Application Note

*用于牵引逆变器的 **AM263x***

Feng Qi

摘要

牵引逆变器是电动汽车的核心。为了实现逆变器的实时控制功能，微控制器必须既能实现外设低延迟，又能确定处理时间。为了加速微控制器的市场推广，使用一个架构良好且能展示其潜力的逆变器软件框架至关重要。使用这样的框架也将帮助客户显著缩短软件开发时间，专注于终端设备的开发。本文的重点是向客户介绍牵引逆变器的框架，并向客户展示 AM263x 的实时控制模块。本文末尾为客户总结了从其他微控制器迁移至 AM263x 的指导原则。

内容

插图清单

表格清单

商标

1 引言

AM263x 是一个片上系统 (SoC)，具有 Arm® Cortex®-R5F 集群和专用于实时控制的加速器。集群可以配置为双核模式或锁步模式。加速器命名为控制子系统，包括 ADC、DAC 和 PWM 等接口模块。本文以内核 0-0 为例，演示了一个 R5F 内核的特性和一个牵引逆变器的一组接口模块。此部分简要介绍 R5F 集群和控制子系统。有关 SoC 的更多详情，可参见 *AM263x Sitara™ 微控制器数据表*和 *AM263x Sitara 处理器技术参考手册*。有关逆变器硬件的详情，可参见*经过 ASIL D 等级功能安全认证的高速牵引和双向直流/直流转换参考设计*。

Arm® Cortex®-R5F 集群包括两个 R5F 内核，附带 L1 缓存和紧耦合存储器 (TCM) 之类的存储器、标准 Arm CoreSight™ 调试和布线架构、集成式矢量中断管理器 (VIM)、ECC 聚合器以及支持协议转换和地址转换的各种其他模块，以便于集成到 SoC。更详细的方框图可参见 AM263x 技术参考手册。解决实时控制问题的关键在于了解缓存和 TCM 的影响。在编写程序时，可以通过链接命令文件将指令和数据分配给片上 RAM 或 TCM。在执行过程中，片上 RAM 中经常使用的指令和数据会自动进入缓存。因此，执行时间显著改善。但是，片上 RAM 中的数据直到从缓存写回后才更新。当数据在缓存中时，只能通过内核中运行的指令访问它，Code Composer Studio™ (CCS) 等集成开发环境 (IDE) 的存储器视图无法读取缓存。然而，通过在内核内部运行通用异步接收器/发送器 (UART) 的一段程序，也可以使用 CSS 读取缓存。UART 方法的详情将在下一部分中介绍。另一方面，分配给 TCM 的指令和数据保存在地址中，可随时提供给存储器视图。通常情况下，程序在缓存和 TCM 中的执行时间非常接近，而片上 RAM 中的程序执行则要慢得多。从片上 RAM 向缓存转移程序的操作需要一些时间，并导致不可预测的延迟。如果延迟对于应用要求很重要，强烈建议将应用程序存储在 TCM 中。有关 TCM 地址的详情可参见 AM263x 技术参考手册。本文中，场定向控制的中断程序和软件旋转变压器数字转换器位于 TCM 中。链接命令文件作为示例放在 CCS 项目文件夹中。

实时控制加速器继承了全球广泛使用的德州仪器 (TI) 经典 C2000 控制模块。它包括模数转换器 (ADC)、模拟比较器、缓冲数模转换器、增强型脉宽调制器 (EPWM)、增强型捕捉、增强型正交编码器脉冲、快速串行接口、Σ-Δ 滤波器模块和纵横制。有关模块的详情可参见 AM263x 技术参考手册。在直观的系统配置工具 SYSCONFIG 的帮助下，还可以通过在配置这些模块时减少对实施细节的暴露。有关 SYSCONFIG 的详情可参见 AM263x 软件开发套件 (SDK)。模块同步的关键是在 EPWM 时基部分中配置 PWM 同步输入/输出，在 EPWM 事件触发器部分中配置 ADC 转换开始 (SOC) 触发器。时基用于对齐多个 PWM 通道，而事件触发器用于同步 ADC、DMA 和中断等特性。牵引逆变器演示的 CCS 项目文件夹中有一个牵引逆变器示例。本例中设置了一个 PWM 通道，通过 DMA 和 DAC 以更高的频率触发旋转变压器激励信号更新，还使用三个 PWM 通道生成逆变器信号和 ADC SOC。这样，来自 DAC 的旋转变压器激励信号将对齐到所需的 ADC 采样相位。多个 ADC 器件可以共享同一个 SOC。也就是说，可以在多个 ADC 器件中同时采集多个样本。在一个 ADC 器件中，可以在 SOC 配置部分中配置采样序列。ADC 中断可在 INT 配置中设置。中断可以在一个 ADC 转换开始或结束时触发。有关 PWM 和 ADC 的部分简单示例可在 AM263x SDK 的“\examples\drivers\epwm”和“\examples\drivers\adc”下找到。有关 API 的更多详情，可在 AM263x SDK 的“\source\drivers”下找到。头文件的很多细节都在注释中。

前面提到，在开发过程中需要考虑一些主题，例如 SoC 架构和 SDK API。有关它们的更多详情，可参见 AM263x 技术参考手册和 SDK 软件包。在后面部分中，本文将重点介绍如何使用示例项目为框架，以加快开发牵引逆变器，以及如何将现有项目代码迁移到 AM263x。

框架包括图 1-1 中突出显示的资源，并内置到图 1-2 中的牵引系统。TIDM-02009 是牵引逆变器参考设计硬件。它包括电源模型、栅极驱动器、电流和电压采样、旋转变压器模拟前端以及一些连接接口。电机包括接受正弦波激励并发送调制反馈、以检测位置的旋转变压器。场定向控制使用集群 0 内核 0 实现。ADC INT1 的实时控制部分将分配给 TCM，以精准确定执行时间。

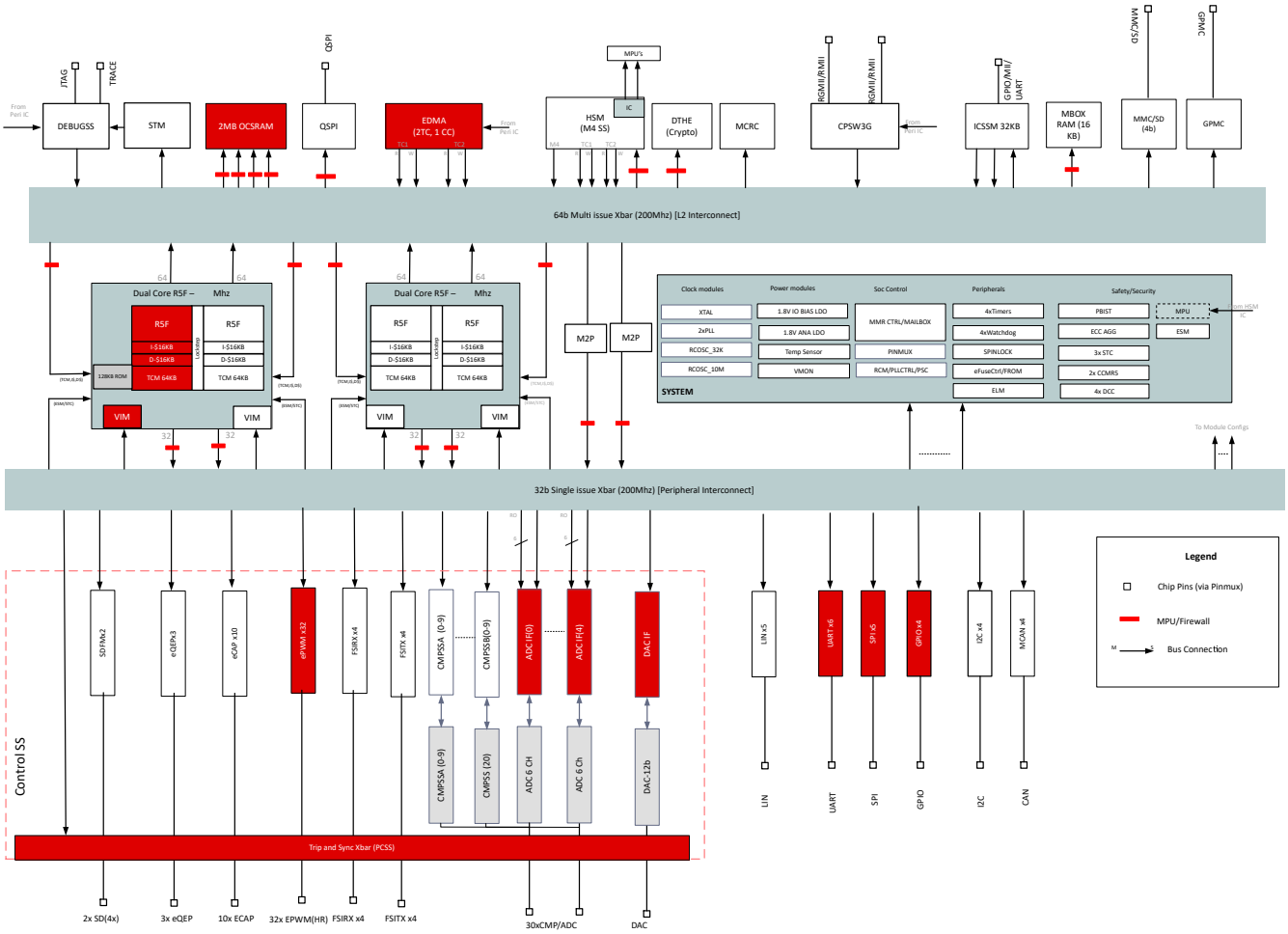

图 1-1. 牵引框架资源

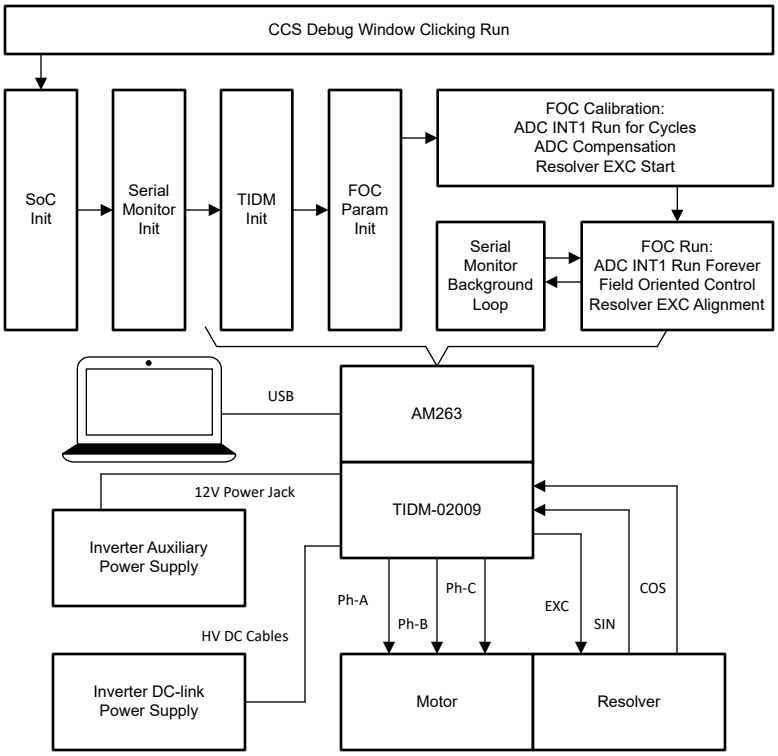

图 1-2. 牵引系统图

2 运行牵引逆变器的分步指南

2.1 创建实时调试接口

对于 AM263x，通过 CCS 与 AM263x 之间的 UART 连接启用实时调试。通过实时调试，全局变量可以添加到表达式窗口，并可在程序持续运行期间读取/写入。通过所列文件中的调试程序建立连接。

- Serial_Cmd_Monitor.c
- Serial_Cmd_Monitor.h
- Serial_Cmd_HAL.c
- Serial_Cmd_HAL.h

虽然这里列出了四个文件，但应用程序只需要两个函数。一个是初始化中调用的“SerialCmd_init()”，另一个是 BareMetal 后台循环或 RTOS 低优先级任务中调用的“SerialCmd_read()”。此部分重点介绍如何创建 UART 连接和如何在 CCS 中启动实时调试。

2.1.1 确认 CCS 特性

如果 CCS 版本早于 11.1，建议检查以下 CCS 驱动程序文件。Cortex_R5 的配置应类似于图 2-1。如果缺少任一行，必须如图 2-1 所示添加行。至于行的内容，需要在目标配置文件中更新 COM 端口和波特率，此文件包含在下一步中。

- ccs\ccs_base\common\targetdb\drivers\gti_uart_driver.xml

```
<isa Type="Cortex_R5" ProcID="0x75803400">
    <driver file="../../../DebugServer/drivers/XPCOMToGTIAdapter.dvr">
        <property Type="stringfield" Value="COM14" id="COM Port" />
        <property Type="stringfield" Value="9600" id="Baud Rate" />
        <property Type="hiddenfield" Value="Little Endian" id="Endianness" />
        <property Type="hiddenfield" Value="32" id="Word Size Page 0" />
        <property Type="hiddenfield" Value="8" id="Minimum Addressable Size Page 0" />
        <property Type="hiddenfield" Value="@ti.com/UARTMonitor;1" id="XPCOM Class ID" />
        <property Type="hiddenfield" Value="Flash DLL Delegate" id="TargetAccess" />
        <connectionType Type="UARTConnection"/>
    </driver>
</isa>
```

图 2-1. R5F 的 CCS GTI UART 驱动程序

2.1.2 创建目标配置文件

图 2-2 中的控制卡在一个 USB 端口中同时提供 JTAG 和 UART 端口。有关硬件连接的详情，可参见 *AM263x 控制卡用户指南*。必须为调试端口创建目标配置文件。图 2-3 至图 2-10 以屏幕截图形式提供分步指南。简而言之，创建一个目标配置文件，然后为其配置 JTAG 和 UART。图 2-10 中的 UART COM 端口应该与 JTAG 探头应用/用户 UART 的 PC 设备管理器 COM 端口匹配。图 2-10 中的波特率应该与下一步中配置的 SoC UART 波特率一致。

