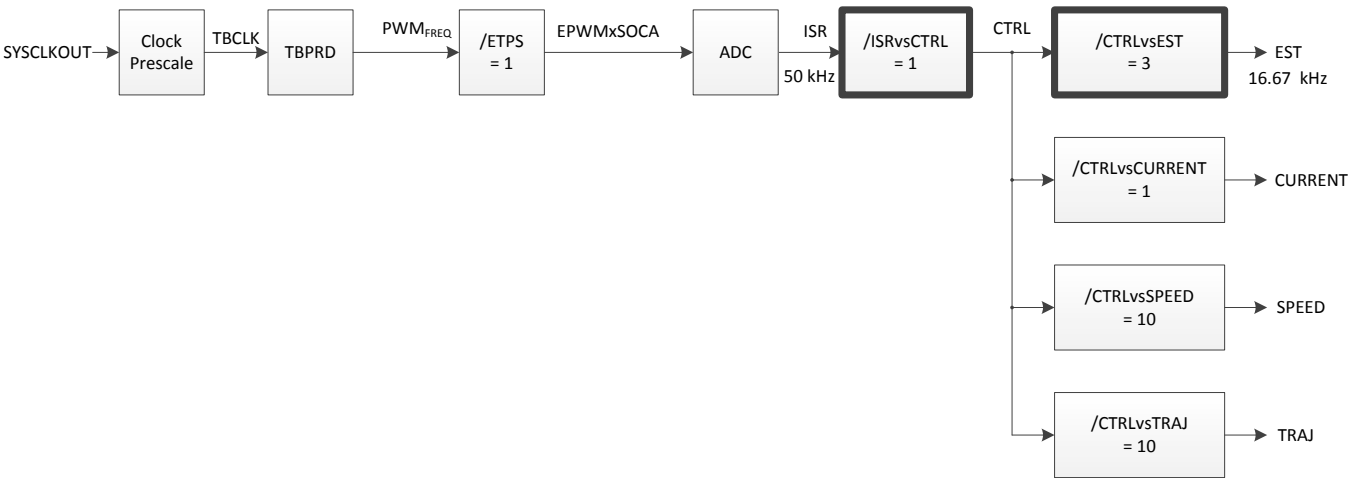


图 **9-25. CTRL vs. EST** 软件执行时钟树 **-** 节拍率 **= 3**

此问题的另一个解决方法是将 ISR vs. CTRL 节拍率改为 2。当只检查节拍计算器时，考虑 2.7µs 的 InstaSPIN 运行时间，我们

Changed 已更改18.2.3 节“设置数字 IO 以连接 QEP 外设”中的段落。 如图所示，选择值为 2 的 ISR vs. CTRL 节拍率足以避免任何转换溢出。

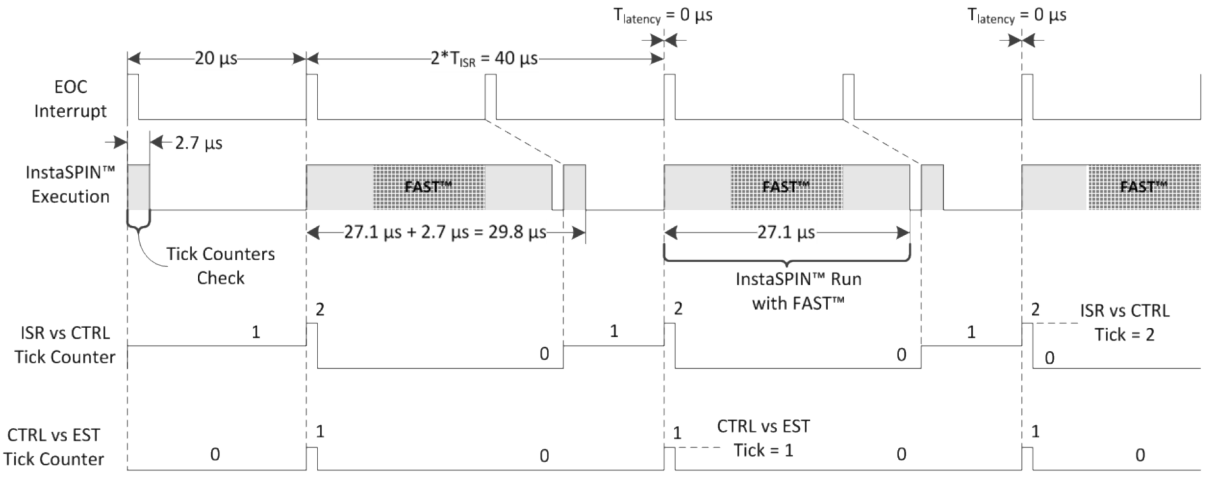


图 9-26. ISR vs. CTRL 时序 - 节拍率 = 2

在这种情况下，由于控制器 (CTRL) 节拍率值为 2，因此测量时序时必须考虑两个中断，如图所示，两个中断的运行时间长于不使用控制器加使用控制器时的执行时间，这样即可避免中断溢出。

$2 * T_{ISR} < (T_{不使用\ CTRL\ 的\ InstaSPIN} + T_{使用\ CTRL\ 的\ InstaSPIN}) \rightarrow 40\mu s > 29.8\mu s \rightarrow$ 预期的 InstaSPIN 结果

除 ISR 之外，其它任务或者其它较低优先级中断的可用 CPU 计算如下：

$2 * T_{ISR} - (T_{不使用\ CTRL\ 的\ InstaSPIN} + T_{使用\ CTRL\ 的\ InstaSPIN}) = 10.2\mu s$

CPU % 剩余 = 100% * 10.2µs / 40µs = 25.5%

2806x MIPS 剩余 = CPU % 剩余 * 最大 MIPS = 25.5 % * 90MIPS = 22.95MIPS

图 9-27 在突出显示框中显示此时序图的相应值。

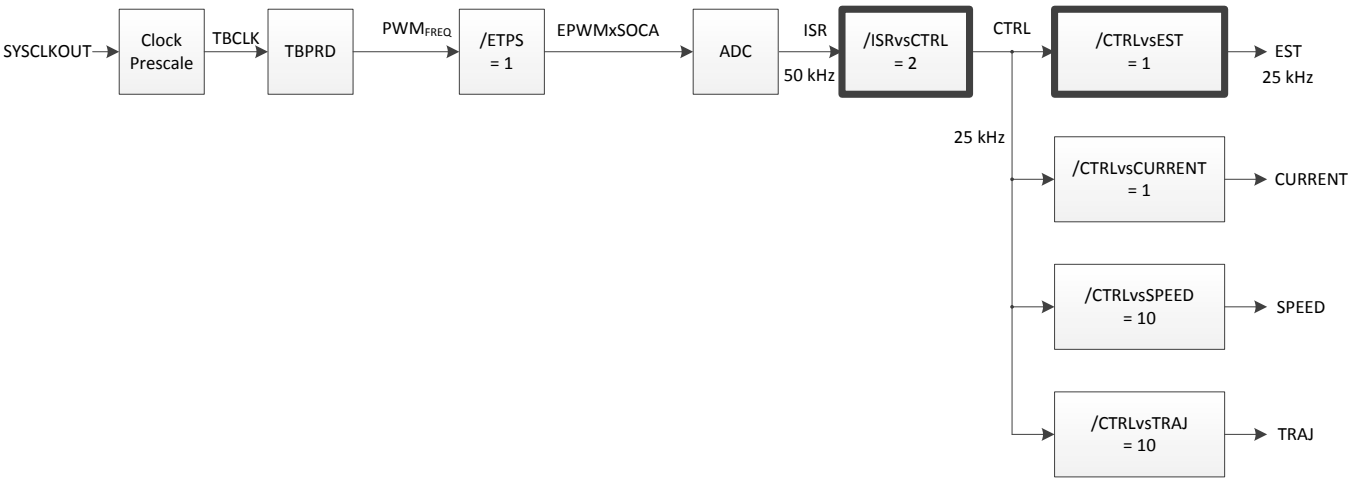


图 9-27. ISR vs. CTRL 软件执行时钟树 - 节拍率 = 2

9.2.5 USER_NUM_CTRL_TICKS_PER_SPEED_TICK

此抽取率为相对于控制器 (CTRL) 执行 InstaSPIN 内部速度控制器的速率。 速度控制器节拍率的典型值在 5 到 10 之间。 这样电流控制器便可稳定在一个比速度控制器更快的速率。 速度控制器的时间常量由耦合到电机转轴的机械负载设定，该值远小于电机电感所设定的时间常量。 以下示例显示典型值为 10 的速度控制器节拍率，时序图中显示了该值是如何由控制器 (CTRL) 抽取的。

使用典型值 10 时，电流控制器执行速度为速度控制器的 10 倍。 这种情况很常见，因为速度控制器通常设定电流控制器的基准值，并且电流控制器需要一段时间才能控制到特定设置点。

图 9-28 显示了如何使用节拍率为 10 的速度控制器。

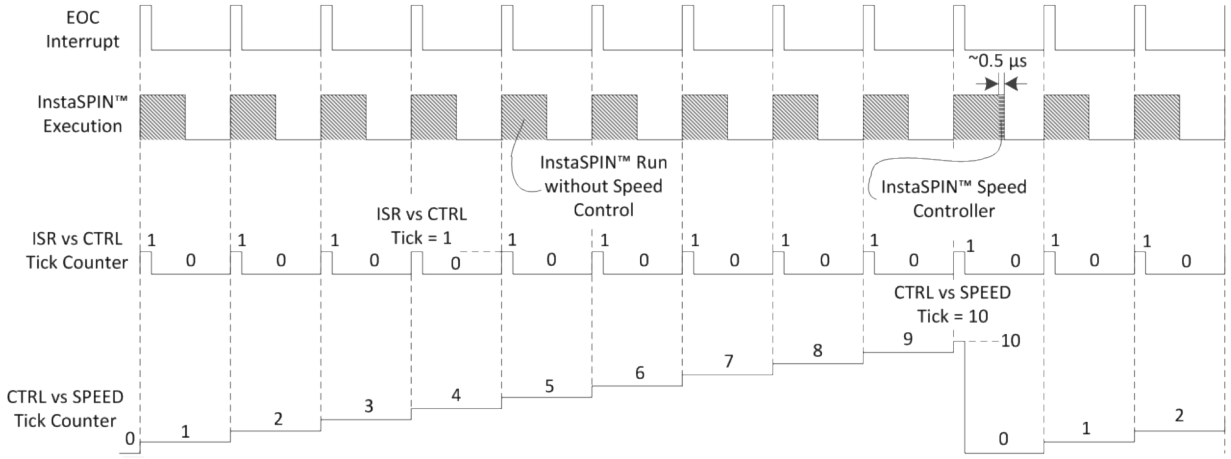


图 9-28. 速度控制器时序 - 节拍率 = 10

图 9-29 在突出显示框中显示此时序图的相应值。

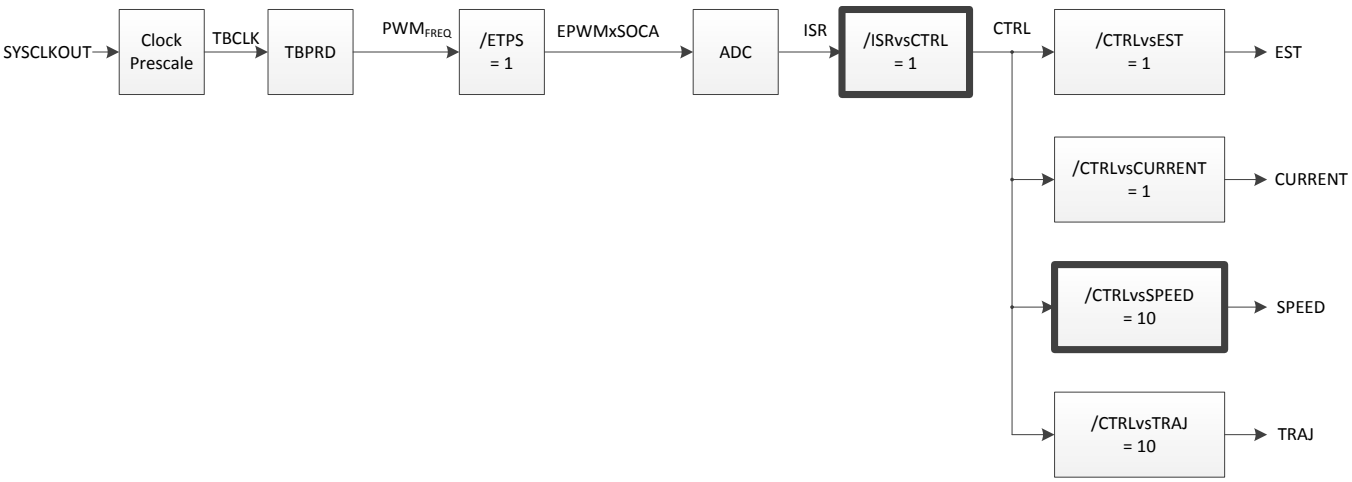


图 9-29. 速度控制器软件执行时钟树 - 节拍率 = 10

9.2.6 USER_NUM_CTRL_TICKS_PER_TRAJ_TICK

软件中的最后一个抽取率与 InstaSPIN 内的轨迹生成相关。 库中使用轨迹模块提供时序。 在库内使用轨迹的一个示例是生成速度基准的斜坡。 使用轨迹的另一个示例是电机在识别过程中加速时。 所有这些时序均由 InstaSPIN 内的轨迹完成。 所有这些时间均基于 CTRL vs. TRAJ 节拍率。 此节拍率选用不同的抽取率值对 CPU 负载没有太大帮助，因此建议匹配此节拍率的速度控制器速率（默认值为 10）。 为便于说明，图 9-30 显示了 CTRL vs TRAJ 节拍率。

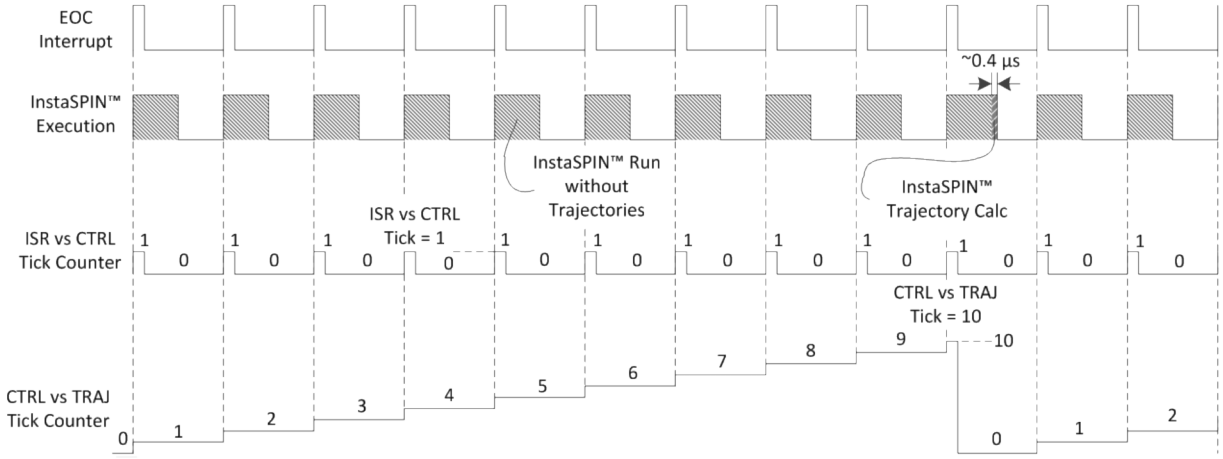


图 9-30. CTRL vs TRAJ 节拍率时序

图 9-31 在突出显示框中显示此时序图的相应值。

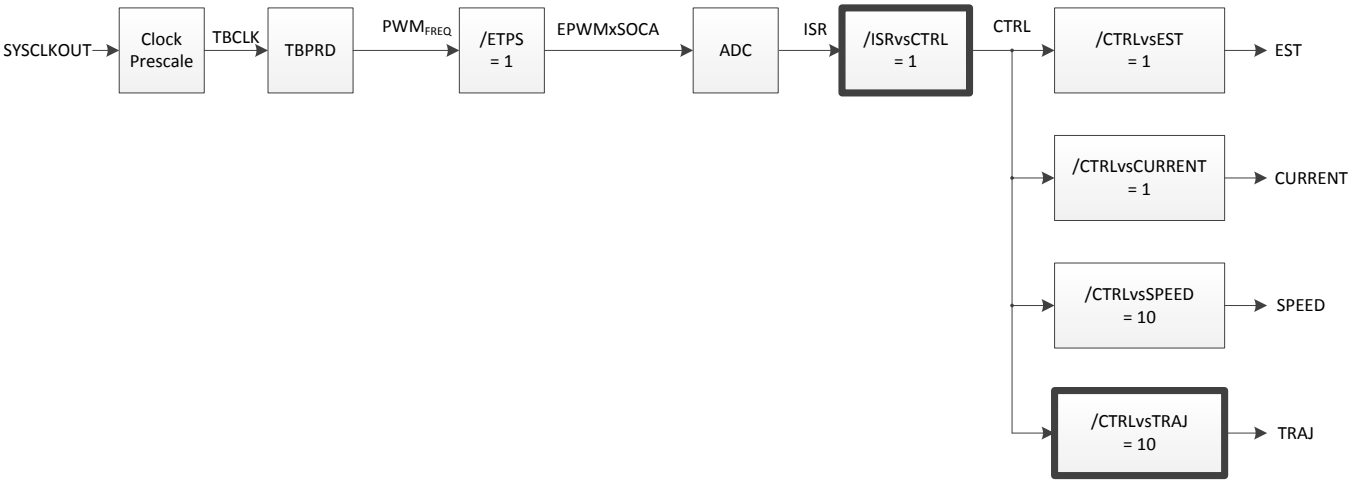


图 9-31. CTRL vs TRAJ 节拍率软件执行时钟树

所有这些节拍率及其关联性均汇总显示在图 9-32 中，并且图中引用了以下时间。

SYSCLKOUT = 90MHz

FOC（不使用 FAST 的 InstaSPIN） = 14.2µs

FAST = 12.9µs

电流控制 = 1.0µs

速度控制 = 0.5µs

轨迹运行 = 0.4µs

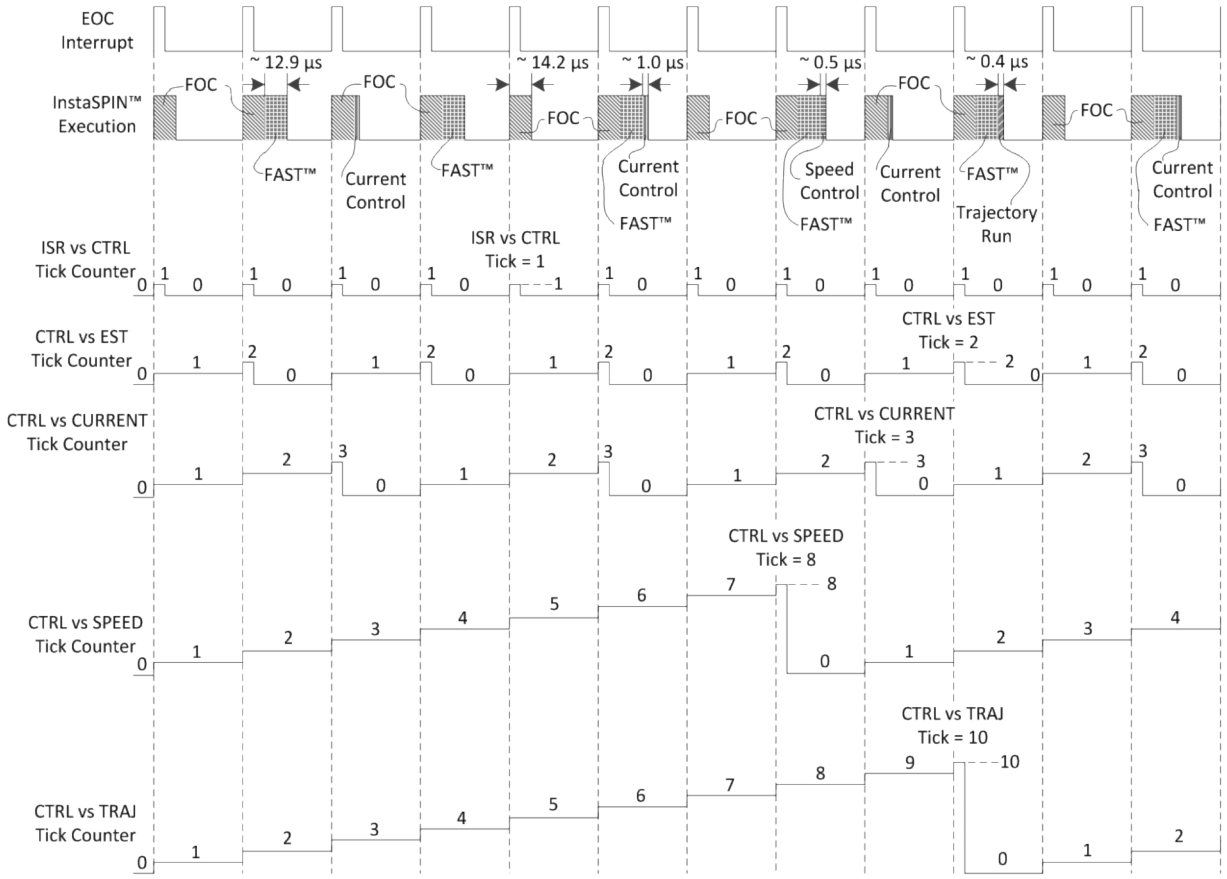


图 **9-32.** 所有节拍率和相关性时序

图 9-33 在突出显示框中显示此时序图的相应值。

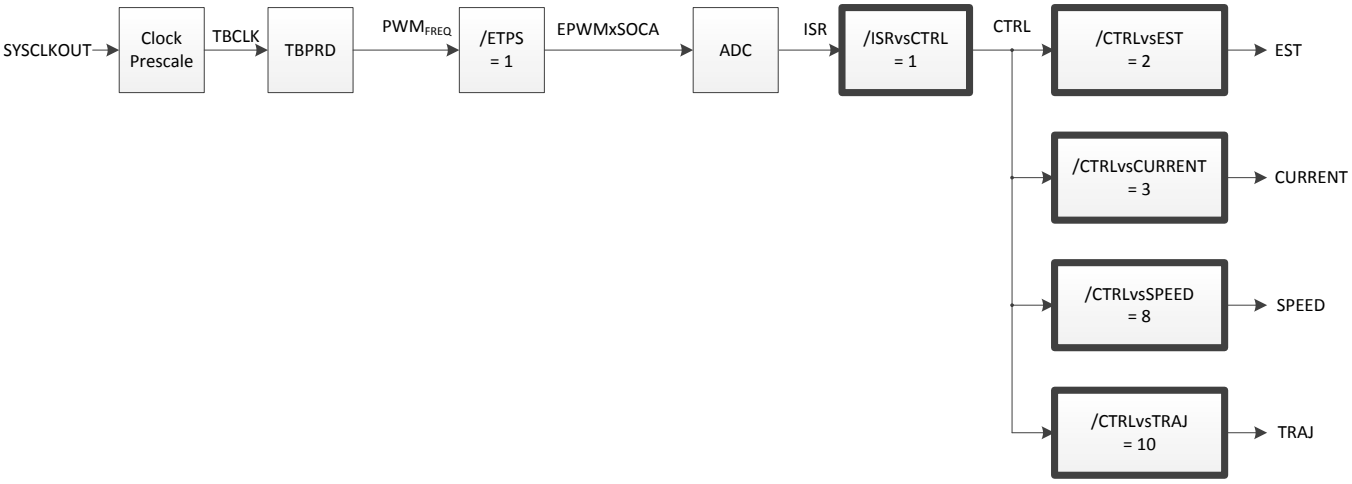


图 **9-33.** 所有节拍率和相关性软件执行时钟树

9.3 硬件抽取

图 9-34 中突出显示的节拍率用于抽取硬件中 InstaSPIN 的执行。

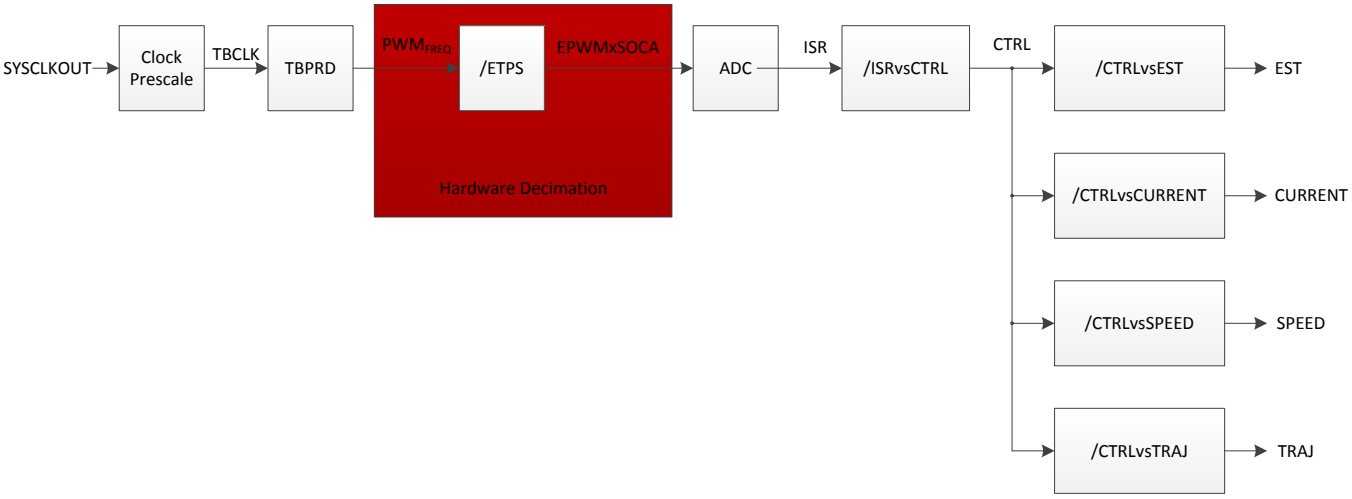


图 **9-34.** 硬件抽取软件执行时钟树

硬件抽取仅可能发生在以不同速率触发 ADC 转换时，而不是每个 PWM 周期均发生。 以下配置在 <user.h> 文件中：

```
//! \brief Defines the number of pwm clock ticks per isr clock tick
//!        Note: Valid values are 1, 2 or 3 only
#define USER_NUM_PWM_TICKS_PER_ISR_TICK        (1)
```

在上述示例中，转换开始 (SOC) 事件在每个 PWM 周期均会触发，从而得出图 9-35。

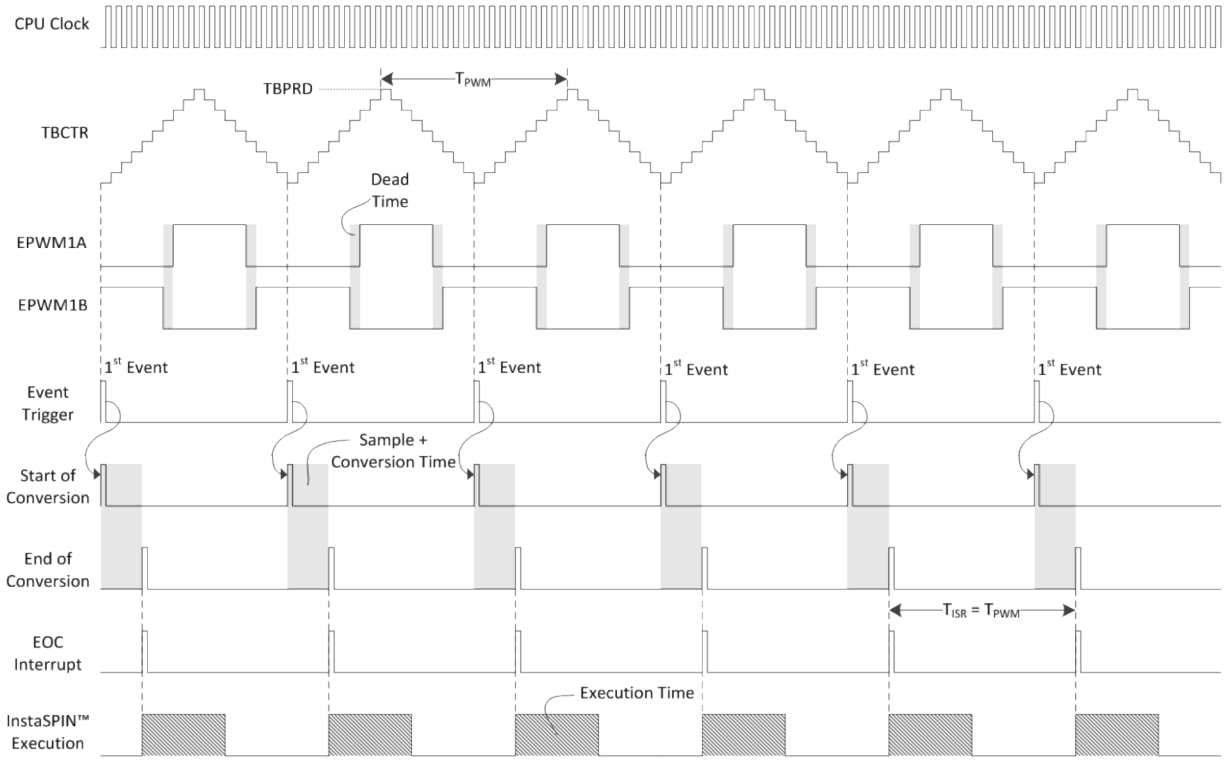


图 **9-35. SOC** 事件时序

图 11-3 在突出显示框中显示此时序图的相应值。

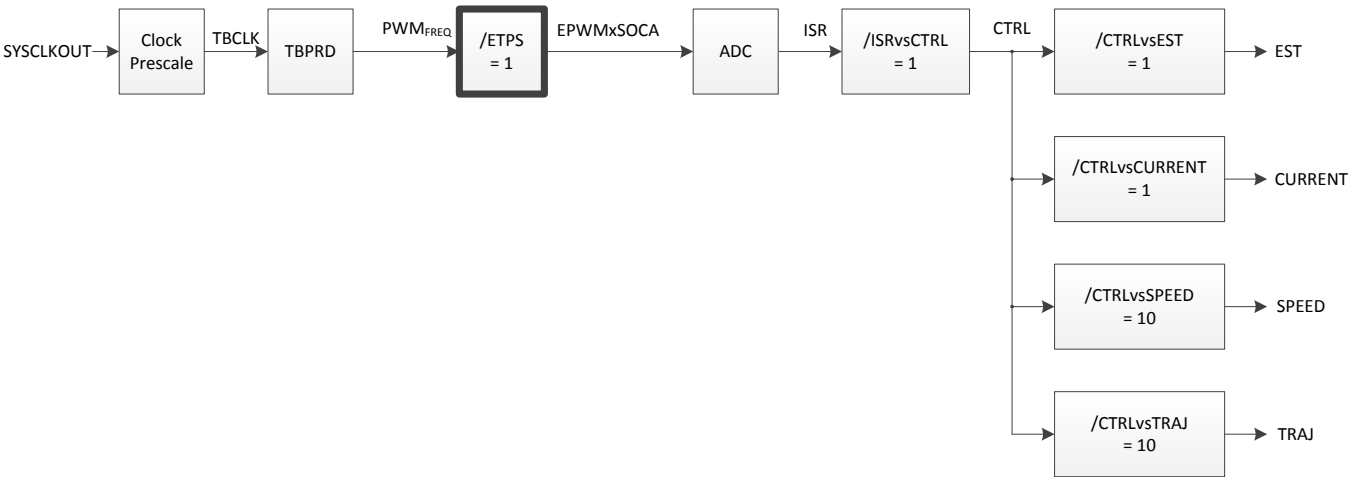


图 9-36. SOC 事件软件执行时钟树

如果应用程序要求更高的 PWM 频率，在硬件中实现此目的的一种方法是每两个或每三个 PWM 周期触发转换。 以下示例显示如何配置 PWM，从而实现每两个 PWM 周期触发转换。

```
//! \brief Defines the number of pwm clock ticks per isr clock tick
//!        Note: Valid values are 1, 2 or 3 only
#define USER_NUM_PWM_TICKS_PER_ISR_TICK        (2)
```

图 9-37 显示了相应的时序图。

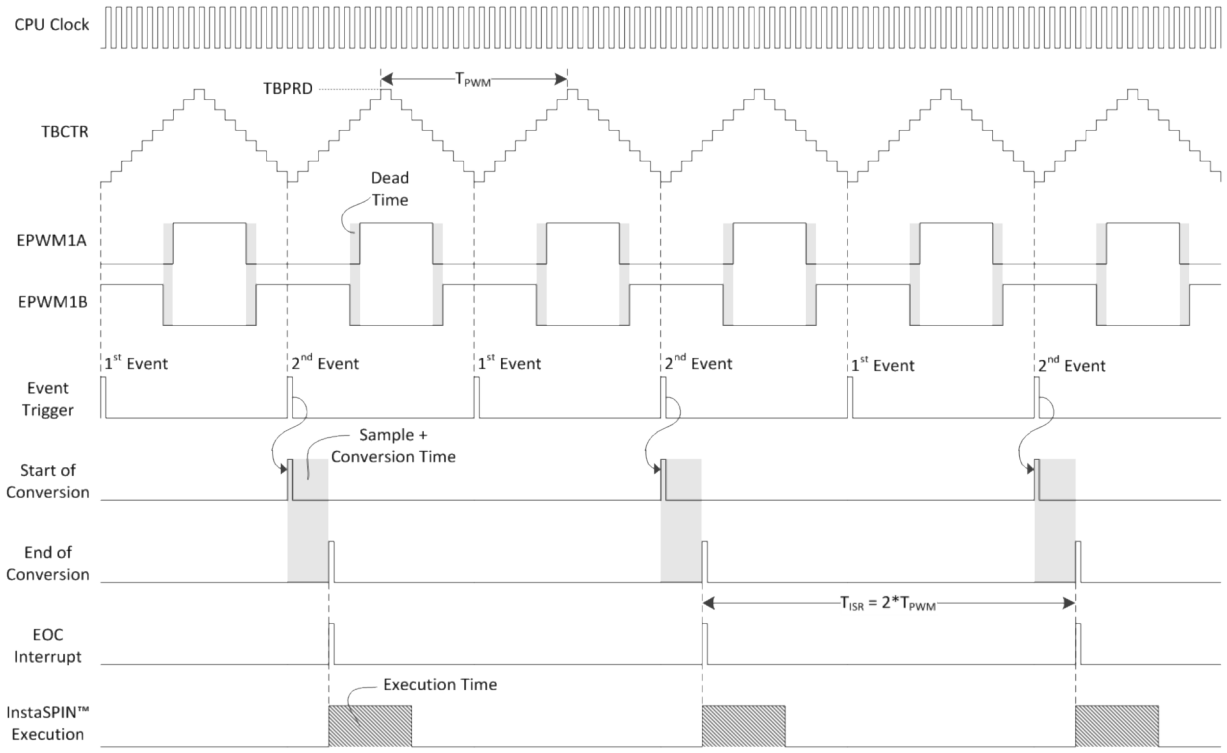


图 **9-37.** 每两个 **PWM** 周期触发 **PWM** 转换的时序

图 9-38 在突出显示框中显示此时序图的相应值。

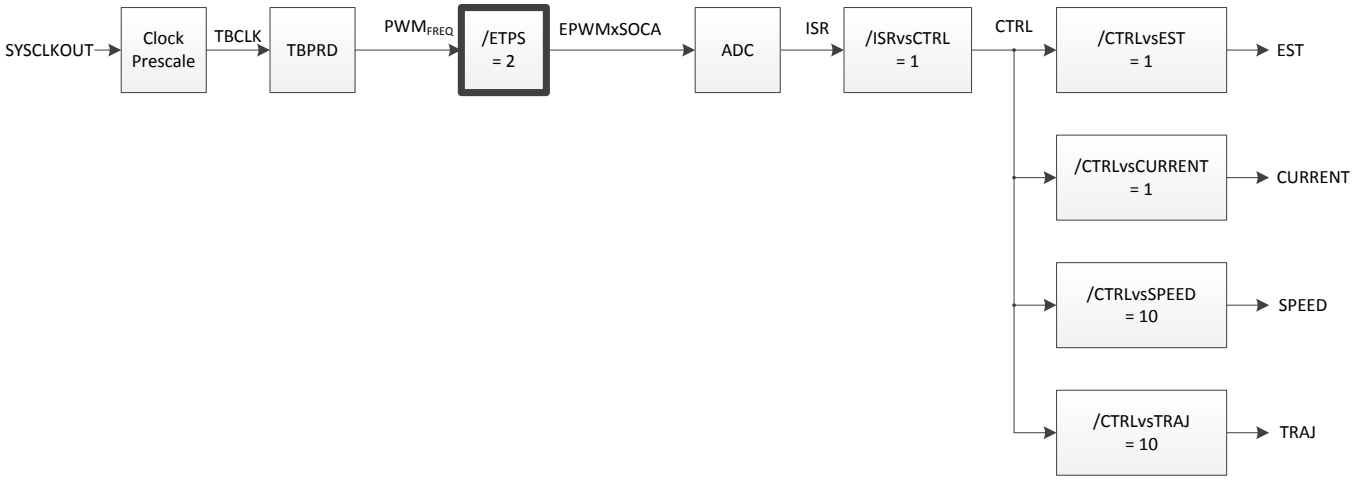


图 **9-38.** 每两个 **PWM** 周期触发 **PWM** 转换的软件执行时钟树

如果要求更高的频率，PWM 模块也可以每三个 PWM 周期触发转换，具体配置如下：

```
//! \brief Defines the number of pwm clock ticks per isr clock tick
//!        Note: Valid values are 1, 2 or 3 only
#define USER_NUM_PWM_TICKS_PER_ISR_TICK        (3)
```

节 18.3.1.2 显示了相应的时序图。

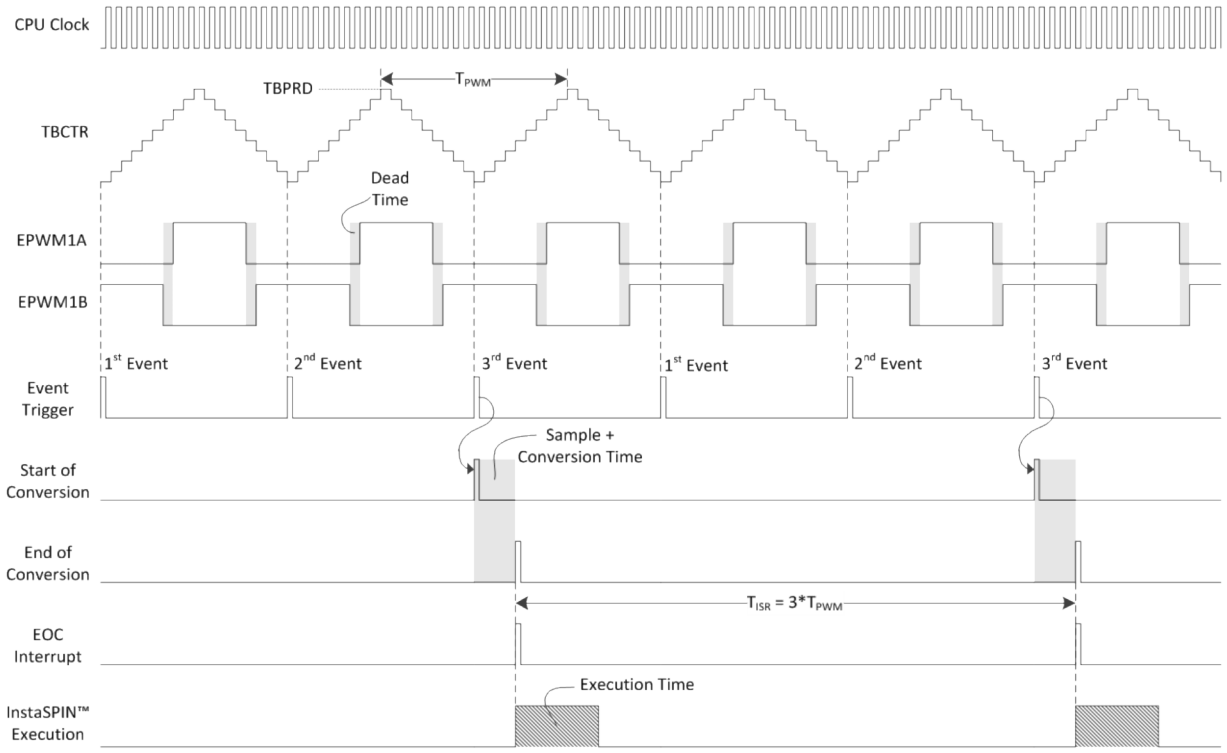


图 9-39. 每三个 PWM 周期触发 PWM 转换的时序

图 9-40 在突出显示框中显示此时序图的相应值。

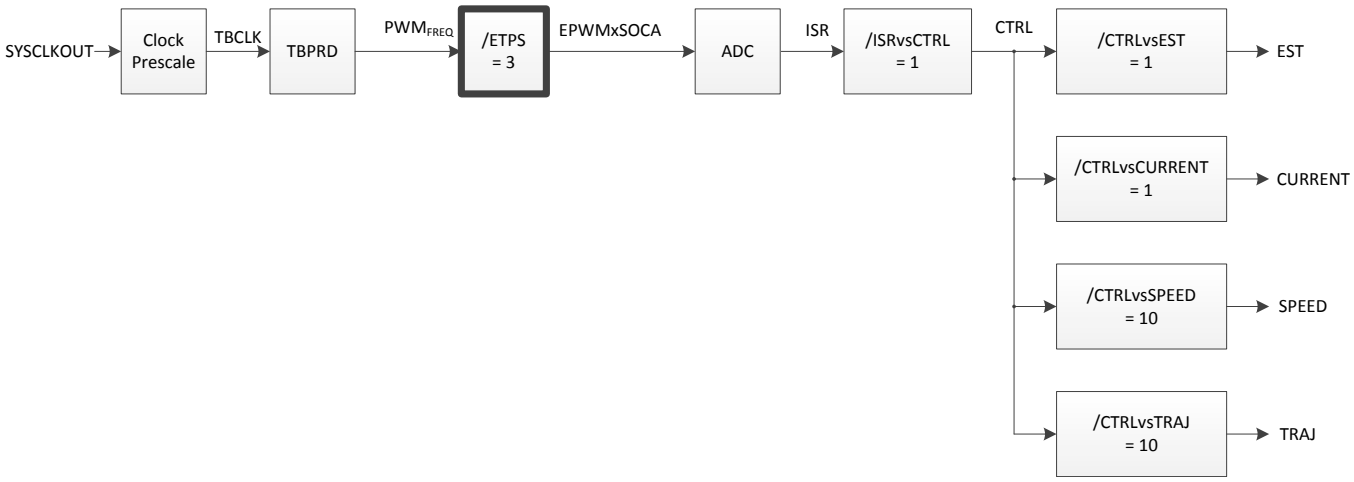


图 9-40. 每三个 PWM 周期触发 PWM 转换的软件执行时钟树

请注意中断周期是如何相对 PWM 周期发生变化的。 这样可实现更高的 PWM 频率，保持较高的中断周期。 较高的中断周期可确保 InstaSPIN 及时执行，即使 PWM 频率更高。

管理启动时间

完全识别电机或从 user.h 文件中加载电机参数后，根据启用的重校准功能，可能会存在四种启动时间。 这些重校准功能包括：

- 偏移重校准
- 定子电阻 (Rs) 重校准

这两种功能可以单独启用或禁用。 用户对不同启动方法进行实验的主要动机是为了满足相关应用的启动要求。 有关启用或禁用这些重校准功能以及配置各重校准功能的时间和电流的详细信息，请参见6.7 节。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

10.1 同时启用偏移和 Rs 重校准功能的启动

这种启动耗时最长但最准确。 在电机旋转达到指令转矩或速度基准之前，该启动包括三个阶段。图 10-1 显示了在同时启用偏移和 Rs 重校准的启动条件下的控制器和估算器状态机。

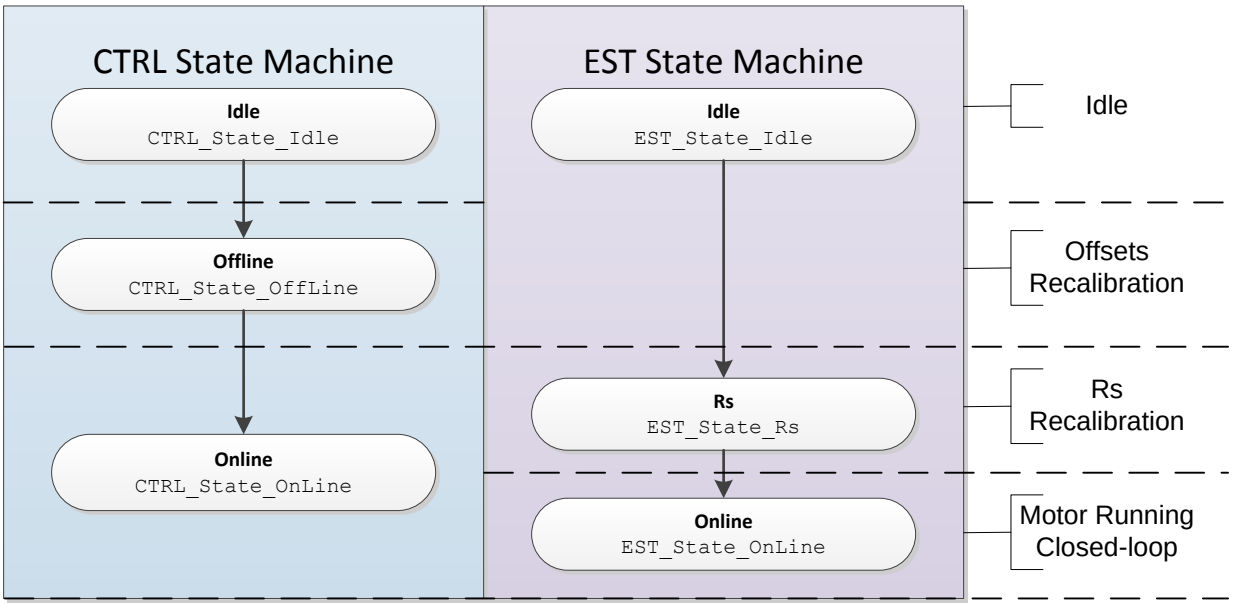


图 10-1. 同时启用偏移和 Rs 重校准功能的启动

图 10-2 显示了每个状态的电流和输出电压。 第一个状态是偏移重校准状态，第二个状态是 Rs 重校准状态。 第三个状态是在线状态，此时电机指令速度或转矩在闭环运行中跟随。

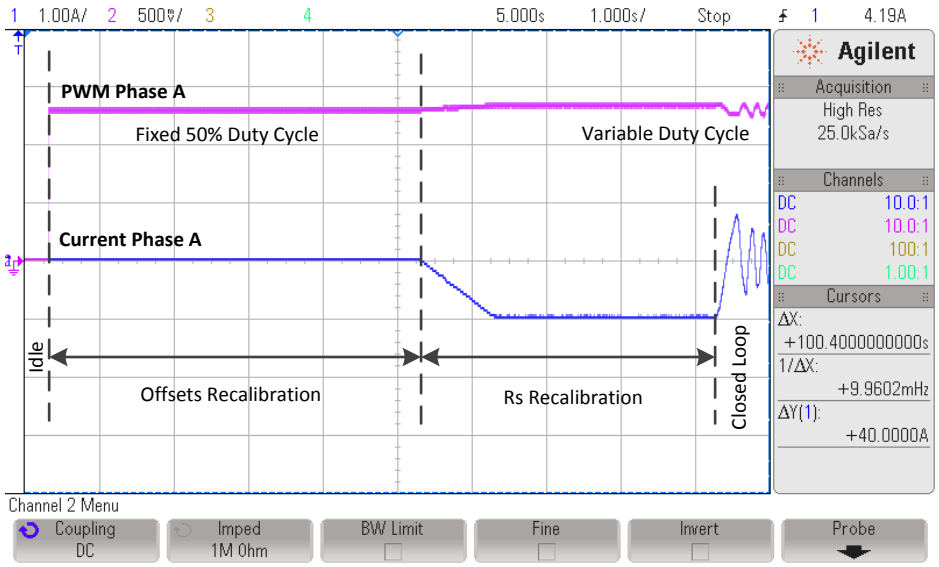


图 10-2. 各个状态的电流和输出电压

与各个状态相关的时序以及用于 Rs 重校准的电流在Chapter 6中有详细介绍。 为了同时启用偏移和 Rs 重校准，必须在启用控制器之前调用以下两个函数：

```
// Enable Offset Recalibration
CTRL_setFlag_enableOffset(handle, TRUE);

// Enable Rs Recalibration
EST_setFlag_enableRsRecalc(obj->estHandle, TRUE);
```

通过调用以下函数启用控制器：

```
// enable the controller
CTRL_setFlag_enableCtrl(ctrlHandle, TRUE);
```

10.2 仅启用偏移重校准时的启动

此启动方法禁用了 Rs 重校准，通常在偏移可能已发生更改但电机未更改的情况下使用。 使用此方法的典型情况是不同开发板上运行同一个电机。 使用此方法的另一种情况是长时间运行同一个开发板，电压和电流反馈的硬件组件值可能由于环境条件或组件耐受性而发生变化。 在第二种情况下，建议根据特定开发板中使用的硬件组件的质量来运行偏移重校准。

图 10-3 显示了进入闭环运行前仅运行偏移重校准时的状态。

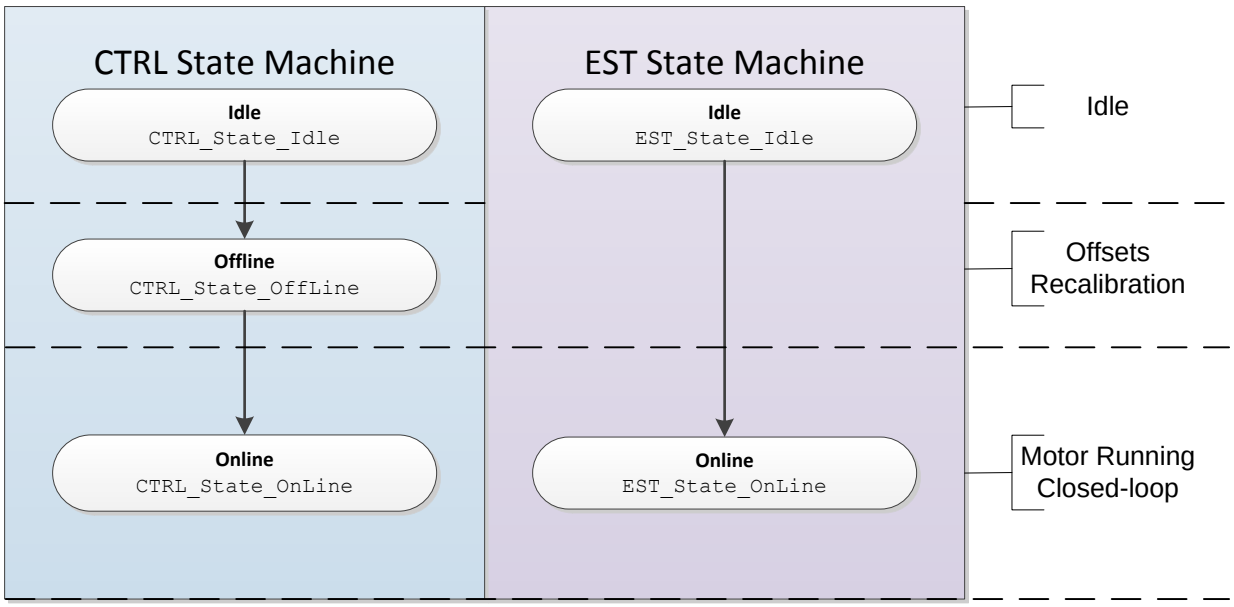


图 10-3. 仅启用偏移重校准时的启动

图 10-4 显示了与偏移状态相关的电流和输出电压波形。 在闭环运行电机前，按固定的 50% 占空比重新校准偏移。 之后，电机将进入闭环运行，此时的电压和电流取决于指令速度和机械负载。

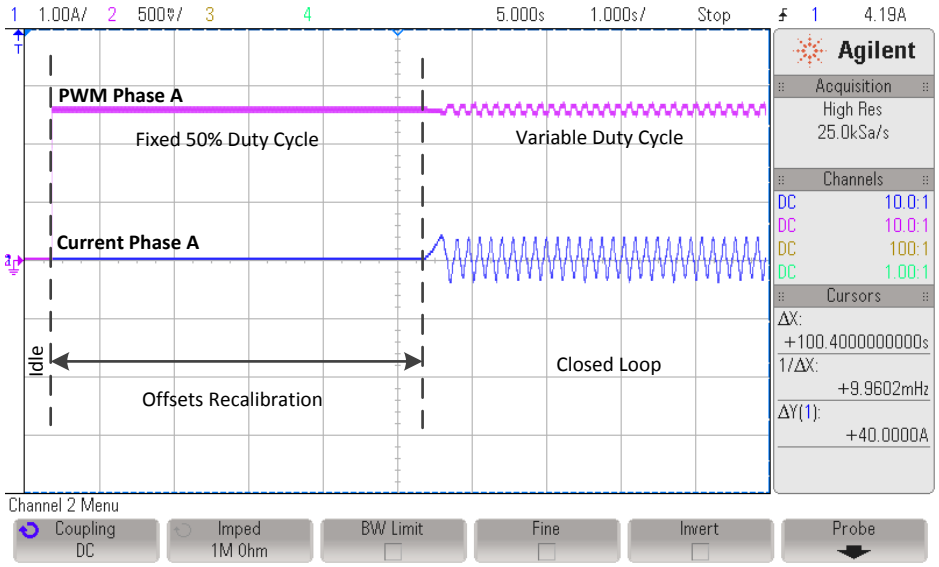


图 **10-4.** 偏移状态电流和输出电压

与偏移状态相关的时序在Chapter 6中有详细介绍。 为了启用偏移重校准同时禁用 Rs 重校准，必须在启用控制器之前调用以下两个函数：

```
// Enable Offset Recalibration
CTRL_setFlag_enableOffset(handle, TRUE);

// Disable Rs Recalibration
EST_setFlag_enableRsRecalc(obj->estHandle, FALSE);
```

通过调用以下函数启用控制器：

```
// enable the controller
CTRL_setFlag_enableCtrl(ctrlHandle, TRUE);
```

10.3 启用 Rs 重校准时的启动

此启动方法禁用了偏移重校准，通常在电阻已发生更改但偏移未更改的情况下使用。 这种情况的一个示例是环境温度发生变化导致定子电阻变化。 另一种情况是系统已在磁场中运行很长时间，建议定期更新定子电阻以确保软件在电机闭环运行前对电机型号进行准确表示。图 10-5 显示了当仅重新校准定子电阻 (Rs) 时进入闭环前的状态。

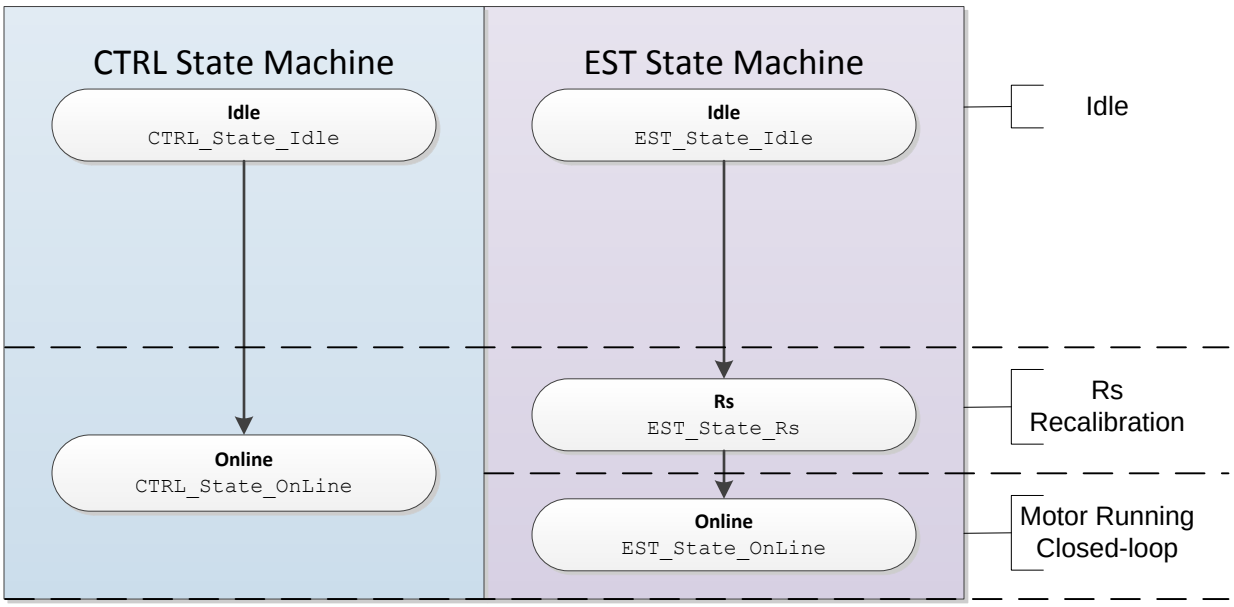


图 **10-5.** 启用 **Rs** 重校准时的启动

图 10-6 显示了进入闭环运行前仅重新校准 Rs 时的电流和输出电压波形。

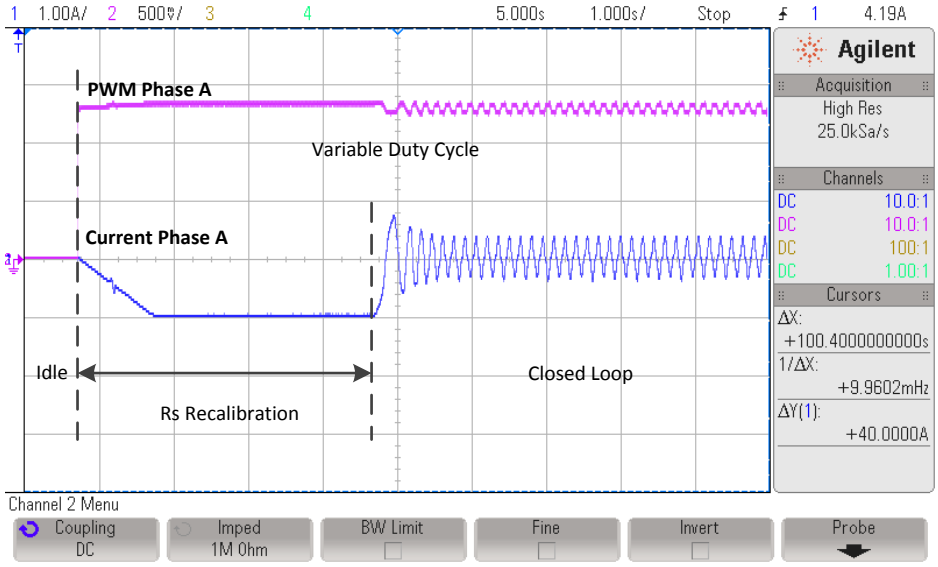


图 **10-6. Rs** 重校准电流和输出电压

与 Rs 重校准状态相关的时序以及用于 Rs 重校准的电流在Chapter 6中有详细介绍。 为了禁用偏移重校准同时启用 Rs 重校准，必须在启用控制器之前调用以下两个函数：

```
// Disable Offset Recalibration
CTRL_setFlag_enableOffset(handle, FALSE);

// Enable Rs Recalibration
EST_setFlag_enableRsRecalc(obj->estHandle, TRUE);
```

通过调用以下函数启用控制器：

```
// enable the controller
CTRL_setFlag_enableCtrl(ctrlHandle, TRUE);
```

10.4 不启用任何重校准时的启动

此启动方法是实现电机闭环运行的最快方法。 此方法不会重校准偏移或电阻。 启用控制器后，电机立即开始闭环运行。 此方法仅应在偏移和定子电阻均已知的情况下使用。 有关在启动时如何处理满载条件的详细信息，请参见Chapter 14。图 10-7 显示了在未进行任何重校准的情况下电机如何在空闲状态后立即开始闭环运行。

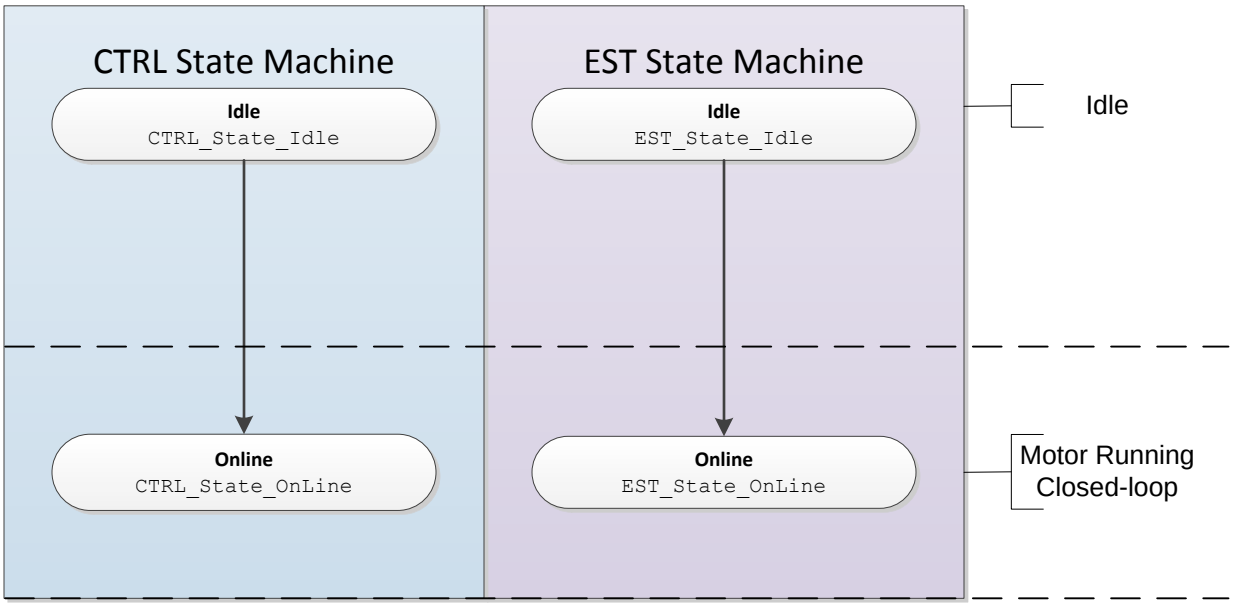


图 **10-7.** 不启用任何重校准时的启动

图 10-8 显示了不考虑偏移和 Rs 重校准时的电流和输出电压波形。

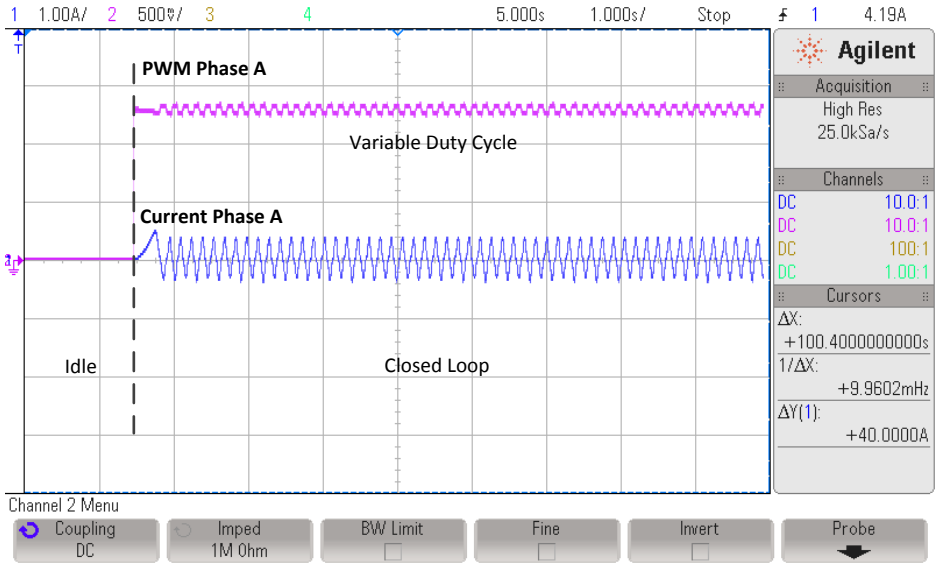


图 **10-8.** 不考虑 **Rs** 重校准时的电流和输出电压

为了同时禁用偏移和 Rs 重校准，必须在启用控制器之前调用以下两个函数：

```
// Disable Offset Recalibration
CTRL_setFlag_enableOffset(handle, FALSE);

// Disable Rs Recalibration
 EST_setFlag_enableRsRecalc(obj->estHandle, FALSE);
```

通过调用以下函数启用控制器：

```
// enable the controller
CTRL_setFlag_enableCtrl(ctrlHandle, TRUE);
```

10.5 忽略惯性估算

如果之前已估算电机惯性或电机惯性已知，则可通过忽略惯性估算过程来缩短系统启动时间。 惯性估算过程应在开发过程中完成，所得出的代表惯性与电机转轴关联。 由于电机惯性是在开发过程中配置，因此 SpinTAC 速度识别不需要包含在最终产品中。

SpinTAC 速度控制器需要使用电机惯性。 在 MotorWare 实验中，将电机惯性配置为 ST_MOTOR_INERTIA_A_PER_KRPM（位于 spintac.h 文件中）的默认值。 有关此定义的详细介绍，请参见节 4.7.1.1。 有关 SpinTAC 速度识别的详细信息，请参见Chapter 7。

如果项目中不使用 MotorWare，则可在初始化过程中使用 SpinTAC 速度控制器全局结构中的惯性参数在 SpinTAC 速度控制器中设置电机惯性。 惯性单位为 PU/(pu/s^2)。 其中，PU 是用户电流单位 [A]，而 pu/s^2 是用户加速度单位 [krpm/s]。 通常，惯性单位为 Kg*m^2 或 N*m*s^2。 用户必须将实际惯性单位转换为 SpinTAC 速度控制器使用的标度单位。

公式 26 可用于在 Kg*m^2 惯性单位和 SpinTAC 速度控制器所需的标度单位之间进行转换。 应将此公式的结果作为惯性输入 SpinTAC 速度控制器。

$$\text{Inertia}\left[\frac{\text{PU}}{\frac{\text{pu}}{\text{s}^2}}\right] = \frac{\omega_{\text{NORM}} \times 2\pi}{\varphi_{\text{EMF}} \times A_{\text{NORM}} \times \text{PP}} \times \text{Inertia}\left[\text{Kg} \times \text{m}^2\right] \quad (26)$$

在此公式中，使用以下符号：

- ω_{NORM} 定义为以 Hz 表示的频率和以 pu 表示的频率之间的比值。 该值在 user.h 中定义为 USER_IQ_FULL_SCALE_FREQ_Hz。 有关详细信息，请参见4.1.1 节。
- φ_{EMF} 定义为电机反电势，单位为韦伯。 该值在 user.h 中定义为 USER_MOTOR_RATED_FLUX。 有关详细信息，请参见4.6.7 节。
- A_{NORM} 定义为以安培表示的电流和以 PU 表示的电流之间的比值。 该值在 user.h 中定义为 USER_IQ_FULL_SCALE_CURRENT_A。 有关详细信息，请参见4.1.5 节。
- PP 定义为电机极对数。 该值在 user.h 中定义为 USER_MOTOR_NUM_POLE_PAIRS。 有关详细信息，请参见4.6.2 节。

尽管可以使用此公式计算系统惯性，但最好还是使用 SpinTAC 速度识别来估算系统惯性。 后者会考虑难以计算其惯性的对象，从而提供最准确的系统惯性值。

调整稳压器

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

11.1 PI 控制器简介

回顾二十世纪二十年代 PI 控制器诞生的历史。 二十年代早期，工程师 Nicolas Minorsky 通过观察舵手如何在不同情况下操控船舶来为美国海军设计自动转向系统。 维基百科中的资料显示，他注意到舵手的操作在静风条件下近似于对误差信号进行简单放大，但是该模型还不足以描述舵手在稳定干扰情况下（如强风）所做出的响应。 于是增加了积分项以更正连续的稳定状态误差。 之后还添加了微分项来进一步提升可控性。

维基百科还指出，他在新墨西哥号战舰上对基于 PI 控制器的自动转向系统进行了各种测试。 调整后能够将角度误差控制在两度以内， 增加微分 (D) 项后，可将误差进一步控制在六分之一度以内，这一结果优于大部分舵手手动控制所能达到的程度。 Minorsky 发表了其研究结果（也是在二十世纪二十年代早期）。 现在我们知道，他的这项发现开创了控制系统设计领域的新纪元。

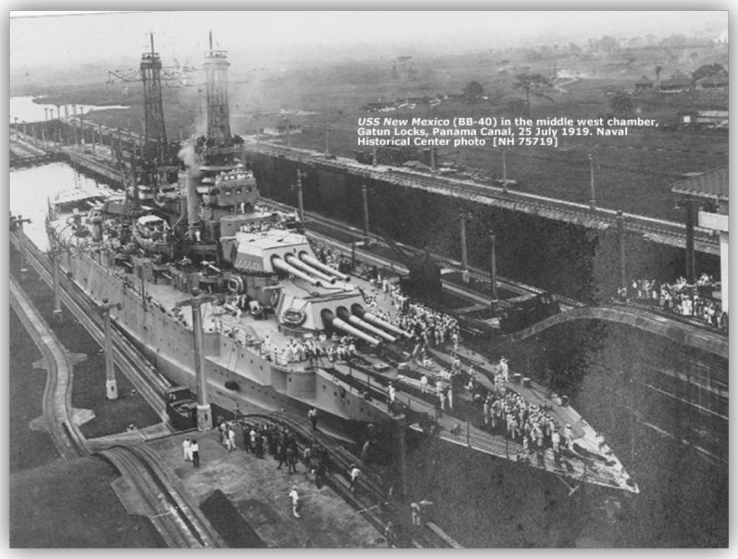


图 11-1. 当时使用比例积分微分 (PID) 控制系统改装后的新墨西哥号战舰

我们经常会在研讨会中被问道：“如何调整 PI 控制器？” 通常，我们会通过波特图或一些模拟数据来证明调整过程在某种程度上依赖于经验，并且对所需响应有非常主观的影响。

《InstaSPIN 用户指南》可集中帮助客户以更加明确的方式设计和调整 PI 控制环路（不受环路是速度环路还是电流环路影响）。 当然，用户仍有很大选择空间，具体取决于用户所需的响应以及对基本 PI 结构本身进行的各种微小变动。 但只要遵守一些基本规则（本文档中稍后将介绍），用户应能够调整 PI 环路。

该分析仅适用于只具有实极点的负载。 如果电机和负载之间的扭转谐振过大导致所研究负载具有显著的复极点，则控制器必须比简单 PI 结构更为复杂才能抵消谐振效应。 但在大多数情况下，刚性联轴器会将扭转谐振调整到可使用标准 PI 控制结构的范围内。 并且，假设负载无粘滞阻尼。 在大部分设计中这些假设均有效。 但是，如果本节中所介绍的调整过程对给定设计无效，则可能是因为负载中存在复极点或粘滞阻尼从而影响了结果。

图 11-2 显示了 PI 控制器的并联拓扑结构。 误差信号拆分进入两个独立的路径：一个直接进行放大，另一个先放大然后求积分。 所包含的积分器将系统的稳定状态误差变为零，因为任何非零的稳定状态误差都会导致积分器输出无界限。 随后，这两个信号路径在输出端通过简单的加法运算重新合并。

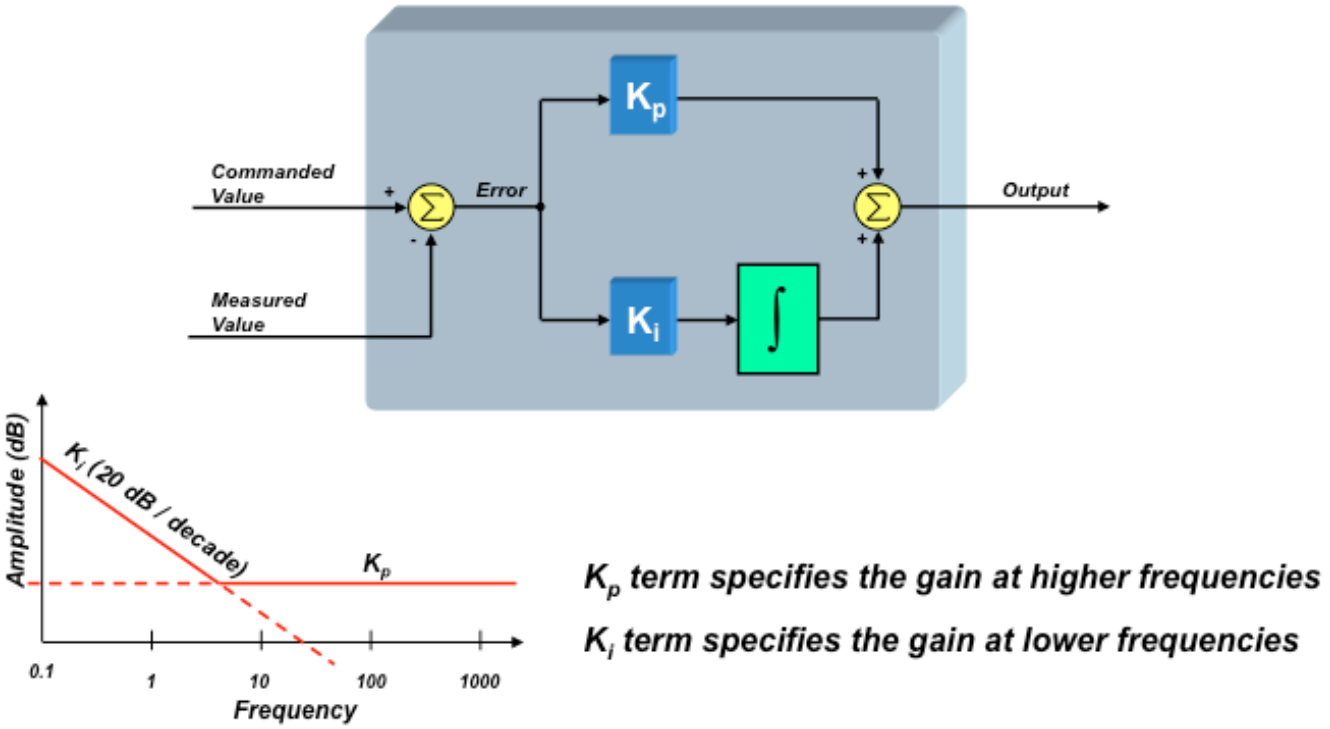


图 **11-2.** 并联拓扑结构

但是如何设置 Kp 和 Ki 的值呢？ 这一命题一直以来就备受争议，并且我们无法直观了解各项对电机控制系统所产生的影响。 Kp 项用于设定控制环路的高频增益，如上图所示。 Ki 项用于设定低频增益，理论上 DC 增益无限制。 界定高频与低频的频率被称为控制器“零点”，它对应于频率图的拐点。

尽管积分器在 PI 控制器运行过程中起重要作用，但它同时也带来了一系列挑战。 例如，假定控制环路中误差为零，这意味着受控信号等于所需信号。 现在向受控信号添加一个较小的偏移，然后观察结果。 由于误差信号不再为零，因此积分器输出将开始增大并尝试使误差信号重新为零。

现在移除偏移然后观察结果。 受控信号最终会重新变为所需值，但不是立即完成。 积分器输出仍然非常大，这会导致清除积分器输出时受控信号强烈过冲所需值。 此时，“受控”信号的系统配置将不再受控，若不加以限制甚至可能会对系统造成损害。 就好像压紧至饱和的弹簧突然得到释放一样。 这就是 PI 控制器的这一效应称为“饱和”效应的原因。 有多种方法能够减轻“饱和”效应，但大部分技术都会不同程度地限制积分器的输出。 本节中稍后将更加详细地讨论这一点。

PI 控制器的另一种常见形式是“串联”拓扑（我们将使用这种形式进行分析），如图 11-3 所示。

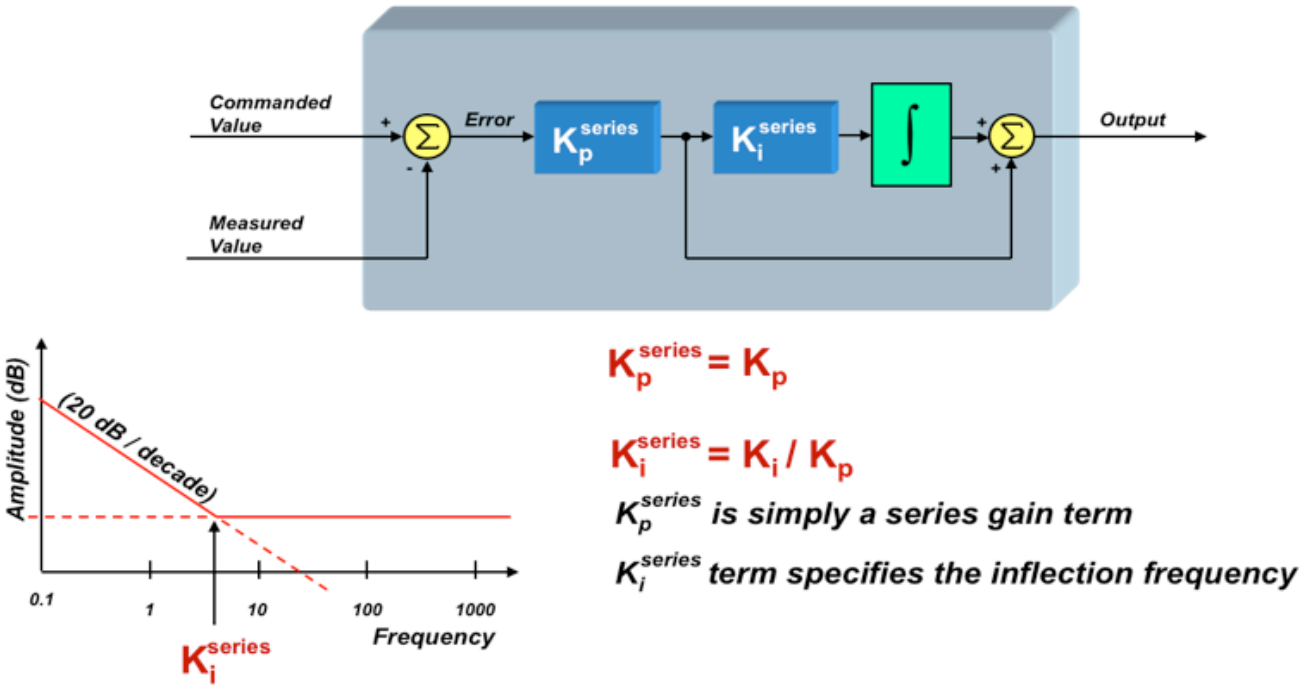


图 **11-3.** 串联拓扑结构

由上图可知：

$$K_p^{series} = K_p$$
$$K_i^{series} = \frac{K_i}{K_p} \quad (27)$$

但在该结构中，K_p^{series} 用于设定所有频率的增益，而 K_i^{series} 直接定义控制器拐点（零点），单位为 rad/s。从软件复杂程度来看，两种形式相同。 不过相对于并联拓扑，许多工程师更倾向于使用串联拓扑，原因是 K_p^{series} 和 K_i^{series} 直接与具体的系统参数相关联。 由于 K_p^{series} 只是开环传输函数的增益项，因此非常容易理解其对控制器性能所产生的影响。 但零点和拐点在系统中有什么重要意义呢？ 这部分内容将在后面进行讨论。

11.2 电流控制器的 PI 设计

上一节中我们简单回顾了 PI 控制器的历史，还介绍了现在常用的两种拓扑形式。 无论使用哪种形式，频率响应都相同，如图 11-4 所示。 从图中可以看出，PI 控制器的增益对系统稳定性有显著的影响。 但这也说明图中的拐点（“零点”频率）对系统性能起重要作用，只不过这种作用可能比较难以理解。 要弄清楚这一点，我们必须借助数学手段导出 PI 控制器的传输函数，然后了解控制器的“零点”在整个系统响应中如何发挥作用。

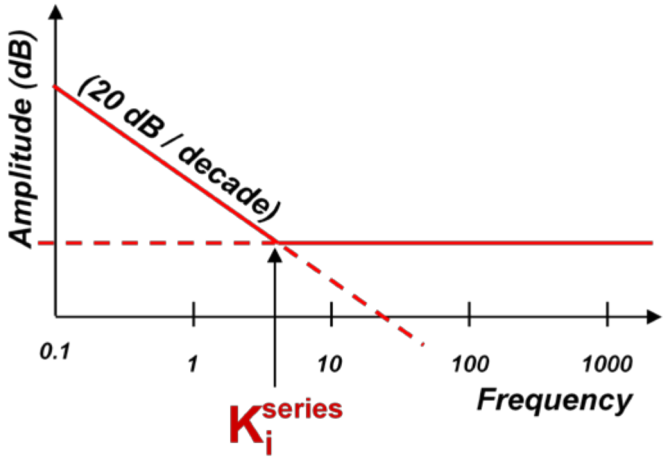


图 **11-4.** 频率响应

使用串联形式的 PI 控制器，我们可将从误差信号到控制器输出的“s 域”传输函数定义为：

$$PI(s) = \frac{K_p^{series} \times K_i^{series}}{s} + K_p^{series} = \frac{K_p^{series} \times K_i^{series}\left(1 + \frac{s}{K_i^{series}}\right)}{s} \quad (28)$$

根据该表达式，我们可以清楚地看到 s = 0 时为极点，$s = K_i^{series}\ rad/sec$ 时为零点。 那么，该零点值为什么如此重要？ 要回答这个问题，让我们将 PI 控制器放入用于调节电机电流的电流控制器，如图 11-5 所示。

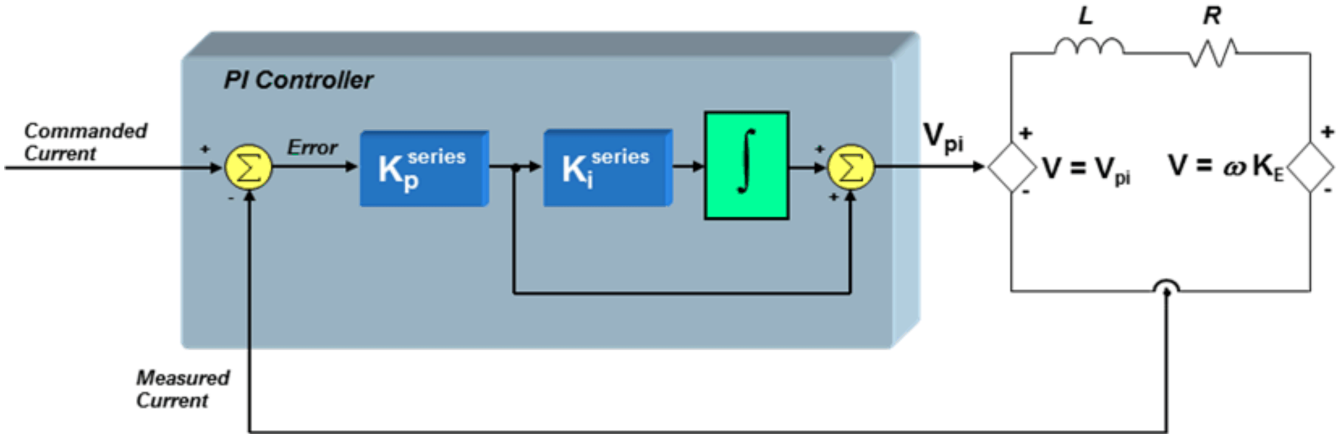


图 **11-5.** 电流控制器中的 **PI** 控制器

我们将电机绕组近似看作简单一阶串联电路，该电路中包含电阻器、电感器和反电势电压源。 假设现在反电势电压为常数（因为通常相对于电流来说反电势电压变化缓慢），我们可将从电机电压到电机电流的小信号传输函数定义为：

$$\frac{I(s)}{V(s)} = \frac{\frac{1}{R}}{\left(1 + \frac{L}{R}s\right)} \quad (29)$$

如果我们还假设 K_p^{series} 项中包括总线电压和 PWM 增益比例，则可将“环路增益”定义为 PI 控制器传输函数和 V 至 I 传输函数的结果：

$$G_{loop}(s) = PI(s) \times \frac{I(s)}{V(s)} = \left(\frac{K_p^{series} \times K_i^{series} \left(1 + \frac{s}{K_i^{series}} \right)}{s} \right) \left(\frac{\frac{1}{R}}{\left(1 + \frac{L}{R} s \right)} \right) \tag{30}$$

为了确定系统总响应（闭环增益），我们必须使用以下表达式，这些表达式您在大学控制系统课程中见到过：

$$G(s) = \frac{G_{loop}(s)}{1 + G_{loop}(s)} \quad \text{(Assuming the feedback term H(s) = 1)} \tag{31}$$

将公式 30 代入公式 31 得到：

$$G(s) = \frac{\left(\frac{K_p^{series} \times K_i^{series} \left(1 + \frac{s}{K_i^{series}} \right)}{s} \right) \left(\frac{\frac{1}{R}}{\left(1 + \frac{L}{R} s \right)} \right)}{1 + \left(\frac{K_p^{series} \times K_i^{series} \left(1 + \frac{s}{K_i^{series}} \right)}{s} \right) \left(\frac{\frac{1}{R}}{\left(1 + \frac{L}{R} s \right)} \right)} \tag{32}$$

请注意，表达式越来越复杂，但通过一些代数变换可将表达式简化为下式：

$$G(s) = \frac{\left(1 + \frac{s}{K_i^{series}} \right)}{\left(\frac{L}{K_p^{series} \times K_i^{series}} \right) s^2 + \left(\frac{R}{K_p^{series} \times K_i^{series}} + \frac{1}{K_i^{series}} \right) s + 1} \tag{33}$$

分母是“s”的二阶表达式，表示传输函数中存在两个极点。 如果我们不谨慎选择 K_p^{series} 和 K_i^{series}，则容易在最后得出复极点。 根据复极点与 jω 轴的靠近程度，系统可能会出现一些谐振峰。 因此我们立即假设选择 K_p^{series} 和 K_i^{series} 时要避免存在复极点。 也就是说，我们可以将分母表示为以下表达式，其中 C 和 D 为实数：

$$\left(\frac{L}{K_p^{series} \times K_i^{series}} \right) s^2 + \left(\frac{R}{K_p^{series} \times K_i^{series}} + \frac{1}{K_i^{series}} \right) s + 1 = (1 + Cs)(1 + Ds) \tag{34}$$