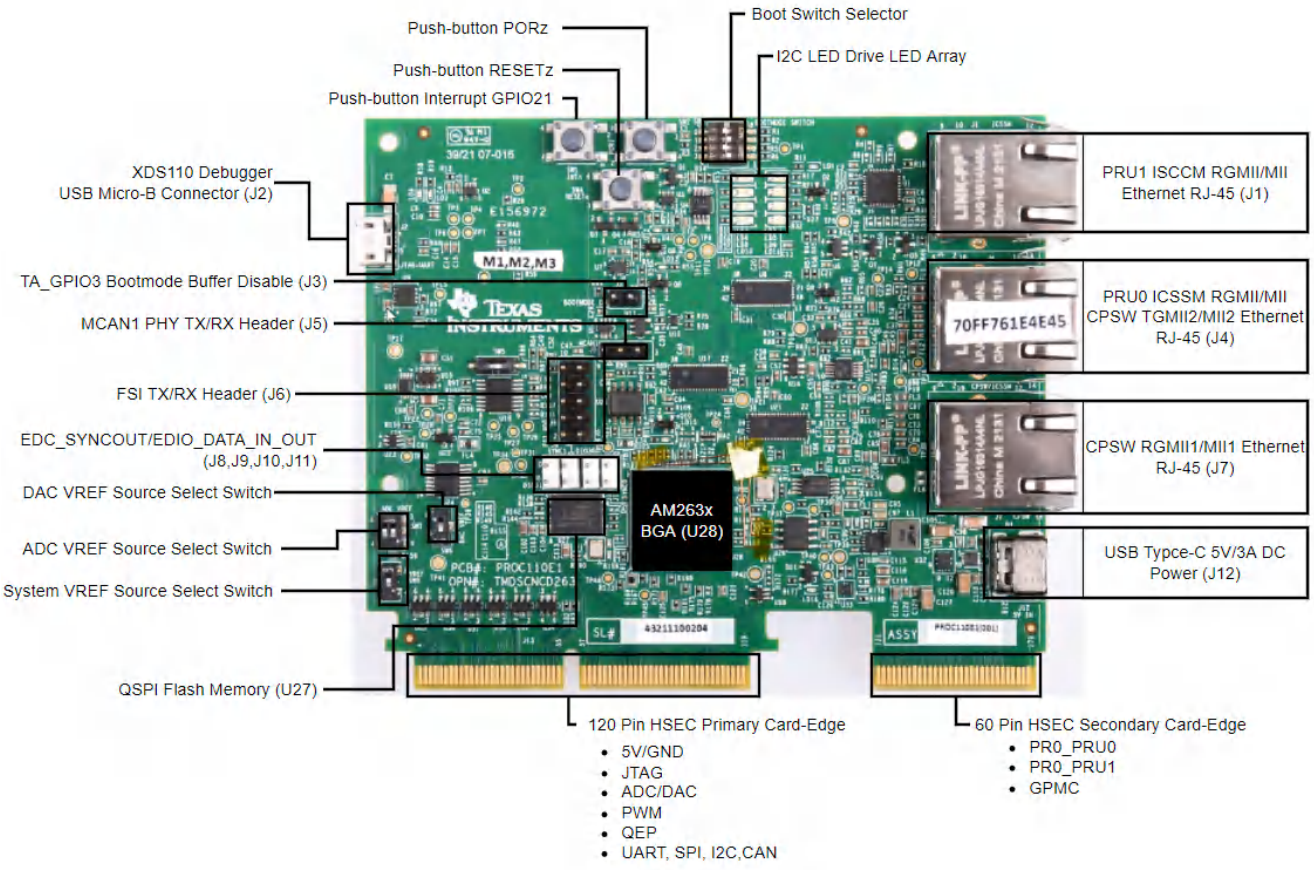

图 2-2. AM263x 控制卡

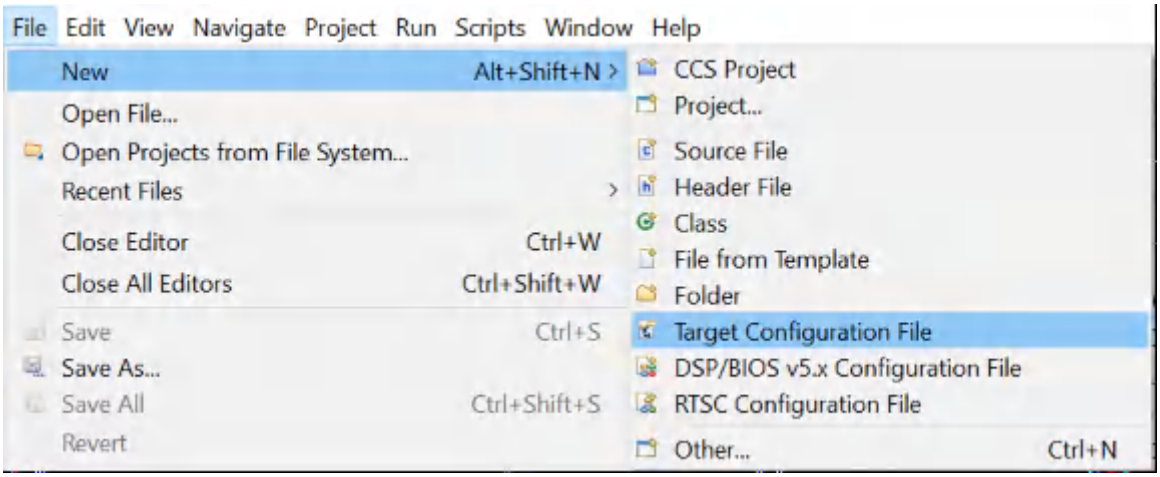

图 2-3. 创建新目标配置文件

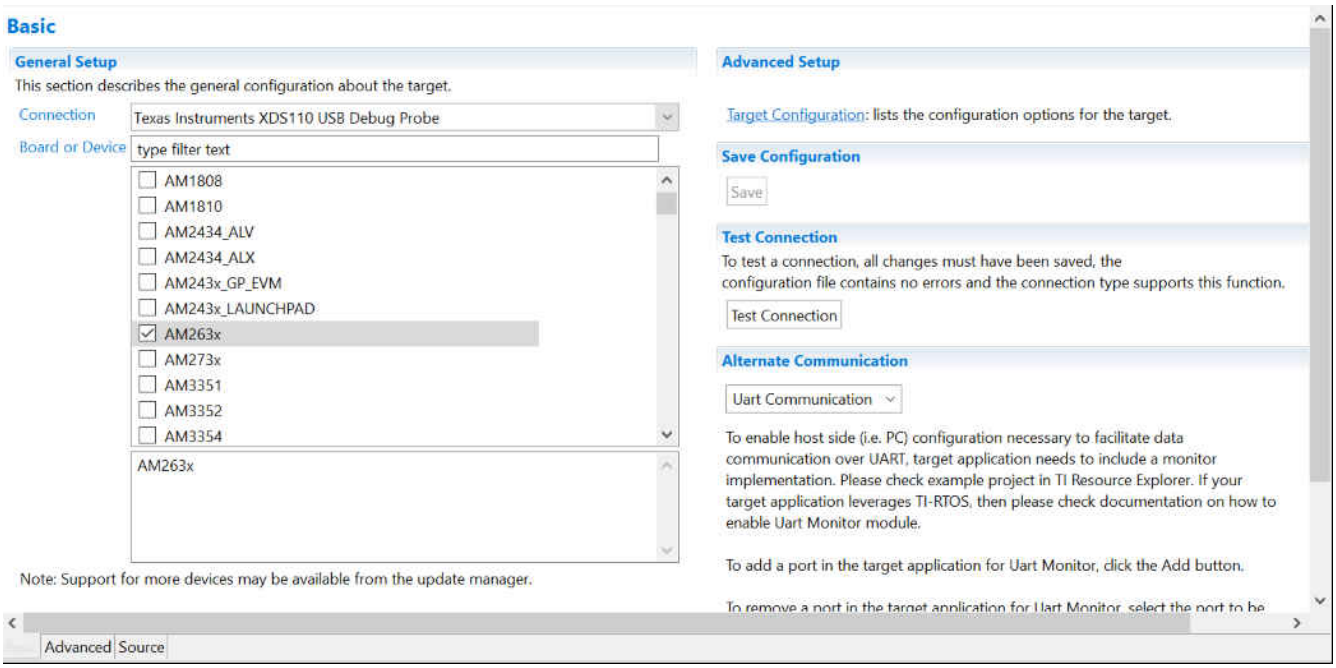

图 2-4. 选择 JTAG 连接和设备

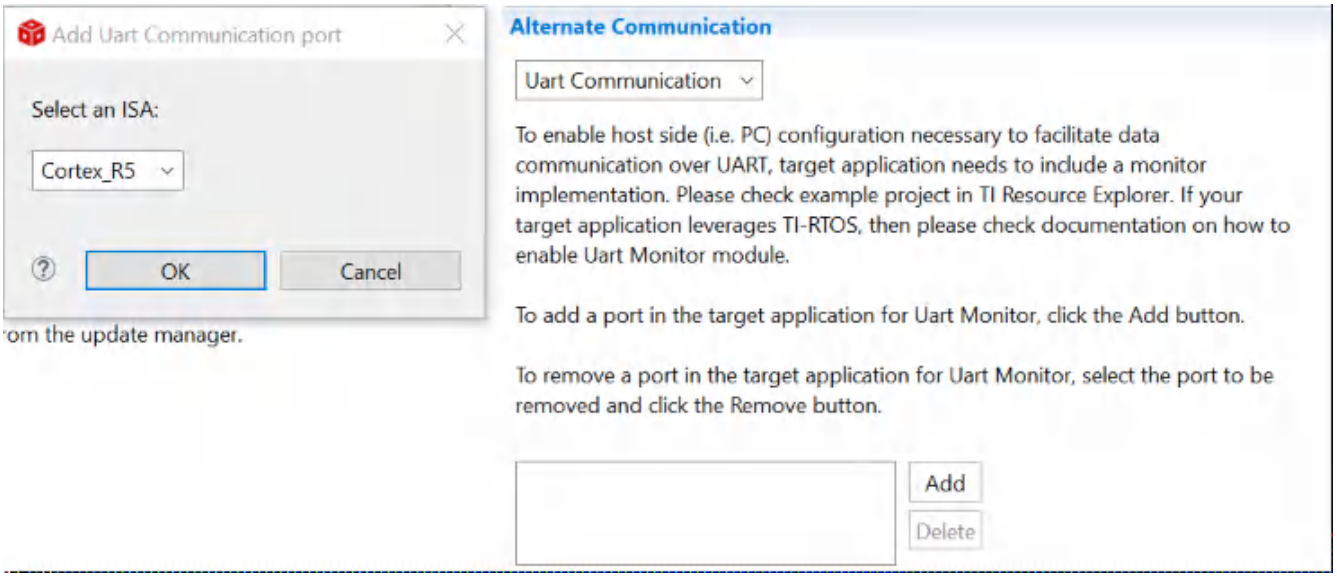

图 2-5. 添加 UART 通信端口

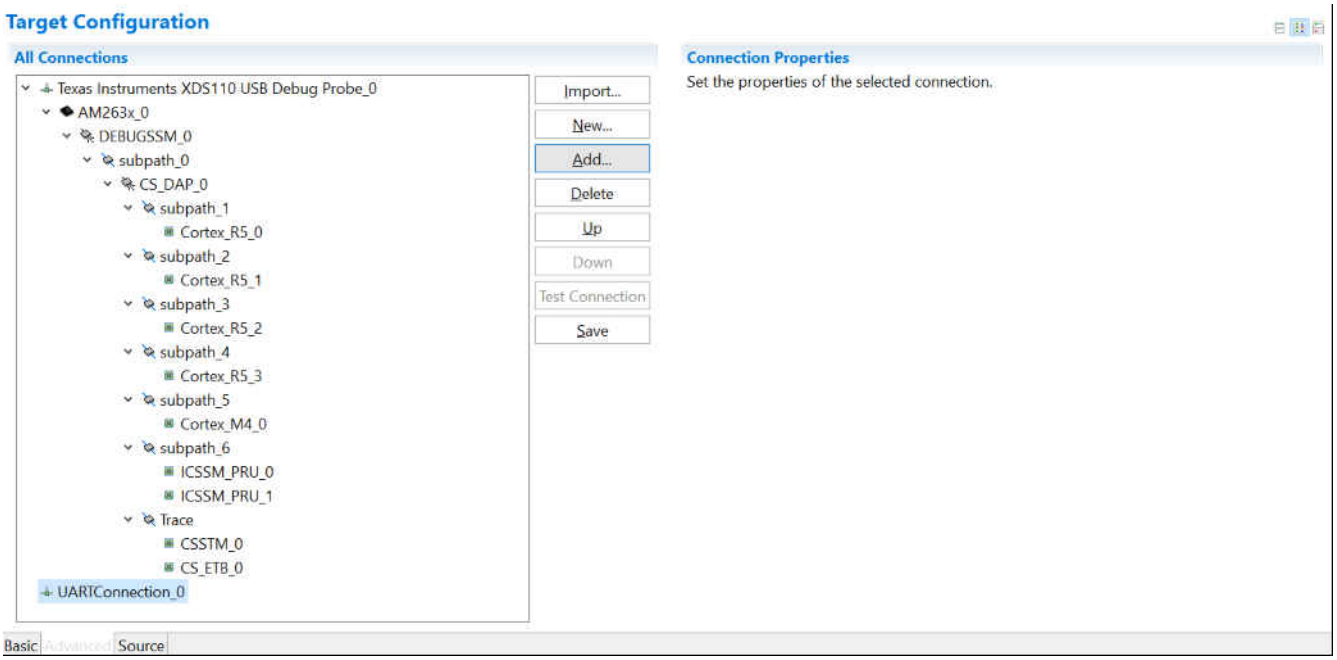

图 2-6. 打开高级目标配置

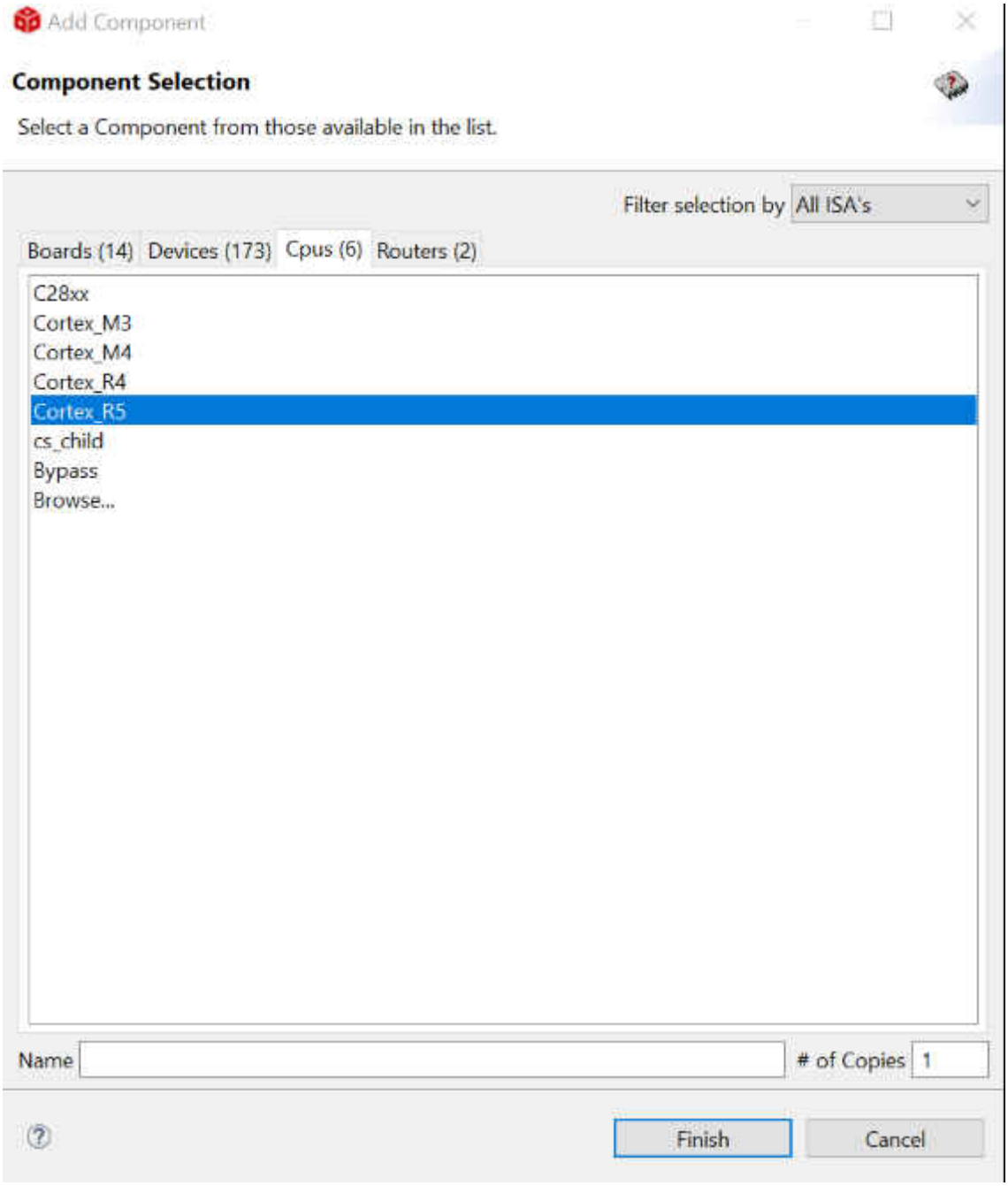

图 2-7. 添加元件

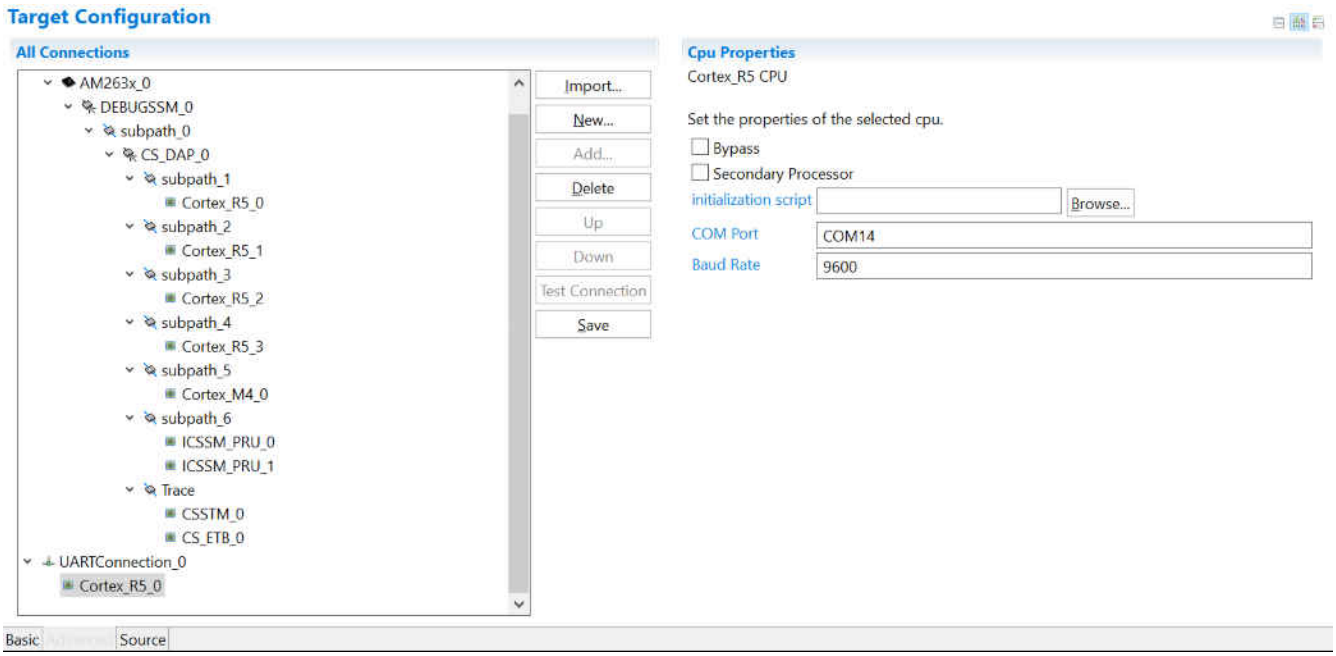

图 2-8. 选择 CPU 属性

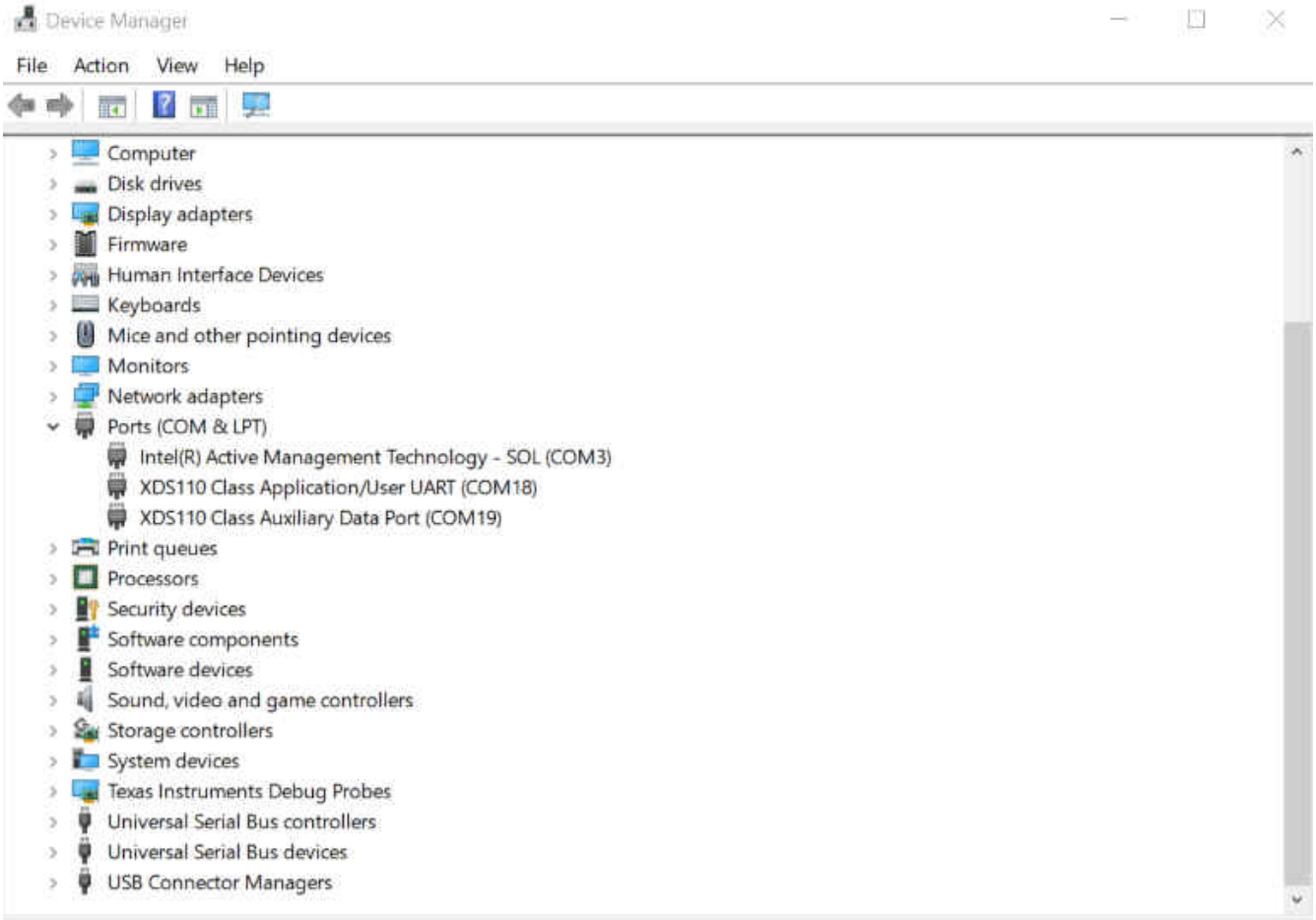

图 2-9. 查找 XDS110 UART COM 端口

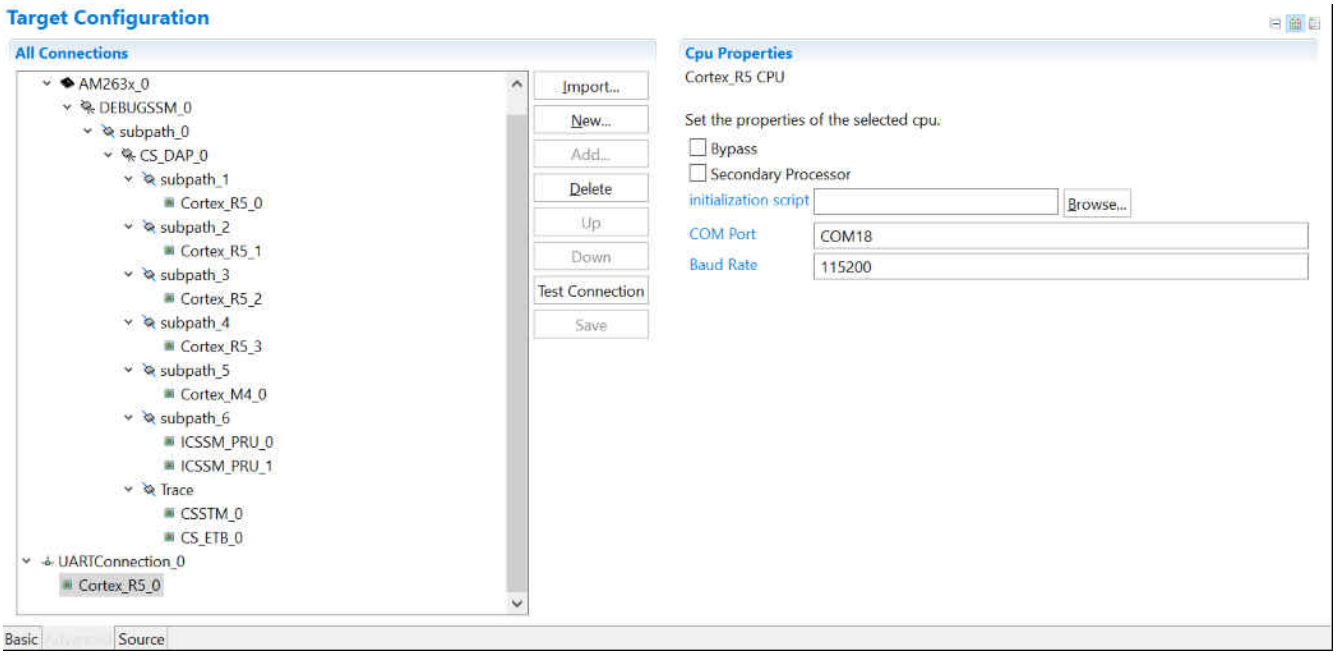

图 2-10. 在高级目标配置中更新 CPU 属性

2.1.3 添加串行命令监视器软件

使用 UART0 作为调试接口的方法有多种，分别是调试日志和串行命令监视器。调试日志是一个内置工具，位于 SDK 的驱动程序移植层。正如串行命令监视器，它的函数必须在中断回调之外。它是一个实现字符串输入和输出的方便工具。但是，输入和输出只通过 UART 控制台。CCS 中没有表达式窗口和图形之类的内置 GUI。建议在图 2-11 所示的“Debug Log”中禁用 UART0，并为图 2-12 所示的串行命令监视器配置 UART0 实例。Sysconfig 中的 UART 名称“CONFIG_UART_CONSOLE”与“Serial_Cmd_HAL.c”中的 handle 名称匹配，因此不必修改初始化和后台循环所需的两个函数。它们可以如图 2-13 所示直接插入。

图 2-11. 在调试日志中禁用 UART 日志

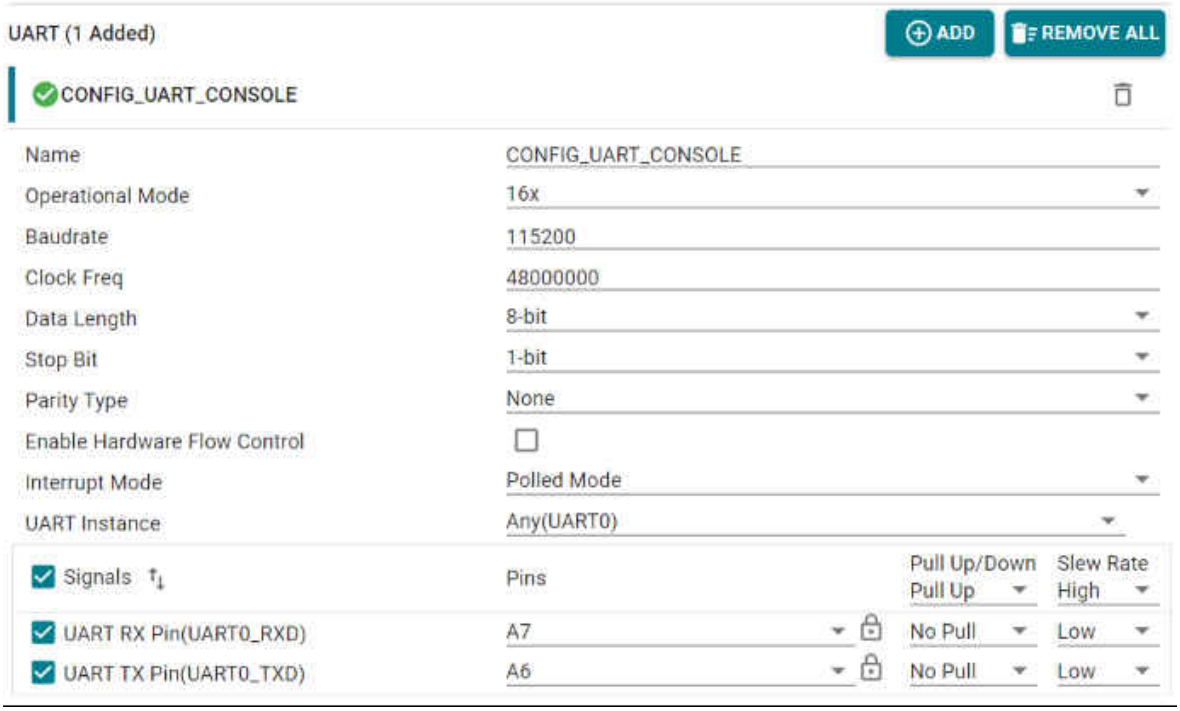

图 2-12. 配置 UART0 实例

```
void trinv_main(void *args)
{
    /* Open drivers to open the UART driver for console */
    Drivers_open();
    Board_driversOpen();

    SerialCmd_init();

    trinv_init();
    DebugP_log("trinv init done \r\n");
    FOC_init();
    DebugP_log("foc init done \r\n");
    FOC_cal();
    DebugP_log("foc cal done \r\n");
    FOC_run();
    DebugP_log("foc run done \r\n");

    while (gFlag){

        SerialCmd_read();

        gLoopTicker+=1;
        if (gLoopTicker>=100)
        {
            gLoopTicker=0;
        }
    }
    Board_driversClose();
    Drivers_close();
}
```

图 2-13. 添加串行监视器函数调用

2.1.4 启动实时调试

在构建程序之后，调试窗口应该会打开，其中包含在节 2.1.2 中创建的目标配置文件。如果创建的目标配置文件尚未打开，可以按照图 2-14 并查看“Target Configuration”窗口的“User Defined”文件夹找到它。创建的目标配置文件应该在“User Defined”文件夹下。右键点击该文件后，显示一个菜单，里面有一个“Launch Selected Configuration”选项，如图 2-15 所示。然后，调试窗口将显示。在许多 CCS 教程中可以找到连接目标、加载映像和通过 JTAG 运行的步骤。在连接到 UART 之前，处理器必须连续运行。由于 UART 连接以程序的连续运行为基础，停止串行命令监视器程序运行的断点、暂停、终止或任何其他事件将打断 UART 连接并冻结 CCS。有时，习惯在这些特性可用时使用它们。建立在使用 UART 连接时断开与目标的 JTAG 连接，如图 2-16 所示。当处理器运行时，可直接选择 UART 连接 → Run → Load → Load Symbols，建立 UART 连接，如图 2-17 所示。

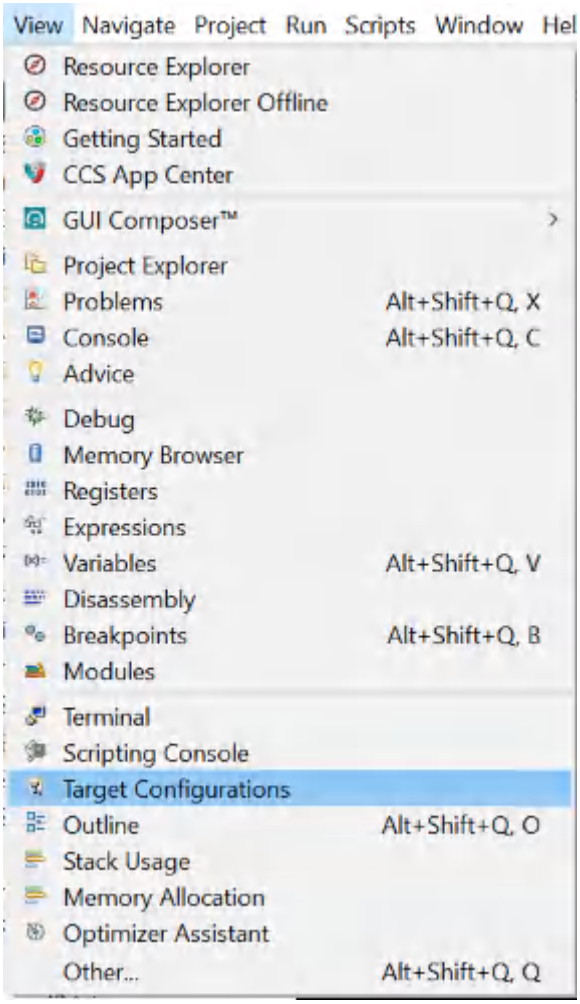

图 2-14. 找到目标配置文件

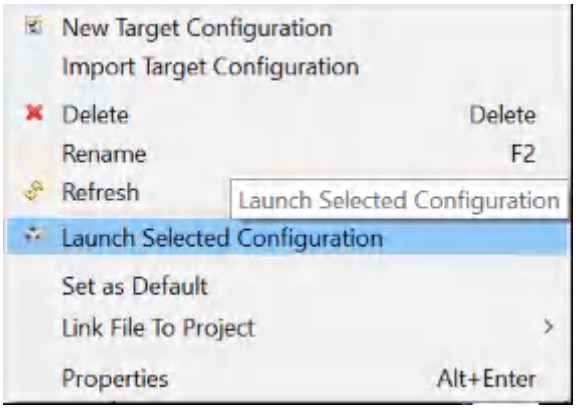

图 2-15. 启动所选配置

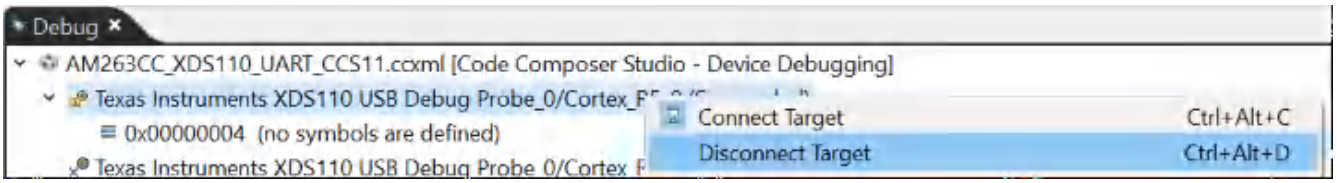

图 2-16. 断开 JTAG 连接

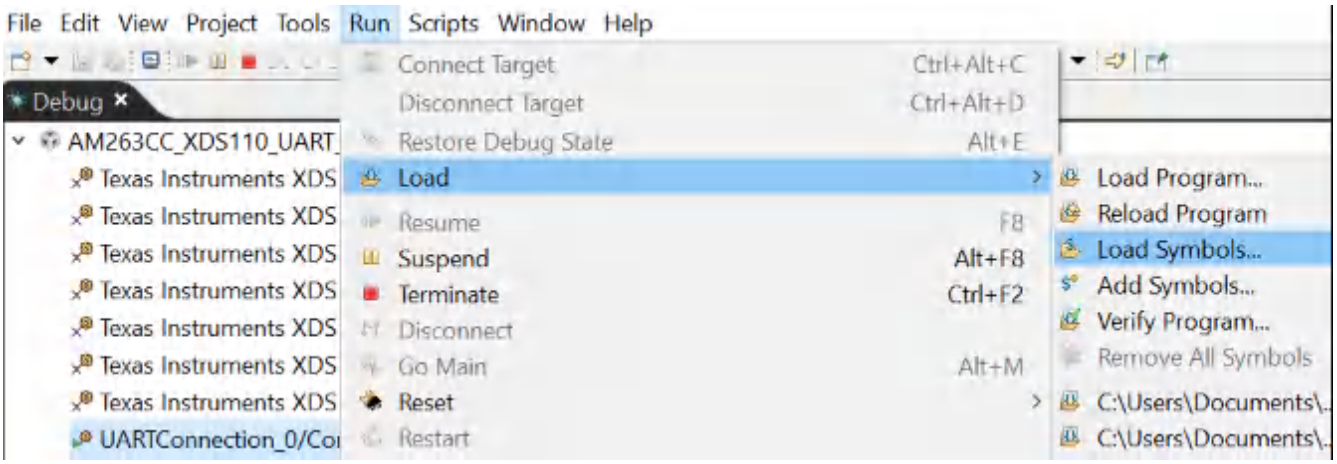

图 2-17. 建立 UART 连接

2.2 使用 Sysconfig 配置控制外设和 ADC 中断

Sysconfig 是一个基于 GUI 的配置工具。它显著简化控制外设和中断的配置。Sysconfig 内的各项描述得非常直观。在 Sysconfig 中完成配置后，配置程序自动生成，并在构建过程中连接到主函数。它们存储在名为“Generated Source”的文件夹下的以下文件下。值得注意的是，在使用 Sysconfig 构建之后，这些文件的内容将刷新。可以从这些文件复制程序，并在应用程序的其他部分中重复使用。有关 Sysconfig 的更多详情，可参见 SDK 文档。在本节的后面部分中，重点介绍如何设置牵引逆变器的控制外设和中断。

- ti_board_config.c
- ti_board_config.h
- ti_board_open_close.c
- ti_board_open_close.h
- ti_dpl_config.c
- ti_dpl_config.h
- ti_drivers_config.c
- ti_drivers_config.h

- ti_drivers_open_close.c
- ti_drivers_open_close.h
- ti_pinmux_config.c
- ti_power_clock_config.c

2.2.1 为时间基准生成 PWM

AM263x 的 PWM 模块继承了 TI 经典 C28 控制器的特性。AM263x 拥有独特的 XBAR 架构，它的 ADC SOC 触发器能够触发 ADC 事件以及 DMA 之类的各种其他事件。此部分将详细介绍如何同步 PWM 模块以及如何配置 PWM 模块，以触发 ADC 和 DMA。有关详细信息，请参见技术参考手册。

图 2-18 呈现了 ePWM7 总结信息，它用于触发通过 DAC 以 200kHz 更新的旋转变压器激励信号的 DMA。这里有几个选项卡可用于配置。但出于 DMA 触发目的，只会更新三个选项卡。

- EPWM 时基
- EPWM 动作限定器
- EPWM 事件触发器

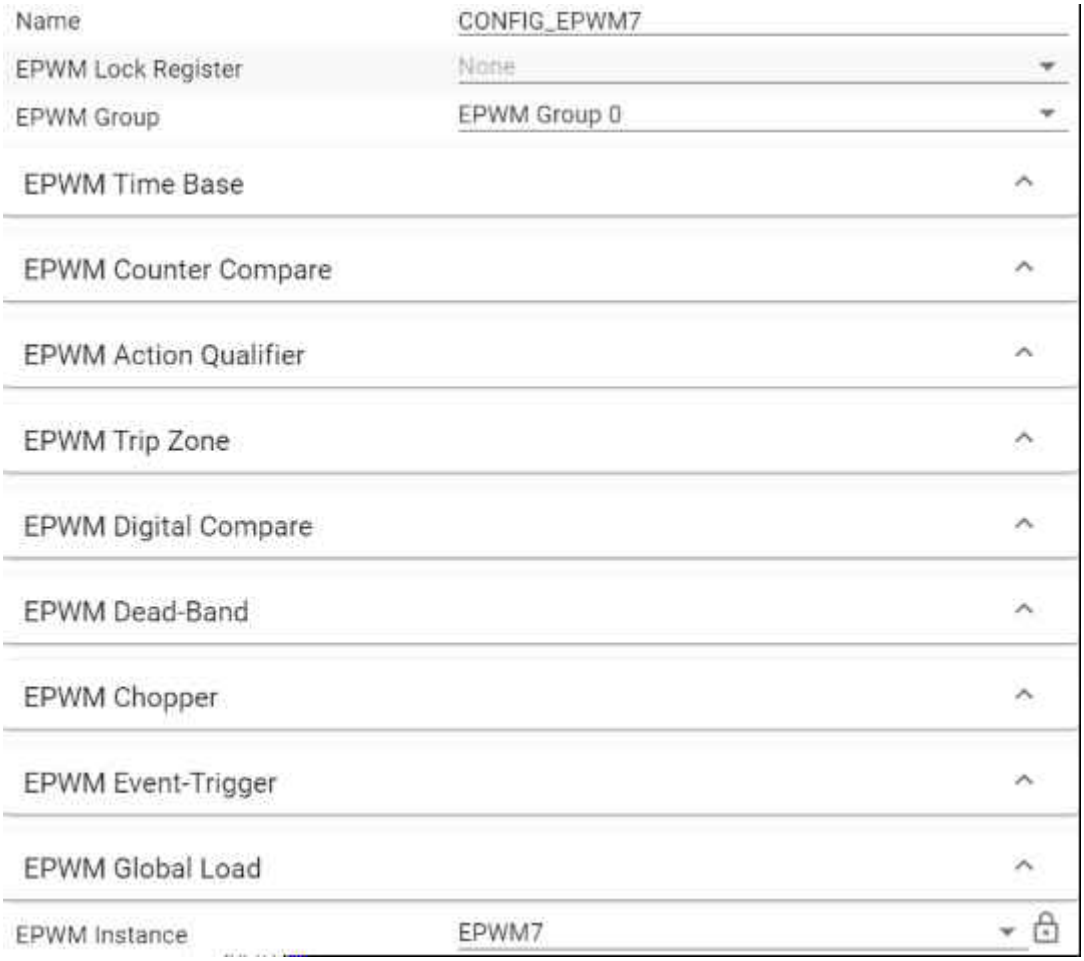

图 2-18. EPWM7 配置总结

图 2-19 显示有关时基的详细信息。EPWM7 配置为 200kHz、向上计数并遵循 ePWM0 的同步输出脉冲。值得注意的是，外设的工作频率是 200MHz，而 R5F 内核的运行频率是 400MHz。