如果将方程右侧的表达式乘开，将结果与方程左侧的表达式进行比较，可以看到要想得到实极点，必须满足以下条件：

$$\frac{L}{K_p^{series} \times K_i^{series}} = C \times D \tag{35}$$

和：

$$\frac{R}{K_p^{series} \times K_i^{series}} + \frac{1}{K_i^{series}} = C \times D \tag{36}$$

作为解公式 35 和公式 36 的第一步，只需要使公式 36 两侧的各项相等。 即：

$$\frac{R}{K_p^{series} \times K_i^{series}} = C \text{ and } \frac{1}{K_i^{series}} = D \tag{37}$$

我们马上就会了解进行上述替换的原因。 如果用公式 34 中所示的等效表达式替换公式 33 的分母，然后再进行公式 37 中建议的替换，则得到：

$$G(s) = \frac{\left(1 + \frac{s}{K_i^{series}}\right)}{\left(1 + \frac{R}{K_p^{series} \times K_i^{series}} s\right)\left(1 + \frac{s}{K_i^{series}}\right)} \tag{38}$$

请注意，替换“D”可导致在闭环增益表达式中得到极点而不会产生零点。 通过正确选择 C 和 D，我们不但最终会得到实极点，而且生成的闭环系统响应只有一个实极点而无零点。 无尖峰频率响应或谐振情况，只是简单的单极点低通响应。

此外，通过将公式 37 中建议的 C 和 D 表达式替换回公式 35，可得到下列等式：

$$K_i^{series} = \frac{R}{L} \quad \checkmark \tag{39}$$

请记住，K_i^{series} 是出现控制器零点时的频率。 因此，为了得到公式 38 所描述的响应，我们需要做的是将 K_i^{series}（控制器零点频率）设置为设备极点。

现在我们已经知道如何设置 K_i^{series}。 但如何设置 K_p^{series} 呢？ 下面重写闭环系统响应 G(s) 的表达式，进行上述所有替换，将得到以下表达式：

$$G(s) = \frac{1}{\frac{L}{K_p^{series}} s + 1} \Rightarrow K_p^{series} = L \times Bandwidth \quad \checkmark \tag{40}$$

总体来说，有一些简单的规则可以帮助您设计电流环路的 PI 控制器：

K_i^{series} 可设定 PI 控制器的零点。 当控制自身传输函数中只有一个实极点的设备参数时（如电机中的电流），应将 K_i^{series} 设为该实极点的值。 这样可消除极点/零点并生成同样只有单个实极点的闭环响应。 即非常稳定、不含谐振峰的响应。

K_p^{series} 可设定闭环系统响应的带宽。 如公式 40 所示，K_p^{series} 越大，电流环路带宽越大。 我们将在后面的章节中讨论如何选择合适的带宽。 无论选择的带宽频率是多少，都可能会出现 K_p^{series} 等于感抗的情况。

在下一节中，我们将讨论如何设计将 PI 电流控制器作为内部环路的级联 PI 速度环路。

11.3 速度控制器的 PI 设计

在上一节中，我们介绍了如何计算电机电流环路控制器的 P 和 I 系数（实际上是串联结构中的 K_p^{series} 和 K_i^{series} 系数）。 我们发现 K_i^{series} 可用于消除闭环系统响应中的零点，从而使系统中仅有一个实极点（即，表现良好且稳定）。K_p^{series} 可设定闭环系统响应的带宽。

在本节中我们先来看一下包含其它 PI 控制器的速度控制环路。 速度环路的设计是否也能如此简单？ 系数值是否需要执行上一节电流控制器所使用的那些系统函数？

实际上，关闭速度环路比关闭电流环路更为复杂一些。 另外，要正确设计速度环路，还需要在电流环路的基础上了解更多的系统参数。 图 11-6 显示了构成级联速度控制环路的全部组件。 所谓“级联”是指控制系统由具有一个或者更多内部环路的外部环路组成。 需要重点强调的是，我们只考虑具有单一集中总惯性的负载与电机轴紧密连接且无粘滞阻尼的情况。

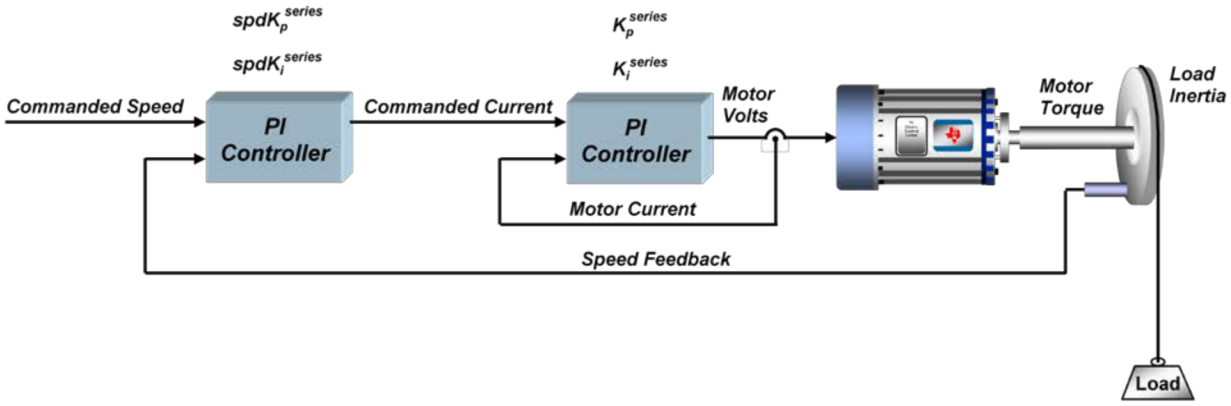


图 11-6. 级联速度控制环路

我们继续讨论上一节中的电流控制环路。 假设按照上一节讨论过的电流环路进行设计，则闭环传输函数为：

$$G_{current}(s) = \frac{1}{\frac{L}{K_p^{series}}s + 1} \tag{41}$$

其中，K_p^{series} 是电流稳压器的 PI 结构中的误差相乘项。

K_i^{series} 对函数外部不可见，因为设置该参数是为了消除电流控制器传输函数中的极点/零点。 为避免将速度控制器的系数与电流控制器的系数相混淆，我们将速度控制器的系数称为 $spdK_p^{series}$ 和 $spdK_i^{series}$，如图 11-6 所示。 如果是串联形式的 PI 控制器，则 $spdK_p^{series}$ 是误差相乘项 ($spdK_p^{series}$)，$spdK_i^{series}$ 是积分相乘项 ($spdK_i^{series}$)。 可使用与上一节相同的方程来定义速度 PI 控制器的传输函数：

$$PI_{speed}(s) = \frac{spdK_p^{series} \times spdK_i^{series}}{s} + spdK_p^{series} = \frac{spdK_p^{series} \times spdK_i^{series}\left(1 + \frac{s}{spdK_i^{series}}\right)}{s} \tag{42}$$

从电机电流到电机转矩的传输函数不尽相同，具体取决于所使用的电机类型。 对于磁场定向控制 (FOC) 下的永磁同步电机，q 轴电流与电机转矩之间的传输函数为：

$$Mtr(s) = \frac{3}{2}\frac{P}{2}\lambda_r = \frac{3}{4}P\lambda_r \tag{43}$$

其中：

P = 转子极数

λr = 转子磁通（该值还等于反电势常数 (Ke)，采用国际单位）

对于 AC 感应电机，q 轴电流与电机转矩之间的传输函数为：

$$Mtr(s) = \frac{3}{4}P\frac{Lm^2}{Lr}I_d \tag{44}$$

其中：

P = 定子极数

Lm = 磁化电感

Lr = 转子电感

I_d = 对应于转子磁通的电流分量

现在，假设所使用的是永磁同步电机。

则最终得到的从电机转矩到负载速度的负载传输函数为：

$$Load(s) = \frac{1}{J}\frac{1}{s} \tag{45}$$

其中：

J = 电机与负载的惯性

将上述各项相乘可得到复合开环传输函数：

$$GH(s) = \left(\frac{spdK_p^{series} \times spdK_i^{series}\left(1 + \frac{s}{spdK_i^{series}}\right)}{s}\right)\left(\frac{1}{\frac{L}{K_i^{series}}s + 1}\right)\left(\frac{3}{4}P\lambda_r\right)\left(\frac{1}{J}\frac{1}{s}\right) \tag{46}$$

然后将该方程后方的所有电机和负载参数整合为单个常量 K：

$$K = \frac{3P\lambda_r}{4J} \tag{47}$$

简化后得到：

$$GH(s) = \frac{K \times spsK_p^{series} \times spdK_i^{series}\left(1 + \frac{s}{spdK_i^{series}}\right)}{s^2\left(1 + \frac{L}{K_p^{series}}s\right)} \tag{48}$$

通过查看公式 48 可确定速度控制器开环传输函数的以下特征：

- 两个极点位于 s = 0 处，因此低频衰减率为每十倍频 40dB。
- 另一个极点位于 $s = \frac{K_p^{series}}{L}$ 处（电流控制器的极点）
- 一个零点位于 $s = spdK_i^{series}$ 处

为了保持稳定运行，$s = \frac{K_p^{series}}{L}$ 处极点的频率必须高于 $s = spdK_i^{series}$ 处零点的频率。 除此之外，还可以使用 $spdK_p^{series}$ 与 $spdK_i^{series}$ 的无数种组合得到不同的系统响应，具体取决于是要更高的带宽还是要更高的稳定性。 可以定义与系统稳定性成正比、与带宽成反比的单个参数，可使用该参数自动设置 $spdK_p^{series}$ 和 $spdK_i^{series}$，从而产生所选带宽下的最大相位裕度。 我们将在下一节中对此过程进行详细介绍。

11.4 根据稳定性和带宽计算 PI 增益

在上一节结束时，我们讨论了使用单个参数帮助调整电机控制系统中的速度 PI 环路的可能性。 要深入研究此参数，让我们回顾整个速度环路的开环传输函数：

$$GH(s) = \frac{K \times spsK_p^{series} \times spdK_i^{series}\left(1 + \frac{s}{spdK_i^{series}}\right)}{s^2\left(1 + \frac{L}{K_p^{series}}s\right)} \quad (49)$$

其中：

- K 是包含多个电机和负载相关项的系数
- $spdK_p^{series}$ 和 $spdK_i^{series}$ 是速度环路的 PI 系数
- L 是电机电感
- K_p^{series} 是电流环路的 PI 系数之一
- s 是拉普拉斯频率变量

假设 $s = \frac{K_p^{series}}{L}$ 处极点的频率高于 $s = spdK_i^{series}$ 处零点的频率，并且单位增益频率介于上述两个频率之间，则得到的波特图应与图 11-7 类似。

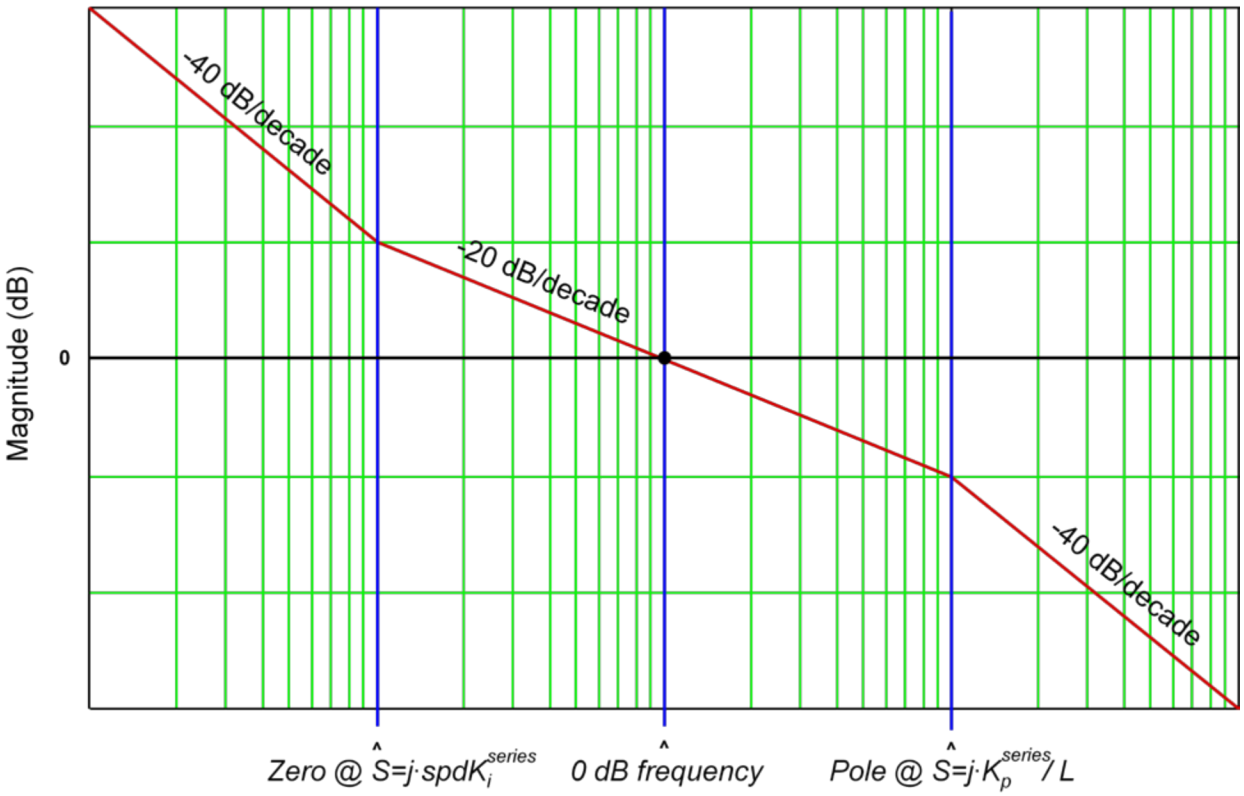


图 11-7. 波特图

曲线形状之所以重要的原因是 0dB 频率处的相移决定着系统稳定性。 在极点频率与零点频率之间分离已确定的情况下，为使相位裕度最大化（相移：180°），0dB 频率的位置应恰好在对数刻度上极点频率与零点频率之间的中点位置。 即，

$$\omega_{0\,dB} = \delta \times \omega_{zero} \quad (50)$$

和

$$\omega_{pole} = \delta \times \omega_{0\ dB} \tag{51}$$

将 节 12.3.5.6 与 公式 51 合并可得到：

$$\omega_{pole} = \delta^2 \times \omega_{zero} \tag{52}$$

通过公式 49，能够发现已在 PI 系数项中对 $\omega_{极点}$和 $\omega_{零点}$进行了定义。 因此，

$$spdK_i^{series} = \frac{K_p^{series}}{\delta^2 L} \quad \checkmark \tag{53}$$

其中，“δ”将定义为阻尼因子。 δ 值越大，零点角频率与电流环路极点频率之间的分离程度越大。 分离程度越大，两个频率之间所能够达到的相位裕度峰值越大。 这种稳定性的提升建立在牺牲速度环路带宽的基础上。 δ = 1 时，零点角频率与电流环路极点频率相等，这样能够消除极点/零点并且系统将变得不稳定。 由于相位裕度 > 0，因此理论上只要 δ > 1 时，系统都是稳定的。但是，δ 的值接近 1 会导致性能上表现出严重的欠阻尼。

稍后会对 δ 进行详细介绍，我们现在将注意力转向最后一个系数的确定：$spdK_p^{series}$。 通过节 12.3.5.6，我们发现速度环路的开环传输函数（公式 49）在频率等于零点拐点频率与 δ 的乘积时为单位增益 (0dB)。 即，

$$\left| \frac{K \times spdK_p^{series} \times spdK_i^{series} \left(1 + \frac{s}{spdK_i^{series}}\right)}{s^2 \left(1 + \frac{s}{\delta^2 \times spdK_i^{series}}\right)} \right|_{s=j\times\delta\times spdK_i^{series}} = 1 \tag{54}$$

对公式 54 中的“s”进行指定替换和整理后，得到：

$$\frac{\delta \times K \times spdK_p^{series}}{\delta^2 \left(\frac{K_p^{series}}{\delta^2 \times L}\right)} = 1 \tag{55}$$

最终可解出 $spdK_p^{series}$：

$$spdK_p^{series} = \frac{K_p^{series}}{L \times \delta \times K} = \frac{\delta \times spdK_i^{series}}{K} \quad \checkmark \tag{56}$$

现在我们来回顾一下整个过程。 我们刚刚为含有两个独立 PI 控制器的电机（一个用于内部电流环路，另一个用于外部速度环路）设计了级联速度控制器。 为了在电流环路中消除极点/零点，将 K_i^{series} 选择为：

$$K_i^{series} = \frac{R}{L} \tag{57}$$

K_p^{series} 可设定电流控制器的带宽：

$$Bandwidth = \frac{K_p^{series}}{L} \tag{58}$$

定义了内部环路电流控制器参数后，接下来选择阻尼因子 (δ) 的值，该值能够精确量化速度环路稳定性和带宽之间的权衡。 然后只需要对 $spdK_p^{series}$ 和 $spdK_i^{series}$ 进行计算：

$$spdK_i^{series} = \frac{K_p^{series}}{\delta^2 \times L} \tag{59}$$

$$spdK_p^{series} = \frac{\delta \times spdK_i^{series}}{K} \tag{60}$$

这种方法不用根据经验调整四个看似与系统性能相关性很小的 PI 系数，而是只需要定义两个有意义的系统参数：电流控制器带宽和速度环路阻尼系数。 完成上述选择后，将自动计算四个 PI 系数。

电流控制器带宽必须是有意义的系统参数，但在速度控制系统中我们通常先指定速度控制器的带宽，然后根据所指定的带宽值设置电流控制器带宽。 在下一节中，我们进一步讨论阻尼因子并找到根据所需速度环路带宽设置电流环路带宽的方法。

11.5 根据阻尼因子计算速度和电流 PI 增益

到目前为止，我们在这几节中讨论了如何简化级联速度控制器设计，即从四个 PI 系数简化为两个“系统”参数。 其中一个参数是电流控制器带宽。 另一个是阻尼因子 (δ)。 阻尼因子是代表系统稳定性与系统带宽之间权衡的一个数字。 请记住，我们只考虑仅含转矩和惯性组件（即，无扭转谐振或粘滞阻尼）的负载。 接下来我们进一步讨论时域和频域中的阻尼因子。

图 11-8 显示了电流控制器带宽随意设置为 100Hz 时系统的开环幅度和相位响应。 由于电流带宽值的作用只是为频率轴提供参考点，因此其数值并不重要。 但无论电流带宽是多少，曲线形状都不会改变。 阻尼因子在 1.5 至 70 范围内扫描，分为 8 个离散段，以显示如何对系统响应产生影响。 值为 1.0 时所对应的条件是：开环增益截距为 0dB 且刚好在电流控制器带宽的频率处。 这会导致在该频率处极点/零点消除且相位裕度为零。 很明显，相位裕度为零对系统来说并不是件好事情。

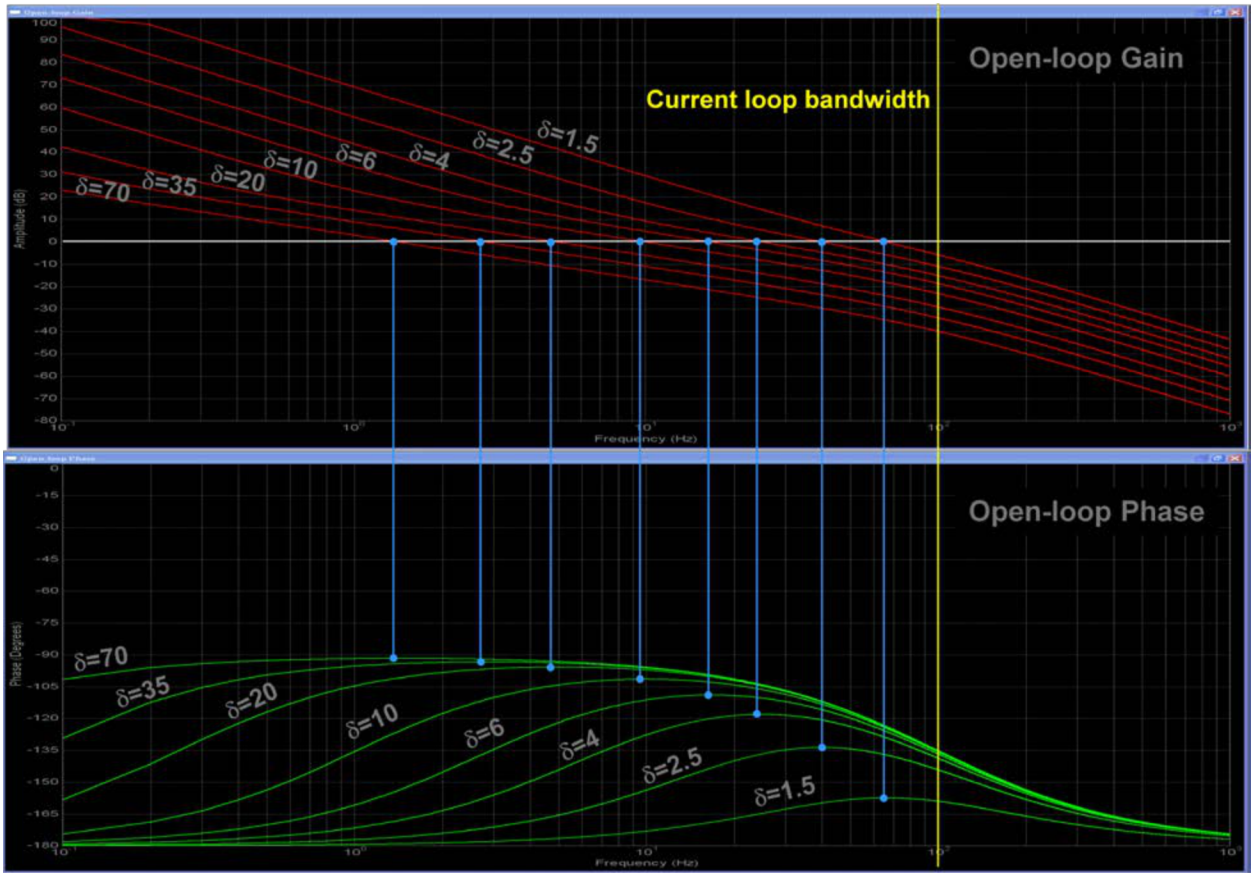


图 11-8. 速度控制器开环幅度和相位响应是 δ 的函数

阻尼因子方程的一个目标是在给定带宽条件下获得最大稳定性。 对于上图中的某个频率，当开环相位曲线中的相位裕度峰值达到自身最大值时，开环增益曲线恰好通过 0db。 随着稳定性因子增加，最终会在信号相移接近 -90 度时达到下降点。 但是，增益裕度会继续增加，其结果是大幅降低系统响应速度。

图 11-9 显示了速度环路的闭环幅度响应，仍假设电流控制器带宽为 100Hz。 与开环响应一样，实际电流控制器带宽与曲线形状不相关，其作用只是为曲线提供特定的频率参考点。

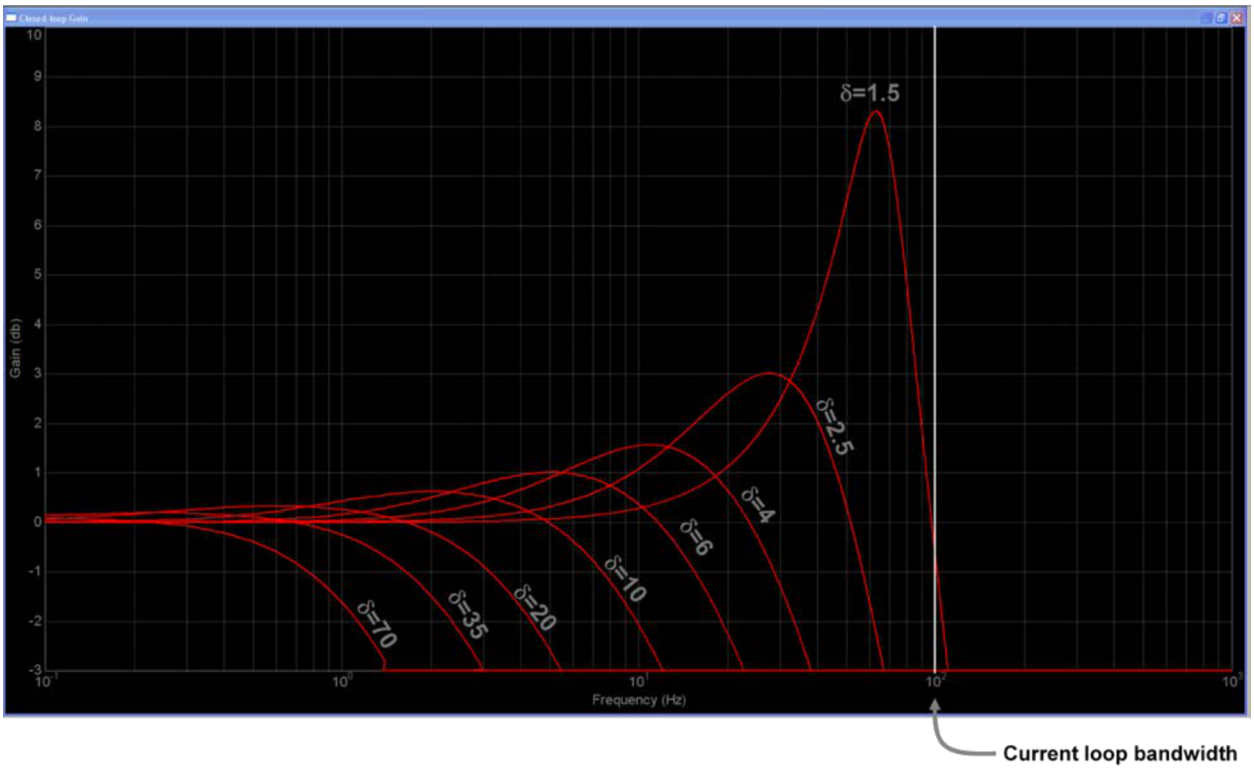


图 11-9. 速度控制器闭环带宽为 δ 的函数

速度闭环响应的 -3dB 截止点与电流控制器极点之间所需的频率分离程度可在显示各个阻尼因子值的图底部清楚地看到。 随着阻尼因子趋于一致，速度环路中的复极点将接近所需频率阶跃响应阻尼振铃。 这一点可能在图 11-10 中能更好地体现出来，图中显示了系统对各个阻尼因子值的标准化阶跃响应。 由于过冲数量过大，因此通常不接受 2 以下的值。 对于范围的上限，通常不接受远大于 30 的值，同样是因为过冲数量过大。 不能接受 30 以上的值还有一个原因是上升时间和稳定时间过长，如阶跃响应曲线所示。 通常将 2 至 30 的值设定为目标值。

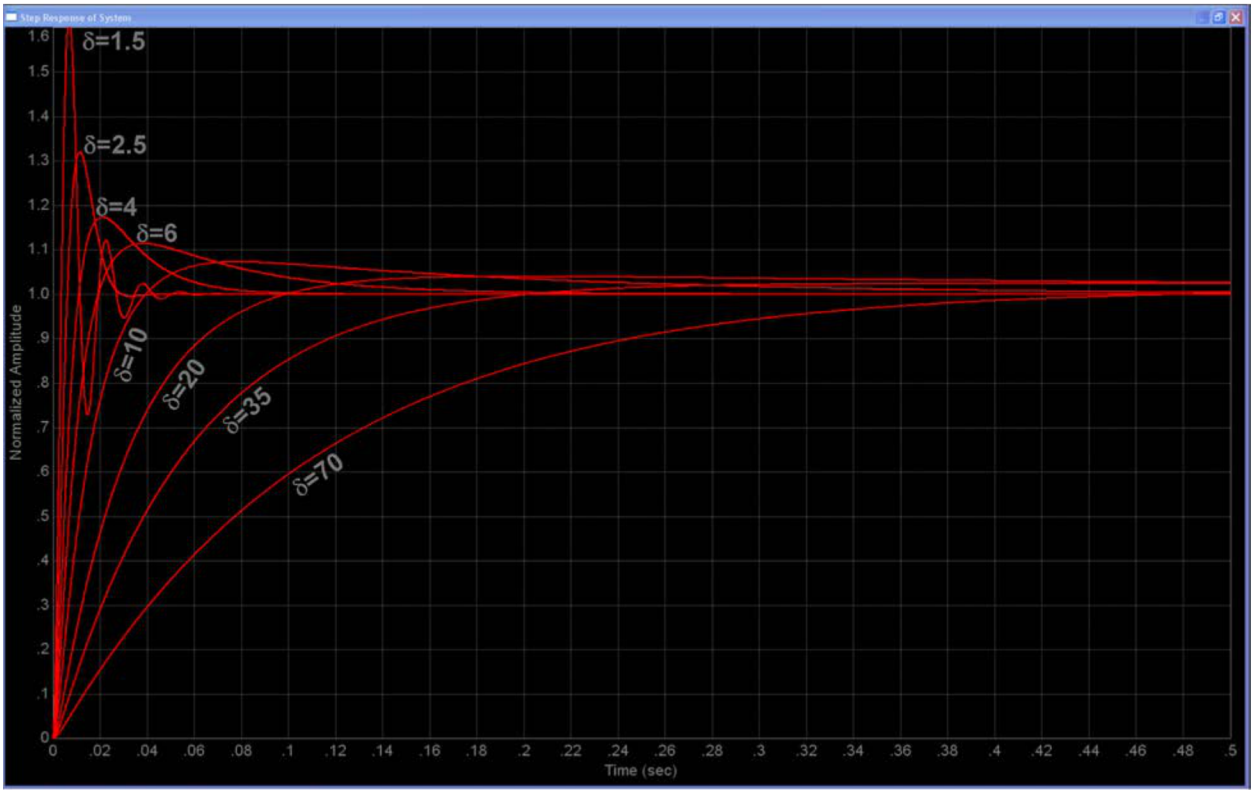


图 **11-10.** 速度控制器阶跃响应为 **δ** 的函数

如果选取的是可以接受的最小阻尼因子值，但对系统响应时间仍不满意，那么应采取哪些方法？ 最好的方法就是增加电流环路带宽。 但是这样会遇到一个问题，由于需要先确定电流环路带宽，然后才能确定给定阻尼因子条件下的速度环路带宽，因此需要反复尝试。 但是我们可以利用这样一个事实，即可将此处显示的频率曲线标准化为与电流环路带宽有关而与频率无关。 即，如果您的电机控制系统与本节中所讨论的系统类似，则无论实际参数值为何，您都能够得到类似的频率曲线（以及瞬态阶跃曲线），而不同之处仅在于频率范围（和时间范围）。 因此，我们可以利用这一事实制定一种方法，将设计过程中所需的重复次数减到最少，同时将将电流环路带宽设置为速度环路带宽的函数：

- 选择所需的速度环路频率响应（-3dB 截止频率）(BW_s)。
- 通过图 11-10 中曲线的形状确定能够产生令人满意的速度环路响应时间的阻尼因子最小值。 我们发现可以选择稍微过冲所需值的阻尼因子，这是因为最后积分器钳位会将过冲部分消除。 这时，频率范围与时间轴无关。
- 使用以下公式（通过曲线拟合分析获得）计算支持设计要求所需的电流环路带宽：

$$BW_c = \frac{K_p^{series}}{L} = BW_s\left(\delta + 2.16 \times e^{-\frac{\delta}{2.8}} - 1.86\right)(rad/sec) \quad (61)$$

其中：

- BW_c 是电流控制器带宽
- K_p^{series} = 电流环路 PI 系数之一
- L = 电机电感
- BW_s 是速度控制器带宽
- δ 是阻尼因子。

继续按照本节之前讨论的方法计算四个 PI 系数。

示例

Anaheim Automation 24V 永磁同步电机具有以下特性：

- Rs = 0.4Ω
- Ls = 0.6mH
- 反电势 = 0.0054V-s/rad（相对于中性点的峰值电压相位，在 SI 系统中也等于以韦伯为单位的磁通）
- 惯性 = 2E-4 kg-m^2
- 转子极数 = 8

所需速度带宽 = 800rad/s，阻尼因子 (δ) 为 4。确定支持速度环路带宽所需的电流环路带宽，然后计算四个 PI 系数。

解决方案

所需电流带宽可直接通过公式 61 确定：

$$BW_c = \frac{K_p^{series}}{0.6E-3} = 800\left(4 + 2.16 \times e^{-\frac{4}{2.8}} - 1.86\right) = 2126\,rad/sec \tag{62}$$

根据公式 62，我们可得出

$$K_p^{series} = 1.28 \tag{63}$$

可以得到

$$K_i^{series} = \frac{R}{L} = 667 \tag{64}$$

也可以得到

$$spdK_i^{series} = \frac{K_p^{series}}{\delta^2 \times L} = 133 \tag{65}$$

最后可以得到

$$K = \frac{3P\lambda_r}{4J} = 162 \tag{66}$$

和

$$spdK_p^{series} = \frac{K_p^{series}}{L \times \delta \times K} = 3.29 \tag{67}$$

本例中模拟得到的速度瞬态阶跃响应结果如图 11-11 所示，其中时间轴已根据此设计示例进行了适当地缩放。

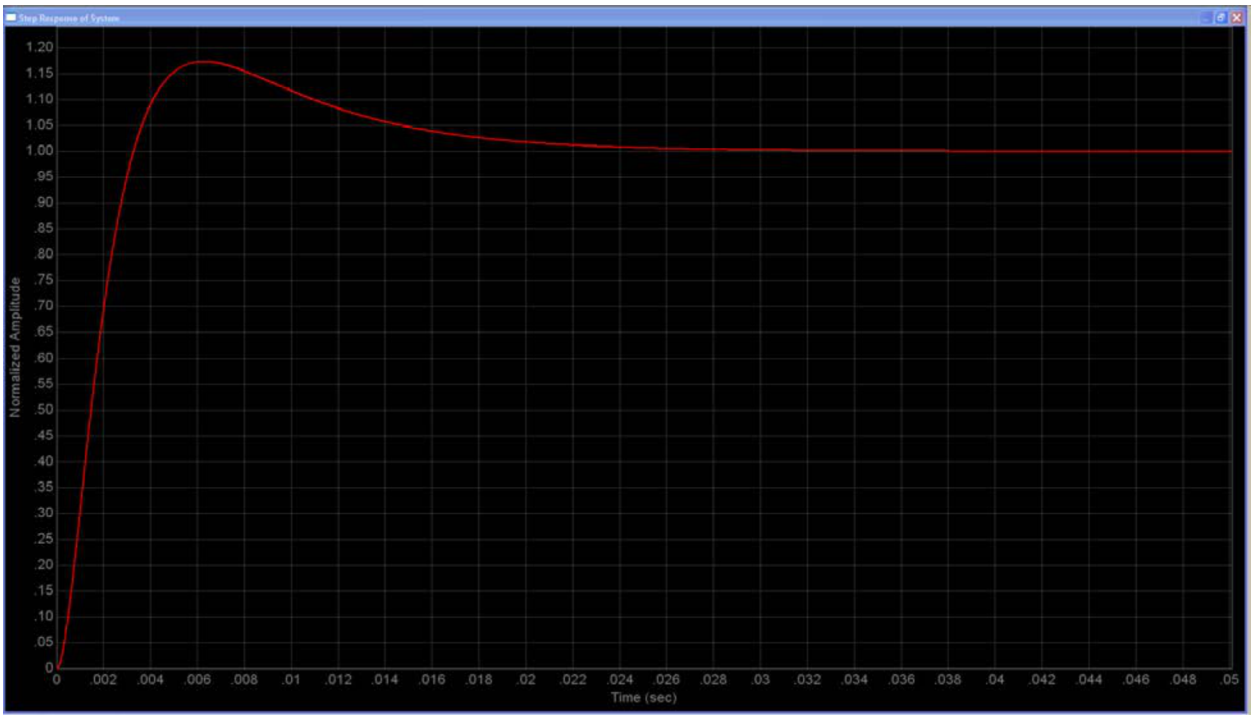


图 **11-11.** 以上示例中的速度控制器设计模拟阶跃响应

到目前为止，我们的分析假设速度环路中仅有三个极点，两个位于 s = 0 处，一个与电流控制器相关。 但如果存在其它极点怎么办？ 例如，在许多系统中速度反馈信号通常由低通滤波器控制。 那么它是如何影响调整过程的呢？ 这将在下一节中进行介绍。

11.6 向速度环路添加极点时的考量

在上一节结束的时候，我们提出了一个之前未讨论到的问题。 在所有 PI 调整部分中一直在讨论的是在理论系统中确定 PI 系数值的方法，其中速度环路的两个极点位于“s”等于零处，第三个极点来自电流控制器中。 通常，传输函数中存在一个或者多个额外的极点。 例如，与上述理想条件的最常见偏差是速度反馈信号需要进行滤波，如图 11-12 所示。

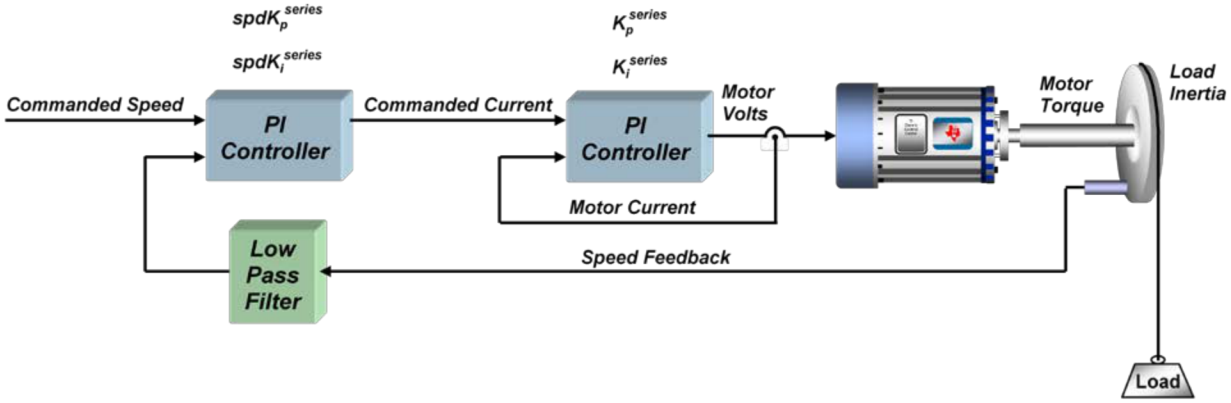


图 **11-12.** 含经过滤波的速度反馈的速度控制器

生成高质量高带宽的速度信号，并且要求设计耗时不得过多同时无需大幅增加系统成本，这确实是一项挑战。 现在已开发出能够收集编码器边沿转换信息的技术，同时使用观测器技术。 但速度信号通常还是需要进行滤波处理。 这会将速度环路的开环传输函数变为图 12-9 中所示的形式。

$$GH(s) = \frac{K \times spdK_p^{series} \times spdK_i^{series} \left(1 + \frac{s}{spdK_i^{series}}\right)}{s^2 \left(1 + \frac{L}{K_p^{series}} s\right) \times \left(1 + \frac{s}{K_{spd\_filter}}\right)} \tag{68}$$

其中：

- K 是包含多个电机和负载相关项的系数
- $spdK_p^{series}$ 和 $spdK_i^{series}$ 是速度环路的 PI 系数
- L 是电机电感
- K_p^{series} 是电流环路的 PI 系数之一
- Kspd_filter 是速度反馈滤波器的极点
- s 是拉普拉斯频率变量。

那么这会对调整过程产生什么影响呢？ 可从多个角度看待这个问题，相应地也有多种可能的解决方案。 选定的阻尼因子和极点相对位置都有助于应对上述挑战。 那么让我们来逐个应对这些挑战。

上一节中概述的方法假设根据应用要求选择合适的速度带宽和阻尼因子，然后利用步骤 3 中列出的方程计算出能够满足设计要求的所需电流控制器带宽。 但事实证明，步骤 3 中计算得出的极点可定义单位增益频率以上所有极点的最小频率。 掌握这一信息后我们就可以为 $spdK_i^{series}$ 定义更加通用的表达式：

$$spdK_i^{series} = \frac{p}{\delta^2} \tag{69}$$

其中：

p = 速度开环频率响应中 0dB 频率以上的极点最小值。

p 的值可通过电流控制器、速度滤波器或其它设备进行设置。 由于 $spdK_p^{series}$ 参考 $spdK_i^{series}$，因此其值也可能会受到影响：

$$spdK_p^{series} = \frac{\delta \times spdK_i^{series}}{K} \tag{70}$$

如果对于给定 δ，无法满足所需闭环速度带宽与 p 之间所需的频率分离程度，则需要有所取舍。 就像水球一样，握紧水球的一部分，水球上的另一部分就会凸起。 这种情况下，可以选择牺牲阻尼因子保证带宽，反之亦然。

如果电流控制器极点和速度滤波器极点的频率分离不超过半个十倍频且 δ 小于 3，则问题会更加严重。两个极点都过于接近 0dB 频率并且同时降低相位裕度，此时会出现比预期更加严重的欠阻尼响应。 例如，图 11-13 显示了系统阶跃响应，其中用于按上一节所述计算 PI 系数的阻尼因子为 2.5。 绿色曲线是假设不对速度信号进行滤波得到的曲线。 红色曲线显示的是增加速度反馈滤波器后的曲线，其中滤波器极点值等于电流控制器极点值。 系统仍然稳定，但是阻尼值 δ 远小于预期的 2.5。 此时我们有两种选择，增加阻尼因子（结果是降低速度环路频率响应速度），或将其中一个极点移至较高频率处。 蓝绿色曲线表示采用第一种

选择，将阻尼因子从 2.5 增加到 3.8 以将过冲降至原始预期值。 遗憾的是，这会导致带宽降低，由瞬态时间变长指示。 黄色曲线表示采用第二种选择，将其中一个极点值增加 3 倍（大约半个十倍频）。 这种情况下，瞬态时间几乎不会受到影响，但是阻尼效果仍然不如绿色波形。 可以继续增加极点值，在大约一个十倍频的分离时可再次获得非常接近绿色波形的响应。 但在许多情况下，移动其中一个极点会对系统造成其它负面影响。

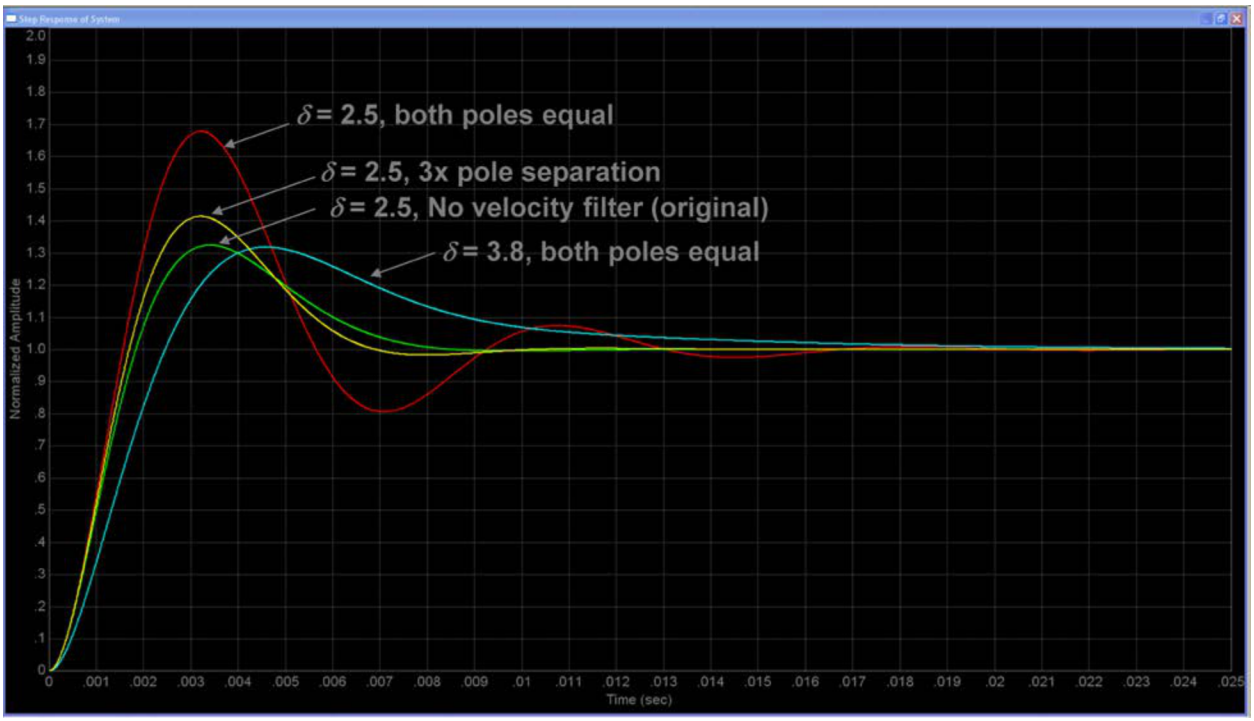


图 **11-13.** 含可变阻尼和极点配置的系统阶跃响应

到目前为止，仅处理过“小信号”条件（即，无饱和效应的线性操作）。 但实际上，阶跃瞬态响应几乎始终包含系统电压或电流级别饱和，这会延长响应时间。 当出现这种情况时，可以提高所需 PI 增益，但这无法缩短响应时间。 事实上，这通常只会加大过冲，因为积分器作用于最终会被清除的增益误差信号。 那么此问题该如何解决呢？ 是否只能使用较小的积分器增益？ 事实证明，有一种解决方法不需要更改积分器增益，我们将在下一节中介绍该方法。

11.7 速度 PI 控制器需要考虑的参数：电流限制、钳位和惯性

到目前为止，我们仅讨论了线性系统中的调整问题。 这是因为在稳定状态条件下，系统稳定后，就会发现系统在线性区域中运行且 AC 信号成分非常少。 因此，通过执行小信号（线性）分析，您可以了解未在饱和状态下运行时的稳定程度。 但通常实际情况是系统会因电压和/或电流限制而达到饱和，尤其是在高瞬态条件下。 该饱和效应将在 PI 控制器中发挥重要作用；尤其是积分器。 由于电机所能产生的最大转矩受限于电流限制，因此系统的加速度也同样受限制。 但是积分器无法感知这一点，而是认为可以通过增加输出来使电机提速。 由于系统已经饱和，因此增加的这部分积分器输出对饱和情况毫无帮助。 积分器所做的只是在系统退出饱和区后产生可导致系统过冲的极大输出。 为此，大多数 PI 积分器输出会被钳位，以防止其在系统已经饱和的情况下继续产生不必要的增量。

简单的静态钳位方案如图 11-14 所示。 最常见的情况是将钳位值设置为与 PI 输出限值相等。 例如，用于调节速度的 PI 控制器输出限值通常就是您所设置的电流限值，这是因为速度 PI 输出是电流 PI 控制器的基准输入信号。 但并不是说积分器限值必须等于 PI 输出限值，有许多设计会根据特定应用采用不同的钳位值。

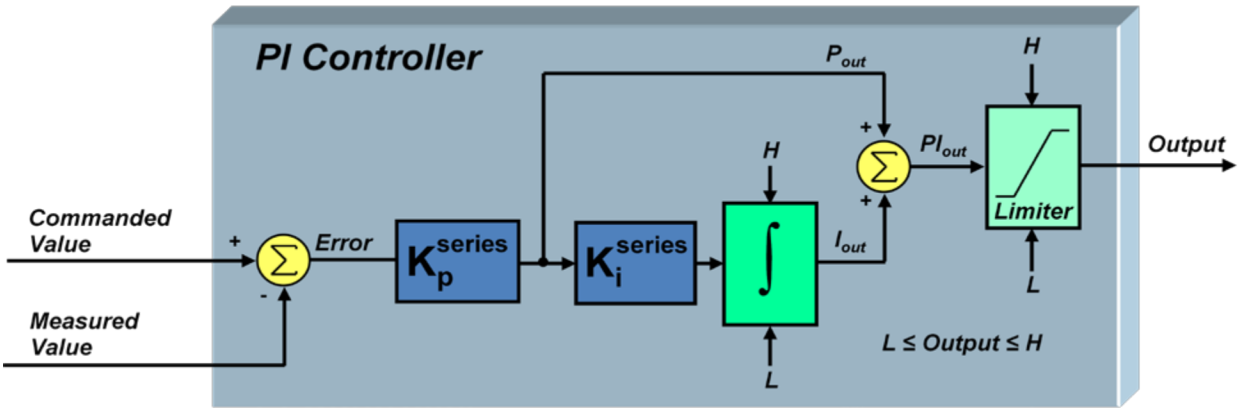


图 **11-14.** 采用静态积分器钳位的 **PI** 控制器

图 11-15 显示了性能比静态机制更为出色的动态钳位方案。 该方案的设计理念基于这样一种原理：如果系统已因 P 增益达到输出饱和，为什么还要继续进行积分？ 只有在积分器输出发生变化会导致 PI 控制器输出变化时，积分器才可以继续不受约束地对误差进行积分。

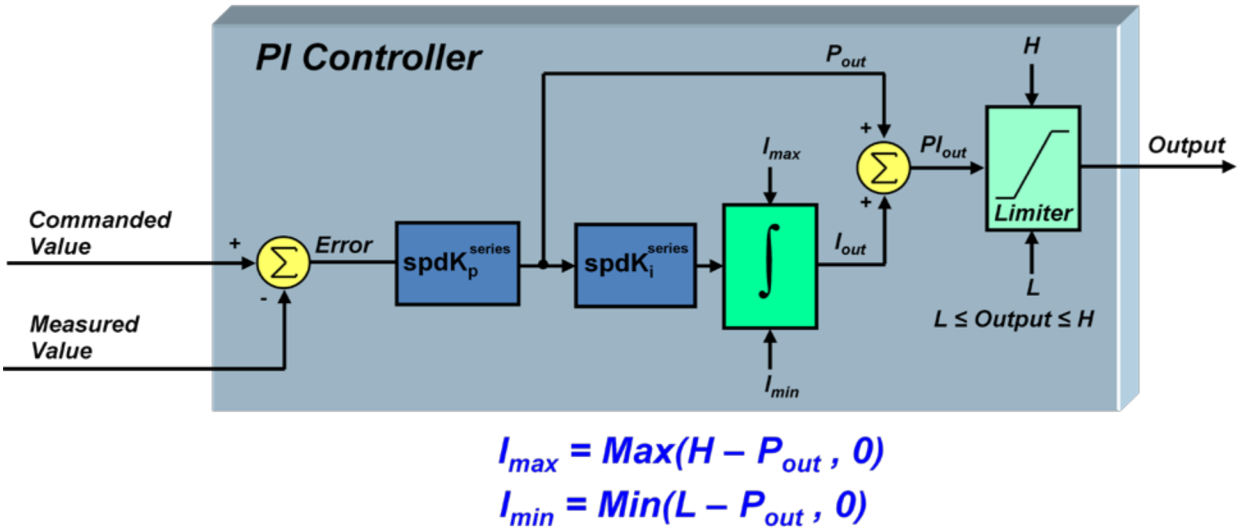


图 **11-15.** 采用动态积分器钳位的 **PI** 控制器

积分器钳位的效率如图 11-16 中的模拟曲线所示。 下面将采用所需速度阶跃从零到目标速度 (1500RPM) 模拟Chapter 5中介绍的系统。 下图所示为无钳位、静态钳位（积分器钳位值等于输出钳位值）和动态钳位条件下的系统过冲效果。 可以看到，无积分器钳位的结果完全不能接受，因为这种条件下过冲极其严重，会触发进一步系统饱和与震荡。 静态积分器钳位极大地改善了这一情况。 但动态钳位能够进一步提升系统性能，其过冲峰值仅为本例中静态钳位过冲峰值的六分之一。

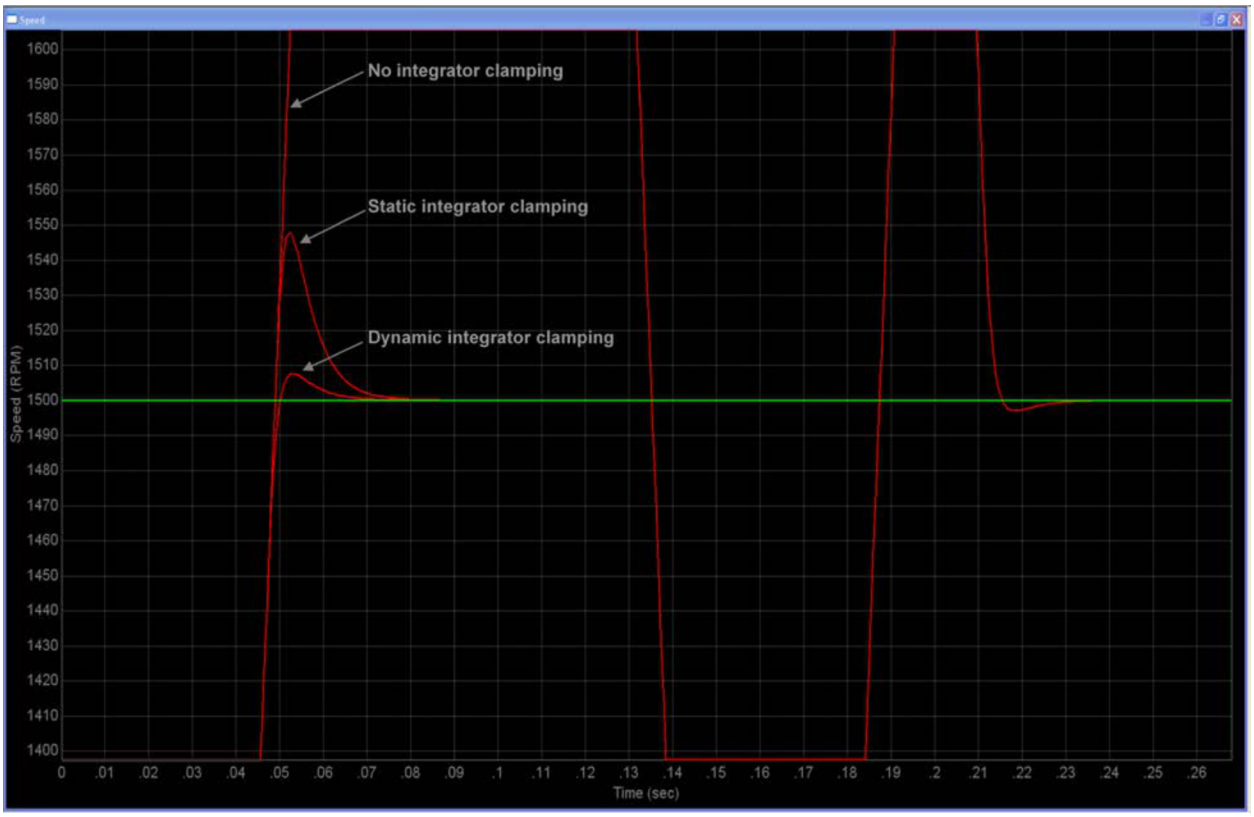


图 **11-16.** 积分器钳位技术示例比较

现在来回顾并讨论一下整个讨论中极为重要的一部分。 如果不知道系统中的关键要素 - 惯性，则这七节中所讨论的所有内容都没有意义。 不了解惯性就无法稳定速度环路。 通常，可通过旋转负载的外形尺寸和质量分布来计算惯性。 如果电机轴上有减速机且齿轮速比足够大，那么由于传输惯性与匝数比的平方成反比，因此负载惯性通常可忽略，只需要处理大多数数据表中所列出的电机惯性。 如果上述选项均无效，还可使用几种技术（通常涉及某种类型的控制加速度、减速度或二者）来测量惯性。 但是，将电机轴上的静态转矩负载纳入考虑的技术并不常见（文中的“静态”负载是指不随时间变化的负载，如摩擦力或升降机负载）。 建议使用以下技术（但截至撰稿时未经测试），该技术的惯性估算效果应优于上述技术：

1. 使用 PI 调整部分中讨论的技术设计电流控制器。
2. 将速度环路的 PI 系数设定为保守值，仅允许电机进行加速（即，此时不关注动态响应迟缓的问题）。
3. 将电机加速至低速状态并使其能够保持稳定（以使惯性转矩为零）。 然后读取平均电机转矩（图 11-17）。
4. 在速度较大时重复步骤 3 并生成平均转矩读数（为速度的函数）图（图 1）。 记录最高速度设置所需的平均电流。 然后关闭电机，使电机停止工作。
5. 禁用速度环路，仅使用电流模式，向电机施加步骤 4 中电流的 1.2 倍至 1.5 倍的电流。 当速度达到步骤 4 中所记录转矩值所对应的速度时，重新记录转矩（图 2）并获取时间戳。
6. 从图 2 中减去图 1（得到的仅为加速转矩）（图 3）。
7. 对于图 3 中的各个点，计算目标点前后的点的增量速度和增量时间。 用增量速度除以增量时间得到该点的局部加速度值。
8. 针对图 3 中的各个转矩值，将其除以步骤 7 中对应点的局部加速度以生成惯性 (J) 曲线图（速度的函数）。
9. 计算出不同速度下的平均惯性值以获得系统惯性的单一估算量。

可预先在测功机试验台上完成该过程，如果可通过控制算法测量转矩（如 InstaSPIN-FOC 的转矩输出），也可将该过程作为目标应用电机调试过程的一部分来执行。

到目前为止，我们仅讨论了 PI 调整中独立于控制拓扑的一般情况。 下一节将介绍设计用于磁场定向控制 (FOC) 系统的 PI 控制器时需要注意的一些细节。

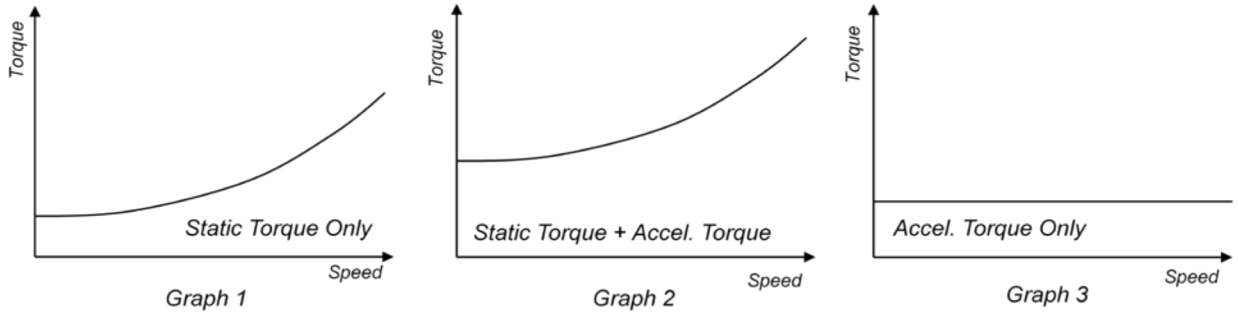


图 **11-17.** 平均电机转矩读数

11.8 为 FOC 系统设计 PI 控制器时的注意事项

下面看一下到目前为止我们针对磁场定向控制 (FOC) 系统所讨论过的关于 PI 调整的各个主题。图 11-18 显示了采用三个 PI 控制器的典型磁场定向系统：两个用于控制电流的正交分量，一个用于控制速度。

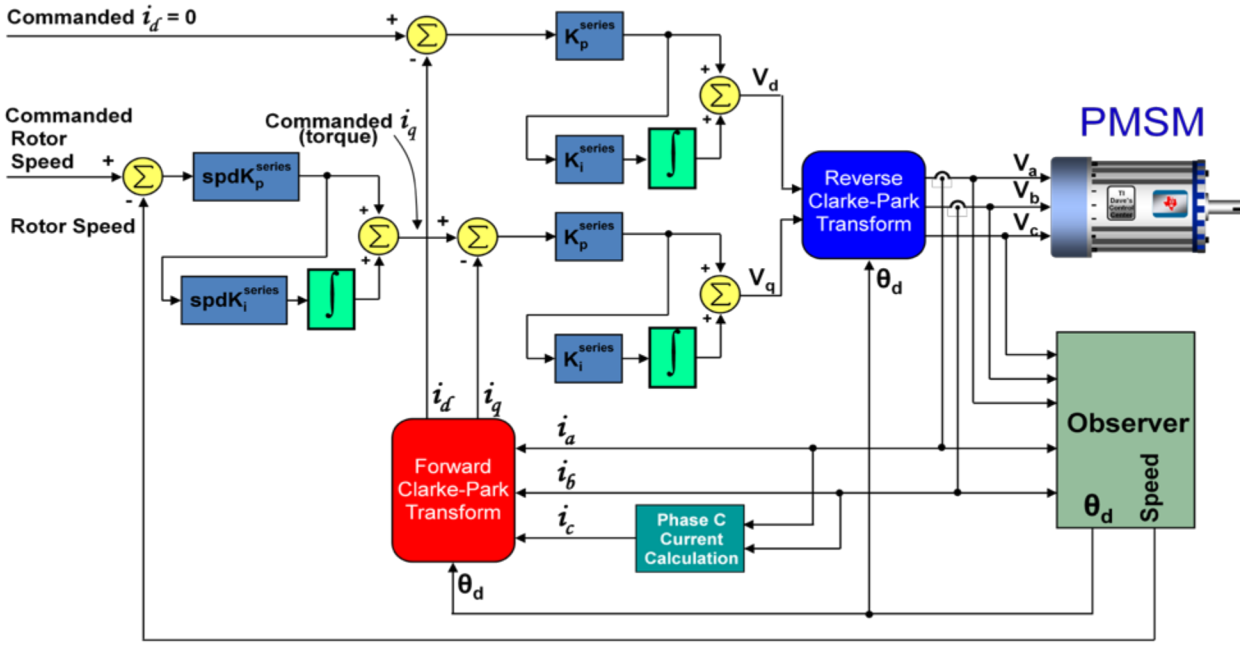


图 **11-18. PMSM** 的典型 **FOC** 速度控制

与其它控制算法相比，磁场定向系统中的速度控制器的设计变化不大。 只需确保在计算速度控制器的系数时使用 *q* 轴电流控制器值。 但是仍存在会对电流控制器设计方法产生影响的细微差异，这部分内容将在之后介绍。

11.8.1 电机类型间的 FOC 差异

从控制器（可决定 PI 系数）的角度看，电机的等效 RL 电路会有所不同，具体取决于电机类型。 对于 BLDC 和永磁同步电机 (PMSM)，R 仅表示定子电阻，L 为定子电感。 但如果使用交流感应电机 (ACIM)，则情况有所不同。 两个轴都需要使用的等效电感值不是定子电感值，而是“串联”电感（有时也称“漏”电感），定义如下:

$$L = L_s\left(1 - \frac{L_m^2}{L_s L_r}\right) = L_s \times \sigma \tag{71}$$

其中：

- L = 等效串联电感
- Ls = 定子电感
- Lm = 磁化电感
- Lr = 转子电感
- σ = 感应电机的“漏磁因数”

同样，对于 ACIM 的电流控制器，d 轴和 q 轴之间的电阻值也会有所不同。 对于 d 轴控制器，等效电阻即为定子电阻 Rs。 但对于 q 轴，等效电阻是定子电阻与转子电阻的总和 (Rs + Rr)。 如果计算 K_p^{series} 和 K_i^{series} 时没有考虑这些细节，则最后得到的 PI 控制器将无法与电机兼容，从而导致无法达到最佳控制水平。

11.8.2 Q 轴与 D 轴间的耦合

事实证明 d 轴电流控制与 q 轴电流控制之间并不是相互独立的。 在电机内部，q 轴电流会对 d 轴电流产生影响，反之亦然。 对于 PMSM，可通过以下微分方程说明这一点。

$$i_d\left(R + DL_s\right) = V_d + \omega L_s i_q \tag{72}$$

$$i_q\left(R + DL_s\right) = V_q - \omega\left(L_s i_d + K_e\right) \tag{73}$$

其中：

- R = 定子电阻
- Ls = 定子电感
- D = 微分运算符
- ω = 电频率
- Ke = 反电势常数

从公式 72 可以看出 d 轴电流不仅受 d 轴电流稳压器的输出电压 (Vd) 影响，而且受以 i_q 为函数的另一个电压的影响。 根据公式 73，Vq 同样要与电压“ω(Ls i_d + Ke)”竞争，二者共同控制 i_q 电流。 对于上述两种稳压器，这种交叉耦合效应表现为意外的干扰，在高速瞬态情况下最为显著。 要修正这一情况，应将前馈去耦合控制用于各个轴以准确消除上述竞争电压项。 此修正会使各个稳压器等效为简单 RL 电路，就好像使用的是 DC 电机。 结果即为稳压器拓扑，如图 11-19 所示。 为判断该技术是否有效，请考虑图 11-20 的模拟结果，其中显示了在采用以及不采用去耦合补偿的条件下 Q 轴电流的阶跃响应及其如何对 d 轴电流产生影响。

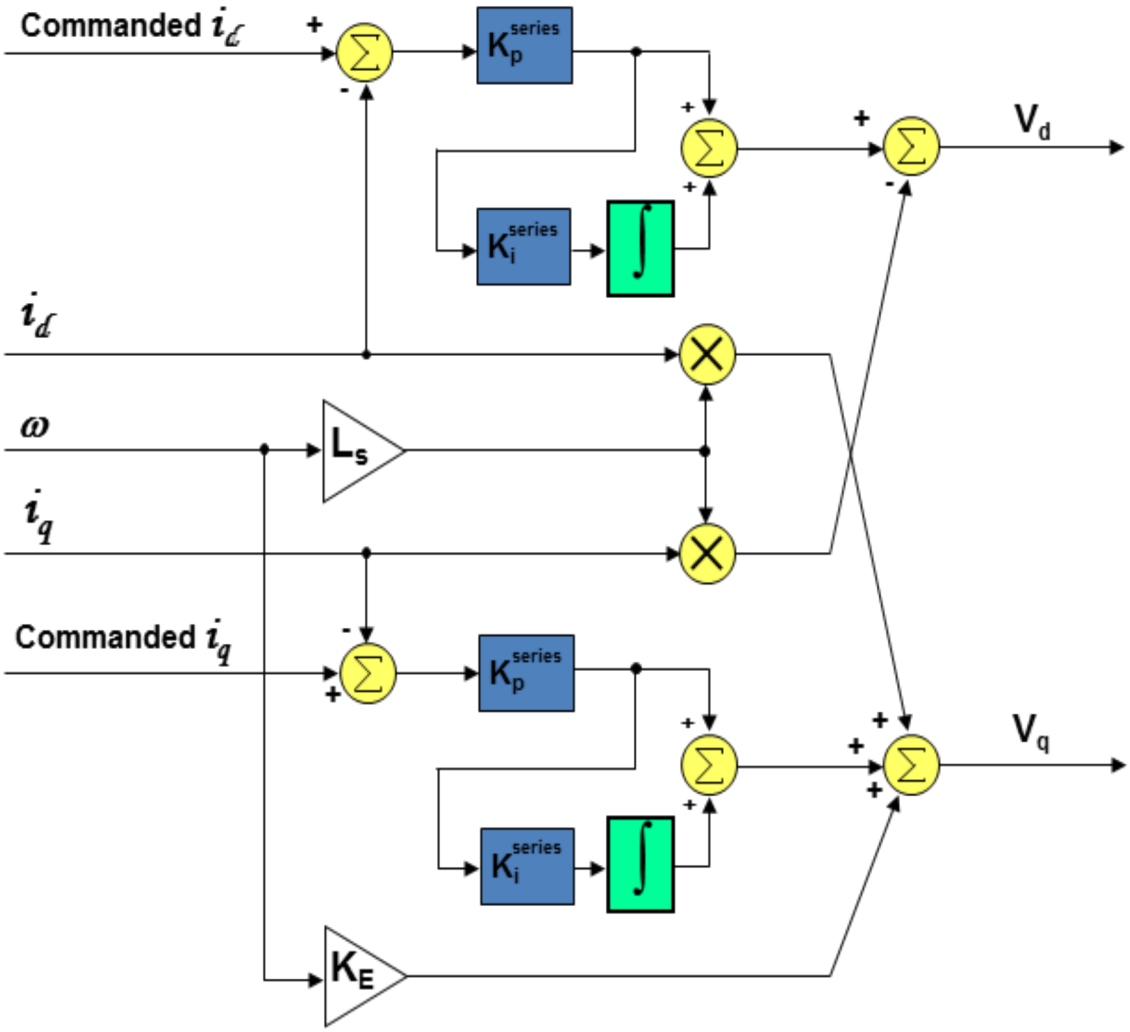


图 **11-19. PMSM** 的去耦合 **PI** 控制器

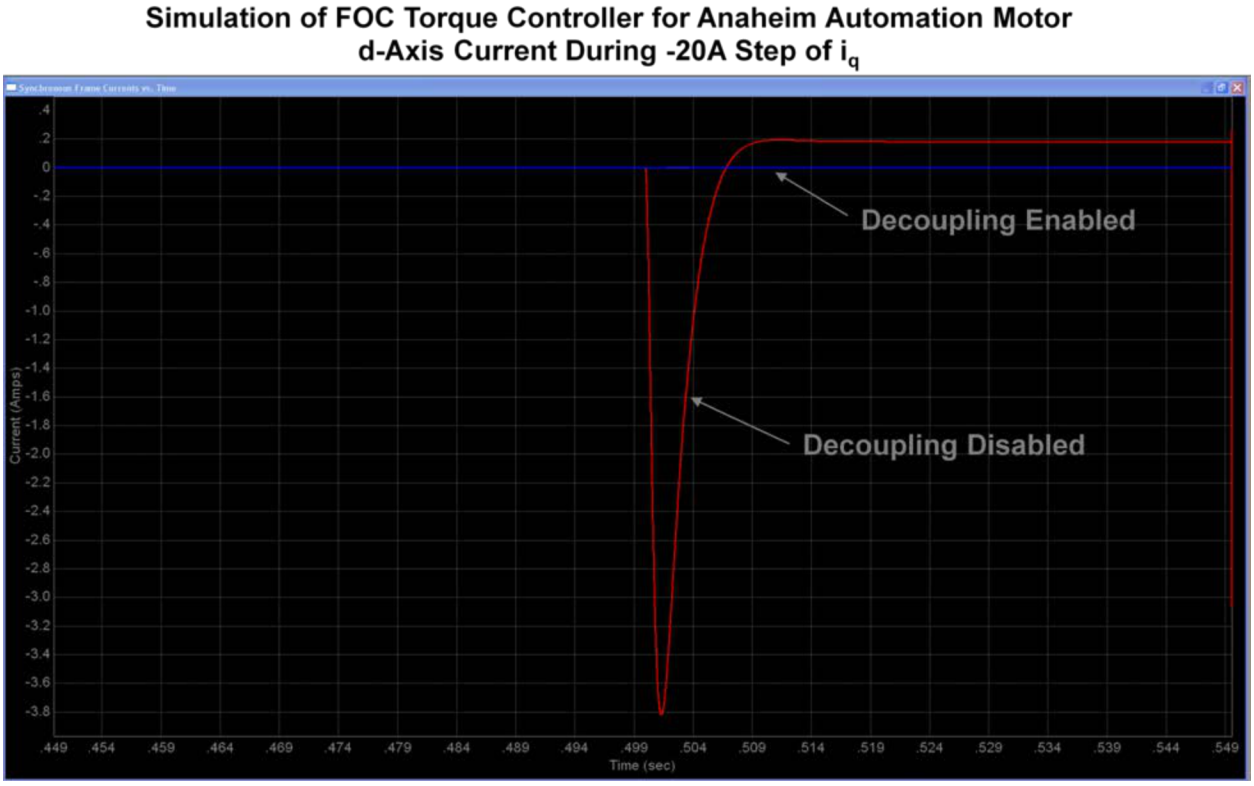


图 11-20. 模拟电流稳压器去耦合有效性

对于 AC 感应电机，修正会更加复杂一些。 用于定义 AC 感应电机运行的微分方程如下所示：

$$i_d\left(R_s + DL_s\sigma\right) = V_d + \omega L_s\sigma i_q - \frac{L_m}{L_r} D\lambda_{rd} \tag{74}$$

$$i_q\left(R_s + DL_s\sigma\right) = V_q - \omega L_s\sigma i_d - \omega\frac{L_m}{L_r}\lambda_{rd} \tag{75}$$

其中：

- Rs = 定子电阻
- Ls = 定子电感
- σ = 公式 71 中定义的漏磁因数
- D 为微分运算符
- ω = 电频率
- Lm = 磁化电感
- Lr = 转子电感
- λrd = d 轴转子磁通

与 PMSM 电机的情况类似，除 Vd 和 Vq 分别竞争对 i_d 和 i_q 的控制外还存在其它电压。 因此，可向 Vd 和 Vq 增加补偿电压以消除其它电压项的影响。 此会使各个轴等效为简单 RL 电路，就好像使用的是 DC 电机。 但如果使用的是 ACIM，请记住，用于计算 K_p^{series} 和 K_i^{series} 的电感值等于定子电感乘以漏磁因数 σ，如公式 71 所示。 用于向 I_d 和 I_q 稳压器输出提供修正电压的补偿模块如图 11-21 所示。

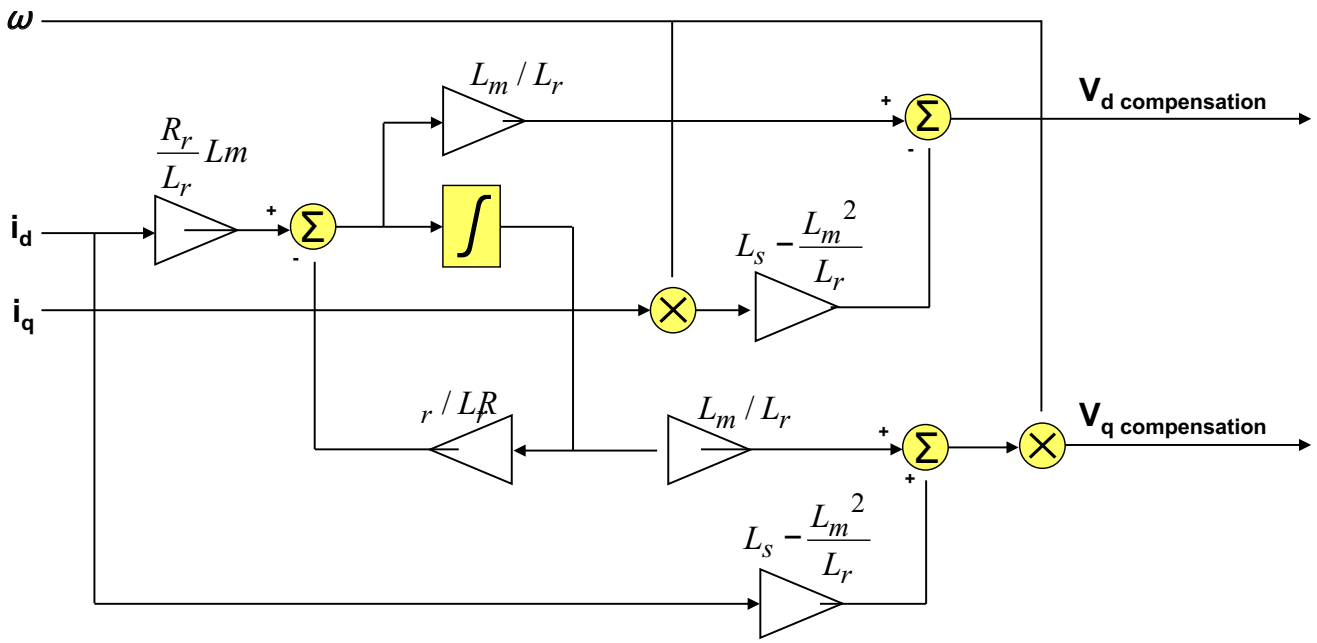


图 **11-21. ACIM** 轴去耦合所使用的补偿模块

11.9 采样和数字系统考量

在整个 PI 调整部分中，我们讨论了一种实用有效的级联速度环路 PI 控制器调整方法，该方法仅需指定速度环路所需的带宽和系统所需的阻尼因子。 利用上述两个参数以及一些基本的电机和负载参数，即可计算出速度环路和内部电流环路的 PI 系数。 但我们并没有讨论过对系统施加的限值，尤其是在使用数字系统时。 显然，为提高响应速度，需要更大的增益，即需要更高的带宽。 但上限是多少呢？

要回答这一问题，请参考图 11-22，其中显示了基于数字 FOC 的变频器 (VFD) 的高级视图。 为简化讨论，将假设整个控制环路都由普通采样信号提供时钟（尽管实际应用中未实施该限制条件）。 许多情况下，速度环路的时钟频率会比电流环路低很多，因为速度环路相关的频率通常要低得多。

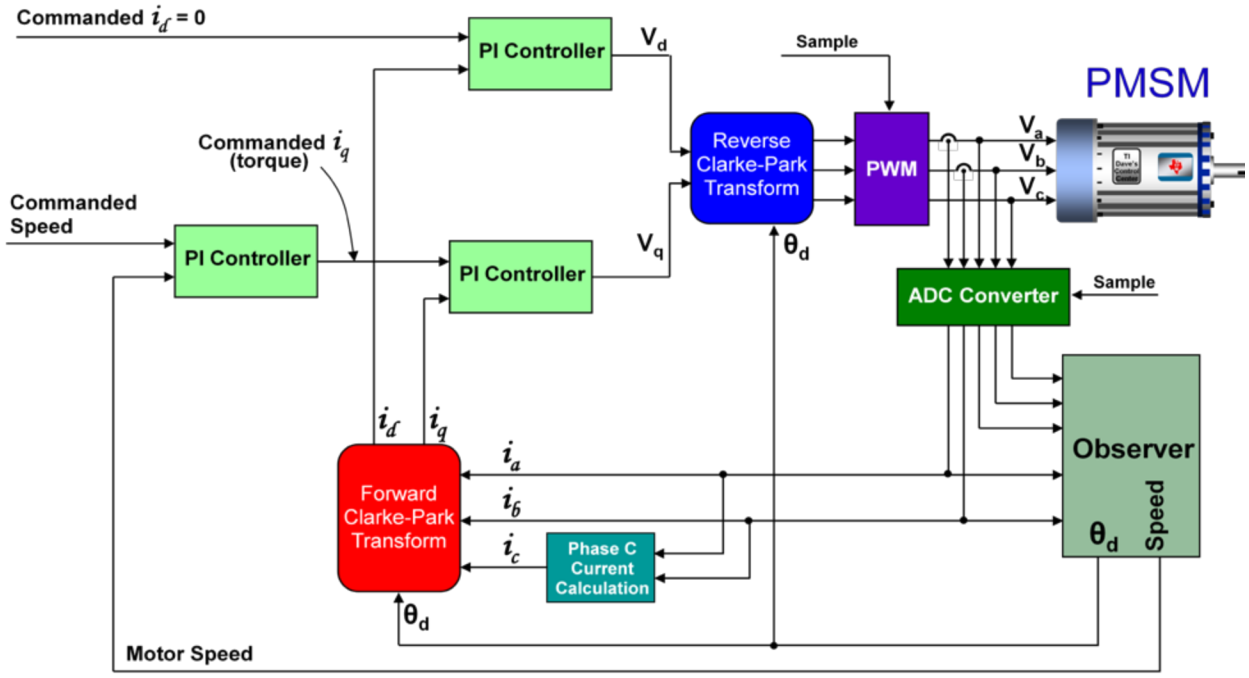


图 **11-22.** 适用于 **PMSM** 的数字磁场定向控制系统

在模拟系统中，电机反馈信号的任何变化都会立即对输出控制电压产生影响。 但在图 11-22 所示的数字控制系统中，系统在 PWM 周期开始时通过 ADC 对电机信号进行采样并执行相应的控制计算，然后将得到的控制电压存入双缓冲 PWM 寄存器。 上述值在 PWM 模块中保持未使用状态，直到下一个 PWM 周期开始时才会将其作为 PWM 输出的时钟信号。 从系统建模的角度看，这就相当于采样频率等于 PWM 更新频率时的采样和保持功能。 采样和保持功能的固定时间延迟表现为滞后相位角，频率越高时滞后情况越严重。 图 11-23 显示了采样和保持功能的标准化频率曲线图，假设采样频率为 1。相位曲线是最重要的图形，因为它显示了采样和保持功能的相位延迟频率能够达到远低于采样频率的水平。 例如，即使在采样频率的十分之一处，采样和保持 (S&H) 功能仍然会产生 -18 度的相移。

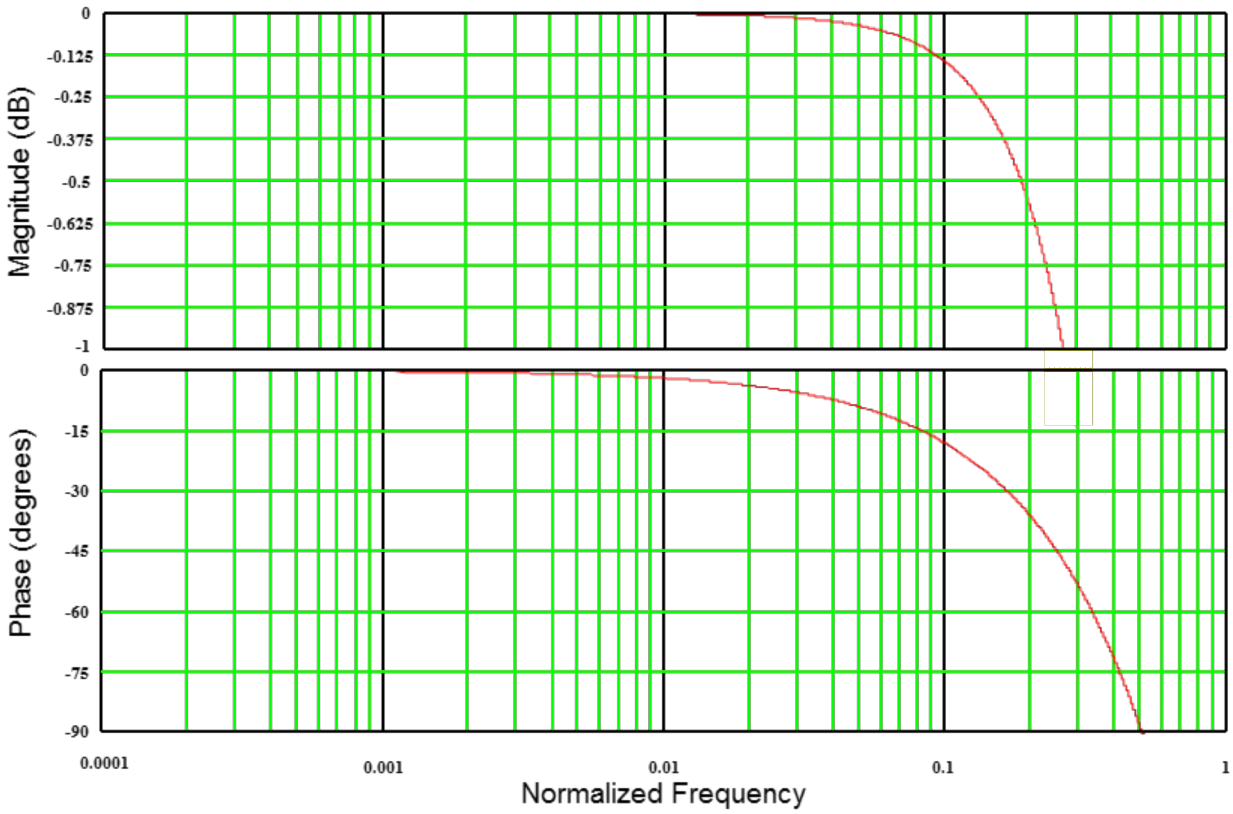


图 11-23. 采样和保持的幅度和相位曲线

与速度环路相比，电流控制器处理带宽更高的信号，因此电流环路通常承受 PWM 模块的大部分 S&H 影响。 由于 S&H 与电流环路的信号路径串联，因此其幅度和相位影响都将直接添加到电流控制器的开环响应中。 如果重写电流控制器的开环响应方程（假设进行了 PI 调整部分中建议的替换），则最终得到公式 76。

$$G_{loop}(s) = PI(s) \times \frac{I(s)}{V(s)} = \left(\frac{BW_c}{s}\right) \tag{76}$$

其中：

BW_c 为选定的电流控制器闭环带宽。

很明显，0dB 频率出现在 s = BW_c 时。 s = 0 处的单极点意味着 0dB 频率有 90 度的相位裕度。 尽管不存在完美比率，但通常倾向于使用经验法则，即采样频率应至少为电流控制器带宽 (BW_c) 的 10 倍。 这样可确保 S&H 相位延迟的影响仅从电流控制器的相位裕度中减去 18 度，从而得到非常稳定的 72 度相位裕度。 当然，如有需要还可设置更高的采样频率，但这通常需要使用具有更高 MIPS 的更加昂贵的处理器。

最后，看一下频率范围的上限。 在低频率下，粘滞阻尼可通过改变 0dB 频率下的相位裕度对速度环路响应时间产生影响。 根据11.3 节，已建立电机转矩和负载速度间的以下传输函数：

$$\frac{\omega(s)}{T(s)} = \frac{1}{J}\frac{1}{s} \tag{77}$$

但如果存在粘滞阻尼，则需要使用部分电机转矩克服阻尼以便流畅移动。 由于粘滞转矩直接与负载速度成正比，因此可将公式 77 改写为：

$$\frac{\omega(s)}{T(s)} = \frac{1}{Js + k_v} = \frac{1}{k_v\left(\frac{J}{k_v}s + 1\right)} \tag{78}$$