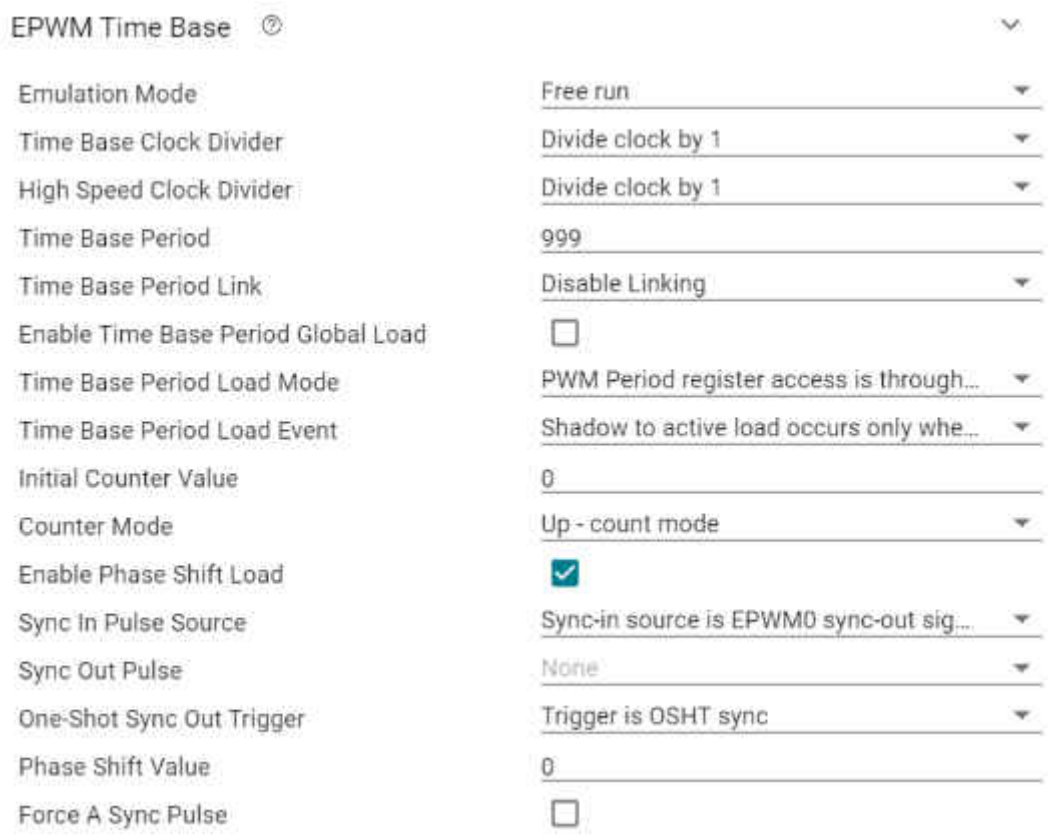

图 2-19. EPWM7 时基配置

图 2-20 显示有关事件触发器的详细信息。下面列出了主要内容。ePWM7 用于生成两个触发器，ADC SOCA 和 SOCB。两个触发器来自不同的源事件。与 CMPA 相关的触发器在“Counter Compare”选项卡中配置，如图 2-21 所示。

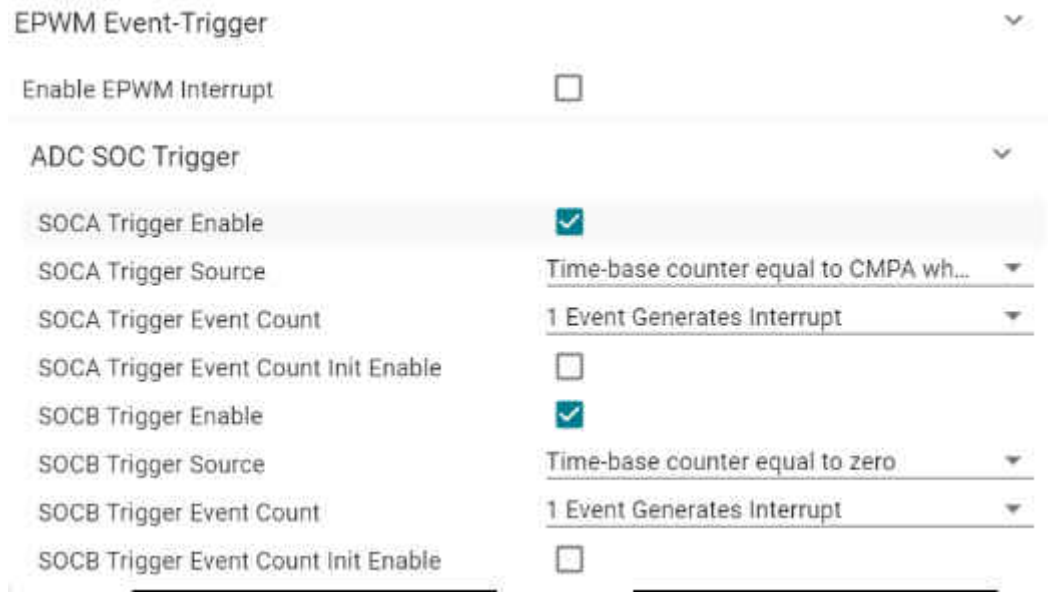

图 2-20. EPWM7 事件触发器配置

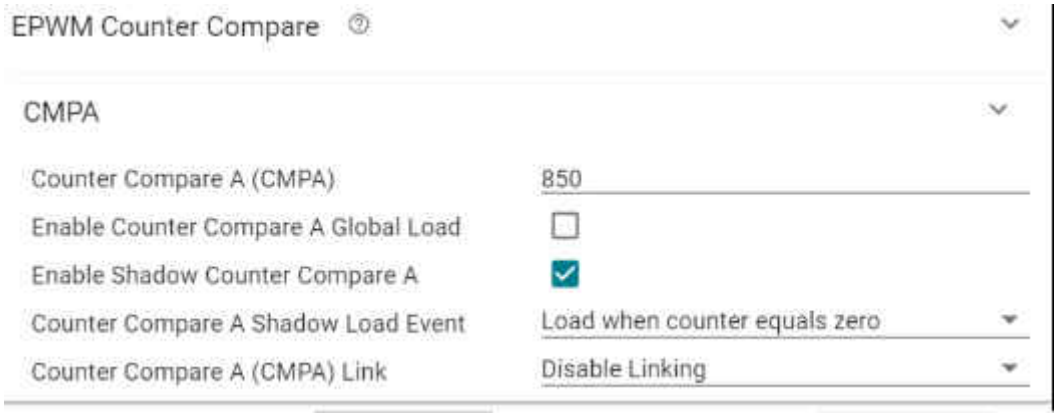

图 2-21. EPWM7 计数器比较配置

EPWM0 用作牵引逆变器的相位 A。所有 EPWM 通道都同步到 EPWM0。它配置为 10kHz、向上计数/向下计数、无同步输入、同步输出和计数器为零时 ADC SOC 触发器，如图 2-22 和图 2-23 所示。

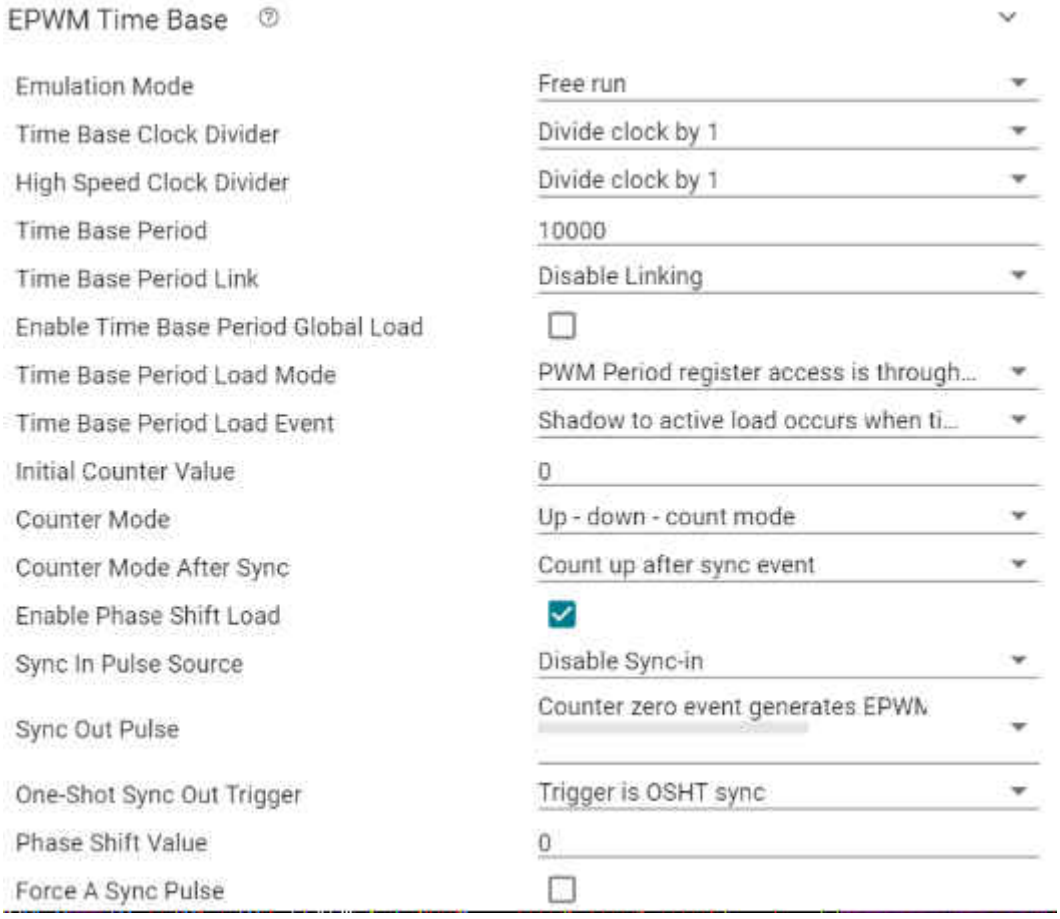

图 2-22. EPWM0 时基配置

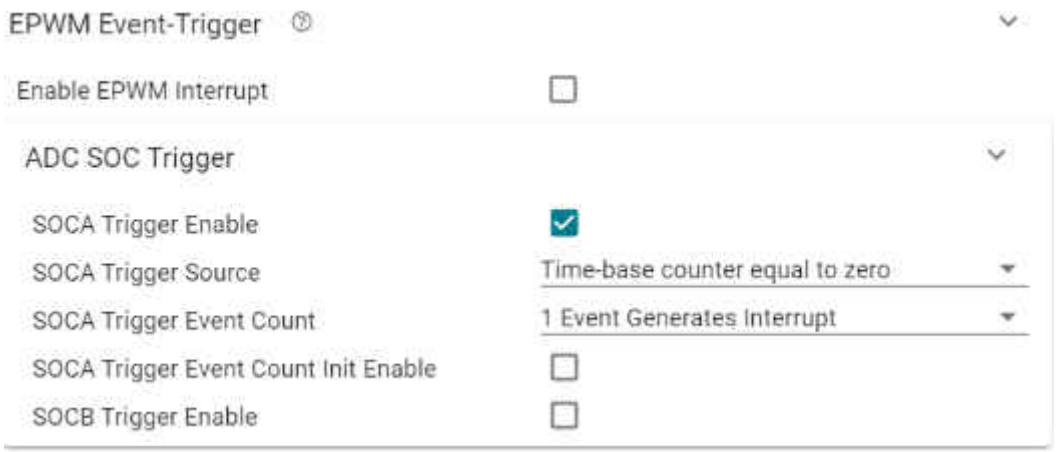

图 2-23. EPWM0 事件触发器配置

如图 2-24 所示配置 EPWM0 的死区。延迟值为 200 表示 1000ns，因为外设时钟频率为 200MHz。

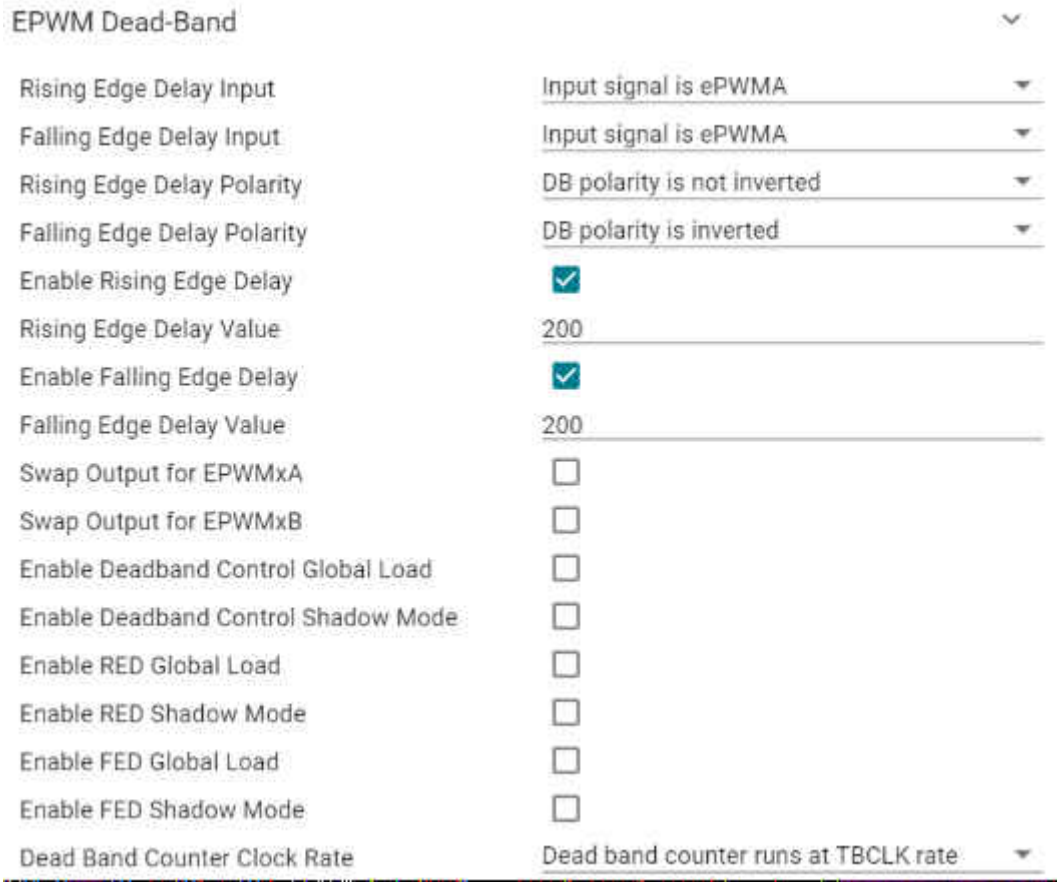

图 2-24. EPWM0 死区配置

2.2.2 同步 ADC 采样和中断服务例程

此部分以 ADC4 为例。图 2-25 提供了 ADC4 总结信息。它与 TI 经典的 C28 控制器相似。

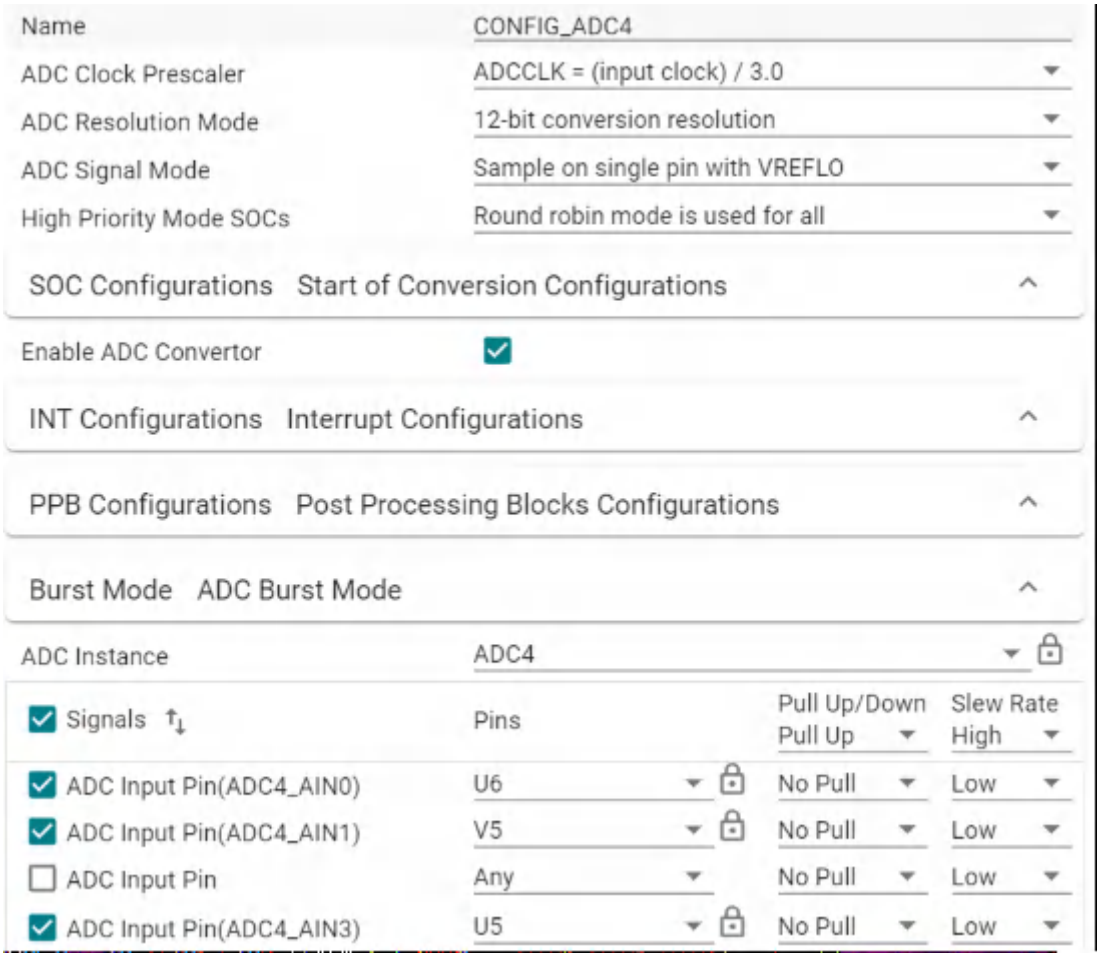

图 2-25. ADC4 配置总结

图 2-26 提供了一个 SOC0 示例。每个 ADC 有 16 个 SOC。ADC 通道可配置为单端或差分通道。此处，它由 PWM SOC 事件触发。

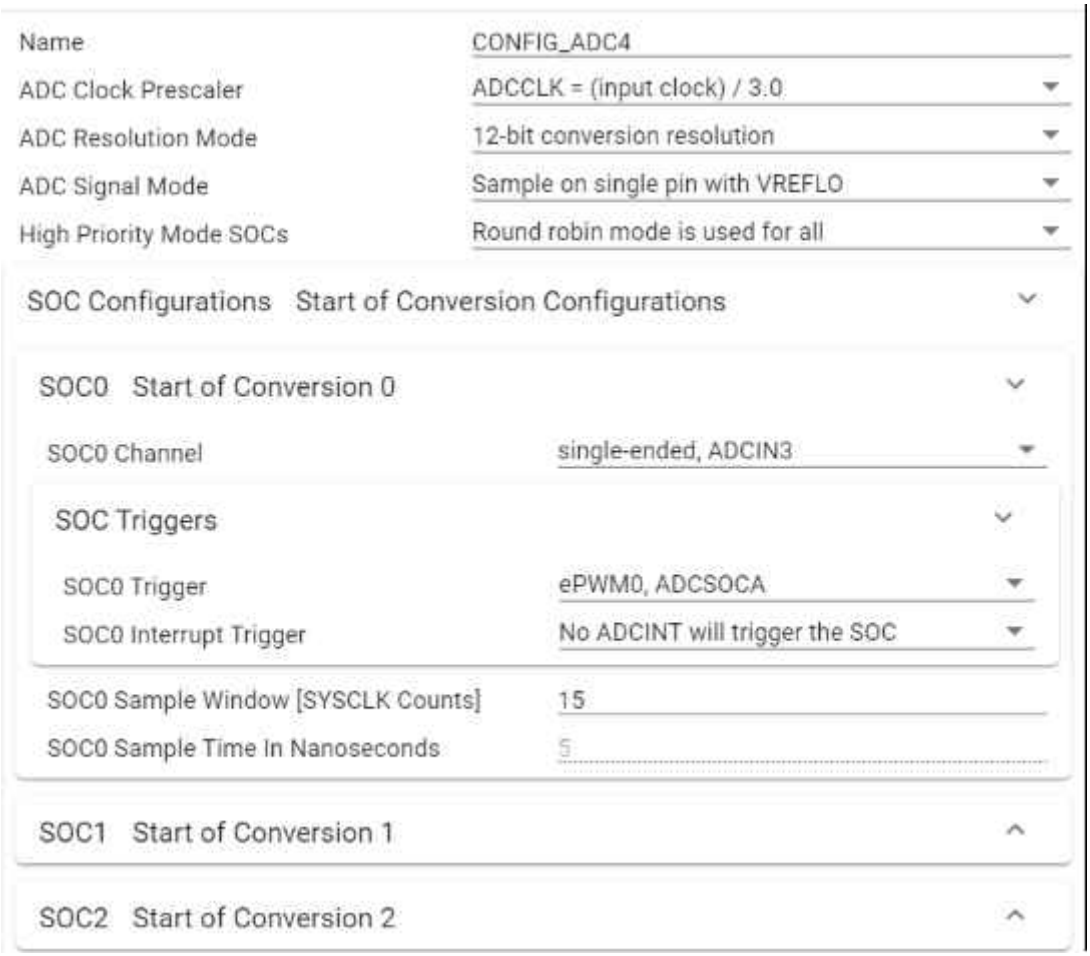

图 2-26. ADC4 SOC 配置

配置转换开始后，可如图 2-27 所示更新 INT。INT 将在 SOC5 转换结束时创建。

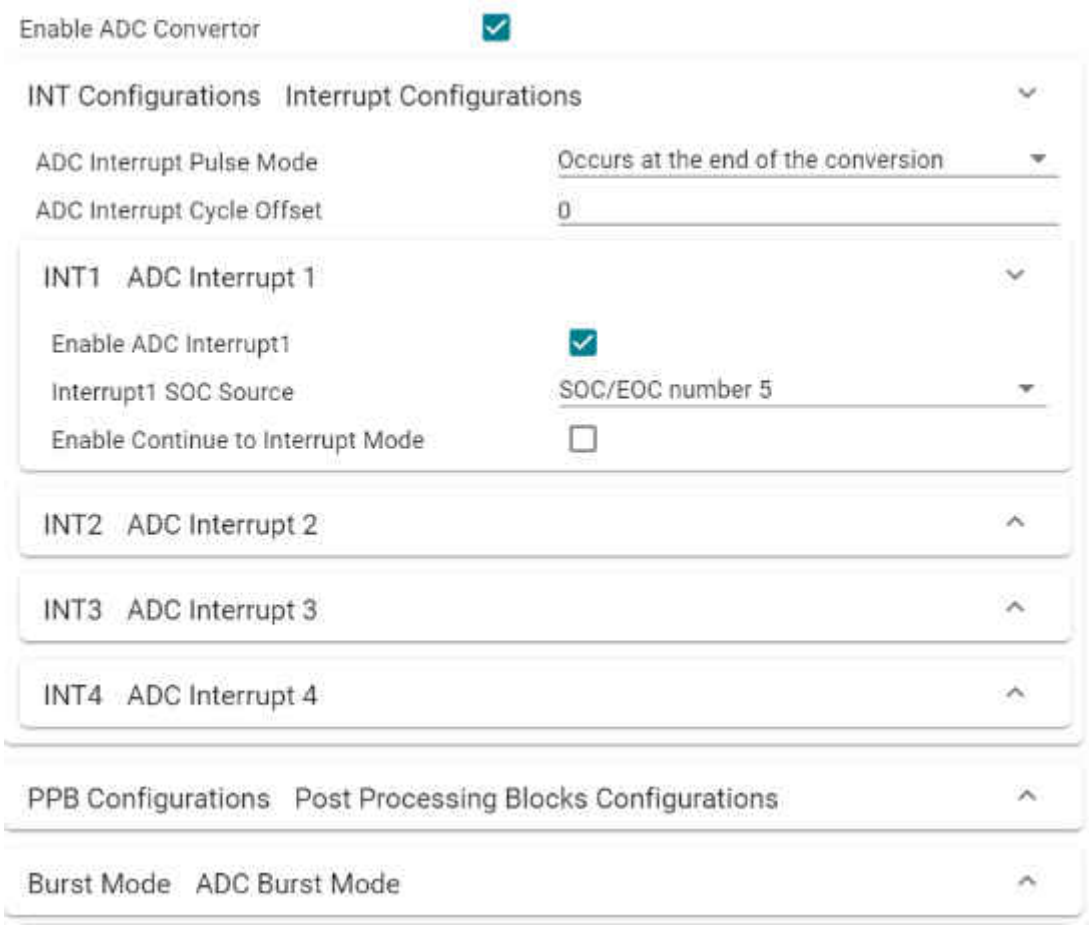

图 2-27. ADC4 INT 配置

在图 2-28 中，ADC INT 通过 XBAR 系统连接到 INT_XBAR_0。后面部分的中断寄存中使用该 XBAR 名称。

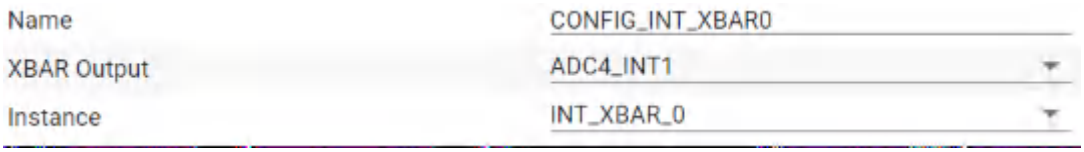

图 2-28. INT XBAR 配置

2.2.3 通过 DAC 为旋转变压器激励配置 DMA

在配置中，用户定义的函数在应用初始化期间更新图 2-29 所示的大部分 EDMA 模块。所以，大多数内容可保持默认状态。

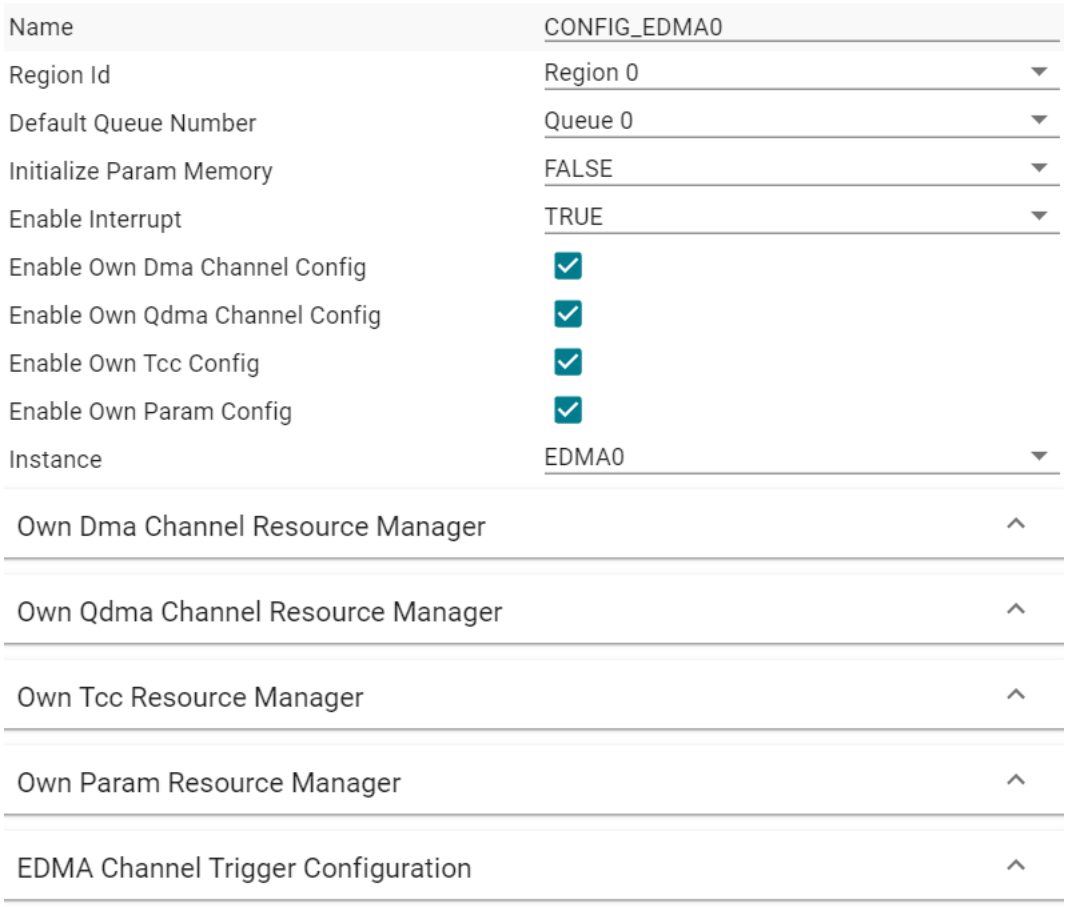

图 2-29. EDMA 配置总结

关键是图 2-30 中的 EDMA 通道触发器配置。所有触发源都在“Channel Trigger”菜单中列出。如前面所示，EPWM7 配置为 EDMA 触发器，以更新生成旋转变压器激励正弦波的 DAC。

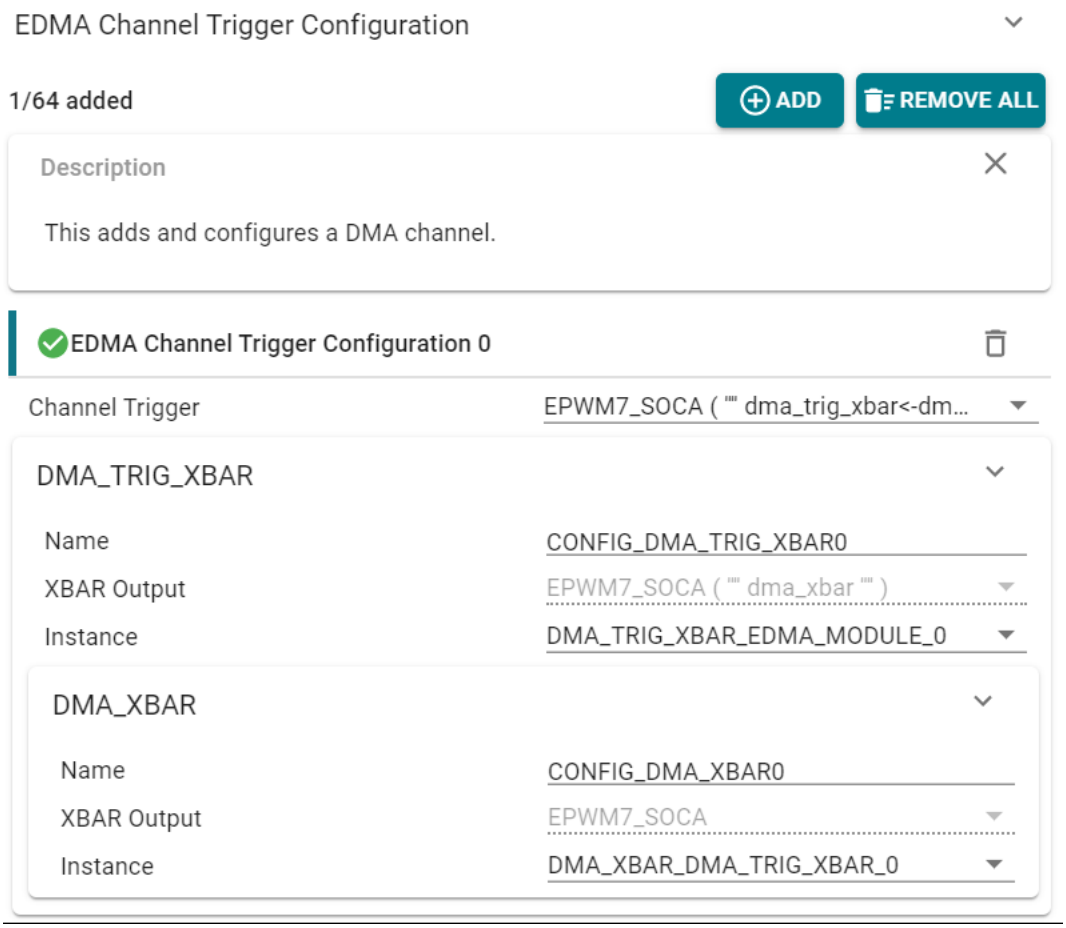

图 2-30. EDMA 通道触发器配置

2.3 使用 MSPI 配置栅极驱动器接口

在 TIDM-02009 中，UC5870-Q1 旨在驱动 SiC MOSFET。除了 PWM 门控信号外，它还需要一个 SPI 接口和几个 GPIO 进行通信和配置。在此部分中，需要检查并在必要时修改控制卡上的 pinmux。然后，配置 SPI 并初始化栅极驱动器。

2.3.1 确认栅极驱动器的控制卡硬件配置

如图 2-31 所示，4 电阻阵列封装和 2 电阻阵列用作电路板边缘接口 HSEC 与板载 SD 卡片夹 MMC0 之间的 pinmux。从 TIDM-02009 及其设计文件中可看到，需要电路板边缘引脚来连接某些栅极驱动器引脚。继续前，应确认 pinux 配置。

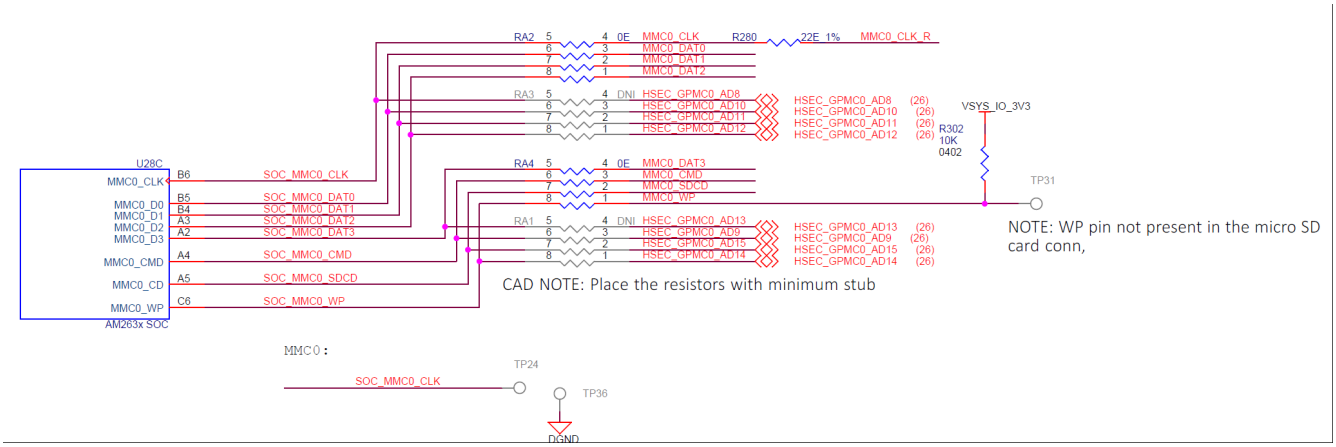

图 2-31. 控制卡 Pinmux

2.3.2 为 UCC5870 栅极驱动器配置 MCSPI

在 Sysconfig 和 UCC5870 的初始化程序中配置 MCSPI。图 2-32 中总结了 Sysconfig 项目。其中涵盖除数据帧大小之外的大多数特性。在此处列出的初始化程序中配置它。对于 UC5870-Q1，根据数据表使用 16 位数据帧。

- Init_UCC5870();
  - MCSPI_setDataWidth(spibaseAddr, spichNum, 16);

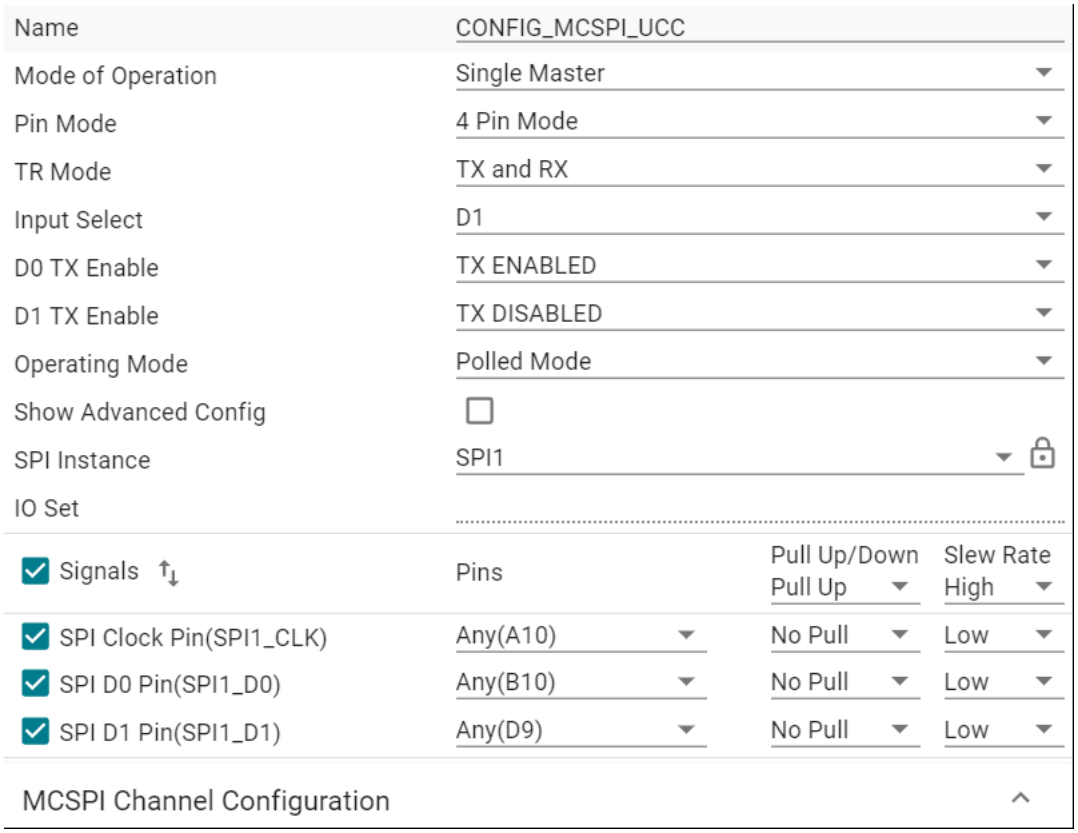

图 2-32. MCSPI 配置总结

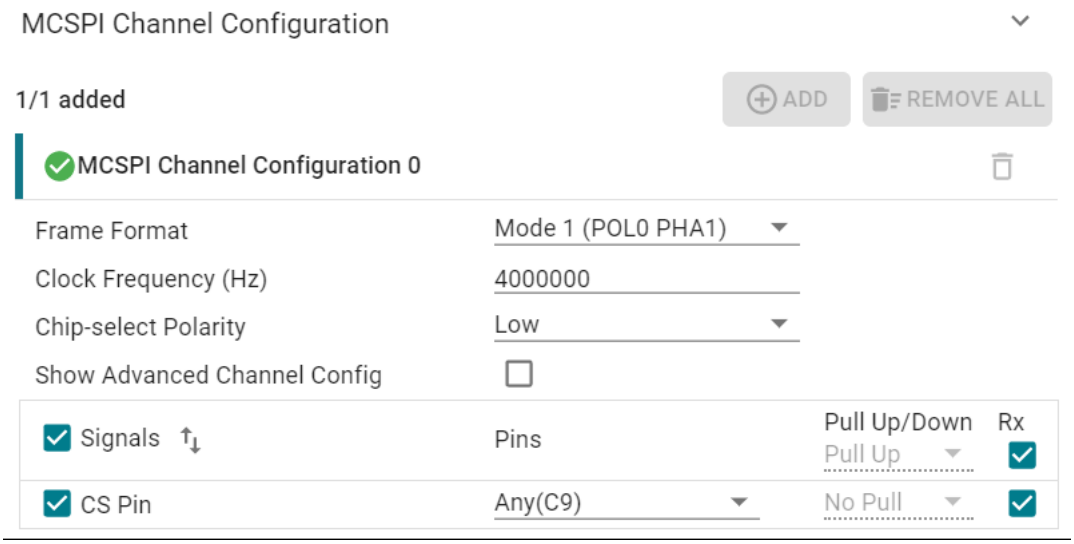

图 2-33. MCSPI 通道配置

2.3.3 初始化 UCC5870 栅极驱动器

UC5870-Q1 数据表是学习器件编程的最佳资源。TIDM-02009 采用基于地址的配置。SPI 通信的实现遵循 UC5870-Q1 中的 SPI 消息命令表。初始化过程遵循 UC5870-Q1 数据表中的“器件功能模式”部分。在系统级别，牵引逆变器初始化调用两个函数，在写入寄存器之前初始化寄存器值，并用这些值初始化栅极驱动器。两个函数如下：