其中：

kv 是粘滞阻尼因子。

根据公式 78，添加粘滞阻尼项后使原本位于 s = 0 的极点移至 s = kv/J 处。图 11-24 显示了增加粘滞阻尼后负载波特图发生的变化。

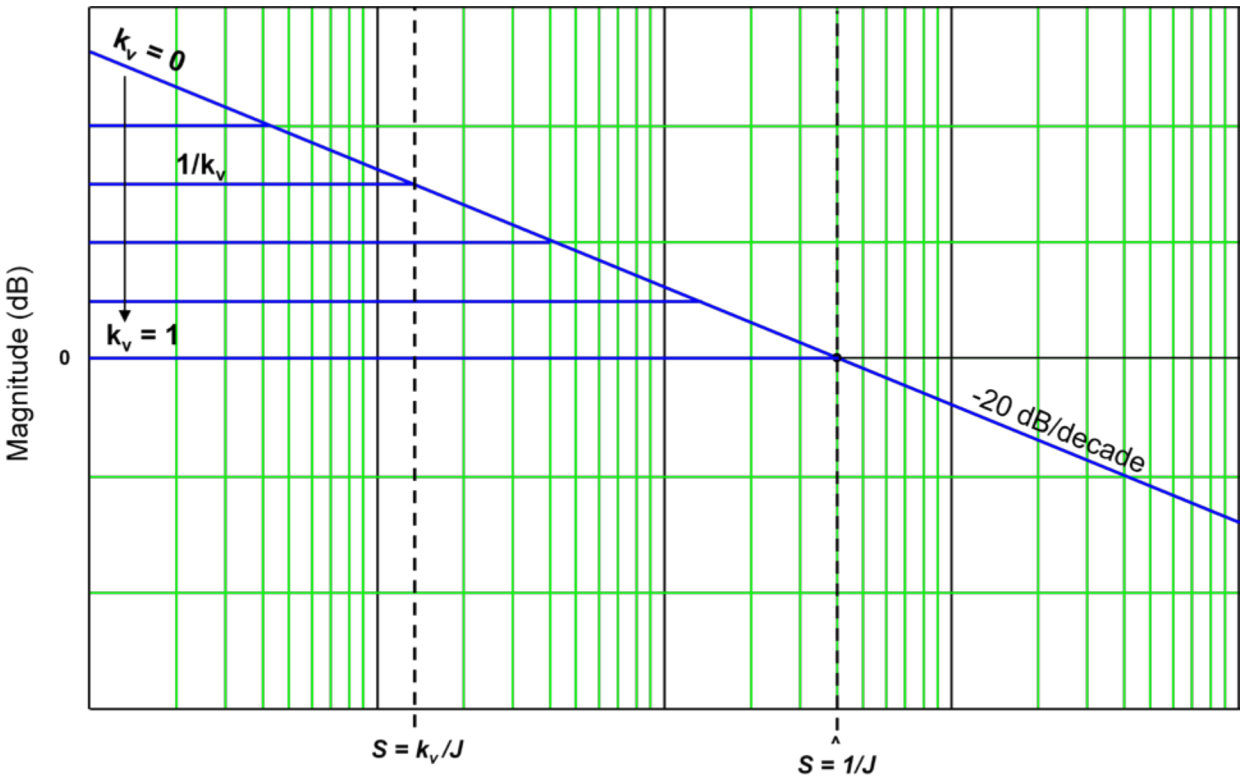


图 **11-24.** 粘滞阻尼 **(kv)** 对负载波特图的影响

如图 11-24 所示，随着粘滞阻尼从零开始增加，低频增益在值为 1/kv 处达到稳定。 在较低频率时，相位曲线的净效应将增加更多的相位裕度，因为具有粘滞阻尼的负载的相位滞后始终少于仅具有惯性的负载。 因此，当 kv 为非零值时稳定性得到实际提升。 但响应时间可能会受到影响，具体取决于图 11-24 中所示极点频率对应的速度开环 0dB 频率位置。 因此，如果系统响应迟缓且电机输出似乎未达到额定转矩，则可能是系统中粘滞阻尼过大。

在 PI 调整部分系列内容结束前，还有一个主题要讨论。 许多情况下，工程师正确计算了 PI 系数。 在代码中使用这些系数后，出现电机旋转失控或静止不动的情况。 为什么会发生这种情况？ 很有可能是以下情况之一出现故障：

11.9.1 积分增益中的采样周期考量

没有考虑采样频率对 I 增益项的影响。图 11-25 显示了如何实现 PI 控制器的典型积分器。 为按比例输出以便匹配模拟积分器所提供的值，必须将信号乘以采样周期“T”。 为了避免进行两次单独的乘法运算，大部分代码示例直接将 T 和 I 增益项放在一起。 如果不考虑 T，则积分器增益将远大于预期值。

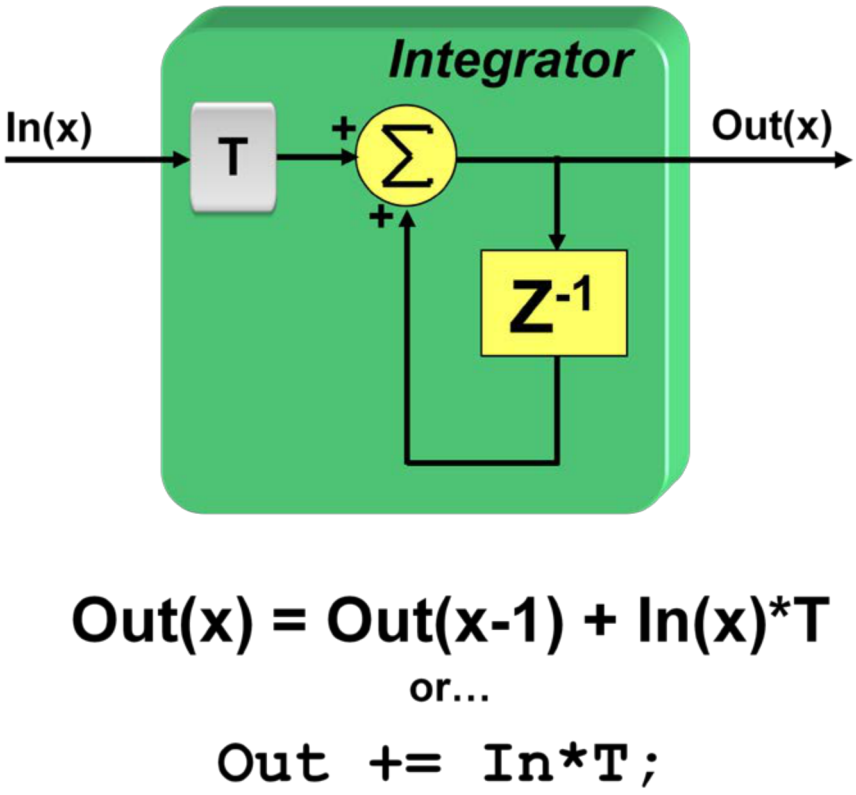


图 **11-25.** 典型数字积分器实现

11.9.2 数字格式考量

到目前为止，我们假设数字格式本身没有任何限制。 如果使用的是浮点处理器，则无需担心 PI 项的小数部分。 但出于成本考虑，大多数电机控制应用都将在定点计算机上实现。 好消息是 TI 已开发出链接库可以解决这一问题，该库位于大多数 C2000 处理器的只读存储器 (ROM) 中。 该链接库名为“IQ Math”库，代表“整数商”。 借助该库，用户可以轻松地在定点计算机上处理浮点值，而不会像使用全浮点支持器件那样受到性能影响。 IQ math 可在代码中创建新的变量类型，通过“IQ”后跟数字来指定。 例如，如果使用 32 位变量，其变量类型为“IQ24”。 这表示所有此类型的变量小数部分均为 24 位，整数部分均为 8 位。 但偶尔会发生这样的情况：有人将 TI 代码复制到其设计中，但没有意识到系数是以 IQ 格式表示的。 例如，如果 I 增益的计算结果为 10,000（IQ0 格式为 0x00002710），但未意识到代码已假设变量为 IQ24 格式，则最终得到的积分器增益为 0.596E-3 而不是 10,000。 很明显这两个值不同。 如果所有 PI 系数都发生了同样的错误，则电机很可能静止不动，因为所有增益均过低。 因此，建议确保系数使用的数字格式已知。

11.9.3 PI 系数换算考量

最后，PI 调整部分系列内容中的 PI 系数换算已完成（假设要表示整个信号链的实际系统值）。 例如，速度 PI 控制器的输出等于电流控制器的实际输入基准电流（单位为安培）。 电流控制器输出等于对电机绕阻施加的实际电压。 但在许多设计中，PI 控制器输出将标准化为标么值换算，其中值 1 代表可能的最大值，值 -1 代表可能的最小值。 例如，电流稳压器输出可按以下方式换算：1 对应 100% PWM，-1 对应 0% PWM。 上述情况下，需要了解 PI 输出与所控制的实际系统参数之间的确切换算系数，以便对 PI 系数做出相应调整。

InstaSPIN-MOTION 控制器

对于大多数运动系统，均需要对系统的速度和/或位置进行调节。 如Chapter 11所述，行业标准速度控制器是 PI 控制器。 PI 控制器存在很多固有的缺陷。

- 为了调整特定速度和负载运行点的控制，需要调整多个参数。 这些参数将生成多维解决方案集，而增益通常需要通过实验确定，这会使调整十分困难。
- 适用于该特定调整的速度和负载的范围非常小。 如果是高度动态系统且有许多不同的速度和负载运行点，则需要针对每个运行点调整 PI 控制器。

SpinTAC 控制器可以解决这些问题。 SpinTAC 可提供高级速度和位置控制，并具有主动抗扰控制 (ADRC)，可通过单个调整参数确定所有增益。 ADRC 适用于具有高度不确定性的模型，这意味着它在系统变动时能够稳定运行。

干扰定义为系统中的任何不良行为。 非模型化的动态过程和干扰所导致的误差将由 SpinTAC 控制器进行估算和补偿。 该控制器的独特之处在于将系统中的所有不良行为视为可估算和抑制的干扰进行处理。 这使得 SpinTAC 控制器可通过单个调整参数对多个位置、速度和负载进行控制。

该单个调整参数称为带宽，用于确定系统的抗扰性以及指示系统抑制干扰的积极程度。 通过使用此单个参数，可以轻松调整 SpinTAC 速度控制器。

调整动态系统控制器时要考虑的两个主要因素是稳定性和性能。

Topic **Page**

12.1 稳定性

控制系统的稳定性是工程系统中的安全问题。 系统稳定性有多种定义：李雅普诺夫稳定性、有界输入有界输出稳定性和输入-状态稳定性。 为简单起见，本文档中使用李雅普诺夫渐近稳定性作为稳定性标准，这意味着系统本身能够逐渐收敛于平衡点。

在速度控制中，平衡点是指阶跃响应的目标速度，或跟踪变化的参考速度时的速度轨迹。 在位置控制中，平衡点是指阶跃响应的结束位置或转换期间变化的位置参考。

评估系统稳定性的一种简单方法是看阶跃响应最终是否收敛于设定点。 稳定系统和不稳定系统的典型阶跃响应如图 12-1 所示。

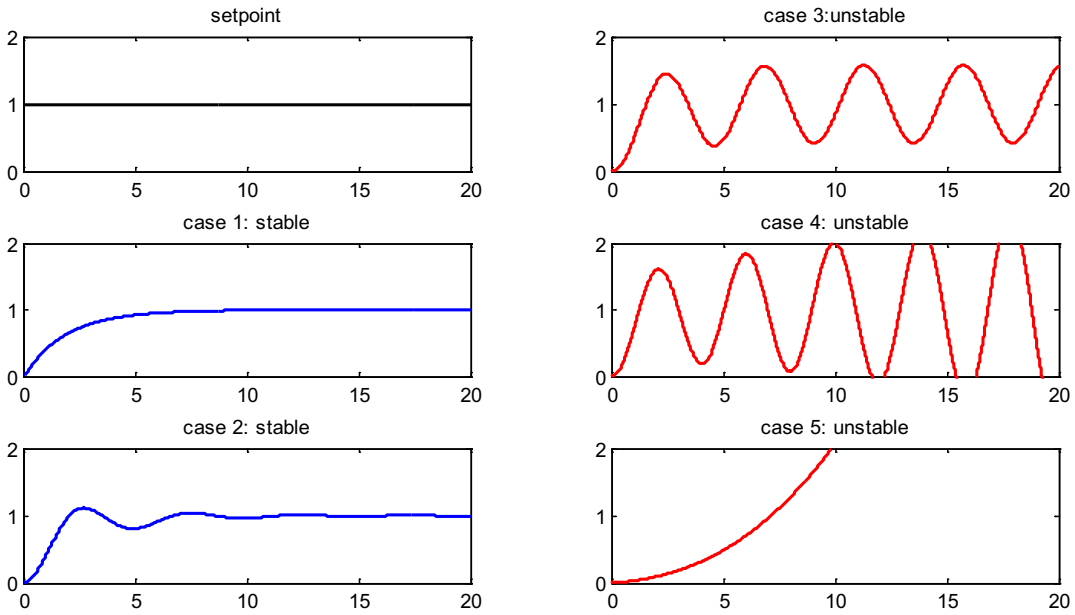


图 12-1. 稳定系统和不稳定系统的典型阶跃响应

上图显示了应用单位阶跃输入时的系统响应。 左上角是设定点输入信号。 情况 1 和 2 为稳定系统；情况 3、4 和 5 为不稳定系统。 在某些实例中会将情况 3 定义为临界稳定，因为其响应为有界振荡。

12.1.1 稳定性量化分析

经典控制设计可以对系统进行建模、生成运行点的近似线性表达式，以及使用波特图分析通过增益裕度和相位裕度来评估稳定性。 增益裕度是指相位曲线到达 -180° 时频率的幅度曲线值，为负数。 相位裕度是指幅度曲线的增益为 0dB 时频率的相位曲线值与 -180° 的差值。

为使系统能够耐受一定程度的非线性和模型不匹配，通常需要使用 6 至 12dB 的增益裕量和 30 至 45度的相位裕量。

12.1.1.1 SpinTAC 速度控制稳定性

假如电机的速度环路动态特性不考虑谐振模式、采样时间和输出饱和等不确定因素，SpinTAC 所控制的速度环路的开环波特图将始终保持稳定，如图 12-2 所示的开环波特图。 相位曲线未达到 -180°，这意味着对增益裕度没有限制。 相位裕度始终为正。

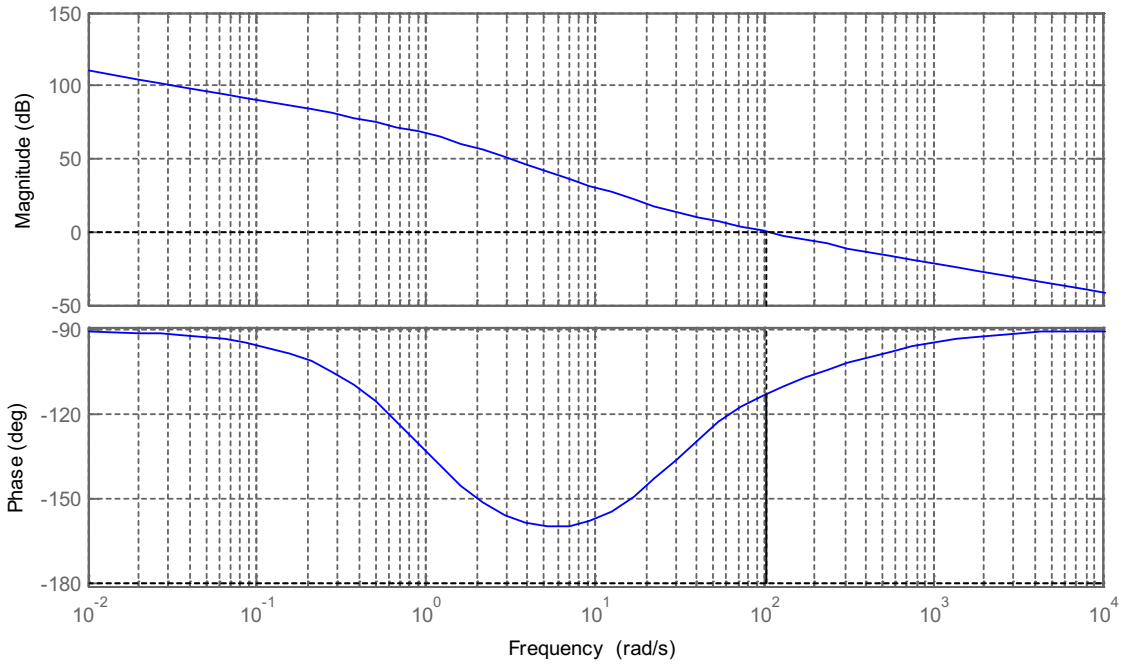


图 **12-2.** 典型的 **SpinTAC** 速度控制开环波特图

但是，大部分机械系统具有谐振模式（通常在高频时），这将导致相位曲线出现 180° 的变化，而幅度曲线中会出现尖峰。

为保持系统稳定，输出饱和与采样时间也会限制控制器增益的可调范围。

12.1.1.2 SpinTAC 位置控制稳定性

SpinTAC 位置控制可以同时控制位置环路和速度环路。

位置控制比速度控制更为复杂。 相位曲线在 -180° 处有一个交叉，可在该处测量增益裕度。 SpinTAC 位置控制可提供良好的负增益裕度以耐受系统的变化。 通常，此负增益裕度代表惯性测量不正确时已配置的惯性值可以比实际系统惯性值大多少（请参见图 12-3）。

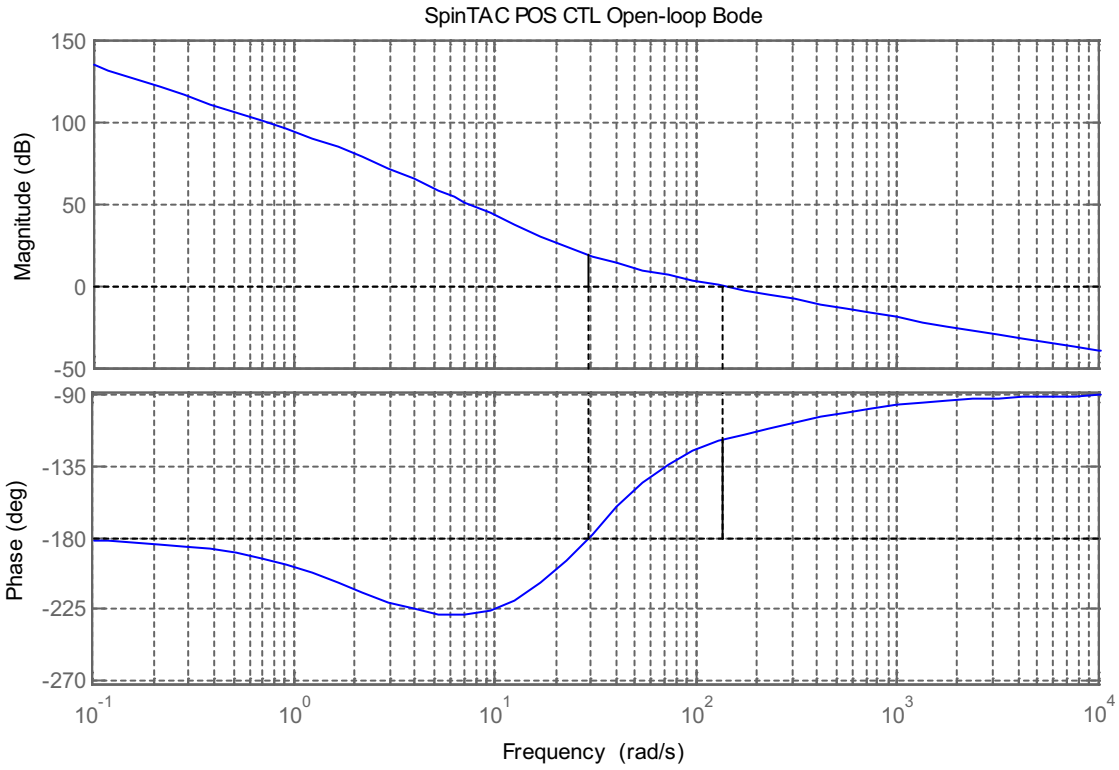


图 **12-3.** 典型的 **SpinTAC** 位置控制开环波特图

与速度控制类似，位置控制也会受高频谐振模式和噪声的影响，为保持系统稳定，控制器增益的可调范围也将受到限制。 这是所有类型控制器的常规控制设计注意事项。

12.1.2 性能

通常可以按两种标准来评估控制器的性能： 参考跟踪性能和抗扰性能。

参考跟踪性能用于显示系统跟随所需轨迹的接近程度。 当设定点发生变化时，此性能可显示系统在合理过冲下达到新设定点的速度。

抗扰性能用于显示向系统施加干扰后系统出现的偏离程度以及对偏离进行补偿的速度。

可在时域和频域对控制器的性能进行评估。

12.1.2.1 频域分析

如果能够获得系统的近似线性模型，则可在频域评估系统性能。 本部分的目的是以工程语言（波特图分析）直观描述 SpinTAC 速度控制和 SpinTAC 位置控制性能，而非要求用户对给定系统进行分析。

SpinTAC 速度控制设计为可同时优化抗扰性能和跟踪性能轨迹并通过单个参数（即带宽）来调整控制。 典型参考跟踪性能波特图和抗扰性能波特图如图 12-4 所示。

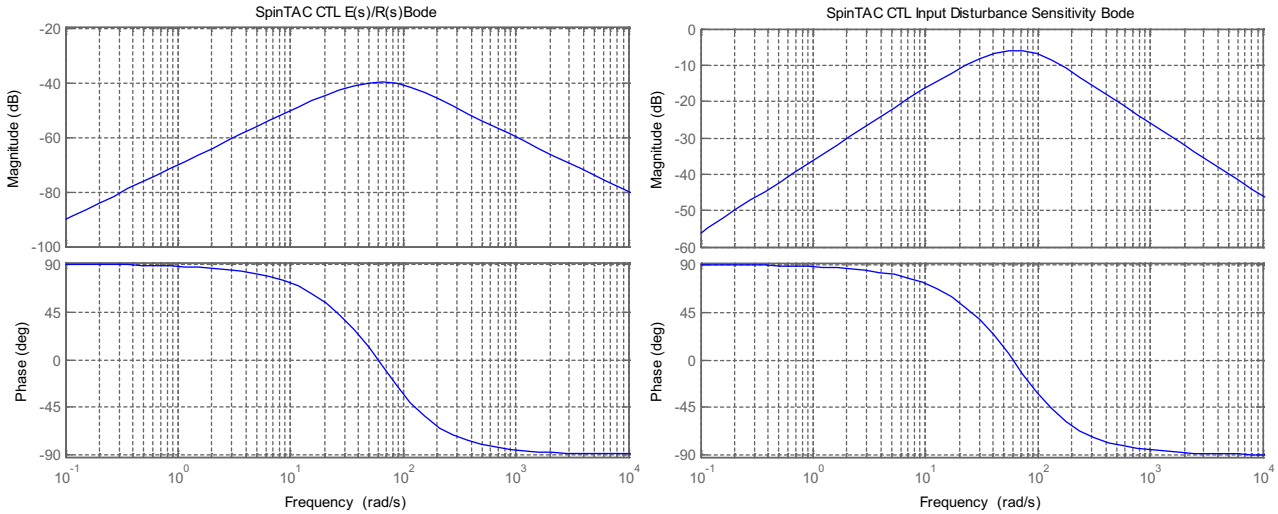


图 12-4. 典型的 SpinTAC 速度控制性能波特图

如图 12-4 所示，误差/参考波特图和输入干扰灵敏度波特图的幅度为负值，单位为 dB。 幅度曲线上的负值越多，系统性能越好。

SpinTAC 位置控制的性能分析与 SpinTAC 速度控制类似。 典型性能波特图如图 12-5 所示。

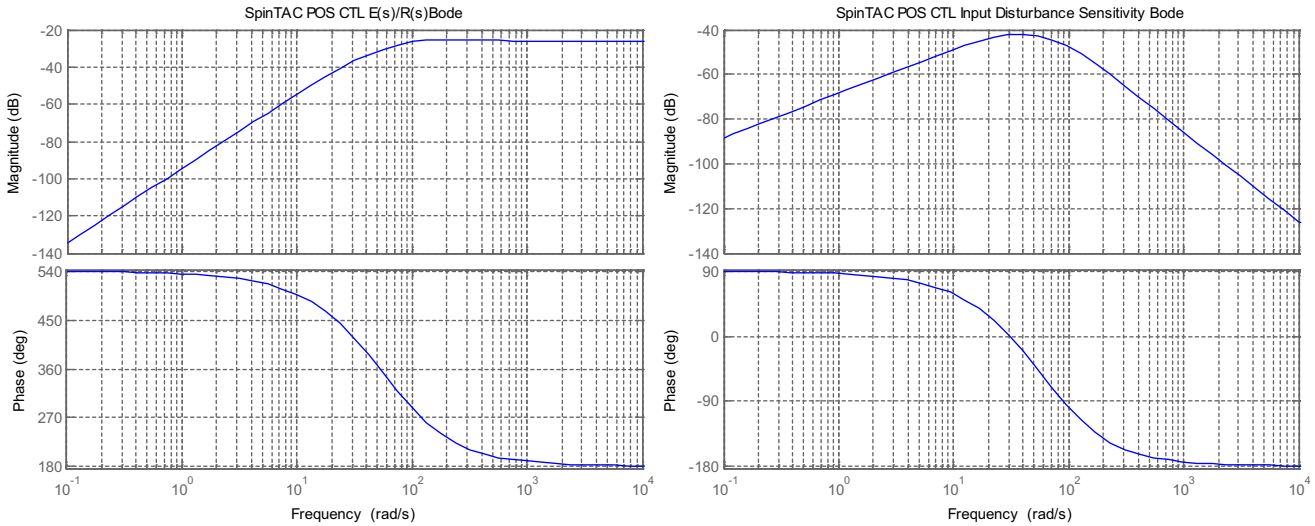


图 12-5. 典型的 SpinTAC 位置控制性能波特图

12.1.2.2 时域分析

可在时域中轻松评估性能。 表 12-1 中列出了时域常见标准。

表 12-1. 时域常见标准

性能标准	说明
过冲	首次经过设定点后偏离设定点的最大值
稳定时间	从启动到最终进入阶跃设定点附近定义的百分比（通常为 2% ~ 5%）范围内所需要的时间
最大绝对误差 (MAE)	偏离设定点的最大绝对值，用于指示最坏情况下的值
积分绝对误差 (IAE)	偏离设定点的积分绝对值，用于指示随着时间推移产生的偏离

12.1.3 稳定性和性能之间的权衡

通常，系统稳定性和性能之间存在一种权衡关系。 实际系统中始终具有噪声和不确定因素（高阶未知动态特性，如谐振模式）， 因此积极调整的控制器会表现出更好的性能， 但是当控制器调整到某种程度后，系统已接近不稳定条件或允许进入系统的噪声过高，此时会降低系统性能。

12.1.4 调整 SpinTAC 控制器

通过使用单个系数调整，SpinTAC 使您可以快速地对速度和位置控制进行测试和调整，提高响应速度。 这种单一增益（带宽）适用于一个应用的整个速度和负载变化范围，降低了系统复杂度，减少了系统调整时间。 基于多变量 PID 的系统经常需要十多个或更多速度和负载的已调整系数集，才能处理所有可能的动态情况。

12.1.4.1 注意事项

在调整闭环控制器时需要考虑噪声、谐振模式和采样时间。 SpinTAC 通过一个调整参数简化了调整过程。 通过调整带宽，可在保持稳定性裕度的前提下轻松达到所需的性能。 当施加的干扰负载大小相同时，随着带宽的增加，距离设定点的偏离将减小。 但随着带宽的增加，控制器用于确定输出的噪声也会增加。 确定正确的调整通常就是在抗扰与生成的合成噪声之间达到平衡。

SpinTAC 控制器将自动考虑采样时间。 它使用系统的采样时间作为对可用带宽的一种限制。 此限制可避免系统具有很高的带宽和很小的采样时间（可导致系统不稳定），从而避免系统进入不稳定状态。

12.1.4.2 调整 InstaSPIN-MOTION 控制器

通过单个参数调整 SpinTAC 控制器。 此参数称为带宽。 SpinTAC 控制器的带宽通过配置参数带宽范围进行调整。 应使用此值调整控制器的带宽值以满足应用的控制要求。 对于位置控制应用，可使用单个带宽设置位置增益和速度增益。

图 12-6 显示了随着带宽增加 SpinTAC 速度控制对转矩干扰的响应。 随着带宽增加，对转矩干扰的响应变得更快，过冲变得更小。 当带宽增加过大后，如果从系统中移除转矩干扰，响应将在目标速度附近振荡。 这表明带宽设置过高。 在本示例中，理想带宽为 40rad/s。 这是因为在这种情况下移除转矩后，响应在目标速度附近振荡最小。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

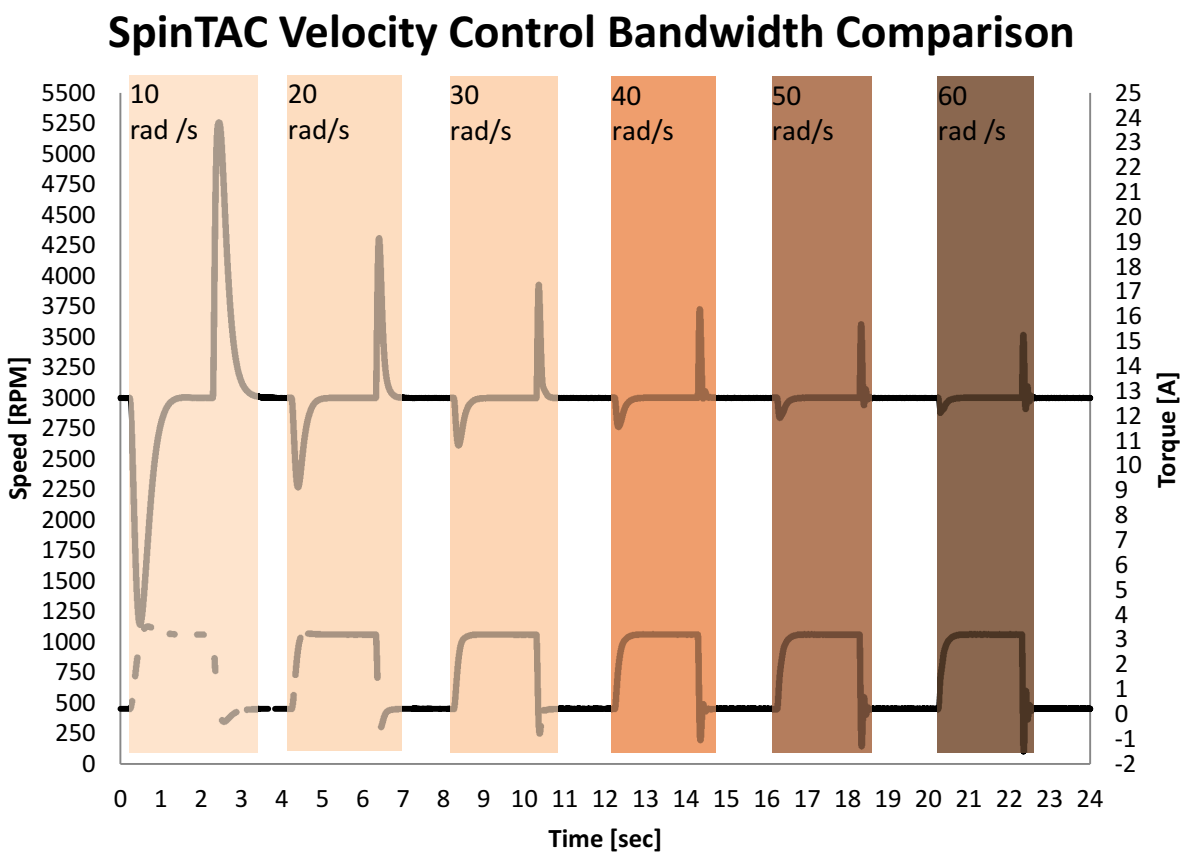


图 **12-6. SpinTAC** 速度控制的带宽比较

图 12-7 显示了 SpinTAC 位置控制对转矩干扰的响应。 随着带宽增加，对转矩干扰的响应变得更快，过冲变得更小。 当带宽增加过大后，电机将开始发出噪声并进行振荡。 这表明 SpinTAC 位置控制的输出已开始振荡。 在本示例中，理想带宽为 50rad/s。 因为此时对干扰有良好的响应且输出振荡最小。

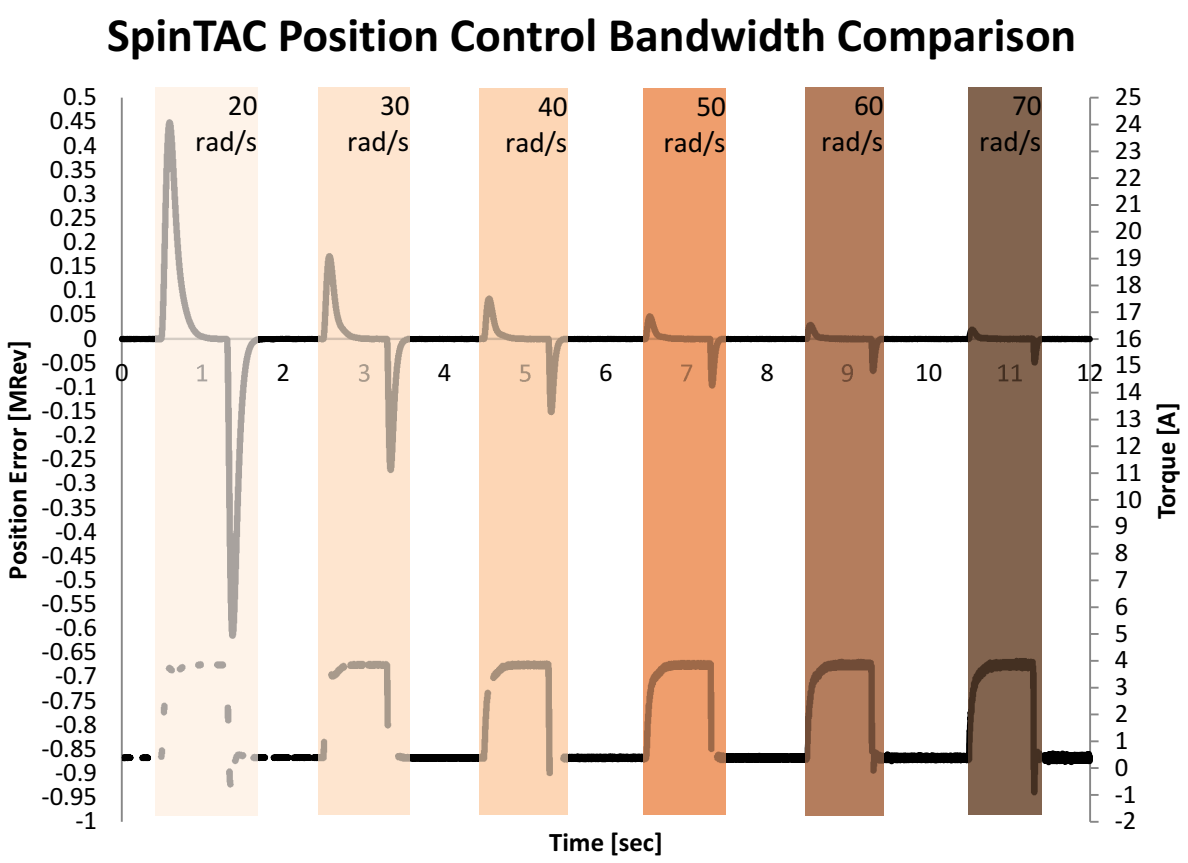


图 **12-7. SpinTAC** 位置控制的带宽比较

要调整 SpinTAC 控制器，第一步是将速度参考设置为零。 电机速度达到零后，用手旋转电机轴感受电机保持零速度的牢固程度，这可用于指示调整电机的积极程度。 按步长增量 1 增加带宽范围“gMotorVars.SpinTAC.VelCtlBwScale”或“gMotorVars.SpinTAC.PosCtlBwScale”，继续感受电机保持零速度的牢固程度。 对于无法接触到转轴的电机，为电机设定一个参考步长。 更改参考并监控控制器试图达到新设定点的积极程度。 一旦 SpinTAC 控制器能够牢固保持零速度，针对保持零速度的带宽范围调整操作就已完毕。 此时需要运行电机以确保此带宽适用于应用的整个工作范围

12.2 SpinTAC 速度控制的软件配置

配置 SpinTAC 速度控制需要四个步骤。 实验 5d“InstaSPIN-MOTION 速度控制器”是一个示例项目，在其中执行了使用 SpinTAC 速度控制所需的步骤。 利用包含在 MotorWare 中的头文件 spintac_velocity.h，您可以快速将 SpinTAC 组件包括在项目中。

12.2.1 包括头文件

这应该通过其余的模块头文件包含来完成。 在实验 5d 示例项目中，该文件包括在 spintac_velocity.h 头文件中。 针对相应项目，可通过包括 spintac_velocity.h 完成此步骤

```
#include "sw/modules/spintac/src/32b/spintac_vel_ctl.h"
```

12.2.2 声明全局结构

这应该通过主源文件中的全局变量声明来完成。 在实验 5d 项目中，此结构包括在已声明为 spintac_velocity.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj st_obj;      // The SpinTAC Object
ST_Handle stHandle; // The SpinTAC Handle
```

此示例为不希望使用已在 spintac_velocity.h 头文件中声明的 ST_Obj 结构的情况。

```
ST_VelCtl_t      stVelCtl;       // The SpinTAC Speed Controller object
ST_VELCTL_Handle stVelCtlHandle; // The SpinTAC Speed Controller Handle
```

12.2.3 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这样会将所有的默认值加载到 SpinTAC 速度控制中。 该步骤可通过运行已在 spintac_velocity.h 头文件中声明的函数 ST_init 和 ST_setupVelCtl 来完成。 如果不希望使用这两个函数，可使用下方的代码示例配置 SpinTAC 速度控制组件。 SpinTAC 速度控制配置是典型配置的代表，适用于大部分电机。

```
// Initialize the SpinTAC Speed Controller Component
stVelCtlHandle = STVELCTL_init(&stVelCtl, sizeof(stVelCtl));

// Setup the maximum current in PU
_iq maxCurrent_PU = _IQ(USER_MOTOR_MAX_CURRENT / USER_IQ_FULL_SCALE_CURRENT_A);

// Instance of the velocity controller
STVELCTL_setAxis(stVelCtlHandle, ST_AXIS0);
// Sample time [s], (0, 1)
STVELCTL_setSampleTime_sec(stVelCtlHandle, _IQ(ST_SPEED_SAMPLE_TIME));
// System inertia upper (0, 127.9999] and lower (0, SgiMax] limits [PU/(pu/s^2)]
STVELCTL_setInertiaMaximums(stVelCtlHandle, _IQ(10.0), _IQ(0.001));
// System control signal high (0, OutMax] & low [OutMin, 0) limits [PU]
STVELCTL_setOutputMaximums(stVelCtlHandle, maxCurrent_PU, -maxCurrent_PU);
// System maximum (0, 1.0] and minimum [-1.0, 0) velocity [pu/s]
STVELCTL_setVelocityMaximums(stVelCtlHandle, _IQ(1.0), _IQ(-1.0));
// System upper (0, 0.2/(T*20)] and lower [0, BwScaleMax] limits for bandwidth scale
STVELCTL_setBandwidthScaleMaximums(stVelCtlHandle,
                                   _IQ24((0.2) / (ST_SPEED_SAMPLE_TIME * 20.0)), _IQ24(0.01));
// System inertia [PU/(pu/s^2)], [SgiMin, SgiMax]
STVELCTL_setInertia(stVelCtlHandle, _IQ(USER_SYSTEM_INERTIA));
// Controller bandwidth scale [BwMin, BwMax]
STVELCTL_setBandwidthScale(stVelCtlHandle, _IQ24(USER_SYSTEM_BANDWIDTH_SCALE));
// Initially ST_VelCtl is not enabled
STVELCTL_setEnable(stVelCtlHandle, false);
// Initially ST_VelCtl is not in reset
STVELCTL_setReset(stVelCtlHandle, false);
```

12.2.4 调用 SpinTAC 速度控制

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此函数。 抽取率由 ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。 在调用 SpinTAC 速度控制函数之前，必须更新速度参考、加速度参考和速度反馈。 本示例使用 SpinTAC 速度移动为 SpinTAC 速度控制提供参考。 有关 SpinTAC 速度移动的详细信息，请参见Chapter 13。

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;    // Get pointer to CTRL object
// Get the mechanical speed in pu/s
_iq speedFeedback = EST_getFm_pu(obj->estHandle);    // Get the mechanical speed in pu/s
// Update the Velocity Reference
STVELCTL_setVelocityReference(stVelCtlHandle,
                                  STVELMOVE_getVelocityReference(stVelMoveHandle));
//Update the Acceleration Reference
STVELCTL_setAccelerationReference(stVelCtlHandle,
                                 STVELMOVE_getAccelerationReference(stVelMoveHandle));
//Update the Velocity Feedback
STVELCTL_setVelocityFeedback(stVelCtlHandle, speedFeedback);
// Run the SpinTAC Speed Controller
STVELCTL_run(stVelCtlHandle);

// Get the Torque Reference from the SpinTAC Speed Controller
iqReference = STVELCTL_getTorqueReference(stVelCtlHandle);

// Set the Iq reference that came out of SpinTAC Velocity Control
CTRL_setIq_ref_pu(ctrlHandle, iqReference);
```

12.2.5 SpinTAC 速度控制故障排除

12.2.5.1 ERR_ID

ERR_ID 可向用户提供错误代码。 表 12-2 中列出了为 SpinTAC 速度控制定义的错误及相应解决方法。

表 12-2. SpinTAC 速度控制 ERR_ID 代码

ERR_ID	问题	解决方法
1	采样时间值无效	将 cfg.T_sec 设置为 (0, 1] 范围内
2	最大参考值无效	将 cfg.VelMax 设置为 (0, 1] 范围内
3	最小参考值无效	将 cfg.VelMin 设置为 (0, 1] 范围内
4	最大控制信号值无效	将 cfg.OutMax 设置为 (0, 1] 范围内
5	最小控制信号值无效	将 cfg.OutMin 设置为 [-1, 0) 范围内
16	最大惯性值无效	将 cfg.InertiaMax 设置为正 _iq24 值
17	最小惯性值无效	将 cfg.InertiaMin 设置为 (0, cfg.InertiaMax] 范围内
18	最大带宽值无效	将 cfg.BwMax 设置为 [0, min(2000, 0.2/cfg.T)] 范围内
19	最小带宽值无效	将 cfg.BwMin 设置为 [0, cfg.BwMax] 范围内
32	轴 ID 无效	将 cfg.Axis 设置为 {ST_AXIS0, ST_AXIS1} 范围内
1012	惯性值无效	没有动作。 惯性饱和至界限 [cfg.InertiaMin, cfg.InertiaMax]
1014	带宽 × 惯性大于 2000	没有动作。 实际带宽饱和至值 2000/惯性
1016	摩擦超出限值	没有动作。 摩擦饱和至调整摩擦范围 [0, 5]
4001	SpinTAC 许可证无效	使用具有有效许可证的芯片
4003	ROM 版本无效	使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。

12.3 速度控制中的最优性能

12.3.1 简介

从您的机械运动系统中获得最佳性能是十分重要的。 未经良好调节的稳压器会导致能源浪费、材料浪费，或者使系统不稳定。 为了确定控制器在您的应用中工作状态如何，有必要在很多不同的速度和负载运行点上评估速度控制器的性能。

12.3.2 比较速度控制器

可根据很多不同的因素对速度控制器进行比较。 然而，两个衡量标准——抗扰和系统配置跟踪——可被用来测试性能，并且确定您的控制器对于您的应用被调整的如何。

12.3.3 抗扰

抗扰用于测试控制器对影响电机速度的外部干扰的抵抗能力。 使用最大速度误差和稳定时间来测量抗扰。 最大速度误差显示从目标速度的偏离，并且表示出对您的控制器进行调整的积极程度。 积极调整将产生一个低最大误差。

稳定时间是指从干扰发生的时间点到速度返回至一个目标速度附近的固定范围内所需要的时间。 这也表示出您的控制环路调整的积极程度。 如果控制器的调整过于积极，控制器的稳定时间将会比较长，这是因为控制器将在稳定前在目标速度附近振荡。

图 12-8 和图 12-9 显示了同一控制器不良调整与最优调整的差异。 可以看出，通过对速度控制器进行调整，在向电机系统施加转矩或从电机系统中移除转矩时，最优调整的控制器可以显著减小最大误差和稳定时间。 本示例有所夸大，但它是为了强调对系统进行良好调整的重要性。

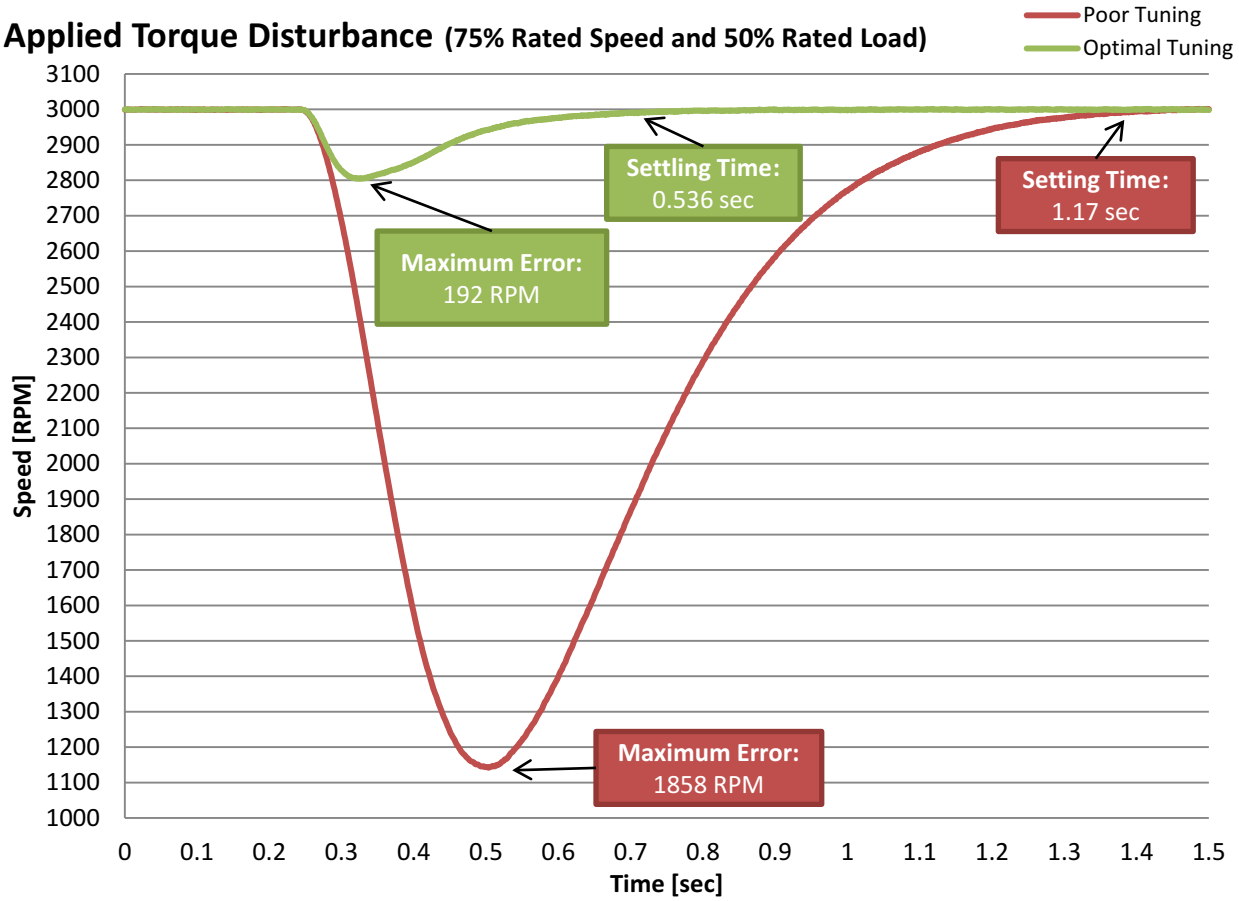


图 **12-8.** 施加转矩干扰的速度调整比较

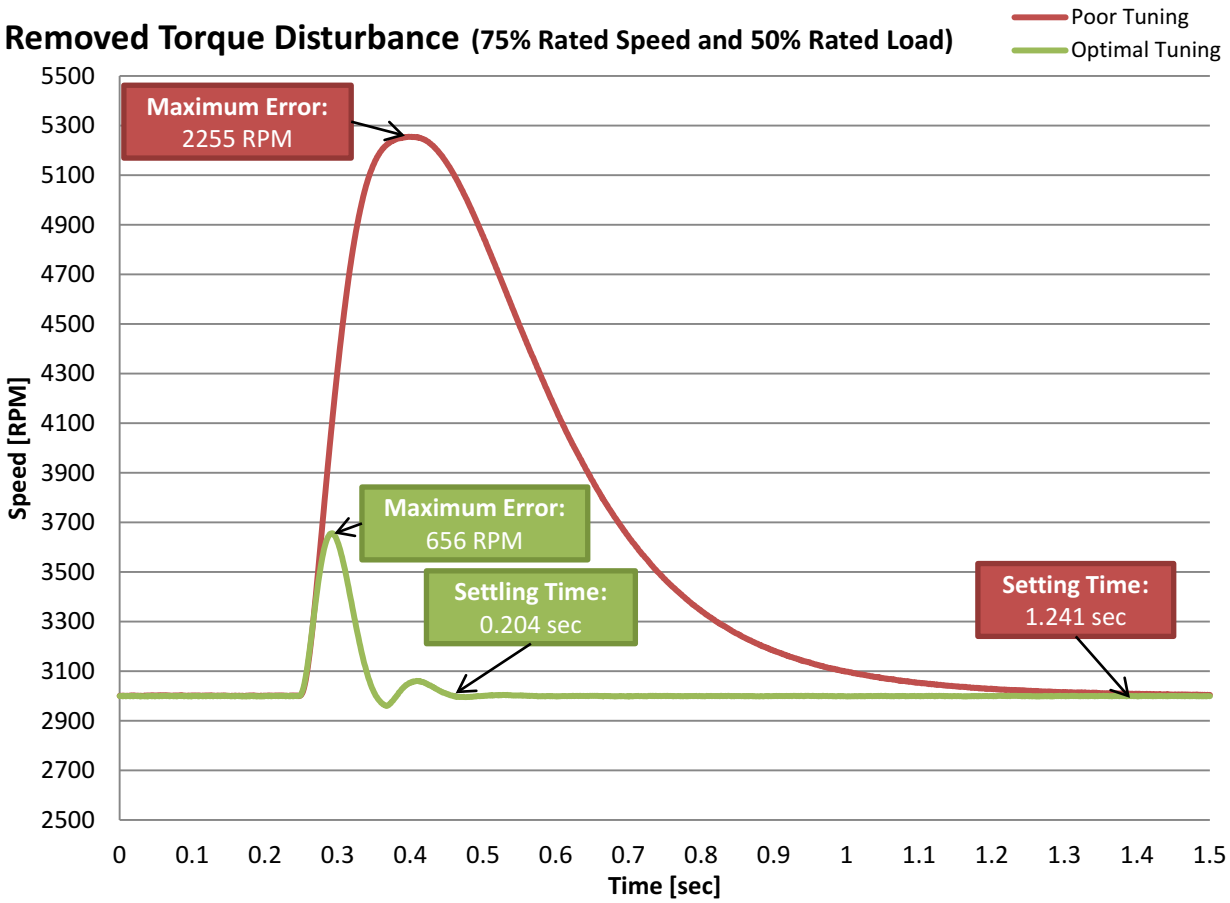


图 **12-9.** 移除转矩干扰的速度调整比较

当进行抗扰测试时，在多个速度和负载组合下进行测试很重要。 当被置于不同的饱和状态时，速度控制器具有不同的性能特点。 为了完全地评估您速度控制器的效率，应该在整个应用范围内完成测试。 这些测试结果将表示此控制器是否符合应用技术规格的要求，或者是否需要针对不同的运行点进行多次调整。

能够创建可重复的干扰也很重要。 这一要求可通过使用一个测力计或扰动电机来实现。 在评估多个控制器时，创建可重复干扰是一个重要因素。 如果测试条件不能被复制，那么就很难充分地比较两个控制器的响应。

12.3.4 系统配置跟踪

系统配置跟踪用于测试控制器跟随变化的目标速度的程度。 这个测试中评估的两个衡量标准是最大误差和绝对平均误差。 最大速度误差显示速度变化时，控制器过冲的大小。 这个值表示您控制器调整的积极程度。 如果控制器的调整不够积极主动，速度将超过目标值，并且将需要花费较长的时间恢复。 如果控制器的调整过于积极主动，它将过冲，然后在稳定至目标速度时发生振荡。 如果控制器被正确调整，它将过冲，然后平稳返回至目标速度。

绝对平均误差是整个系统配置范围内瞬时速度误差绝对平均值。 这个测量结果显示整个系统配置上的偏离量。 它将电机运行时的全部小值误差考虑在内。 如果控制器的调整过于积极，它将导致较大的绝对平均误差，这是由控制器将在整个系统配置内振荡所造成的。 如果控制器的调整不够积极，它将导致较大的绝对平均误差，这是因为它始终落后于系统配置对电机的指令操作。

图 12-10 显示了同一控制器默认调整与最优调整的差异。 可以看出，通过调整速度控制器，可以使运动系统更加紧密地跟踪参考。 通过调整控制器，可以显著降低最大误差、绝对平均误差和最大过冲。

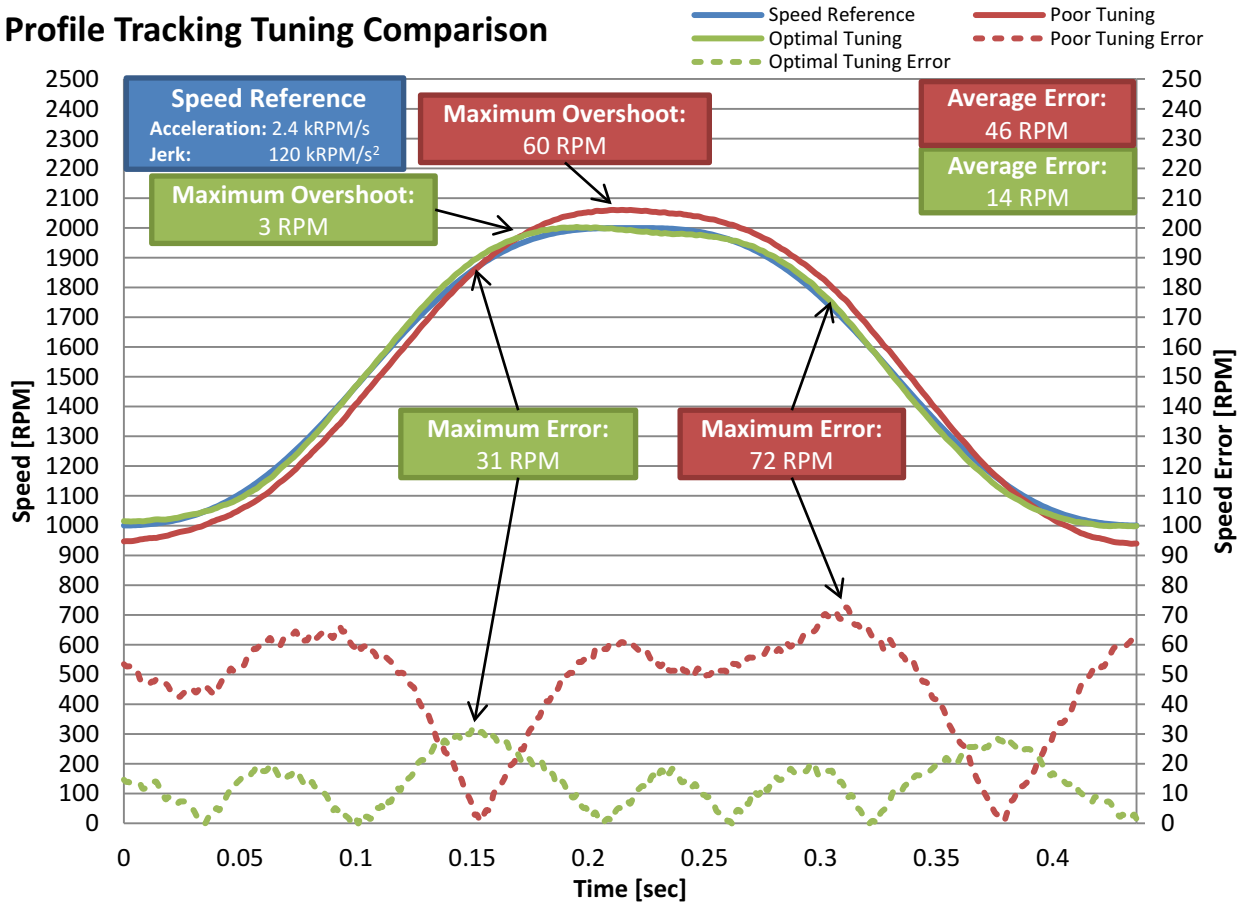


图 **12-10.** 系统配置跟踪的速度调整比较

在您的系统配置中测试多个速度和加速度以及多个不同负载同样重要。 当被置于不同情况下，速度控制器具有不同的性能特点。 为了完全评估您的速度控制器的效率，应该在整个应用范围内执行测试。 这包括您何时为测试设计系统配置。 设计时应该注意，以确保在系统配置中建立应用速度和加速度。 这些测试结果将告诉您控制器是否满足应用技术规格，或者您的控制器是否需要针对不同的运行点进行多次调整。

创建可重复系统配置和负载也很重要。 可使用 SpinTAC 速度移动和 SpinTAC 速度规划创建可重复系统配置；有关详细信息，请参见Chapter 13。 需要使用可重复系统配置，这样便可使用顺序一致的同一基准，在相同的时间长度内测试全部控制器。 这样确保了测试条件尽可能一样。 在系统配置跟踪测试期间施加负载时，创建可重复干扰很重要。 这一要求可通过使用一个测力计或扰动电机来实现。 在评估控制器时，创建可重复干扰是一个重要因素。 如果测试条件不能被复制，那么就很难充分地比较两个控制器的响应。

12.3.5 InstaSPIN-MOTION 速度控制优势

12.3.5.1 单个参数调整

InstaSPIN-MOTION 在实现应用的最优性能方面具有多种优势。 SpinTAC 速度控制在宽工作范围内提供单个参数调整，从而帮助您实现最优性能。 通过使用单个调整参数，可以根据应用的正确调整设置快速进行调整。 基于 SpinTAC 速度控制的核心功能，即主动抗扰控制 (ADRC)，单个调整参数适用于很宽的工作范围。 在大多数情况下，单个调整设置适用于应用的整个工作范围。 相比之下，PI 控制器至少需要两个调整参数，而且需要为应用中多个不同的运行点调整这两个参数， 因此调整应用所花费的时间要远远大于通过 SpinTAC 速度控制进行调整的时间。

为了比较 SpinTAC 速度控制和传统 PI 速度控制器之间的差异，我们连接了 Anaheim Automation BLY172S 电机（随附 DRV8312 修订版 D 评估套件）和 Magtrol HD-400 测力计。 使用11.5 节中的示例中的 PI 调整参数。 以下特性用于计算速度 PI 增益：

- Rs = 0.4Ω
- Ls = 0.6mH
- 反电势 = 0.0054V-s/rad
- 惯性 = 335E-4 kg-m2（包括电机和测力计的惯性）
- 转子极数 = 8
- 速度带宽 = 800rad/s
- 阻尼因子 = 4

这会导致以下速度 PI 增益：

- spdKpseries = 5.495
- spdKiseries = 132.88

使用12.1.4 节中概括的方法通过实验调整 SpinTAC 速度控制。 该调整产生了以下增益：

- 带宽 = 45rad/s

以上 PI 速度控制器和 SpinTAC 速度控制的增益已用于以下测试。

12.3.5.2 抗扰测试

SpinTAC 速度控制中的 ADRC 技术具有出色的抗扰性能。 它可以主动地实时评估干扰并对这些干扰进行补偿（请参见图 12-11 和图 12-12）。 SpinTAC 速度控制检测到系统中的干扰后，将实施更正以便快速平滑地将电机速度调整为目标速度。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

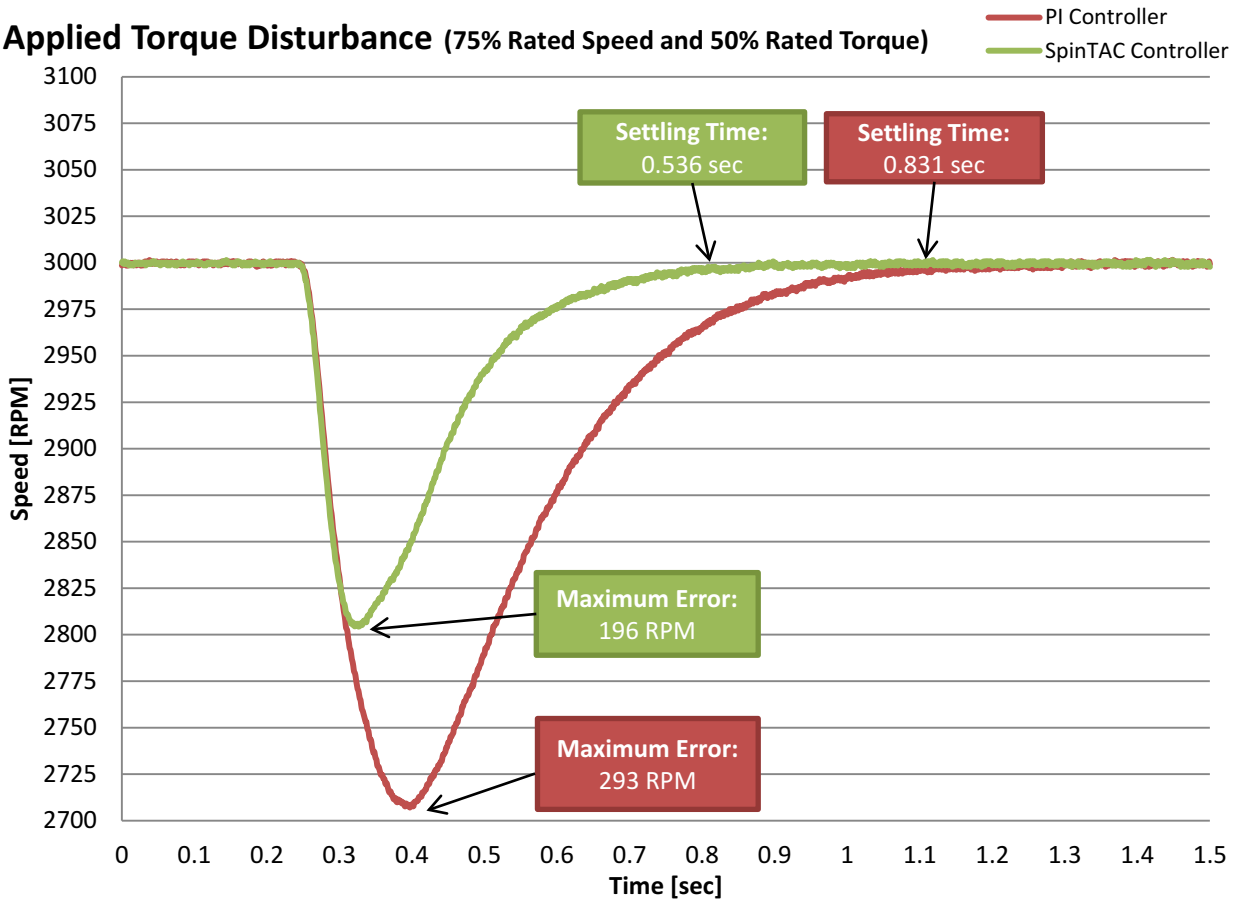


图 **12-11.** 施加转矩干扰时的 **PI** 和 **SpinTAC** 速度控制比较

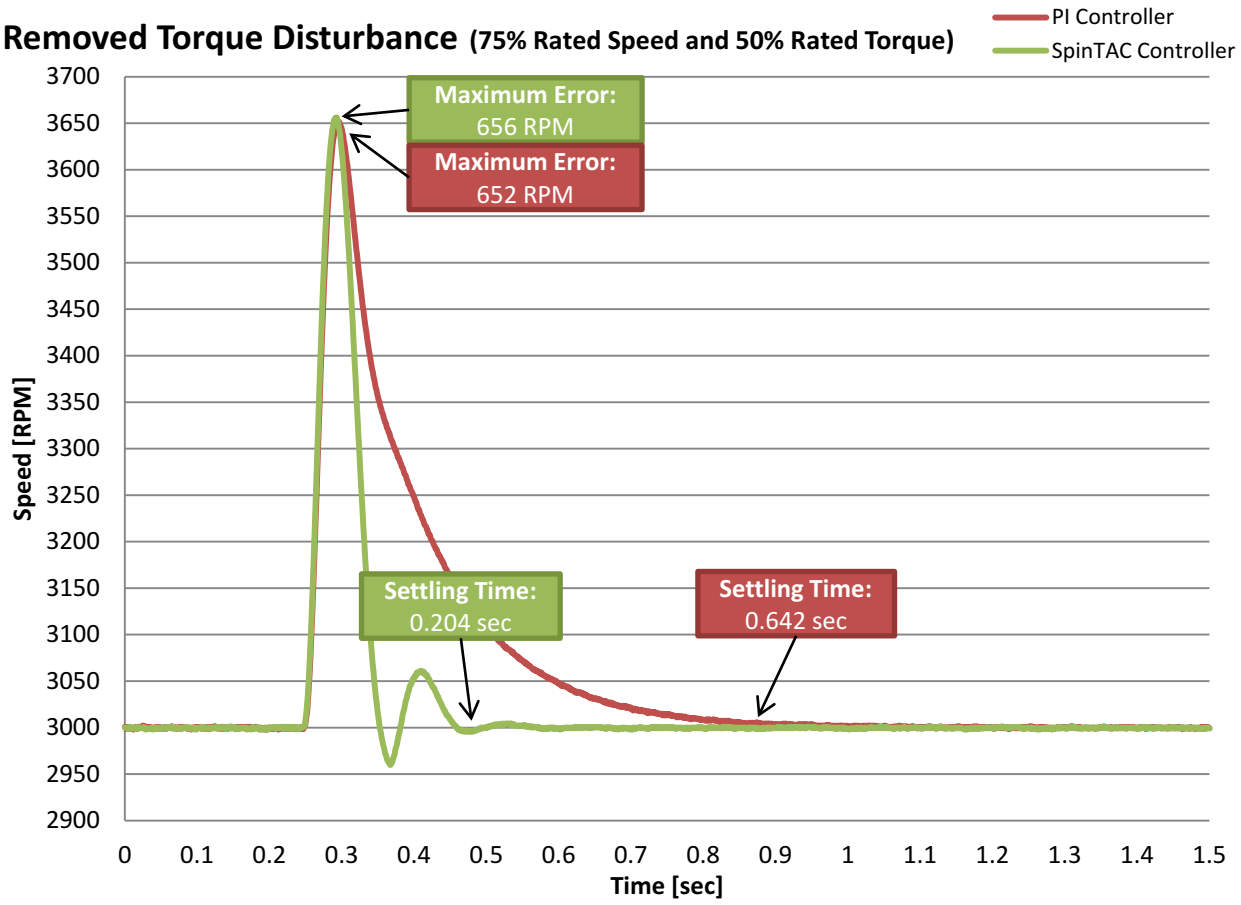


图 **12-12.** 移除转矩干扰时的 **PI** 和 **SpinTAC** 速度控制比较

与传统 PI 控制器相比，SpinTAC 速度控制具有更短的恢复时间和更小的最大误差。 这样便可在应用时减少速度波动。 减少速度波动使应用能够在更稳定的速度下运行，同时能够减少应用的功耗。

12.3.5.3 前馈

SpinTAC 速度控制还具有前馈功能， 因而可实现出色的系统配置（请参见图 12-13）。 前馈用于通知 SpinTAC 速度控制所需的加速或减速的加速度， 因此，与 PI 控制器相比，SpinTAC 速度控制可以更快地响应系统配置变化， 从而减小最大误差和绝对平均误差。

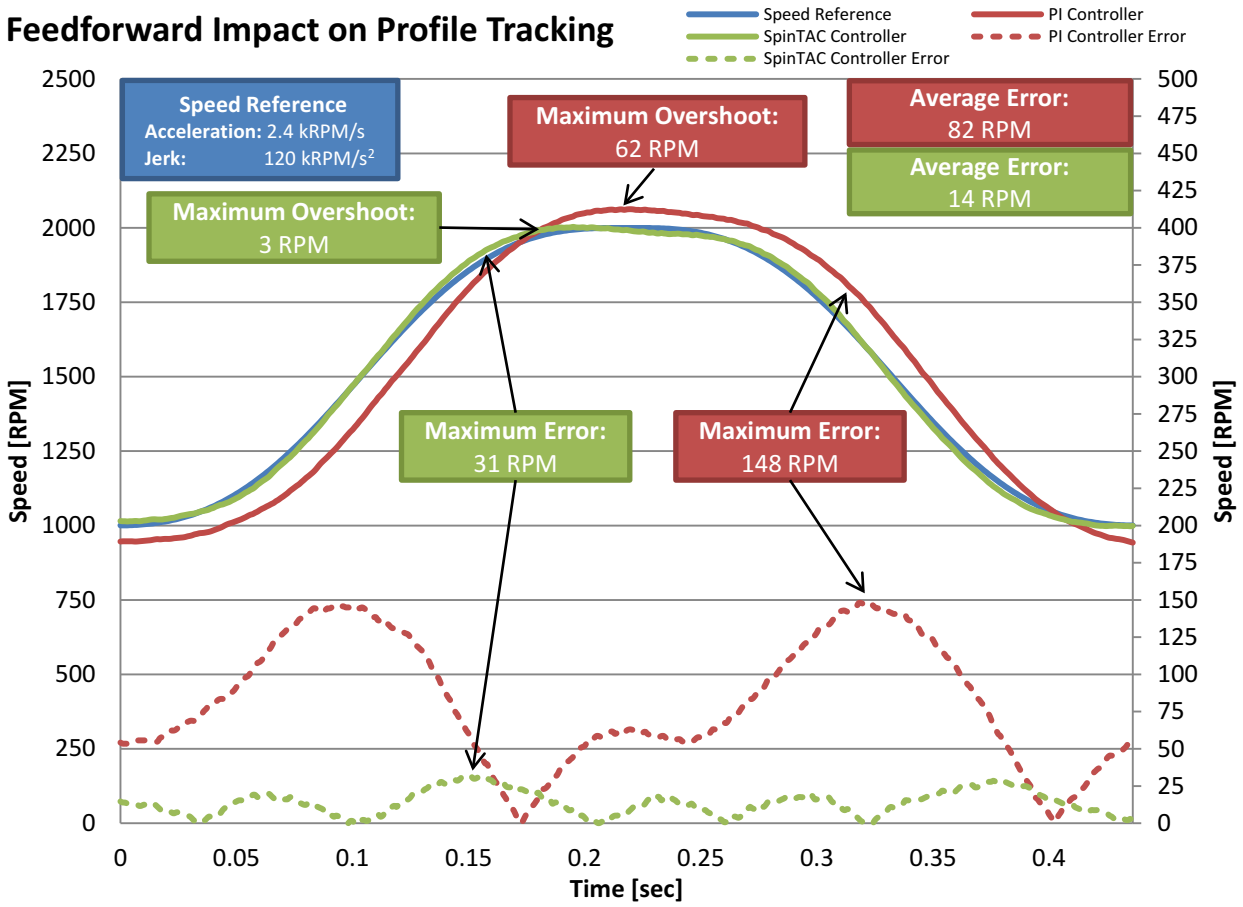


图 **12-13. PI** 和 **SpinTAC** 中前馈对速度系统配置跟踪的影响比较

SpinTAC 速度控制具有较小的最大误差和绝对平均误差， 因此，与传统 PI 控制器相比显著提高了跟踪性能， 从而可以在控制器跟踪变化的参考时减少不必要的运动和能源浪费。 该功能与节 12.3.5.2中所述的抗扰能力结合使用时更能凸显其重要性。 如果系统在跟踪变化的速度参考时出现干扰，则可能会导致大量过冲以及能源或材料浪费。

12.3.5.4 无积分器饱和

积分器饱和问题是指标准 PI 控制器的积分器组件积累大幅度误差。 当积分器进入饱和状态且速度中存在稳定状态误差时会发生该问题。 此稳定状态误差将继续增大积分器中的值，将导致饱和的条件移除后，该误差将导致速度大幅过冲速度参考。 SpinTAC 速度控制不存在该问题。 ADRC 技术可实时评估系统误差，无需依赖任何可能造成积分器饱和问题的积分器。

图 12-14 显示了传统 PI 控制器遇到积分器饱和问题的情况。 这种情况下，电机无法克服转矩干扰，只能以低于设定点的速度运行。 这会将控制器置于饱和状态，使 PI 控制器的积分器项随时间增大。 移除该转矩干扰后，PI 的积分器项将导致产生极大的过冲，这会大大增加返回速度设定点所花费的时间。 SpinTAC 控制器不会带来任何此类负面影响，因为它不包含可导致积分器饱和问题的积分器项。 比较图 12-14 和图 12-12 也很有趣。 可以看到，对于 50% 和 80% 的额定转矩干扰，SpinTAC 对于移除转矩干扰的响应十分类似。

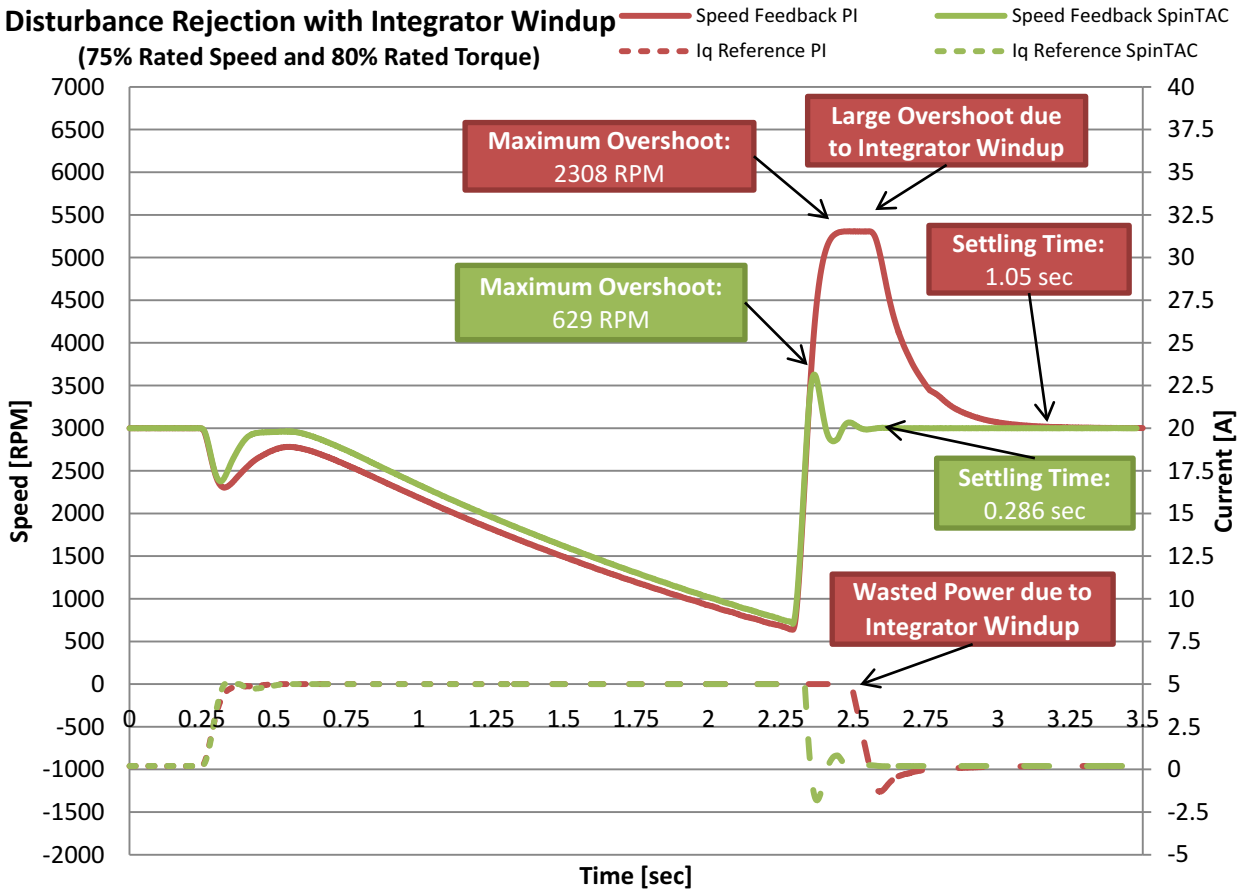


图 **12-14.** 抗扰期间 **PI** 和 **SpinTAC** 速度控制的积分器饱和比较

12.3.5.5 最小启动过冲

InstaSPIN-MOTION 具有在启动期间产生最小过冲的控制器， 因此可减少应用在启动电机时消耗的能源以及缩短恢复和运行在目标速度所需的时间。 对于循环开关的压缩机来说，这可以显著节省能源。

图 12-15 比较了 SpinTAC 速度控制和传统 PI 速度控制器的阶跃响应。 从图中可以看出，传统 PI 速度控制器的过冲和稳定时间要远远大于 SpinTAC 速度控制。 PI 速度控制器产生的较大过冲会导致能源浪费和不必要的运动， 因此不是理想系统响应。 SpinTAC 速度控制具有更佳的响应，可以减小过冲和缩短稳定时间。

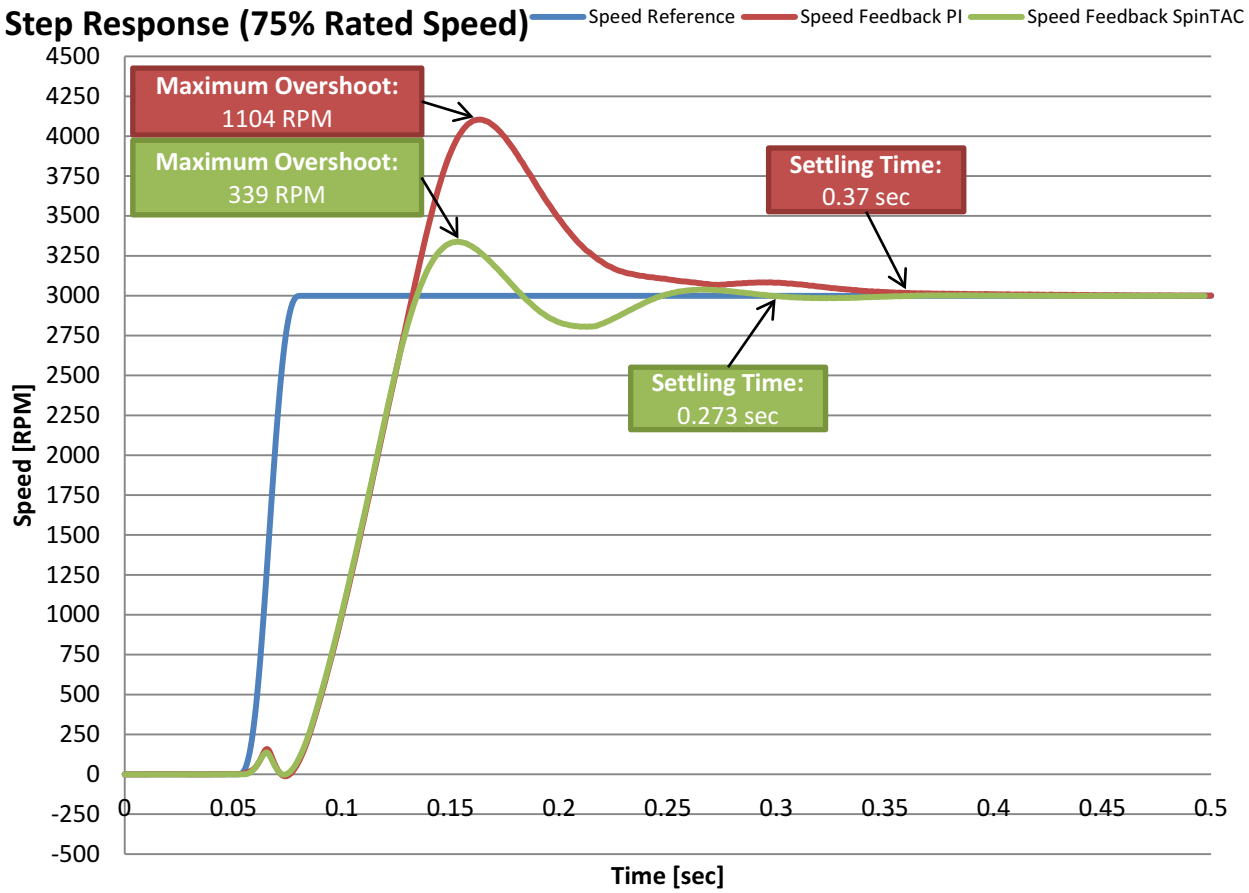


图 **12-15. PI** 和 **SpinTAC** 速度控制的阶跃响应比较

12.3.5.6 结论

InstaSPIN-MOTION 中包括的 SpinTAC 速度控制可直接替代传统 PI 速度控制器。 它可在应用的整个工作范围内获得更佳的性能。 由于使用单个调整参数，因此与 PI 控制器相比，调整操作更为简单。 这样便可在开发时减小调整速度控制器方面的工作量，使您能够重点关注应用的其它部分。 SpinTAC 速度控制可减少过冲和缩短稳定时间， 还具有更佳的系统配置跟踪功能。 所有这些功能结合起来，可以降低应用中不必要运动所造成的能源消耗。

12.4 SpinTAC 位置控制的软件配置

配置 SpinTAC 位置控制需要四个步骤。 实验 13a“调整 InstaSPIN-MOTION 位置控制器”是一个示例项目，在其中执行了使用 SpinTAC 位置控制所需的步骤。 利用包含在 MotorWare 中的头文件 spintac.h，您可以快速将 SpinTAC 组件包括在项目中。

12.4.1 包括头文件

这应该通过其余的模块头文件包含来完成。 在实验 13a 示例项目中，该文件包含在 spintac_position.h 头文件中。 针对相应项目，可通过包括 spintac_position.h 完成此步骤。

```
#include "sw/modules/spintac/src/32b/spintac_pos_ctl.h"
```

12.4.2 声明全局结构

这应该通过主源文件中的全局变量声明来完成。 在实验 13a 项目中，此结构包括在已声明为 spintac_position.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj       st_obj;    // The SpinTAC Object
ST_Handle stHandle;  // The SpinTAC Handle
```

此示例为不希望使用已在 spintac_position.h 头文件中声明的 ST_Obj 结构的情况。

```
ST_PosCtl_t       stPosCtl;       // The SpinTAC Position Controller object
ST_POSCTL_Handle stPosCtlHandle; // The SpinTAC Position Controller Handle
```

12.4.3 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这样会将所有的默认值加载到 SpinTAC 位置控制中。 该步骤可通过运行已在 spintac_position.h 头文件中声明的函数 ST_init 和 ST_setupPosCtl 来完成。 如果不希望使用这两个函数，可使用下方的代码示例配置 SpinTAC 位置控制组件。 SpinTAC 位置控制配置是典型配置的代表，适用于大部分电机。

```
// Initialize SpinTAC Position Control
stPosCtlHandle = STPOSCTL_init(&stPosCtl, sizeof(stPosCtl));

// Setup the maximum current in PU
_iq maxCurrent_PU = _IQ(USER_MOTOR_MAX_CURRENT / USER_IQ_FULL_SCALE_CURRENT_A);

// Instance of the position controller
STPOSCTL_setAxis(stPosCtlHandle, ST_AXIS0);
// Sample time [s], (0, 1)
STPOSCTL_setSampleTime_sec(stPosCtlHandle, _IQ(ST_SAMPLE_TIME));
// System inertia upper (0, 127.9999] and lower (0, InertiaMax] limits [PU/(pu/s^2)]
STPOSCTL_setInertiaMaximums(stPosCtlHandle, _IQ(10.0), _IQ(0.001));
// System velocity limit (0, 1.0] [pu/s]
STPOSCTL_setVelocityMaximum(obj->;posCtlHandle, _IQ24(1.0));
// System control signal high (0, 1] & low [-1, 0) limits [PU]
STPOSCTL_setOutputMaximums(stPosCtlHandle, maxCurrent_PU, -maxCurrent_PU);
// System maximum (0, 1.0] and minimum [-1.0, 0) velocity [pu/s]
STPOSCTL_setVelocityMaximums(stPosCtlHandle, _IQ(1.0), _IQ(-1.0));
// System maximum value for mechanical revolutions [MRev]
STPOSCTL_setPositionRolloverMaximum_mrev(stPosCtlHandle, _IQ24(ST_MREV_ROLLOVER));
// Sets the values used for converting between pu and MRev
STPOSCTL_setUnitConversion(stPosCtlHandle, USER_IQ_FULL_SCALE_FREQ_Hz,
                                                    USER_MOTOR_NUM_POLE_PAIRS);
// System maximum allowable error [MRev]
STPOSCTL_setPositionErrorMaximum_mrev(stPosCtlHandle,
                                                _IQ24(ST_POS_ERROR_MAXIMUM_MREV));
// Disturbance type {true: Ramp; false: Square}
STPOSCTL_setRampDisturbanceFlag(stPosCtlHandle, false);
// System upper (0, 0.1/(cfg.T_sec*20)] and lower [0, BwScaleMax] limits for BWScale
STPOSCTL_setBandwidthScaleMaximums(stPosCtlHandle,
                                _IQ24((0.1) / (ST_SAMPLE_TIME * 20.0)), _IQ24(0.01));
// Enables the feedback filter
STPOSCTL_setFilterEnableFlag(stPosCtlHandle, true);
// System inertia [PU/(pu/s^2)], [InertiaMin, InertiaMax]
STPOSCTL_setInertia(stPosCtlHandle, _IQ(USER_MOTOR_INERTIA));
```

```
// Controller bandwidth scale [BwScaleMin, BwScaleMax]
STPOSCTL_setBandwidthScale(stPosCtlHandle, _IQ24(USER_SYSTEM_BANDWIDTH_SCALE));
// Initially ST_PosCtl is not enabled
STPOSCTL_setEnable(stPosCtlHandle, false);
// Initially ST_PosCtl is not in reset
STPOSCTL_setReset(stPosCtlHandle, false);
```

12.4.4 调用 SpinTAC 位置控制

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此函数。 抽取率由 ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。 在调用 SpinTAC 位置控制函数之前，必须更新位置参考、速度参考、加速度参考和位置反馈。 本示例使用 SpinTAC 位置移动为 SpinTAC 位置控制提供参考。 有关 SpinTAC 位置移动的详细信息，请参见Chapter 13。

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;    // Get pointer to CTRL object
// Get the mechanical speed in pu/s
_iq speedFeedback = EST_getFm_pu(obj->estHandle);
STPOSCTL_setPositionReference_mrev(stPosCtlHandle,
                                   STPOSMOVE_getPositionReference_mrev(stPosMoveHandle));
// Update the Velocity Reference
STPOSCTL_setVelocityReference(stPosCtlHandle,
                                      STPOSMOVE_getVelocityReference(stPosMoveHandle));
// Update the Acceleration Reference
STPOSCTL_setAccelerationReference(stPosCtlHandle,
                                    STPOSMOVE_getAccelerationReference(stPosMoveHandle));
// Update the Position Feedback
STPOSCTL_setPositionFeedback_mrev(stObj->posCtlHandle,
                                        STPOSCONV_getPosition_mrev(stPosConvHandle));
// Run SpinTAC Position Control
STPOSCTL_run(stPosCtlHandle);

// Get the Torque Reference from SpinTAC Position Control
iqReference = STPOSCTL_getTorqueReference(stPosCtlHandle);

// Set the Iq reference that came out of SpinTAC Position Control
CTRL_setIq_ref_pu(ctrlHandle, iqReference);
```

12.4.5 SpinTAC 位置控制故障排除

12.4.5.1 ERR_ID

ERR_ID 可向用户提供错误代码。 表 12-3 中列出了为 SpinTAC 位置控制定义的错误及相应解决方法。

表 12-3. SpinTAC 位置控制 ERR_ID 代码

ERR_ID	问题	解决方法
1	采样时间值无效	将 cfg.T_sec 设置为 (0, 1] 范围内
2	最大速度参考值无效	将 cfg.VelMax 设置为 (0, 1] 范围内
4	最大控制信号值无效	将 cfg.OutMax 设置为 (0, 1] 范围内
5	最小控制信号值无效	将 cfg.OutMin 设置为 [-1, 0) 范围内
13	位置翻转界限值无效	将 cfg.ROMax_mrev 设置为 [2, 100] 范围内
14	机械旋转 [MRev] 至 [pu] 的比率值无效	将 cfg.mrev_TO_pu 设置为 [0.002, 1] 范围内
15	最大位置误差值无效	将 cfg.PosErrMax_mrev 设置为 [0, cfg.ROMax_mrev/2] 范围内
16	最大惯性值无效	将 cfg.InertiaMax 设置为正 _iq24 值

表 12-3. SpinTAC 位置控制 ERR_ID 代码 (continued)

ERR_ID	问题	解决方法
17	最小惯性值无效	将 cfg.InertiaMin 设置为 (0, cfg.InertiaMax] 范围内
18	最大带宽值无效	将 cfg.BwMax 设置为 [0, min(2000, 0.2/cfg.T)] 范围内
19	最小带宽值无效	将 cfg.BwMin 设置为 [0, cfg.BwMax] 范围内
20	指定的干扰值无效	将 cfg.RampDist 设置为 {false, true} 范围内
32	轴 ID 无效	将 cfg.Axis 设置为 {ST_AXIS0, ST_AXIS1} 范围内
35	滤波器启用值无效	将 cfg.FiltEN 设置为 {false, true} 的范围内
1012	惯性值无效	没有动作。 惯性饱和至界限 [cfg.InertiaMin, cfg.InertiaMax]
1014	带宽 × 惯性大于 2000	没有动作。 实际带宽饱和至值 2000/惯性
1016	摩擦超出限值	没有动作。 摩擦饱和至调整摩擦范围 [0, 5]
2002	位置误差超出最大值	增加带宽以减小位置误差或降低系统配置速度、加速度和急动
4001	SpinTAC 许可证无效	使用具有有效许可证的芯片
4003	ROM 版本无效	使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。

12.5 位置控制中的最优性能

12.5.1 简介

从您的机械运动系统中获得最佳性能是十分重要的。 未经良好调节的稳压器会导致能源浪费、材料浪费，或者使系统不稳定。 为了确定位置控制器在您的应用中工作状态如何，有必要在很多不同的运行点上评估位置控制器的性能。

12.5.2 比较位置控制器

可根据很多不同的因素对位置控制器进行比较。 然而，两个衡量标准——抗扰和系统配置跟踪——可被用来测试性能，并且确定您的控制器对于您的应用被调整的如何。

12.5.3 抗扰

抗扰用于测试控制器对影响电机速度和位置的外部干扰的抵抗能力。 使用最大误差和稳定时间来测量抗扰。最大误差显示从目标位置的偏离，并且表示出对您的控制器进行调整的积极程度。 积极调整将产生一个低最大误差。

稳定时间是指从干扰发生的时间点到位置返回至一个目标位置附近的固定范围内所需要的时间。 这也表示出您的控制环路调整的积极程度。 如果控制器的调整过于积极，控制器的稳定时间将会比较长，这是因为控制器将在稳定前在目标位置附近振荡。

图 12-16 和图 12-17 显示了同一控制器不良调整与最优调整的差异。 可以看出，通过对位置控制器进行调整，在向电机系统施加转矩或从电机系统中移除转矩时，最优调整的控制器可以显著减小最大误差和稳定时间。 本示例有所夸大，但它是为了强调对系统进行良好调整的重要性。

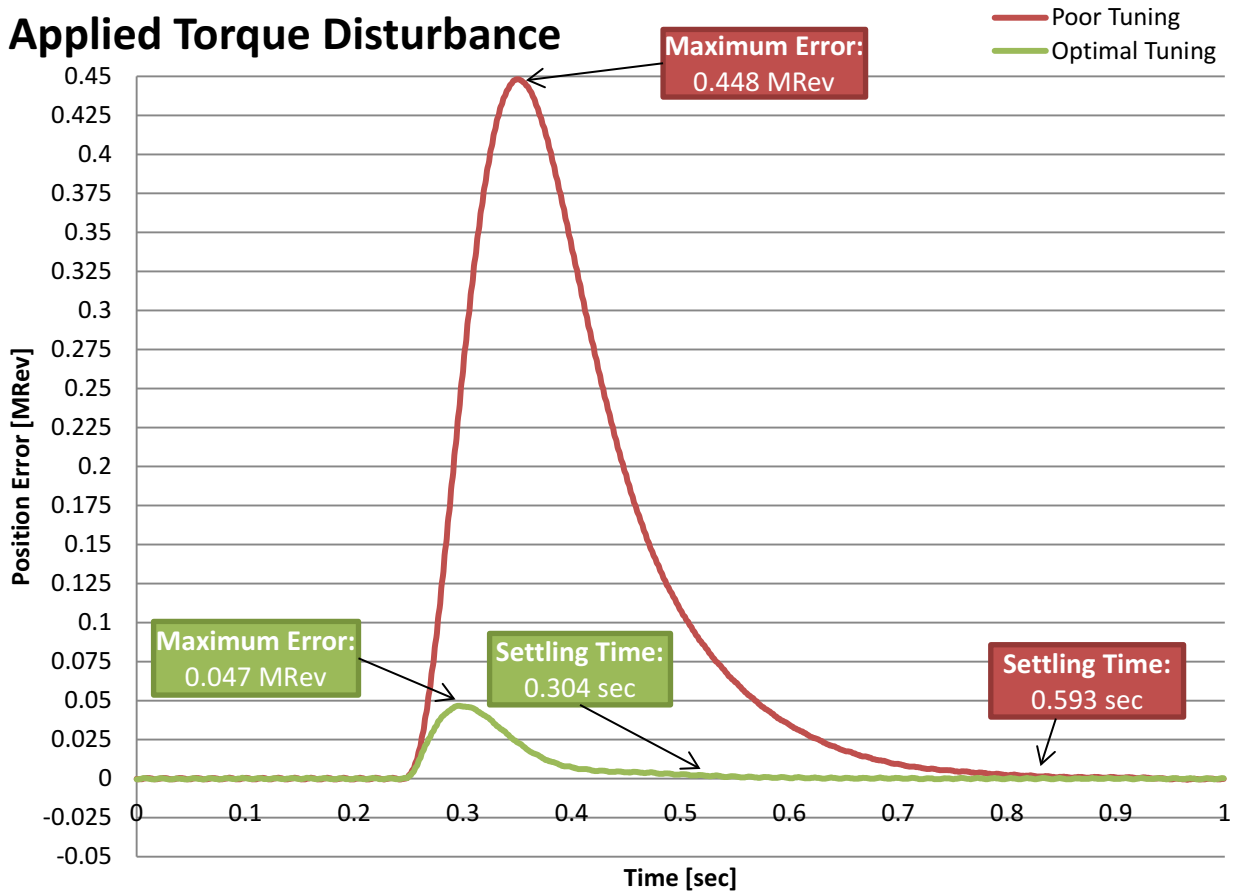


图 **12-16.** 施加转矩干扰的位置调整比较

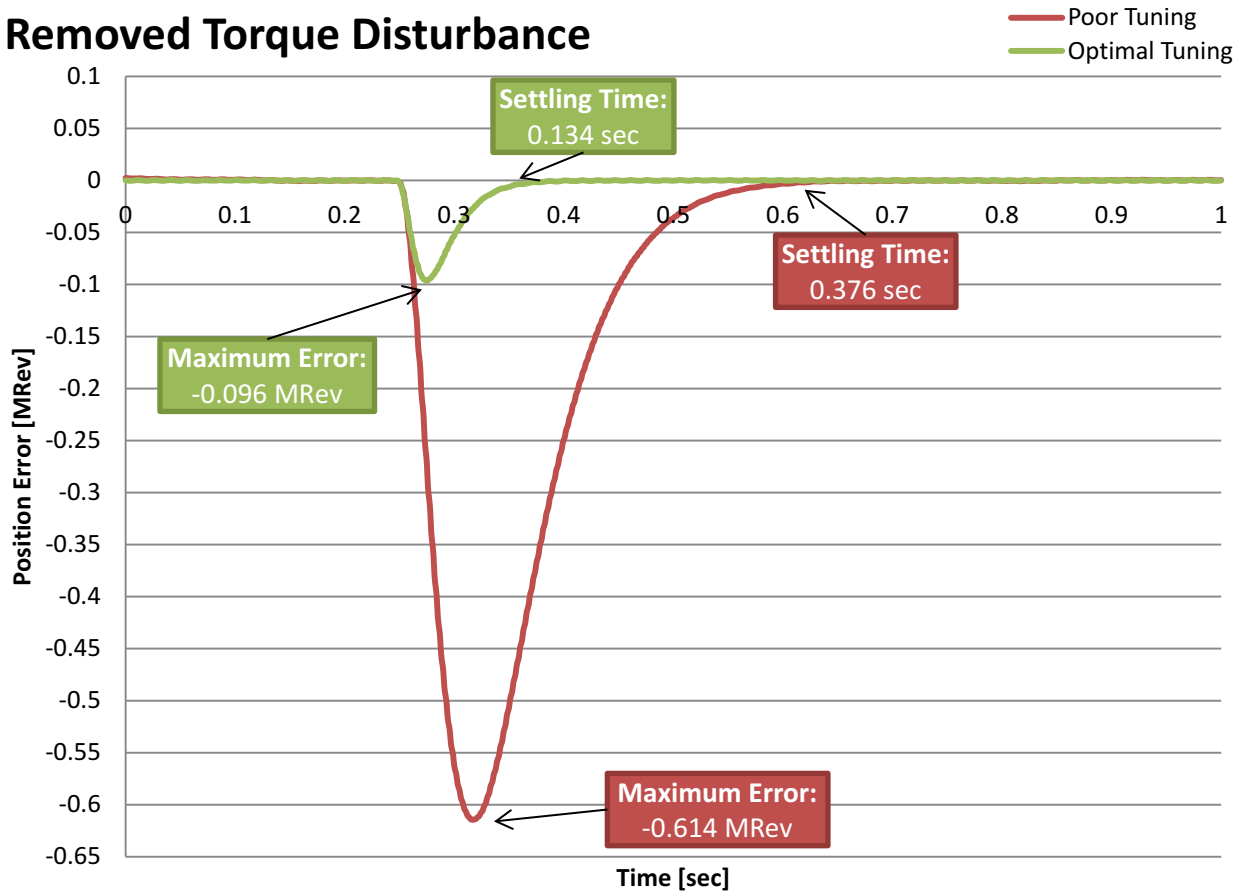


图 **12-17.** 移除转矩干扰的位置调整比较

当进行抗扰测试时，在多个速度和负载组合下进行测试很重要。 当被置于不同情况下，位置控制器具有不同的性能特点。 为了正确地评估位置控制器的效率，应该在整个应用范围内完成测试。 这些测试结果将表示此控制器是否符合应用技术规格的要求，或者是否需要针对不同的运行点进行多次调整。

能够创建可重复的干扰也很重要。 这一要求可通过使用一个测力计或扰动电机来实现。 在评估多个控制器时，创建可重复干扰是一个重要因素。 如果测试条件不能被复制，那么就很难充分地比较两个控制器的响应。

12.5.4 系统配置跟踪

系统配置跟踪用于测试控制器跟随变化的目标位置的程度。 这个测试中评估的衡量标准是最大误差和绝对平均误差。 最大误差显示位置变化时控制器过冲的大小。 这个值表示控制器调整的积极程度。 如果控制器的调整不够积极主动，位置将超过目标值，并且将需要花费较长的时间恢复。 如果控制器的调整过于积极主动，它将过冲，然后在稳定至目标位置时发生振荡。 如果控制器被正确调整，它将过冲，然后平稳返回至目标位置。

绝对平均误差是整个系统配置范围内瞬时误差绝对平均值。 这个测量结果显示整个系统配置上的偏离量。它将电机运行时的全部小值误差考虑在内。 如果控制器的调整过于积极，它将导致较大的绝对平均误差，这是由控制器将在整个系统配置内振荡所造成的。 如果控制器的调整不够积极，它将导致较大的绝对平均误差，这是因为它始终落后于系统配置对电机的指令操作。

图 12-18 显示了同一控制器默认调整与最优调整的差异。 可以看出，通过调整位置控制器，可以使运动系统更加紧密地跟踪参考。 通过调整控制器，可以显著降低最大误差、绝对平均误差和最大过冲。

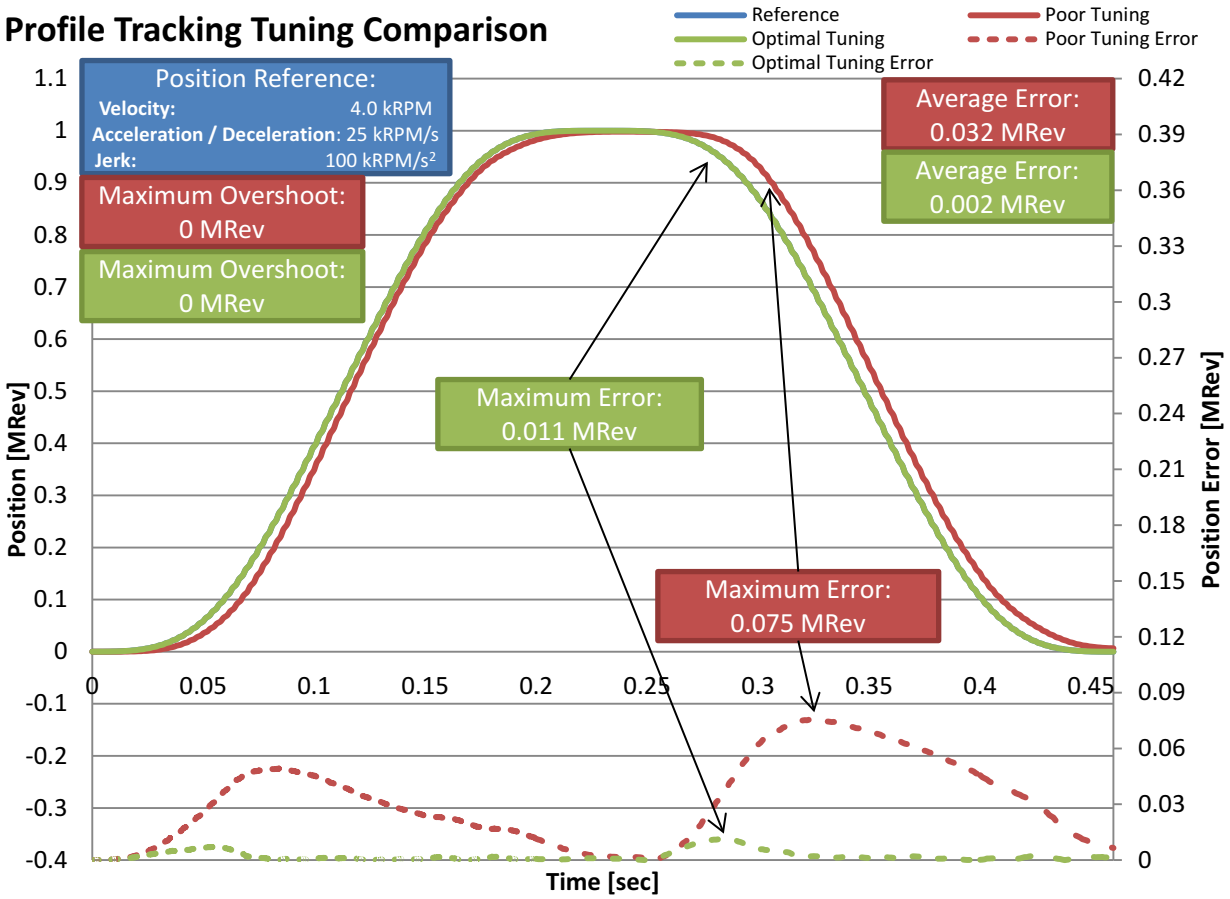


图 **12-18.** 系统配置跟踪的位置调整比较

在您的系统配置中测试多个速度和加速度以及多个不同负载同样重要。 当被置于不同情况下，位置控制器具有不同的性能特点。 为了正确地评估位置控制器的效率，应该在整个应用范围内完成测试。 这包括您何时为测试设计系统配置。 设计时应该注意，以确保在系统配置中建立应用速度和加速度。 这些测试结果将告诉您控制器是否满足应用技术规格，或者您的控制器是否需要针对不同的运行点进行多次调整。

创建可重复系统配置和负载也很重要。 可使用 SpinTAC 位置移动和 SpinTAC 位置规划创建可重复系统配置；有关详细信息，请参见Chapter 13。 需要使用可重复系统配置，这样便可使用顺序一致的同一基准，在相同的时间长度内测试全部控制器。 这样确保了测试条件尽可能一样。 在系统配置跟踪测试期间施加负载时，创建可重复干扰很重要。 这一要求可通过使用一个测力计或扰动电机来实现。 在评估控制器时，创建可重复干扰是一个重要因素。 如果测试条件不能被复制，那么就很难充分地比较两个控制器的响应。

12.5.5 InstaSPIN-MOTION 位置控制优势

12.5.5.1 单个参数调整

InstaSPIN-MOTION 在实现应用的最优性能方面具有多种优势。 传统 PI 位置控制器需要三个级联控制环路，分别用于电流、速度和位置，而 SpinTAC 位置控制仅需要两个环路，一个用于电流，另一个用于组合的位置-速度环路（请参见表 12-4）。 由于具有这些级联控制环路，PI 控制器至少需要四个用于速度和位置的调整参数，在应用中的每个运行点都需要对所有这些参数进行调整。

表 12-4. InstaSPIN-MOTION 位置控制优势

控制环路	传统 PI 控制	SpinTAC 位置控制
电流	参数识别期间自动确定	参数识别期间自动确定
速度	提供建议的起始值，但需要通过调整和测试进行验证。 相关计算在11.5 节中提供。	单个参数调整在整个工作范围内均有效。 可通过单个参数调整位置和速度，在整个工作范围内有效。
位置	不提供起始值。 不提供计算。	

SpinTAC 位置控制通过提供位置和速度的单个参数调整帮助您实现最优性能。 通过使用单个调整参数，可以根据应用的正确调整设置快速进行调整。 基于 SpinTAC 位置控制的核心功能，即主动抗扰控制 (ADRC)，单个调整参数适用于很宽的工作范围。 SpinTAC 位置控制可减少优化应用所需的时间和复杂度。

为了比较 SpinTAC 位置控制和传统 PI 控制系统的差异，需要将 Teknic M2310PLN04K 电机（可从 TI 网上商店购买）与 Magtrol HD-400 测力计结合使用。 将通过11.5 节中调整示例确定的 PI 调整参数用作起始点。 为了调整位置 PI 稳压器，必须重新调整速度 PI 稳压器。 这是一个迭代过程。 每次修改速度增益，都会评估其对位置增益的影响。

使用12.4 节中概括的方法通过实验调整 SpinTAC 位置控制。 在调整 SpinTAC 位置控制之前，通过实验 05c 中概括的步骤识别系统惯性。

12.5.5.2 抗扰

SpinTAC 位置控制中的 ADRC 技术具有出色的抗扰性能。 它可以主动地实时评估干扰并对这些干扰进行补偿（请参见图 12-19 和图 12-20）。 SpinTAC 位置控制检测到系统中的干扰后，将实施更正以便快速平滑地将电机位置调整为目标位置。

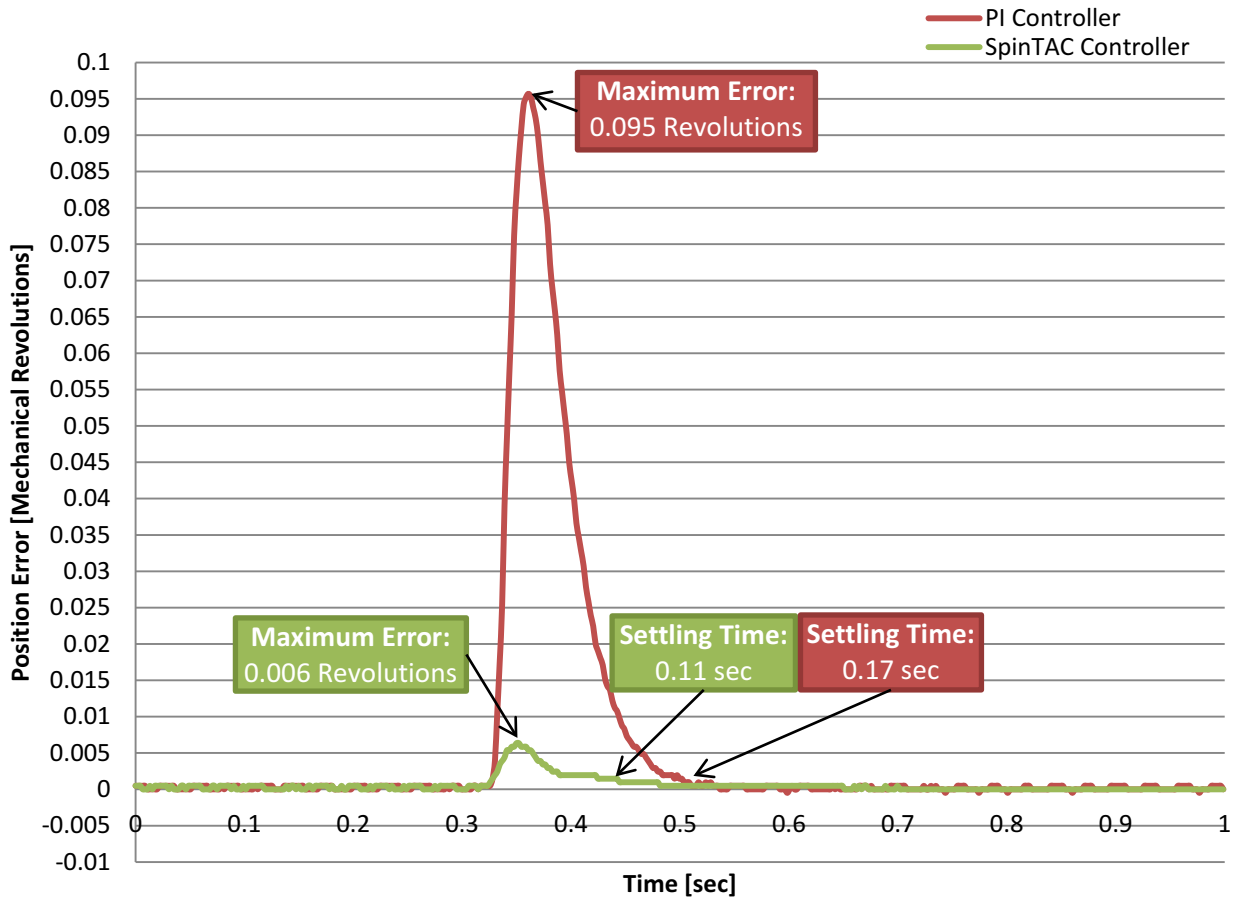


图 **12-19.** 施加转矩干扰时的 **PI** 和 **SpinTAC** 位置控制比较

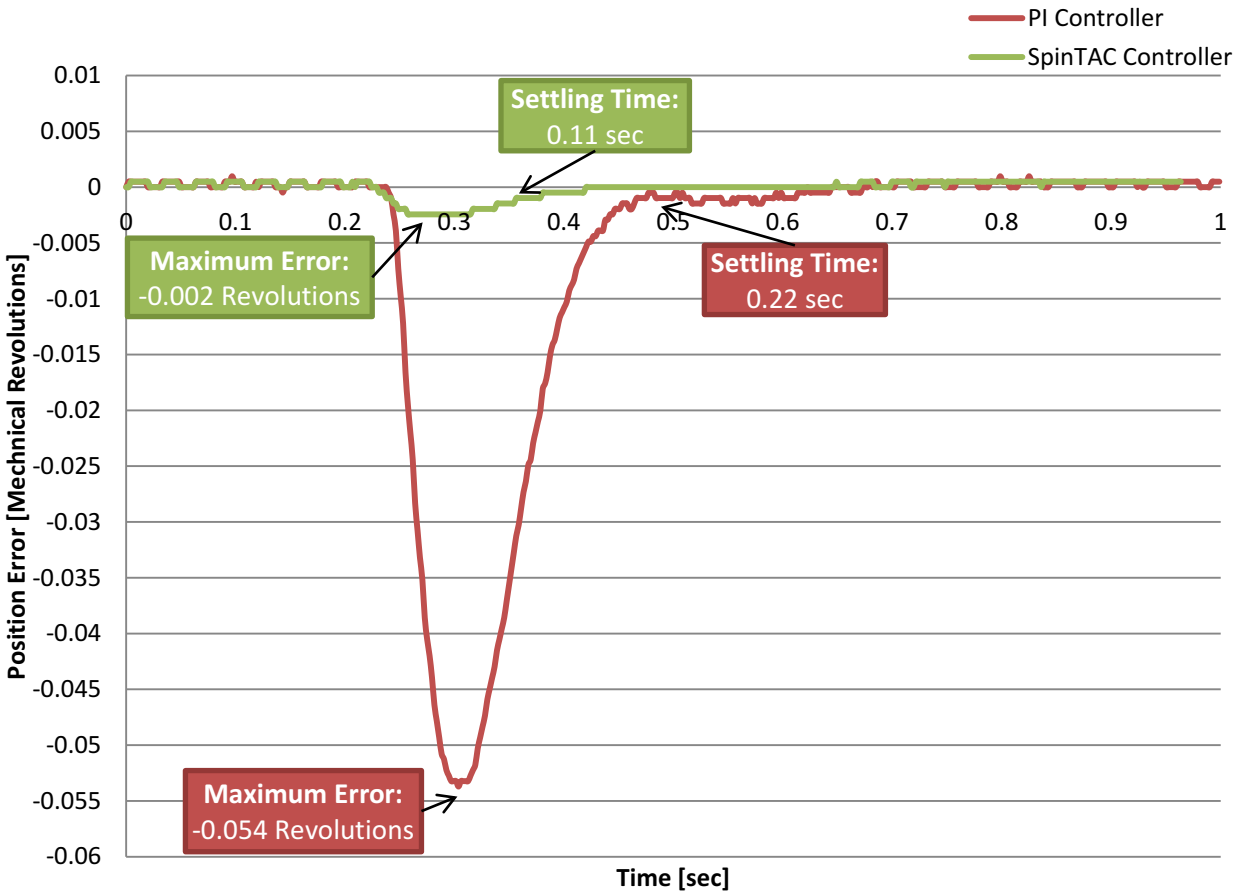


图 **12-20.** 移除转矩干扰时的 **PI** 和 **SpinTAC** 位置控制比较

与传统 PI 控制器相比，SpinTAC 位置控制具有更短的恢复时间和更小的最大误差， 因此可以减少位置振荡，实现更稳定的性能并降低功耗。

12.5.5.3 前馈

SpinTAC 位置控制还具有前馈功能， 因而可实现出色的系统配置（请参见图 12-21）。 前馈用于通知 SpinTAC 位置控制到达位置目标所需的加速或减速的加速度， 因此，与 PI 控制器相比，SpinTAC 位置控制可以更快地响应系统配置变化， 从而减小最大误差和绝对平均误差。

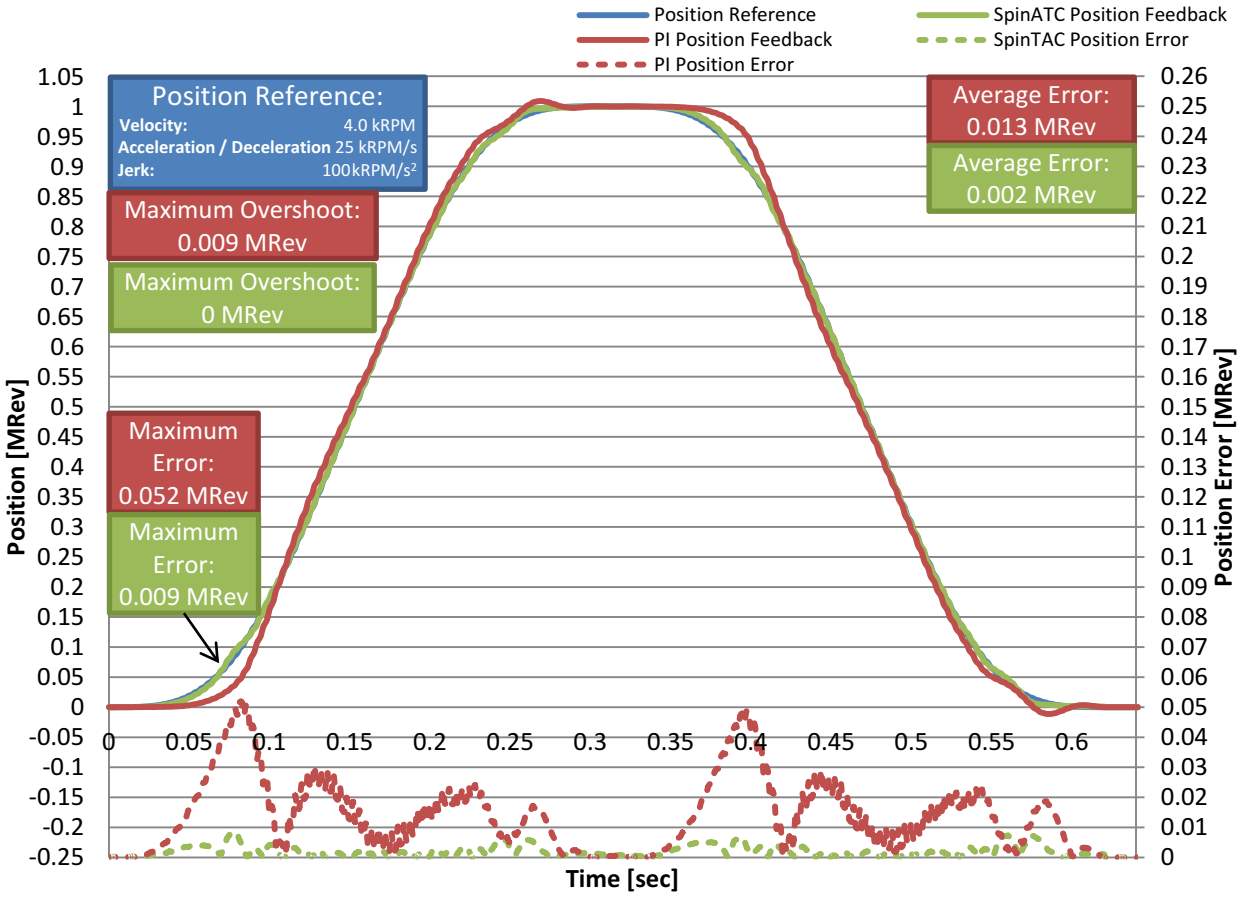


图 **12-21. PI** 和 **SpinTAC** 中前馈对位置系统配置跟踪的影响比较

SpinTAC 位置控制具有较小的最大误差和绝对平均误差， 因此，与传统 PI 控制器相比显著提高了跟踪性能， 从而可以在参考变化时减少不必要的运动和能源浪费。

12.5.5.4 低速运行和平滑启动

高端安保和会议室摄像机等应用以极低速运转（例如，0.1rpm），并且需要准确且平滑的位置控制来转动、倾斜和缩放。 很难在低速时对驱动这些摄像机的电机进行调整，这些电机通常需要使用至少四个调整集来控制位置和速度。

低速运行时很难克服系统惯性，这将导致启动时发生跳动，造成抖动或未聚焦画面。图 12-22 是超低速时微小位置移动的示例。 与 PI 控制器相比，SpinTAC 可以更加准确地跟踪可导致平滑运动的参考位置。SpinTAC 位置控制在低速运行和高速运行时都能够有效克服系统惯性，从而实现非常平滑的低速运动。

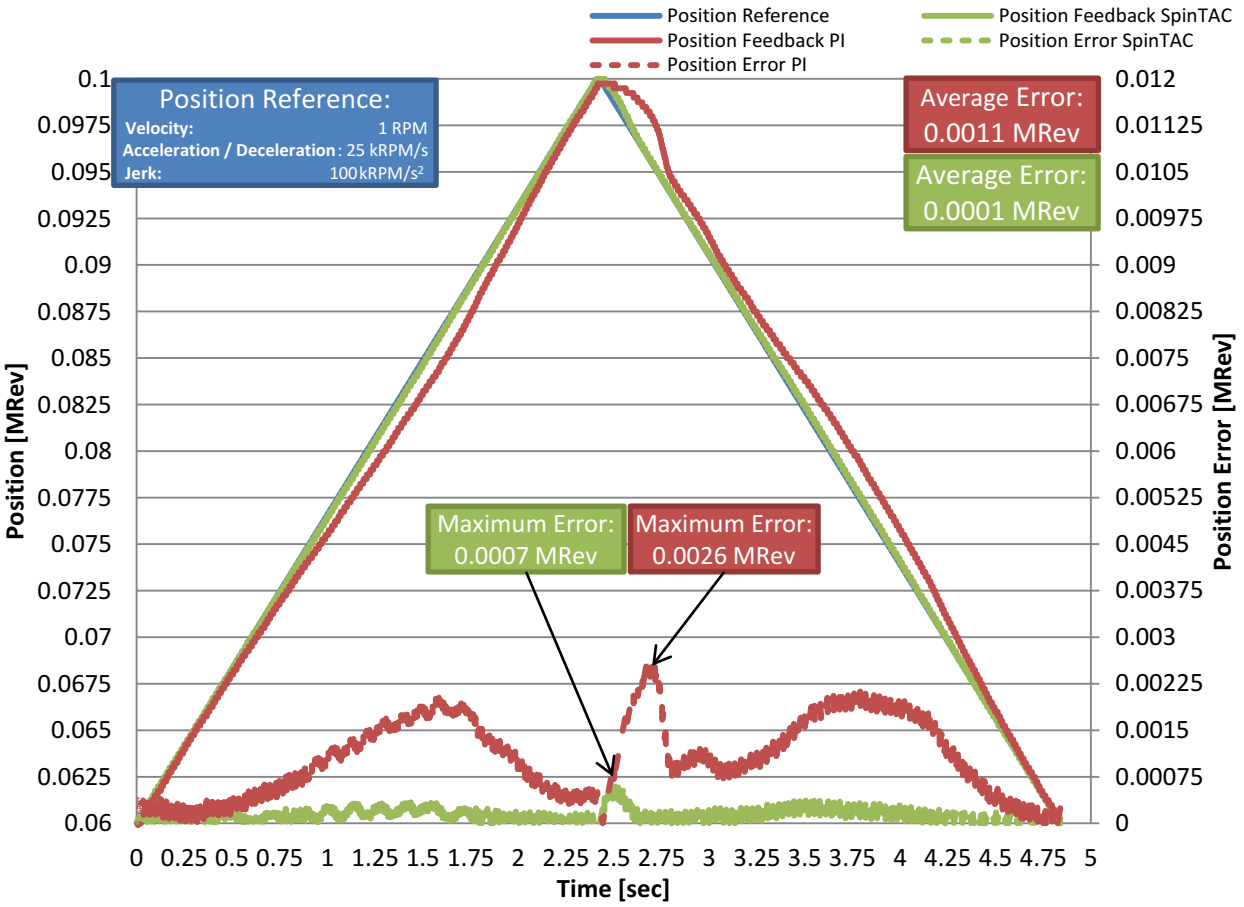


图 **12-22. PI** 和 **SpinTAC** 低速位置系统配置跟踪的比较

12.5.5.5 最小阶跃响应稳定时间

SpinTAC 位置控制具有更短的阶跃响应稳定时间， 使系统能够更积极地响应控制变化。 系统可以更快到达目标位置，减少延迟。 两种控制器均经过零过冲调整，但在发生动态特性变化后，与 PI 控制器相比，SpinTAC 响应更快且仍具有最小的过冲。

图 12-23 比较了 SpinTAC 位置控制和传统 PI 位置控制系统的阶跃响应。 从图中可以看出，传统 PI 位置控制系统的稳定时间要远远大于 SpinTAC 位置控制。 较长的稳定时间意味着应用需要花费更长时间到达目标位置。

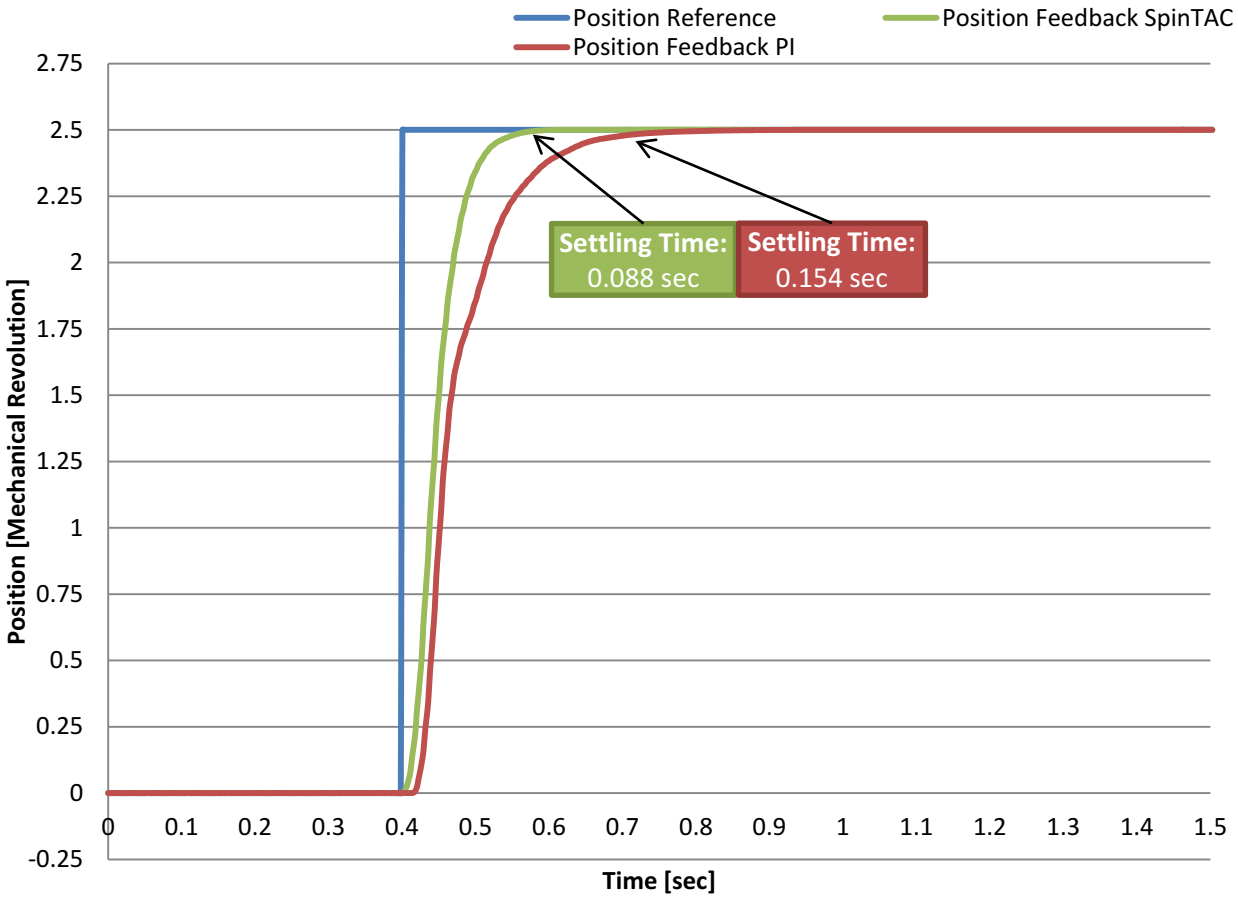


图 **12-23. PI** 和 **SpinTAC** 位置控制的阶跃响应比较

12.5.5.6 结论

InstaSPIN-MOTION 中包括的 SpinTAC 位置控制可替代用于速度和位置控制的传统 PI 控制器。 它可在应用的整个工作范围内获得更佳的性能。 由于使用单个调整参数，因此与 PI 控制器相比，调整操作更为简单。 这样便可在开发时减小调整位置控制器方面的工作量，使您能够重点关注应用的其它部分。 SpinTAC 速度控制可减少过冲和缩短稳定时间， 还具有更佳的系统配置跟踪功能。 所有这些功能结合起来，可以降低应用中不必要运动所造成的能源消耗。

轨迹规划

建立运动系统的第一步是控制电机的速度或位置。 下一步是建立转换不同速度和位置的方法，然后为电机运动排序，从而完成应用任务。 利用 InstaSPIN-MOTION，您能够快速构建复杂的运动序列，根据逻辑进行状态转换。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

13.1 生成 InstaSPIN-MOTION 系统配置

SpinTAC Move 是基于约束的时间最优系统配置生成器。 此系统配置生成器在运行期间计算运动系统配置时不会使用闪存查找表。 因此，内存占用量较少。 用户提供约束条件（速度限制 [仅位置]、加速度/减速度限制和急动限制）后，SpinTAC Move 会计算出当前设定点和目标设定点之间的最优配置。 这样，您就可以更加灵活地设计应用运动系统配置。

除了符合工业标准的梯形和 s 曲线系统配置，SpinTAC Move 还提供了 Linestream 专有 st 曲线。 这种曲线的运动变化比梯形或 s 曲线系统配置更为平滑。 st 曲线的主要特点是连续急动。

图 13-1 对比了 SpinTAC 位置移动中提供的各种不同曲线类型。 S 曲线和 st 曲线之间最明显的区别在于最下方的急动曲线，在其中可以看出，st 曲线会对急动进行连续调整，从而提供比 s 曲线更平滑的运动。 对于速度转换，仅需要考虑图 13-1 中下方的三幅图

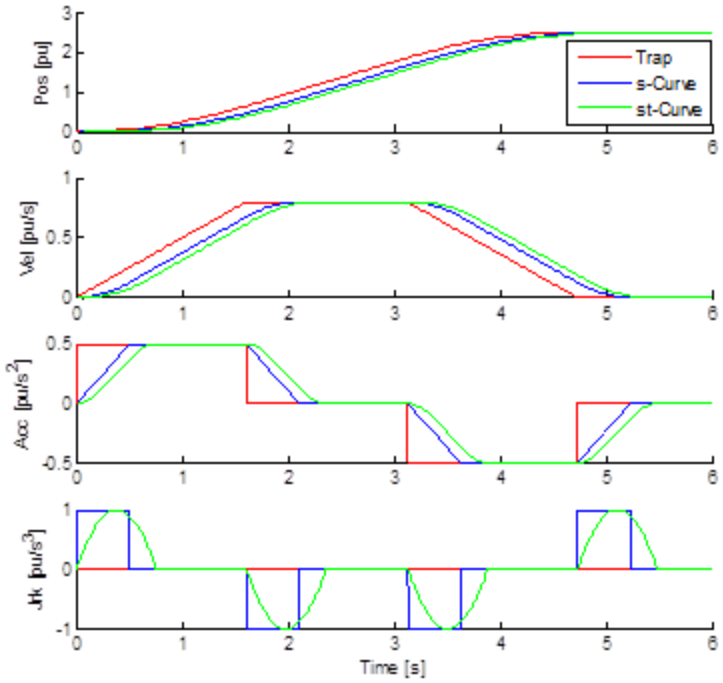


图 13-1. 对比 SpinTAC 位置移动所提供的曲线

SpinTAC Move 根据采样时间生成系统配置。 它会参照速度采样时间来校准运动系统配置，并确保完成系统配置的时间始终是采样时间的整数倍。 这种确定方式可确保满足给定的一系列约束条件，SpinTAC Move 将始终生成完全相同的系统配置。

13.1.1 急动对系统性能的影响

急动代表加速度变化率。 因此，更大的加速度变化率表示将以更快的速率增大加速度。 某些应用中含有的易碎物体对加速度变化的承受能力有限，对此，应着重考虑急动。 而某些应用中切削刀具的加速度迅速变化会导致刀具过早磨损或导致切口不平，对此，急动也是重要考虑因素。 对于需要考虑系统急动的应用而言，使用提供 st 曲线的 SpinTAC Move 很有必要。 急动还会影响电机速度变化时的耗电量。 急动越小，电机速度变化时的耗电量越低。 这是因为更小的急动会降低加速度的增长速率。 对于不需要直接考虑急动的应用，急动也会对系统性能造成影响。

图 13-2 对比了三种轨迹曲线。 这三条曲线的初速度、末速度和加速度均相同。 各曲线的急动已经过修改。 急动度越大，曲线达到目标速度就越快。 这种更快运动的结果是，电机在执行该轨迹曲线时消耗更多电流。 最大电流如图所示。 该测试在系统空载条件下执行。 如果系统中加入负载，则增大急动度对最大电流的影响会更为明显。

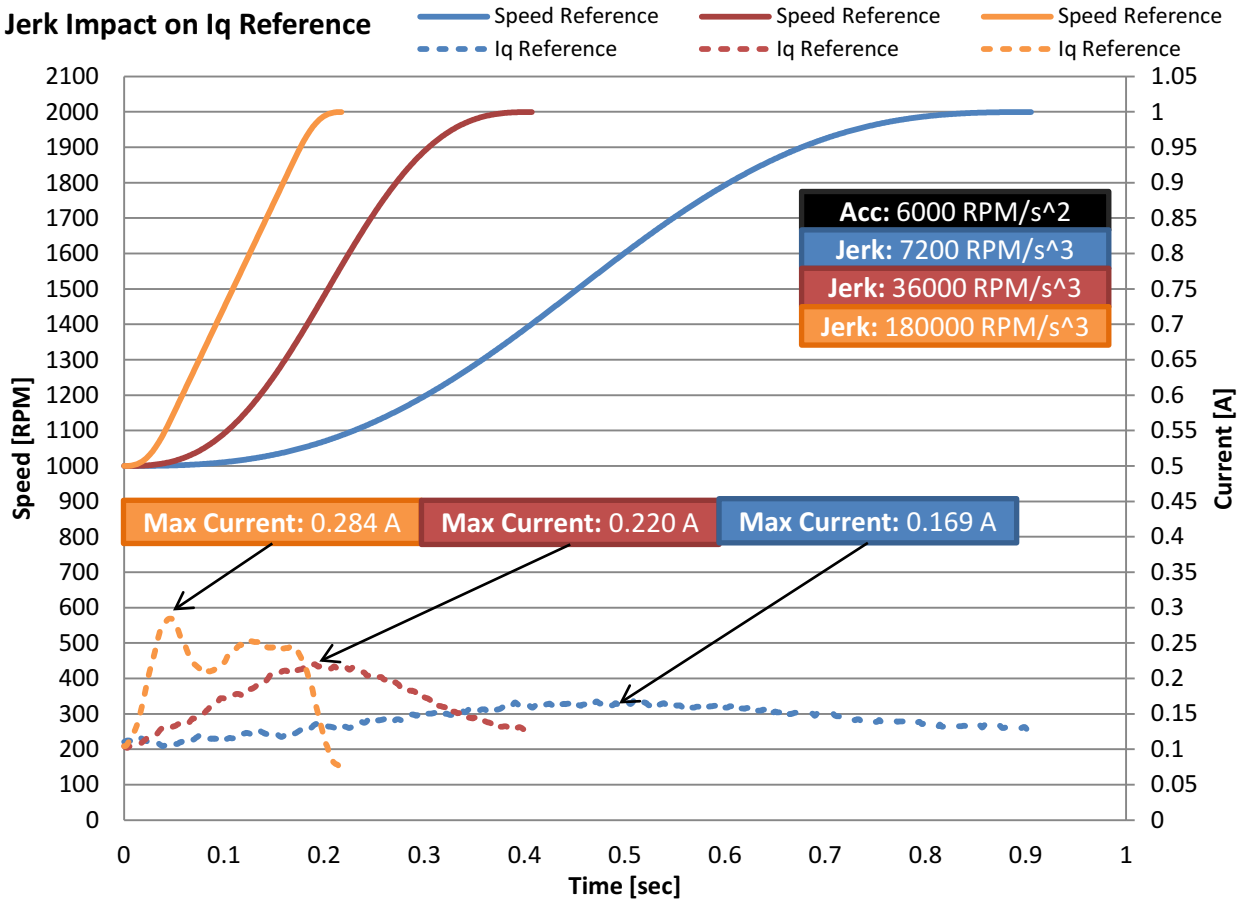


图 **13-2.** 急动对 **Iq** 参考电流的影响

13.2 SpinTAC 速度移动的软件配置

配置 SpinTAC 速度移动需要四个步骤。 实验 6a“使用 SpinTAC Move 进行平滑系统运动”是一个示例项目，其中执行了使用 SpinTAC 速度移动生成轨迹变化所需的步骤。 利用包含在 MotorWare 中的头文件 spintac.h，您可以快速将 SpinTAC 组件包括在项目中。

13.2.1 包括头文件

这应该通过其余的模块头文件包含来完成。 实验 6a 示例项目中，该文件包括在 spintac_velocity.h 头文件中。 针对相应项目，可通过包括 spintac_velocity.h 完成此步骤。

```
#include    "sw/modules/spintac/src/32b/spintac_vel_move.h"
```

13.2.2 声明全局结构

这应该通过主源文件中的全局变量声明来完成。 在实验 8a 项目中，此结构包括在已声明为 spintac_velocity.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj    st_obj;   // The SpinTAC Object
ST_Handle stHandle; // The SpinTAC Handle
```

此示例为不希望使用已在 spintac_velocity.h 头文件中声明的 ST_Obj 结构的情况。

```
ST_VelMove_t      stVelMove;       // The SpinTAC Velocity Move object
ST_VELMOVE_Handle stVelMoveHandle; // The SpinTAC Velocity Move Handle
```

13.2.3 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这样会将所有的默认值加载到 SpinTAC 速度移动中。 该步骤可通过运行已在 spintac_velocity.h 头文件中声明的函数 ST_init 和 ST_setupVelMove 来完成。 如果不希望使用这两个函数，可使用下方的代码示例配置 SpinTAC 速度移动组件。 SpinTAC 速度移动配置是典型配置的代表，适用于大部分电机。

```
// Initialize the SpinTAC Velocity Move Component
stVelMoveHandle = STVELMOVE_init(&stVelMove, sizeof(ST_VelMove_t));

// Instance of SpinTAC Velocity Move
STVELMOVE_setAxis(stVelMoveHandle, ST_AXIS0);
// Sample time [s], (0, 1]
STVELMOVE_setSampleTime_sec(stVelMoveHandle, _IQ24(ST_SAMPLE_TIME));
// System maximum limit for:
//  speed [pu/s] [IQ24(0.001), _IQ24(1)],
//  acceleration [pu/s^2] [IQ24(0.002), _IQ24(120)],
//  jerk references [pu/s^3] [_IQ20(0.0005), IQ20(2000)]
STVELMOVE_setProfileMaximums(stVelMoveHandle, _IQ24(1.0), _IQ24(10.0), _IQ20(62.5));
// Acceleration limit for the profile [pu/s^2] [IQ24(0.001), _IQ24(120)]
STVELMOVE_setAccelerationLimit(stVelMoveHandle, _IQ24(0.4));
// Jerk limit for the profile [pu/s^3] [_IQ20(0.0005), _IQ20(2000)]
STVELMOVE_setJerkLimit(stVelMoveHandle, _IQ20(1.0));
// Set profile curve type { ST_MOVE_CUR_TRAP, ST_MOVE_CUR_SCRV, ST_MOVE_CUR_STCRV }
STVELMOVE_setCurveType(stVelMoveHandle, ST_MOVE_CUR_STCRV);
// ST_VelMove is not in test mode
STVELMOVE_setTest(stVelMoveHandle, FALSE);
// Initially ST_VelMove is not enabled
STVELMOVE_setEnable(stVelMoveHandle, FALSE);
// Initially ST_VelMove is not in reset
STVELMOVE_setReset(stVelMoveHandle, FALSE);
```

13.2.4 调用 SpinTAC 速度移动

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此函数。 抽取率由 ST_ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。 在调用 SpinTAC 速度移动函数之前，必须更新目标速度、加速度限制、急动限制和曲线类型。

```
// If we are not in reset, and the SpeedRef_krpm has been modified
if((STVELMOVE_getReset(stVelMoveHandle) == FALSE)
   && (_IQmpy(gMotorVars.SpeedRef_krpm, _IQ24(ST_SPEED_PU_PER_KRPM))
```

```
       != STVELMOVE_getVelocityEnd(stVelMoveHandle)))
{
   // Get the configuration for SpinTAC Velocity Move
   STVELMOVE_setCurveType(stVelMoveHandle, gMotorVars.SpinTAC.VelMoveCurveType);
   STVELMOVE_setVelocityEnd(stVelMoveHandle,
                          _IQmpy(gMotorVars.SpeedRef_krpm, _IQ24(ST_SPEED_PU_PER_KRPM)));
   STVELMOVE_setAccelerationLimit(stVelMoveHandle,
                        _IQmpy(gMotorVars.MaxAccel_krpmps, _IQ24(ST_SPEED_PU_PER_KRPM)));
   STVELMOVE_setJerkLimit(stVelMoveHandle,
                    _IQ20mpy(gMotorVars.MaxJrk_krpmps2, _IQ20(ST_SPEED_PU_PER_KRPM)));
   // Enable SpinTAC Move
   STVELMOVE_setEnable(stVelMoveHandle, TRUE);
   //If starting from zero speed, enable ForceAngle, otherwise disable ForceAngle
   if(_IQabs(STVELMOVE_getVelocityStart(stVelMoveHandle)) < _IQ24(ST_MIN_ID_SPEED_PU))
   {
     EST_setFlag_enableForceAngle(ctrlObj->estHandle, TRUE);
     gMotorVars.Flag_enableForceAngle = TRUE;
   }
   else
   {
     EST_setFlag_enableForceAngle(ctrlObj->estHandle, FALSE);
     gMotorVars.Flag_enableForceAngle = FALSE;
   }
}

// Run SpinTAC Move
STVELMOVE_run(stVelMoveHandle);
```

13.2.5 SpinTAC 速度移动故障排除

13.2.5.1 ERR_ID

ERR_ID 可向用户提供错误代码。 表 13-1 中列出了 SpinTAC 速度移动中定义的错误及相应解决方案。

表 13-1. SpinTAC 速度移动 ERR_ID 代码

ERR_ID	问题	解决方案
1	采样时间值无效	将 cfg.T_sec 设置为 (0, 0.01] 范围内
2	系统最大速度值无效	将 cfg.VelMax 设置为 (0, 1] 范围内
10	系统最大加速度值无效	将 cfg.AccMax 设置为 [0.001, 120] 范围内
12	系统最大急动值无效	将 cfg.JrkMax 设置为 [0.0005, 1000] 范围内
32	轴 ID 无效	将 cfg.Axis 设置为 {ST_AXIS0, ST_AXIS1} 范围内
1002	加速度限制值无效	将 AccLim 设置为 [0.002, cfg.AccMax] 范围内
1004	急动限制值无效	将 JrkLim 设置为 [0.001, cfg.JrkMax] 范围内
1005	曲线类型无效	将 cfg.CUR_MOD 设置为 {ST_PRO_TRAP, ST_PRO_SCRV, ST_PRO_STCRV} 范围内
1006	速度起始值无效	将 cfg.VelStart 设置为 [-cfg.VelMax, cfg.VelMax] 范围内
1007	速度结束值无效	将 VelEnd 设置为 [-cfg.VelMax, cfg.VelMax] 范围内
4001	SpinTAC 许可证无效	使用具有有效许可证的芯片
4003	ROM 版本无效	使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。

13.3 SpinTAC 位置移动的软件配置

配置 SpinTAC 位置移动需要四个步骤。 实验 13b“使用 SpinTAC Move 进行位置转换”是一个示例项目，其中执行了使用 SpinTAC 位置移动生成轨迹变化所需的步骤。 利用包含在 MotorWare 中的头文件 spintac_position.h，您可以快速将 SpinTAC 组件包括在项目中。

13.3.1 包括头文件

这应该通过其余的模块头文件包含来完成。 实验 6a 示例项目中，该文件包括在 spintac_position.h 头文件中。 针对相应项目，可通过包括 spintac_position.h 完成此步骤。

```
#include "sw/modules/spintac/src/32b/spintac_pos_move.h"
```

13.3.2 声明全局结构

这应该通过主源文件中的全局变量声明来完成。 在实验 13b 项目中，此结构包括在已声明为 spintac_position.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj       st_obj;    // The SpinTAC Object
ST_Handle stHandle;  // The SpinTAC Handle
```

此示例为不希望使用已在 spintac_position.h 头文件中声明的 ST_Obj 结构的情况。

```
ST_PosMove_t      stPosMove;       // The SpinTAC Position Move object
ST_POSMOVE_Handle stPosMoveHandle; // The SpinTAC Position Move Handle
```

13.3.3 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这样会将所有的默认值加载到 SpinTAC 位置移动中。 该步骤可通过运行已在 spintac_position.h 头文件中声明的函数 ST_init 和 ST_setupPosMove 来完成。 如果不希望使用这两个函数，可使用下方的代码示例配置 SpinTAC 位置移动组件。 SpinTAC 位置移动配置是典型配置的代表，适用于大部分电机。

```
// Initialize the SpinTAC Speed Controller Component
stPosMoveHandle = STPOSMOVE_init(&stPosMove, sizeof(stPosMove));

// Instance of SpinTAC Move
STPOSMOVE_setAxis(stPosMoveHandle, ST_AXIS0);
// Sample time [s], (0, 1]
STPOSMOVE_setSampleTime_sec(stPosMoveHandle, _IQ24(ST_SAMPLE_TIME));
// Set the type of profile to generate {ST_POS_MOVE_VEL_TYPE , ST_POS_MOVE_POS_TYPE}
STPOSMOVE_setProfileType(stPosMoveHandle, ST_POS_MOVE_POS_TYPE);
// Set the maximum value for mechanical revolutions before rollover [MRev]
STPOSMOVE_setMRevMaximum_mrev(stPosMoveHandle, _IQ24(10.0));
// Set the unit conversion values, this will convert between Mrev and pu
STPOSMOVE_setUnitConversion(stPosMoveHandle, USER_IQ_FULL_SCALE_FREQ_Hz,
                                                    USER_MOTOR_NUM_POLE_PAIRS);
// System maximum limit for:
//  speed [pu/s] [IQ24(0.002), _IQ24(1)]
//  acceleration [pu/s^2] [IQ24(0.001), _IQ24(120)]
//  deceleration [pu/s^2] [IQ24(0.001), _IQ24(120)]
```

```
//  jerk references [pu/s^3] [_IQ20(0.0005), _IQ20(2000)]
STPOSMOVE_setProfileMaximums(obj->posMoveHandle,
                             _IQ24(USER_MOTOR_MAX_SPEED_KRPM * ST_SPEED_PU_PER_KRPM),
                                                  _IQ24(10), _IQ24(10), _IQ20(62.5));
// Velocity limit for the profile [pu/s] [IQ24(0.002), _IQ24(1)]
STPOSMOVE_setVelocityLimit(obj->posMoveHandle,
                           _IQ24(USER_MOTOR_MAX_SPEED_KRPM * ST_SPEED_PU_PER_KRPM));
// Acceleration limit for the profile [pu/s^2] [IQ24(0.001), _IQ24(120)]
STPOSMOVE_setAccelerationLimit(stPosMoveHandle, _IQ24(0.4));
// Deceleration limit for the profile [pu/s^2] [IQ24(0.001), _IQ24(120)]
STPOSMOVE_setDecelerationLimit(obj->posMoveHandle, _IQ24(0.4));
// Jerk limit for the profile [pu/s^3] [_IQ20(0.0005), _IQ20(2000)]
STPOSMOVE_setJerkLimit(stPosMoveHandle, _IQ20(1.0));
// Set profile curve type { ST_MOVE_CUR_TRAP, ST_MOVE_CUR_SCRV, ST_MOVE_CUR_STCRV }
STPOSMOVE_setCurveType(stPosMoveHandle, ST_MOVE_CUR_STCRV);
// ST_PosMove is not in test mode
STPOSMOVE_setTest(stPosMoveHandle, false);
// Initially ST_PosMove is not enabled
STPOSMOVE_setEnable(stPosMoveHandle, false);
// Initially ST_PosMove is not in reset
STPOSMOVE_setReset(stPosMoveHandle, false);
```

13.3.4 调用 SpinTAC 位置移动

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此函数。 抽取率由 ST_ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。 在调用 SpinTAC 位置移动函数之前，必须更新目标速度、加速度限制、急动限制和曲线类型。

```
// If we are not running a profile, and the PosStep_MRev has been modified
if((STPOSMOVE_getStatus(stObj->posMoveHandle) == ST_MOVE_IDLE)
&& (gMotorVars.PosStepInt_MRev != 0 || gMotorVars.PosStepFrac_MRev != 0)) {
    // Get the configuration for SpinTAC Position Move
   STPOSMOVE_setCurveType(stObj->posMoveHandle, gMotorVars.SpinTAC.PosMoveCurveType);
   STPOSMOVE_setPositionStep_mrev(stObj->posMoveHandle, gMotorVars.PosStepInt_MRev,
                                                        gMotorVars.PosStepFrac_MRev);
   STPOSMOVE_setVelocityLimit(stObj->posMoveHandle,
                        _IQmpy(gMotorVars.MaxVel_krpm, _IQ24(ST_SPEED_PU_PER_KRPM)));
   STPOSMOVE_setAccelerationLimit(stObj->posMoveHandle,
                      _IQmpy(gMotorVars.MaxAccel_krpmps, _IQ24(ST_SPEED_PU_PER_KRPM)));
   STPOSMOVE_setDecelerationLimit(stObj->posMoveHandle,
                      _IQmpy(gMotorVars.MaxDecel_krpmps, _IQ24(ST_SPEED_PU_PER_KRPM)));
   STPOSMOVE_setJerkLimit(stObj->posMoveHandle,
                    _IQ20mpy(gMotorVars.MaxJrk_krpmps2, _IQ20(ST_SPEED_PU_PER_KRPM)));
   // Enable the SpinTAC Position Profile Generator
   STPOSMOVE_setEnable(stObj->posMoveHandle, true);
   // clear the position step command
   gMotorVars.PosStepInt_MRev = 0;
   gMotorVars.PosStepFrac_MRev = 0;

}

STPOSMOVE_run(stObj->posMoveHandle);
```

13.3.5 SpinTAC 位置移动故障排除

13.3.5.1 位置系统配置限制

某些限制组合无法产生有效系统配置。 这种情况只会在为 SpinTAC 位置移动提供的限制非常小时发生。

- 如果速度限制设置为低于 0.001 pu/s，则无法保证系统配置有效。

  此时，电机可能会由于达到最小速度限制或存在错误条件而无法运动。 还有可能因发生计算溢出而导致电机向预定方向的反方向运动。 在某些情况下，将速度限制设置为低于 0.001 pu/s 时生成的系统配置可能有效，而且速度限制不小于 0.001 pu/s 的系统配置都将有效。

13.3.5.2 ERR_ID

ERR_ID 可向用户提供错误代码。 表 13-2 中列出了 SpinTAC 位置移动中定义的错误及相应解决方法。

表 13-2. SpinTAC 位置移动 ERR_ID 代码

ERR_ID	问题	解决方案
1	采样时间值无效	将 cfg.T_sec 设置为 (0, 0.01) 范围内
2	系统最大速度值无效	将 cfg.VelMax 设置为 [0.002, 1] 范围内
10	系统最大加速度值无效	将 cfg.AccMax 设置为 [0.002, 120] 范围内
11	系统最大减速度值无效	将 cfg.DecMax 设置为 [0.002, 120] 范围内，并将 cfg.DecMax/cfg.AccMax 设置为 [1, 10] 范围内
12	系统最大急动值无效	将 cfg.JrkMax 设置为 [0.001, 2000] 范围内
13	位置翻转界限值无效	将 cfg.ROMax_mrev 设置为 [2, 100] 范围内
14	机械旋转 [MRev] 至 [pu] 的比率值无效	将 cfg.mrev_TO_pu 设置为 [0.008, 1] 范围内
32	轴 ID 无效	将 cfg.Axis 设置为 {ST_AXIS0, ST_AXIS1} 范围内
1001	速度限制值无效	将 VelLim 设置为 (0, cfg.VelMax] 范围内
1002	加速度限制值无效	将 AccLim 设置为 [0.001, cfg.AccMax] 范围内
1003	减速度限制值无效	将 DecLim 设置为 [0.001, cfg.DecMax] 范围内，并将 DecLim/AccLim 设置为 [1, 10] 范围内
1004	急动限制值无效	将 JrkLim 设置为 [0.0005, cfg.JrkMax] 范围内
1005	曲线类型无效	将 cfg.CurveType 设置为 {ST_MOVE_CUR_TRAP, ST_MOVE_CUR_SCRV, ST_MOVE_CUR_STCRV} 范围内
1006	速度起始值无效	将 cfg.VelStart 设置为 [-cfg.VelMax, cfg.VelMax] 范围内
1007	速度结束值无效	将 VelEnd 设置为 [-cfg.VelMax, cfg.VelMax] 范围内
1008	位置起始值无效	将 cfg.PosStart_mrev 设置为 [-cfg.ROMax, cfg.ROMax] 范围内
1009	位置阶跃值无效	将 PosStepInt_mrev 设置为 [-2147483647, 2147483647] 范围内，并将 PosStepFrac_mrev 设置为 (-1, 1) 范围内
1101	计算溢出，VelLim 超出范围	定点计算溢出时会发生此错误。 典型情况为：VelLim 或 PosStep 过小。 解决方法是将值增大。
1102	计算溢出，AccLim 超出范围	
1103	计算溢出，DecLim 超出范围	
1104	计算溢出，JrkLim 超出范围	
1105	系统配置模式值无效	将 cfg.ProfileType 设置为 {ST_POS_MOVE_VEL_TYPE, ST_POS_MOVE_POS_TYPE} 范围内
2001	从 ST_POS_MOVE_VEL_TYPE 至 ST_POS_MOVE_POS_TYPE 的模式切换无效	设置系统配置达到零速，然后通过设置 cfg.ProfileType = ST_POS_MOVE_POS_TYPE 切换至位置控制系统配置
4001	SpinTAC 许可证无效	使用具有 SpinTAC 有效许可证的芯片
4003	ROM 版本无效	使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

13.4 InstaSPIN-MOTION 序列规划

SpinTAC 速度规划是一个运动序列规划器。 您可以利用该规划器构建应用的运动序列，而无须建立有限状态机。 SpinTAC 速度规划中包含多种高级特性，可实现复杂的有限状态机。 它可以进行条件转换，使电机转换为多种可能状态之一。 该组件还具备各种变量，可用于与外部组件（例如传感器或传动器）进行交互或用作内部计数器（以便跟踪特定状态下的事件数量）。 SpinTAC 速度规划可在运行时进行配置，而且可以在多种状态机之间进行切换。

SpinTAC Plan 既适用于位置解决方案，也适用于速度解决方案。 SpinTAC Plan 的特性和功能是相同的。 唯一的区别在于，在配置过程中，SpinTAC 位置规划需要配置一些附加字段。

13.4.1 SpinTAC 速度规划元素

SpinTAC 速度规划具备一些可搭配使用，从而生成运动序列的元素。 这些不同的元素为：状态、转换、条件、变量和操作。 每种元素都配置为通过单独的 API 调用。 有关 API 的详细信息，请参见3.5 节

13.4.1.1 状态

状态描述了系统配置的稳定运行情况。 用户可指定末速度（SpinTAC 速度规划）或位置阶跃（SpinTAC 位置规划），以及 SpinTAC 速度规划在转换为另一状态前保持原状态的最短时间。 在洗衣机示例中，状态定义为：空闲、注水、顺时针搅动、逆时针搅动、排水和甩干。

13.4.1.2 转换

转换定义了可行的状态间变化。 它们可在各状态间建立连接。 如果满足特定条件，转换可令状态发生变化。 用户可指定初始状态和目标状态、系统配置限制，以及在转换前要评估的条件。

13.4.1.3 条件

条件可用于在转换或操作中进行逻辑检查。 一个转换或动作最多可以具备两个条件。 电机必须在满足条件后才能从一个状态转换为下一个状态或者执行相应操作。 为确定是否满足条件，可使用一个变量与特定值或值范围进行比较。 这样就可以根据条件返回 true 或 false 值。

13.4.1.4 变量

变量允许 SpinTAC 速度规划与项目的其余部分进行交互。 SpinTAC 速度规划中有三种变量：输入、输出和输入-输出。 输入变量用于接收来自 SpinTAC 速度规划外部的值，并对条件进行评估。 输出变量用于与系统的其余部分进行交互。 输出变量可通过 SpinTAC 速度规划进行修改，但 SpinTAC 速度规划不能使用输出变量来检查条件。 注意：用户必须写入用于执行输出变量相关事件（例如打开阀门）的代码。 输入-输出变量通常充当计数器或定时器，由操作或条件使用。

13.4.1.5 操作

操作会改变变量值。 操作可以将变量设置为等于某值，或者向变量添加某值。 操作可能在特定状态下发生，也可能在 SpinTAC 速度规划进入或退出某状态时发生。 各个操作具有相关条件。 这样，只有在满足条件时才会执行操作。 如果某个操作配置为 ENTER（进入），SpinTAC Plan 将在进入状态时开始评估该操作的条件。 只有在满足条件后才会执行一次该操作。 类似地，如果某个动作配置为 EXIT（退出），SpinTAC Plan 将在退出状态时开始评估该操作的条件。 满足相应条件后会执行一次操作。

13.4.2 SpinTAC 速度规划元素限制

SpinTAC 速度规划可配置任意数量的元素。 唯一的限制条件在于要为 SpinTAC 速度规划的配置分配的内存量。 确定内存分配量时，一方面应考虑效率问题，使得 SpinTAC 速度规划能够高效地使用系统内存，另一方面也要考虑灵活性，确保能够定制元素配置。 SpinTAC 速度规划每种元素的内存占用量都不相同。 相关信息汇总在表 13-3 中。

表 13-3. SpinTAC 速度规划元素的内存要求

规划元素	SpinTAC 速度规划（双字）	SpinTAC 位置规划（双字）
操作	5	5
条件	3	3
变量	2	2
转换	5	7
状态	4	7

要实现这种额外的灵活性，则需要声明一个需要将其地址送入 SpinTAC 速度规划的配置数组。 该配置数组的大小需要根据规划中的元素数量来确定。表 13-3 给出了每种元素的内存要求。 最好对要使用的规划元素数量进行声明。 这样能够简化配置数组所需内存量的计算过程。

13.4.2.1 确定 SpinTAC 速度规划配置数组大小的示例

我们的示例状态机具备以下元素：

- 4 个操作
- 3 个条件
- 3 个变量
- 6 个转换
- 3 种状态

共需要 83 位双字配置空间。 该值根据上述元素数量和表 13-3 中包含的内存用量计算得出。

```
4 Actions     * 5 Double Words = 20 Double Words
3 Conditions  * 3 Double Words = 9 Double Words
3 Variables   * 2 Double Words = 6 Double Words
6 Transitions * 5 Double Words = 30 Double Words
3 States      * 4 Double Words = 12 Double Words

Adding this together
20 + 9 + 6 + 30 + 12 = 83 Double Words

The declaration of the SpinTAC Velocity Plan configuration array should be as follows
uint32_t stVelPlanCfgArray[83];
```

实验项目 6c 中介绍了另一示例“运动序列实例：洗衣机”。 该示例详细说明了如何使用枚举来轻松地确定 SpinTAC 速度规划配置数组的大小。

要确定 SpinTAC 位置规划配置数组的大小，需要遵循上文所述步骤，但要使用 SpinTAC 位置规划的内存用量值。

13.4.3 SpinTAC 速度规划示例：洗衣机搅动

洗衣机搅动阶段是介绍 SpinTAC 速度规划基本元素的另一示例。 搅动周期为基本运动系统配置。 在本示例中，SpinTAC 速度规划不会与任何外部传感器或阀门进行交互，也不会进行条件转换。 该运动序列可在 SpinTAC 速度规划中轻松实现。图 13-3 展示了洗衣机搅动过程的状态转换图。

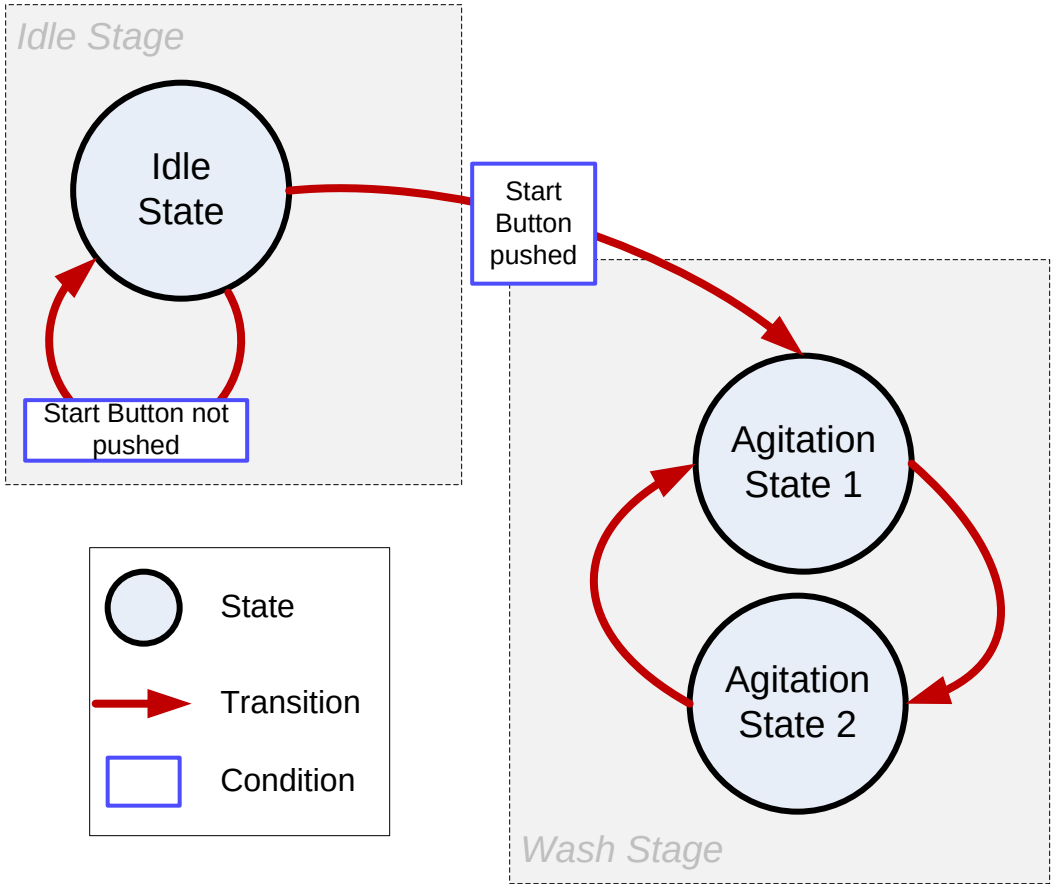


图 13-3. 洗衣机搅动示例的状态转换图

洗衣机搅动过程分为两个阶段：空闲和洗涤。 在按下启动按钮前，洗衣机会一直保持空闲状态。 按下启动按钮后，洗衣机就会立即进入洗涤阶段，开始以正向和反向速度交替搅动，直至洗衣机搅动状态机收到停止命令。

图 13-6 描述了整个洗衣机运动序列中的电机速度，但它还可以用于描述洗衣机搅动运动序列中的电机速度。 请参考图 13-6 的洗涤阶段部分，其中包含了正反向交替的电机速度。 这代表洗衣机的搅动阶段。

13.4.4 SpinTAC 速度规划示例：车库门

车库门系统是介绍 SpinTAC Plan 基本元素的一个示例。 车库门属于基本运动系统配置，其中包括条件转换、变量和操作。 它引入了 SpinTAC 速度规划示例中的所有不同组件。 该运动序列可在 SpinTAC 速度规划中轻松实现。 图 13-4 展示了车库门示例的状态转换图。

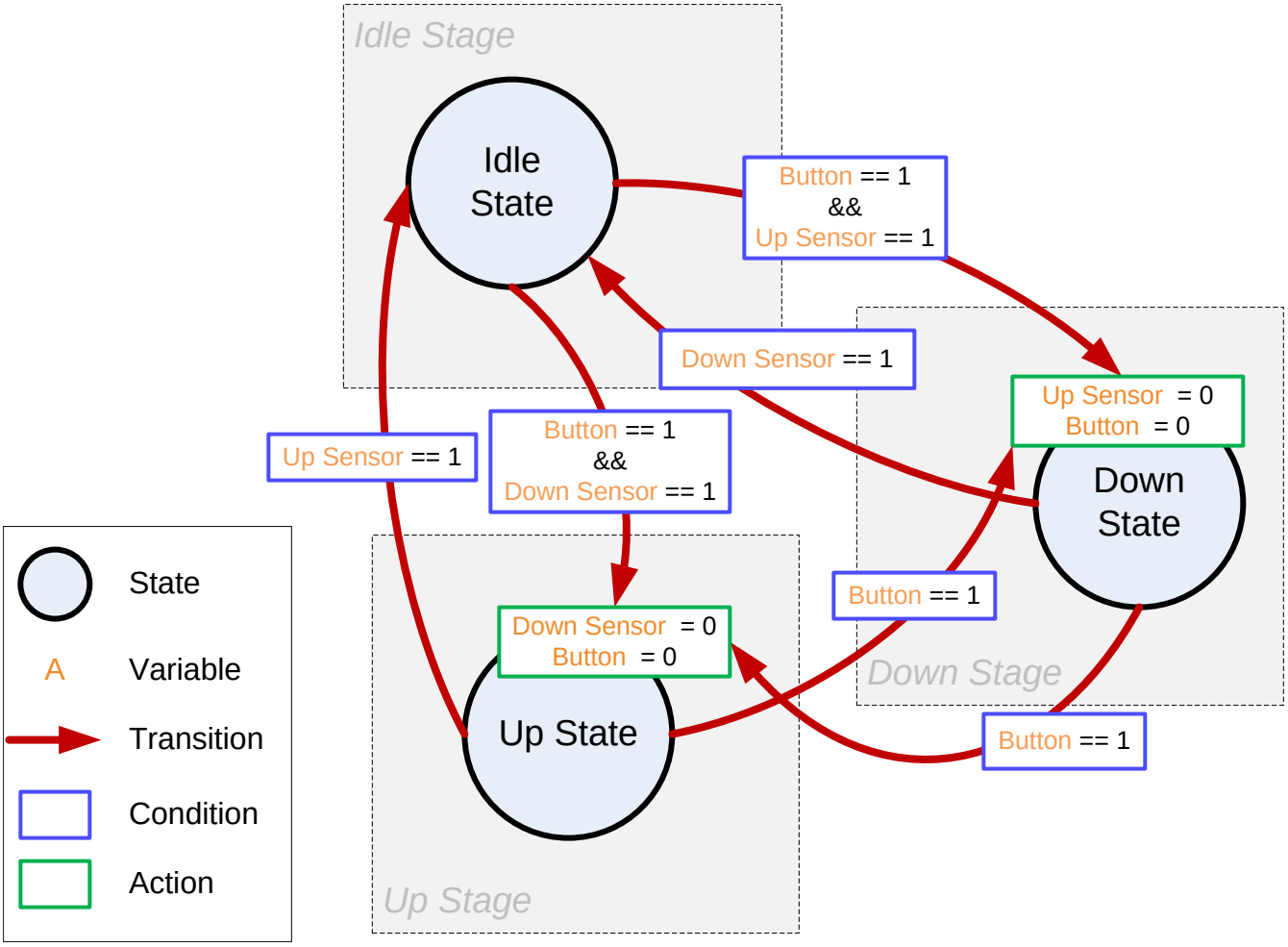


图 **13-4.** 车库门示例的状态转换图

车库门共有三个阶段：空闲、上升和下降。 在按下按钮前，车库门会一直保持空闲状态。 按下按钮后，车库门就会立即根据当前位置转换至上升或下降状态。 如果在车库门转换为上升或下降时按下按钮，它就会改变运行方向。

13.4.5 SpinTAC 速度规划示例：洗衣机

洗衣机是使用 SpinTAC 速度规划的绝佳示例。 洗衣机拥有复杂的运动序列。 在本示例中，SpinTAC 速度规划会与传感器和阀门进行交互，还会进行有条件的状态转换。 整个运动序列可在 SpinTAC 速度规划中轻松实现。图 13-5 展示了洗衣机的状态转换图。

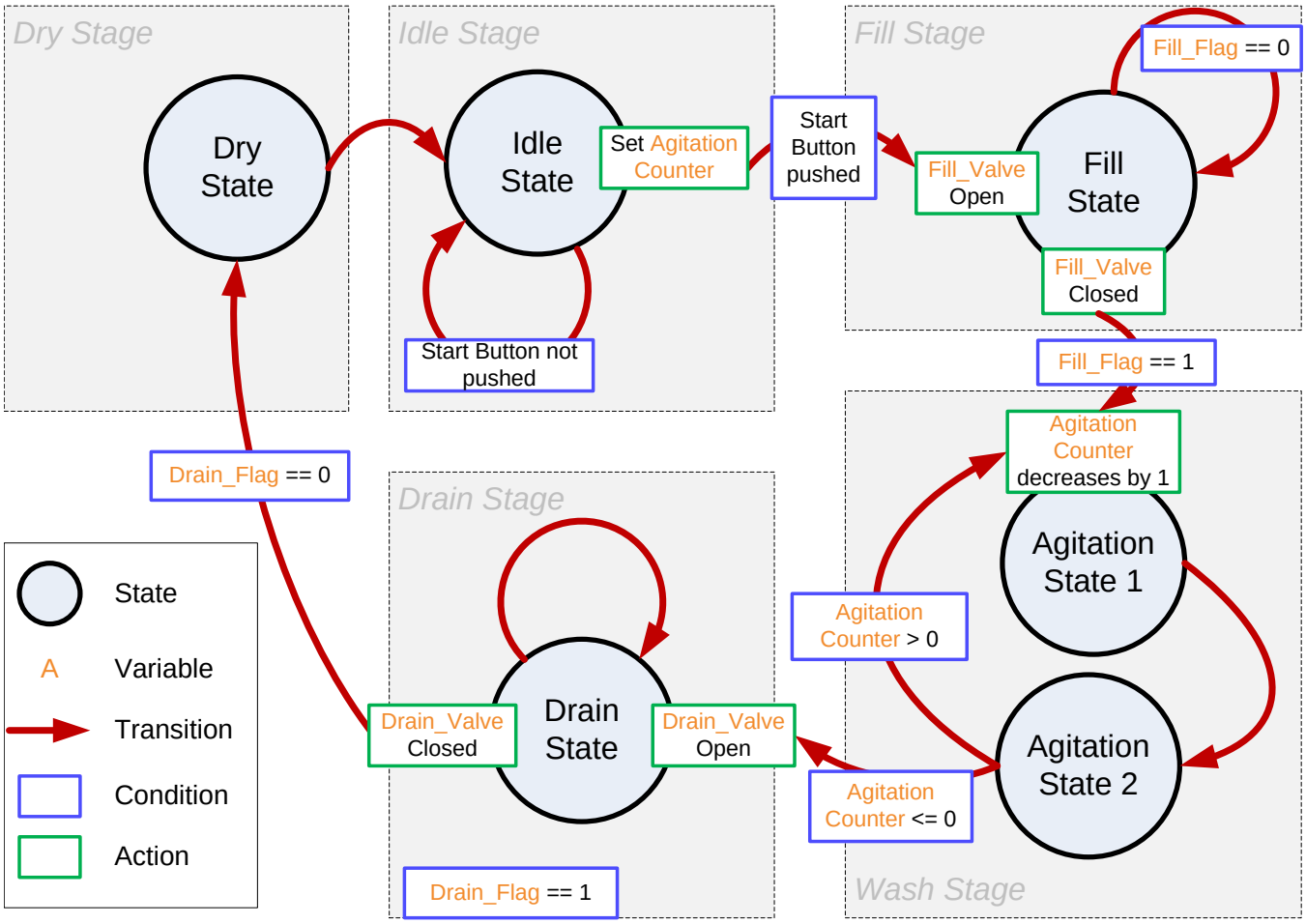


图 **13-5.** 洗衣机示例的状态转换图

洗衣机分为五个阶段：空闲、注水、洗涤、排水和甩干。

在按下启动按钮前，洗衣机会一直保持空闲状态。 按下启动按钮后，洗衣机会立即进入注水阶段，此时搅动计数器会被设为配置值，表示要执行的搅动周期数。

进入注水阶段后，注水阀会打开。 水位传感器会指示洗衣机桶内水已注满。 注水后，注水阀关闭，转而进入洗涤阶段。

在洗涤阶段中，电机会正反方向交替搅动，直至搅动计数器归零。然后洗衣机进入排水阶段。

进入此阶段后，排水阀打开。 排水传感器会指示水已排空。 排水完毕之后，排水阀关闭，洗衣机进入甩干阶段。

在甩干阶段，电机会在配置的时间内以一定速度旋转。 到达配置时间后，洗衣机进入空闲阶段。 此时整个运转过程结束。

图 13-6 描述了洗衣机运动序列中的电机速度曲线。 电机以 0RPM 等待注水阶段完成。 此时，电机以 250RPM 和 -250RPM 的速度来回搅动 20 个周期。 20 个搅动周期后，电机速度会回到 0RPM，直至洗衣机排水结束。 在甩干阶段，电机会以高达 2000RPM 的速度旋转，甩干衣物。 甩干阶段结束后，电机速度会回到 0RPM 并进入空闲状态。