- Init_UCC5870_Regs();
- Init_UCC5870();

2.4 从 ADC 采样并通过 CCS 读取样本

按照前面部分的配置，ADC 在 PWM0 零计数点对信号采样，并在 PWM0 每个周期的选定转换结束点创建 ADC INT。此部分介绍如何寄存 INT、读取样本并在图形中绘制它们。

2.4.1 寄存和启用中断

INT 配置后，必须用以下行寄存和启用中断。行 1 使用默认值初始化硬件中断参数。行 2 用上一部分中配置的数字更新中断数。在“cslr_intr_r5fss0_core0.h”中可找到有关宏定义的更多详情。如果使用其他集群或内核，在 SDK 中可找到相似的文件。行 3 将回调函数分配给硬件中断。它只接收函数地址。行 4 使用更新的硬件中断参数构造硬件中断。如果有错误，行 5 会通过 JTAG 将故障消息发送给 CCS 控制台。如果中断状态不清除，它会在行 6 清除该状态之后开始运行。为了保持中断持续运行，必须在每次执行回调函数时调用行 6。

1. HwiP_Params_init(&hwiPrms);
2. hwiPrms.intNum = CSLR_R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_0;
3. hwiPrms.callback = &FOCrun_ISR;
4. status = HwiP_construct(&gAdcHwiObject, &hwiPrms);
5. DebugP_assert(SystemP_SUCCESS == status);
6. ADC_clearInterruptStatus(CONFIG_ADC4_BASE_ADDR,ADC_INT_NUMBER1);

2.4.2 添加日志代码，以固定速率读取图中样本

由于 SoC 和 CCS 之间的通信用时是不确定的，所以有必要以给定的采样率从中断记录值。这对于通过图形窗口观察数据很关键。一个简单的日志已经实现。16 个指针可用于 16 个全局变量。硬件中断开始之前，指针需要分配给全局变量或 NULL。需要调用以下命令行。在指针分配后调用行 1。每次中断完成所有计算后调用行 2。行 2 函数中实现了一个名为 gLogScaler 的量程。它表示两个日志记录点之间跳过的中断数。通过设置全局变量，记录数据的频率可以低于或等于中断频率。

1. LoopLog_init();
2. LoopLog_run();

2.4.3 在表达式和图形窗口中读取 ADC 样本

以下行 1 至行 8 读取三相电流、旋转变压器 sin/cos 和直流总线电压的 ADC 样本。SDK API 包装到一个文件中的 Macros。按住 Ctrl 键简单地左键点击几次变量名称，有助于找到定义它的位置。行 9 和 10 提供了示例。ADC_readResult 要读取 ADC 结果，ADC_readPPBResult 要在后处理块之后读取 ADC 结果。有关后处理块的详细信息，可参见技术参考手册。

1. motor1.I_abc_A[0] = (float32_t)IFBU_PPB;
2. motor1.I_abc_A[1] = (float32_t)IFBV_PPB;
3. motor1.I_abc_A[2] = (float32_t)IFBW_PPB;
4. resolver1.sin_samples[0] = (float32_t)R_SIN1;
5. resolver1.sin_samples[1] = (float32_t)R_SIN2;
6. resolver1.cos_samples[0] = (float32_t)R_COS1;
7. resolver1.cos_samples[1] = (float32_t)R_COS2;
8. motor1.dcBus_V = (float32_t)VDC_EVT;
9. ADC_readResult(CSL_CONTROLSS_ADC1_RESULT_U_BASE, ADC_SOC_NUMBER0)
10. ADC_readPPBResult(CSL_CONTROLSS_ADC1_RESULT_U_BASE, ADC_PPB_NUMBER1)

为显示在图形窗口中读取和绘制 ADC 的示例，日志指针连接到了三相电流，并调用日志函数。通过在表达式窗口中右键点击 gLog_CH[7] 并选择图形，空载时的 A 相电流将绘制到图形窗口中，如图 2-34 所示。在这种情况下，A 相电流指向 gLog_CH[7]，如以下列表中所示。它可以分配给任何日志通道。要将 gLog_CH 添加到表达式窗口，只需右键点击它并添加到监视表达式。如需更多详情，请查看 CCS 教程。

1. gLog_ptr[7] = &motor1.I_abc_A[0];
2. gLog_ptr[8] = &motor1.I_abc_A[1];
3. gLog_ptr[9] = &motor1.I_abc_A[2];

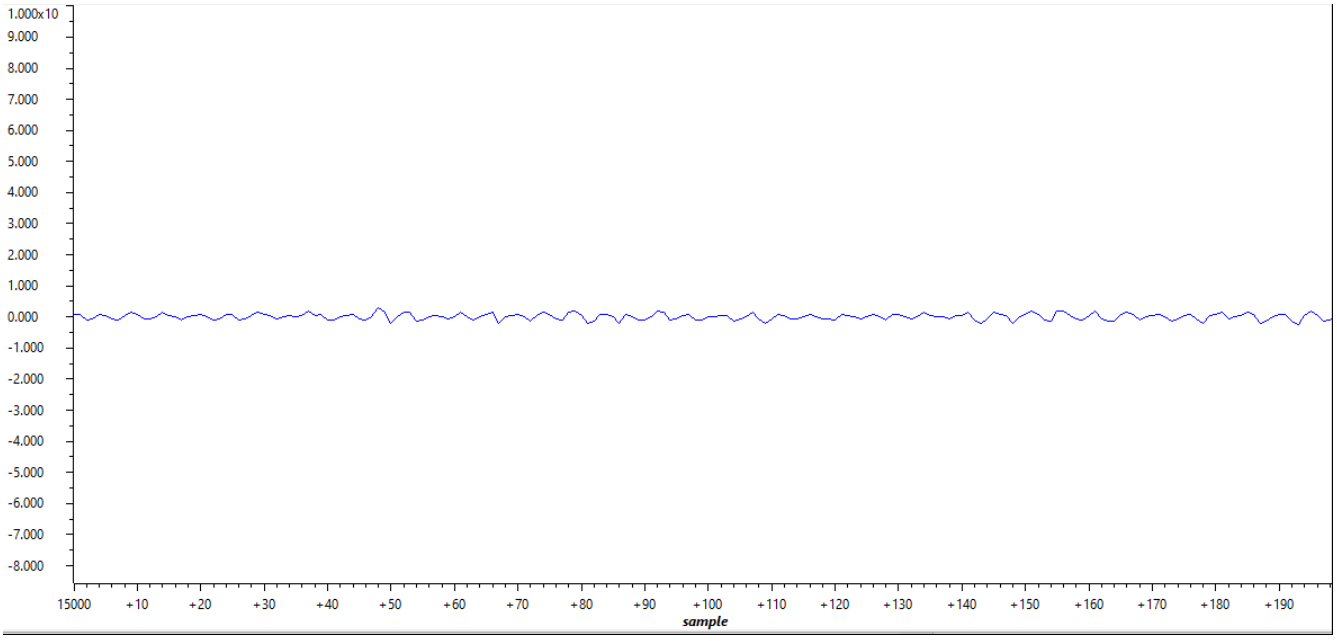

图 2-34. 绘制的空载 A 相电流

在低电压（例如 12V）下启动期间，建议使用来自电源的电压读数覆盖直流总线电压，因为在 12V 下来自 TIDM-02009 的误差不可忽略。

2.5 生成空间矢量 PWM 和在开环中驱动电机

空间矢量 PWM 是一种经典的电机控制方法。此部分仅显示高级函数，而不像教科书那样详细。高级函数可轻松替换为 C28 库或其他控制器库中的相似函数。关键在于，减少使用本地变量并将全局变量分配给 TCM，以获得确定性的执行时间。

2.5.1 设置 SVPWM 发生器输入

SVPWM 发生器的输入为 Vd 和 Vq。需要调用以下行来赋值。Motor1 是存储在 TCM 中的结构。在程序文件中可找到有关其定义的更多详情。按住 Ctrl 键简单地左键点击几次变量名称，有助于追踪它的定义位置。代码逻辑与 TIDM-02009 的 C28 程序相同。Vd 和 Vq 是实际值，而不是标幺值。

```
1.  motor1.Vout_dq_V[0] = VdTesting;
2.  motor1.Vout_dq_V[1] = VqTesting;
```

以下行生成电机速度和电机角度。行 1 至行 4 设置斜坡控制器 rc1 和斜坡发生器 rg1。SpdRef 是 0 和 1 之间的标幺值。在行 5 和行 6 中，生成的 omega 和 theta 分配给 motor1。行 7 将 theta 限制在 0 到 TWO_PI 的范围内。文件中定义了 TWO_PI 值。按住 Ctrl 键简单地左键点击几次变量名称，有助于追踪它的定义位置。值得注意的是，在启动硬件中断之前，需要相应地初始化 rc1、rg1 和 motor1。

```
1.  rc1.TargetValue = SpdRef;
2.  rampControl(&rc1);
3.  rg1.Freq = rc1.SetpointValue;
4.  rampGen(&rg1);
5.  motor1.omega_e = rg1.Freq * BASE_FREQ * TWO_PI;
6.  motor1.theta_e = rg1.Out * TWO_PI;
7.  theta_limiter(&(motor1.theta_e));
```

接下来的几行将输入送到 SVPWM 发生器，并保持标幺值输出。行 1 将输入限制在一定范围内。行 2 是 Park 逆变换。在 CMSIS DSP 库和其他地方可找到相似的函数。角度信息已经包含在 motor1 的结构中。行 3 是 SVPWM 发生器。逻辑与 TIDM-02009 的 C28 程序相同。以前的 C28 库中还有其他实现方式。值得注意的是，此版本中有实际值到标幺值的转换。行 4 将标幺值输出限制在一定范围内。

```
1. dq_limiter_run(&motor1);
2. ipark_run(&motor1);
3. SVGEN_run(&motor1, &pwm1);
4. PWM_clamp(&pwm1);
```

SVPWM 生成后，标幺值输出通过下面行 1 中的函数传递到 EPWM 计数器比较。行 2 提供有关设置 EPWM0 计数器比较的详情。EPWM_setCounterCompareValue 是设置计数器比较值的 SDK API 的名称。对于向上/向下模式或中心线模式，此处计算该值。

```
1. TRINV_HAL_setPwmOutput(&pwm1);
2. EPWM_setCounterCompareValue(CONFIG_EPWM0_BASE_ADDR, EPWM_COUNTER_COMPARE_A,
   (uint16_t)((pwm->inv_half_prd * pwm->Vabc_pu[0]) +pwm->inv_half_prd));
```

2.5.2 在图形窗口中读取 SVPWM 占空比

就像节 2.4.3 中在图形窗口绘制 ADC 样本一样，以下标幺值输出需要连接到日志指针。

```
1. gLog_ptr[4] = &pwm1.Vabc_pu[0];
2. gLog_ptr[5] = &pwm1.Vabc_pu[1];
3. gLog_ptr[6] = &pwm1.Vabc_pu[2];
```

图 2-35 中呈现了 A 相占空比的绘制图形。

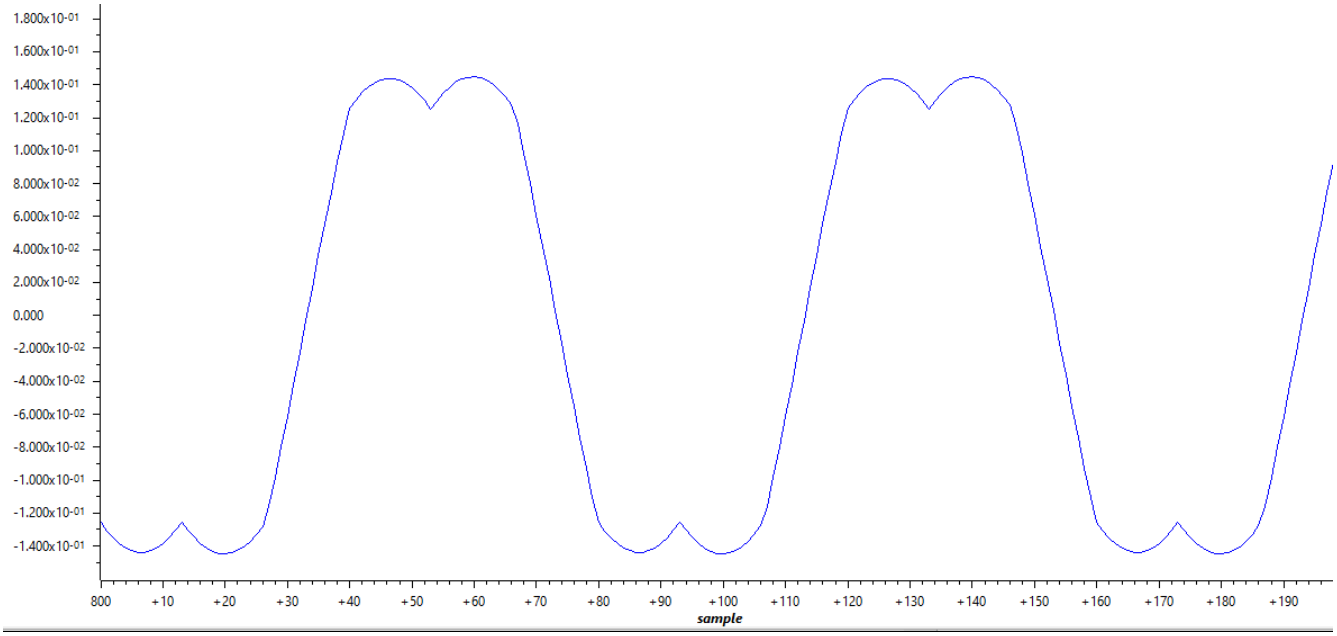

图 2-35. A 相占空比

2.5.3 逆变器上电并在开环中旋转电机

逆变器上电后，便可以在开环中旋转电机。建议起始速度为低速。标幺值全局变量 SpdRef 可设置在 0.01 左右。几个标志控制所列示例程序的执行。“runMotor”是“gTFlag_MockTheta”和“gTFlag_SpdDemo”的栅极。如果“runMotor”为“FALSE”，不会发送速度命令。“gTFlag_MockTheta”将在开环和电流闭环中使用仿真角度和速度。“gTFlag_SpdDemo”将提供速度命令，以演示速度闭环。“gTFlag_MockTheta”和“gTFlag_SpdDemo”不应该同时为“TRUE”。必须使用“runMotor”停止电机，然后在“gTFlag_MockTheta”和“gTFlag_SpdDemo”之间切换。当“gTFlag_MockVdq”为“TRUE”时，在 SVPWM 生成前来自程序的 Vd 和 Vq 将通过从表达式窗口中手动输入进行覆盖。当“gTFlag_MockId”或“gTFlag_MockIq”为“TRUE”时，电流环路控制器输入端的电流值将通过从表达式窗口手动输入进行替换。

- runMotor
- gTFlag_MockTheta
- gTFlag_MockVdq
- gTFlag_MockId
- gTFlag_MockIq
- gTFlag_SpdDemo

在此部分中，将“gTFlag_MockTheta”和“gTFlag_MockVdq”设置为“TRUE”后，“runMotor”可更改为“TRUE”。正确选择“SpdRef”、“VdTesting”和“VqTesting”后，值得注意的是，Vd 和 Vq 是实际值，而不是标幺值。如节 2.4.3 所述，可读取 A 相电流。图 2-36 中绘制了它。在接通低频交流电流后，电机应该开始旋转。否则，建议检查电机、逆变器和控制卡。逆变器硬件详情可参见 TIDM-02009。用户指南中应提供控制卡详情。

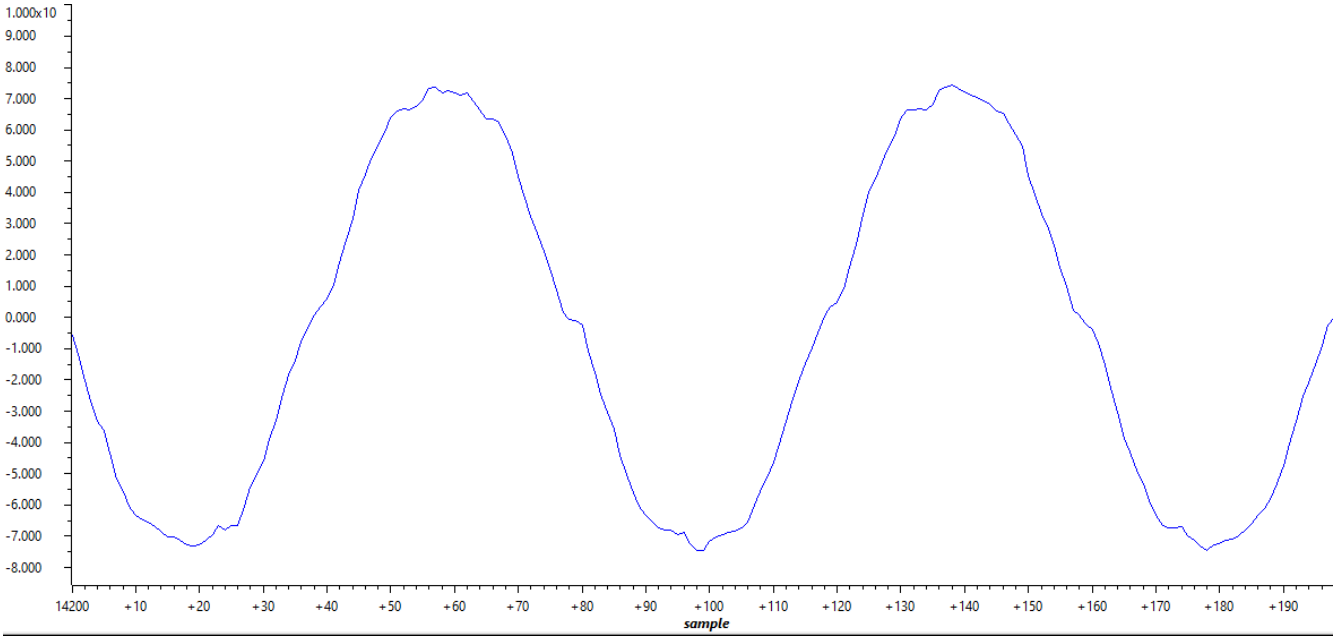

图 2-36. A 相电流开环

2.6 以模拟速度闭合电流环路

此部分要以模拟速度闭合电流环路。模拟速度由节 2.5.1 中提出的斜坡创建。手动分配 Id 和 Iq 基准。

2.6.1 添加变换和读取开环中的 Id-Iq

要闭合电流环路，需要添加以下变换。行 1 是 Clark 变换，行 2 是 Park 变换。在 CMSIS DSP 库和其他地方可找到相似的函数。角度信息已经包含在 motor1 的结构中。这里的实现与 TIDM-02009 的 C28 程序相似。