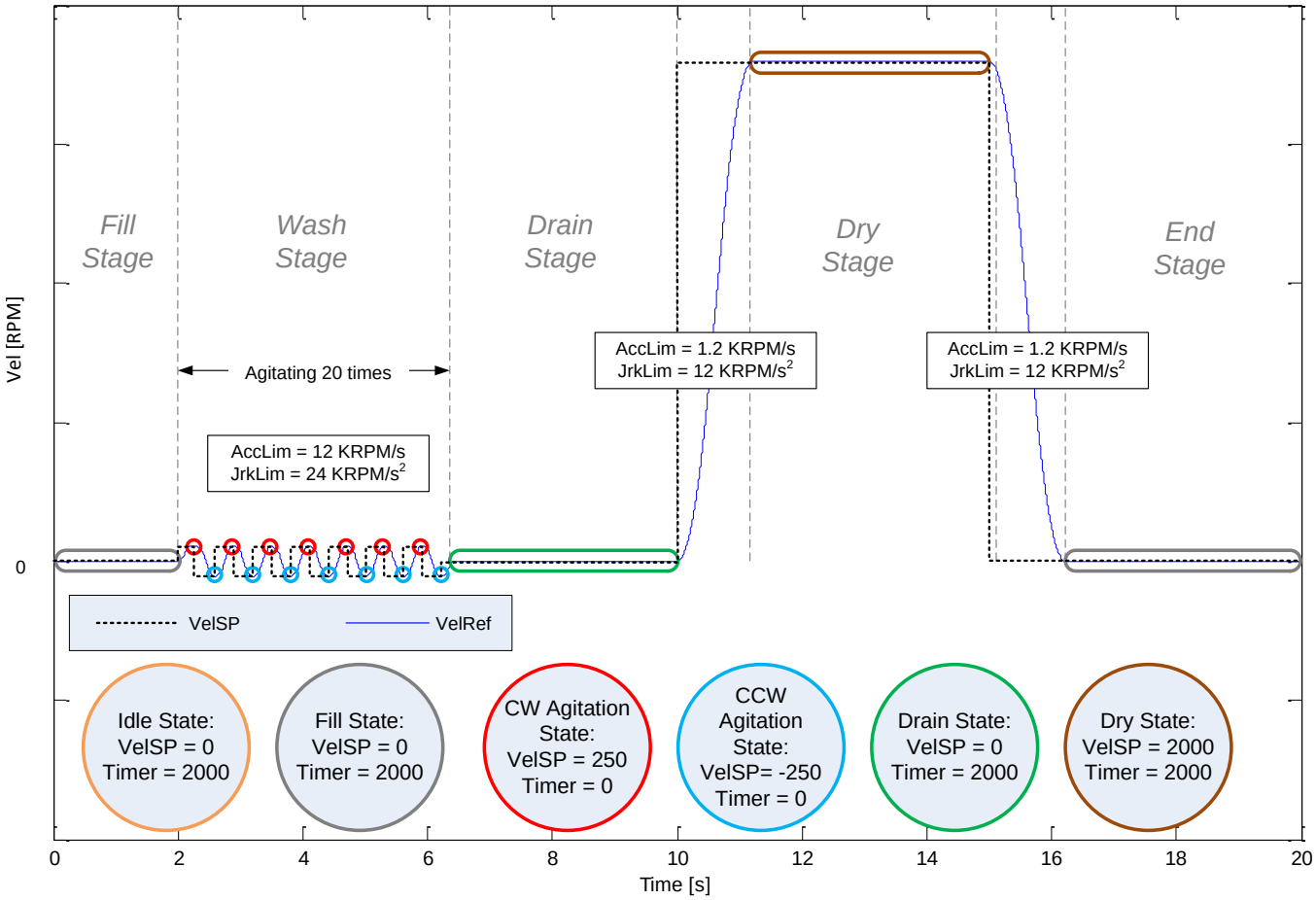


图 **13-6.** 洗衣机示例中的速度曲线

13.4.6 SpinTAC 位置规划示例：自动售货机

自动售货机是 SpinTAC 位置规划的一个使用示例。 在该示例中，自动售货机的售货过程是一个循环过程，一次仅出售一件商品。 图 13-7 中展示了该示例的状态转换图。

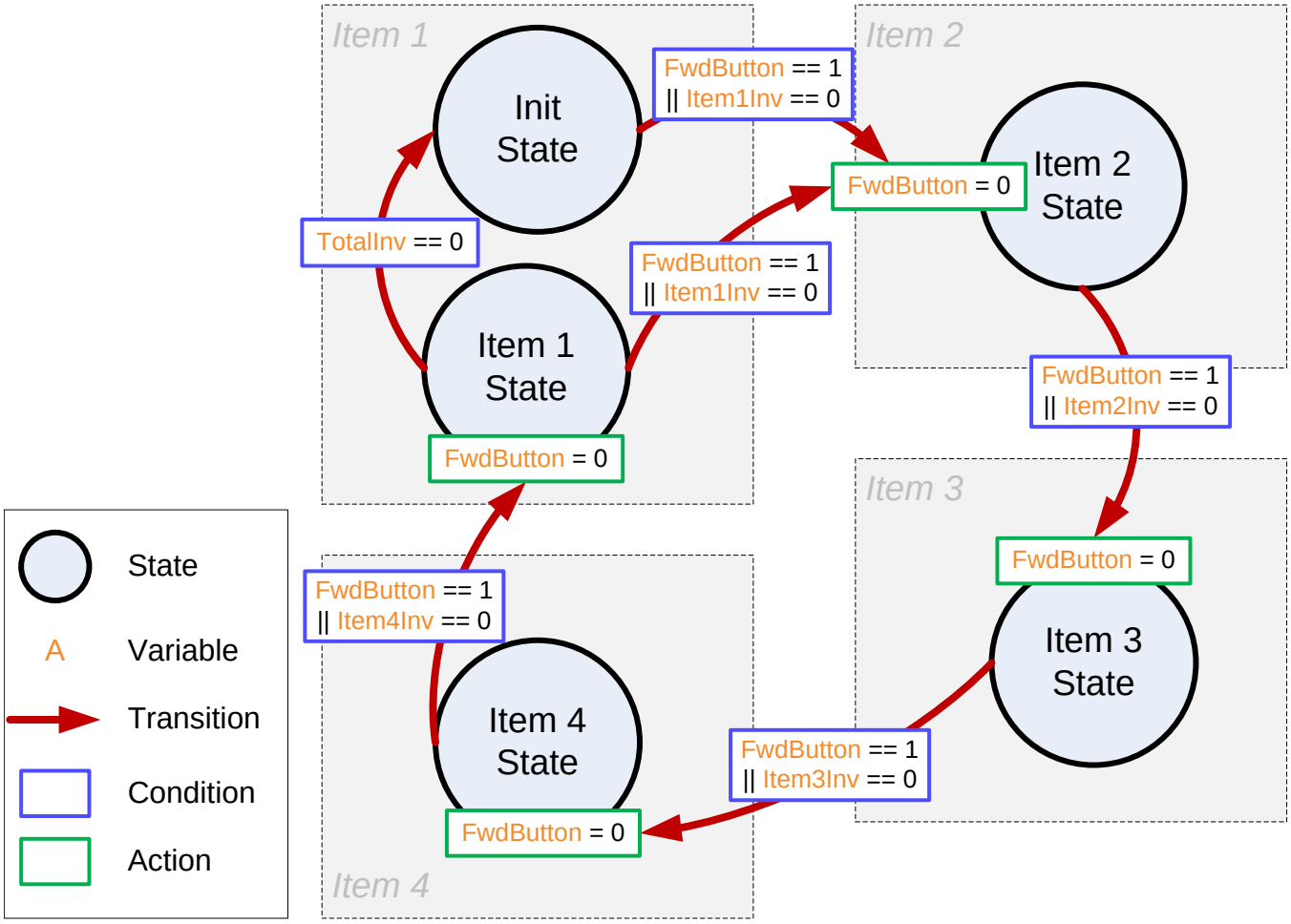


图 **13-7.** 自动售货机示例的状态转换图

在该示例中，自动售货机将展示一件商品，直至用户按下前进按钮 (FwdButton)。 检测到该按钮已按下后，自动售货机将立即向前推送出一件商品，然后展示下一件可售卖的商品。

用户将商品从自动售货机中取出时，应用程序会更新商品存货记录。 当存货量减至零时，自动售货机会跳过该商品的售卖状态。 如果所有商品的存货量均减至零，自动售货机将返回初始状态，而 SpinTAC 位置规划将停止运行。 这表明自动售货机需要补货。

实验项目 13d 中执行了 SpinTAC 位置规划的该示例“运动序列实例：自动售货机”。

13.5 SpinTAC 速度规划的软件配置

配置 SpinTAC 速度规划需要七个步骤。 实验 6c“运动序列实例：洗衣机”是一个示例项目，其中执行了使用 SpinTAC 速度规划所需的步骤。 利用包含在 MotorWare 中的头文件 spintac_velocity.h，您可以快速将 SpinTAC 组件包括在项目中。

13.5.1 包括头文件

这应该通过其余的模块头文件包含来完成。 在实验 6c 示例项目中，该文件包括在 spintac_velocity.h 头文件中。 针对相应项目，可通过包括 spintac_velocity.h 完成此步骤。

```
#include "sw/modules/spintac/src/32b/spintac_vel_plan.h"
```

13.5.2 定义配置数组的大小

该步骤应通过其余的宏定义来完成。 在实验 6c 示例项目中，该步骤将在主源文件最顶部完成。 通常，最好在规划中使用枚举来定义并标注状态。 这样可以轻松确定配置数组的大小，以便满足应用需求。 在实验 6c 示例项目中，该操作已完成。 13.4.2 节 中介绍了有关确定运动序列配置数组大小的内容。

```
#define ST_VELPLAN_CFG_ARRAY_DWORDS ((ST_VEL_PLAN_ACT_DWORDS   * 6) + \
                                     (ST_VEL_PLAN_COND_DWORDS  * 4) + \
                                     (ST_VEL_PLAN_VAR_DWORDS   * 5) + \
                                     (ST_VEL_PLAN_TRAN_DWORDS  * 7) + \
                                     (ST_VEL_PLAN_STATE_DWORDS * 6))
```

13.5.3 声明全局结构

这应该通过主源文件中的全局变量声明来完成。 在实验 6c 项目中，此结构包括在已声明为 spintac_velocity.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj st_obj;      // The SpinTAC Object
ST_Handle stHandle; // The SpinTAC Handle
unsigned long gWaterLevel = 0;     // Stores the water level in the washer
_iq gVelPanVar[ST_PLAN_MAX_VAR_NUM];  // Stores the values of SpinTAC Plan variables
// Configuration array for SpinTAC Plan
uint32_t stVelPlanCfgArray[ST_VELPLAN_CFG_ARRAY_DWORDS];
```

此示例为不希望使用已在 spintac_velocity.h 头文件中声明的 ST_Obj 结构的情况。

```
ST_VelPlan_t      stVelPlan;        //The SpinTAC Plan Object
ST_VELPLAN_Handle stVelPlanHandle; // The SpinTAC Plan Handle
uint32_t gWaterLevel = 0;     // Stores the water level in the washer
_iq gVelPanVar[ST_PLAN_MAX_VAR_NUM];   // Stores the values of SpinTAC Plan variables
// Configuration array for SpinTAC Plan
uint32_t stVelPlanCfgArray[ST_VELPLAN_CFG_ARRAY_DWORDS];
```

13.5.4 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这样会将所有默认值都加载到 SpinTAC 速度规划中。 该步骤可通过运行函数 ST_init（已在 spintac_velocity.h 头文件中声明）和函数 ST_setupVelPlan（已在 main.c 中声明）来完成。 如果不希望使用上述两个函数，可使用下方的代码示例配置 SpinTAC 速度规划组件。 本示例加载的是13.4.5 节 中介绍的洗衣机系统配置。 有关 SpinTAC Plan API 的详细信息，请参见节 3.5.3.1。

```
// init the ST VelPlan object
stVelPlanHandle = STVELPLAN_init(&stVelPlan, sizeof(ST_VelPlan_t));

// Pass the configuration array pointer into SpinTAC Velocity Plan
// Parameters: handle, pointer to array, size of array, number of actions, number of
conditions, number of variables, number of transitions, number of states
STVELPLAN_setCfgArray(stVelPlanHandle, &stVelPlanCfgArray[0], sizeof(stVelPlanCfgArray), 6, 4,
5, 7, 6);

// Establish the Velocity, Acceleration, and Jerk Maximums
_iq velMax = STVELMOVE_getVelocityMaximum(stVelMoveHandle);
```

```
_iq accMax = STVELMOVE_getAccelerationMaximum(stVelMoveHandle);
_iq jrkMax = STVELMOVE_getJerkMaximum(stVelMoveHandle);

// Configure SpinTAC Velocity Plan: Sample Time, VelMax, AccMax, DecMax, JrkMax, LoopENB
STVELPLAN_setCfg(stVelPlanHandle, _IQ24(ST_SPEED_SAMPLE_TIME),
                                  velMax, accMax, jrkMax, FALSE);
// Configure halt state: VelEnd, AccMax, JrkMax, Timer
STVELPLAN_setCfgHaltState(stVelPlanHandle, 0, accMax, jrkMax, 1000L);

//Example: STVELPLAN_addCfgState(handle, VelSetpoint[pups], StateTimer[ticks]);
//StateIdx0: Idle
STVELPLAN_addCfgState(stVelPlanHandle, 0,                                  2000L);
// StateIdx1: Fill
STVELPLAN_addCfgState(stVelPlanHandle, 0,                                  2000L);
// StateIdx2: AgiCW
STVELPLAN_addCfgState(stVelPlanHandle, _IQ24(0.25 * ST_SPEED_PU_PER_KRPM),  200L);
// StateIdx3: AgiCCW
STVELPLAN_addCfgState(stVelPlanHandle, _IQ24(-0.25 * ST_SPEED_PU_PER_KRPM), 200L);
// StateIdx4: Drain
STVELPLAN_addCfgState(stVelPlanHandle, 0,                                  2000L);
// StateIdx5: Dry
STVELPLAN_addCfgState(stVelPlanHandle, _IQ24(2 * ST_SPEED_PU_PER_KRPM),     2000L);

//Example: STVELPLAN_addCfgVar(handle, VarType,      InitialValue);
// VarIdx0: FillSensor {0: not filled; 1: filled}
STVELPLAN_addCfgVar(stVelPlanHandle,   ST_VAR_IN,    0);
// VarIdx1: DrainSensor {0: not drained; 1: drained}
STVELPLAN_addCfgVar(stVelPlanHandle,   ST_VAR_IN,    0);
// VarIdx2: CycleCounter
STVELPLAN_addCfgVar(stVelPlanHandle,   ST_VAR_INOUT, 0);
// VarIdx3: FillValve {0: valve closed; 1: valve open}
STVELPLAN_addCfgVar(stVelPlanHandle,   ST_VAR_OUT,   0);
// VarIdx4: DrainValve {0: valve closed; 1: valve open}
STVELPLAN_addCfgVar(stVelPlanHandle,   ST_VAR_OUT,   0);

//Example: STVELPLAN_addCfgCond(handle, VarIdx, Comparison,  Value1, Value2)
// CondIdx0: WaterFull Water is filled
STVELPLAN_addCfgCond(stVelPlanHandle,   0,      ST_COMP_EQ,  1,     0);
// CondIdx0: AgiNotDone SgitCycleCounter is greater than 0 (not done)
STVELPLAN_addCfgCond(stVelPlanHandle,   2,      ST_COMP_GT,  0,     0);
// CondIdx1: AgiDone SgitCycleCounter is equal or less than 0 (done)
STVELPLAN_addCfgCond(stVelPlanHandle,   2,      ST_COMP_ELW, 0,     0);
// CondIdx0: WaterEmpty Water is drained
STVELPLAN_addCfgCond(stVelPlanHandle,   1,      ST_COMP_EQ,  1,     0);
// Note: Set Value2 to 0 if Comparison is for only one value.

//Example: STVELPLAN_addCfgTran(handle, FromState, ToState, CondOption, CondIdx1,
CondiIdx2, AccLim[pups2], JrkLim[pups3]);
// From IdleState to FillState
STVELPLAN_addCfgTran(stVelPlanHandle, 0, 1, ST_COND_NC, 0, 0, _IQ24(0.1), _IQ20(1));
// From FillState to AgiState1
STVELPLAN_addCfgTran(stVelPlanHandle, 1, 2, ST_COND_FC, 0, 0, _IQ24(0.1), _IQ20(1));
// From AgiState1 to AgiState2
STVELPLAN_addCfgTran(stVelPlanHandle, 2, 3, ST_COND_NC, 0, 0, _IQ24(1), _IQ20(1));
// From AgiState2 to AgiState1
STVELPLAN_addCfgTran(stVelPlanHandle, 3, 2, ST_COND_FC, 1, 0, _IQ24(1), _IQ20(1));
// From AgiState2 to DrainState
STVELPLAN_addCfgTran(stVelPlanHandle, 3, 4, ST_COND_FC, 2, 0, _IQ24(0.1), _IQ20(1));
// From DrainState to DryState
STVELPLAN_addCfgTran(stVelPlanHandle, 4, 5, ST_COND_FC, 3, 0, _IQ24(0.2), _IQ20(1));
// From DryState to IdleState
STVELPLAN_addCfgTran(stVelPlanHandle, 5, 0, ST_COND_NC, 0, 0, _IQ24(0.1), _IQ20(1));
// Note: set CondIdx1 to 0 if CondOption is ST_COND_NC; set CondIdx2 to 0 if CondOption is
ST_COND_NC or ST_COND_FC

//Example: STVELPLAN_addCfgAct(handle, StateIdx, VarIdx, Operation, Value, ActionTriger);
```

```
// In IdleState, preset AgiCycleCounter to 20
STVELPLAN_addCfgAct(stVelPlanHandle, 0, ST_COND_NC, 0, 0, 2, ST_ACT_EQ, 20, ST_ACT_EXIT);
// Decrease AgiCycleCounter by 1 every time enters AgiState1
STVELPLAN_addCfgAct(stVelPlanHandle, 2, ST_COND_NC, 0, 0, 2, ST_ACT_ADD, -1, ST_ACT_ENTR);
// In FillState, set VarIdx3 to 1 to open FillValve
STVELPLAN_addCfgAct(stVelPlanHandle, 1, ST_COND_NC, 0, 0, 3, ST_ACT_EQ, 1, ST_ACT_ENTR);
// In FillState, set VarIdx3 to 0 to close FillValve when FillSensor = 1
STVELPLAN_addCfgAct(stVelPlanHandle, 1, ST_COND_NC, 0, 0, 3, ST_ACT_EQ, 0, ST_ACT_EXIT);
// In DrainState, set VarIdx4 to 1 to open DrainValve
STVELPLAN_addCfgAct(stVelPlanHandle, 4, ST_COND_NC, 0, 0, 4, ST_ACT_EQ, 1, ST_ACT_ENTR);
// In DrainState, set VarIdx4 to 0 to close DrainValve when DrainSensor = 1
STVELPLAN_addCfgAct(stVelPlanHandle, 4, ST_COND_NC, 0, 0, 4, ST_ACT_EQ, 0, ST_ACT_EXIT);

// If there was an error during the configuration, force Plan into the Halt State
if(STVELPLAN_getErrorID(stVelPlanHandle) != FALSE) {
  // Configure FSM: Ts, VelMax, AccMax, DecMax, JrkMax, LoopENB
  STVELPLAN_setCfg(stVelPlanHandle, _IQ24(ST_SPEED_SAMPLE_TIME),
                                    velMax, accMax, jrkMax, FALSE);
// Configure halt state: VelEnd, AccMax, JrkMax, Timer
STVELPLAN_setCfgHaltState(stVelPlanHandle, 0, accMax, jrkMax, 1000L);}
```

13.5.5 调用 SpinTAC 速度规划

该操作可在主循环中完成。 本代码示例中包括与注排水阀及传感器进行交互所需的代码。 作为洗衣机仿真的一部分，它还会更新水位信息。

```
if(gMotorVars.VelPlanRun == TRUE) {
  STVELPLAN_setEnable(stVelPlanHandle, TRUE);
}
// Run SpinTAC Velocity Plan
STVELPLAN_run(stVelPlanHandle);

// Update sensor values for SpinTAC Plan
// Get values for washer valve components
STVELPLAN _getVar(stVelPlanHandle, 3, &gVelPanVar[3]); // Get value of FillVale
STVELPLAN _getVar(stVelPlanHandle, 3, &gVelPanVar[4]);  // Get value of DrainValve
if(gVelPanVar[3] == TRUE) {
  // if FillValve is open, increase water level
  gWaterLevel += 1;
}
else if(gVelPanVar[4] == TRUE) {
  // if DrainValve is open, decrease water level
  gWaterLevel -= 1;
}
if(gWaterLevel >= WASHER_MAX_WATER_LEVEL) {
  // if water level is greater than maximum, set fill sensor to true
  gWaterLevel = WASHER_MAX_WATER_LEVEL;
  gVelPanVar[0] = TRUE;
}
else {
  // if water level is less than maximum, set FillSensor to false
  gVelPanVar[0] = FALSE;
}
if(gWaterLevel <= WASHER_MIN_WATER_LEVEL) {
  // if water level is at the minimum & set DrainSensor to true
  gWaterLevel = WASHER_MIN_WATER_LEVEL;
  gVelPanVar[1] = TRUE;
}
else {
  // if water level is greater than minimum, set DrainSensor to false
  gVelPanVar[1] = FALSE;
}
// Set values for washer sensor components
```

```
STVELPLAN _getVar(stVelPlanHandle, 0, gVelPanVar[0]); // Set value for FillSensor
STVELPLAN _getVar(stVelPlanHandle, 1, gVelPanVar[1]); // Set value for DrainSensor

if(STVELPLAN_getStatus(stVelPlanHandle) != ST_PLAN_IDLE) {
  // Send the profile configuration to SpinTAC Move
  gMotorVars.SpeedRef_krpm = _IQmpy(STVELPLAN_getVelocitySetpoint(stVelPlanHandle),
                            _IQ24(ST_SPEED_KRPM_PER_PU));
  gMotorVars.MaxAccel_krpmps = _IQmpy(STVELPLAN_getAccelerationLimit(stVelPlanHandle),
                             _IQ24(ST_SPEED_KRPM_PER_PU));
  gMotorVars.MaxJrk_krpmps2 = _IQ20mpy(STVELPLAN_getJerkLimit(stVelPlanHandle),
                             _IQ20(ST_SPEED_KRPM_PER_PU));
}
else {
  STVELPLAN_setEnable(stVelPlanHandle, FALSE);
}
```

13.5.6 调用 SpinTAC 速度规划节拍

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此函数。 抽取率由 ST_ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。

```
// Run SpinTAC Velocity Plan Tick
STVELPLAN_runTick(stVelPlanHandle);
```

13.5.7 根据 SpinTAC 速度移动状态更新 SpinTAC 速度规划

该步骤应在主 ISR 中完成。 SpinTAC 速度移动在完成系统配置时需要调用该函数。 这是为了警告 SpinTAC 速度规划，提示已达到其向 SpinTAC 速度移动提供的目标速度。 此函数应置于 SpinTAC 速度移动的函数调用之后。

```
// Update Plan when the profile is completed
if(STVELMOVE_getDone(stVelMoveHandle) != FALSE) {
STVELPLAN_setUnitProfDone(stVelPlanHandle, TRUE);
}
else {
  STVELPLAN_setUnitProfDone(stVelPlanHandle, FALSE);
}
```

13.6 SpinTAC 速度规划故障排除

13.6.1 ERR_ID

ERR_ID 为用户提供错误代码，用于识别引发错误的特定 SpinTAC 速度规划函数。 表 13-4 列出了在 SpinTAC 速度规划中定义的 ERR_ID。

表 13-4. SpinTAC 速度规划 ERR_ID

ERR_ID	规划函数
3000	STVELPLAN_addCfgCond
3001	STVELPLAN_delCfgCond
3002	STVELPLAN_setCfgCond
3003	STVELPLAN_getCfgCond
3004	STVELPLAN_addCfgTran
3005	STVELPLAN_delCfgTran

表 13-4. SpinTAC 速度规划 ERR_ID (continued)

ERR_ID	规划函数
3006	STVELPLAN_setCfgTran
3007	STVELPLAN_getCfgTran
3008	STVELPLAN_addCfgAct
3009	STVELPLAN_delCfgAct
3010	STVELPLAN_setCfgAct
3011	STVELPLAN_getCfgAct
3012	STVELPLAN_addCfgVar
3013	STVELPLAN_delCfgVar
3014	STVELPLAN_setCfgVar
3015	STVELPLAN_getCfgVar
3016	STVELPLAN_addCfgState
3017	STVELPLAN_delCfgState
3018	STVELPLAN_setCfgState
3019	STVELPLAN_setVar
3020	STVELPLAN_getVar
3021	STVELPLAN_setCfg
3022	STVELPLAN_setCfgHaltState
3023	STVELPLAN_setCfgArray
3024	STVELPLAN_addCfgVarCond
3025	STVELPLAN_delCfgVarCond
3026	STVELPLAN_setCfgVarCond
3027	STVELPLAN_getCfgVarCond
4001	STVELPLAN_run（SpinTAC 许可证无效。 使用具有 SpinTAC 有效许可证的芯片。）
4003	STVELPLAN_run（ROM 版本无效。 使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。）

13.6.2 配置错误

配置错误通过 SpinTAC 速度规划主结构中包含的 CfgError 结构上报。 该结构中包含了用于存储错误附加信息的元素。 这些元素将在下文介绍：

- CfgError.ERR_idx：识别出现错误的已配置元素实例。
- CfgError.ERR_code：识别引发错误的特定错误条件。

特定条件的 ERR_code 对于所有规划函数来说都是相同的。 表 13-5 中列出了在 SpinTAC 速度规划中定义的 ERR_code 和条件。

表 13-5. SpinTAC 速度规划 ERR_code

ERR_code	说明	解决方案
1	SpinTAC Plan 正在运行	运行配置前让 SpinTAC Plan 进入空闲状态。
2	超过最大状态数量	已配置最大状态数量。
3	超过最大条件数量	已配置最大条件数量。
4	超过最大转换数量	已配置最大转换数量。
5	超过最大操作数量	已配置最大操作数量。
6	超过最大变量数量	已配置最大变量数量。
7	采样时间值无效	将采样时间 cfg.T_sec 设置为 (0, 0.01] 范围内。
8	VelMax 值无效	在 (0, 1] 范围内选择 VelMax 值。

表 13-5. SpinTAC 速度规划 ERR_code (continued)

ERR_code	说明	解决方案
9	AccMax 值无效	在 [0.001, 120] 范围内选择 AccMax 值。
10	JrkMax 值无效	在 [0.0005, 2000] 范围内选择 JrkMax 值。
11	LoopENB 值无效	在 {false, true} 范围内选择 LoopENB 值。
12	VelEnd 值无效	在 (0, VelMax] 范围内选择 VelEnd 值。
13	AccLim 值无效	在 [0.001, AccMax] 范围内选择 AccLim 值。
14	JrkLim 值无效	在 [0.0005, JrkMax] 范围内选择 JrkLim 值。
15	Timer_tick 值无效	选择正整数值。
16	状态索引无效	该索引应为已配置的状态索引。
17	条件索引无效	该索引应为已配置的条件索引。
18	转换索引无效	该索引应为已配置的转换索引。
19	操作索引无效	该索引应为已配置的操作索引。
20	变量索引无效	该索引应为已配置的变量索引。
21	变量类型无效	从 ST_PlanVar_e 的值中选择变量类型。
22	对比值无效	从 ST_PlanComp_e 的值中选择对比值。
23	操作无效	从 ST_PlanActOptn_e 的值中选择操作。
24	AndOr 值无效	从 ST_PlanCond_e 的值中选择 AndOr 值。
25	变量类型错误	ST_VAR_OUT 变量的值无法设置。 ST_VAR_OUT 变量无法在条件中使用。 ST_VAR_IN 变量无法在操作中使用。
26	对比值错误	Value1 不应大于 Value2。
27	状态索引错误	在转换时，FromState 与 ToState 不能相同，但这些状态必须与已配置的某个状态相同。
28	转换时条件索引错误	在转换时：CondIdx1 与 CondIdx2 不能相同，但这些条件必须与已配置的某个条件相同
29	EnterExit 值错误	从 ST_PlanActTrgr_e 的值中选择 EnterExit 值
30	删除变量时 AndOr 错误	AndOr 值与 VarIdx 值冲突。 删除变量时，会在转换中引起配置错误。
31	变量附有操作，无法删除	在删除变量之前从操作配置中移除变量。
37	规划元素声明的规划配置数组过小	从配置中移除元素或者声明更大的配置数组。
38	状态附有转换，无法删除	在删除状态之前从转换配置中移除状态。
39	状态附有操作，无法删除	在删除状态之前从操作配置中移除状态。
40	变量对比值错误	变量对比条件的对比枚举值不能超过 ST_COMP_ELW。
41	变量不能与自身进行比较	确保送往函数的变量索引不同且有效。
42	无法通过基于值的条件索引获得基于变量的条件	发送已知包含基于变量的条件的索引。
43	条件附有转换，无法删除	在删除条件之前从转换配置中移除条件。

13.7 SpinTAC 位置规划的软件配置

配置 SpinTAC 位置规划需要七个步骤。 实验 13d“运动序列实例：自动售货机”是一个示例项目，其中执行了使用 SpinTAC 位置规划所需的步骤。 利用包含在 MotorWare 中的头文件 spintac_position.h，您可以快速将 SpinTAC 组件包括在项目中。

13.7.1 包括头文件

这应该通过其余的模块头文件包含来完成。 在实验 13d 示例项目中，该文件包括在 spintac_position.h 头文件中。 针对相应项目，可通过包括 spintac_position.h 完成此步骤。

```
#include "sw/modules/spintac/src/32b/spintac_pos_plan.h"
```

13.7.2 定义配置数组的大小

该步骤应通过其余的宏定义来完成。 在实验 13d 示例项目中，该步骤将在主源文件最顶部完成。 通常，最好在规划中使用枚举来定义并标注状态。 这样可以轻松确定配置数组的大小，以便满足应用需求。 在实验 13d 示例项目中，该操作已完成。 13.4.2 节 中介绍了有关确定运动序列配置数组大小的内容。

```
#define ST_POSPLAN_CFG_ARRAY_DWORDS ((ST_POS_PLAN_ACT_DWORDS     * 4) + \
                                     (ST_POS_PLAN_COND_DWORDS    * 6) + \
                                     (ST_POS_PLAN_VAR_DWORDS     * 6) + \
                                     (ST_POS_PLAN_TRAN_DWORDS    * 6) + \
                                     (ST_POS_PLAN_STATE_DWORDS   * 5))
```

13.7.3 声明全局结构

这应该通过主源文件中的全局变量声明来完成。 在实验 13d 项目中，此结构包括在已声明为 spintac_position.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj       st_obj;    // The SpinTAC Object
ST_Handle stHandle;  // The SpinTAC Handle
_iq gVendFwdButton = 0; // Button to advance the displayed item
_iq gVendSelectButton = 0; // Button to vend the displayed item
uint16_t gVendInventory[4] = {VEND_INITIAL_INVENTORY, VEND_INITIAL_INVENTORY,
                              VEND_INITIAL_INVENTORY, VEND_INITIAL_INVENTORY};
VEND_State_e gVendAvailableItem = VEND_ITEM0;  // Current item available to vend
// Configuration array for SpinTAC Position Plan
uint32_t stPosPlanCfgArray[ST_POSPLAN_CFG_ARRAY_DWORDS];
```

此示例为不希望使用已在 spintac_position.h 头文件中声明的 ST_Obj 结构的情况。

```
ST_PosPlan_t      stPosPlan;       //The SpinTAC Position Plan Object
ST_POSPLAN_Handle stPosPlanHandle; //The SpinTAC Position Plan Handle
_iq gVendFwdButton = 0; // Button to advance the displayed item
_iq gVendSelectButton = 0; // Button to vend the displayed item
uint16_t gVendInventory[4] = {VEND_INITIAL_INVENTORY, VEND_INITIAL_INVENTORY,
                              VEND_INITIAL_INVENTORY, VEND_INITIAL_INVENTORY};
VEND_State_e gVendAvailableItem = VEND_ITEM0;  // Current item available to vend
// Configuration array for SpinTAC Position Plan
uint32_t stPosPlanCfgArray[ST_POSPLAN_CFG_ARRAY_DWORDS];
```

13.7.4 初始化配置变量

这应该在死循环之前的项目主函数中完成。 这样会将所有的默认值加载到 SpinTAC 位置移动规划中。 该步骤可通过运行函数 ST_init（已在 spintac_position.h 头文件中声明）和函数 ST_setupPosPlan（已在 main.c 中声明）来完成。 如果不希望使用该函数，可使用下方的代码示例配置 SpinTAC 位置规划组件。 本示例加载的是13.4.6 节 中介绍的自动售货机系统配置。 有关 SpinTAC Plan API 的详细信息，请参见 图 3-19。

```
// Pass the configuration array pointer into SpinTAC Velocity Plan
STPOSPLAN_setCfgArray(stPosPlanHandle, &stPosPlanCfgArray[0],
                                       sizeof(stPosPlanCfgArray), 4, 6, 6, 6, 5);

// Establish the Velocity, Acceleration, Deceleration, and Jerk Maximums
velMax = STPOSMOVE_getVelocityMaximum(stPosMoveHandle);
accMax = STPOSMOVE_getAccelerationMaximum(stPosMoveHandle);
```

```
decMax = STPOSMOVE_getDecelerationMaximum(stPosMoveHandle);
jrkMax = STPOSMOVE_getJerkMaximum(stPosMoveHandle);

// Establish the Velocity, Acceleration, Deceleration, and Jerk Limits
velLim = _IQ24(0.1 * ST_SPEED_PU_PER_KRPM);
accLim = _IQ24(0.5 * ST_SPEED_PU_PER_KRPM);
decLim = _IQ24(0.5 * ST_SPEED_PU_PER_KRPM);
jrkLim = _IQ24(1.0 * ST_SPEED_PU_PER_KRPM);

// Configure SpinTAC Velocity Plan: Sample Time, VelMax, AccMax, DecMax, JrkMax, LoopENB
STPOSPLAN_setCfg(stPosPlanHandle, _IQ24(ST_SAMPLE_TIME), velMax, accMax, decMax, jrkMax, false);
// Configure halt state: PosStepInt, PosStepFrac, VelMax, AccMax, JrkMax, Timer
STPOSPLAN_setCfgHaltState(stPosPlanHandle, 0, 0, velMax, accMax, jrkMax, 1000L);

//Example: STPOSPLAN_addCfgState(handle, PosStepInt[MRev], PosStepFrac[MRev],
StateTimer[ticks]);
STPOSPLAN_addCfgState(stPosPlanHandle, 0, 0,            200L); // StateIdx0: Init
STPOSPLAN_addCfgState(stPosPlanHandle, 0, _IQ24(0.25),  200L); // StateIdx1: Item0
STPOSPLAN_addCfgState(stPosPlanHandle, 0, _IQ24(0.25),  200L); // StateIdx2: Item1
STPOSPLAN_addCfgState(stPosPlanHandle, 0, _IQ24(0.25),  200L); // StateIdx2: Item2
STPOSPLAN_addCfgState(stPosPlanHandle, 0, _IQ24(0.25),  200L); // StateIdx2: Item3

//Example: STPOSPLAN_addCfgVar(handle, VarType,      InitialValue);
STPOSPLAN_addCfgVar(stPosPlanHandle,   ST_VAR_INOUT, 0);  // VarIdx0: FwdButton
STPOSPLAN_addCfgVar(stPosPlanHandle,   ST_VAR_IN,    10); // VarIdx1: Item0Inv
STPOSPLAN_addCfgVar(stPosPlanHandle,   ST_VAR_IN,    10); // VarIdx2: Item1Inv
STPOSPLAN_addCfgVar(stPosPlanHandle,   ST_VAR_IN,    10); // VarIdx3: Item2Inv
STPOSPLAN_addCfgVar(stPosPlanHandle,   ST_VAR_IN,    10); // VarIdx4: Item3Inv
STPOSPLAN_addCfgVar(stPosPlanHandle,   ST_VAR_IN,    40); // VarIdx5: TotalInv

//Example: STPOSPLAN_addCfgCond(handle, VarIdx, Comparison,   Value1, Value2)
// CondIdx0: Fwd Button Pressed
STPOSPLAN_addCfgCond(stPosPlanHandle,   0,      ST_COMP_EQ,   1,      0);
// CondIdx1: Item0 Empty
STPOSPLAN_addCfgCond(stPosPlanHandle,   1,      ST_COMP_ELW,  0,      0);
// CondIdx2: Item1 Empty
STPOSPLAN_addCfgCond(stPosPlanHandle,   2,      ST_COMP_ELW,  0,      0);
// CondIdx3: Item2 Empty
STPOSPLAN_addCfgCond(stPosPlanHandle,   3,      ST_COMP_ELW,  0,      0);
// CondIdx4: Item3 Empty
STPOSPLAN_addCfgCond(stPosPlanHandle,   4,      ST_COMP_ELW,  0,      0);
// CondIdx5: TotalInv Empty
STPOSPLAN_addCfgCond(stPosPlanHandle,   5,      ST_COMP_ELW,  0,      0);

//Example: STPOSPLAN_addCfgTran(handle, FromState, ToState, CondOption, CondIdx1, CondIdx2,
VelLim[pups], AccLim[pups2], DecLim[pups2], JrkLim[pups3]);
// NOTE: The deceleration limit must be set between the following bounds [acceleration limit,
10*acceleration limit]
// From Init to Item1
STPOSPLAN_addCfgTran(stPosPlanHandle, 0, 2, ST_COND_OR, 0, 2, velLim, accLim, decLim, jrkLim);
// From Item3 to Init
STPOSPLAN_addCfgTran(stPosPlanHandle, 1, 0, ST_COND_FC, 5, 0, velLim, accLim, decLim,
jrkLim);
// From Item0 to Item1
STPOSPLAN_addCfgTran(stPosPlanHandle, 1, 2, ST_COND_OR, 0, 1, velLim, accLim, decLim,
jrkLim);
// From Item1 to Item2
STPOSPLAN_addCfgTran(stPosPlanHandle, 2, 3, ST_COND_OR, 0, 2, velLim, accLim, decLim,
jrkLim);
// From Item2 to Item3
STPOSPLAN_addCfgTran(stPosPlanHandle, 3, 4, ST_COND_OR, 0, 3, velLim, accLim, decLim,
jrkLim);
// From Item3 to Item0
STPOSPLAN_addCfgTran(stPosPlanHandle, 4, 1, ST_COND_OR, 0, 4, velLim, accLim, decLim,
jrkLim);
```

```
//Example: STPOSPLAN_addCfgAct(handle, StateIdx, VarIdx, Operation, Value, ActionTriger);
// In Item1, clear Fwd Button
STPOSPLAN_addCfgAct(stPosPlanHandle, 1, ST_COND_NC, 0, 0, 0, ST_ACT_ENTR);
// In Item2, clear Fwd Button
STPOSPLAN_addCfgAct(stPosPlanHandle, 2, ST_COND_NC, 0, 0, 0, ST_ACT_ENTR);
// In Item3, clear Fwd Button
STPOSPLAN_addCfgAct(stPosPlanHandle, 3, ST_COND_NC, 0, 0, 0, ST_ACT_ENTR);
// In Item4, clear Fwd Button
STPOSPLAN_addCfgAct(stPosPlanHandle, 4, ST_COND_NC, 0, 0, 0, ST_ACT_ENTR);

if(STPOSPLAN_getErrorID(stPosPlanHandle) != false) {
  // Configure FSM: Ts, VelMax, AccMax, DecMax, JrkMax, LoopENB
  STPOSPLAN_setCfg(stPosPlanHandle, _IQ24(ST_SAMPLE_TIME), velMax, accMax, decMax, jrkMax,
false);
  // Configure halt state: PosStepInt[MRev], PosStepFrac[MRev], VelMax, AccMax, JrkMax, Timer
  STPOSPLAN_setCfgHaltState(stPosPlanHandle, 0, 0, velMax, accMax, jrkMax, 1000L);
}
```

13.7.5 调用 SpinTAC 位置规划

该操作可在主循环中完成。 本代码示例中包括与注排水阀及传感器进行交互所需的代码。 作为自动售货机仿真的一部分，它还会更新商品存货信息。

```
// SpinTAC Position Plan
if(gPosPlanRunFlag == ST_PLAN_STOP
                                  && gMotorVars.SpinTAC.PosPlanRun == ST_PLAN_START) {
  if(STPOSMOVE_getDone(stPosMoveHandle) == true) {
    if(STPOSPLAN_getErrorID(stPosPlanHandle) != false) {
      STPOSPLAN_setEnable(stPosPlanHandle, false);
      STPOSPLAN_setReset(stPosPlanHandle, true);
      gMotorVars.SpinTAC.PosPlanRun = gPosPlanRunFlag;
    }
    else {
      STPOSPLAN_setEnable(stPosPlanHandle, true);
      STPOSPLAN_setReset(stPosPlanHandle, false);
      gPosPlanRunFlag = gMotorVars.SpinTAC.PosPlanRun;
    }
  }
}
if(gMotorVars.SpinTAC.PosPlanRun == ST_PLAN_STOP) {
  STPOSPLAN_setReset(stPosPlanHandle, true);
  gPosPlanRunFlag = gMotorVars.SpinTAC.PosPlanRun;
}
if(gPosPlanRunFlag == ST_PLAN_START
                                  && gMotorVars.SpinTAC.PosPlanRun == ST_PLAN_PAUSE) {
  STPOSPLAN_setEnable(stPosPlanHandle, false);
  gPosPlanRunFlag = gMotorVars.SpinTAC.PosPlanRun;
}
if(gPosPlanRunFlag == ST_PLAN_PAUSE
                                  && gMotorVars.SpinTAC.PosPlanRun == ST_PLAN_START) {
  STPOSPLAN_setEnable(stPosPlanHandle, true);
  gPosPlanRunFlag = gMotorVars.SpinTAC.PosPlanRun;
}

// if we have selected an item from the machine
if(gVendSelectButton == 1) {
  if(STPOSPLAN_getStatus(stPosPlanHandle) != ST_PLAN_IDLE) {
    // decrease our inventory
    gVendInventory[gVendAvailableItem - 1]--;
  }
  // toggle the select button off
  gVendSelectButton = 0;
```

```
}

// Update variables passed into Plan
STPOSPLAN_setVar(stPosPlanHandle, VEND_Fwd, gVendFwdButton);
STPOSPLAN_setVar(stPosPlanHandle, VEND_Item0Inv, gVendInventory[VEND_ITEM0 - 1]);
STPOSPLAN_setVar(stPosPlanHandle, VEND_Item1Inv, gVendInventory[VEND_ITEM1 - 1]);
STPOSPLAN_setVar(stPosPlanHandle, VEND_Item2Inv, gVendInventory[VEND_ITEM2 - 1]);
STPOSPLAN_setVar(stPosPlanHandle, VEND_Item3Inv, gVendInventory[VEND_ITEM3 - 1]);
STPOSPLAN_setVar(stPosPlanHandle, VEND_TotalInv,
      gVendInventory[0] + gVendInventory[1] + gVendInventory[2] + gVendInventory[3]);

// Run SpinTAC Position Plan
STPOSPLAN_run(stPosPlanHandle);

// display the selected item
if(STPOSPLAN_getCurrentState(stPosPlanHandle) > 0) {
  gVendAvailableItem = (VEND_State_e)STPOSPLAN_getCurrentState(stPosPlanHandle);
}
else {
  gVendAvailableItem = VEND_ITEM0;
}

// Update variables passed out of Plan
if(STPOSPLAN_getFsmState(stPosPlanHandle) == ST_FSM_STATE_STAY) {
  STPOSPLAN_getVar(stPosPlanHandle, VEND_Fwd, &gVendFwdButton);
}

if(STPOSPLAN_getStatus(stPosPlanHandle) != ST_PLAN_IDLE) {
  // Send the profile configuration to SpinTAC Position Profile Generator
  STPOSPLAN_getPositionStep_mrev(stPosPlanHandle,
        (_iq24 *)&gMotorVars.PosStepInt_MRev, (_iq24 *)&gMotorVars.PosStepFrac_MRev);
  gMotorVars.MaxVel_krpm = _IQmpy(STPOSPLAN_getVelocityLimit(stPosPlanHandle),
                                                        _IQ24(ST_SPEED_KRPM_PER_PU));
  gMotorVars.MaxAccel_krpmps =_IQmpy(STPOSPLAN_getAccelerationLimit(stPosPlanHandle),
                                                        _IQ24(ST_SPEED_KRPM_PER_PU));
  gMotorVars.MaxDecel_krpmps =_IQmpy(STPOSPLAN_getDecelerationLimit(stPosPlanHandle),
                                                        _IQ24(ST_SPEED_KRPM_PER_PU));
  gMotorVars.MaxJrk_krpmps2 = _IQ20mpy(STPOSPLAN_getJerkLimit(stPosPlanHandle),
                                                        _IQ20(ST_SPEED_KRPM_PER_PU));
}
else {
  if(gPosPlanRunFlag == ST_PLAN_START
                                && gMotorVars.SpinTAC.PosPlanRun == ST_PLAN_START)
  {
    gMotorVars.SpinTAC.PosPlanRun = ST_PLAN_STOP;
    gPosPlanRunFlag = gMotorVars.SpinTAC.PosPlanRun;
  }
}
```

13.7.6 调用 SpinTAC 位置规划节拍

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此函数。 抽取率由 ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。

```
// Run SpinTAC Position Plan Tick
STPOSPLAN_runTick(stPosPlanHandle);
```

13.7.7 根据 SpinTAC 位置移动状态更新 SpinTAC 位置规划

该步骤应在主 ISR 中完成。 SpinTAC 位置移动在完成系统配置时需要调用该函数。 这是为了警告 SpinTAC 位置规划，提示已达到其向 SpinTAC 位置规划提供的目标位置。 此函数应置于 SpinTAC 位置移动的函数调用之后。

```
// Update SpinTAC Position Plan when the profile is completed
if(STPOSMOVE_getDone(stPosMoveHandle) != false) {
  STPOSPLAN_setUnitProfDone(stPosPlanHandle, true);
}
else {
  STPOSPLAN_setUnitProfDone(stPosPlanHandle, false);
}
```

13.8 SpinTAC 位置规划故障排除

13.8.1 ERR_ID

ERR_ID 为用户提供错误代码，用于识别引发错误的特定 SpinTAC 位置规划函数。 表 13-6 列出了在 SpinTAC 位置规划中定义的 ERR_ID。

表 13-6. SpinTAC 位置规划 ERR_ID

ERR_code	规划函数
3000	STPOSPLAN_addCfgCond 中存在配置错误
3001	STPOSPLAN_delCfgCond 中存在配置错误
3002	STPOSPLAN_setCfgCond 中存在配置错误
3003	STPOSPLAN_getCfgCond 中存在配置错误
3004	STPOSPLAN_addCfgTran 中存在配置错误
3005	STPOSPLAN_delCfgTran 中存在配置错误
3006	STPOSPLAN_setCfgTran 中存在配置错误
3007	STPOSPLAN_getCfgTran 中存在配置错误
3008	STPOSPLAN_addCfgAct 中存在配置错误
3009	STPOSPLAN_delCfgAct 中存在配置错误
3010	STPOSPLAN_setCfgAct 中存在配置错误
3011	STPOSPLAN_getCfgAct 中存在配置错误
3012	STPOSPLAN_addCfgVar 中存在配置错误
3013	STPOSPLAN_delCfgVar 中存在配置错误
3014	STPOSPLAN_setCfgVar 中存在配置错误
3015	STPOSPLAN_getCfgVar 中存在配置错误
3016	STPOSPLAN_addCfgState 中存在配置错误
3017	STPOSPLAN_delCfgState 中存在配置错误
3018	STPOSPLAN_setCfgState 中存在配置错误
3019	STPOSPLAN_setVar 中存在配置错误
3020	STPOSPLAN_getVar 中存在配置错误
3021	STPOSPLAN_setCfg 中存在配置错误
3022	STPOSPLAN_setCfgHaltState 中存在配置错误
3023	STPOSPLAN_setCfgArray 中存在配置错误
3024	STPOSPLAN_addCfgVarCond 中存在配置错误
3025	STPOSPLAN_delCfgVarCond 中存在配置错误
3026	STPOSPLAN_setCfgVarCond 中存在配置错误

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

表 13-6. SpinTAC 位置规划 ERR_ID (continued)

ERR_code	规划函数
3027	STPOSPLAN_getCfgVarCond 中存在配置错误
4001	STPOSPLAN_run（SpinTAC 许可证无效。 使用具有 SpinTAC 有效许可证的芯片。）
4003	STPOSPLAN_run（ROM 版本无效。 使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。）

13.8.2 配置错误

配置错误通过 SpinTAC 位置规划主结构中包含的 CfgError 结构上报。 该结构中包含了用于存储错误附加信息的元素。 这些元素将在下文介绍：

- CfgError.ERR_idx：识别出现错误的已配置元素实例。
- CfgError.ERR_code：识别引发错误的特定错误条件。

特定条件的 ERR_code 对于所有规划函数来说都是相同的。 表 13-7 中列出了在 SpinTAC 位置规划中定义的 ERR_code 和条件。

表 13-7. SpinTAC 位置规划 ERR_code

ERR_code	说明	解决方案
1	SpinTAC Plan 正在运行	运行配置前让 SpinTAC Plan 进入空闲状态。
2	超过最大状态数量	已配置最大状态数量。
3	超过最大条件数量	已配置最大条件数量。
4	超过最大转换数量	已配置最大转换数量。
5	超过最大操作数量	已配置最大操作数量。
6	超过最大变量数量	已配置最大变量数量。
7	采样时间值无效	将采样时间 cfg.T_sec 设置为 (0, 0.01] 范围内。
8	VelMax 值无效	在 (0, 1] 范围内选择 VelMax 值。
9	AccMax 值无效	在 [0.001, 120] 范围内选择 AccMax 值。
10	JrkMax 值无效	在 [0.0005, 2000] 范围内选择 JrkMax 值。
11	LoopENB 值无效	在 {false, true} 范围内选择 LoopENB 值。
12	VelEnd 值无效	在 (0, VelMax] 范围内选择 VelEnd 值。
13	AccLim 值无效	在 [0.001, AccMax] 范围内选择 AccLim 值。
14	JrkLim 值无效	在 [0.0005, JrkMax] 范围内选择 JrkLim 值。
15	Timer_tick 值无效	选择正整数值。
16	状态索引无效	该索引应为已配置的状态索引。
17	条件索引无效	该索引应为已配置的条件索引。
18	转换索引无效	该索引应为已配置的转换索引。
19	操作索引无效	该索引应为已配置的操作索引。
20	变量索引无效	该索引应为已配置的变量索引。
21	变量类型无效	从 ST_PlanVar_e 的值中选择变量类型。
22	对比值无效	从 ST_PlanComp_e 的值中选择对比值。
23	操作无效	从 ST_PlanActOptn_e 的值中选择操作。
24	AndOr 值无效	从 ST_PlanCond_e 的值中选择 AndOr 值。
25	变量类型错误	ST_VAR_OUT 变量的值无法设置。 ST_VAR_OUT 变量无法在条件中使用。 ST_VAR_IN 变量无法在操作中使用。
26	对比值错误	Value1 不应大于 Value2。
27	状态索引错误	在转换时，FromState 与 ToState 不能相同，但这些状态必须与已配置的某个状态相同。

表 13-7. SpinTAC 位置规划 ERR_code (continued)

ERR_code	说明	解决方案
28	转换时条件索引错误	在转换时：CondIdx1 与 CondIdx2 不能相同，但这些条件必须与已配置的某个条件相同
29	EnterExit 值错误	从 ST_PlanActTrgr_e 的值中选择 EnterExit 值。
30	删除变量时 AndOr 错误	AndOr 值与 VarIdx 值冲突。 删除变量时，会在转换中引起配置错误。
31	变量附有操作，无法删除	在删除变量之前从操作配置中移除变量。
32	VelLim 值无效	在 (0, VelMax) 范围内选择 VelLim 值。
33	DecLim 值无效	在 [0.001, DecMax] 范围内选择 DecLim 值，并将 DecLim/AccLim 比例限制在 [0.1, 10] 之内。
34	DecMax 值无效	在 [0.001, 120] 范围内选择 DecLim 值，并将 DecLim/AccLim 比例限制在 [0.1, 10] 之内。
35	PosStepInt_mrev 或 PosStepFrac_mrev 对 HaltState 无效	将 PosStepInt_mrev 设置为 [-2, 2] 范围内，并将 PosStepFrac_mrev 设置为 (-1, 1) 范围内。
36	PosStepInt_mrev 或 PosStepFrac_mrev 对状态无效	将 PosStepInt_mrev 设置为 [-2147483647, 2147483647] 范围内，并将 PosStepFrac_mrev 设置为 (-1, 1) 范围内。
37	规划元素声明的规划配置数组过小	从配置中移除元素或者声明更大的配置数组。
38	状态附有转换，无法删除	在删除状态之前从转换配置中移除状态。
39	状态附有操作，无法删除	在删除状态之前从操作配置中移除状态。
40	变量对比值错误	变量对比条件的对比枚举值不能超过 ST_COMP_ELW。
41	变量不能与自身进行比较	确保送往函数的变量索引不同且有效。
42	无法通过基于值的条件索引获得基于变量的条件	发送已知包含基于变量的条件的索引。
43	条件附有转换，无法删除	在删除条件之前从转换配置中移除条件。
44	首个状态的 PosStep 必须为 0 [MRev]	配置首个状态，使 PosStepInt 和 PosStepFrac 均等于 0。

13.9 结论

InstaSPIN-MOTION 提供了一种设计轨迹变化和运动序列的简便方法， 方便您快速执行应用。 借此，您可以非常快捷地完成运动序列的设计和测试工作。 SpinTAC 速度移动可生成基于约束、时间最优且可重复使用的轨迹系统配置。 这些系统配置由执行应用运动序列的 SpinTAC 速度规划触发。

管理启动、低速和换向时的满负载

在电机轴承受机械负载低速运转时，需要对 InstaSPIN-FOC 中 FAST 算法的某些方面加以考虑。 本文档介绍在电机低速运转期间进行无传感器控制时，常见电机控制问题的几个方面。 有关 FAST 性能的综合实验结果，请参见《*MS320F2806xF InstaSPIN-FOC 技术参考手册*》（文献编号 SPRUHI9）、《*TMS320F2805xF InstaSPIN-FOC 技术参考手册*》（文献编号 SPRUHP4）和《*TMS320F2802xF InstaSPIN-FOC 技术参考手册*》（文献编号 SPRUHP4）。

本章介绍如下电机控制情形：

1. 满载低速运行
2. 满载换向
3. 满载电机启动
4. 满载从静止状态快速加速
5. 过载和电机过热

在这些实验中，系统使用了以下组件：

- 德州仪器 (TI) C2000 处理器：装有 InstaSPIN-FOC 版本 1.6 的 TMS320F28069F
- 德州仪器 (TI) 逆变器模块：TMDSHVMTRPFCKIT 版本 1.1
- 具有以下特性的 IPM 电机：
  - 额定电压 = 300V
  - 额定电流 = 4A
  - 电机最大电流 = 6A
  - 定子电阻 (Rs) = 2.6Ω
  - 定子正交电感 (Ls_q) = 13.5mH
  - 定子直接电感 (Ls_d) = 11.5mH
  - 转子磁通量 (ψ) = 0.5V/Hz = 0.08Wb
  - 额定转矩 = 1.9N.m
- Magtrol 测力计型号：HD-715-8N
- Magtrol 测力计控制器型号：DSP6001

待测电机连接至测力计，如 图 14-1 所示。

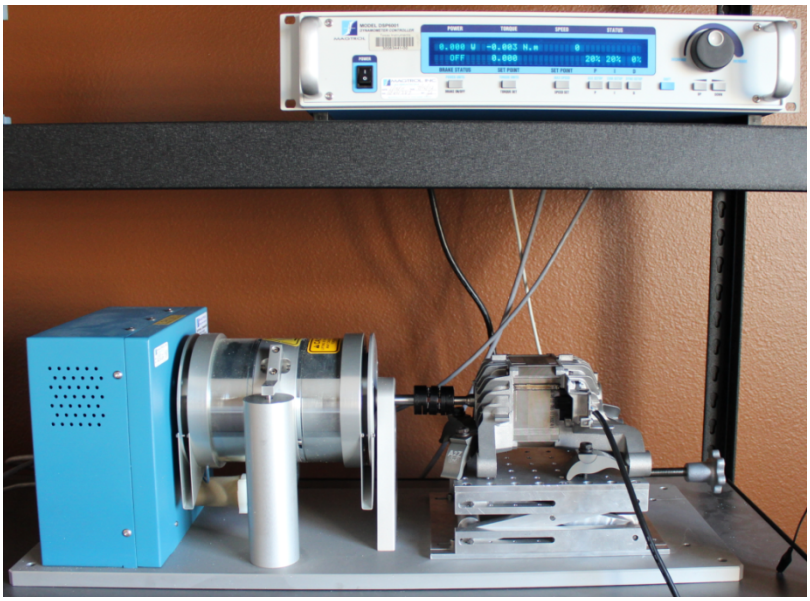


图 **14-1.** 测试装置照片

Topic **Page**

14.1 满载低速运行

为了以低速运行，首先让我们讨论无机械负载的低速运行情况。 无负载时可控的最慢速度可通过电机转速和电流波形直观地观察得到。 例如，测试电机以 30RPM（或 2Hz）的低速运行时，可能产生一个 30 ±3RPM 的速度变化。 不过，为保持速度相当恒定，所用的速度控制器必须有良好的响应能力，足以补偿速度变化。

14.1.1 满载低速运行时的考量

总之，电机以低速运行时需要考虑以下方面：

- 启用偏移重校准；在节 14.1.1.1中说明。
- 启用定子 Rs 重校准；在节 14.1.1.2中说明。
- 禁用强制角；在节 14.1.1.3中说明。
- 调整速度控制器以避免电机停转；在节 14.1.1.4中说明。
- 调整电压反馈电路；在节 14.1.1.5中说明。

14.1.1.1 启用偏移重校准

要达到预期的低速满载性能，需要精确校准电流和电压的偏移值。 为此，应该在闭环中运行电机前启用控制器对象的偏移重校准。 以下示例代码将启用偏移重校准。

```
// enable automatic calculation of bias values
CTRL_setFlag_enableOffset(ctrlHandle, TRUE);
```

注意，必须在启用控制器之前调用偏移重校准的使能函数，即调用 CTRL_setFlag_enableCtrl(ctrlHandle, TRUE) 函数。

偏移重校准对于实现低速运行性能至关重要。

偏移重校准对于实现低速运行性能至关重要，具体而言就是校准偏移电压。由于在低速范围内，电机的反馈电压趋向于很小的值，所以必须要能够精确校准偏移电压。

14.1.1.2 启用定子 Rs 重校准

电机以低速运行时需要考虑的另一个重要因素是软件中的电机模型表示。 定子电阻对于参数估算的准确性非常重要。 因此应该在启用控制器前启用 Rs 重校准。 下列示例代码将启用 Rs 重校准。

```
// enable Rs recalibration
EST_setFlag_enableRsRecalc(obj->estHandle,TRUE);
```

14.1.1.3 禁用强制角

电机以低速运行时必须禁用强制角，使得估算器能够不受外部强制角的干扰而得出估算值。 可以使用以下代码示例来禁用强制角：

强制角必须禁用以允许估算器收敛。

```
// disable the forced angle
EST_setFlag_enableForceAngle(obj->estHandle,FALSE);
```

14.1.1.4 调整速度控制器以避免电机停转

为了以低速连续驱动电机而不发生电机停转，用户必须调整速度控制器，使速度控制器的响应速度快到足以避免电机在转矩瞬变期间完全停止。 以下函数可以用于更新 InstaSPIN 内部的速度控制器：

```
_iq New_Kp_spd;
_iq New_Ki_spd;

// set the kp and ki speed controller gains
CTRL_setKp(handle,CTRL_Type_PID_spd, New_Kp_spd);
CTRL_setKi(handle,CTRL_Type_PID_spd, New_Ki_spd);
```

14.1.1.5 调整电压反馈电路

低速运转的前提是假定来自电机的反电动势电压非常小。 由于 FAST 算法要求使用相电压作为算法的输入，我们鼓励用户对每伏特电压使用最大的 ADC 位数。 例如，如果系统的最大输入电压为 400V，则强烈建议对输入电压进行反馈，从而在电机的反电动势出现低电压时（即在低速运行期间），ADC 转换器输出提供最大位数值以便能评估到更多信息来支持电机模型。

为说明使用每伏最大位数的重要性，请使用 TMDSHVMTRPFCKIT 版本 1.1 实现电压反馈，如图 14-2 所示。

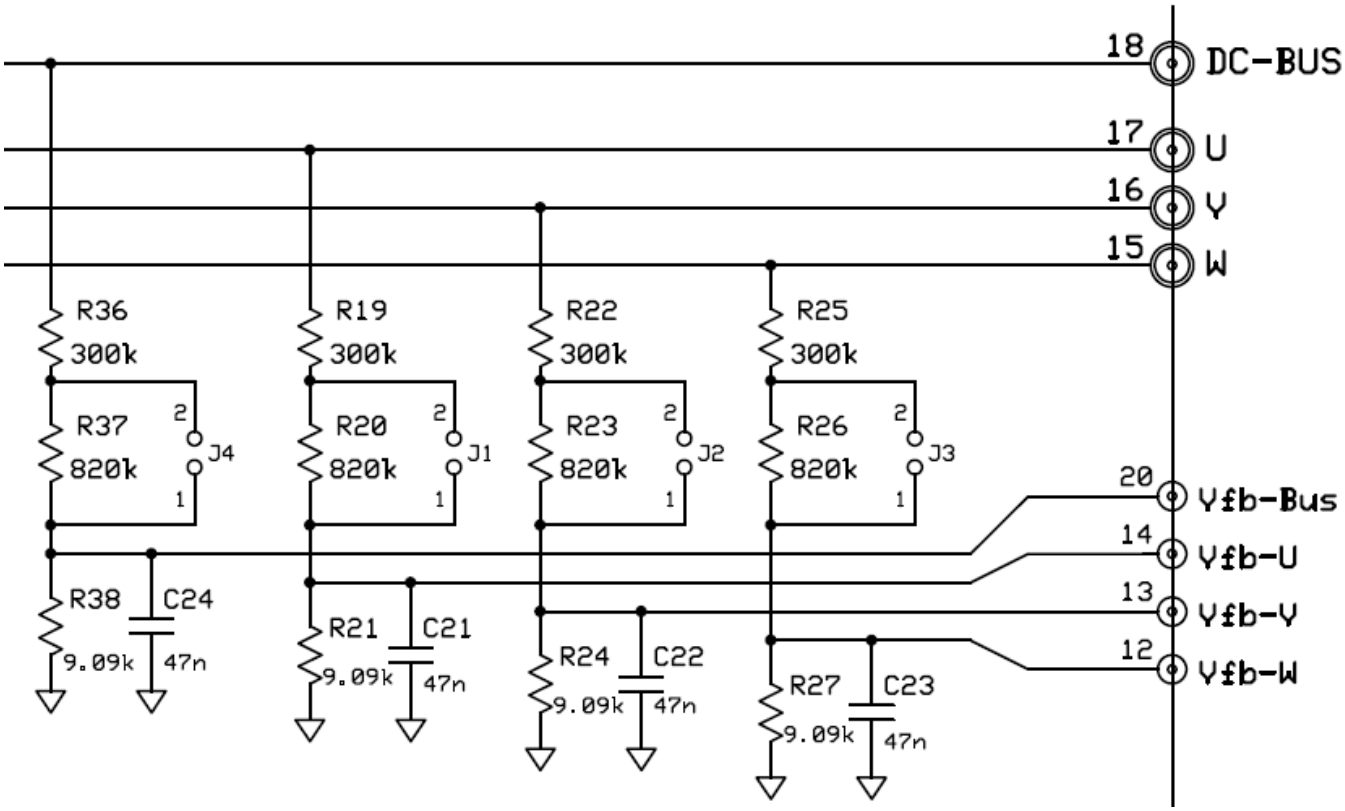


图 **14-2.** 高压套件的电压反馈电路 **(TMDSHVMTRPFCKIT)**

电路的跳线 J1、J2、J3 和 J4 开路（请参见 PCB 的 M5 部分），当 DC-BUS 为 409.9V 时，将得到最大 3.3V 的 Vfb-Bus。这意味着该电路能够为输入为 409.9V 的电机提供最佳的 ADC 分辨率。不过，如果电机运行在仅 100V 的电压下时，表示电机内部电压的 ADC 分辨率将会非常小，实际上大约为实际值的 ¼。 如果已知电机运行电压为 100V，则应对以上电路作出调整以便 DC-BUS 为 100V 时，ADC 引脚上出现最大 3.3V 的 Vfb-Bus。

*调整电压反馈电路以最大化 **ADC 3.3V** 输入范围。*

更改电压反馈值将影响 user.h 文件中配置的两个参数。 例如在上述原理图中，若 J1、J2、J3 和 J4 开路，则需要在 user.h 中定义以下参数：

```
#define USER_ADC_FULL_SCALE_VOLTAGE_V    (409.9)
#define USER_VOLTAGE_FILTER_POLE_Hz    (375.5)
```

这可通过如下公式导出（J1、J2、J3 和 J4 开路）：

ADC_IN_{max} = 3.3V

R_1 = 300kΩ + 820kΩ = 1120kΩ

R_2 = 9.09kΩ

$$USER\_ADC\_FULL\_SCALE\_VOLTAGE\_V = \frac{ADC\_IN_{max} \times R_1 + R_2}{R_2} = 409.9\ V$$

$$R_{Parallel} = \frac{R_2 \times R_1}{R_2 + R_1} = 9.017\ k\Omega$$

$$C = 47\ nF$$

$$USER\_VOLTAGE\_FILTER\_POLE\_Hz = \frac{1}{2 \times \pi \times R_{Parallel} \times C} = 375.5\ Hz$$

当 J1、J2、J3 和 J4 短路时，需要在 user.h 中重新定义以下参数：

```
#define USER_ADC_FULL_SCALE_VOLTAGE_V    (112.2)
#define USER_VOLTAGE_FILTER_POLE_Hz    (383.8)
```

这可通过如下公式导出：

ADC_IN_{max} = 3.3V

R_1 = 300kΩ

R_2 = 9.09kΩ

$$USER\_ADC\_FULL\_SCALE\_VOLTAGE\_V = \frac{ADC\_IN_{max} \times R_1 + R_2}{R_2} = 112.2\ V \quad (79)$$

$$R_{Parallel} = \frac{R_2 \times R_1}{R_2 + R_1} = 8.823\ k\Omega$$

$$C = 47\ nF$$

$$USER\_VOLTAGE\_FILTER\_POLE\_Hz = \frac{1}{2 \times \pi \times R_{Parallel} \times C} = 383.8\ Hz$$

14.1.2 瞬时满载低速运行示例

在考虑以上要求后，让我们来看看待测电机使用测力计时的几个低速响应示例。

14.1.2.1 4Hz，无负载至满载瞬态

图 14-3 显示了此条件下的电流波形：

- 测力计 = 1.9N·m（满载）
- 速度控制器 = 4Hz (60RPM ± 1RPM)

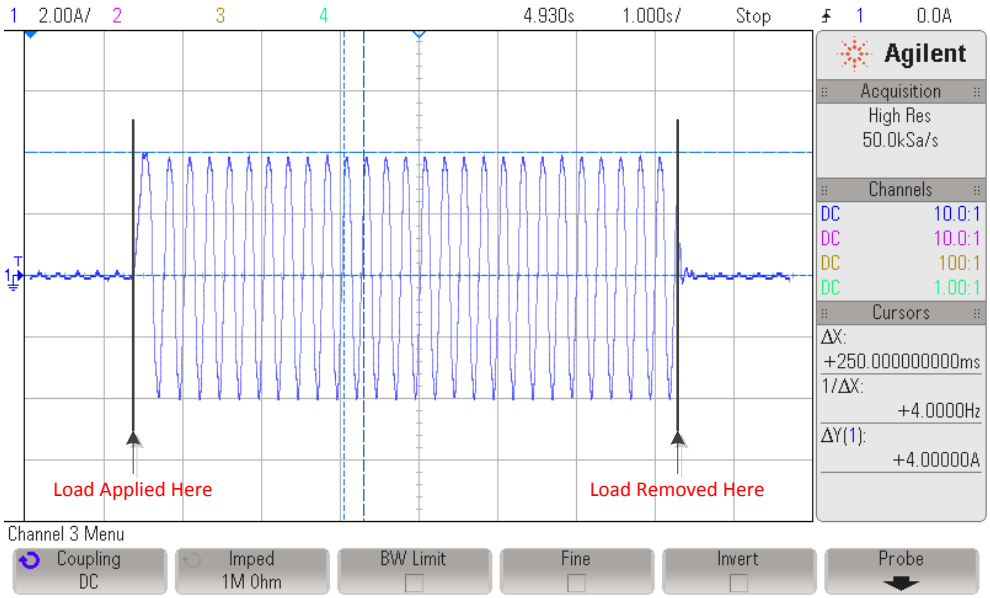


图 14-3. 4Hz，无负载至满载瞬态图

将 1.9Nm 的电机额定转矩瞬间施加到电机轴，产生 4A 的电流。示波器显示的电频率为 4Hz。 对于一个 4 对电极的电机，此频率一旦稳定后，将得到一个 60 ± 1RPM 的转速。 请注意，应用负载时的时间可能不同于采集变量时的时间。 不过，示波器上显示的应用转矩与采集变量时的完全相同。 时间不同的原因是，尽管具有相同的参数，但是捕获电流是在不同测试中采集到的。

FAST 代表磁通量 (Flux)、角度 (Angle)、速度 (Speed) 和转矩 (Torque)。图 14-4、图 14-5、图 14-6 和图 14-7 显示了 FAST 算法的表现以及转矩阶跃命令是如何影响 FAST 输出变量的。

*即使在 **100%** 阶跃负载的情况下，**FAST** 变量也保持不变。*

图 14-4 是估算的电机磁通量。 它实际为 FAST 提供的磁链，可以看出相当稳定。 造成磁通量变化的原因是多方面的，比如电机参数准确性以及特定负载情况下电机磁路设计的好坏。

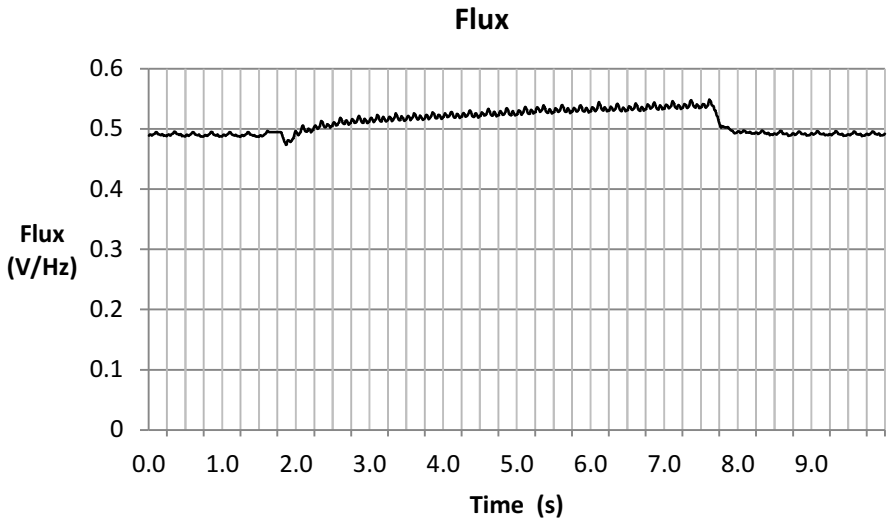


图 **14-4.** 磁通量图

图 14-5、图 14-6 和图 14-7 显示了由 FAST 提供的磁通角。 如图所示，角度随着电机负载的增加和减少而变化。

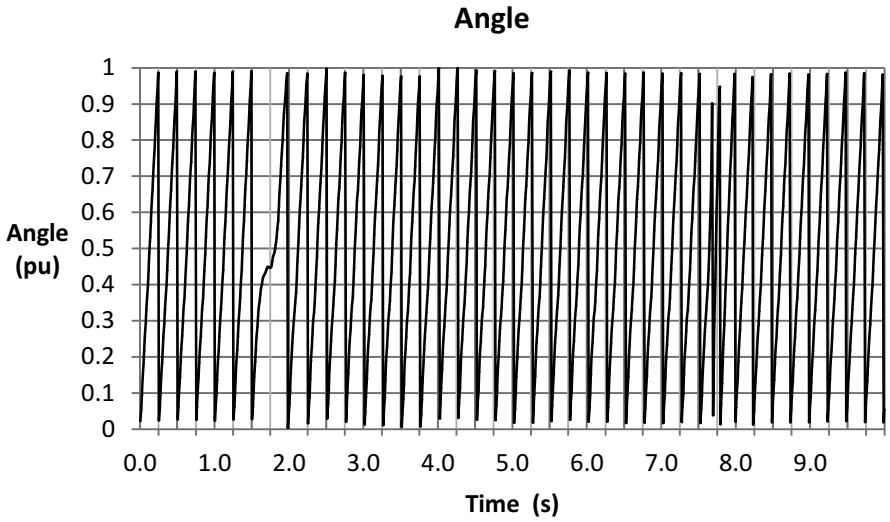


图 **14-5.** 角度图

如果在电机负载时角度放大，可以看出角度的变化率是如何变为很低的，并且一旦速度控制器进行相应校正后，该变化率就变回到指定速度。

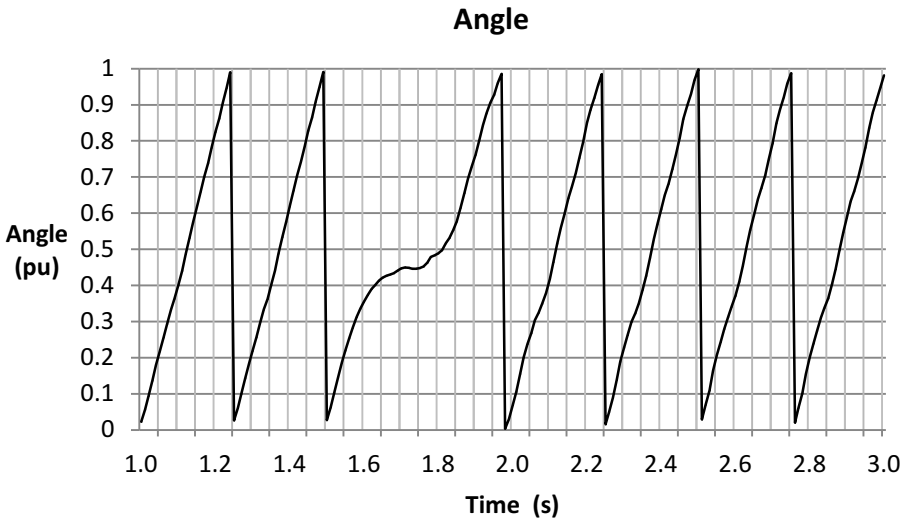


图 **14-6.** 放大的角度图 **-** 电机负载

当负载从电机轴移除时，可以观察到同样的情况。 由于速度控制器提供的转矩命令，将使转速提高，一段时间后，速度控制器将转速下调至 4Hz (60RPM)。

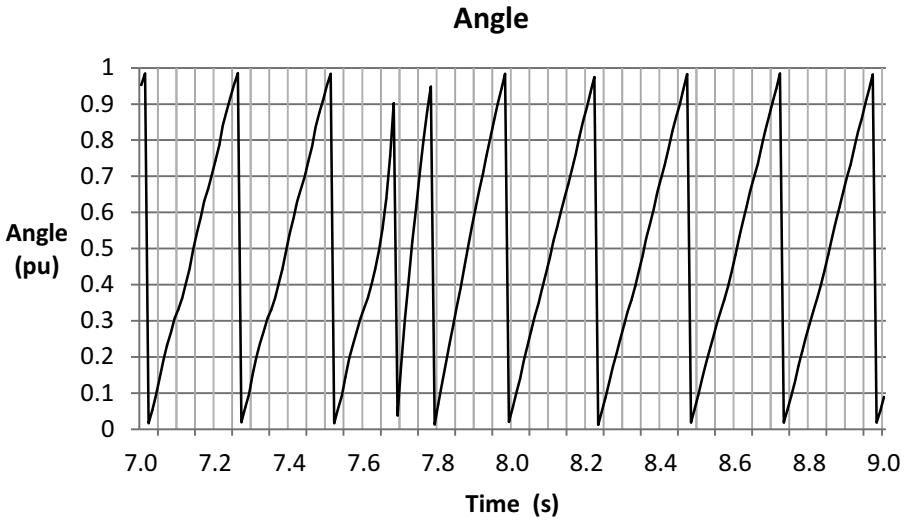


图 **14-7.** 放大的角度图 **-** 移除负载

值得一提的是，显示速度变化的目的不是为了说明速度控制器的作用。 实际上，速度控制器与 FAST 估算器毫无关系。图 14-8 说明即便在转矩命令使电机停转一小会时，估算器是如何追踪电机转速的。

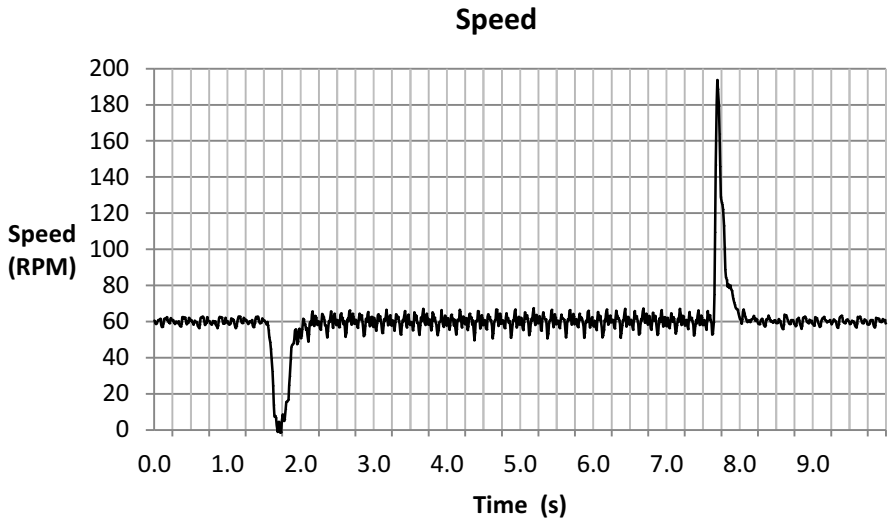


图 **14-8.** 速度图

图 14-9 显示了 FAST 产生的转矩信号。 转矩信号有助于了解电机轴的瞬时转矩，以及在不使用转矩传感器的情况下计算电机负载。 甚至在控制阶跃的时候，该高带宽信号也能显示转矩的轨迹。

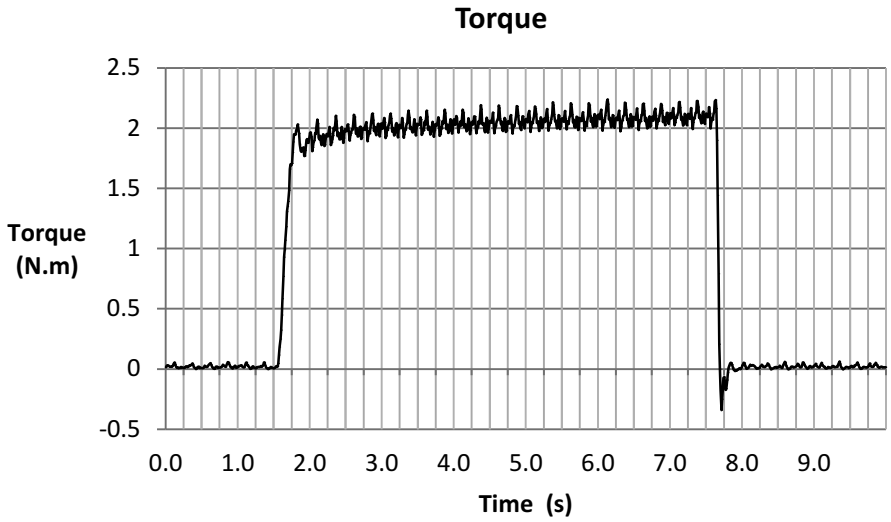


图 **14-9.** 转矩图

同时，我们还绘制了 Iq 电流波形图来显示当转矩阶跃被控制时 FAST 允许的磁场定向控制性能。 如 Iq 电流图所示 (图 14-10)，阶跃负载施加到转轴时，在示例中可以看出相应的电流需量呈阶跃变化。 估算器的角度追踪功能在 Iq 控制器中实现此阶跃响应。 您可能也注意到，转矩曲线不像电流曲线那样平。 这是因为之前的磁通量图中的磁链变化，这个变化则可能是由于软件电机模型与实际模型不匹配所致。

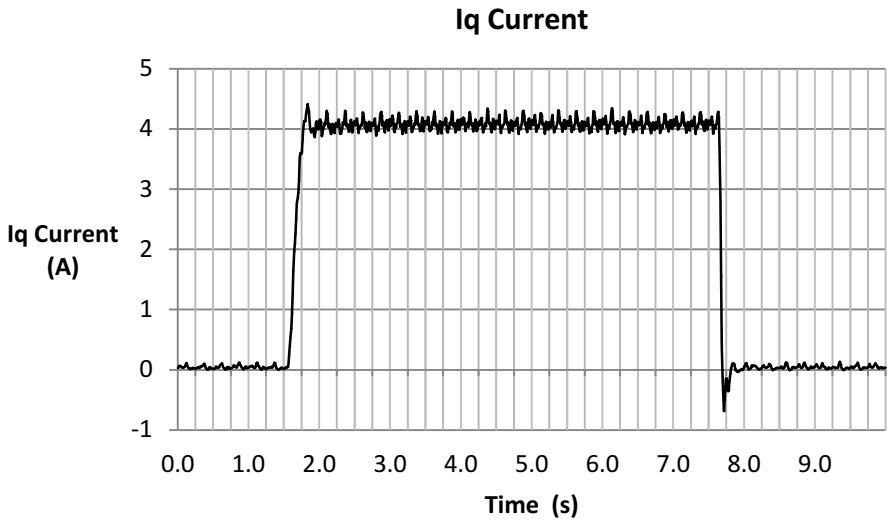


图 **14-10. Iq** 电流图

在之前的示例中，可以看出当负载大幅度变化时，电机速度可能降为零。 此响应可以通过速度控制器循环本身来改善，但是测试的目的是说明 FAST 提供的变量如何是不变的和有效的，甚至当执行 100% 阶跃的负载命令时也如此。

14.1.2.2 2Hz，无负载至满载瞬态

图 14-11 显示了此条件下的电流波形：

- 测力计 = 1.9N·m（满载）
- 速度控制器 = 2 Hz (30RPM ± 3RPM)

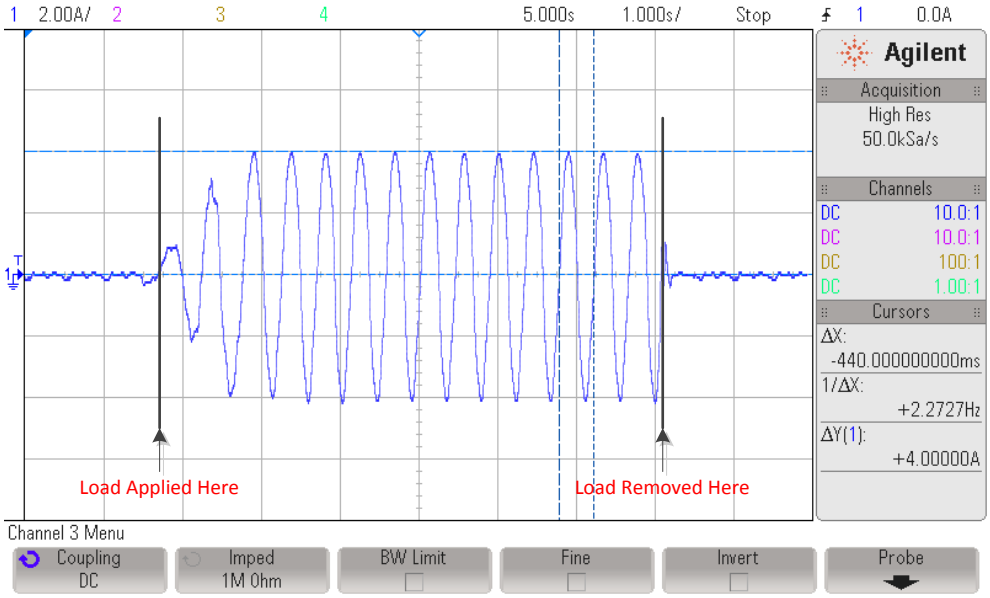


图 **14-11. 2Hz**，无负载至满载瞬态图

将 1.9Nm 的电机额定转矩瞬间施加到电机轴，产生 4A 的电流。示波器显示的电频率为 2.2Hz，大约比参考控制速度高出 3RPM。 对于一个 4 对电极的电机，此频率一旦稳定后，将得到一个 30 ± 3RPM 的转速。

使用滞后测力计将遇到转矩生成问题，并且在测力计转轴中出现的启动转矩产生了高于指令转矩的瞬时转矩，从而时常导致电机停转。 这是我们为何相较之前的示例，以较慢的速率来增加测力计转矩命令的主要原因，目的是为了避免测力计产生大于致使电机暂时停转的指令转矩。

图 14-12、图 14-13、图 14-14 和图 14-15 显示 FAST 算法的表现。 FAST 代表磁通量、角度、速度和转矩，这就是转矩阶跃命令影响那些变量的方法。 第一个变量是电机的磁链。

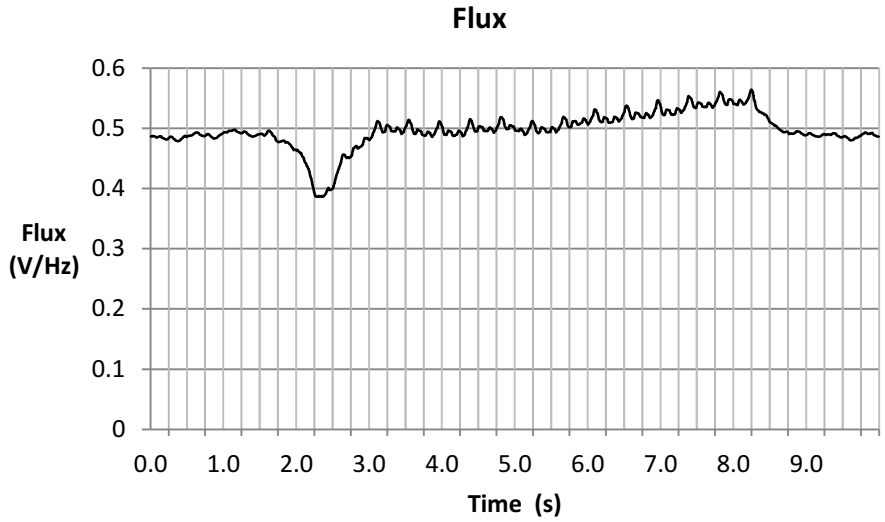


图 **14-12.** 磁通量图

图 14-13、图 14-14 和图 14-15 显示了由 FAST 提供的磁通角。 如之前的测试所见，角度随着电机负载的增加和减少而变化。

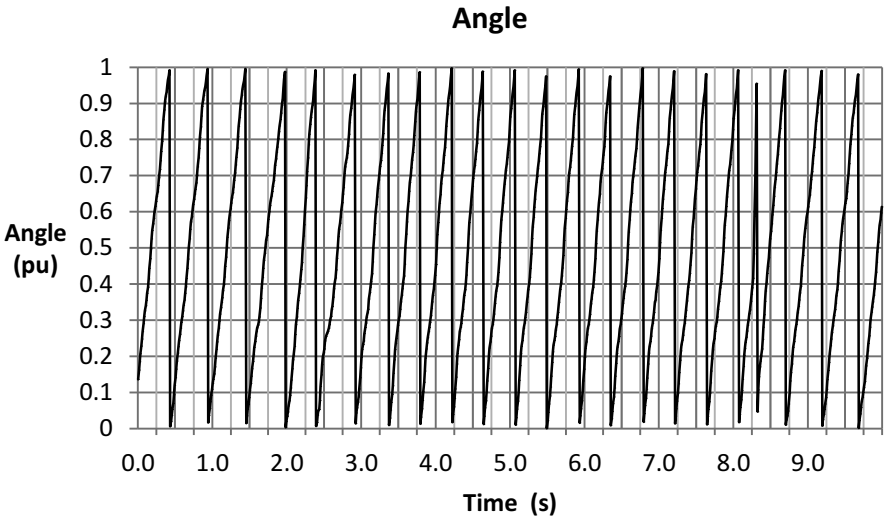


图 **14-13.** 角度图

放大角度图，可以看出电机加载和负载移除时的角度瞬态情况。 当运行速度降低时，结合了滞后测力计的转矩脉动的信号质量，使得角度看起来不像是完好的锯齿波。 即使这样，角度信息还是能为 2 Hz 速度满载瞬态的完全 FOC 控制提供了足够准确的信息。

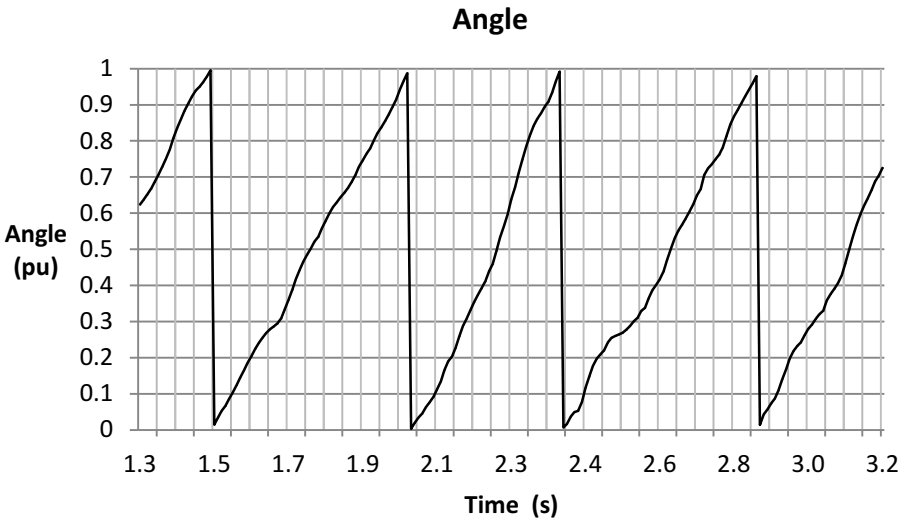


图 **14-14.** 放大的角度图 **-** 电机负载增加

若在负载从电机轴上移除时放大，可以看到瞬时角度轨迹。

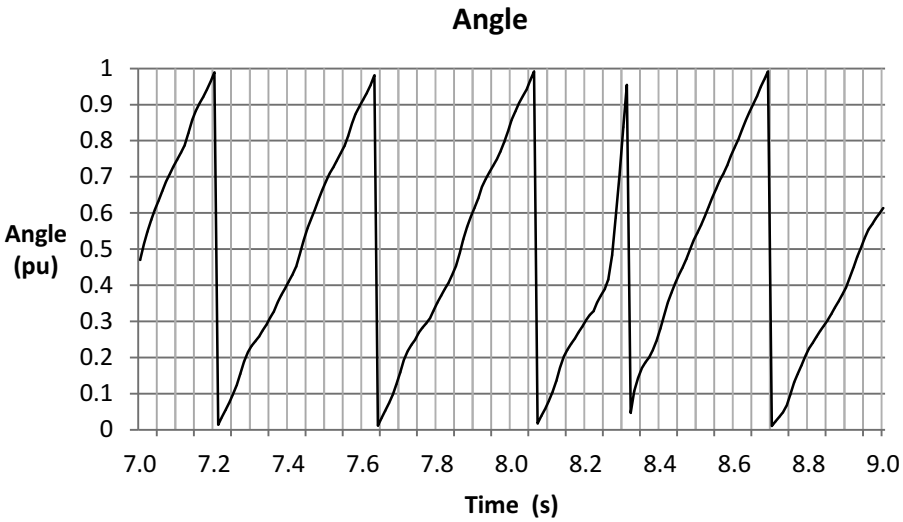


图 **14-15.** 放大的角度图 **-** 电机负载减少

速度图如图 14-16 所示。 目标速度为 30RPM，并且可以看出，与 60RPM 相比，估算速度有较多的波纹。 这是因为滞后测力计中出现了脉动转矩，并且估算速度的输出是瞬时的，而不是每个电周期输出。 因此任何角度斜坡上的失真都会反映到速度示波器上。

***FAST** 变量总是能支持 **FOC** 系统在低速下应用满转矩，即使存在 **100%** 阶跃负载的情况也如此。*

同时，当通过关闭测力计控制器完全移除负载时，即便存在很快的加速度，速度估算也与实际速度相符，如图 14-11 所示。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

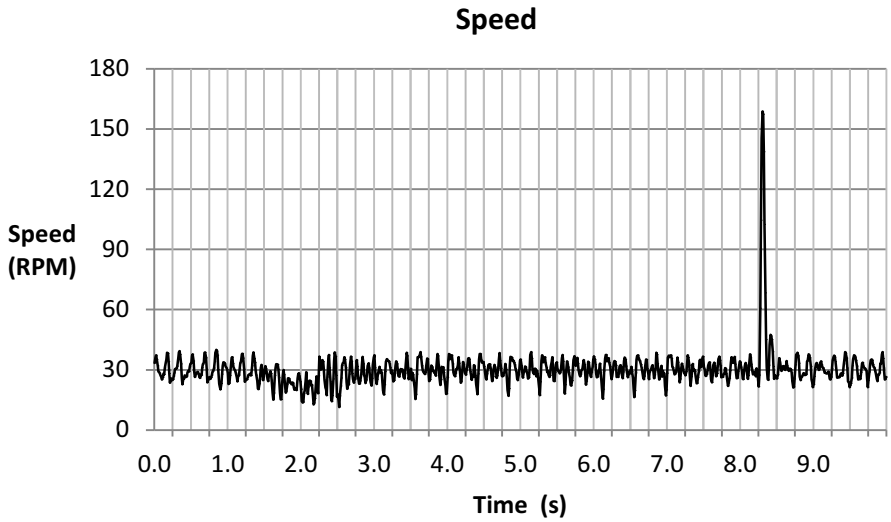


图 **14-16.** 速度图

转矩信号如图 14-17 所示。 由于估算器的频率低，而且在低速运行时，滞后测力计中出现转矩脉动，从而出现振荡。

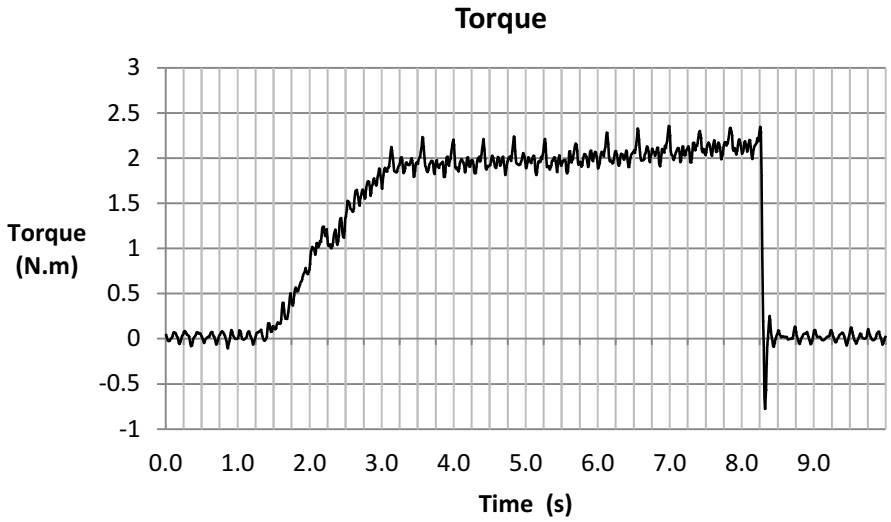


图 **14-17.** 转矩图

从图 14-18 可知，电流控制器跟随指令转矩曲线变化，将 Iq 电流变为额定的 4A。

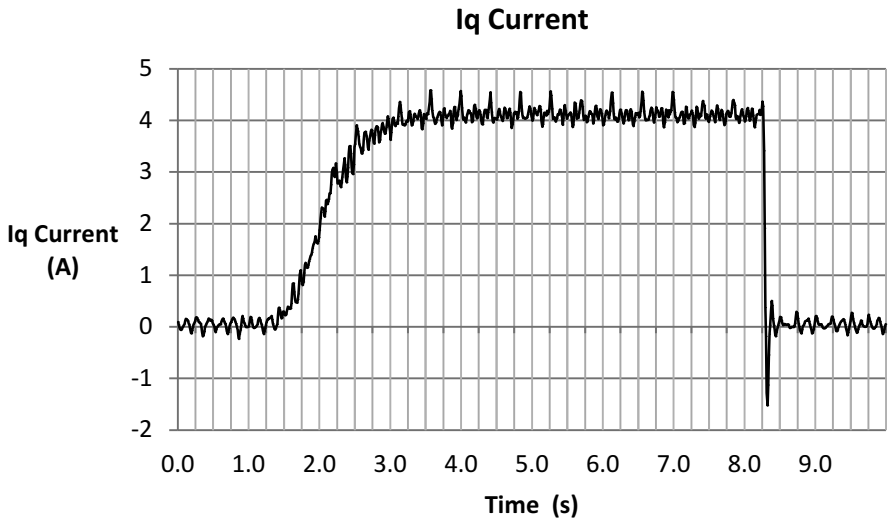


图 **14-18. Iq** 电流图

14.2 满载换向

为进行换向，以任何加速度从高速或低速正向转为高速或低速反向，非常有必要关注我们在转矩瞬态响应示例中考虑的那些要点。

14.2.1 满载低速运行时的换向考量

在这种模式下运行时需要考虑以下事项：

- 启用偏移重校准；在节 14.1.1.1中说明。
- 启用定子 Rs 重校准；在节 14.1.1.2中说明。
- 禁用强制角；在节 14.1.1.3中说明。
- 调整速度控制器以避免电机停转；在节 14.1.1.4中说明。
- 调整电压反馈电路；在节 14.1.1.5中说明。

14.2.2 满载低速运行时的换向示例

在考虑了以上要求后，让我们来看看几个待测电机使用测力计时的几个换向响应示例。

14.2.2.1 满载情况下从 -4 至 +4Hz

图 14-19 显示了此条件下的电流波形。 请注意电流是如何改变相位的，这表示方向发生改变。

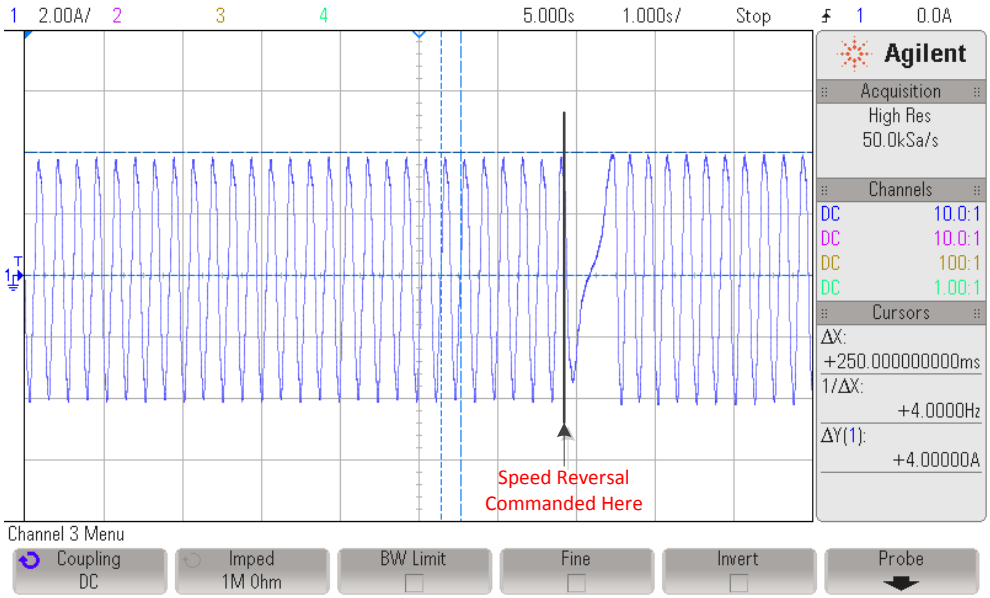


图 **14-19.** 满载情况下从 **-4** 至 **+4Hz** 图

从磁通量估算可知，在方向改变时出现了瞬变，尽管在几秒的时间后便稳定下来。 从图 14-20 可见，磁通量不断升高。 这可能表示软件的电机模型与实际系统存在轻微的不匹配。 磁通量的误差可能由于软件电机模型与实际电机相比不准确所致，不准确可能是电机过热或者电流和电压的感测容差引起。 如果磁通量持续升高，这可能表示由于电机负载导致电机升温，从而定子电阻收敛为一个新值。 在这种情况下，建议使用 InstaSPIN 的 Rs 在线特性。 有关如何运行 Rs 在线特性的示例，请参见Chapter 15。

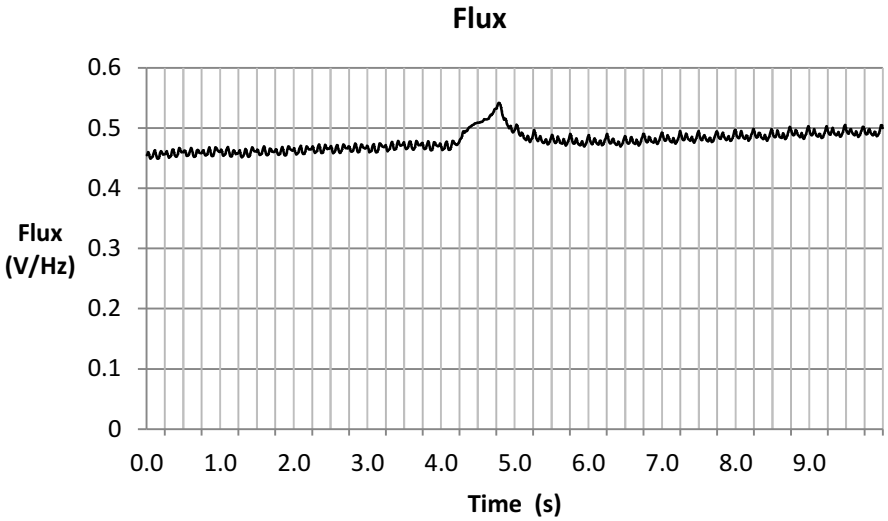


图 **14-20.** 磁通量图

在图 14-21 中，可以看出当速度通过零时，磁通角改变相位。

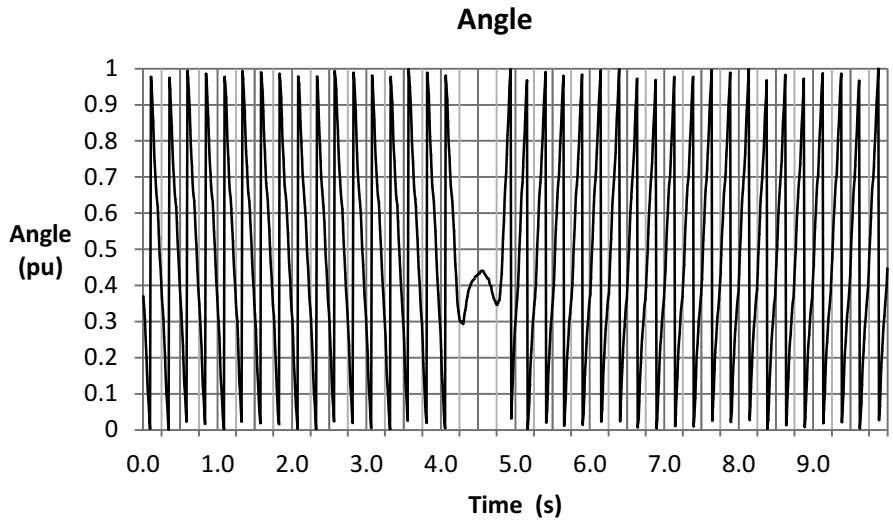


图 **14-21.** 角度图

如果在电机换向时放大（图 14-22），可以更清楚地看出换向实现的过程。 我们可以看出方向实际改变了两次。 这是因为当速度接近零时，算法试图找到角度旋转的方向。

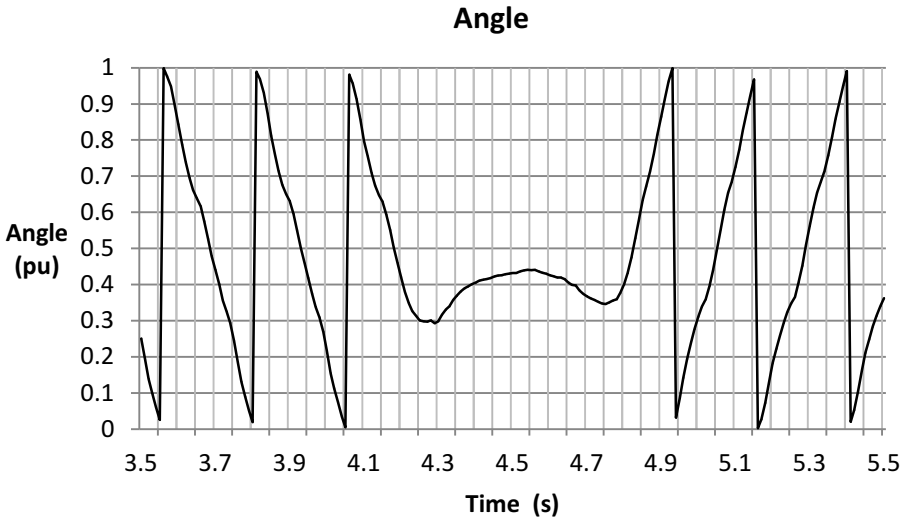


图 **14-22.** 放大的角度图

电机的估算速度也显示了当速度通过零时可能发生方向（符号）错误（图 14-23）。 即速度在 ±10RPM 范围内时，转换成的频率为 ±0.66Hz。

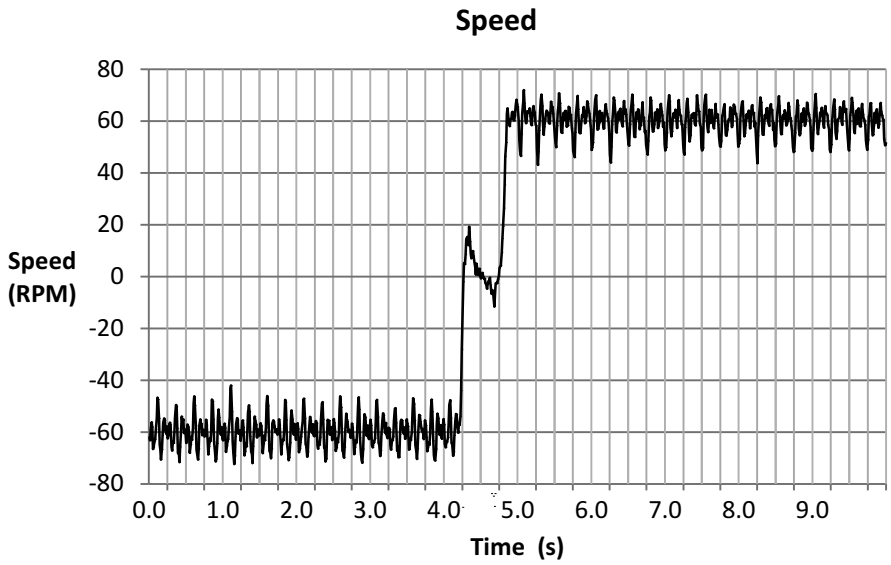


图 **14-23.** 速度图

从下图可知，FAST 的转矩信号从 -1.9Nm 持续增长至 +1.9Nm（图 14-24），并且在正向侧有一个小幅的过冲。 该过冲可能是电机速度通过零时，滞后测力计中的电流积累所致，而这种现象对这类测力计很常见。

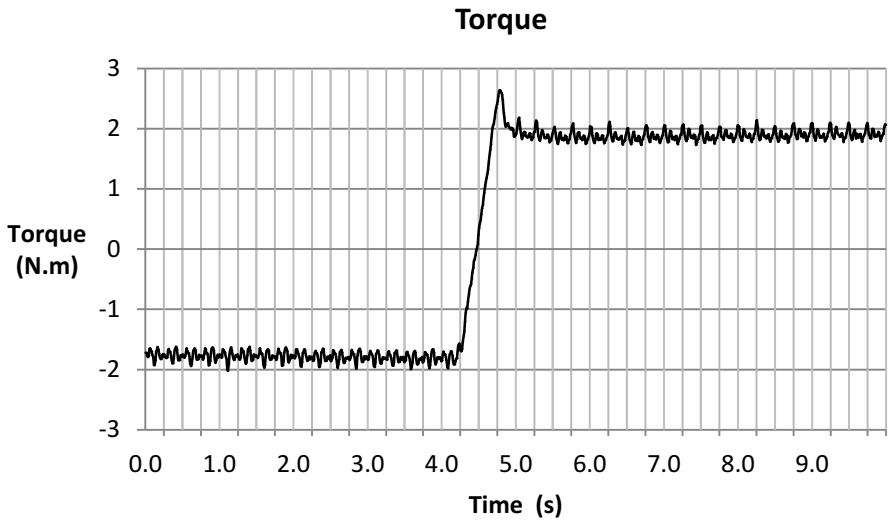


图 **14-24.** 转矩图

图 14-25 显示了正交电流 Iq。 可以看出 Iq 的波形与估算转矩的波形十分相似。 该图中没有太多值得注意的现象，但我们可以看出电流的曲线比转矩更平直。 这是由于在驱动满载一段时间后磁通量估算收敛为一个新值所致。

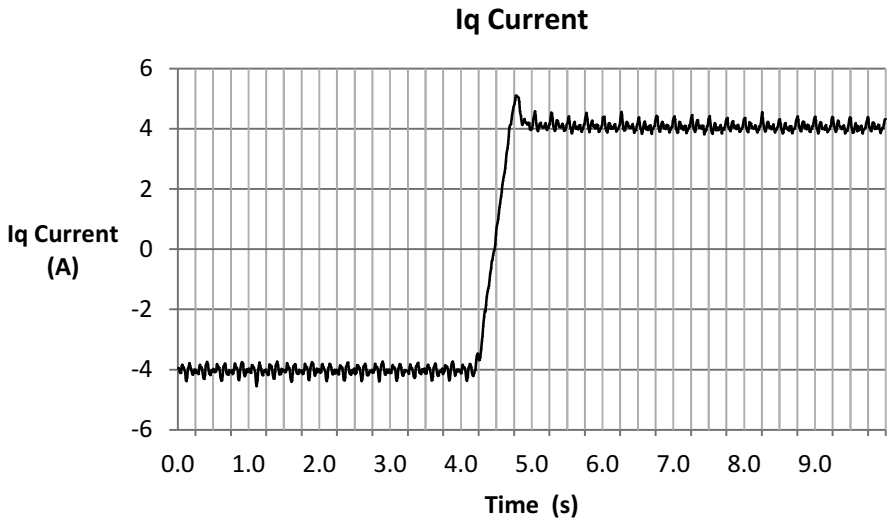


图 **14-25. Iq** 电流图

14.2.2.2 满载情况下从 -2 至 +2 Hz

图 14-26 显示了此条件下的电流波形。 请注意电流是如何改变相位的，这表示方向发生改变。

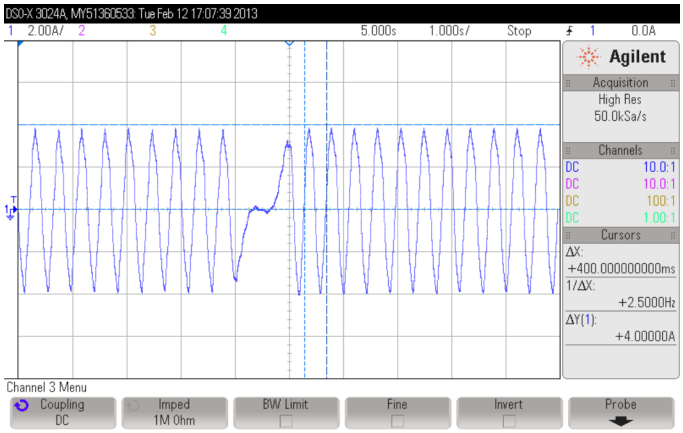


图 **14-26.** 满载情况下从 **-2** 至 **+2 Hz** 图

在本例中，实际速度约为 2.5Hz (37RPM)，所以本例中的总误差大约为 7RPM。 速度计算误差是由于磁通量误差所致。 磁通量的误差可能由于软件电机模型与实际电机相比不准确所致，不准确可能是电机过热或者电流和电压的感测容差引起。

为获得最佳换向性能：启用偏移和 Rs 重校准、禁用强制角、调整速度控制器避免停转以及调整电压反馈电路。

磁通量在此测试中较高，如图 14-27 所示。 这仍然可能是由于电机模型差异导致的，由于电机多次经历负载测试而过热，从而会产生这种差异。

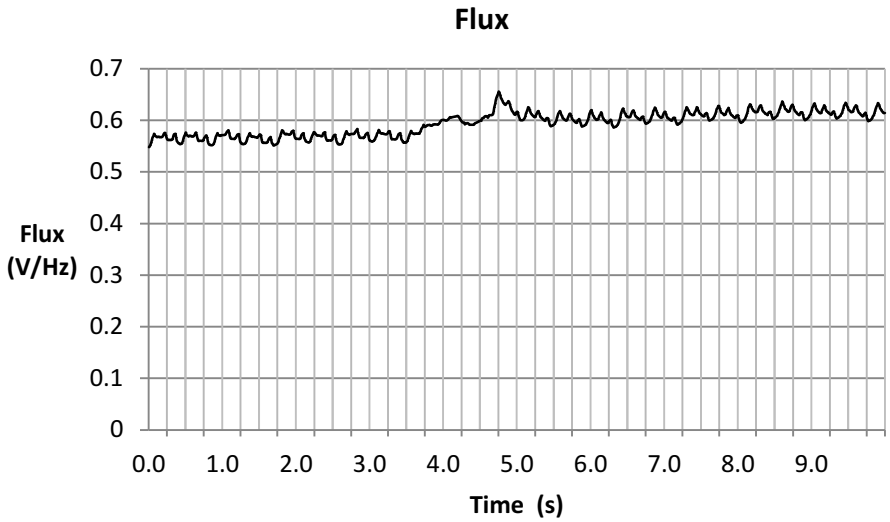


图 **14-27.** 磁通量图

从图 14-28 可看出换向时的角度。

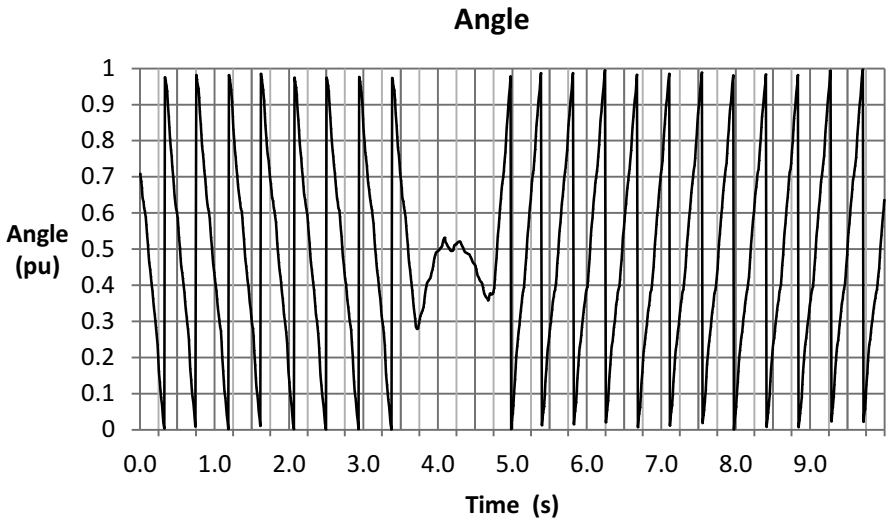


图 **14-28.** 角度图

如果放大角度（图 14-29），我们可以看到在电机换向时，FAST 如何能够在甚至是满载换向的情况下提供稳定的角度。

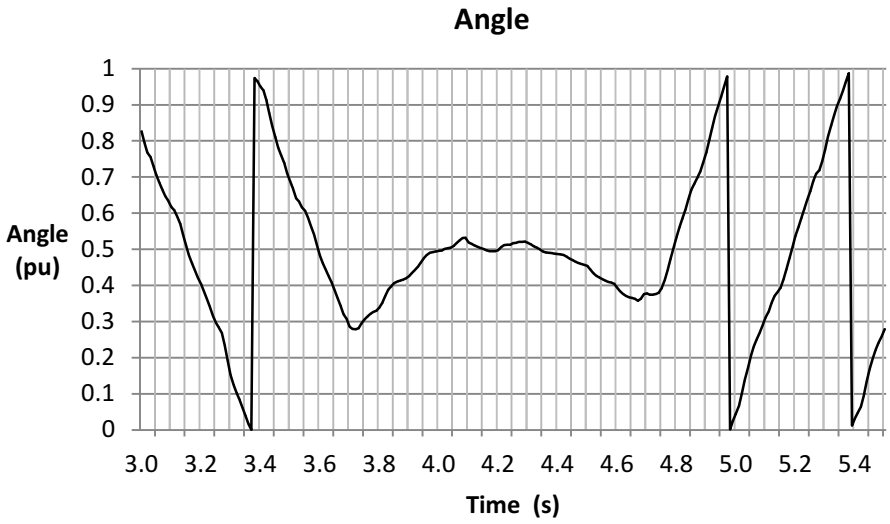


图 **14-29.** 放大的角度图

速度在这时也二次改变了符号（图 14-30），这也是因为在接近零速，尤其是满载时，速度估算器追逐速度符号；而且，我们可以在某个瞬间看出速度是如何在 ±10RPM（或 ±0.66Hz）之间改变符号的。

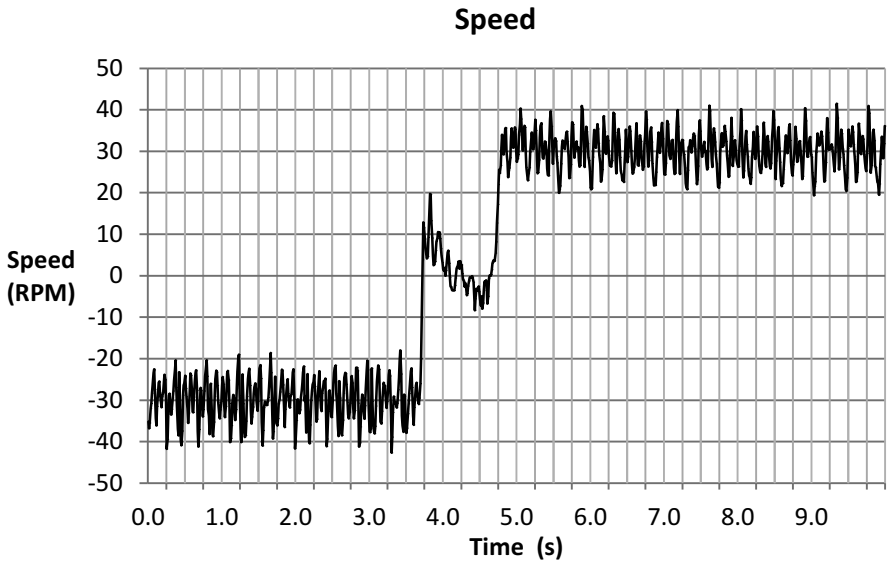


图 **14-30.** 速度图

转矩估算器提供了清晰的过零点和最终值 (图 14-31)。 不过，由于转矩的估算需使用磁通量估算值，因此随磁通量的变化（见本例中的图 14-26）出现一个小的偏量。

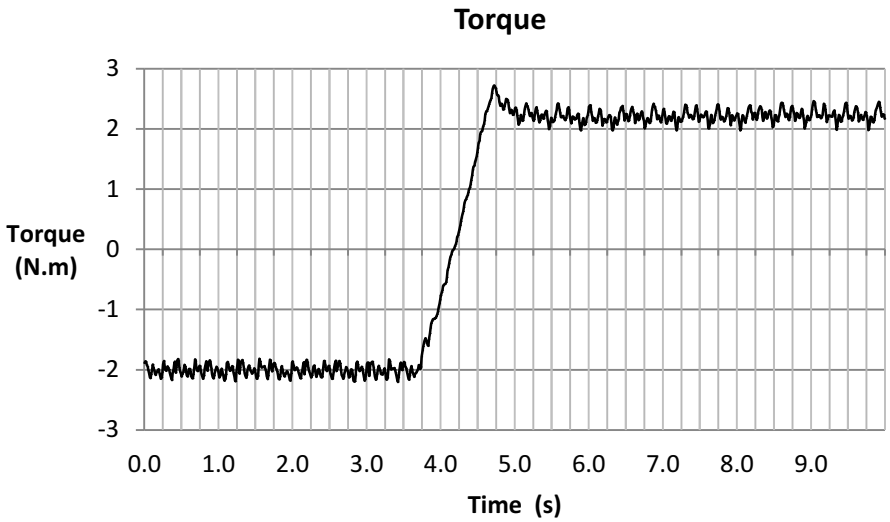


图 **14-31.** 转矩图

另外还在 图 14-32 显示了电流 Iq。

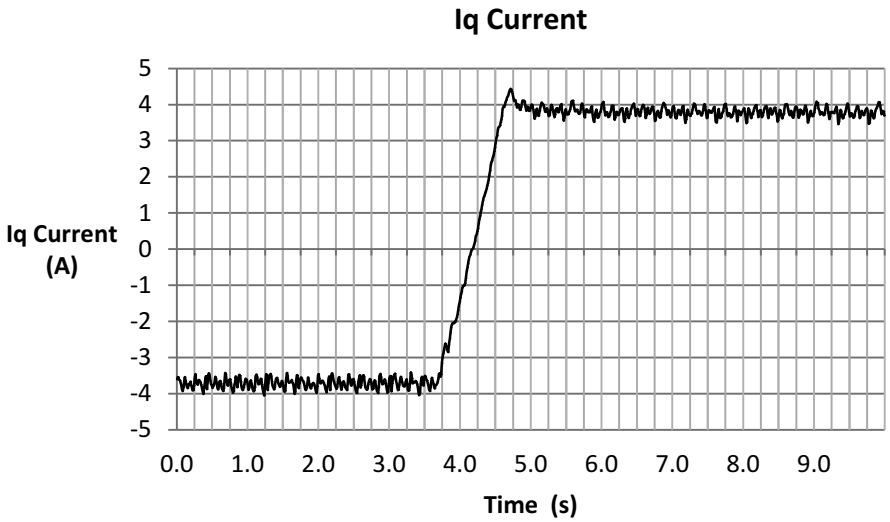


图 **14-32. Iq** 电流图

14.3 满载电机启动

在本小节中，我们讨论在使用 InstaSPIN 和 FAST 算法的情况下，允许满载启动所需考虑的事项。 在介绍相关考虑事项后，我们将讨论几个实例。

14.3.1 满载电机启动的考量

在之前的章节中讨论的考虑事项仍然适用于此运行模式：

- 启用偏移重校准；在节 14.1.1.1中说明。
- 启用定子 Rs 重校准；在节 14.1.1.2中说明。
- 启用强制角；在节 14.3.1.1中说明。
- 调整速度控制器以避免电机停转；在节 14.1.1.4中说明。

- 调整电压反馈电路；在节 14.1.1.5中说明。

14.3.1.1 启用强制角

为在满载下从静止启动电机，估算器需要一个初始转角以允许电机存在些反电动势，如图 14-33 所示。 通常，FAST 需要不到 1 个电周期便能锁定真实角度。 为启用转角，用户必须启用 InstaSPIN 的强制角特性。 一旦电机已经启动，建议禁用强制角以便电机可以实现换向。 不过，如果电机在任何低速运行或者换向测试期间停转了几秒，则建议重新启用强制角模式以摆脱电机停转状态。

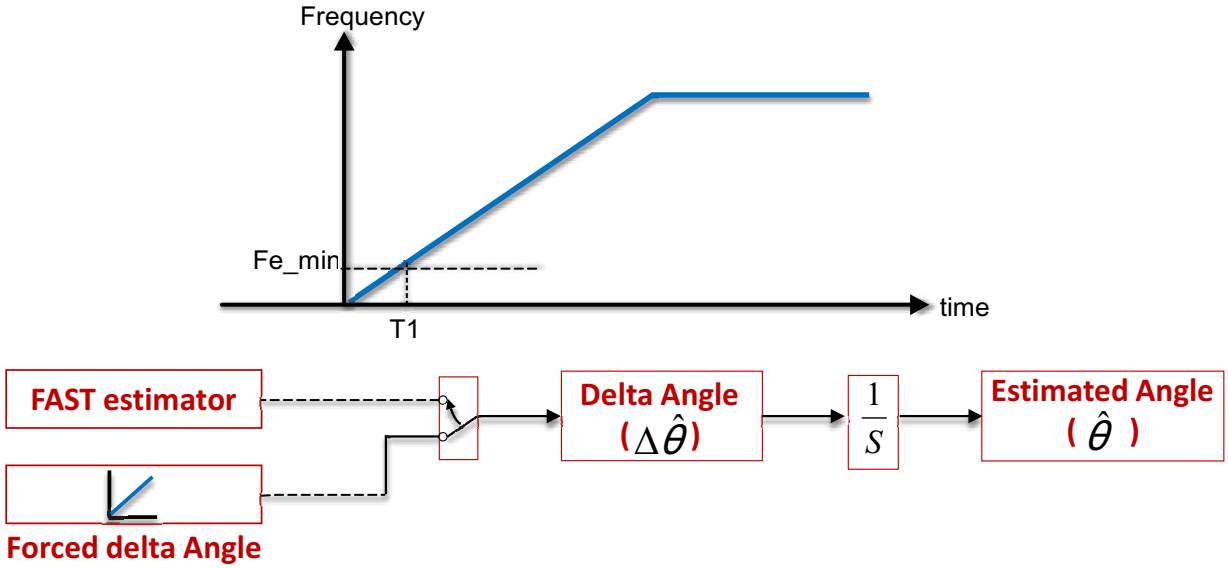


图 **14-33.** 启用强制角

- 强制角用于在转子速度较低时，强制估算器的 d 轴角度。 在零至 Fe_min 频率的范围将使用默认为 1Hz 的设置激活强制角。
- 通过使用 FAST 估算器输出的角度信息，在时间 T1 过后启动闭环向量控制。
- FAST 算法在一个电频率周期内收敛到转子角度。 FAST 算法对任何速度（即使是零速）都是稳定的。
- 在改变速度方向时，为实现平滑换向，请禁用强制角。

以下示例代码在启动控制器之前启用强制角。

```
// enable the forced angle
EST_setFlag_enableForceAngle(obj->estHandle,TRUE);
```

*为实现最佳启动性能 **FAST** 配置有所不同：启用强制角。*

14.3.2 满载电机启动示例

在考虑上述事宜后，让我们来看看几个待测电机使用测力计时的几个满载启动示例。

14.3.2.1 满载情况下从静止至 4Hz

图 14-34 显示了此条件下的电流波形。 请注意电流是如何从 0A 一直升高到 6A 的，6A 被设置为速度控制器的最大输出或者是 Iq 电流控制器的最大参考值（请参见Chapter 5的最大电流）。 另外还可以看出在强制角的一个周期内，电机电流降回 4A，即生成最大转矩的额定电流。 请记住，本例中电机的最大电流是 6A，而产生额定转矩的额定电流为 4A。

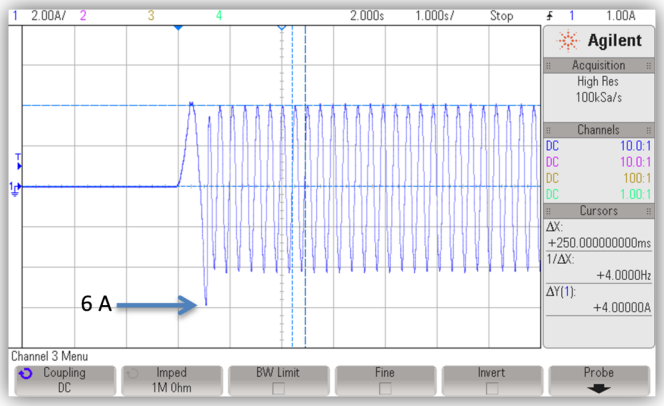


图 **14-34.** 满载情况下从静止至 **4Hz** 图

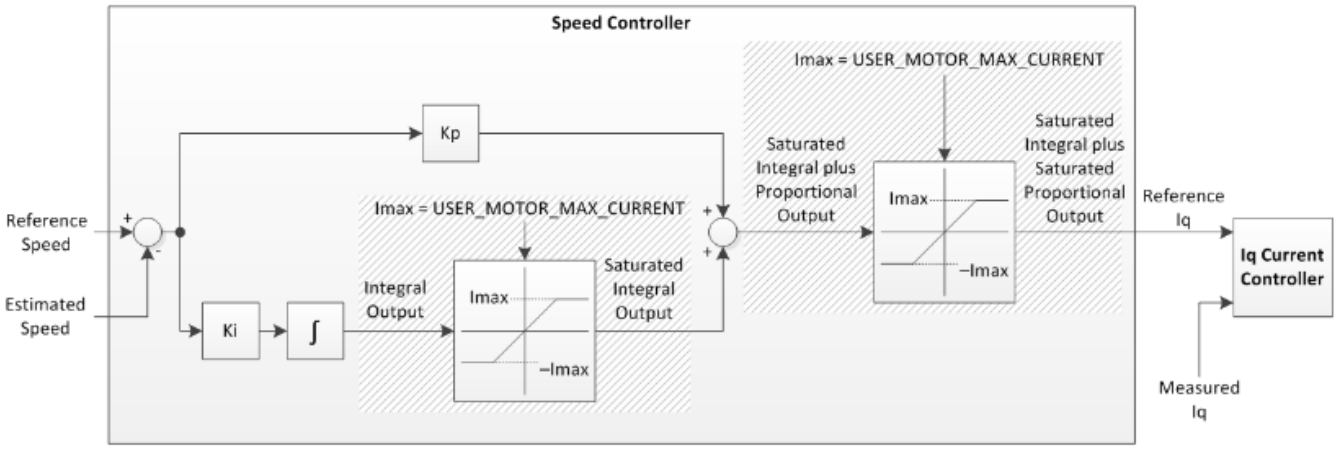


图 **14-35.** 速度控制器周期

图 14-36 为磁通量图，从中可以看出磁通量如何从瞬态转为稳定的情况。

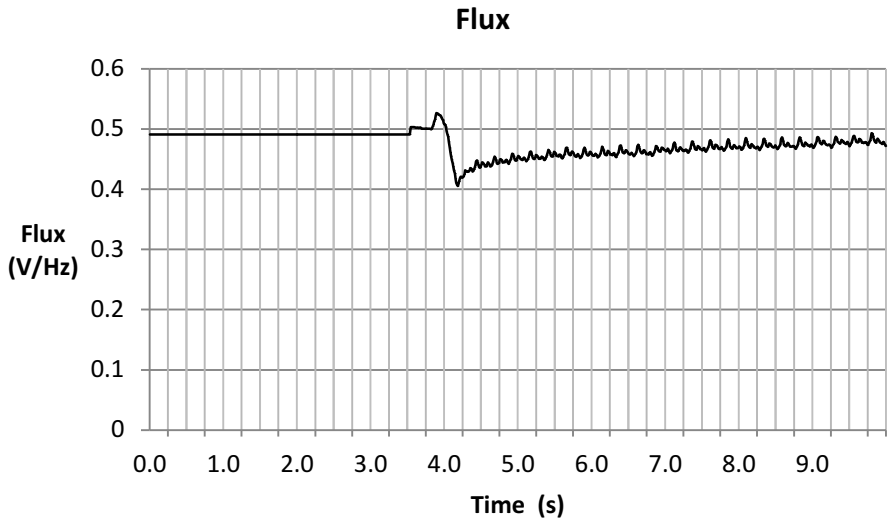


图 **14-36.** 磁通量图

从角度图可知，最初在不到一个周期的时间便实现了强制角（图 14-37）。

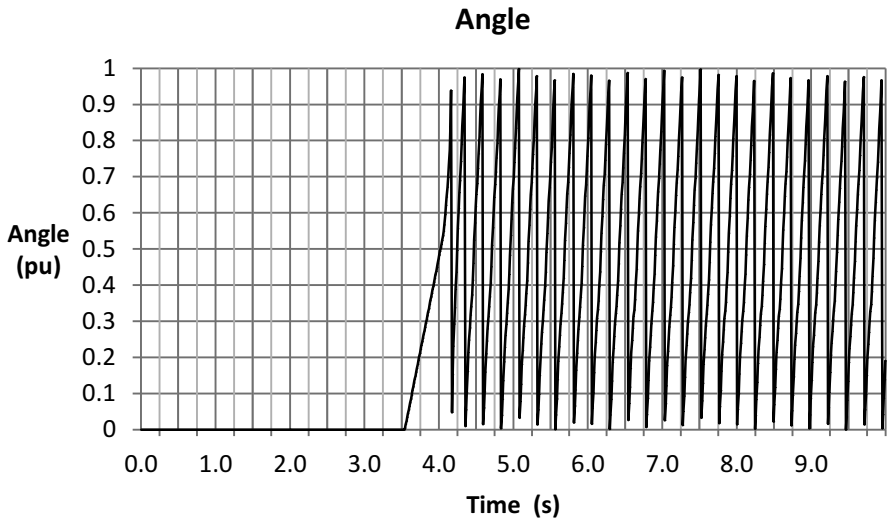


图 **14-37.** 角度图

实际上，如果放大角度图，就可以知道角度被强制了多少个周期或者到底一个周期的百分之多少。 在本例中，角度强制只使用了半个电周期。

通常，需要不到 1 个电周期便能锁住角度。

这可通过图 14-38 计算得出，从图中得到每秒 1 个电周期的斜率，因为角度从 0 增加到 0.5 用了 0.5 秒。

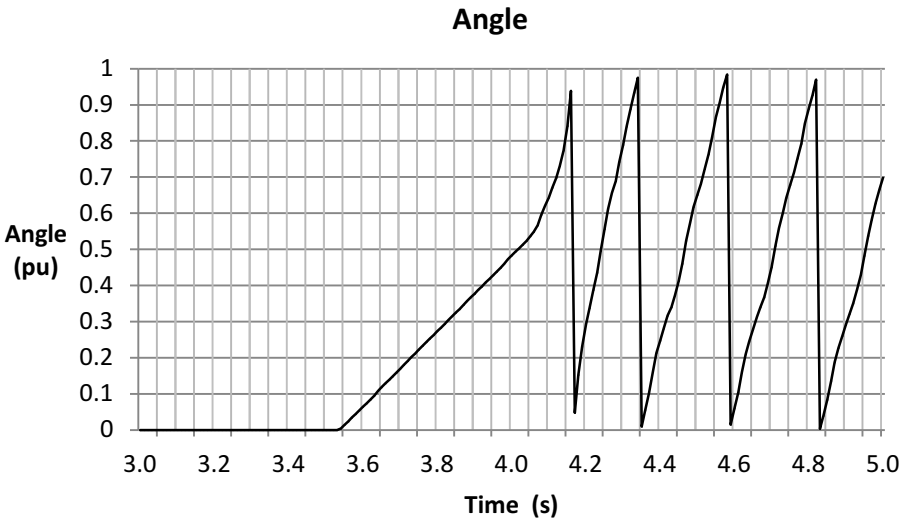


图 **14-38.** 放大的角度图

如果需要其它强制角频率，用户可以通过修改 user.h 文件中的下列值来更改频率：

```
//! \brief Defines the forced angle frequency, Hz
#define USER_FORCE_ANGLE_FREQ_Hz (1.0)
```

为满足启动时间要求，可能需要通过提供更快的强制角更改此频率。 不过，更快的强制角需要在开环中更快的速度，但不可以慢到足以在开环中转动负载。

从本例中的估算速度可以分析出很多信息（图 14-39）。 首先，可以看出最初并没有与强制角对准，这也是速度变为小于零并持续了一段时间的原因。 同时可知，等到电机对准之后，电机加速至指令速度的电流大于所需电流。 这便是速度过冲很多的原因。 通常，估算器需要最多一个电周期便能赶上转子的磁通角。 因此在相反方向上通常需要最多一个电周期。

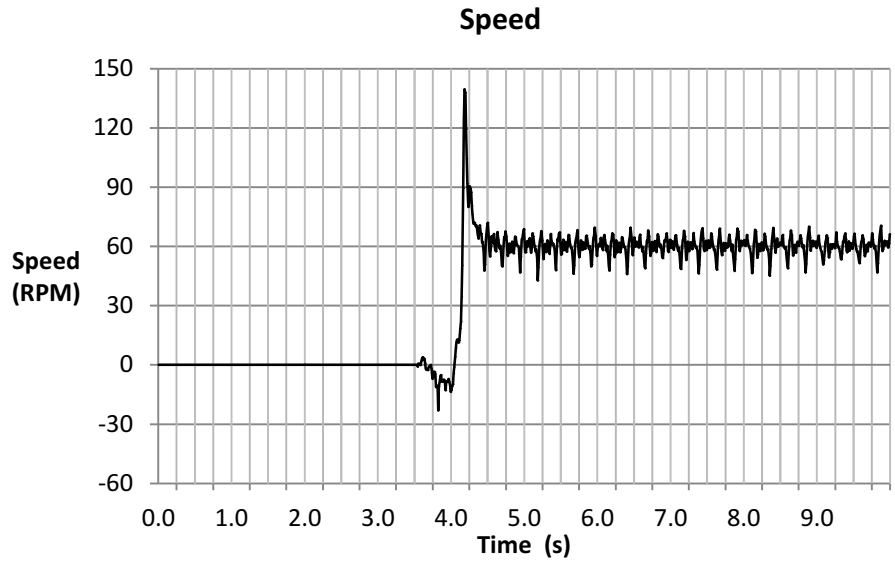


图 **14-39.** 速度图

由于估算的磁通量存在瞬态导致估算扭矩过冲，如图 14-40 所示。 不过，在估算角度与电机角度对准后，该瞬态很快减少，估算转矩趋于稳定且与测力计控制器显示的转矩相符。

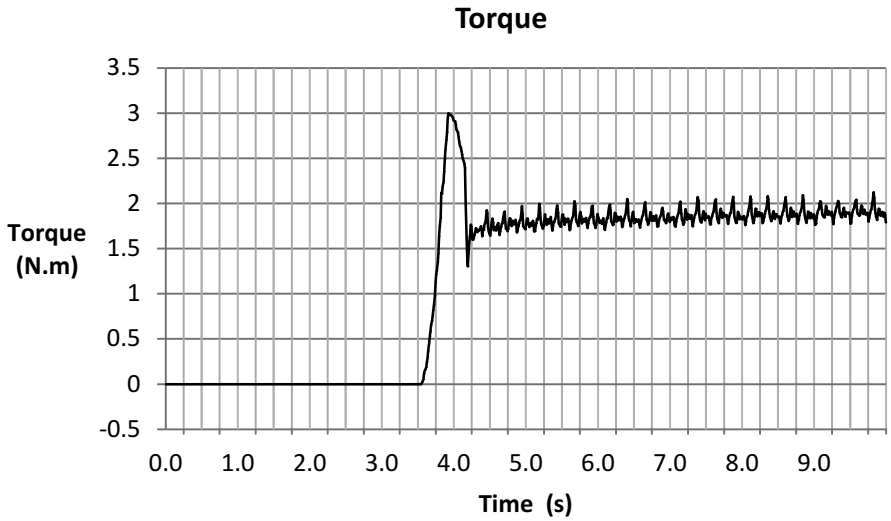


图 **14-40.** 转矩图

本例中的最后一张图是 Iq 电流（图 14-41）。 已知 Iq 参考值的限制为 6A，这是该电机的最大安全电流。从图中可以看出在启动时该电流如何达到最大值，然后在角度对准后，电流降回到 4A 的额定电流。

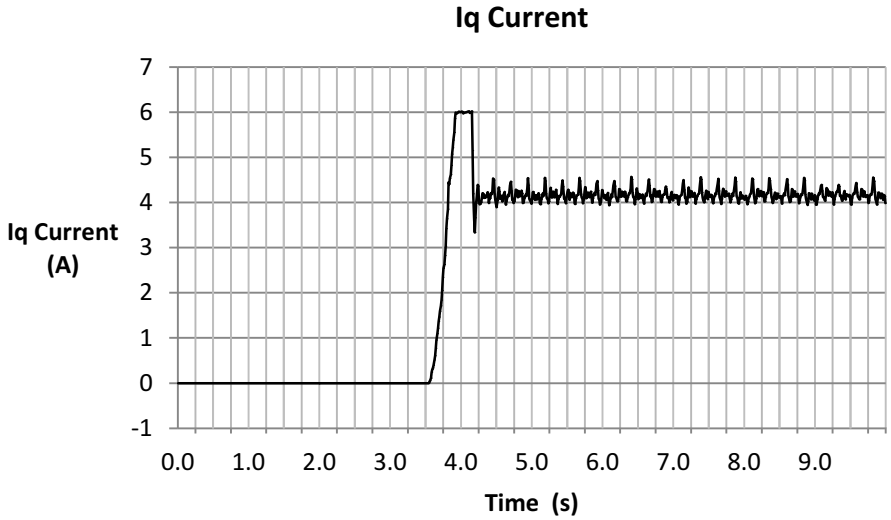


图 **14-41. Iq** 电流图

14.3.2.2 满载情况下从静止至 2Hz

在图 14-42 中，电流再次达到该电机的最大安全电流 6A，但这次用时更长。 不过，应用强制角的时间仍然在一个电周期之内。 还可以看出我们在接近估算器的极限，由于测量频率显示为 2.38Hz，此时的指令速度为 2Hz，实际相当于 6RPM 的差异。

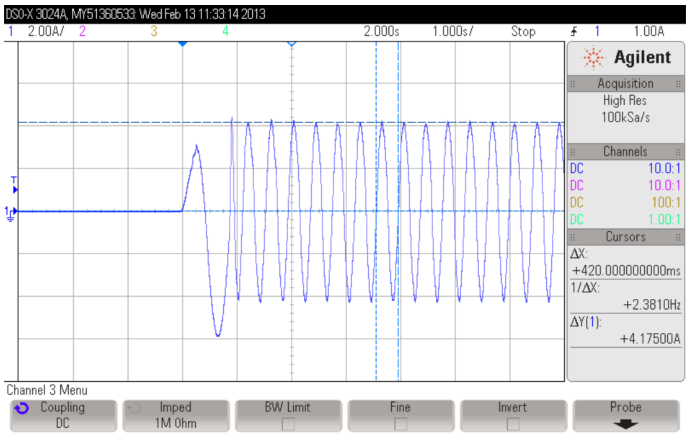


图 **14-42.** 满载情况下从静止至 **2Hz** 图

估算的磁通量显示了一些误差和一次瞬态（图 14-43），尽管这仍然与我们通过其它以 2Hz 运行的测试获得的结果相当。 磁通量不同于 0.5v/Hz 的额定磁通量，导致实际的电频率与估算的电频率有差异。

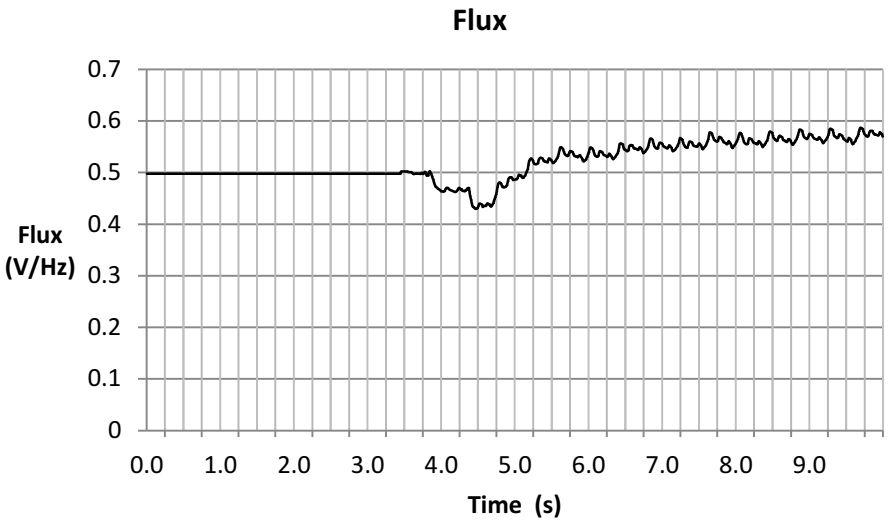


图 **14-43.** 磁通量图

角度估算说明了强制角的时间还不到一个电周期（图 14-44）。

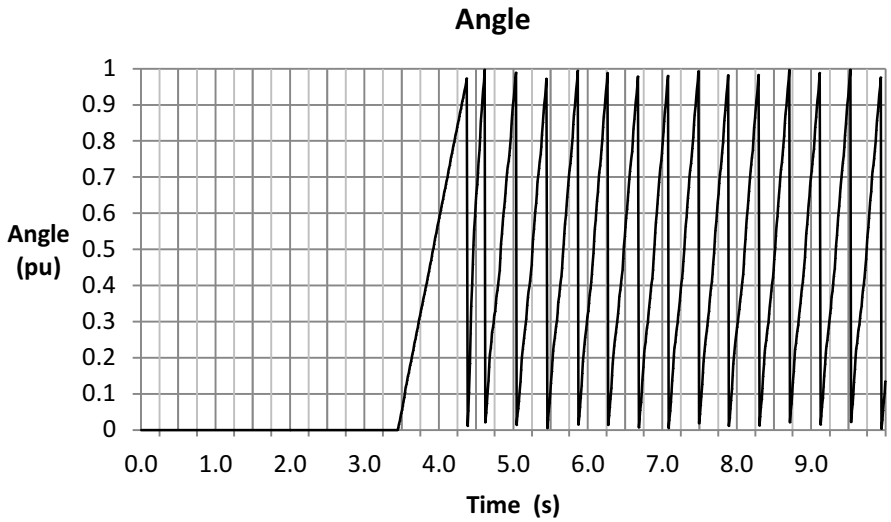


图 **14-44.** 角度图

放大这个角度，可以看出频率改变之前的第一个周期小于一秒（图 14-45）。 一旦电机角度和估算角度一致，生成的转矩将更高，导致电机加速超过了目标速度 30RPM。 这可以解释为何紧随强制角周期的第一个周期为何比后面的周期频率更高。

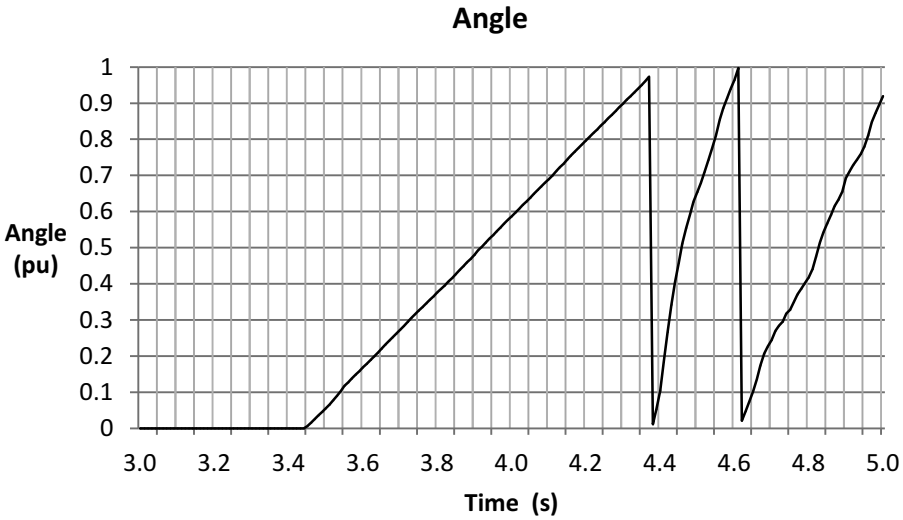


图 **14-45.** 放大的角度图

从图 14-46 中可以看出速度过冲。 还可以看出电机在加速至指令方向前，反转了一小会。

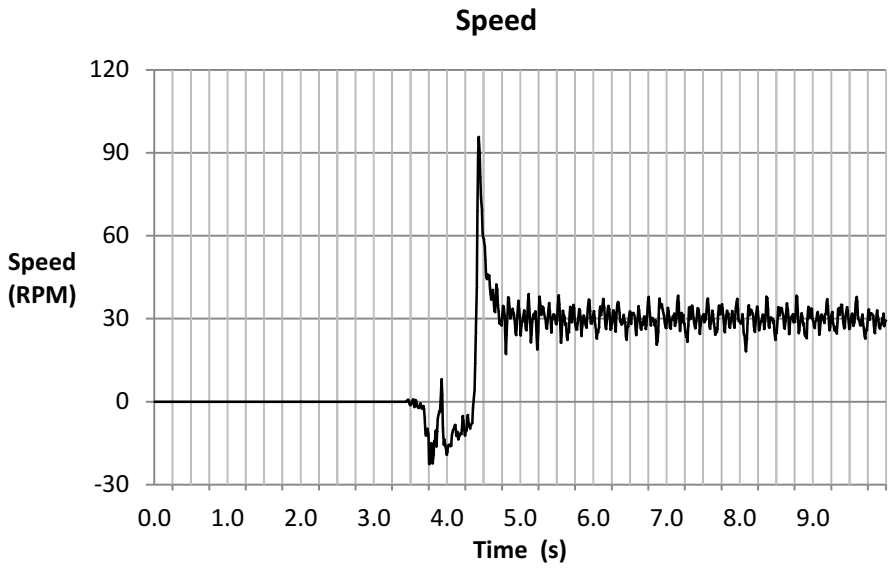


图 **14-46.** 速度图

图 14-47 中的估算转矩情况与上一个示例一致：先是出现由于估算磁通量误差导致的一个瞬态，然后出现一个稳态误差，这可能是由于测量不准和电机升温所致。

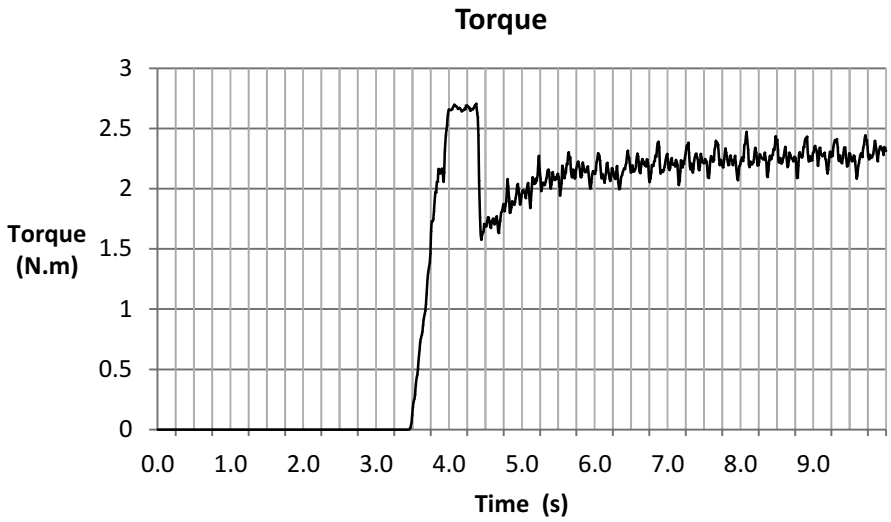


图 **14-47.** 转矩图

本例的最后一张图（图 14-48）显示电流 Iq，图中实际上显示了超过 4A 的转矩产生，生成了比上一个示例更高的转矩。

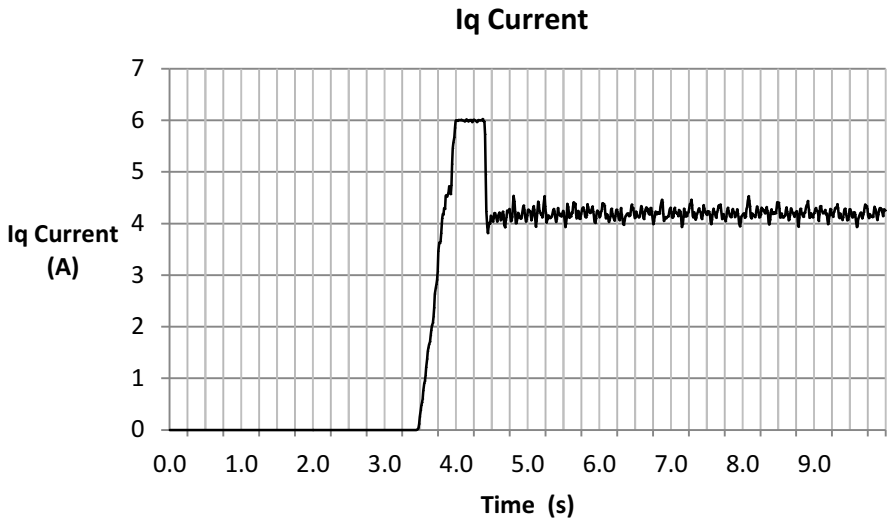


图 **14-48. Iq** 电流图

14.4 满载从静止状态快速加速

本小节讨论使用 FAST 算法满载启动时，实现从静止状态以最快的速度加速到指令速度需要考虑的事项。我们在图 14-4 中也讨论过缩短启动时间问题。 本小节将讨论满载快速加速启动的难点。

无论是使用什么速度控制器和速度控制器增益，为了实现从用户启用系统时尽快使电机闭环运行，都需要考虑以下几点或进行相关配置。 在介绍相关考虑事项后，我们将讨论几个实例。

14.4.1 无需电机对准时满载最快速度启动电机的考量

在之前的章节中讨论的考虑事项仍然适用于此运行模式：

- 加载有效偏移并禁用偏移重校准；在节 14.4.1.1中说明。
- 加载有效 Rs 并禁用 Rs 重校准；在节 14.4.1.2中说明。
- 启用强制角；在节 14.3.1.1中说明。
- 调整速度控制器以避免电机停转；在节 14.1.1.4中说明。
- 调整电压反馈电路；在节 14.1.1.5中说明。

14.4.1.1 加载有效偏移并禁用偏移重校准

通过进行此配置，系统可以避免在用户发出电机启动命令后花时间进行偏移重校准。 不过，由于低速运行和电机启动要求偏移是正确的，用户必许在调用 CTRL_setFlag_enableCtrl(ctrlHandle, TRUE) 函数之前，加载这些预校准的偏移。 下列示例代码加载已知的预校准偏移至 HAL 对象。

```
// disable automatic calculation of bias values
CTRL_setFlag_enableOffset(ctrlHandle,FALSE);

// set the current bias
HAL_setBias(halHandle,HAL_SensorType_Current,0,_IQ(I_A_offset));
HAL_setBias(halHandle,HAL_SensorType_Current,1,_IQ(I_B_offset));
HAL_setBias(halHandle,HAL_SensorType_Current,2,_IQ(I_C_offset));

// set the voltage bias
HAL_setBias(halHandle,HAL_SensorType_Voltage,0,_IQ(V_A_offset));
HAL_setBias(halHandle,HAL_SensorType_Voltage,1,_IQ(V_B_offset));
```

```
HAL_setBias(halHandle,HAL_SensorType_Voltage,2,_IQ(V_C_offset));
```

请注意，I_A_offset、I_B_offset、I_C_offset、V_A_offset、V_B_offset 和 V_C_offset 为之前系统运行的预校准偏移。 重校准启用后，若这些偏移发生更新，可使用以下示例从 HAL 对象获取它们。

```
// enable automatic calculation of bias values
CTRL_setFlag_enableOffset(ctrlHandle,TRUE);

// Return the bias value for currents
I_A_offset = HAL_getBias(halHandle,HAL_SensorType_Current,0);
I_B_offset = HAL_getBias(halHandle,HAL_SensorType_Current,1);
I_C_offset = HAL_getBias(halHandle,HAL_SensorType_Current,2);

// Return the bias value for voltages
V_A_offset = HAL_getBias(halHandle,HAL_SensorType_Voltage,0);
V_B_offset = HAL_getBias(halHandle,HAL_SensorType_Voltage,1);
V_C_offset = HAL_getBias(halHandle,HAL_SensorType_Voltage,2);
```

14.4.1.2 加载有效 Rs 并禁用 Rs 重校准

为了避免花时间重校准电阻值，还务必要确保 user.h 中的电阻值正确并禁用电阻重校准特性。 如下显示了 user.h 中提供的电阻：

```
#define USER_MOTOR_Rs (2.6)
```

以下示例代码将禁用 R_s 重校准：

```
EST_setFlag_enableRsRecalc(obj->estHandle,FALSE);
```

14.4.1.3 无需电机对准时满载最快速度启动电机的示例

图 14-49 显示了在无需对准的情况下执行快速加速时的其中一个相电流。 可以看出，电流在不到一个周期的时间内达到最大值，然后加速到 200RPM 的指令速度参考值。

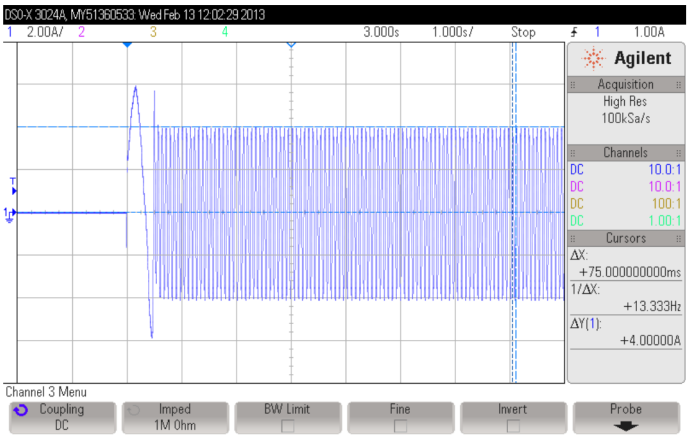


图 **14-49.** 无需对准时的快速加速图

磁通量也存在一个瞬态，该瞬态是在估算角度与实际电机角度没对准时发生的（图 14-50）。在瞬态之后，磁通量稳定在一个相对恒定的值。

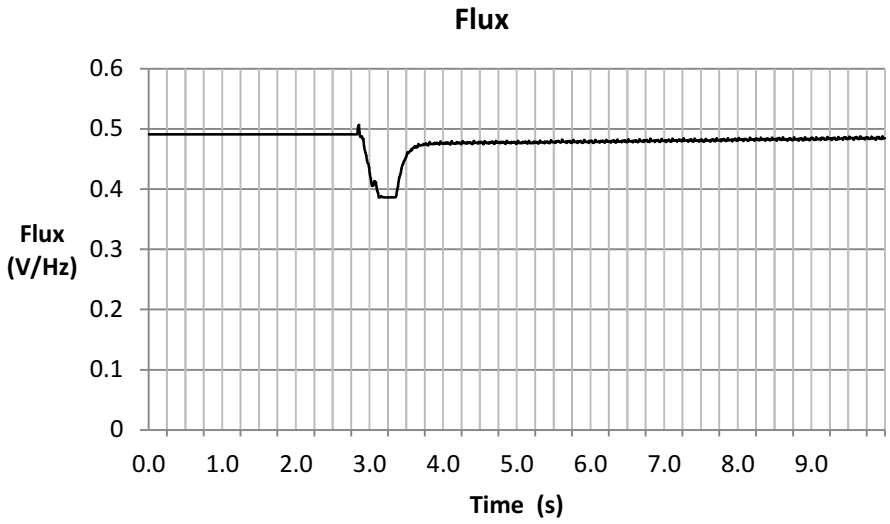


图 **14-50.** 磁通量图

可以看出在第一个周期角度被强制执行（图 14-51），之后由于已经运行在闭环系统中，系统使用估算角度值而非强制角快速改变频率。

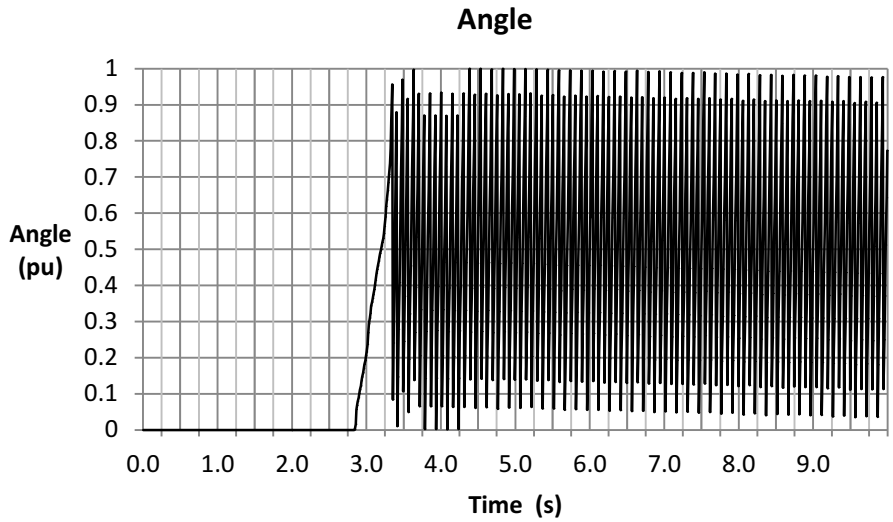


图 **14-51.** 角度图

通过放大角度（图 14-52）可知，强制角持续了不到一个电周期便加速到了指令速度 200RPM。

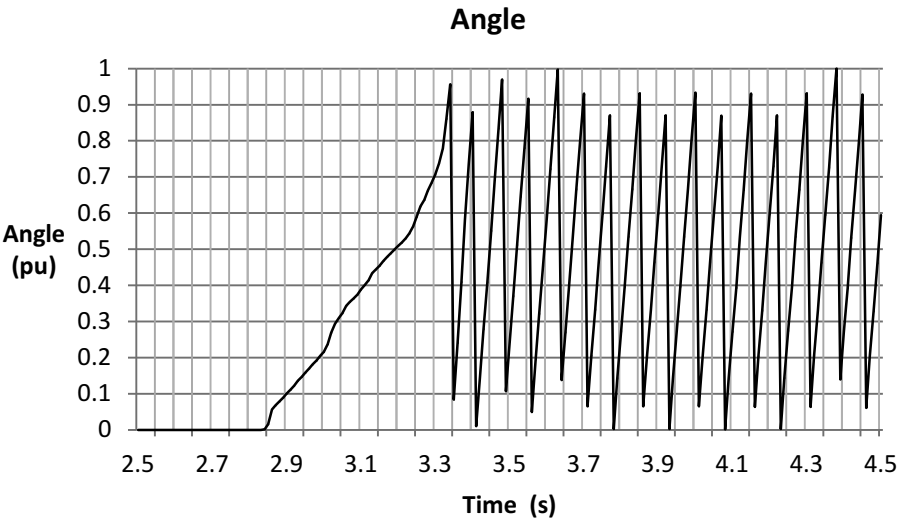


图 **14-52.** 放大的角度图

由于最初未对准，可以看出速度变负数一小段时间（图 14-53），在估算角度与电机角度对准后，速度便很快加速到了指令速度 200RPM。 事实上，由于在角度被强制时，速度控制器的积分部分积累了过多的电流，因此发生了过冲现象。

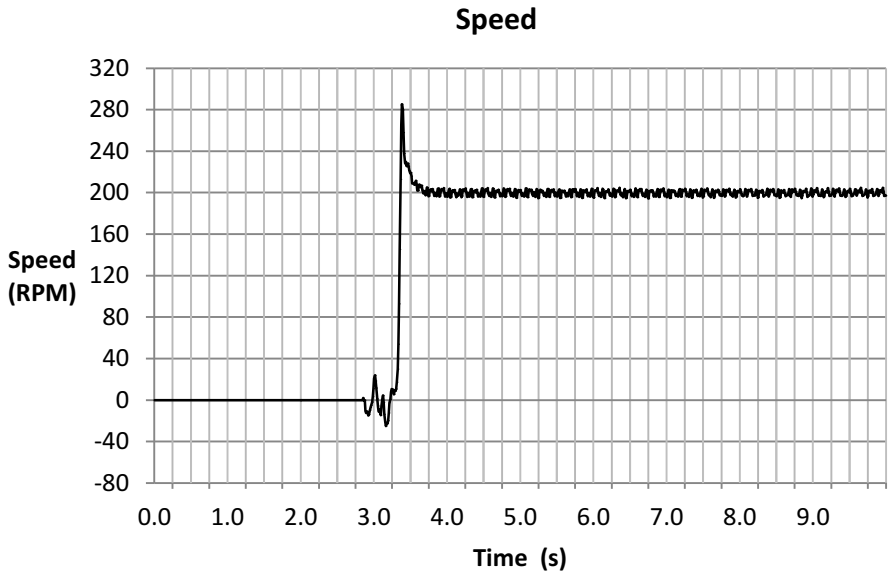


图 **14-53.** 速度图

我们可以看出磁通角误差对转矩信号的影响（图 14-54），以及由于强制角导致的电流过剩情况。 在磁通量瞬态之后，转矩信号变准确并且在速度稳定后变得恒定。

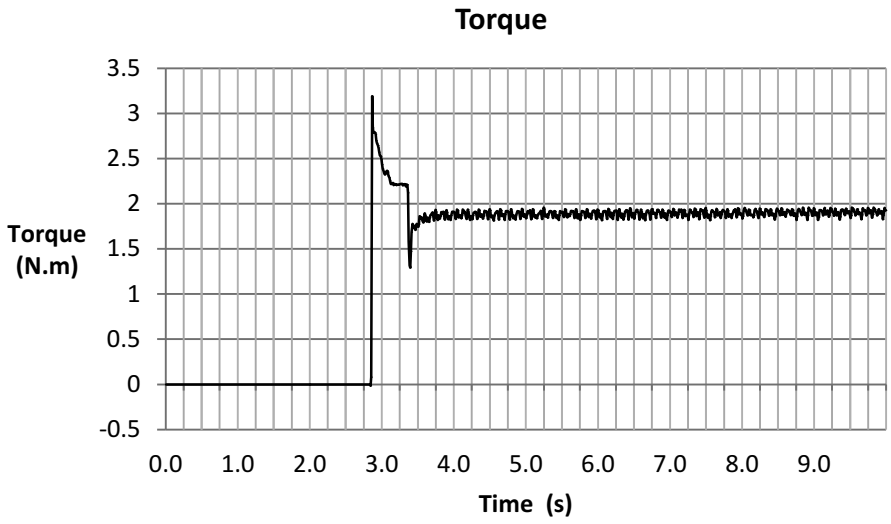


图 **14-54.** 转矩图

速度阶跃命令使得电流 Iq 出现一个大约 0.5A 的过冲，如图 14-55 所示。 该过冲通过开始时的小脉冲反映出来，然后降低到 6A 的电流（我们在 user.h 中配置的最大电机电流）限制内。 在电机加速到指令速度且估算的磁通量稳定后，电流降到 4A 的额定值以产生最大转矩。

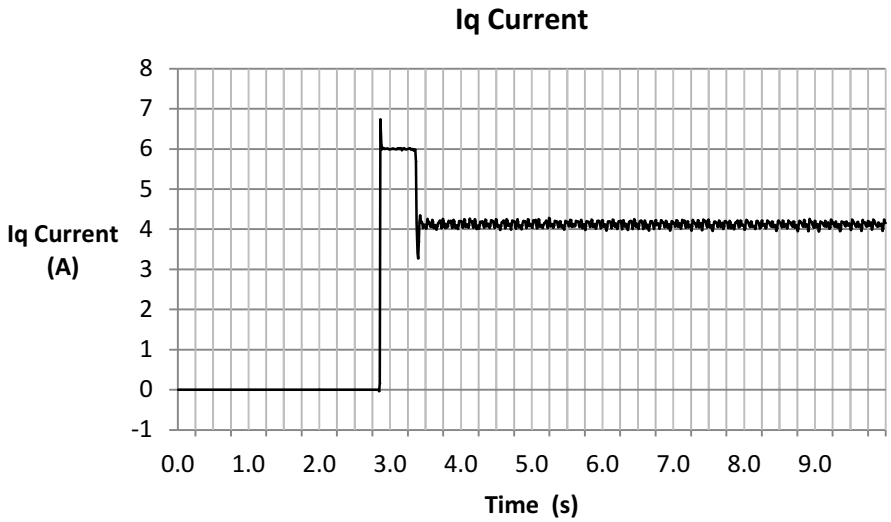


图 **14-55. Iq** 电流图

我们通过本示例可以得出的结论是，我们所能达到的最快加速度取决于初始估算角度与电机实际角度之间的最初差异。 这是因为强制角特性将在角度对准期间启用，并且最多占用一个电周期。 user.h 中提供的强制角的电周期默认频率为 1Hz，因此从静止加速最多需要 1 秒。 不过，根据电机轴的负载类型，可以增大此强制角频率以实现从静止最快地加速启动。

14.4.2 需要电机对准时满载最快速度启动电机的考量

为解决开始时的电机对准问题，我们将讨论在电机闭环运行之前使用 Rs 重校准工具实现电机对准。 在本例中，Rs 重校准电流用于使电机轴旋转至最初的对准位置。 在接下来的章节中讨论的考虑事项也适用于此运行模式：

- 加载有效偏移并禁用偏移重校准；在节 14.4.1.1中说明。
- 启用定子 Rs 重校准；在节 14.1.1.2中说明。
- 最大化电流斜率；在节 14.4.2.1中说明。
- 启用强制角；在节 14.3.1.1中说明。
- 调整速度控制器以避免电机停转；在节 14.1.1.4中说明。
- 调整电压反馈电路；在节 14.1.1.5中说明。

14.4.2.1 最大化电流斜率

为了重校准 Rs，需要向电机施加反向 DC 电流，因此在此模式下运行时必须考虑这一点。 该 DC 电流必须在电机进入闭环运行之前尽快移除。 为此，应使用以下示例代码将最大电流斜率值更改成一个最大值。

```
// set max current slope value to a maximum
EST_setMaxCurrentSlope_pu(obj->estHandle,_IQ(127.99));
```

最大斜率函数的输入参数从 IQ24 获得一个值，最大值为 127.99。 当开始输入电流以重校准电阻时，这会引起一个电流的阶跃增长，而且要实现尽快移除这个电流，也会引发一个阶跃。

14.4.2.2 需要电机对准时满载最快速度启动电机的示例

在本例中，应用程序必须允许闭环运行前的初始对准时间。 该对准时间作为 Rs 重校准时间在 user.c 中进行配置，并且将加载到以下两个数组成员中：

```
pUserParams->RsWaitTime[EST_Rs_State_RampUp] = (uint_least32_t)(1.0*USER_EST_FREQ_Hz);
pUserParams->RsWaitTime[EST_Rs_State_Fine] = (uint_least32_t)(5.0*USER_EST_FREQ_Hz);
```

在本例中，总共有 6 秒的时间用于对准电机。 本例中的对准应当在充足的时间和足够的电流条件下实现，以便在电机进入闭环运行前，能观察到电机调整位置并对准。 在将来的 InstaSPIN 版本中，将增加初始位置检测 (IPD) 功能以完全避免电机对准。 请记住，仅在 EST_Rs_State_Fine 状态期间会有内阻 Rs 更新。 图 14-56 显示了运行时的其中一个电流波形。 可以看出在 DC 值从 -1A 跳变为 0A 的电流加速很规整（移除用于电机对准的电流），然后电流呈正弦形式从 0A 升至 4A（峰峰电流 8A）。 这是由于电机最初已对准，没有出现因为未对准引起的反向运行。 另外，对准过程应该让用户观察到。

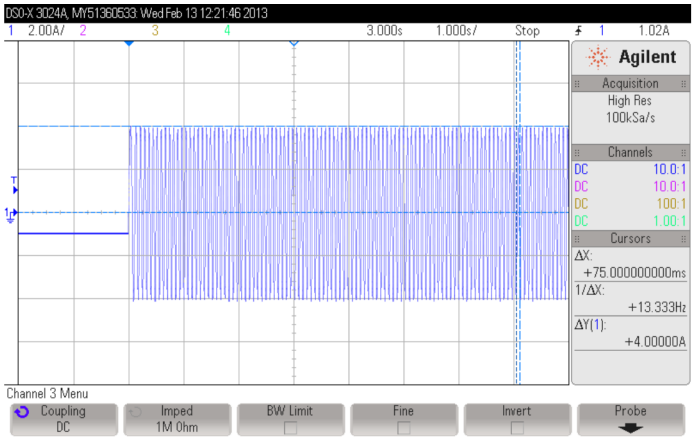


图 14-56. 需要电机对准时满载最快速度启动电机的图例

如果放大第一部分的电流（图 14-57），可以看出 -1A 的电流由于高电流斜率配置通过一个阶跃立即被移除，之后速度便跟随电流变化。 此时，可根据需要大胆调整速度控制器以实现所需的加速度响应。