1. clarke_run(&motor1);
2. park_run(&motor1);

理想情况下，Id 和 Iq 在开环稳态运行时接近恒定值。大多数情况下，在表达式窗口中读取它们并不难。如果需要图形视图，可按照节 2.4.3 将它们添加到图形视图。以下是日志指针的设置。行 1 是 Id，行 2 是 Iq。绘制的 Id 和 Iq 分别位于图 2-37 和图 2-38 中。

1. gLog_ptr[10] = &motor1.I_dq_A[0];
2. gLog_ptr[11] = &motor1.I_dq_A[1];

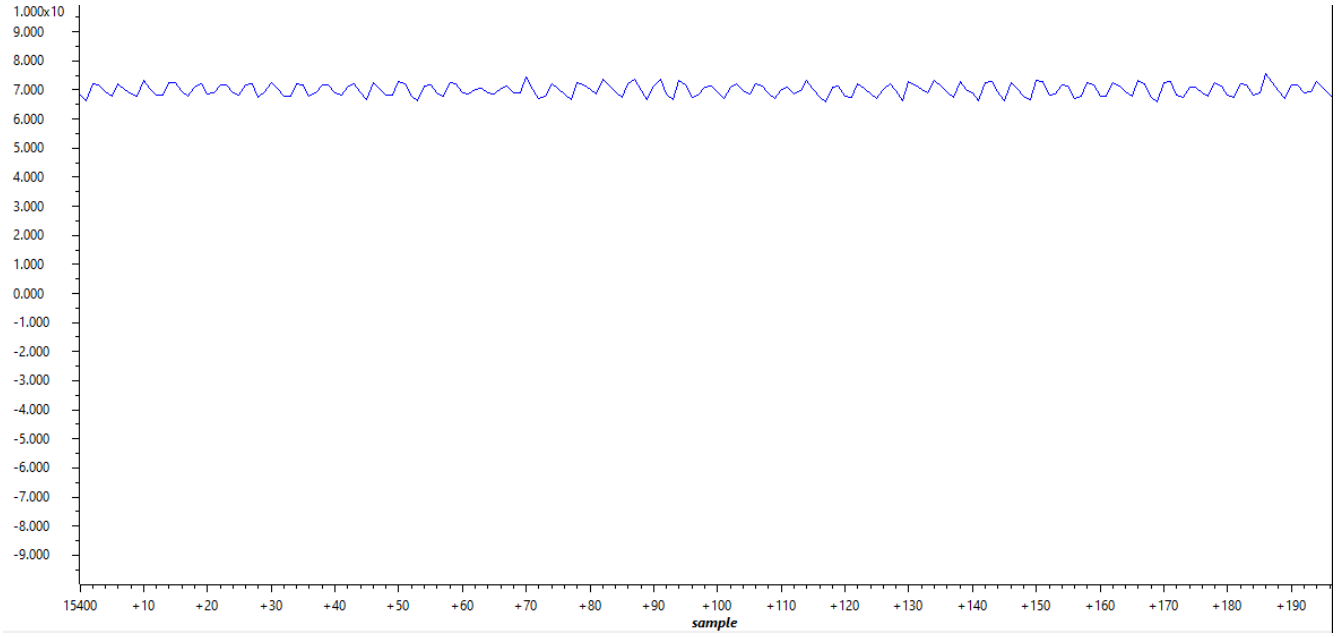

图 2-37. 开环 Id

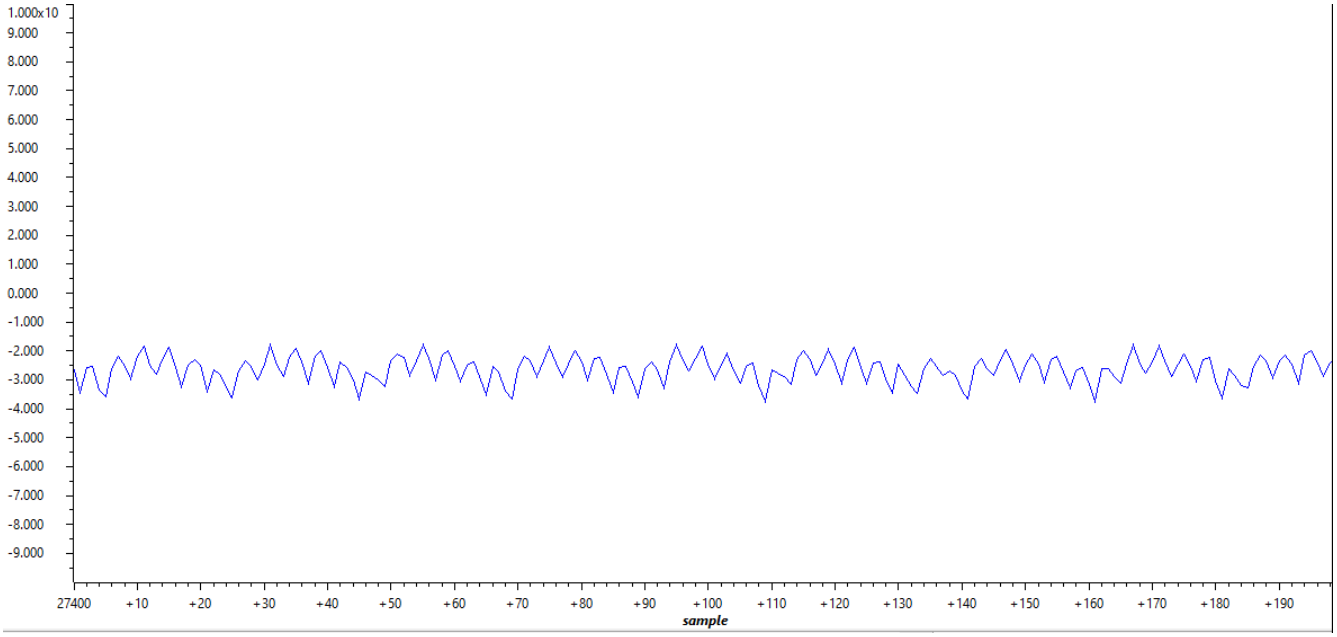

图 2-38. 开环 Iq

2.6.2 添加控制器，以闭合电流环路

必须添加 PI 控制器，以闭合电流环路。这里 PI 控制器的实现与 TIDM-02009 的 C28 程序相似。CMSIS DSP 库和其他地方有不同的实现。这里无意介绍 PI 控制器的详情。主要信息是确保控制器位于 TCM 中。从 TCM 运行对于确定性的执行时间至关重要。否则，在缓存和 OCRAM 之间交换内容或在 OCRAM 中直接运行将占用大量的

时间。行 1 中提供了属性设置示例。行 2 和 3 中显示了一个函数调用的另一个示例。属性设置必须将链接命令文件和存储器保护单元配置匹配起来。更多详情可参见技术参考手册。

```
1. __attribute__ ((section(".tcmb_code"))) static inline float32_t PI_run_series(PI_Obj * pi)
2. motor1.pi_id.fbackValue = motor1.I_dq_A[0];
3. motor1.Vout_dq_V[0] = PI_run_series(&(motor1.pi_id));
```

2.6.3 读取 Id-Iq，以闭合电流环路

理想情况下，Id 和 Iq 在闭合电流环路稳态运行期间是恒定值。大多数情况下，在表达式窗口中读取它们并不难。如果需要图形视图，可按照节 2.4.3 将它们添加到图形视图。以下是日志指针的设置。行 1 是 Id，行 2 是 Iq。绘制的 Id 和 Iq 分别位于图 2-39 和图 2-40 中。

```
1. gLog_ptr[10] = &motor1.I_dq_A[0];
2. gLog_ptr[11] = &motor1.I_dq_A[1];
```

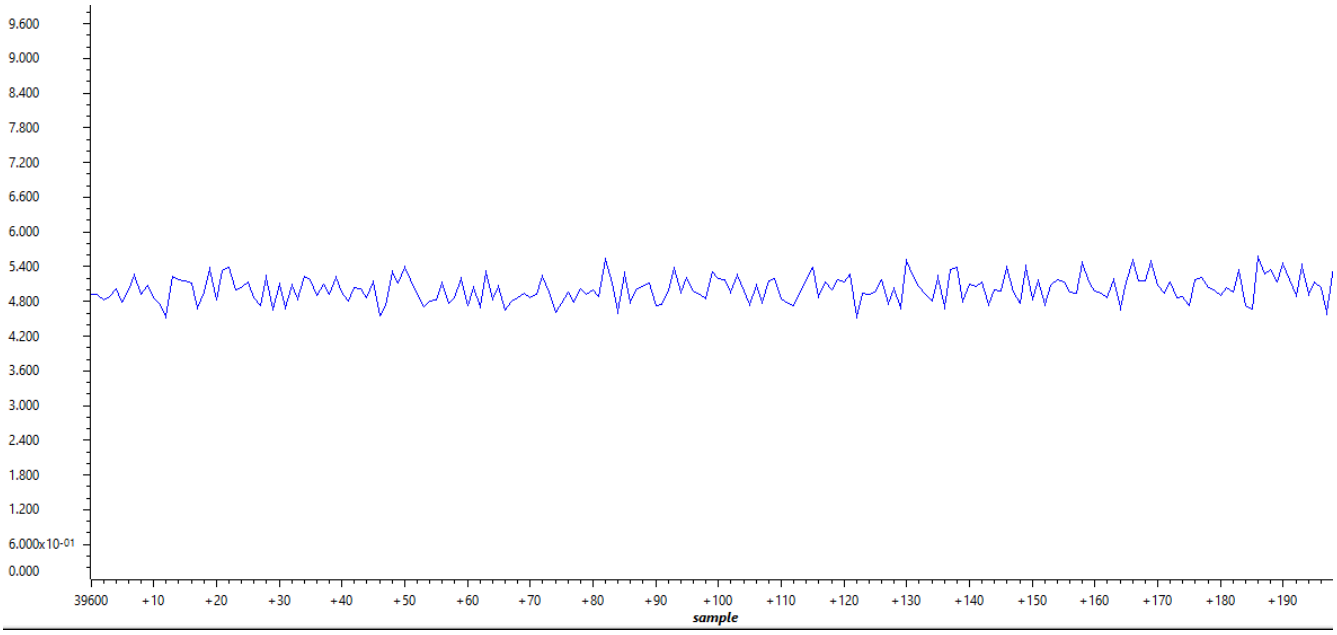

图 2-39. 闭环 Id

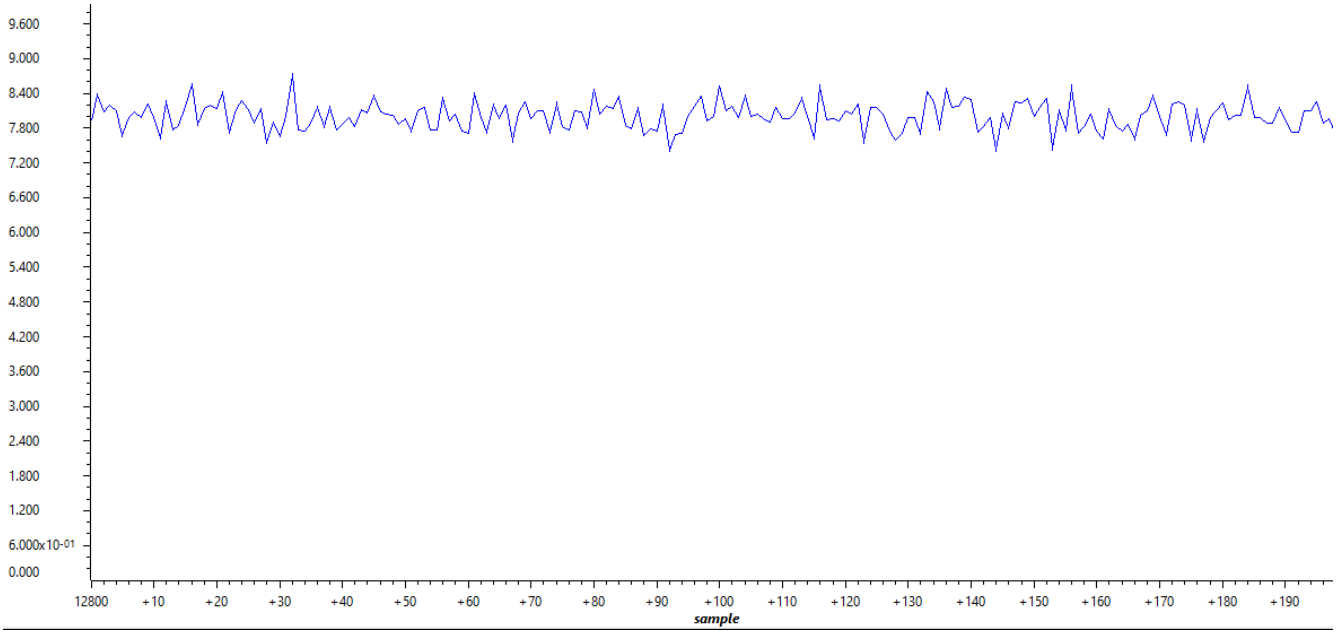

图 2-40. 闭环 Iq

2.7 添加软件旋转变压器数字转换器

软件旋转变压器数字转换器是使用节 2.2.2 和节 2.2.3 中配置的 ADC、DMA 和 DAC 实现的。图 2-41 总结了配置概况。在本文中，PWM X 是 EPWM0，PWM Y 是 EPWM7。10kHz 的 EPWM0 触发 A 相电源开关和同步源。20kHz 的 EPWM7 专用于通过 DAC 触发 EDMA0，以产生旋转变压器激励。ADC 转换开始 (SOC) 是在 EPWM0 计数为零时触发的。ADC4 的转换结束 (EOC) 是包含 FOC 环路的 ADC INT1 的源。此部分将介绍软件旋转变压器的功能和读数。

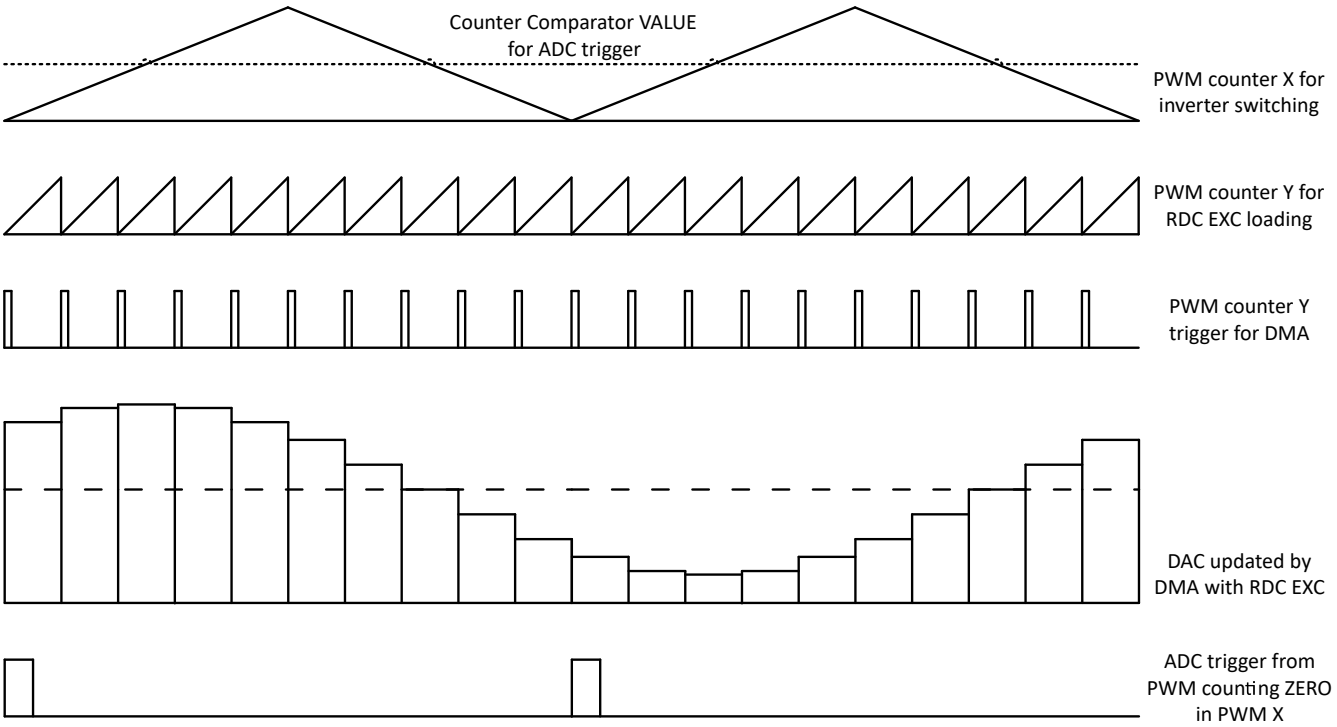

图 2-41. 软件旋转变压器同步和激励

2.7.1 为旋转变压器硬件生成激励

在节 2.2.3 中，EDMA 配置为每当发生 EPWM7 SOCA 事件时传输数据。为指定关于传输的更多细节，下面行 1 中定义了函数，以初始化 EDMA，包括数据位置、数据大小和 DAC 值寄存器的位置。为更新详情，下面行 2 中定义了函数，以更改 EDMA 输入表和 DAC 激励输出的内容。

- uint16_t RDCexc_start(uint16_t *table, uint16_t table_size, EDMA_Handle dma_handle, uint32_t dma_ch, uint32_t dac_base)
- uint16_t RDCexc_update(uint16_t *table, uint16_t table_size, EDMA_Handle dma_handle, uint32_t dma_ch, uint32_t dac_base)

RDCexc_start 应该在控制环路运行前调用，RDCexc_update 可以在控制环路运行期间调用。RDCexc_update 的目的是减少激励相位与采样时间之间的移位。通过在调用 RDCexc_update 之前偏移输入表的指针来调整激励相位。以下行中提供了两个函数的示例。

1. RDCexc_start(gRDCtable_ptr,20,gEdmaHandle[0],DMA_TRIG_XBAR_EDMA_MODULE_0,CONFIG_DAC0_BASE_ADDR);
2. RDCexc_update(gRDCtable_ptr,20,gEdmaHandle[0],DMA_TRIG_XBAR_EDMA_MODULE_0,CONFIG_DAC0_BASE_ADDR);