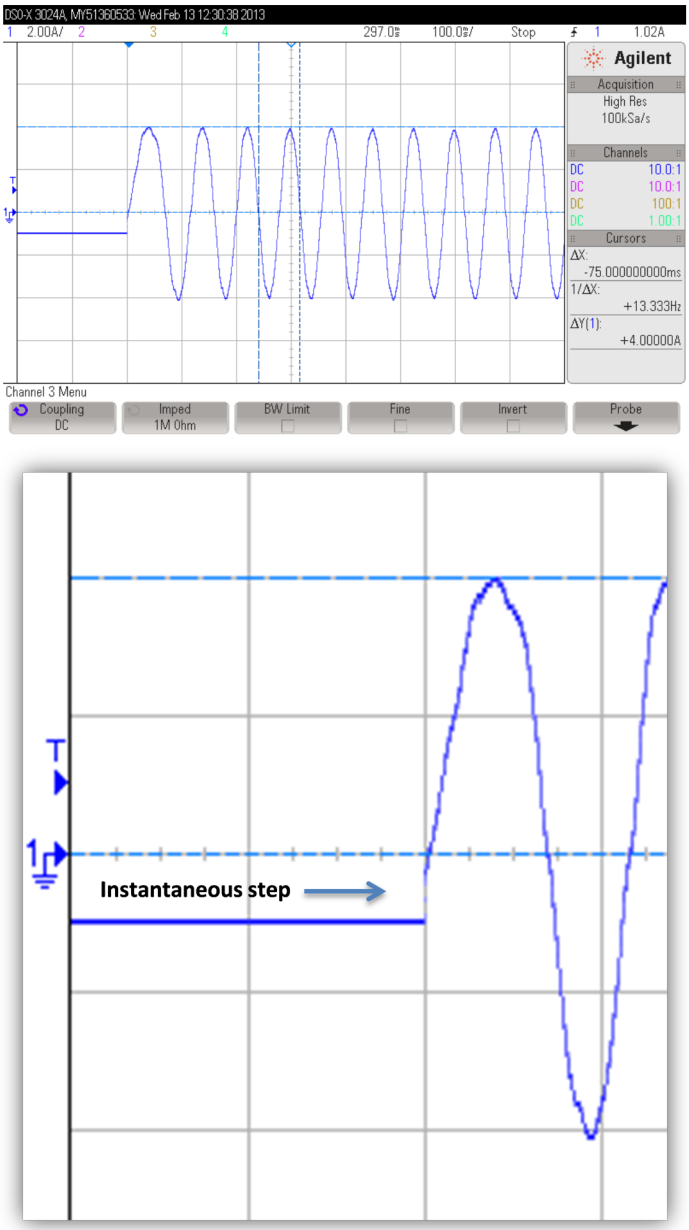


图 **14-57.** 放大的电流图

可以看出磁通量不像之前那样包含一个瞬态，随着电机在闭环系统中运行，磁通量现在只是稳定到一个恒定值（图 14-58）。

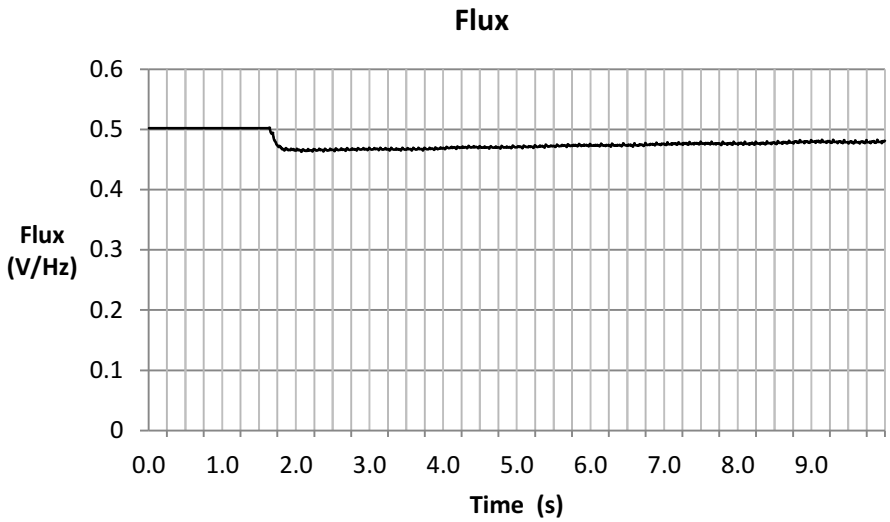


图 **14-58.** 磁通量图

角度波形从零升到一个高频率（图 14-59），说明由于需要初始对准，强制角在电机加速期间甚至没有激活。

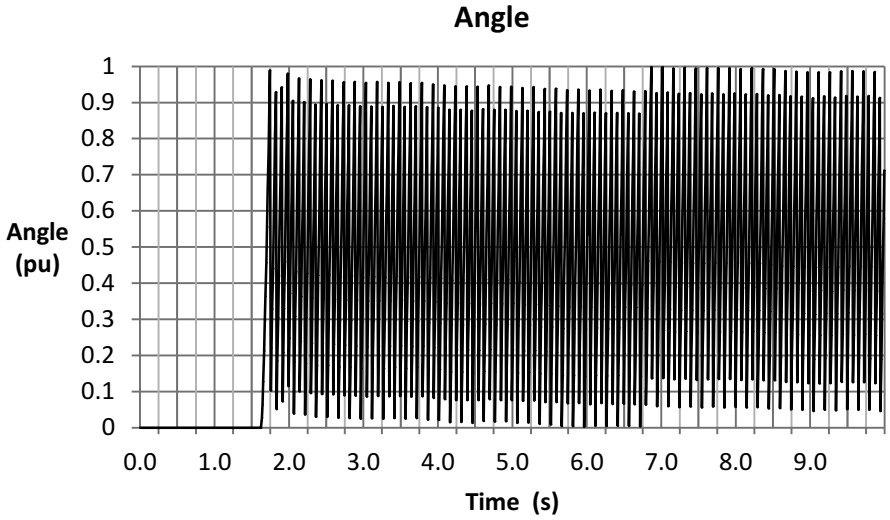


图 **14-59.** 角度图

若是放大角度图 (图 14-60)，可以再次看到角度曲线斜升过程没有强制角介入的特征，电机直接从静止状态进入闭环运行。

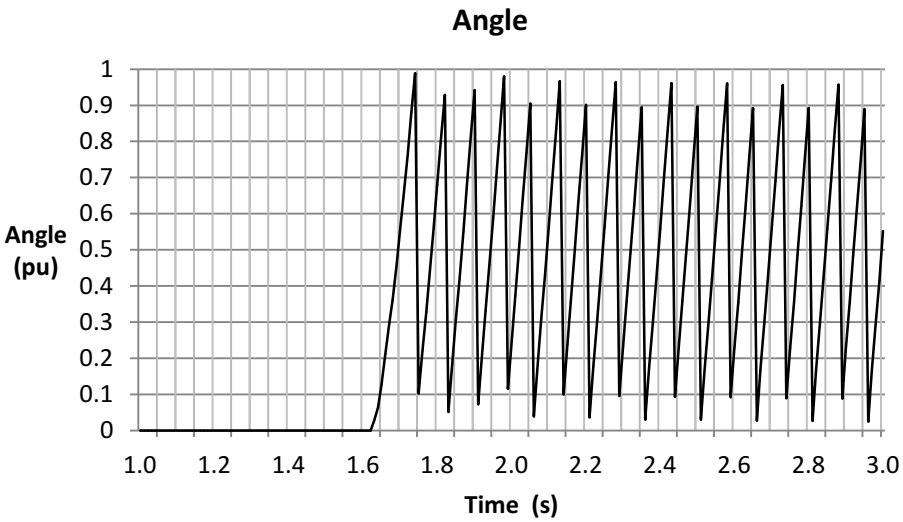


图 **14-60.** 放大的角度图

观察估算速度（图 14-61），可以看出没有负向旋转。 速度响应直接是速度控制器的速度响应，没有强制角初始未对齐现象。

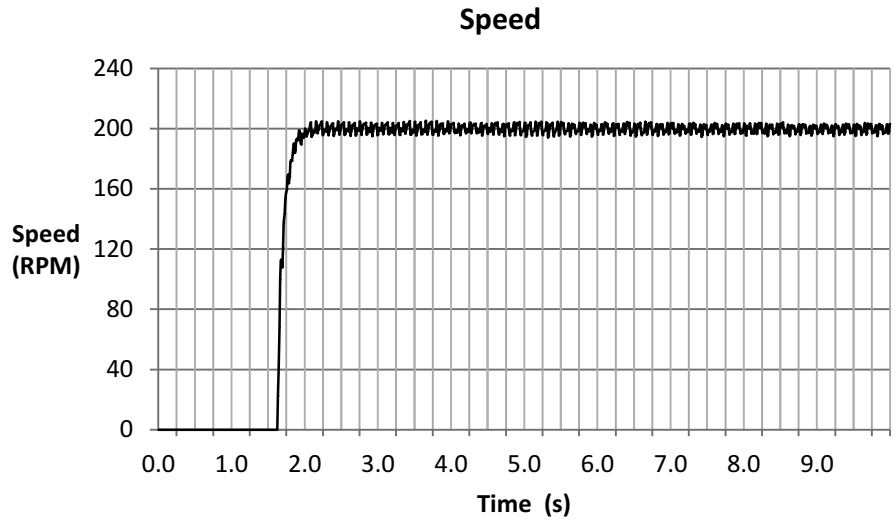


图 **14-61.** 速度图

同样，估算转矩从零增加到大约 2Nm 的目标值（图 14-62），然后稳定在大约 1.9Nm 处（我们在测力计中设置的目标数值）。

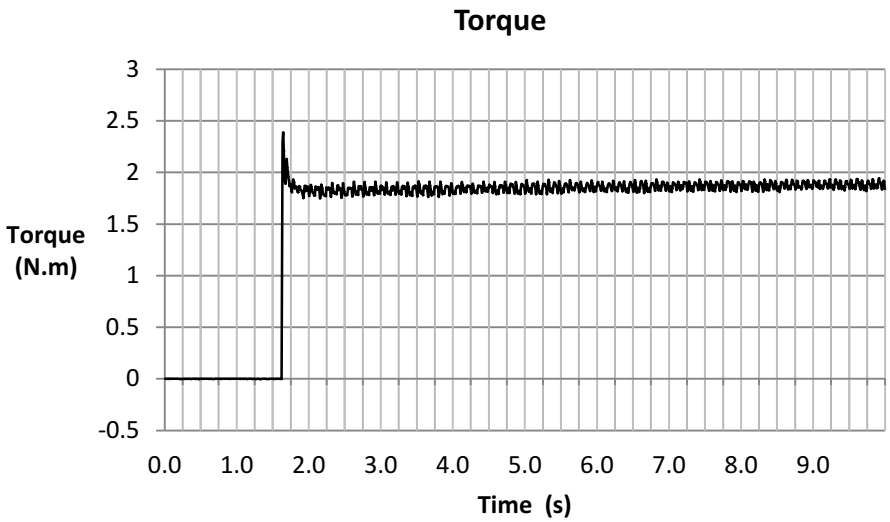


图 **14-62.** 转矩图

可以看出电流最初发生过冲，原因是速度控制器发出了高加速度命令（图 14-63）；在过冲后，电流稳定在大约 4A，该电流可以产生最大转矩。

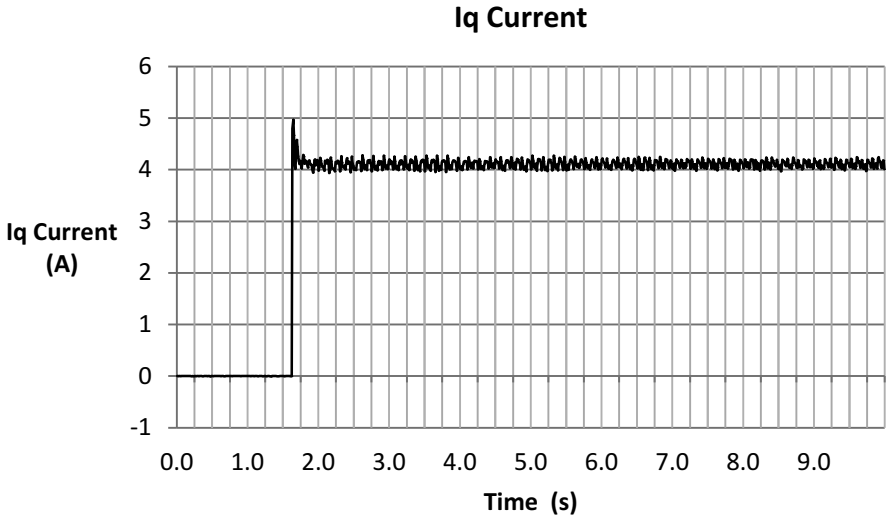


图 **14-63. Iq** 电流图

14.5 过载和电机过热

本节将讨论要使电机在过载时仍然能运行很长一段时间所需考虑的事项。 为实现此目的，我们将使用InstaSPIN 的 Rs 在线特性，以便能在电机运行期间识别和重新校准定子电阻 (Rs)。

14.5.1 过载和电机过热的考量

在接下来的章节中讨论的考虑事项也适用于此运行模式：

- 启用偏移重校准；在节 14.1.1.1中说明。
- 启用定子 Rs 重校准；在节 14.1.1.2中说明。
- 启用 Rs 在线特性，在Chapter 15中说明。

- 启用强制角；在节 14.3.1.1中说明。
- 调整速度控制器以避免电机停转；在节 14.1.1.4中说明。
- 调整电压反馈电路；在节 14.1.1.5中说明。

14.5.2 过载和电机过热示例

在本例中，负载增加到超出电机额定转矩大约 30%（图 14-64）。 测力计设置的转矩命令为 2.5Nm。 请记住，电机的额定转矩大约为 1.9Nm，所以实际在电机轴上施加了 130% 的负载。 这会引起电机过热，因此需要使用 Rs 在线特性使电机在运行期间保持一个准确的电阻值。

在图 14-64 中，可以看到一个较慢的转角叠加在 5A 振幅的电流上，这是由于 Rs 在线特性的影响。 有关 Rs 在线特性的相关信息，请参见Chapter 15。

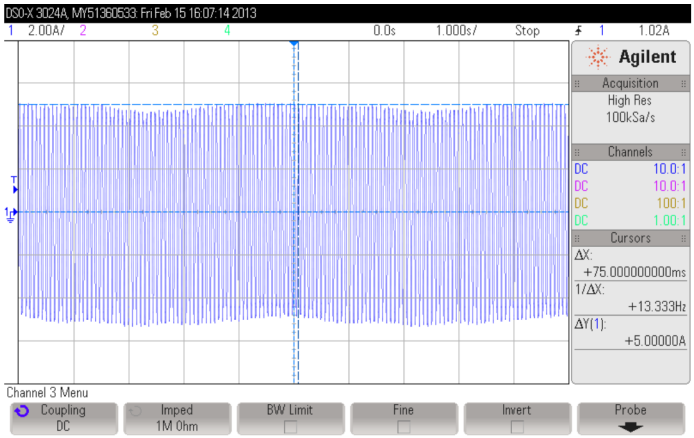


图 **14-64.** 过载和电机过热图

放大电流（图 14-65），可以清楚的看出频率为 13.33 Hz，正好是我们指定速度参考值（本例为 200 RPM）的频率。 相关转换大家都很熟悉，取决于电极对数。 速度 (RPM) = 速度 (Hz) * 60/电机对数 = 13.33Hz * 60/4 = 200RPM。

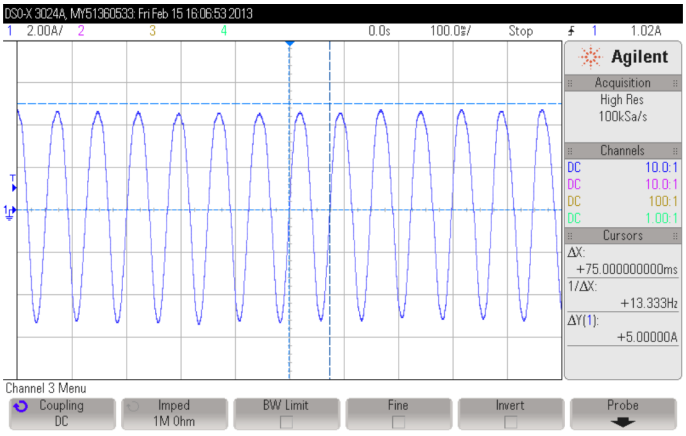


图 **14-65.** 放大的过载和电机过热图

定子电阻采集了 5 分钟（300 秒），在图 14-66 中显示。 可以看出起始值为 2.8 欧姆，经过 200 秒的时间后，达到大约 3.45 欧姆并稳定在这个值。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

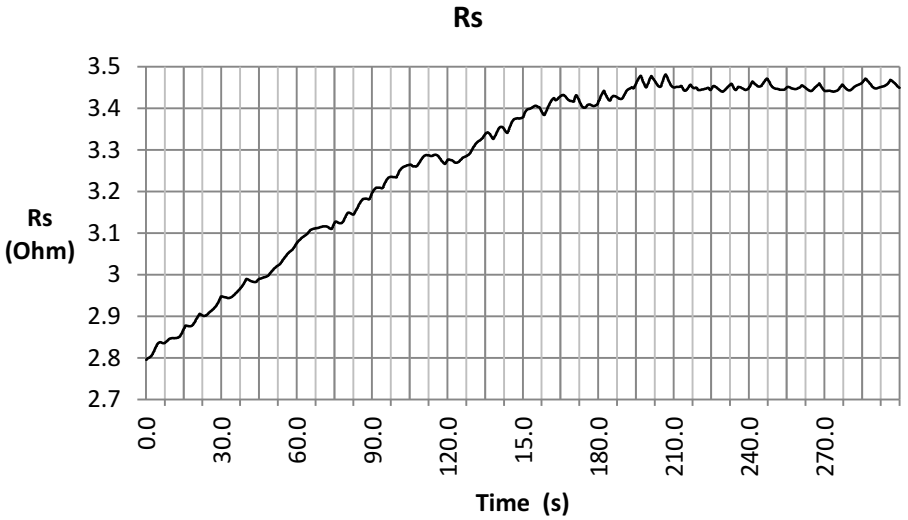


图 **14-66.** 定子电阻图

基于此增长情况，可以得出电阻增大了 (3.45-2.8)/2.8 * 100% = 23%。 应用Chapter 15中的公式，可得出此电阻变化表示的电机温度为：

$$T = T_0 + \frac{\frac{R}{R_0} - 1}{\alpha}$$

$$T = 30°C + \frac{\frac{3.45\Omega}{2.8\Omega} - 1}{0.00393°C^{-1}}$$

$$T = 89°C$$

工作 10 分钟后得到以下估算值，此时电阻测量值大约为 3.5 欧姆。 估算的磁通量（图 14-67）反映了非常接近额定磁通量 0.5V/Hz 的一个值。

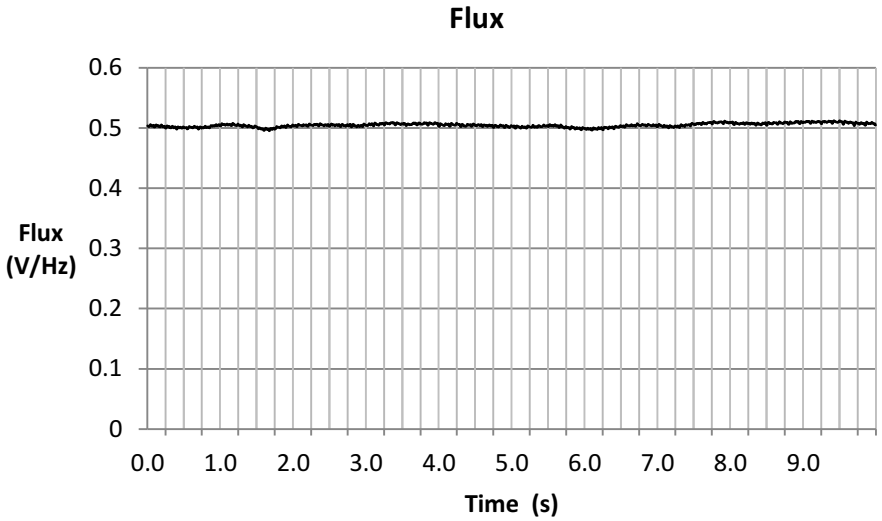


图 **14-67.** 磁通量图

我们绘制了在电机过载 30% 时的角度图（图 14-68）。

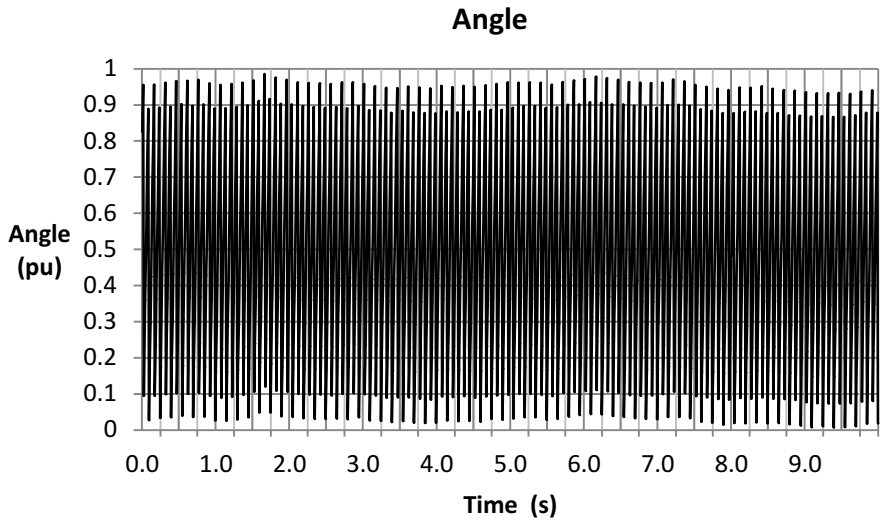


图 **14-68.** 角度图

放大角度图（图 14-69）并扩大到 1 秒的信息，我们可以看到清晰且连续的斜坡曲线。 请记住，该数据是按每 200 个样本的频度采集的，所以在每个周期的结束时会有不连续的地方。

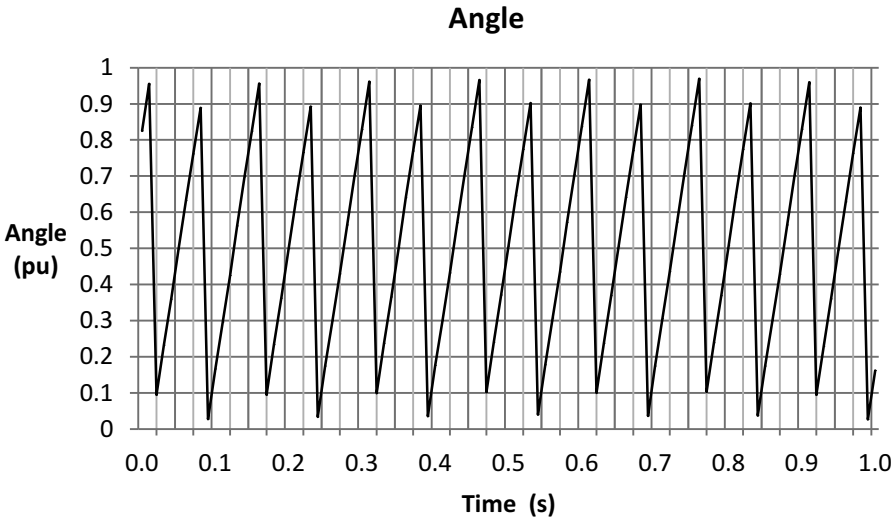


图 **14-69.** 放大的角度图

估算速度大约为 200 RPM，如图 14-70 所示。

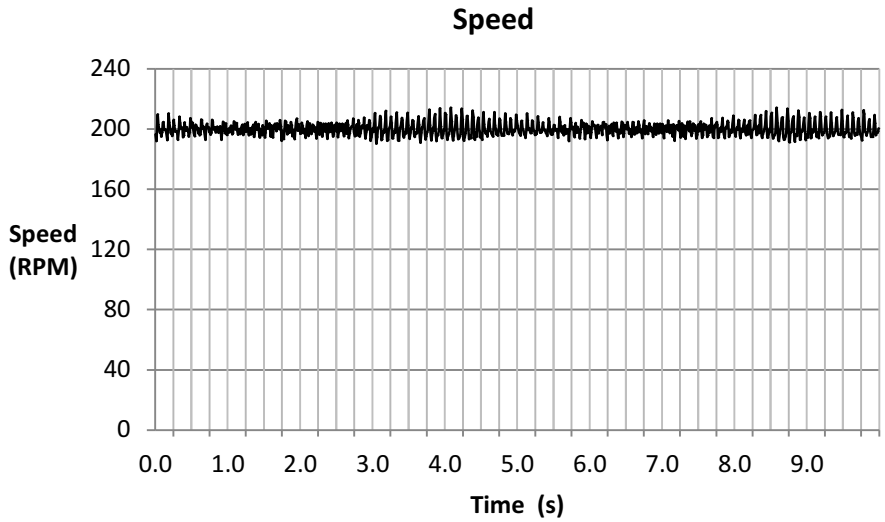


图 **14-70.** 速度图

估算转矩大约为 2.5Nm。 在电机过载时，估算转矩同样是准确的和相对恒定的，如图 14-71 所示。

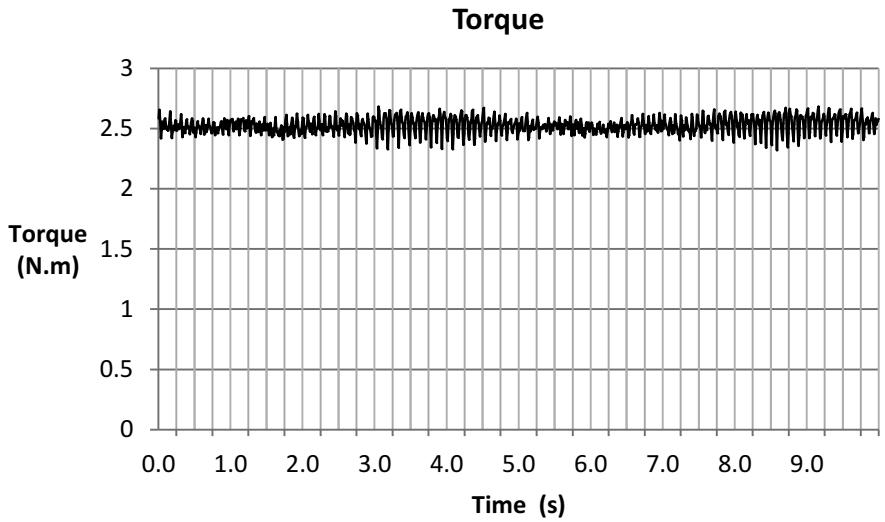


图 **14-71.** 转矩图

本例中的电流大约为 5.2A（图 14-72），由于长时间 30% 过载导致定子电阻升高，为产生指令转矩，电流必须更高。

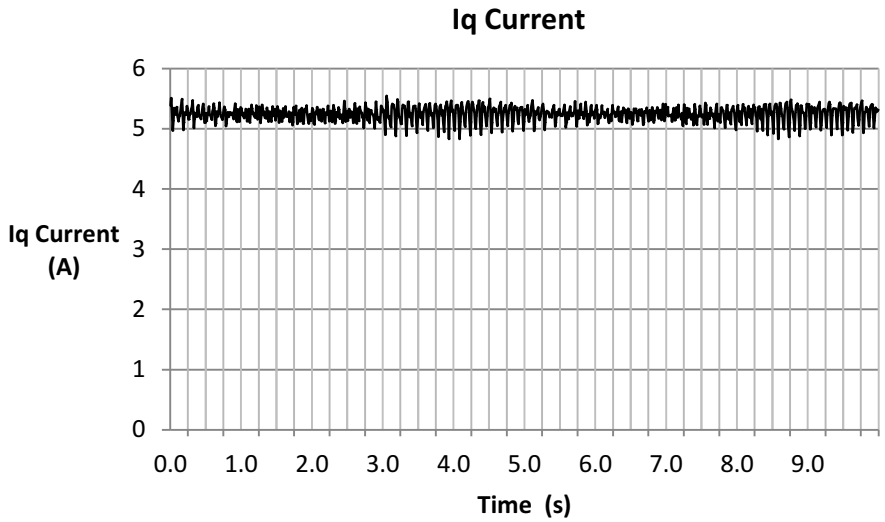


图 **14-72. Iq** 电流图

14.6 InstaSPIN-MOTION 和低速运行的考量

电机低速运行时需要考虑的一点是，必须调整速度控制器以避免电机停转。 InstaSPIN-MOTION 提供的 SpinTAC 速度控制器是一种先进的速度控制器，它有一个通用的参数调节器并支持宽泛的工作范围。 由于这些技术的结合，使得用户很容易实现调整 SpinTAC 控制器以避免电机在低速运行时停转。 针对低速运行调整 SpinTAC 速度控制器时，非常重要的一点是带宽可能需要设置为高出以额定速度运行时的值。 完成此设置需要达到的目标是，SpinTAC 速度控制器将能够更积极地动作以消除干扰和调节低速。 有关 InstaSPIN-MOTION 提供的 SpinTAC 速度控制器的更多信息，请参见Chapter 12。

Rs 在线重校准

Rs 在线重校准是一种 InstaSPIN-FOC 特性，当电机运行在闭环中时，该特性可用于重新校准定子电阻 Rs。 这里的术语“在线”用于描述在闭环磁场定向控制 (FOC) 下运行电机的系统。 可在 FAST 估算器内部实现这一特性，虽然未提供源代码，但这个特性的所有参数均可根据应用需求进行修改，这将在本文档中进行说明。图 15-1 显示了 FAST 估算器，其中突出显示了与 Rs 在线特性相关的接口区域。

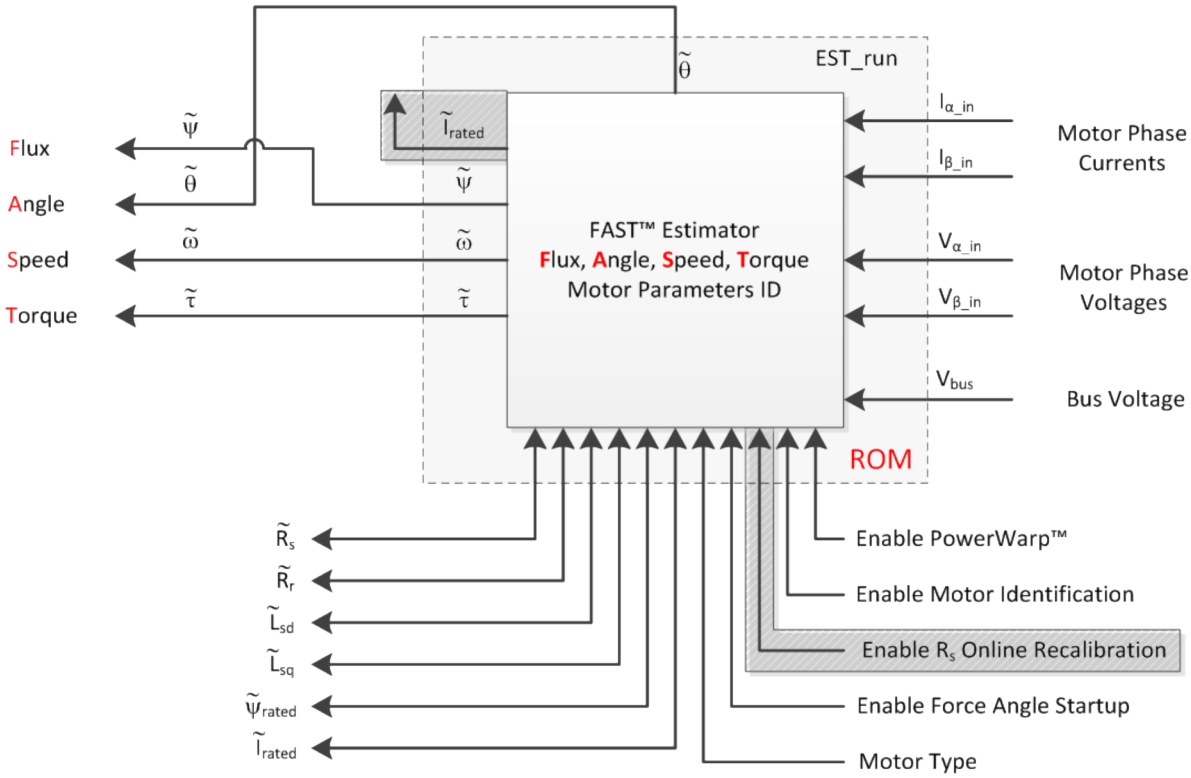


图 **15-1. FAST** 估算器 **- Rs** 在线特性突出显示

Topic **Page**

15.1 概要

当线圈（也称为电机绕组）的运行温度发生变化时，电机线圈的定子电阻（即 Rs）可能会出现大幅度变化。 有几种因素可导致运行温度升高。 以下示例列出了可能会影响定子线圈温度的一些条件：

- 流过线圈的电流过大。
- 电机机壳不能实现自冷。
- 恶劣的运行环境导致温度升高。
- 电机附近有其它发热元件。

温度升高的结果是电机绕组的电阻增大。 电阻与温度的关系可根据绕组自身所使用的材料进行明确定义。

15.2 电阻与 温度

用于制造绕组的常见材料是铜。 以下公式表示电阻与温度关系的线性近似关系：

$$R = R_0[1 + \alpha(T - T_0)] \quad (80)$$

其中：

- R：在温度 T 时的电阻，以欧姆 (Ω) 为单位
- R_0：在温度 T_0 时的电阻，以欧姆 (Ω) 为单位
- α：材料的温度系数，以负一次方摄氏度 ($^oC^{-1}$) 为单位
- T：材料的最终温度，以摄氏度 (^oC) 为单位
- T_0：材料的参考温度，以摄氏度 (^oC) 为单位

例如，定子电阻 Rs 在 20ºC 时为 10Ω，其绕组材料为铜，温度系数为 0.00393$^oC^{-1}$。 如果电机温度升高到 150ºC，则新的定子电阻为：

$$R = R_0[1 + \alpha(T - T_0)] \quad (81)$$

$$R = 10\Omega[1 + 0.00393^oC^{-1}(150^oC - 20^oC)] \quad (82)$$

$$R = 15.109\Omega \quad (83)$$

可以看出，随着温度升高电阻有显著的变化，在本例中，大约增加了 52%。

15.3 低速运行（包括启动）时所需的 Rs 准确值

这种电阻变化会对用于 FAST 的电机模型产生影响，在低速运行时更是如此。 这是因为在低速运行时电机模型内部的大部分压降取决于定子电阻和电流 DC 分量：

$$R_s i_s \quad (84)$$

另一方面，当电机在中速至高速运行时定子电阻的变化不会对电机模型的性能产生显著影响，因为此时电机模型的内部压降主要取决于反电势和电感与电流的微分的乘积，即：

$$L_s \frac{di_s}{dt} + e_s \quad (85)$$

低速性能要求知道准确的定子电阻值，尤其是在满载运行时（包括在满载下从静止启动电机）。 下一节将介绍如何在 InstaSPIN-FOC 环境中使用 Rs 在线重校准。

15.4 Rs 在线重校准介绍

将 Rs 在线重校准添加到 InstaSPIN-FOC 中，从而可在电机运行在闭环中时提供准确的定子电阻。 对电阻进行实时更新并根据新电阻来更新电机模型，从而使电机在整个工作范围（从无负载到满载）运行时都能提供最佳性能结果。

仔细观察 InstaSPIN-FOC 框图，Rs 在线重校准可通过设置一个标志来启用。 当电机运行时，通过向电流直接分量（又称为 D 坐标轴电流）注入电流，即可测量定子电阻。图 15-2 突出显示了用于 Rs 在线重校准的区域。

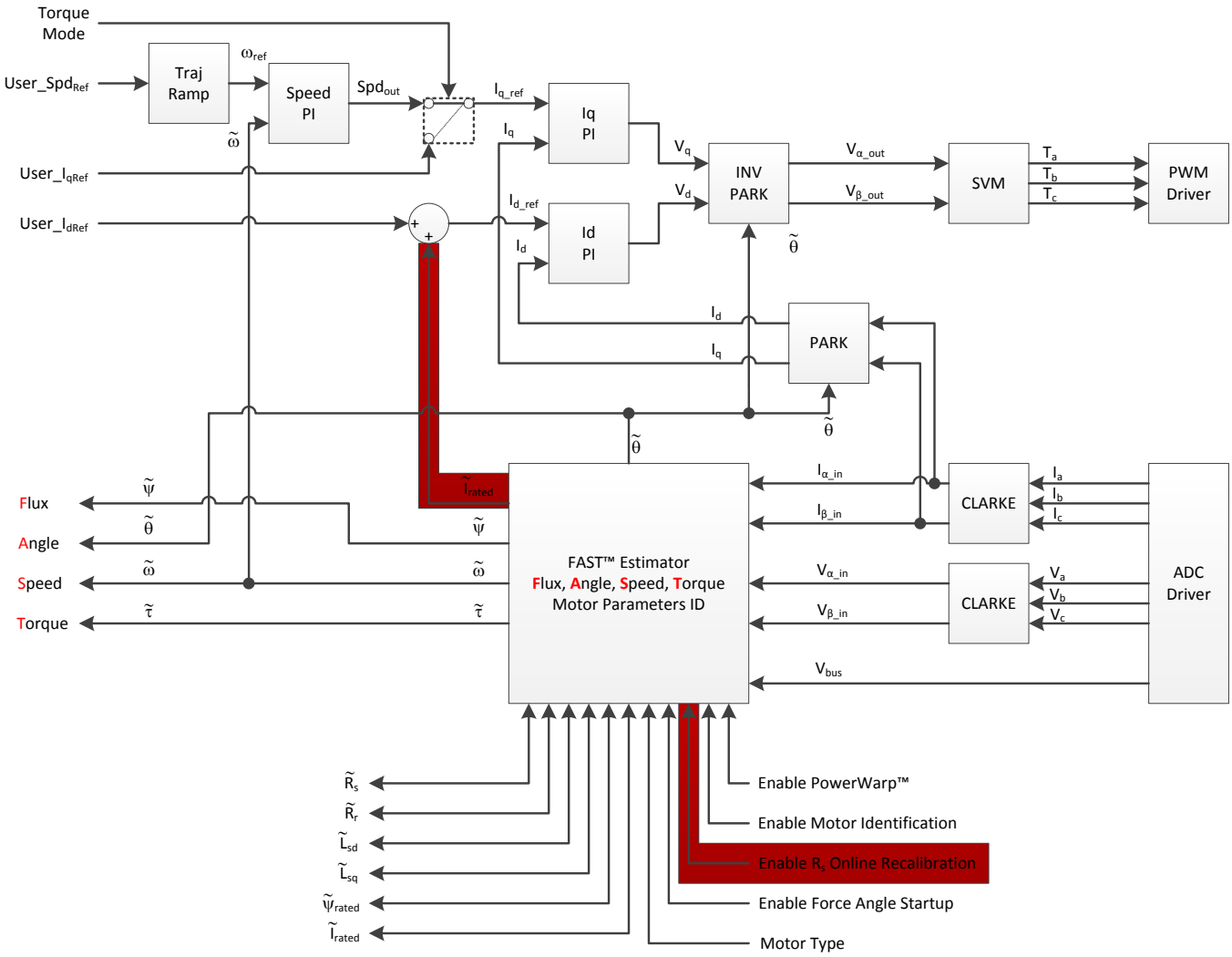


图 **15-2. Rs** 在线重校准

从框图中可以看出，在线电阻重校准可通过向 I_{rated} 添加额外分量来实现。 此添加操作在 FAST 估算器内部执行，FAST 输出的 I_{rated} 已经包含了 Rs 在线重校准所需的电流。 对于永磁电机，I_{rated} 可以为零，对于感应电机则为磁化电流。 在磁场减弱和磁场增强时，可以添加另一个电流 User_I_{dRef}，而这个额外电流不会影响 Rs 在线重校准。

需要注意的是，Rs 在线重校准可以根据 FAST 模块所注入的额外 Id 的交流值进行计算。 这些电流交替出现正值和负值以便运行内部算法。 除此之外，用户仍然可以基于来自 FAST 的值控制用户的 I_d 参考。 例如，典型情况是电机在磁场减弱区域运行时 I_d 参考值为负，此时 FAST 会提供一个新的 I_{rated} 来计算在线电阻重校准。 这将是预期出现的典型使用情况，这种情况下会在磁场减弱和在线电阻重校准两方面给出预期的结果。

由于电流注入已完成，流入电机的相位电流波形会出现失真（不是标准正弦波），具体取决于所注入电流与转轴机械负载的比率。 对于轻负载，正弦波形会受到很大的影响，而对于部分负载至满载，电流波形变化几乎令人无法察觉。 下面显示了一些图形，分别显示了电机运行速度为 500RPM 时，启用和禁用 Rs 在线重校准两种情况下不同机械负载所对应的电流波形。 可以看出，在某些情况下，相位电流的正弦波形失真。

图 15-3 显示了在禁用 Rs 在线重校准的情况下，电流在轻负载时为正弦波。

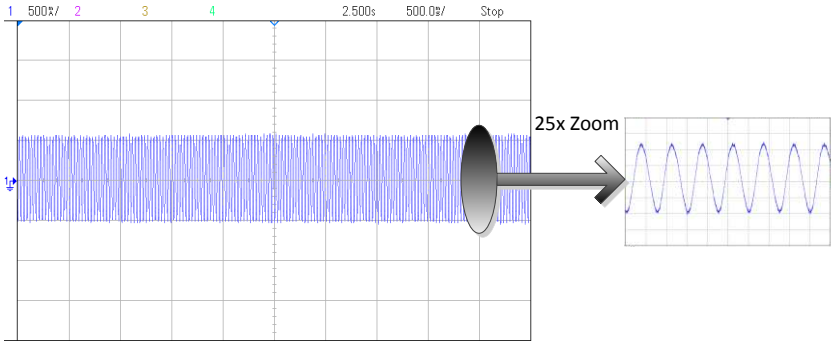


图 15-3. 轻负载下的相位电流 - 禁用 Rs 在线重校准

在相同机械负载条件下，如果启用 Rs 在线重校准，从图 15-4 可以看出电流波形为失真的正弦波。

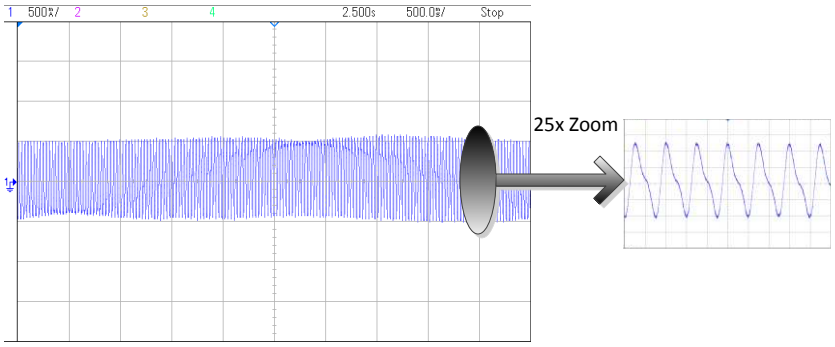


图 15-4. 轻负载下的相位电流 - 启用 Rs 在线重校准

另一方面，当电机存在机械负载时，无论是否启用 Rs 在线重校准，都很难观察到波形的变化。图 15-5 显示了禁用 Rs 在线重校准且存在机械负载情况下的电流波形。

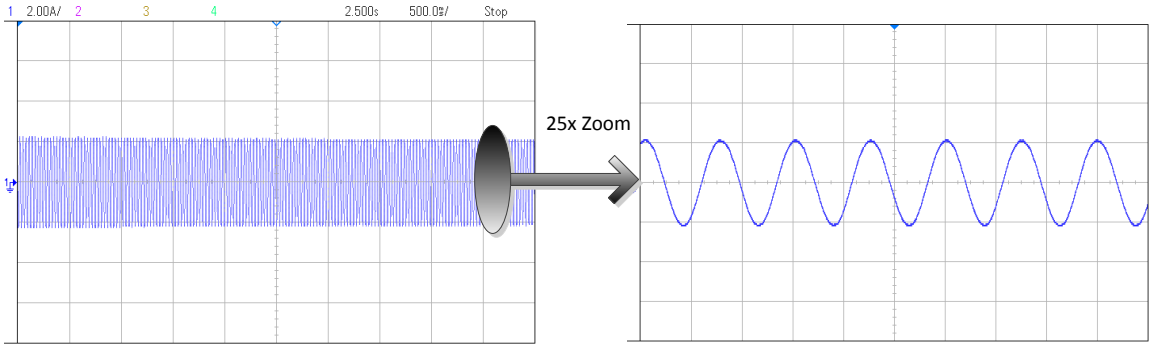


图 15-5. 存在机械负载时的相位电流 - 禁用 Rs 在线重校准

图 15-6 显示了启用 Rs 在线重校准且存在机械负载情况下的电流波形。

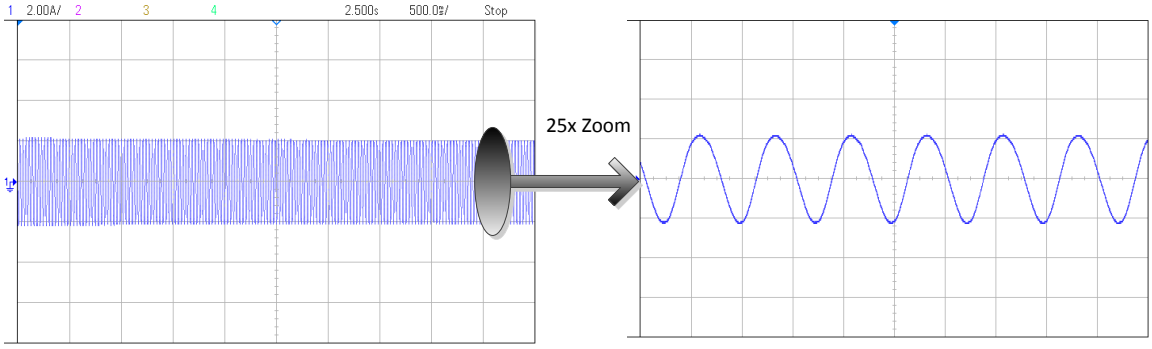


图 15-6. 存在机械负载时的相位电流 - 启用 Rs 在线重校准

从最后一个图形可以看出，即使启用 Rs 在线重校准，在中负载至高负载情况下失真会小很多，并且随着负载增加，越来越难以观察到失真。 在本文档后面的几节中，将会介绍实现 Rs 在线重校准时所需的最小额外电流。

Rs 在线与 Rs 离线

InstaSPIN-FOC 还包含在电机旋转前执行的另一种电阻重校准，称为 Rs 离线重校准。 Rs 离线重校准要求电机静止，并向 I_d 注入 DC 电流。 而 Rs 在线重校准是要求电机旋转以重新校准电阻，并向 I_d 注入 AC 电流。

Rs 离线和 Rs 在线重校准都是 InstaSPIN-FOC 提供最佳低速性能的重要组成部分。 在典型应用中，图 15-7 显示了 Rs 离线和 Rs 在线重校准的使用。

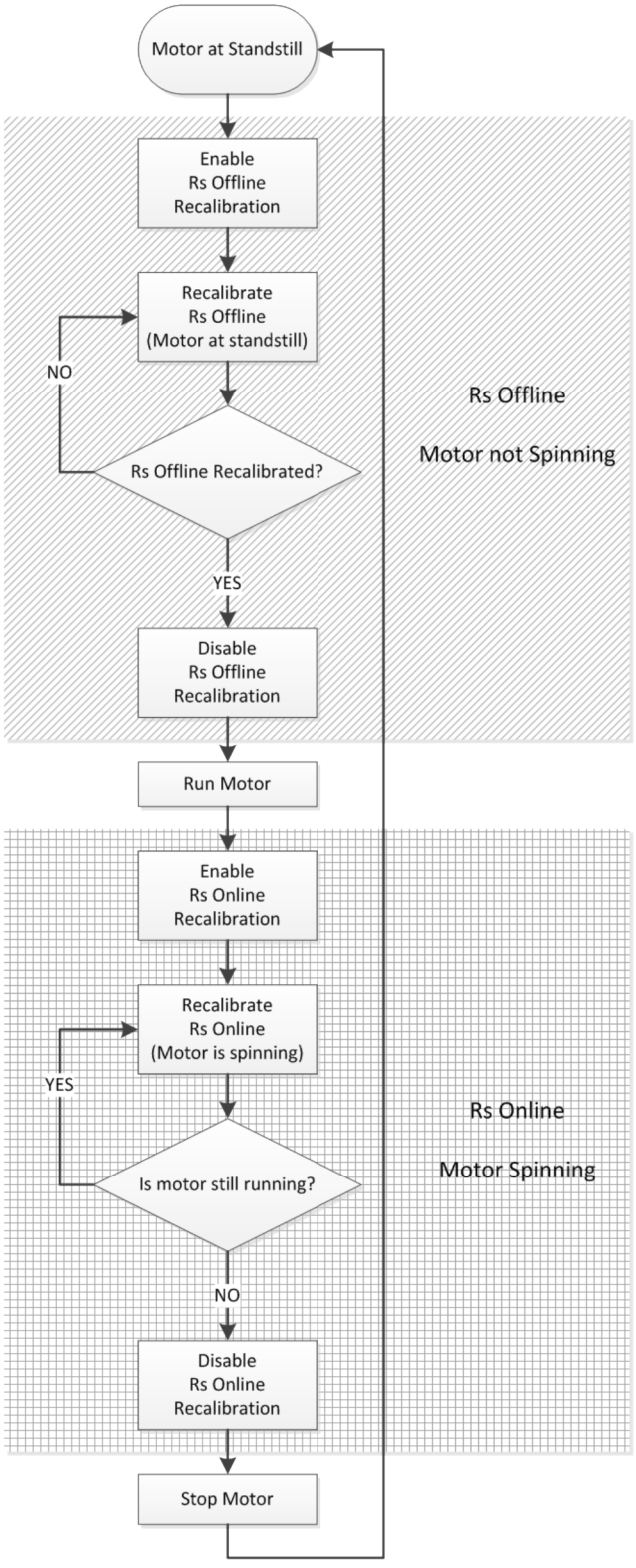


图 **15-7. Rs** 在线和 **Rs** 离线重校准流程图

启用 Rs 在线重校准

为了启用 Rs 在线重校准，需要设置一些参数。 其中一个最重要的参数是为实现 Rs 在线重校准需要向 D 坐标轴电流 (Id) 注入电流的大小。 通常，建议使用的最小电流为额定电流的 5%，以便在电机运行时通过可测量电流获得准确的电阻重校准结果。

```
_iq RsOnLineCurrent_A = _IQ(USER_MOTOR_MAX_CURRENT * 0.05);
```

请注意，乘法由预编译器执行，使用 USER_MOTOR_MAX_CURRENT 定义的浮点值乘以 0.05 来表示 5%，然后将浮点结果转换为全局 IQ 值。 有关 IQmath 库的详细信息，请参见《*C28x IQMath Library – 虚拟浮点引擎 – 模块用户指南*》（文献编号 SPRC990）

另外，在启用 Rs 在线重校准之前，用户必须设置：

- 电阻表示的初始 Q 格式值。 可使用函数 EST_setRsOnLine_qFmt () 完成此操作。 初始值必须取自在电机静止时通过 Rs 离线重校准测量的值。 可通过调用函数 EST_getRs_qFmt () 来读取此值。
- 将用于 Rs 在线重校准的 Id 幅度设置为零。 可通过调用函数 EST_setRsOnLineId_mag_pu () 来执行此操作。
- 将 Id 的标么值设置为零。 可通过调用函数 EST_setRsOnLineId_pu () 来执行此操作。
- 启用标志并将标志更新为 FALSE，可通过调用函数 EST_setFlag_enableRsOnLine () 和 EST_setFlag_updateRs () 来完成。

当禁用 Rs 在线重校准时，需要将 Id_mag_pu 和 Id_pu 的值都设置为零，以防止估算器保留任何剩余电流参考。 在 InstaSPIN 的未来版本中，不需要执行此操作，只需要将 Id_mag_pu 重置为零。但是对于 2806xF 器件，仍需要在启用或禁用 Rs 在线重校准前将上述两个值设置为零。

在估算器内部，两个标志分别用于执行不同的任务。 设置 enableRsOnLine 标志后即可运行完整的 Rs 在线特性，可以更新内部变量使其保持最新的电阻值，并向 Id 注入电流。 设置第二个标志 updateRs 后，电机模型便可使用电阻值。 如果未设置 updateRs 标志，但设置了 enableRsOnLine 标志，则仍可使用电阻来监控电阻变化，但是内部电机模型不能使用此变化电阻。 如果电机温度显著升高且未更新电机模型中的电阻（将 updateRs 标志设置为 TRUE），则 InstaSPIN 的性能会受到影响，并且无法达到所需的电机低速性能要求。 另外，在满载时电机可能无法启动。

以下代码示例显示如何设置初始值以及如何检查条件以确保在状态机处于正常状态时设置初始值。 需要在启用 Rs 在线重校准前执行此操作：

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;

// get the controller state
gMotorVars.CtrlState = CTRL_getState(ctrlHandle);

// get the estimator state
gMotorVars.EstState = EST_getState(obj->estHandle);

if((gMotorVars.CtrlState <= CTRL_State_OffLine) ||
       ((gMotorVars.CtrlState == CTRL_State_OnLine)    &&
        (gMotorVars.EstState == EST_State_Rs)))
  {
   EST_setRsOnLine_qFmt(obj->estHandle,EST_getRs_qFmt(obj->estHandle));
   EST_setRsOnLineId_mag_pu(obj->estHandle,_IQ(0.0));
   EST_setRsOnLineId_pu(obj->estHandle,_IQ(0.0));
   EST_setFlag_enableRsOnLine(obj->estHandle,FALSE);
   EST_setFlag_updateRs(obj->estHandle,FALSE);
```

```
  }
```

由于此代码示例仅包含全局变量而不含时间关键代码，因此可在主函数中的死循环中（中断程序外）执行此代码示例。 请记住，“if”条件中的代码需要在启用 Rs 在线重校准前执行。

如上述代码示例所示，重置 Rs 在线重校准所有参数的其中一个条件是 CtrlState 小于或等于 CTRL_State_OffLine。 此条件表示当状态机空闲（电机未通电）或执行偏移重校准时应禁用并重置 Rs 在线重校准。 另一个必须重置这些初始值的条件是控制状态为在线 (CTRL_State_OnLine) 且估算器状态为 EST_State_Rs，即正在执行 Rs 离线重校准。 所有这些条件表示电机静止。 有关 InstaSPIN 中这些状态与整个状态机的关系，请参见Chapter 6。 CTRL_State_OffLine 状态在 CTRL 状态机框图中显示为*离线*，CTRL_State_OnLine 状态在 CTRL 状态机框图中显示为*在线*，而 EST_State_Rs 状态在 EST 状态机框图中显示为 ***Rs***。

当电机运行时，需要确保 Rs 在线重校准特性给定的电阻与 Rs 离线特性给出的初始电阻足够接近。 这是为了确保两个电阻值之间进行平稳转换，从而确保不会对闭环系统造成任何干扰。 可通过以下代码执行此操作，其中包含了上一个代码示例中的“else”条件。 只要电机未处于静止状态（即电机正在旋转），就会执行以下条件。 此条件可用于启用 Rs 在线重校准，如下所示。

```
else
  {

   // Scale factor to convert Amps to per units.
   // USER_IQ_FULL_SCALE_CURRENT_A is defined in user.h
   _iq sf = _IQ(1.0/USER_IQ_FULL_SCALE_CURRENT_A);

   Rs_pu = EST_getRs_pu(obj->estHandle);
   RsOnLine_pu = EST_getRsOnLine_pu(obj->estHandle);
   Rs_error_pu = RsOnLine_pu -    Rs_pu;

   EST_setFlag_enableRsOnLine(obj->estHandle,TRUE);
   EST_setRsOnLineId_mag_pu(obj->estHandle,_IQmpy(RsOnLineCurrent_A,sf));

   // Enable updates when Rs Online is only 5% different from    Rs Offline
   if(_IQabs(Rs_error_pu) <    _IQmpy(Rs_pu,_IQ(0.05)))
     {
      EST_setFlag_updateRs(obj->estHandle,TRUE);
     }
  }
```

请注意，在此示例中通过调用以下函数启用 Rs 在线重校准：

```
EST_setFlag_enableRsOnLine(obj->estHandle,TRUE)
```

但是，直到将更新标志设置为 TRUE 之后，才会更新 Rs 在线值。 当 Rs 在线值与 Rs 离线所提供的初始值足够接近时（在本例中为该值的 5%），将更新标志设置为 TRUE。 对于要求启动转矩为满转矩的应用，建议针对 Rs 在线和 Rs 离线的差值选择更小的比例，即用 3% 左右的值代替 5%。 一旦 Rs 在线值和 Rs 离线值的差值小于 5%，就可以通过调用以下函数来设置 Rs 在线更新标志：

```
EST_setFlag_updateRs(obj->estHandle,TRUE)
```

启用这两个标志后，Rs 在线重校准将实时重新计算电阻，而估算器会根据新电阻值更新其内部电机模型。

另外，如代码示例所示，可通过调用以下函数来设置用于估算在线电阻的注入电流的幅度：

```
EST_setRsOnLineId_mag_pu(obj->estHandle,_IQmpy(RsOnLineCurrent_A,sf));
```

禁用 Rs 在线重校准

为了禁用 Rs 在线重校准，用户可以参考第一个代码示例（在此处再次列出），在其中写入初始值并禁用标志。 按照之前的讨论，此代码示例还将检查状态机的状态是否正确以便正确地禁用 Rs 在线重校准：

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;

// get the controller state
gMotorVars.CtrlState = CTRL_getState(ctrlHandle);

// get the estimator state
gMotorVars.EstState = EST_getState(obj->estHandle);

if((gMotorVars.CtrlState <= CTRL_State_OffLine) ||
       ((gMotorVars.CtrlState == CTRL_State_OnLine)    &&
        (gMotorVars.EstState == EST_State_Rs)))
  {
   EST_setRsOnLine_qFmt(obj->estHandle,EST_getRs_qFmt(obj->estHandle));
   EST_setRsOnLineId_mag_pu(obj->estHandle,_IQ(0.0));
   EST_setRsOnLineId_pu(obj->estHandle,_IQ(0.0));
   EST_setFlag_enableRsOnLine(obj->estHandle,FALSE);
   EST_setFlag_updateRs(obj->estHandle,FALSE);
  }
```

修改 Rs 在线参数

可以在 InstaSPIN-FOC 的 Rs 在线特性中调整和修改几个参数。 下列参数将在本节中详细地进行讨论：

- 注入电流幅度
- 慢速转角
- Rs 在线值递增和递减的增量
- 滤波器参数

调整注入电流幅度

本节描述的第一个参数是实现 Rs 在线重校准所需的 D 坐标轴电流 (Id) 的注入电流。 启用 Rs 在线重校准后，此电流可由估算器模块自身生成，其幅度由用户配置。 如前文所述，建议使用的注入电流值约为电机额定电流的 5%，以便从电机返回可测量电流，因而实现准确的 Rs 在线重校准。 例如，当电机额定电流为 5A 时，注入电流为 0.25A。以下代码示例将 Rs 在线重校准的注入电流设置为 0.25A：

```
// Scale factor to convert Amps to per units.
// USER_IQ_FULL_SCALE_CURRENT_A is defined in user.h
_iq sf = _IQ(1.0/USER_IQ_FULL_SCALE_CURRENT_A);

// Value corresponding to 0.25 Amps
_iq RsOnLineCurrent_A = _IQ(0.25);
```

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;

// Scale value from Amps to per units and set through the use of an API
EST_setRsOnLineId_mag_pu(obj->estHandle,_IQmpy(RsOnLineCurrent_A,sf));
```

当 Rs 在线重校准运行时，图 15-8 显示了电流波形的振幅，同时测得在未向电机施加机械负载时的电流值约为 0.25A。 可以看出，无负载时的电流为 0.1A，如果针对 Rs 在线重校准添加 0.25A 的电流，则峰值电流为 $I_s = \sqrt{I_q^2 + I_d^2} = \sqrt{0.1^2 + 0.25^2} = 0.27\,A$

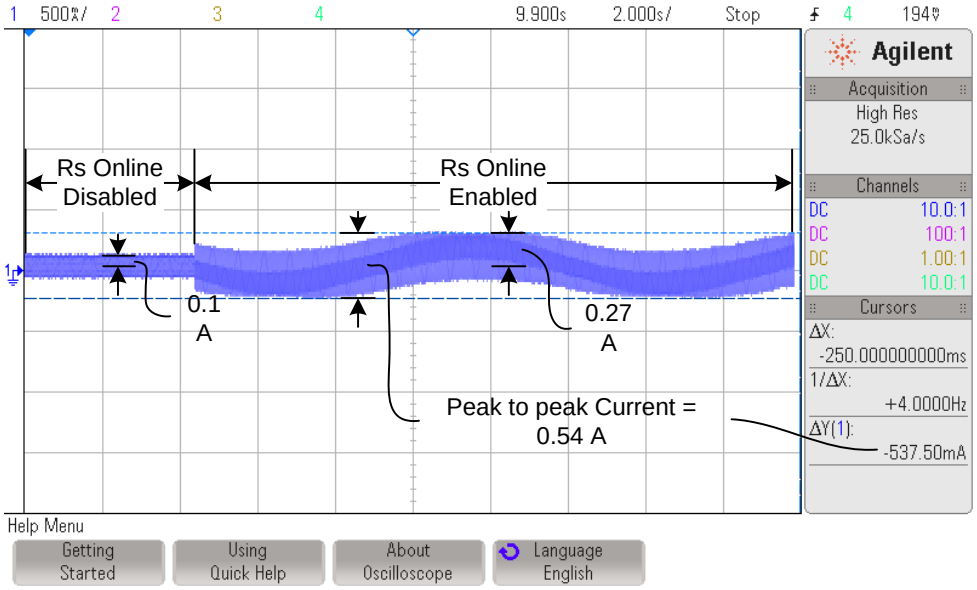


图 **15-8.** 针对 **Rs** 在线重校准添加 **0.25A** 电流的结果

随着电机负载的增加，Rs 在线重校准所需的额外电流按比例减小，如图 15-9 所示。 这种情况下的负载电流为 0.35A，仍使用 0.25A 的 Rs 在线注入电流。使用上一个示例中的公式，计算出最大电流为：$I_s = \sqrt{I_q^2 + I_d^2} = \sqrt{0.35^2 + 0.25^2} = 0.43\,A$

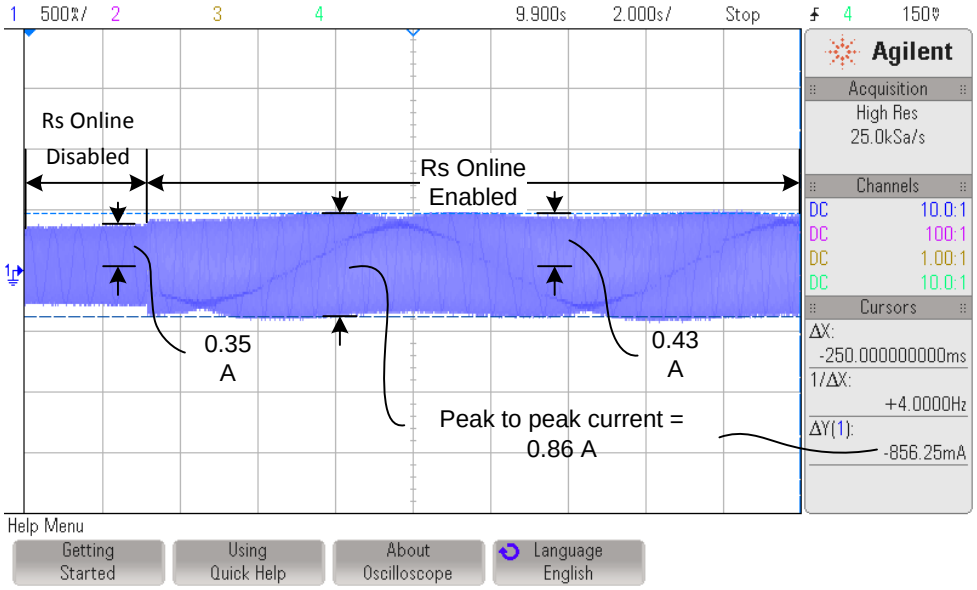


图 **15-9.** 针对 **Rs** 在线重校准增加负载的结果

还有示例的电机额定电流为 10A，则 Rs 在线注入电流为 0.5A。使用不同电流值的同一代码示例如下所示：

```
// Scale factor to convert Amps to per units.
// USER_IQ_FULL_SCALE_CURRENT_A is defined in user.h
_iq sf = _IQ(1.0/USER_IQ_FULL_SCALE_CURRENT_A);

// Value corresponding to 0.5 Amps
_iq RsOnLineCurrent_A = _IQ(0.5);

CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;

// Scale value from Amps to per units and set through the use of an API
EST_setRsOnLineId_mag_pu(obj->estHandle,_IQmpy(RsOnLineCurrent_A,sf));
```

与之前的示例类似，可以使用相同的公式计算最大电流值：

$$I_s = \sqrt{I_q^2 + I_d^2} = \sqrt{0.6^2 + 0.5^2} = 0.78\,A \tag{86}$$

本示例中的相应示波器图形如图 15-10 所示。

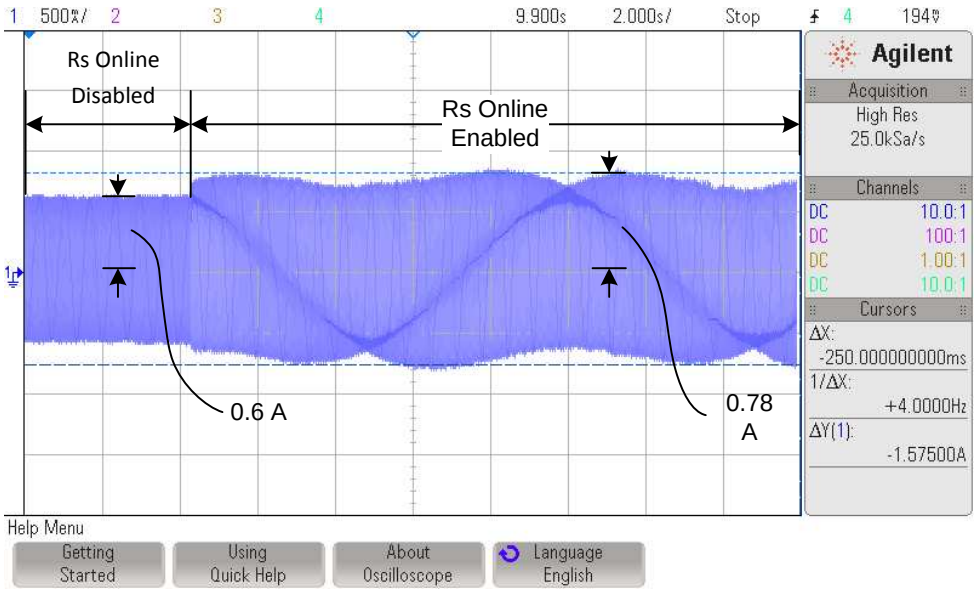


图 **15-10.** 启用 **Rs** 在线重校准时的最大电流

通常，如果考虑在电机满载的情况下，电机的电流为 10A，针对 Rs 在线重校准添加电机电流的 5%，则最大总电流为：

$$I_s = \sqrt{I_q^2 + I_d^2} = \sqrt{10.0^2 + 0.5^2} = 10.0125\,A \tag{87}$$

即，提供给电机的额外电流仅为 0.0125A，仅相当于额定电流的 0.125%。 从额外发热的角度来看，这个电流通常不会对电机产生影响。

例如，电机的额定电流为 2.2A，使用额定电流的 5% 作为 Rs 在线电流，即 2.2*0.05 = 0.11A。额外电流仅为额定电流的 0.125%，即 2.2*0.00125 = 0.0028A。观察启用 Rs 在线重校准之前和之后的示波器图形，甚至无法注意到额外电流 (图 15-11)。 读者同样需要注意的是，示波器的垂直方向已从 500mA/ 放大至 2.00A/，因为出于电机机械负载等原因导致电流的振幅远远高于之前。

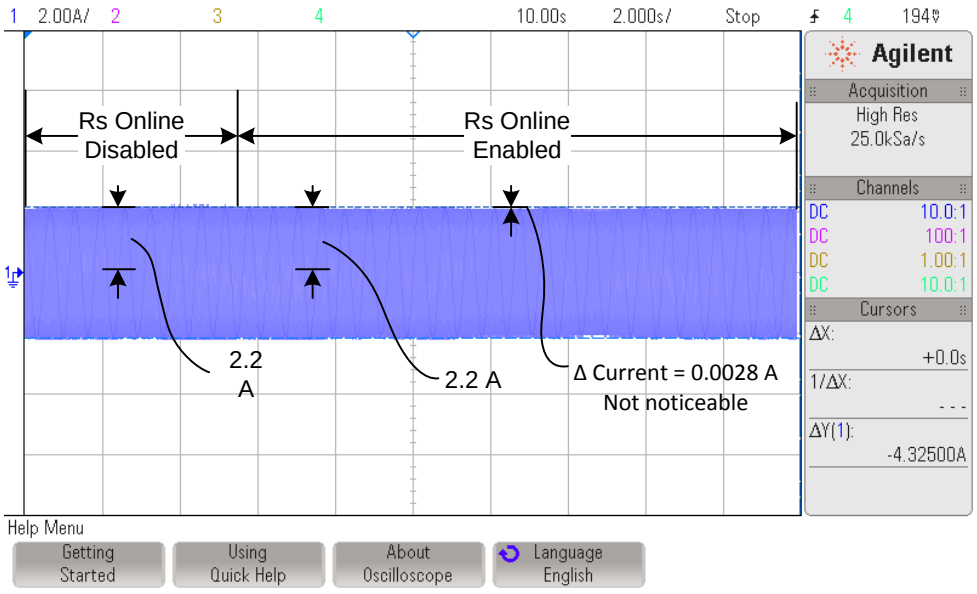


图 **15-11.** 电机电流 **2.2A** 与 **5% Rs** 在线电流

调整慢速转角

实现 Rs 在线特性要求特定内部矢量随着电机的旋转缓慢转动。 这个慢速旋转的矢量默认设置为值 0.00001（以标么值表示），因此它会生成一个频率为 0.00001 * 估算器频率（以 Hz 为单位）的矢量。 所以如果估算器频率为 10kHz，则旋转矢量的频率为 0.1Hz，周期为 10 秒。 需要使用此旋转矢量以便 Rs 在线重校准能够估算出在矢量不同点测量的平均电阻。

尽管不了解 Rs 在线重校准如何估算变化电阻的细节，但一定要了解此转角可用于估算在不同电流矢量下的电阻。 随着时间的推移，在线电阻为所有矢量慢速旋转时多次测量所产生的平均电阻。 为了让用户了解设置的旋转矢量值，除了在示波器上观察电流外，用户还可以使用以下代码示例。

```
// These defines are in user.h
#define USER_NUM_ISR_TICKS_PER_CTRL_TICK (1)
#define USER_NUM_CTRL_TICKS_PER_EST_TICK (1)
#define USER_PWM_FREQ_kHz (10.0)
#define USER_ISR_FREQ_Hz (USER_PWM_FREQ_kHz * 1000.0)
#define USER_CTRL_FREQ_Hz (uint_least32_t)(USER_ISR_FREQ_Hz \
                          /USER_NUM_ISR_TICKS_PER_CTRL_TICK)
#define USER_EST_FREQ_Hz (uint_least32_t)(USER_CTRL_FREQ_Hz \
                          /USER_NUM_CTRL_TICKS_PER_EST_TICK)

// Initialize obj to the controller handle
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;

_iq delta_pu_to_kHz_sf = _IQ((float_t)USER_EST_FREQ_Hz/1000.0);
_iq RsOnLine_Angle_Delta_pu = EST_getRsOnLineAngleDelta_pu(obj->estHandle);

// By default, the returned value in the following line will be close to:
// _IQ(0.00001), representing 0.0001 kHz, or 0.1 Hz
_iq RsOnLine_Angle_Freq_kHz = _IQmpy(RsOnLine_Angle_Delta_pu, \
                              delta_pu_to_kHz_sf);
```

有关 InstaSPIN 中使用的软件执行时钟树以及抽取因数（也称为节拍率）的详细信息，请参见Chapter 9。

如果角度增量从未改变，则由库将其设置为以标么值表示的默认值 0.00001，从而根据 0.00001 乘以估算频率得到慢速转角频率。 如果估算频率与 PWM 频率相同，均设置为 10kHz，则慢速转角的频率为 0.00001 * 10000 = 0.1Hz（周期为 10 秒）。 可以从电流波形中观察到慢速转角，以及随角度的变化电流以何种方式注入到 I_d 中。图 15-12 显示了电流波形如何以等于慢速转角频率的频率发生变化。

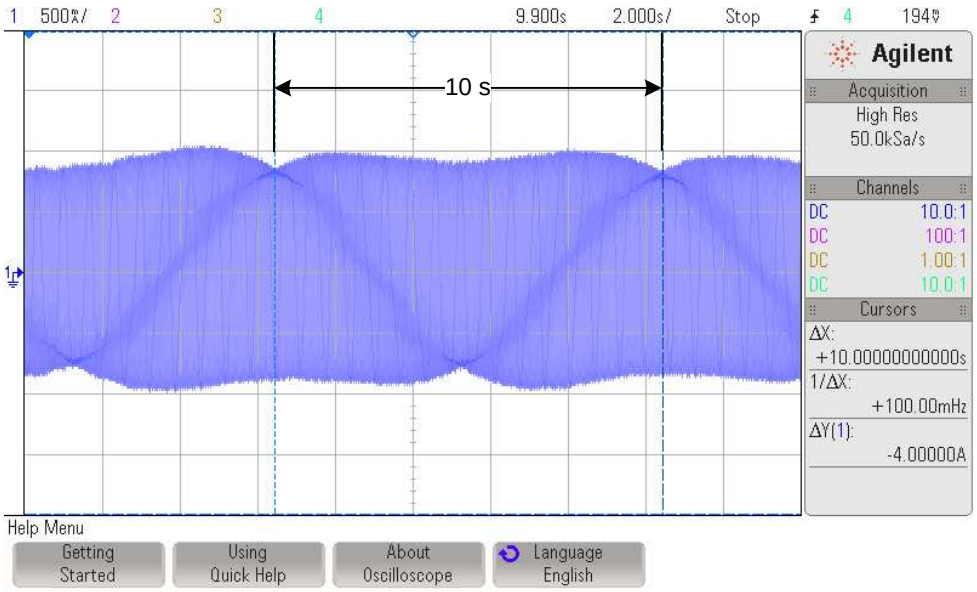


图 **15-12.** 频率等于慢速转角频率时的电流波形变化

例如，在某个应用中如果电机的温度上升速度过快，因此要求此转角速度更快一些，此时可能需要更改此转角。 旋转矢量不需要随温度的升高而改变。 只需要根据预期可能出现的系统最坏温度动态情况进行一次设置，而不需要随温度的变化对此值进行微调。 例如，如果系统的温度动态要求转角更改为 0.2Hz（周期为 5 秒），以下代码示例可用于将慢速转角更改为新值 0.2Hz：

```
// This new define represents the desired RsOnLine rotating angle frequency
#define RSONLINE_ANGLE_FREQ_Hz (0.2)

// Initialize obj to the controller handle
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;

// The scale factor (sf) calculation is done by the pre-compiler
_iq delta_hz_to_pu_sf = _IQ(1.0/(float_t)USER_EST_FREQ_Hz);
_iq RsOnLine_Angle_Freq_Hz = _IQ(RSONLINE_ANGLE_FREQ_Hz);
_iq RsOnLine_Angle_Delta_pu = _IQmpy(RsOnLine_Angle_Freq_Hz, \
                              delta_hz_to_pu_sf);

EST_setRsOnLineAngleDelta_pu(obj->estHandle,
          RsOnLine_Angle_Delta_pu);
```

在此代码示例中可以看到，函数现在为角度增量设置一个值，所以它会等待写入一个参数，这里为变量 RsOnLine_Angle_Delta_pu。

完成上述代码示例中的配置后产生的示波器图形如图 15-13 所示。

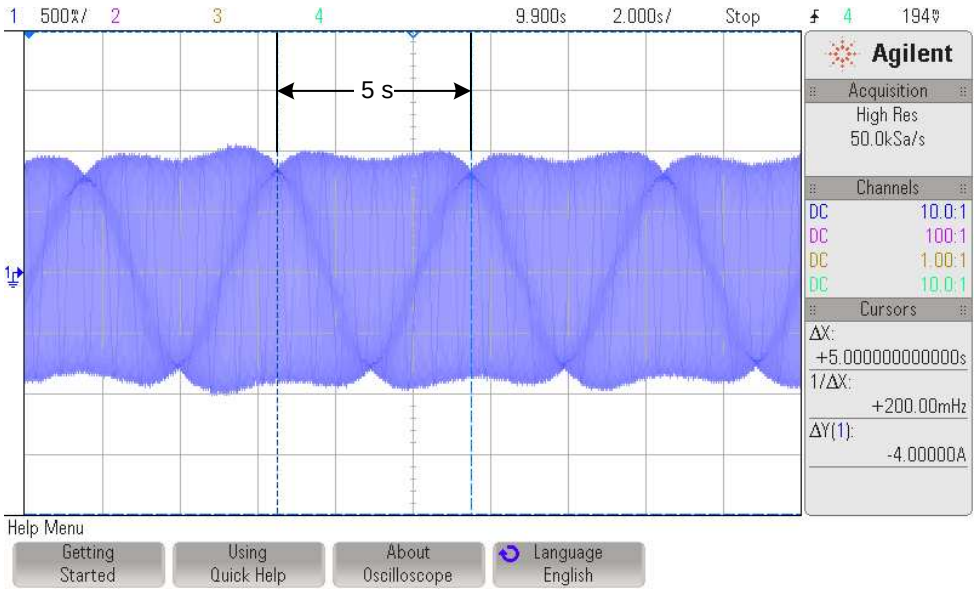


图 **15-13. RsOnLine_Angle_Delta_pu** 的结果

调整 **Rs** 在线值递增和递减的增量

在估算器中（尤其是运行 Rs 在线特性的估算器部分），可根据电阻的方向通过增加和/或减去固定增量值来更新实际电阻值。 通常，不需要更改此参数，除非是由于电机升温过快等原因导致电阻变化过快。 默认情况下，会将递增增量和递减增量设置为以 IQ30 格式表示的值 0.00001，可通过以下代码示例对其进行验证：

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;
_iq30 delta_dec = EST_getRsOnLine_delta_dec_pu(obj->estHandle);
_iq30 delta_inc = EST_getRsOnLine_delta_inc_pu(obj->estHandle);
```

为了修改这些增量，可使用下列代码示例更改此值，例如设置为默认值的两倍或 0.00002（以 IQ30 格式表示）：

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;
EST_setRsOnLine_delta_dec_pu(obj->estHandle, _IQ30(0.00002));
EST_setRsOnLine_delta_inc_pu (obj->estHandle, _IQ30(0.00002));
```

请注意，这两个函数用于设置增量，而不是获取增量，因此除句柄外它们还需要带一个参数。

图 15-14 显示了电阻如何根据增量值对初始值差异进行响应。 例如，在第一次启用 Rs 在线重校准的最开始阶段，初始电阻值与稳定状态的电阻值不同。 将增量值设置为 0.00001 会得到以下图形，此图中显示的斜率 = (0.77-0.4)/3.1 = 0.12Ω/s。

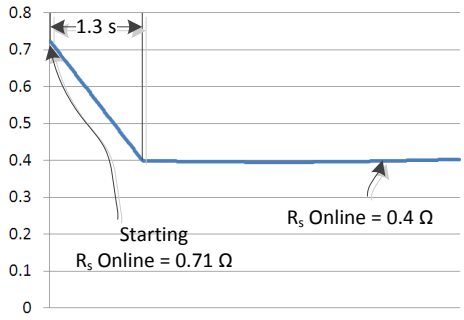


图 **15-14.** 初始值差异的电阻响应

当增量值更改为默认值的两倍时，可获得更快的稳定时间如图 15-15 所示，以及两倍的斜率 = (0.71 - 0.4) / 1.3 = 0.24Ω/s。

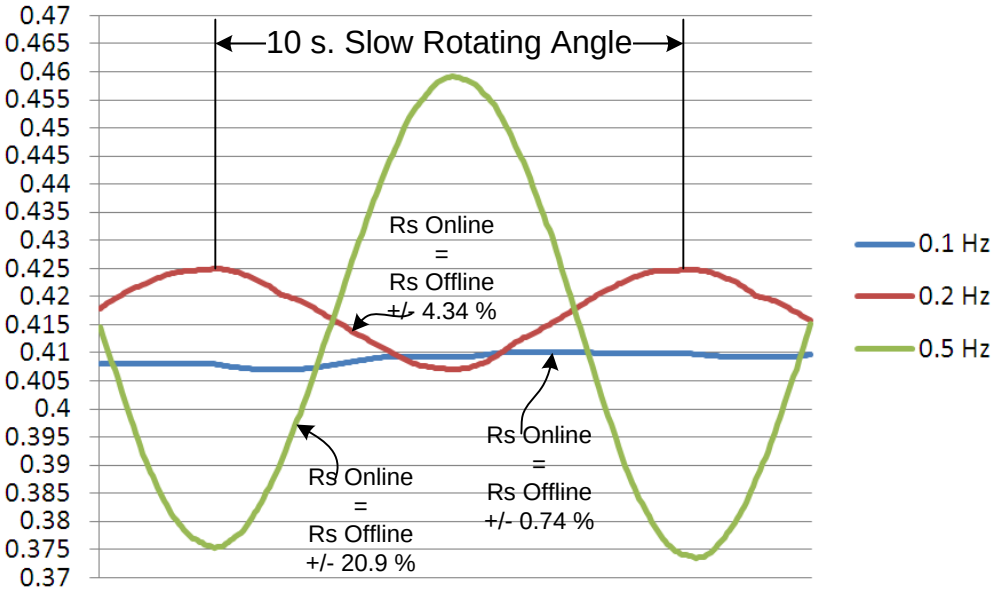


图 **15-15.** 增量值更改为默认值的两倍

如果要提供平滑的电阻变化曲线，建议为此变量选择足够慢的变化率，如果要跟踪系统的温度变化情况，则建议为此变量选择足够快的变化率。 通常，初始值 0.00001 即可满足要求，但在特定应用中，尤其是预期温度会发生剧烈变化时，需要对增量进行微调。

调整滤波器参数

Rs 在线估算器内部包含两个一阶级联滤波器，使用这两个滤波器是为了获得准确且稳定的定子电阻值。 总共有四个滤波器，其中一组的两个级联滤波器用于对电流进行滤波，另一组的两个级联滤波器用于对电压进行滤波。 每组的两个滤波器默认使用相同的系数。 但是，根据应用所需的响应，可以分别读取和写入这两个滤波器的系数。 这些滤波器的默认截止频率已设置为 0.2Hz。 用户可以通过以下代码示例来验证这个截止频率：

```
EST_getRsOnLineFilterParams(obj->estHandle, EST_RsOnLineFilterType_Current,
                            &pfilter_i0->b0, &pfilter_i0->a1, &pfilter_i0->y1,
                            &pfilter_i1->b0, &pfilter_i1->a1, &pfilter_i1->y1);
EST_getRsOnLineFilterParams(obj->estHandle, EST_RsOnLineFilterType_Voltage,
                            &pfilter_v0->b0, &pfilter_v0->a1, &pfilter_v0->y1,
```

```
                          &pfilter_v1->b0, &pfilter_v1->a1, &pfilter_v1->y1);

_iq pu_to_kHz_sf = _IQ((float_t)USER_EST_FREQ_Hz/1000.0);

cutoff_freq_kHz_i0 = _IQmpy(pfilter_i0->b0, pu_to_kHz_sf);
cutoff_freq_kHz_i1 = _IQmpy(pfilter_i1->b0, pu_to_kHz_sf);
cutoff_freq_kHz_v0 = _IQmpy(pfilter_v0->b0, pu_to_kHz_sf);
cutoff_freq_kHz_v1 = _IQmpy(pfilter_v1->b0, pu_to_kHz_sf);
```

默认情况下，截止频率变量会返回值 0.0002，表示 0.0002KHz 或 0.2Hz。 通常，应对这个截止频率进行调整，以便使电阻值可以随应用的预期温度动态增加。图 15-16 是 Rs 在线特性如何随截止频率变化的示例。图中使用的截止频率为 0.1Hz、0.2Hz 和 0.5Hz。 使用 0.1Hz 的慢速转角，如果滤波器的截止频率高于转角的频率，电阻将趋向于跟随转角，因此建议将这个滤波器至少设置为与慢速转角相同的频率，从而使电阻的过滤效果更好。

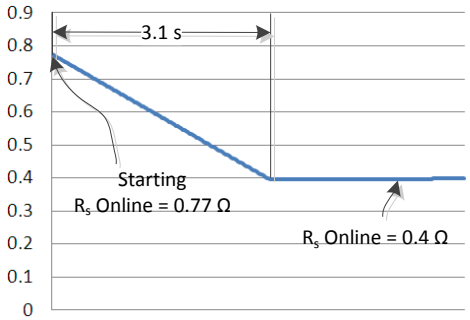


图 15-16. 随截止频率变化的 Rs 在线电阻值

可以看出，较低的截止频率可导致电阻变化较小。 但是，如果系统温度动态使温度上升速度过快，则使用低截止频率可能会出现问题。

在温度上升速度极快的情况下，需要将截止频率更改为更高的值。 如果应用要求更改截止频率，以下代码示例显示如何进行更改。 在本示例中，需要在运行中更改截止频率，因此必须先读取并修改截止频率，然后将其写回到相同的输出中，从而可以避免影响滤波器的输出。

```
EST_getRsOnLineFilterParams(obj->estHandle, EST_RsOnLineFilterType_Current,
                          &pfilter_i0->b0, &pfilter_i0->a1, &pfilter_i0->y1,
                          &pfilter_i1->b0, &pfilter_i1->a1, &pfilter_i1->y1);
EST_getRsOnLineFilterParams(obj->estHandle, EST_RsOnLineFilterType_Voltage,
                          &pfilter_v0->b0, &pfilter_v0->a1, &pfilter_v0->y1,
                          &pfilter_v1->b0, &pfilter_v1->a1, &pfilter_v1->y1);

// Use global variable desired_frequency_kHz to set the desired cutoff
// frequency of the filters. Use the same cutoff frequency for all filters
cutoff_freq_kHz_i0 = desired_frequency_kHz;
cutoff_freq_kHz_i1 = cutoff_freq_kHz_i0;
cutoff_freq_kHz_v0 = cutoff_freq_kHz_i0;
cutoff_freq_kHz_v1 = cutoff_freq_kHz_i0;

// Use the following scale factor to convert kHz to per unit value
_iq kHz_to_pu_sf = _IQ(1000.0/(float_t)USER_EST_FREQ_Hz);

// Calculate the per unit value for all filters
cutoff_freq_pu_i0 = _IQmpy(cutoff_freq_kHz_i0, kHz_to_pu_sf);
cutoff_freq_pu_i1 = _IQmpy(cutoff_freq_kHz_i1, kHz_to_pu_sf);
cutoff_freq_pu_v0 = _IQmpy(cutoff_freq_kHz_v0, kHz_to_pu_sf);
cutoff_freq_pu_v1 = _IQmpy(cutoff_freq_kHz_v1, kHz_to_pu_sf);
```

```
// Calculate coefficients for all filters
pfilter_i0->b0 = cutoff_freq_pu_i0;
pfilter_i0->a1 = cutoff_freq_pu_i0 - _IQ(1.0);
pfilter_i1->b0 = cutoff_freq_pu_i1;
pfilter_i1->a1 = cutoff_freq_pu_i1 - _IQ(1.0);
pfilter_v0->b0 = cutoff_freq_pu_v0;
pfilter_v0->a1 = cutoff_freq_pu_v0 - _IQ(1.0);
pfilter_v1->b0 = cutoff_freq_pu_v1;
pfilter_v1->a1 = cutoff_freq_pu_v1 - _IQ(1.0);

// Configure Rs Online to use the new filter coefficients
EST_setRsOnLineFilterParams(obj->estHandle, EST_RsOnLineFilterType_Current,
                            pfilter_i0->b0, pfilter_i0->a1, pfilter_i0->y1,
                            pfilter_i1->b0, pfilter_i1->a1, pfilter_i1->y1);
EST_setRsOnLineFilterParams(obj->estHandle, EST_RsOnLineFilterType_Voltage,
                            pfilter_v0->b0, pfilter_v0->a1, pfilter_v0->y1,
                            pfilter_v1->b0, pfilter_v1->a1, pfilter_v1->y1);
```

监控 Rs 在线电阻值

可使用两种方法在观察窗口中或在全局变量中检查 Rs 在线电阻值。 可以根据用户的要求选择一种方法来检查值，因此需要设置温度阈值，或者只需要确保已将电机正确连接到系统。

Rs 在线浮点值

第一种方法只需要调用一个可返回浮点值的函数。 电阻值的单位为欧姆 (Ω)，使用示例如下：

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;
float_t RsOnLine_Ohm = EST_getRsOnLine_Ohm(obj->estHandle);
```

这种方法对于一般监控是很方便的。 但是，这种方法需要执行一些浮点指令，需要花费大量执行时间，因而不是最有效的方法。

Rs 在线定点值

监控 Rs 在线值也可以使用一种更为具体的方法，即仅使用定点数学运算和移位操作。 以下代码示例显示了如何根据 InstaSPIN-FOC 中所提供的函数计算以欧姆为单位的 Rs 在线定点值。 由于避免了浮点数学运算，执行时间得到优化，甚至可以在中断程序中计算这个值。

```
#define VarShift(var,nshift) (((nshift) < 0) ? ((var)>>(-(nshift))) \
                                             : ((var)<<(nshift)))
#define USER_IQ_FULL_SCALE_VOLTAGE_V (48.0)
#define USER_IQ_FULL_SCALE_CURRENT_A (40.0)

CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;
uint_least8_t RsOnLine_qFmt = EST_getRsOnLine_qFmt(obj->estHandle);
_iq fullScaleResistance = _IQ(USER_IQ_FULL_SCALE_VOLTAGE_V \
                              /USER_IQ_FULL_SCALE_CURRENT_A);
_iq RsOnLine_pu = _IQ30toIQ(EST_getRsOnLine_pu(obj->estHandle));
_iq pu_to_ohms_sf = VarShift(fullScaleResistance, 30 - RsOnLine_qFmt);
_iq RsOnLine_Ohms = _IQmpy(RsOnLine_pu, pu_to_ohms_sf);
```

使用 Rs 在线特性作为温度传感器

这一独有特性可用于监控电机旋转时的电阻，从而使用户能够根据电阻增量来监控线圈的温度。 为了给出实现温度传感器的示例，请考虑表 15-1 中的值。

表 15-1. 温度传感器实现值

参数	值	说明
R	12.0Ω	温度 T 时的电阻值。可通过 Rs 在线特性得出此值。
R_0	10.0Ω	温度 T0 时的电阻值。 可通过 Rs 离线特性得出此值。
α	0.00393ºC^{-1}	材料的温度系数，本例中使用的材料为铜。
T_0	20ºC	材料的参考温度
T	?	材料的最终温度 可根据 Rs 在线特性计算得出。

启用 Rs 在线特性后，假设电阻从 10.0Ω 增加到 12.0Ω。电机绕组的温度可以根据以下公式计算得出，该公式通过之前章节中的公式导出：

$$T = T_0 + \frac{\frac{R}{R_0} - 1}{\alpha}$$

$$T = 20°C + \frac{\frac{12.0\Omega}{10.0\Omega} - 1}{0.00393°C^{-1}}$$

$$T = 70.89°C \tag{88}$$

以下示例代码显示如何实现温度监控器：

```
#define COPPER_TEMP_COEF_INV_C (0.00393)
#define RS_AT_ROOM_TEMP_OHMS          (10.0)
#define ROOM_TEMP_C                   (20.0)

// Derived defines, pre-calculated by the compiler, not the CPU

#define INV_COPPER_TEMP_COEF_C          (1.0/COPPER_TEMP_COEF_INV_C)
#define INV_RS_AT_ROOM_TEMP_INV_OHMS    (1.0/RS_AT_ROOM_TEMP_OHMS)

CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;
float_t RsOnLine_Ohm = EST_getRsOnLine_Ohm(obj->estHandle);
float_t Temperature_C = \
                    (ROOM_TEMP_C) + \
                    (RsOnLine_Ohm * (INV_RS_AT_ROOM_TEMP_INV_OHMS) - 1.0) * \
                    (INV_COPPER_TEMP_COEF_C);
```

此代码示例可在除中断外的其它时间在后台执行。 由于温度变化要远远慢于 CPU 时序，因此执行时间并不重要。

通过使用 Rs 在线特性，用户即可为电机设置温度限制，从而避免系统受到损害和发生故障。 为了简化温度计算，建议使用查询表，以避免实时执行此公式的时间损失。

Rs 在线特性相关状态框图（CTRL 和 EST）

在整个文档中存在多处对控制器 (CTRL) 和估算器 (EST) 状态框图的参考。 在本节中所显示的两种状态机均可用作参考。 有关电机识别过程的详细信息，请参见Chapter 6。

PowerWarp™

FAST 的 PowerWarp 算法可自适应降低电流消耗，从而最大限度地将组合（转子和定子）铜损耗降到最低，同时不影响交流感应电机的输出功率级。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

16.1 概述

对于要求最低功耗的应用，解决方案为使用交流感应电机 (ACIM) PowerWarp。 通过在 InstaSPIN-FOC 中简单设定一个使能标志，FAST 估算器即可重新计算磁化电流，这样便可使用最小电流来生成指定负载和速度所需的转矩。

- PowerWarp 算法是 InstaSPIN-FOC 的一项功能，此项功能被设计用来改进部分负载条件下的感应电机效率
- 请注意，输出功率在 PowerWarp 算法启用时保持不变

PowerWarp 在部分负载条件下对电机效率的影响最大。 然而，由于电机消磁，控制系统对意外瞬态情况的响应能力也会下降。 但这种情况下，磁通角跟踪不受影响

如图 16-1 中所示，当 PowerWarp 启用时，它作用于 FAST 估算器所提供的磁化电流（图中突出显示）。该磁化电流在图表中被称为 I_{rated}。

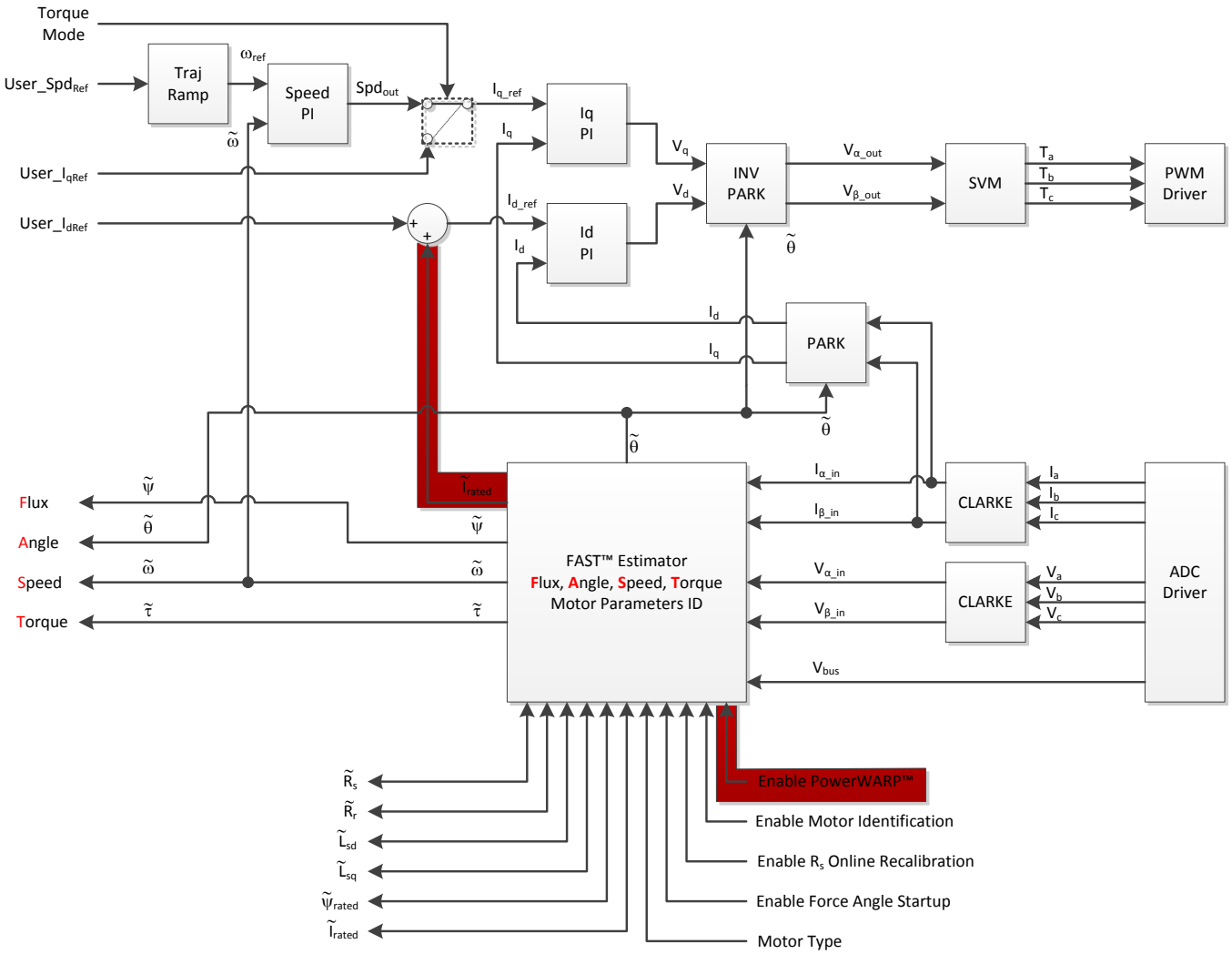


图 16-1. 采用 PowerWarp 的 FAST 估算器

16.2 启用 PowerWarp

代码中所有与 PowerWarp 有关的函数均称为 POWERWARP。 以下代码示例显示如何启用此标志。 如下所示，函数名称中提及与 PowerWarp 有关的 POWERWARP。

```
CTRL_setFlag_enablePOWERWARP(ctrlHandle,TRUE);
```

使用此标志不会产生任何影响，除非控制器和估算器在线运行，即不识别电机而是使其在闭环中运行。 另一个条件是电机类型必须是交流感应电机。 如果电机是 PM 电机，则不会产生任何效果。 因此，通过总结 PowerWarp 启用后能够生效的各种情况，必须满足以下条件：

- 控制器处于运行状态并且电机在闭环中。
- 估算器处于运行状态并且电机已被识别。
- 电机类型必须为感应电机。

在图 16-2 中，InstaSPIN 控制器的突出显示状态即电机在闭环中运行的部分。 此状态也称为在线状态，且该状态下可执行 PowerWarp 算法（有关 CTRL 和 EST 状态的完整信息，请参见Chapter 6）。

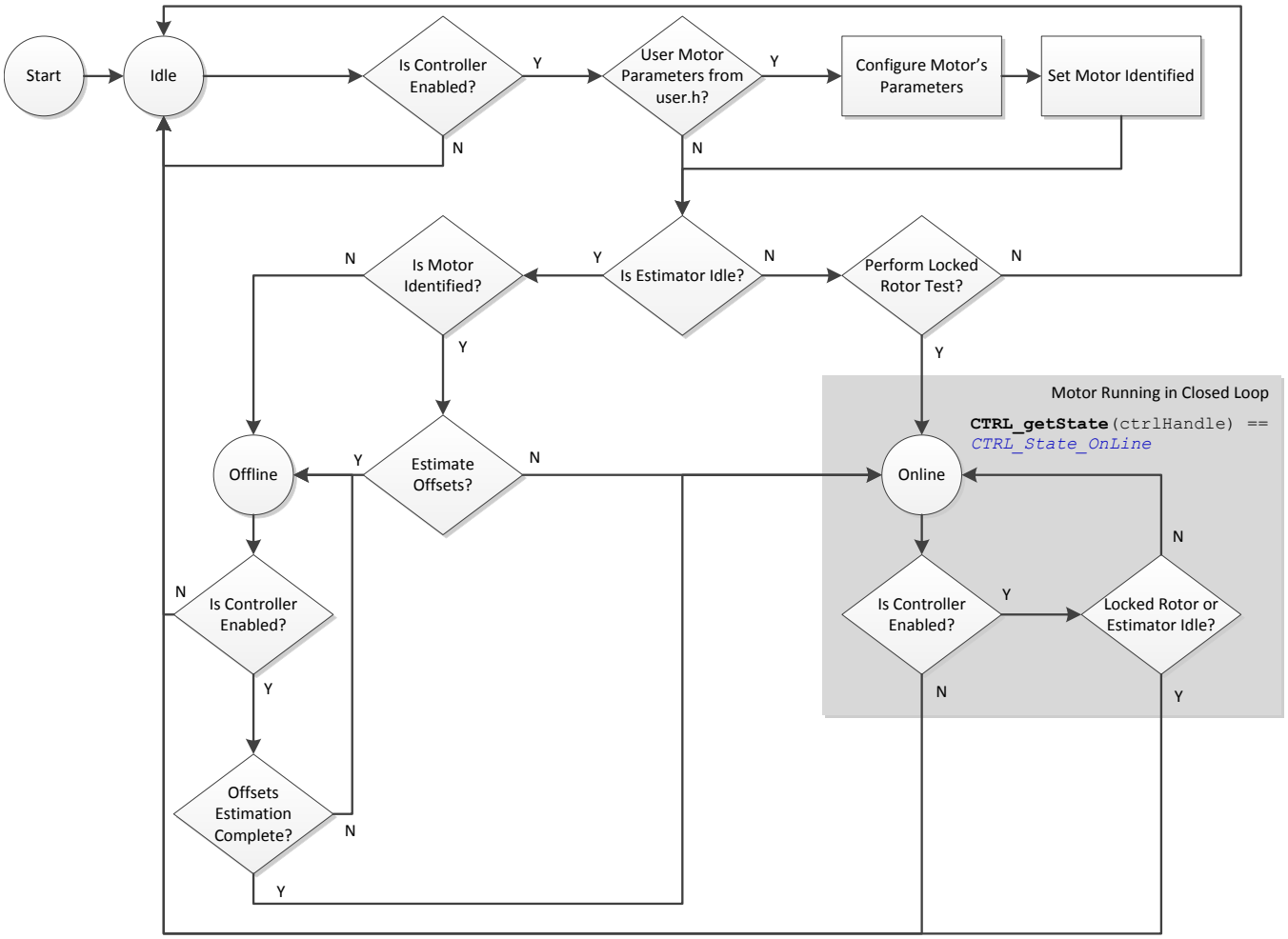


图 16-2. InstaSPIN 控制器流程图 - 在闭环中执行 PowerWarp

类似地，以下状态机中突出显示了可执行 PowerWarp 算法的估算器状态机。 图 16-3 中所示的状态机表示电机在闭环中运行的状态（从估算器的角度来看）。

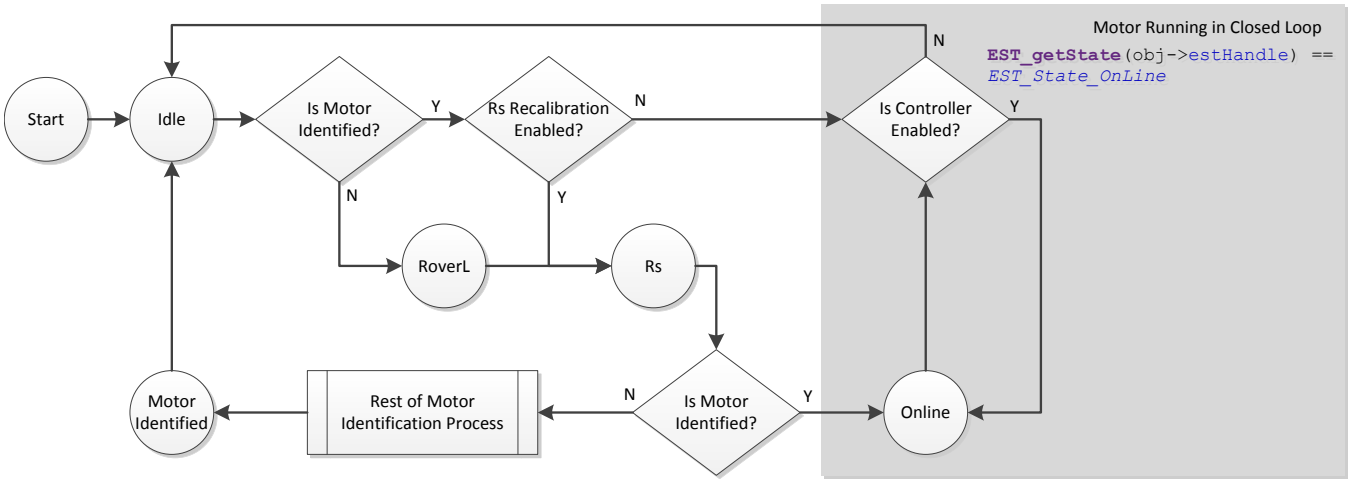


图 16-3. FAST 估算器状态机流程图 - 在闭环中执行 PowerWarp

此代码示例显示了如何检查状态机状态（包括控制器 (CTRL) 和估算器 (EST) 状态机）以及电机类型：

```
CTRL_Obj *obj = (CTRL_Obj *)ctrlHandle;
ctrlState = CTRL_getState(ctrlHandle);
estState = EST_getState(obj->estHandle);
motorType = CTRL_getMotorType(ctrlHandle);

if( (ctrlState == CTRL_State_OnLine) &&
   (estState == EST_State_OnLine) &&
   (motorType == MOTOR_Type_Induction) )
  {
   CTRL_setFlag_enablePOWERWARP(ctrlHandle,TRUE);
  }
```

16.3 PowerWarp 电流斜率

为了在额定磁化电流和 PowerWarp 算法给出的感生电流之间实现平稳转换，启用和禁用此模式时将生成线性转换。 启用 PowerWarp 时，可通过 user.h 中定义的以下代码设置电流变化率：

```
#define USER_MAX_CURRENT_SLOPE_POWERWARP
(0.3 * USER_MOTOR_RES_EST_CURRENT
/USER_IQ_FULL_SCALE_CURRENT_A
/USER_TRAJ_FREQ_Hz)
```

启用 PowerWarp 后的电流变化率等于估算电阻所使用的电流乘以 0.3Hz。 例如，估算电阻所使用的电流是 1A，则采用 PowerWarp 时 ACIM 电机额定电流变化率为：0.3A/s。 因此，当 PowerWarp 将额定电流由 3A 降至 1.5A 时，需要耗费 (3-1.5)/0.3 = 5 秒钟才能达到新的额定电流。

以下定义了另一种电流斜率，适用于在软件中更改了 I_{rated} 时或禁用 PowerWarp 后。

```
#define USER_MAX_CURRENT_SLOPE
```

```
(USER_MOTOR_RES_EST_CURRENT
/USER_IQ_FULL_SCALE_CURRENT_A
/USER_TRAJ_FREQ_Hz)
```

默认情况下，此电流斜率设置为测量电阻时每秒使用的电流。 例如，测量电阻所使用的电流为 1A，则 PowerWarp 电流为 1A，如果 I_{rated} 电流为 3A，则电流返回 I_{rated} 将耗费 (3 – 1)/1 = 2 秒钟。

16.4 实例

下图所示为 PowerWarp 的实例和此模式相关的节能效果。 测试电机参数如下：

直流感应异步机器 (GE 5K33GN2A)

- 额定功率：¼ Hp
- 额定转矩：9.22 Lb.in
- 额定电压：208~230 (V)
- 额定满负载电流：1.3~1.4 (A)
- 额定满负载速度：1725rpm
- 极对数：2
- 频率：60Hz

注意：使用 PowerWarp 后，电机效率将从 27% 大幅提升至 68%（负载转矩为 1 lb.in 时）。 由于 PowerWarp 会降低 ACIM 电机产生转矩的能力，因此需要较大转矩时，使用 PowerWarp 的节能效果也会下降。 同样，在额定转矩下，启用或禁用 PowerWarp 时效率曲线相同。

尽管 PowerWarp 会降低产生转矩的能力，但启用 PowerWarp 算法可维持提供给轴的机械功率，不会对机械系统性能产生影响。 换言之，从纯机械角度来看，启用 PowerWarp 后，输出的机械转矩和速度不会发生改变。 发生改变的是提供给电机以产生特定机械输出的电功率。

如图 16-4 所示，电机效率在低负载时显著提高，尽管控制系统会保持稳定，但动态转矩和速度响应会有所折衷。

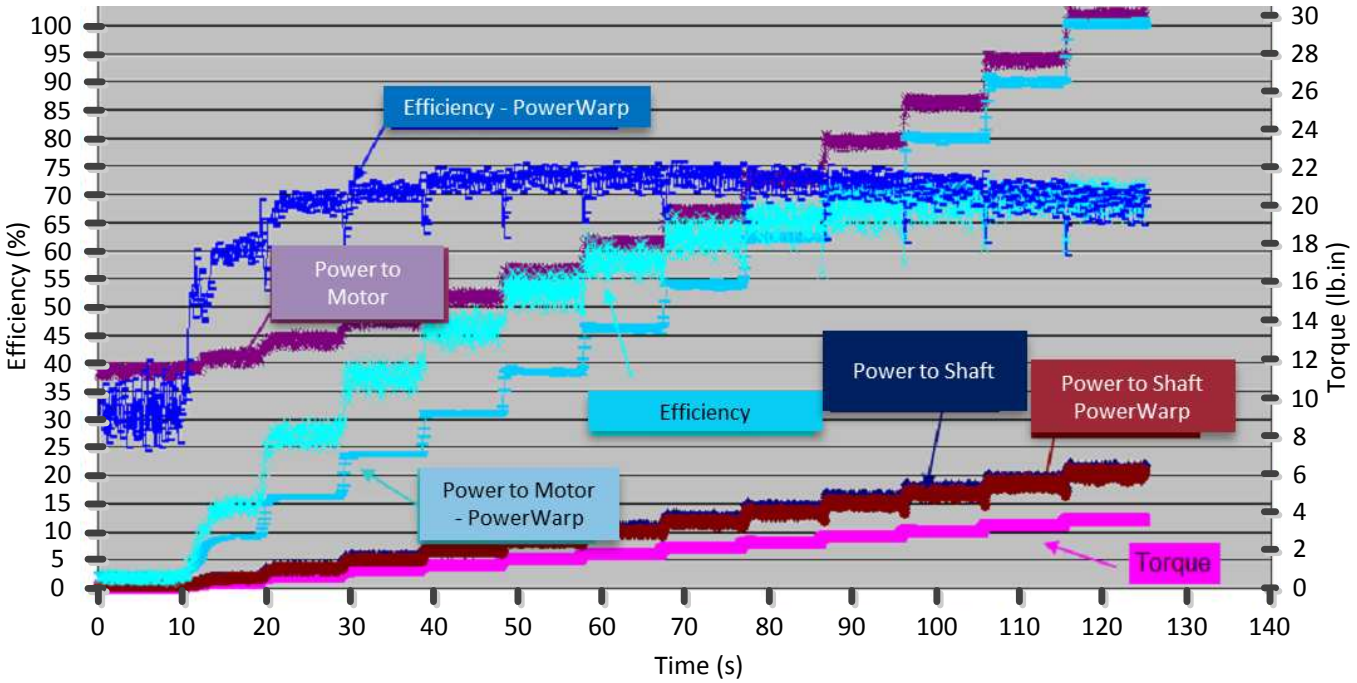


图 **16-4. PowerWarp** 提升电机效率

本实例使用默认电流斜率。

```
#define USER_MAX_CURRENT_SLOPE_POWERWARP (0.3 * USER_MOTOR_RES_EST_CURRENT \
                                    / USER_IQ_FULL_SCALE_CURRENT_A \
                                    / USER_TRAJ_FREQ_Hz)
```

图 16-5 显示启用 PowerWarp 时 FAST 估算器降低额定电流所耗时间。

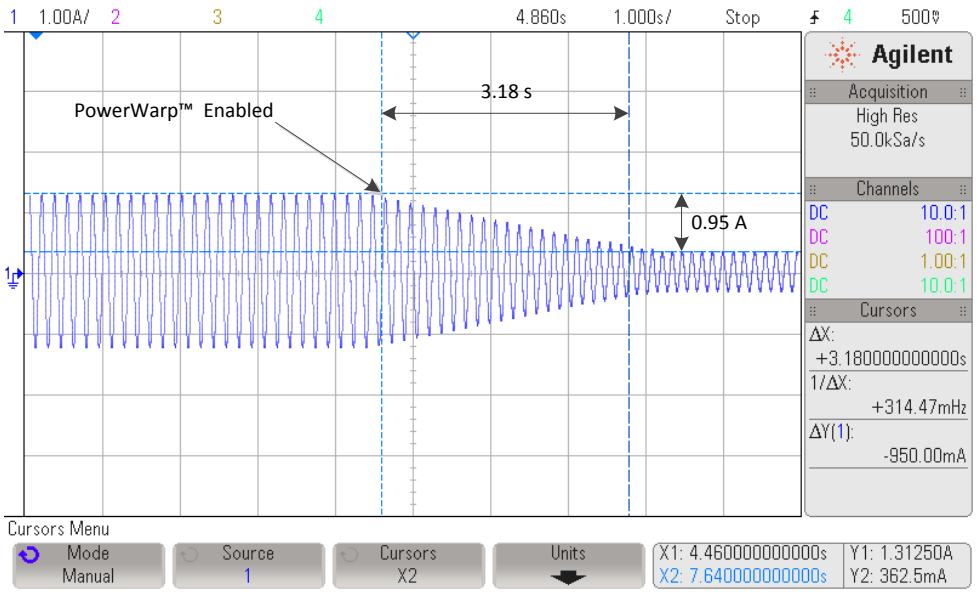


图 **16-5.** 启用 **PowerWarp** 时电流下降

如图所示，达到最低电流所耗时间为：

$$(I_{rated} - I_{PowerWarp})/0.3 = (1.3125 - 0.3625)/0.3 = 3.167 \text{ 秒} \quad (89)$$

同样，禁用 PowerWarp 时，电流斜率也采用默认值。

```
#define USER_MAX_CURRENT_SLOPE
(USER_MOTOR_RES_EST_CURRENT
/USER_IQ_FULL_SCALE_CURRENT_A
/USER_TRAJ_FREQ_Hz)
```

图 16-6 显示此斜率。 所耗时间为：(1.3125 – 0.3625)/1 = 1.0 秒。

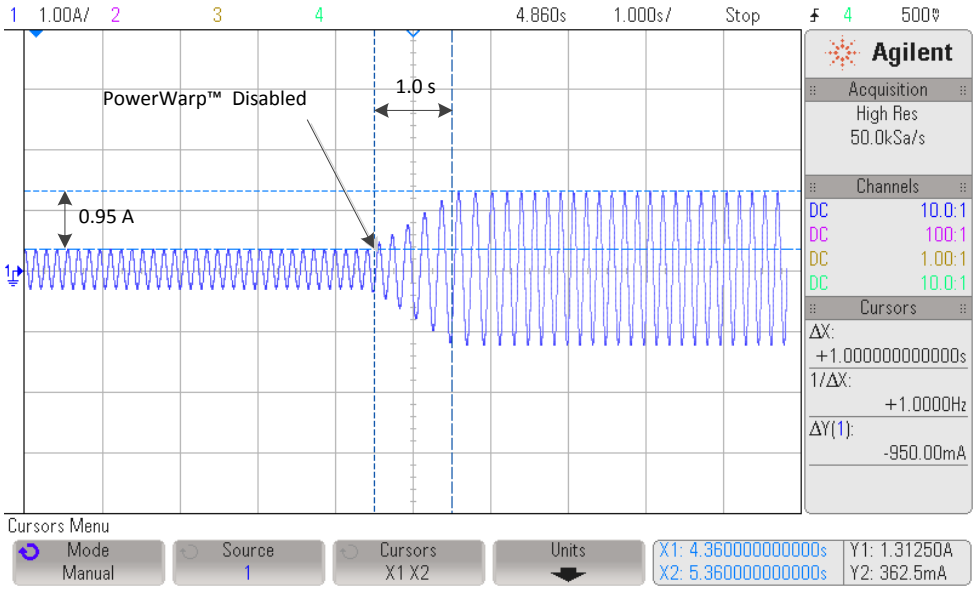


图 **16-6.** 禁用 **PowerWarp** 时的电流斜率

16.5 案例研究

有两对电机运行相同负载（本例中为风扇）。 一对电机不间断运行 15 个月，然后对比启用 PowerWarp 算法的 InstaSPIN-FOC 和通过晶闸管 (TRIAC) 控制的感应电机的能耗性能。 使用启用了 PowerWarp 算法的 InstaSPIN-FOC 后，节能效果随时间显著增加，平均节能约达 81%。 换言之，启用 PowerWarp 算法后，每台电机的能耗与使用 TRIAC 控制器相比仅占其 19%。 此百分比计算方法如下：

TRIAC 控制器的总能耗 = 2096.6kWh (90)

图 16-7 所示为每月的能耗和累积的节能总量（单位 kWh）。

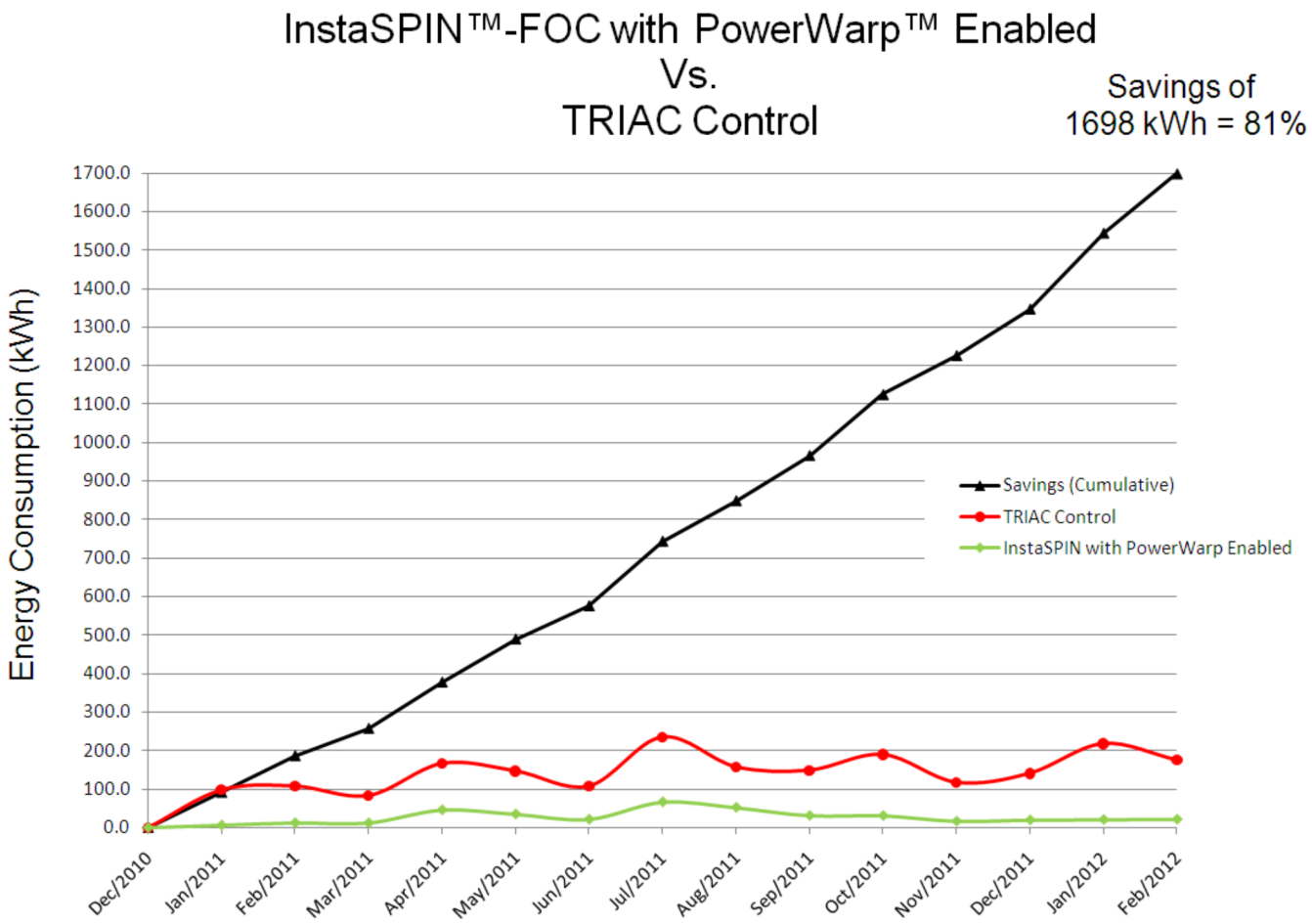


图 16-7. 启用 PowerWarp 算法与 TRIAC 控制感应电机

系统第二对电机在启用 PowerWarp 的条件下运行 InstaSPIN，但该组现在与 Volts-per-Hertz 控制（也称为频率控制）进行比较。 这种情况下节能效果也很显著，原因是 PowerWarp 优化了电流消耗从而将铜损耗降到最低。 此时平均节能为 48%，计算方式如下：

频率控制器的总能耗 = 916.5kWh

启用 PowerWarp 时的 InstaSPIN 总能耗 = 478.9kWh

平均节能 = 100% * (1 – 478.9/916.5) = 47.75%

图 16-8 显示上述结果。

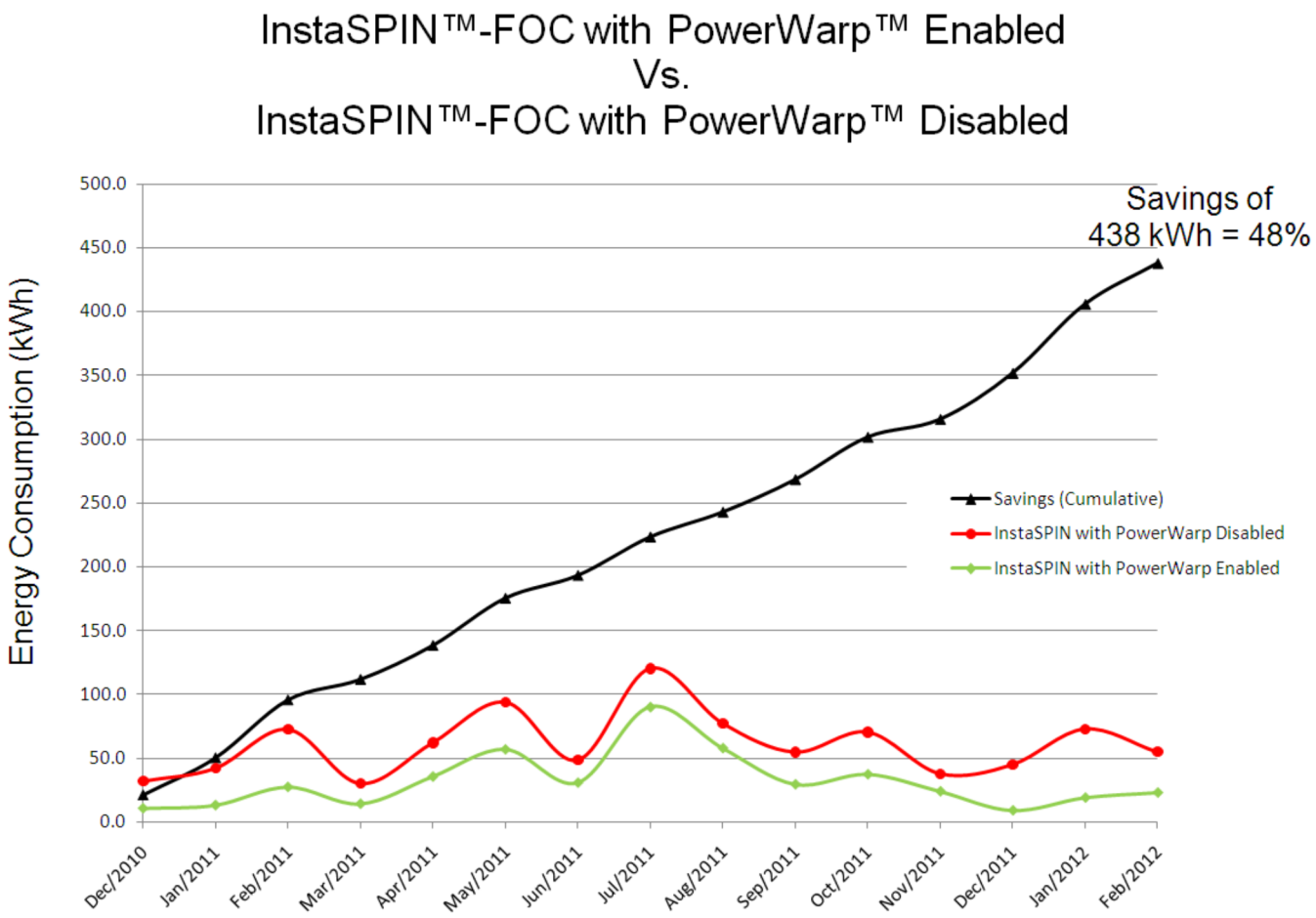


图 **16-8.** 启用 **PowerWarp** 时的 **InstaSPIN-FOC** 与 禁用 **PowerWarp** 时的 **InstaSPIN-FOC**

分流电流测量

本章介绍单分流、双分流和三分流电流测量技术。 我们将说明三分流技术不会大幅增加电流测量电路成本的原因， 以及电机控制应用使用三分流技术更有优势的原因。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

17.1 简介

InstaSPIN 可用于多种不同类型的电流测量技术。 这些技术包括 3 分流和 2 分流电阻器以及 LEM 传感器测量。 对于较低功耗的驱动器，分流电阻器是测量电机控制器中相电流的最常用方法。 目前，InstaSPIN 软件套件不提供单分流电阻器解决方案。 下文将介绍不提供单分流电流测量以及 3 分流测量是测量电流的理想方法的具体原因。

17.2 信号

在介绍不同类型的电阻器分流电流测量之前，我们将了解测量电流所涉及的开关信号以及具体的测量位置。 大部分磁场定向控制 (FOC) 驱动器使用空间矢量调制 (SVM) 技术向逆变器开关发送占空比命令，从而驱动电机。 典型 SVM 电压波形外观如图 17-1 中所示的蓝色信号。 接下来，SVM 波形被图 17-1 中红色信号的三角波形采样。 每当三角波大于 SVM 时，h 桥相位的下限开关打开。 所生成的 PWM 波形用于控制 h 桥相位，如下图所示。

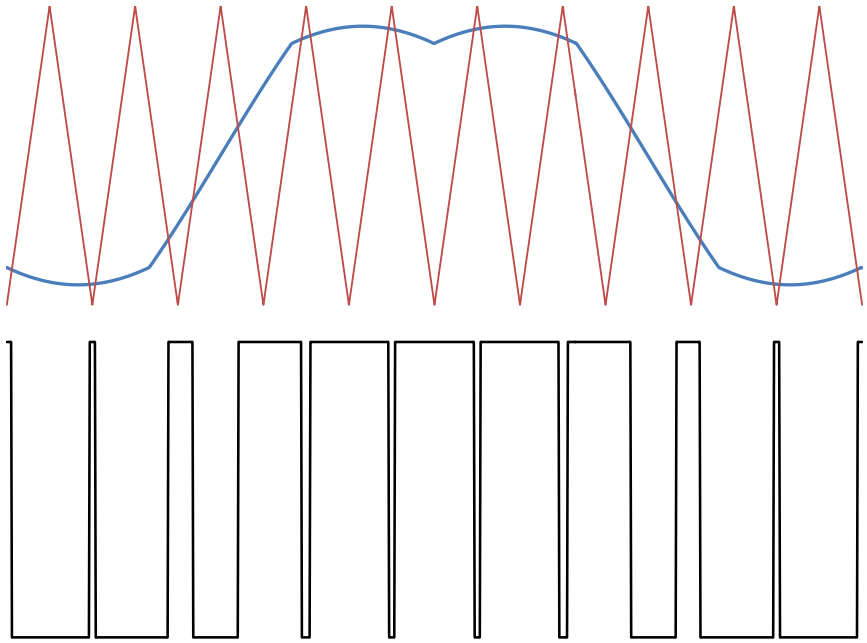


图 **17-1.** 通过计数器采样的典型 **SVM** 波形

要测量流经某个相位的电流，常用方法是在该相位底部放置一个电阻器。 这样，无论采用何种电阻器配置，只有在下限开关打开时才能测量 1 分流、2 分流或者 3 分流电流。 在 PWM 波形中，当方波信号为低电平时，下限开关打开。 如要对电流采样，信号必须干净。 干净的电流信号或者电流信号表示必须不含任何振铃或噪声。 接下来介绍现场应用的不同电阻器分流电流测量技术。

17.3 1 分流

单分流电流测量技术可测量电源电流，并且根据开关状态分别重新生成电机的三相电流。图 17-2 说明了单分流电阻在逆变器电路中的位置。

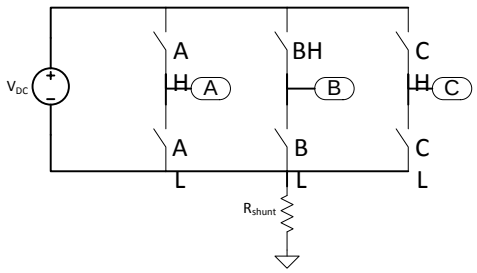


图 **17-2.** 逆变器单分流电流测量电路

SVM 中有八个不同的开关选项。表 17-1 解释了每一种选项并显示了电压空间矢量的方向以及每种状态下可测量的电流。 当开关状态为 0 和 7 时，仅存在循环电流，因而不能使用单分流技术测量电流。 要合理使用单分流技术测量电流，必须同时考虑电流测量和开关状态。

表 **17-1.** 八种 **SVM** 开关状态

开关状态	AH	BH	CH	矢量	测量
0	0	0	0	•	偏移
1	1	0	0	→	Ia
2	1	1	0	↗	-Ic
3	0	1	0	↖	Ib
4	0	1	1	←	-Ia
5	0	0	1	↙	Ic
6	1	0	1	↘	-Ib
7	1	1	1	•	偏移

图 17-3 显示了 SVM PWM 波形和生成的电流测量信号。 这种情况下，I_C 和 I_A 的电流传导时间足够长，这样运算放大器的转换率和整个测量系统的稳定时间将具有足够长的时间进入稳定状态，以便 ADC 有足够时间对电流采样。 如下文所述，使用单分流技术时，必须能够在允许的最短时间内测量电流。

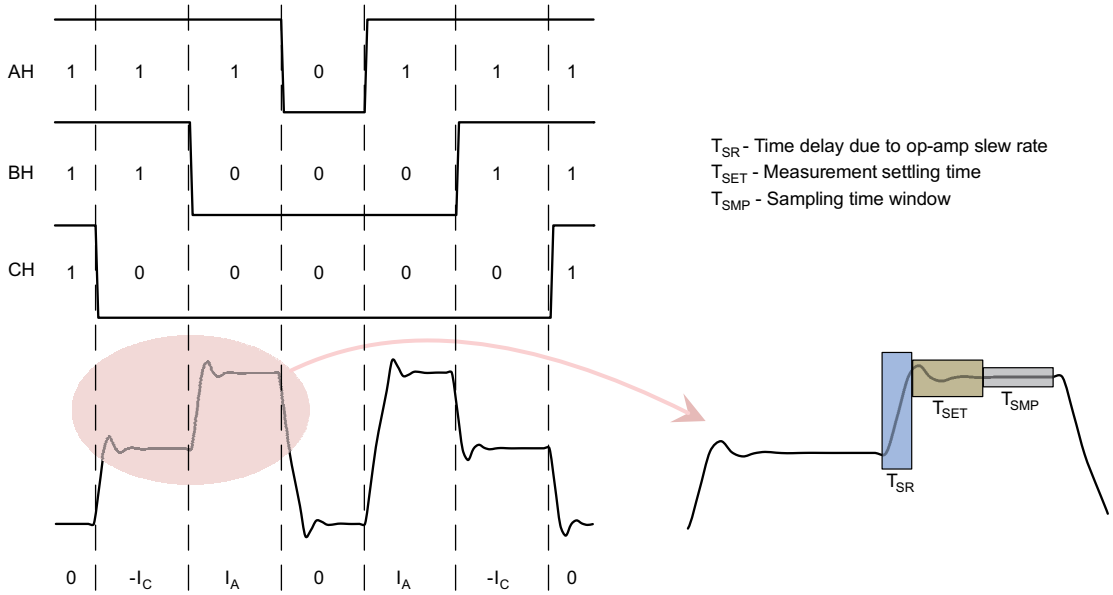


图 **17-3.** 采样时间足够长时的单分流电流测量技术

在图 17-4 中，想象电压空间矢量绕圆圈逆时针行进。 当空间矢量指向六边形的六个角时，电流采样对应的时间窗口将完全消失。 在位于 0、60、120、180、240 和 300 度的区域内，只有一个电流可测量，其它两个电流必须通过另一种方式才能找到。

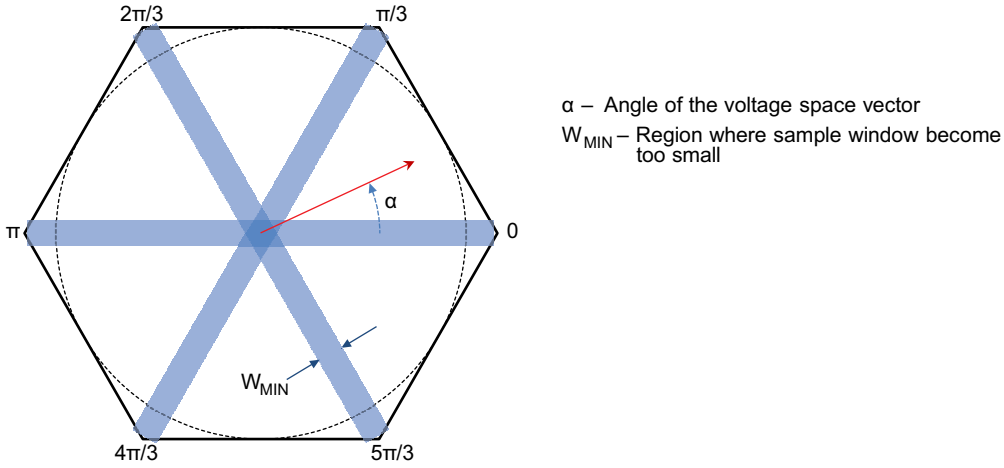


图 **17-4. SVM** 和不允许测量电流的区域

在图 17-5 中，空间矢量的指向非常接近 π/3，这会引起 I_A 的电流测量窗口收缩。 由于运算放大器的转换率和较长的稳定时间，I_A 将被忽略并导致 FOC 控制器出错。 解决此问题的一个方法是强制一个测量窗口保持足够长时间的打开状态，以便适应转换率和稳定时间。 此方法的相关技巧说明如图 17-6 所示。 最大占空比波形右移，最小占空比波形左移。 这类 PWM 相移的优势是确保每个相位的电压波形不会失真。 软件仍需对生成的电流波形进行补偿，即使电流测量窗口可根据需要调节大小，但最好还是保持该窗口尽可能小。 当空间向量达到直流总线设置的电压限制时，信号相移可用的空间将很少。 因此，为获得直流总线的最佳利用率并且能够稳定测量电流，要求所选的运算放大器具有超高转换率和较短的稳定时间。

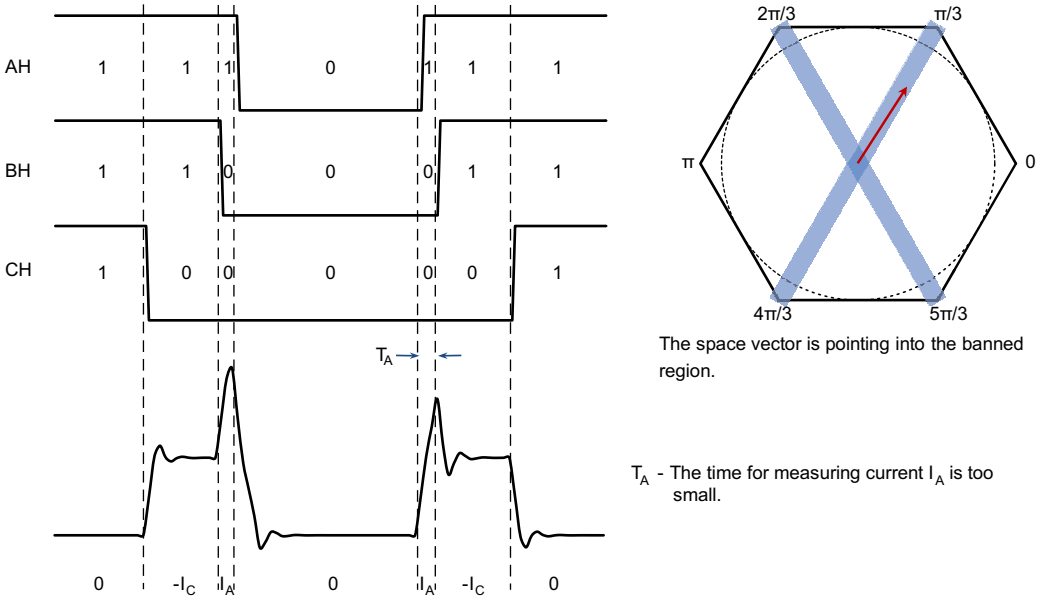


图 **17-5.** 电流采样窗口消失时的示例

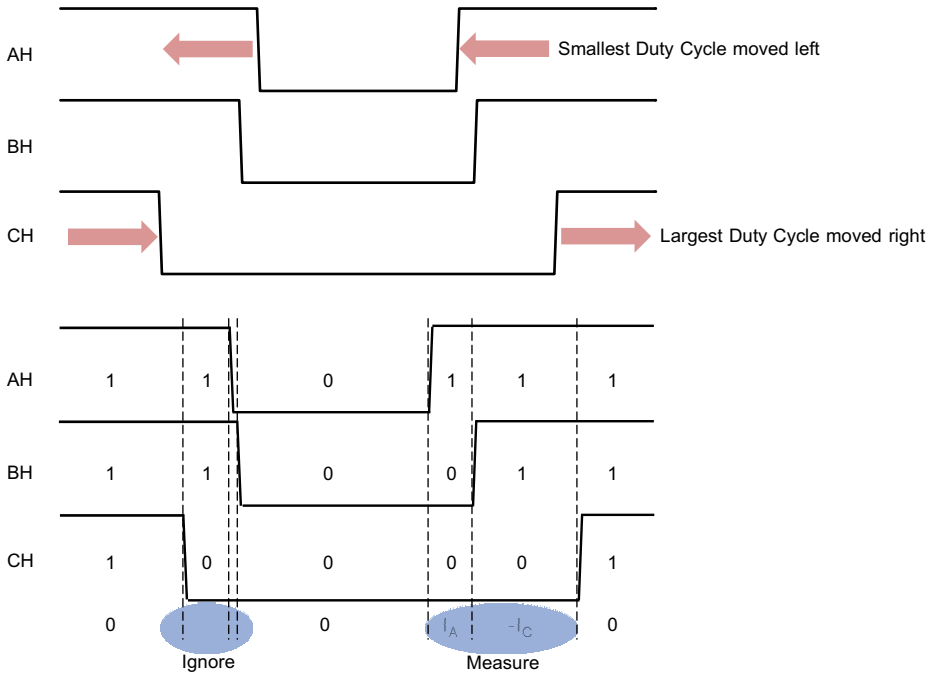


图 **17-6.** 通过 **PWM** 相移获得足够大的电流测量窗口

电流波纹是使用单分流技术时遇到的另一个问题。 电机是电感和电阻电路元件，因此具有一个 R/L 时间常量。 在图 17-3 所示的电流波形中，电机的电子时间常量较大，这会导致电流达到特定水平，并且测量电流 I_C 和 I_A 可被视为流入电机的平均电流。 电机的 R/L 时间常量较小时，电流波形看起来更像锯齿波。 此时，必须在尽可能接近总传导时间中间点的位置对电流采样，这样才能获得电机平均电流。 这一过程将对所选运算放大器提出更加严苛的性能要求。

下面通过快速计算来了解需要哪些类型的运算放大器参数。 首先，正常的 PWM 频率在 20kHz 左右，周期为 50µs。 在 20kHz 时，当进入死区时间或者任何 PWM 非对称调整的时间为 0.5µs 或更长时间时，将出现电流失真。 C2000 F2805xF 和 F2806xF 系列处理器具有 90MHz 时钟速度，能够转换为 45MHz ADC 时钟。 最小采样窗口为 7 个 ADC 时钟周期或者 156ηs。 考虑转换率延迟时，最差情况下的延迟是在 3.3V 最大电压转换期间。忽略稳定时间，可保持信号测量低于 0.5µs 的转换率为 3.3/344ηs 或 9.6V/µs。 稳定时间约占此时间的一半，因此，为安全起见，运算放大器转换率应选择 20V/µs。

17.4 2 分流

双分流测量技术采用基尔霍夫电流定律 (KCL) 的原理，即，流入节点的电流之和等于零。 仅通过测量两相电流，即可利用 KCL 计算出第三相电流。 双分流电流测量技术的电路如图 17-7 所示。

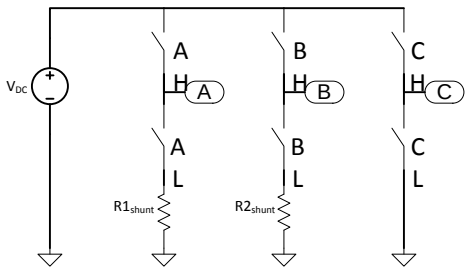


图 **17-7.** 逆变器双分流电流测量电路

相比单分流电路，双分流测量电路的优势在于可看到循环电流。 此时，所有的电流仅在开关状态为零时测量。图 17-8 显示开关波形以及 ADC 电流采样的位置示例。 I_A 的 PWM 接近 100% 占空比，在本例中，这将导致 I_A 电流升高。 I_B 的 PWM 占空比约为 50%，在此期间，其电流值保持在零安培左右。 相电流仅在此相的下限开关导通时才可测量。 在本例中，I_A 仅在很短的时间内可测量，然而 I_B 则可进行长时间观察。 当测量的相位以接近 100% 占空比在 PWM 运行时，这是使用双分流技术的固有问题。 例如，在对 I_A 进行采样时，测量的电流信号尚不稳定，因而给出了不正确的电流信号表示。 在电机运行期间使用双分流技术时需注意的另一点是：目前所测电流为双极。 因此，零安培此时代表 ADC 满量程的一半。

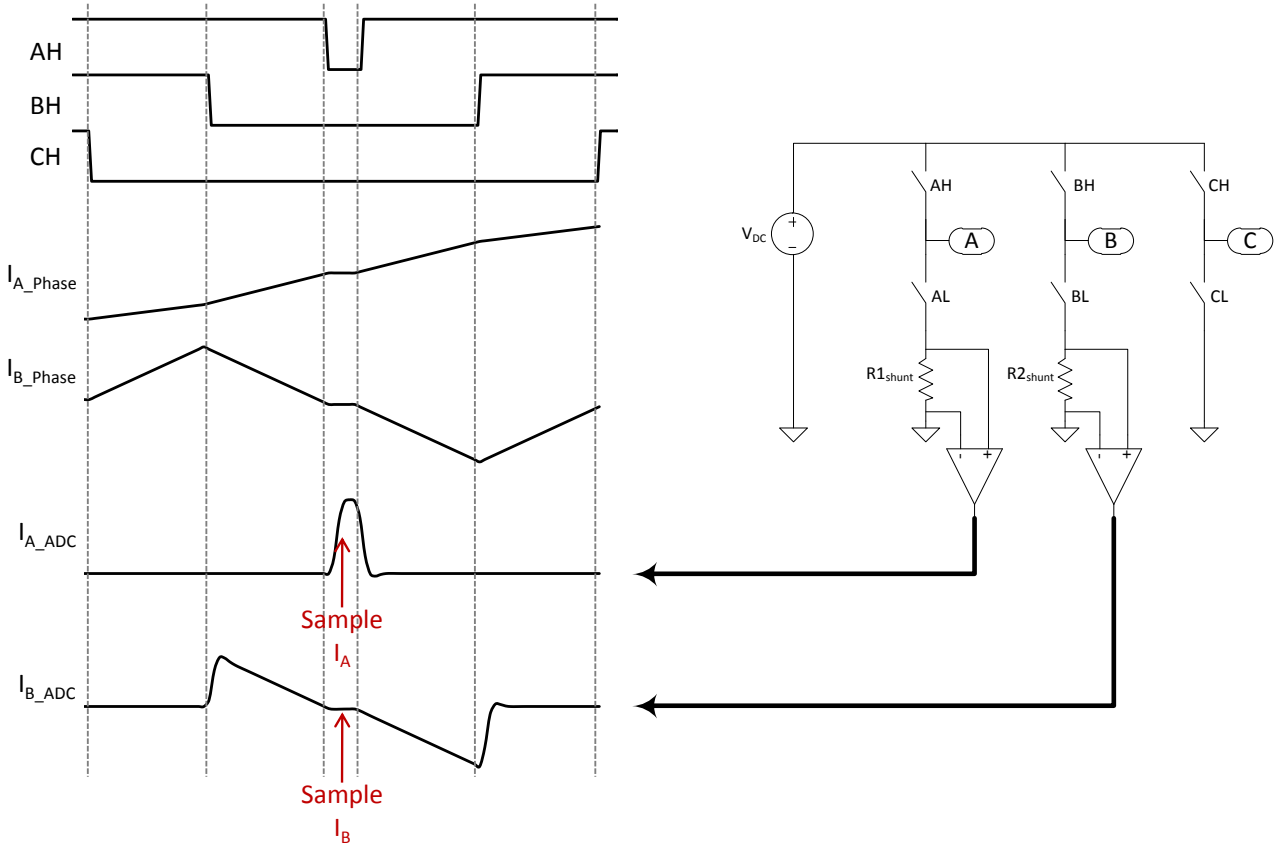


图 17-8. 使用双分流测量技术时的电流采样

随着占空比增加，测量相位分流电阻器两端电压所需的时间要更快。 例如，如果推荐 20kHz PWM 波形使用 95% 的占空比，则测量相位所需的导通时间为 2.5µs。 理想条件下，运算放大器转换率应为该导通时间的 1/10（即 0.25µs）。 运算放大器的满量程输出电压转换为 1.65V。最小转换率经计算为 6.6V/µs。 随着占空比进一步增大，转换率必须相应增大才能准确地捕捉信号。 相比单分流技术，尽管双分流技术可以降低运算放大器的速度要求，但在某个占空比时仍需要很大的转换率。 对于单分流或双分流测量技术而言，采用昂贵的快速运算放大器是无法避免的。

17.5 3 分流

三分流测量电路示例如图 17-9 所示。 三分流测量技术非常稳健，即使与单分流或双分流测量技术相比，也具有惊人的成本效益。 首先，使用单分流和双分流电路时很难实现过调制。 其次，单分流和双分流技术必须选择价格更高的快速转换率运算放大器。 三分流技术可以在电流信号间进行跳跃采样，即，在各个周期内选择三相中的两相，这样可为电流信号提供较长的稳定时间。 如果可以实现大电流测量窗口，则可以使用速率更慢且更廉价的运算放大器。 例如，图 17-10 显示了三个 PWM 开关信号和要采样的分流电阻器。 如图所示，电流信号有很长时间来达到稳定。

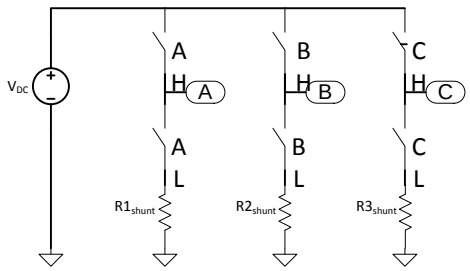


图 **17-9.** 逆变器三分流电流测量电路

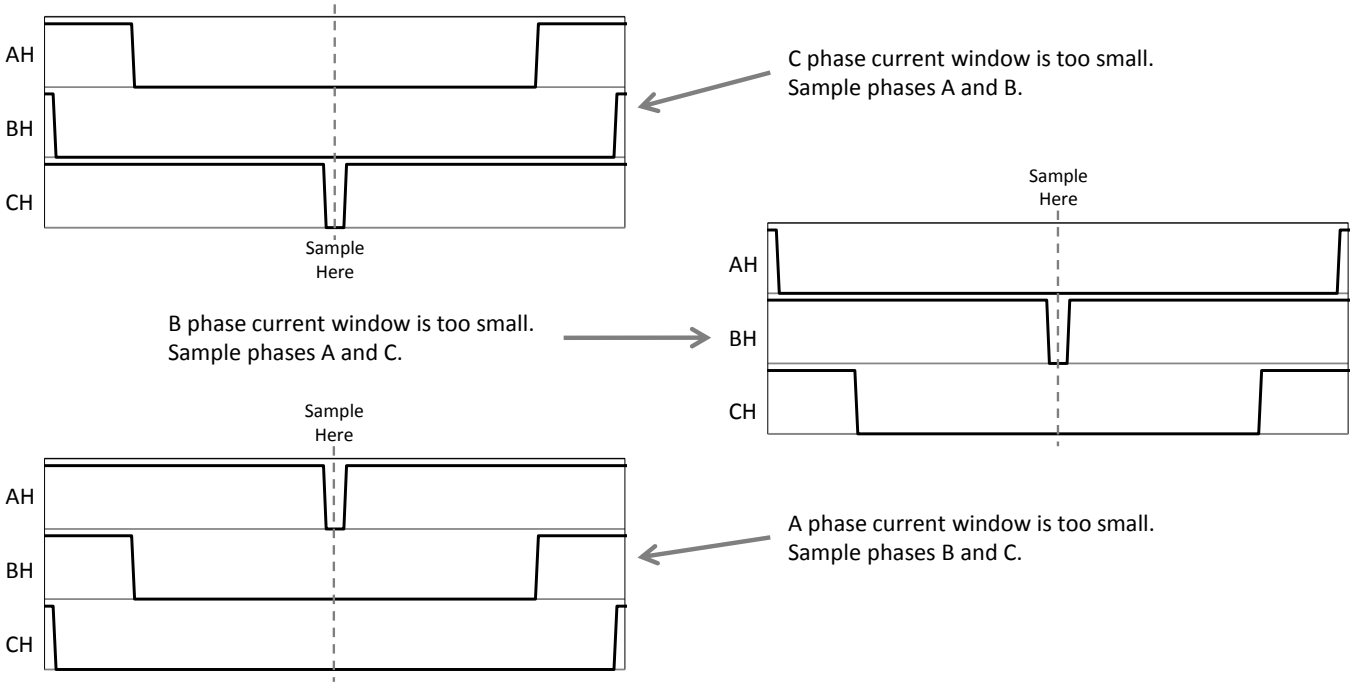


图 **17-10.** 使用三分流采样技术

使用三分流技术时，运算放大器的转换率可以低于 1V/µs。 使用低速放大器电路对于电流测量有诸多优势。 第一，成本较低。 第二，低速放大器提取噪声所用的带宽较低。 第三，在很多电流放大器电路中，相电流测量之间会发生串扰。 这可能会导致 C 相位开关中的尖峰显示在 A 相电流测量中。 低速放大器电路可以减弱串扰信号。

17.6 开发套件

TI 提供三个套件，其中的硬件支持单分流、双分流和三分流电流测量。表 17-2 列出了每个套件对应的电流和电压额定值。

表 **17-2. TI** 开发套件的电流和电压额定值

套件	电流额定值 (A)	电压额定值 (V)
DRV8312	6.5	50
DRV8301	82.5	50
HVKIT	10	350

下文将介绍套件的相关原理图并说明它们可用于三种不同电流测量技术的原因。

17.6.1 DRV8312

DRV8312 套件是所有 TI 套件中功耗最低的。 它使用 TI DRV8312 电源模块，该模块包含六个电源 MOSFET 和相应的栅极驱动器。 该套件中可插入任何 C2000 100 引脚 DIMM 卡（控制卡）。 DRV8312 的独有特性是它可以维持高达 500kHz 的开关频率，同时保持非常高的效率。

关于后续介绍，请参见《*Stellaris® DK-LM3S-DRV8312 开发套件用户指南*》（文献编号 SPMU276）中的原理图。 OA3、OA4 和 OA2 是分别为相位 A、B 和 C 组成 Kelvin 电流连接的运算放大器。 此电路的差分增益为 19.08V/V。R52、R53 和 R50 为每个相位对应的分流电阻器，每个大小为 10 mΩ。 导致 ADC 3.3V 输出的电流量为 3.3/.010/19.08 = 17.30A/V。此电流增益用于 InstaSPIN 的 user.h 文件中的参数 USER_ADC_FULL_SCALE_CURRENT_A。

要在 2 分流方法和 3 分流方法之间进行选择，请根据使用的电流分流数量将 user.h 参数 USER_NUM_CURRENT_SENSORS 设为 2 或 3。

17.6.2 DRV8301

DRV8301 套件为低电压高电流套件。 它使用由 DRV8301 栅极驱动器控制的离散 MOSFET。

17.7 结论

电阻器分流测量是测量电机控制逆变器中电流的非常合理的技术。 有三个广泛使用的示例，1 分流、2 分流和 3 分流电阻器测量。 尽管在最初看来 1 分流和 2 分流技术节约了成本，但是它们需要更快速且更昂贵的放大器电路（请参见表 17-3）。1 分流和 2 分流测量也限制了电流反馈的能力，这将限制驱动器使用提供给逆变器的满电压的能力。 3 分流技术很出色并且在成本方面并没有什么不同，这归功于使用廉价、低速电流放大器电路。

表 17-3. 针对相应检测电阻器数量的推荐运算放大器转换率

分流电阻器数量	运算放大器转换率
1	>20V/µs
2	>6V/µs
3	>1V/µs

传感系统

传感系统也可通过 InstaSPIN-MOTION 进行控制。 位置控制解决方案需依赖精确的电角才能实现对电机角度的控制。 尽管无传感器估算器在速度控制应用领域表现良好，但无传感器估算器无法为位置控制提供足够精确的电机角度。 下文给出的示例说明了如何使用正交编码器为 InstaSPIN-MOTION 提供电角度反馈以实现位置控制。

FAST 软件编码器仍可用于位置控制应用，充当备用编码器来检测主要电角源是否发生故障。

Topic **Page**

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

18.1 正交编码器的硬件配置

正交编码器需要使用 F2806xM 和 F2805xM 器件上的增强型正交编码器脉冲 (QEP) 外设。 电路板设计中需要将正交编码信号馈入微控制器。 如果使用的是 TI 评估套件，则设置的电路板确保能够在搭配正交编码器使用时正常工作。 要连接正交编码器，请参见特定评估套件的硬件手册。

18.1.1 引脚用量

QEP 外设可接受来自正交编码器的 A、B 和 I 输入。表 18-1 列出了每个 QEP 外设所需的引脚用量。 这些外设应与编码器上相应的引脚相连。

表 **18-1.** 将正交编码器连接到 **eQEP** 模块所需的引脚

引脚类型	引脚名	每个电机的引脚量
数字	EQEPxA	3
	EQEPxB	
	EQEPxI	

18.2 正交编码器的软件配置

此软件配置专注于通过 MotorWare 基础架构实现正交编码器的正常工作。 使用正交编码器的项目均需完成上述步骤。 实验 12b“使用 InstaSPIN-MOTION 控制传感系统”是一个示例项目，其中执行了使用正交编码器进行反馈时所需步骤。

18.2.1 针对 EQEP 操作配置电机

需要在 user.h 文件中定义一个附加参数。 此参数即 USER_MOTOR_ENCODER_LINES。 其值应设为电机编码器上的行数或（脉冲数）。 在实验 12b 示例项目中，此宏定义包含在 user.h 文件中。

```
#elif (USER_MOTOR == Teknic_M2310PLN04K)
#define USER_MOTOR_TYPE                 MOTOR_Type_Pm
#define USER_MOTOR_NUM_POLE_PAIRS       (4)
#define USER_MOTOR_Rr                   (NULL)
#define USER_MOTOR_Rs                   (0.4076258)
#define USER_MOTOR_Ls_d                 (0.0001972132)
#define USER_MOTOR_Ls_q                 (0.0001972132)
#define USER_MOTOR_RATED_FLUX           (0.03975862)
#define USER_MOTOR_MAGNETIZING_CURRENT  (NULL)
#define USER_MOTOR_RES_EST_CURRENT      (1.0)
#define USER_MOTOR_IND_EST_CURRENT      (-0.5)
#define USER_MOTOR_MAX_CURRENT          (7.0)
#define USER_MOTOR_FLUX_EST_FREQ_Hz     (20.0)
// Number of lines on the motor's quadrature encoder
#define USER_MOTOR_ENCODER_LINES        (1000.0)
```

18.2.2 初始化 EQEP 句柄

hal.c 中的 HAL_init 函数可初始化 QEP 驱动器的句柄。 这将为 QEP 句柄提供要修改的寄存器的位置。 在实验 12b 示例项目中，该步骤将在 hal.c 文件中完成。

```
// initialize EQEP handle
obj->qepHandle[0] = QEP_init((void*)QEP1_BASE_ADDR,sizeof(QEP_Obj));
```

18.2.3 设置数字 IO 以连接 QEP 外设

默认情况下，QEP 驱动器所使用的引脚均设为通用数字 IO。 这些引脚需在 hal.c 文件中的 HAL_setupGpios 函数中设置，以连接 QEP 驱动器。 在实验 12b 示例项目中，该步骤将在 hal.c 文件中完成。

```
// EQEPA
GPIO_setMode(obj->gpioHandle,GPIO_Number_20,GPIO_20_Mode_EQEP1A);

// EQEPB
GPIO_setMode(obj->gpioHandle,GPIO_Number_21,GPIO_21_Mode_EQEP1B);

//    EQEP1I
GPIO_setMode(obj->gpioHandle,GPIO_Number_23,GPIO_23_Mode_EQEP1I);
```

18.2.4 启用 eQEP 时钟

在 hal.c 文件中的 HAL_setupPeripheralClks 函数中，需启用所要使用的 QEP 驱动器的时钟。 这样 QEP 驱动器就能正常工作。 在实验 12b 示例项目中，该步骤将在 hal.c 文件中完成。

```
CLK_enableEqep1Clock(obj->clkHandle);    // Enable clock to eQEP Module
```

18.2.5 初始化 ENC 模块

ENC 模块用于将 QEP 驱动器生成的原始计数转换为 FOC 系统所要使用的电角。 需要在项目的主源文件中声明 ENC 模块对象和句柄。 在实验 12b 示例中，该操作在 proj_lab12b.c 中完成。

```
ENC_Handle encHandle;
ENC_Obj enc;
```

完成对 ENC 模块对象和句柄的声明后，需立即对其进行初始化。 该操作将对句柄进行分配，使其指向 ENC 模块所使用的特定存储器。 该步骤应在主源文件中的主循环前执行。

```
// initialize the ENC module
encHandle = ENC_init(&enc, sizeof(enc));
```

18.2.6 设置 ENC 模块

在执行主源文件中的主循环前，需设置 ENC 模块。 这样可将重要值传送到 ENC 模块，从而实现将来自 QEP 驱动器的原始正交计数转换为 FOC 可用的电角。

```
// setup the ENC module
ENC_setup(encHandle, hal_obj->qepHandle[0], 1, USER_MOTOR_NUM_POLE_PAIRS,
USER_MOTOR_ENCODER_LINES, 0, USER_IQ_FULL_SCALE_FREQ_Hz, USER_ISR_FREQ_Hz, 8000.0);
```

18.2.7 调用 eQEP 函数

在主 ISR 中，需要调用函数计算来自 EQEP 驱动器的转子角。 该函数被称为 ENC_calcElecAngle。 执行每个 ISR 实例时都需要进行此操作。

```
// compute the electrical angle
ENC_calcElecAngle(encHandle);
```

18.2.8 为 FOC 提供 eQEP 角度

需要在 ctrl.h 文件的函数 CTRL_runOnline_User 中修改为 FOC 提供的角度的来源。 默认情况下，FOC 设置为从 FAST 估算器获取角度，但在采用传感控制时，FOC 将接收来自 ENC 模块的角度。 在实验 12b 示例项目中，该步骤将在 ctrlQEP.h 文件中完成。

```
// generate the motor electrical angle
angle_pu = EST_getAngle_pu(obj->estHandle);

// Update electrical angle from ENC module
angle_pu = ENC_getElecAngle(encHandle);
```

18.3 InstaSPIN-MOTION 位置转换

SpinTAC™ 位置转换用于将 ENC 模块输出的电角转换为机械角和用于系统其它部分的速度反馈。 ENC 模块可产生用于 FOC 的转子电角。 SpinTAC 位置转换还能够估算交流感应电机的滑移速度。 使用带有物理传感器的交流感应电机时需要此功能。

18.3.1 SpinTAC 位置转换的软件配置

配置 SpinTAC 位置转换需要四个步骤。 实验 12b“使用 InstaSPIN-MOTION 控制传感系统”是一个示例项目，其中执行了使用 SpinTAC 位置转换所需的步骤。 利用 MotorWare 中的头文件 spintac_velocity.h，可以快速将 SpinTAC 组件纳入项目中。

18.3.1.1 包括头文件

该步骤应通过头文件包括的模块其余部分来完成。 在实验 12b 示例项目中，该文件包括在 spintac_velocity.h 头文件中。 针对相应项目，可通过包括 spintac_velocity.h 完成此步骤。

```
#include "sw/modules/spintac/src/32b/spintac_pos_conv.h"
```

18.3.1.2 声明全局结构

该步骤应通过主源文件中的全局变量声明来完成。 在实验 12b 项目中，此结构包括在已声明为 spintac_velocity.h 头文件一部分的 ST_Obj 结构内。

```
ST_Obj   st_obj;    // The SpinTAC Object
ST_HandlestHandle;  // The SpinTAC Handle
```

此示例为不希望使用已在 spintac_velocity.h 头文件中声明的 ST_Obj 结构的情况。

```
ST_PosConv_t      stPosConv;        // The SpinTAC Position Converter Object
ST_POSCONV_Handle stPosConvHandle;  // The SpinTAC Position Converter Handle
```

18.3.1.3 初始化配置变量

该步骤应在无限循环前在项目主函数中完成。 这样会将所有的默认值加载到 SpinTAC 位置转换中。 该步骤可通过运行已在 spintac_velocity.h 头文件中声明的函数 ST_init 和 ST_setupPosConv 来完成。 如果不希望使用这两个函数，可使用下方的代码示例配置 SpinTAC 位置转换。 SpinTAC 位置转换配置是典型配置的代表，适用于大部分电机。

```
// init the ST PosConv object
stPosConvHandle = STPOSCONV_init(&stPosConv, sizeof(stPosConv));

// Setup the SpinTAC Position Converter
// Sample time [s], (0, 1]
STPOSCONV_setSampleTime_sec(stPosConvHandle, _IQ(ST_SPEED_SAMPLE_TIME));
// The upper [0, 16] and lower [-16, 0] bounds of the input position signal [ERev]
STPOSCONV_setERevMaximums_erev(stPosConvHandle, _IQ(1.0), 0);
// Sets the unit conversions used in the SpinTAC Position Converter
STPOSCONV_setUnitConversion(stPosConvHandle, USER_IQ_FULL_SCALE_FREQ_Hz,
                                        ST_SAMPLE_TIME, USER_MOTOR_NUM_POLE_PAIRS);
// The Rollover bound of the output position signal [MRev]
STPOSCONV_setMRevMaximum_mrev(stPosConvHandle, _IQ(10.0));
// Low-pass time constant [tick]
STPOSCONV_setLowPassFilterTime_tick(stPosConvHandle, 3);
// ST_PosConv should start enabled
STPOSCONV_setEnable(stPosConvHandle, true);
// ST_PosConv should not be in reset
STPOSCONV_setReset(stPosConvHandle, false);
```

18.3.1.4 调用 SpinTAC 位置转换

该步骤应在主 ISR 中完成。 该组件需要在适当的抽取率下调用此功能。 抽取率由 spintac_velocity.h 头文件中声明的 ST_ISR_TICKS_PER_SPINTAC_TICK 确定；有关详细信息，请参见节 4.7.1.4。 在调用 SpinTAC 位置转换功能前，需要将 ENC 模块计算得出的电角传送到 SpinTAC 位置转换中。

```
// update the electrical angle
STPOSCONV_setElecAngle_erev(stPosConvHandle, ENC_getElecAngle(encHandle));
// run the SpinTAC Position Converter
STPOSCONV_run(stPosConvHandle);
```

18.3.2 SpinTAC 位置转换故障排除

18.3.2.1 ERR_ID

ERR_ID 可向用户提供错误代码。 表 18-2 中列出了 SpinTAC 位置转换中定义的错误及相应解决方法。

表 **18-2. SpinTAC 位置转换 ERR_ID 代码**

ERR_ID	问题	解决方法
1	采样时间值无效	将 cfg.T_sec 设置为 (0, 0.01] 范围内
13	位置翻转限定值无效	将 cfg.ROMax_mrev 设置为 [2, 100] 范围内

表 18-2. SpinTAC 位置转换 ERR_ID 代码 (continued)

ERR_ID	问题	解决方法
21	从 [MRev] 到 [ERev] 的换算系数为无效值	将 cfg.PolePairs 设置为 [1, 32] 范围内
25	从位置（单位 [MRev]）到速度（单位 [pu/s]）的换算系数为无效值	将 cfg.erev_TO_pu_ps 设置为正 IQ24 值
26	输入的锯齿位置上限值无效	将 cfg.ROMax_erev 设置为 [0, 16] 范围内
27	输入的锯齿位置下限值无效	将 cfg.ROMin_erev 设置为 [-16, 0] 范围内
37	输入的 cfg.OneOverFreqTimeConst 无效	将 cfg.OneOverFreqTimeConst 设置为正值
38	输入的 cfg.SampleTimeOverTimeConst 无效	将 cfg.SampleTimeOverTimeConst 设置为正值
1010	速度反馈低通滤波时间常量无效	将 cfg.LpfTime_tick 设置为 [1, 100] 范围内
4003	ROM 版本无效	使用 ROM 版本有效的芯片或使用与当前 ROM 版本兼容的 SpinTAC 库。

疑难电机

详细介绍关于电机识别和运行存在困难的电机的相关问题。 为这些特殊案例提供相关技术来调整 InstaSPIN-FOC 和 InstaSPIN-MOTION。

添加系统功能

有关如何将系统软件集成到 MotorWare 项目（如 InstaSPIN-FOC）的详细信息。

SPRUHJ1 — http://www-s.ti.com/sc/techlit/SPRUHJ1

构建 InstaSPIN-FOC 和 InstaSPIN-MOTION 电路板

详细介绍如何设计采用 InstaSPIN-FOC 或 InstaSPIN-MOTION 以满足应用要求的 PCB。

术语和缩略词定义

ACIM — 交流感应电机。

ADRC — 主动抗扰控制。 实时地估算和补偿系统干扰。

CCStudio — Code Composer Studio.

FAST — 统一观测器结构，此结构充分利用所有使用磁通量来进行能量转换的电机间的相似之处来自动识别所需的电机参数并提供以下电机反馈信号：

- 针对稳定磁通监视和场强减弱的高质量 磁通信号。
- 与独立于 ACIM 全部转子参数的传统观测器技术相比，在更宽的速度范围内具有出色的转子磁通角度估算精度。
- 实时低噪声电机转轴 速度信号。
- 针对负载监视和失衡检测的准确高带宽 转矩信号。

FOC — 磁场定向控制。

强制角 — 用于启动时的 100% 力矩，直到 FAST 转子磁通角跟踪器在第一个电周期内收敛。

InstaSPIN-FOC — 由选择器件（FAST 观测器，FOC，速度和电流环路）上的 TI 片上 ROM 提供完整的无传感器 FOC 解决方案，从而有效地控制您的电机，而又无需使用任何机械转子传感器。

InstaSPIN-MOTION — 一款综合电机、运动和速度控制软件解决方案，此解决方案以最高效率为运行在不同运动状态转换中的电机应用提供稳健耐用的系统性能。 InstaSPIN-MOTION 建立于 InstaSPIN-FOC 之上，并且包含 InstaSPIN-FOC，它与 LineStream Technologies 公司的 SpinTAC™ 运动控制套件组合在一起。

IPM — 内部永磁电机。

LineStream Technologies — 嵌入式控制软件领域的先驱。 LineStream 公司引以拥有来自六个不同国家的电机控制专家团队而感到骄傲，这些专家总共说十五种不同的语言，并且具有超过八十年的从业经验，LineStream 公司正在快速成为世界上嵌入式电机控制领域的杰出代表。

电机参数 ID 或电机识别 — 一个特性被添加到 InstaSPIN-FOC，从而为用户提供一个工具，这样，即使在电机参数未知时，电机也可以以最高性能运行。

PI — 比例积分调节器。

PMSM — 永磁同步电机。

PowerWarp™ — 交流感应电机 (ACIM) 的一种工作模式，可最大限度降低轻负载条件下的电机损耗。

Rs 离线重校准 — 当电机未运行时，InstaSPIN-FOC 特性被用来重新校准定子电阻，Rs。

Rs 在线重校准 — 当电机运行在闭环中时，InstaSPIN-FOC 特性被用来重新校准定子电阻，Rs。

SpinTAC™ 运动控制套件 — 包括一个高级速度控制器，一个运动引擎，和一个运动序列规划器。 SpinTAC 抗扰速度控制器主动估算且实时补偿系统干扰，从而提升总体产品性能。 SpinTAC 运动电机根据用户定义的参数来计算理想基准信号（具有前馈）。 SpinTAC 支持标准工业曲线，以及 LineStream 的专有“平滑轨迹”曲线。 SpinTAC 运动序列规划器运行用户定义的状态转换图，从而轻松设计复杂运动序列。

SVM — 空间矢量调制。

修订历史记录（E 到 F）

Changes from E Revision (June 2014) to F Revision **Page**

- Added 已添加4.8 节，在 user.h 中设置 ACIM 电机参数 217

修订历史记录（D 到 E）

Changes from D Revision (March 2014) to E Revision **Page**

- Changed 已更改Chapter 1“简介”中的第二段 20
- Changed 已更改Chapter 1“简介”中的段落 21
- Added 已将 TMS320F2805xF 和 TMS320F2805xM 器件信息添加到1.1 节“InstaSPIN-FOC 和 FAST 概述” 22
- Changed 已更改Chapter 2“快速入门套件 - TI 提供的软件和硬件”中的第一段 48
- Added 已在Chapter 2“快速入门套件 - TI 提供的软件和硬件”中添加低压/中等电流套件 48
- Changed 3.3.1 节 标题更改为控制器 API 函数 – ctrl.c、ctrl.h、ctrl_obj.h 68
- Added 已添加节 3.3.1.1，CTRL 枚举和结构 68
- Added 已添加节 3.3.1.2，CTRL 状态控制和错误处理 71
- Added 已添加节 3.3.1.3，CTRL Get 函数 72
- Added 已添加节 3.3.1.4，CTRL 计数器函数 84
- Added 已添加节 3.3.1.5，CTRL Set 函数 86
- Added 已添加节 3.3.1.6，CTRL Run 和 Compute 函数 96
- Changed 3.3.2 节 标题更改为估算器 API 函数 - FAST 库 - est.h、est_states.h 98
- Added 已添加节 3.3.2.1，EST 枚举和结构 98
- Added 已添加节 3.3.2.2，EST Set 函数 99
- Added 已添加节 3.3.2.3，EST Get 函数 109
- Added 已添加节 3.3.2.4，EST Run 和 Compute 函数 130
- Added 已添加节 3.3.2.5，EST 计数器函数 132
- Added 已添加3.3.3 节，硬件抽象层 (HAL) API 函数 - hal.c、hal.h、hal_obj.h 135
- Changed 3.3.4 节 标题更改为用户设置 – user.c、user.h、userParams.h 153
- Added 已添加节 3.3.4.1，USER 枚举和结构 153
- Added 已添加节 3.3.4.2，USER Set 和 Compute 函数 159
- Added 已添加节 3.3.4.3，USER 错误处理函数 161
- Changed 已更改3.3.5 节“其他函数”中的 softwareUpdate1p6 () 162
- Changed 表 3-2，SpinTAC 版本结构中的修订版本和日期 167
- Changed 已更改3.5.9 节“SpinTAC 函数”中的 void **STVELPLAN_addCfgAct** 函数 193
- Changed 已更改3.5.9 节“SpinTAC 函数”中的 void **STPOSPLAN_addCfgAct** 函数 197
- Changed 已更改4.1.4 节，USER_VOLTAGE_SF 204
- Changed 已更改4.1.7 节，USER_CURRENT_SF 205
- Deleted 已删除 USER_MAX_DUTY_CYCLE 部分 206
- Added 4.2.3 节，USER_MAX_VS_MAG_PU 206
- Changed 已更改4.2.5 节，USER_ISR_FREQ_Hz 206
- Changed 已更改4.2.6 节，USER_ISR_PERIOD_usec 207
- Changed 已更改4.3.11 节，USER_CTRL_PERIOD_sec 208
- Changed 已更改4.4.1 节，USER_MAX_NEGATIVE_ID_REF_CURRENT_A 208
- Changed 已更改4.4.2 节，USER_ZEROSPEEDLIMIT 209
- Changed 已更改4.4.3 节，USER_FORCE_ANGLE_FREQ_Hz 209
- Changed 已更改4.4.4 节，USER_MAX_CURRENT_SLOPE_POWERWARP 209
- Changed 已更改4.4.8 节，USER_IDRATED_FRACTION_FOR_RATED_FLUX 210
- Changed 已更改4.4.9 节，USER_IDRATED_FRACTION_FOR_L_IDENT 210
- Changed 已更改4.4.10 节，USER_IDRATED_DELTA 210
- Changed 已更改4.4.13 节，USER_POWERWARP_GAIN 210

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道1568 号，中建大厦32 楼邮政编码： 200122

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265