全局作用域中表的一个元素连接到 gRDCtable_ptr。表中包含多个完整正弦波周期的点。该元素位于表的中间，以便在工作期间向右或向左偏移指针，以调整激励相位。一个周期中有 20 个点。EDMA 输入表的长度为 20。EDMA 句柄、XBAR 信息和 DAC 地址可在配置文件中找到。

值得注意的是，TIDM-02009 母板与 AM263x 控制卡的某些版本之间可能存在 DAC 通道引脚失配。权变措施是在母板上旋转变压器板激励引脚和 DAC-A 引脚之间焊接一根蓝线。图 2-42 中提供了母板俯视图，图 2-43 中显示了连接。

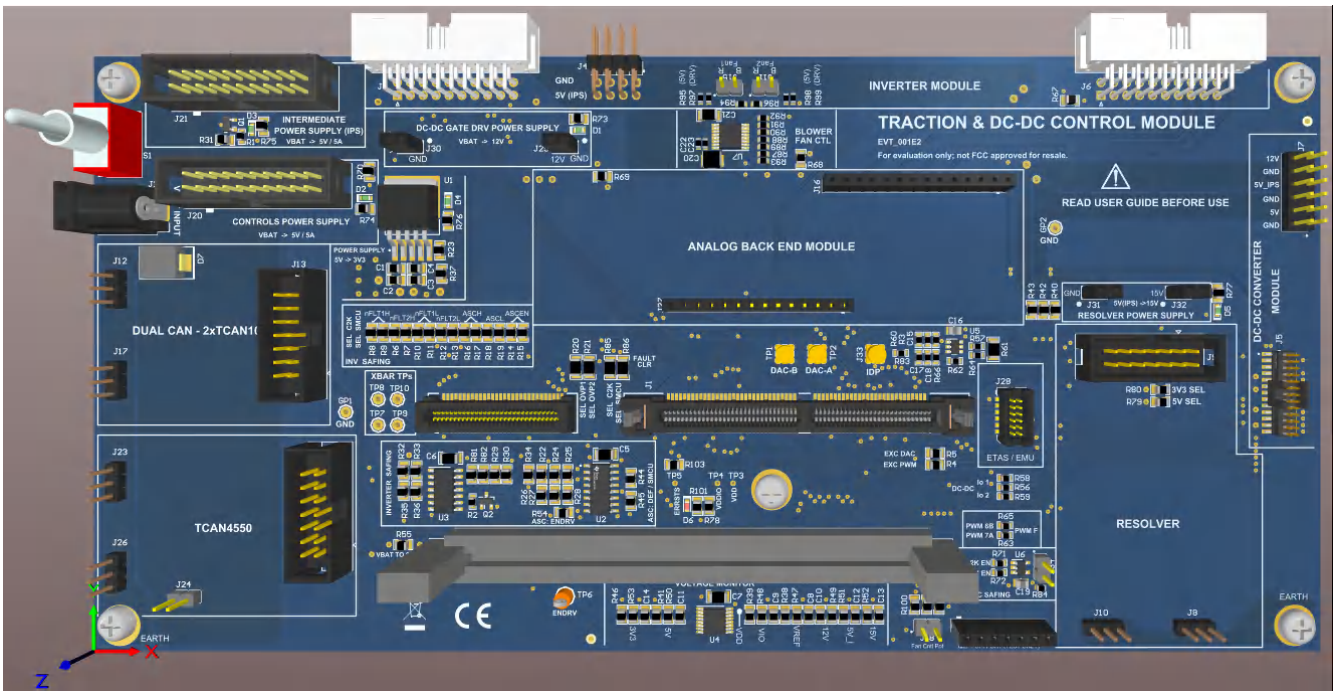

图 2-42. 母板俯视图

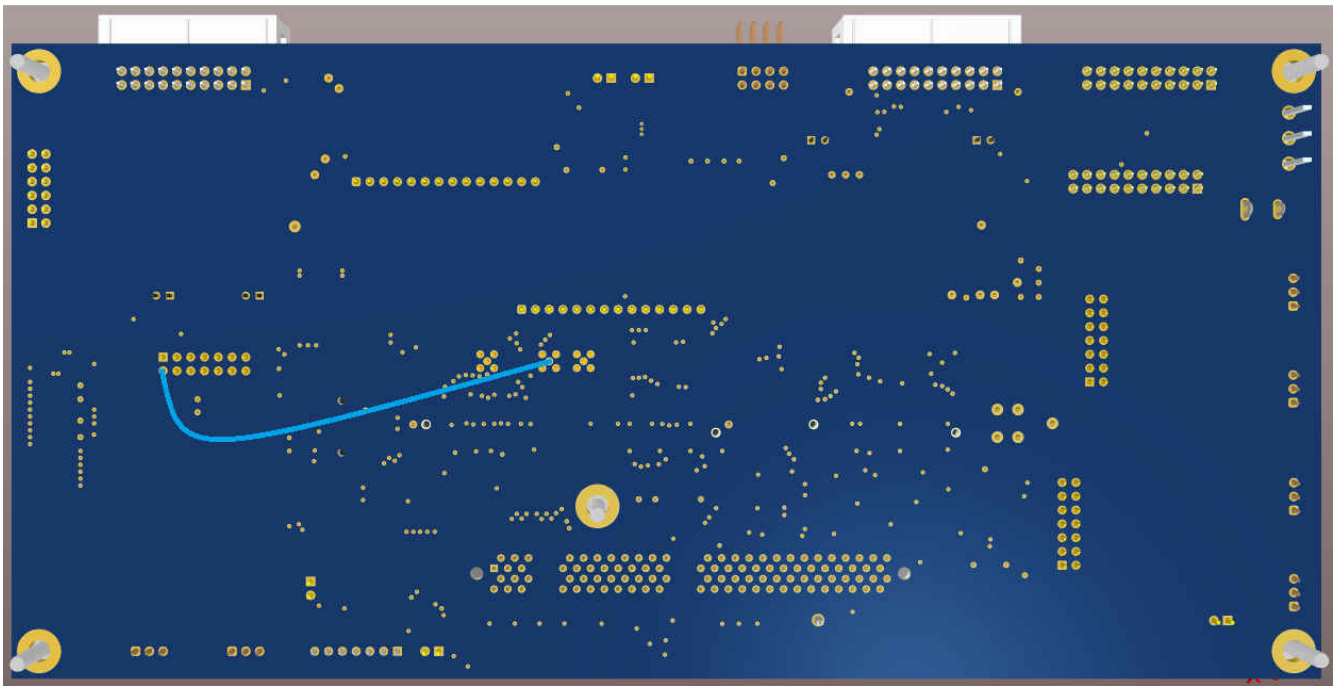

图 2-43. 旋转变压器激励和 DAC-A 间的母板蓝线

2.7.2 添加旋转变压器软件

软件旋转变压器的实现与 TIDM-02009 中的 C28 相似。两个函数如下所列。行 1 用于初始化旋转变压器结构，行 2 用于计算角度和速度。初始化结构后，需要根据旋转变压器硬件的特性更新实现细节。有关详细信息，请参见结构定义。

1. static inline void resolver_init(Resolver_t *resolver);
2. static inline void resolver_run(Resolver_t *resolver);

值得注意的是，实现 sin/cos 等数学函数的方法有很多。在此设计中，旋转变压器锁相环输入的 sin/cos 函数从“resolver_run”中移出，因此更容易找到所有数学函数并进行更改。以下行 1 和 2 必须在上面的行 2 之前调用。这里使用标准 C 库函数举例。

1. resolver1.res_theta0_sin = sinf(resolver1.res_theta0);
2. resolver1.res_theta0_cos = cosf(resolver1.res_theta0);

2.7.3 读取旋转变压器软件输出

在初始化时，软件旋转变压器的输出使用指针传递给全局变量，如下面的行 1 和行 2 所示。由于在旋转过程中电机角度总是变化，所以若不以固定采样频率在图形窗口中绘制它，便没有意义。行 3 和行 4 将旋转变压器角度和速度的地址传递给两个日志指针。节 2.4.3 中所述的日志函数每当中断指定数量便记录它们。

1. resolver1.resolver_theta = &resolver_theta;
2. resolver1.resolver_omega = &resolver_omega;
3. gLog_ptr[12] = &resolver_theta;
4. gLog_ptr[13] = &resolver_omega;

2.8 以转子速度闭合速度环路

2.8.1 添加速度环路控制器

必须添加 PI 控制器，以闭合速度环路。这里 PI 控制器的实现与 TIDM-02009 的 C28 程序相似。由于本文中使用的电机是感应电机，因此有另一滑移补偿器件可闭合速度回路。根据终端设备的实际电机拆卸或更换它。此处意在演示如何使用框架。以下行 1 和 2 显示 PI 控制器函数调用，行 3 到 6 显示滑移补偿。由于行 6 中有除法，因此添加了行 4 和 5，以避免除数为零。

1. motor1.pi_spd.fbackValue = resolver_omega;
2. motor1.pi_iq.refValue = PI_run_series(&(motor1.pi_spd));
3. aci1.IMDs += aci1.Kr * (motor1.I_dq_A[0] - aci1.IMDs);
4. aci1.IMDs = ((aci1.IMDs < 0.001) && (aci1.IMDs >= 0)) ? 0.001 : aci1.IMDs;
5. aci1.IMDs = ((aci1.IMDs > -0.001) && (aci1.IMDs < 0)) ? -0.001 : aci1.IMDs;
6. aci1.Wslip = aci1.Kt * motor1.I_dq_A[1] / aci1.IMDs;

2.8.2 添加速度环路演示程序

为演示速度环路，创建了一个简短的演示，向前旋转电机 10 秒，向后旋转电机 10 秒，然后停止。这是一个无限循环，有一个以中断频率定时的计数器。这部分程序如图 2-44 所示。由于速度环路基准的输入以电气速度的 rad/s 为单位，因此需要根据电机特性将机械速度的 RPM 输入转换为电气速度。

```
gDemoCnt++;
if(gDemoCnt < gDemoPos )
{
    gSpeedRef = 0;
}else if(gDemoCnt < gDemoNeg)
{
    gSpeedRef = gDemoSpd;
}else if(gDemoCnt < gDemoMax)
{
    gSpeedRef = -gDemoSpd;
}
else
{
    gDemoCnt = 0;
}
/* mech rpm to elec rad/s */
motor1.pi_spd.refValue = gSpeedRef / 60 * PAIRS * TWO_PI;
```

图 2-44. 速度环路演示程序

2.8.3 从图形窗口读取电机速度

电机旋转时，电机角度不断变化。最好通过图形窗口读取它。程序与节 2.4.3 类似。按照下面的行 1 分配指针。

1. gLog_ptr[14] = &gDemoRPM;

电机正向旋转角度如图 2-45 所示，反向旋转角度如图 2-46 所示。角度是从软件旋转变压器创建的。记录从正向到反向转换的正弦包络。

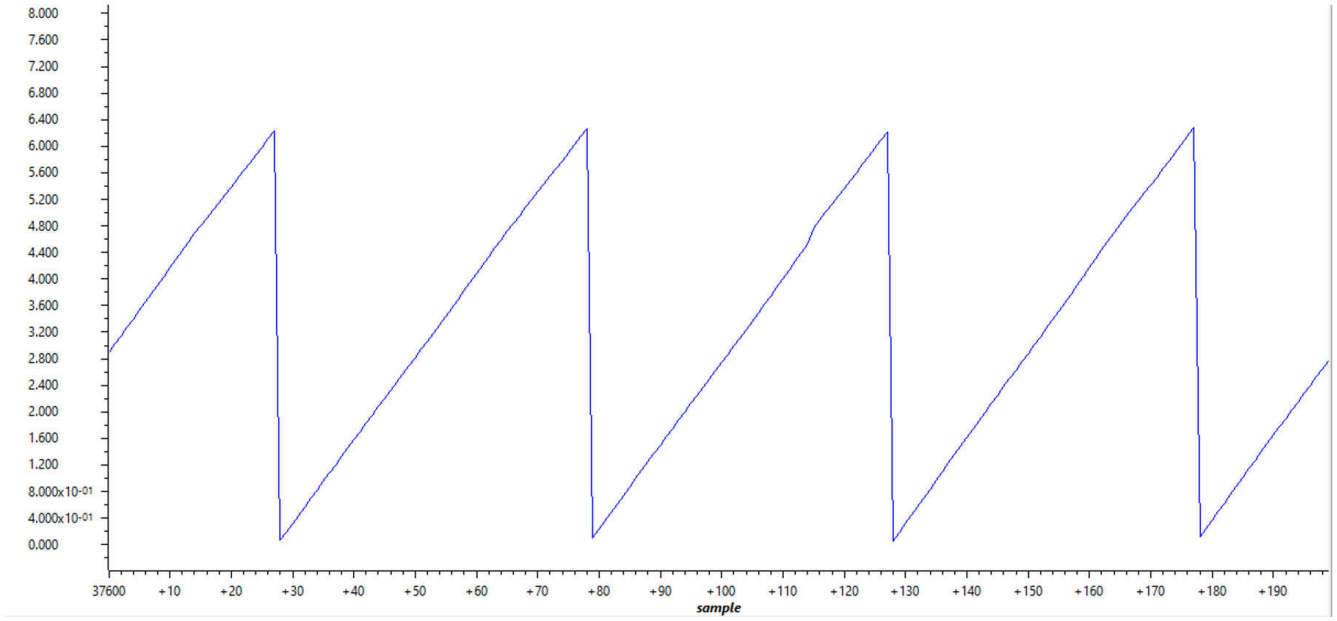

图 2-45. 电机正向旋转

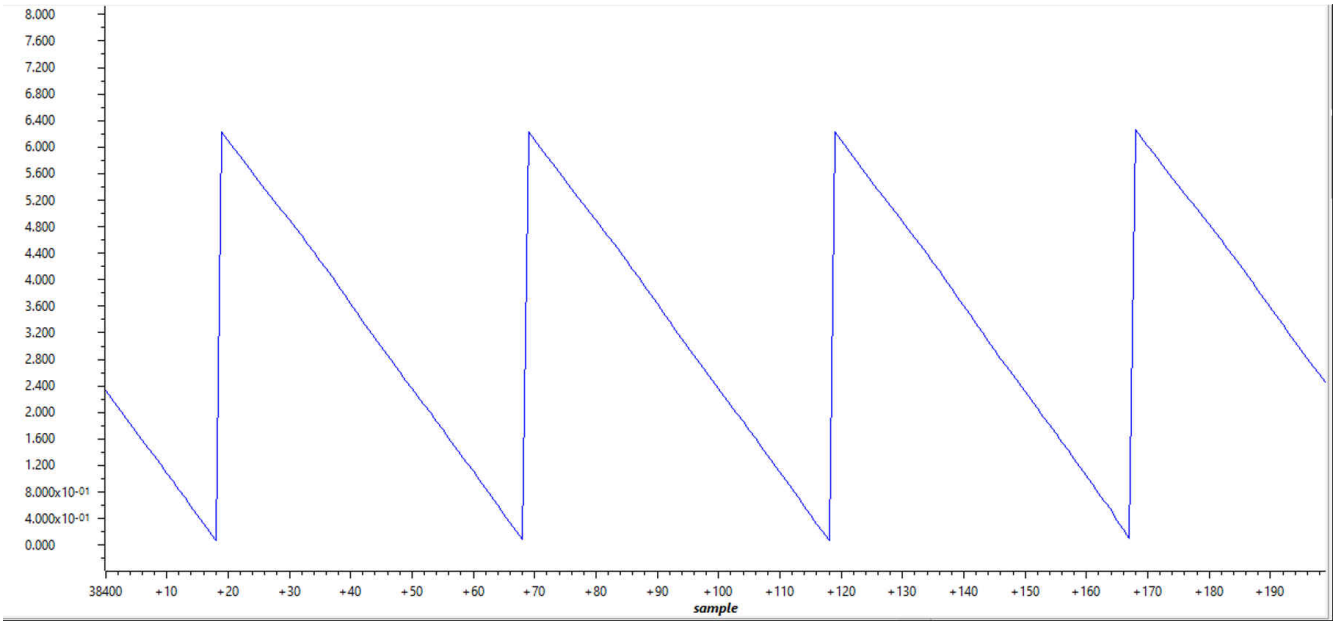

图 2-46. 电机反向旋转

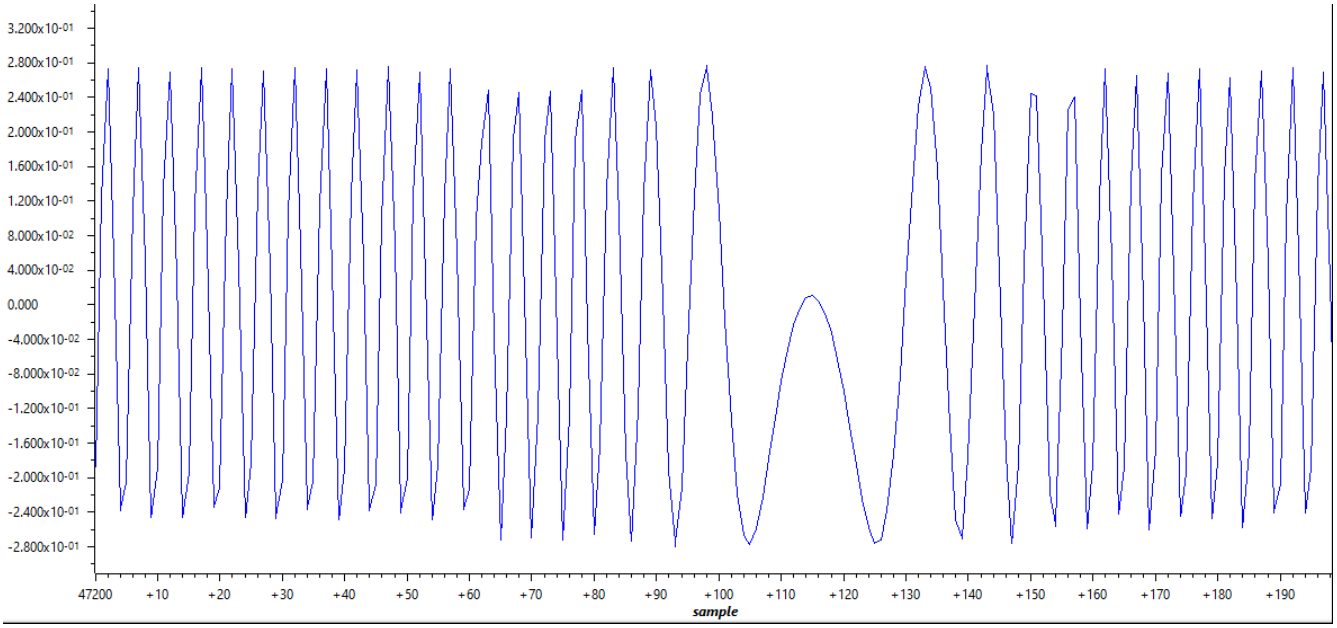

图 2-47. 正向到反向转换的旋转变压器正弦包络

3 代码迁移的简要指南

此部分首先概述了 AM263x SoC 架构和 SDK 资源。请牢记可用的资源，以下两部分总结了从 AM24 和 C28 迁移代码的一些技巧。

3.1 SoC 架构概览

SoC 架构概览如图 3-1 所示。SoC 包括 2 个 R5F 内核集群、基于 L2 存储器系统的 64 位总线、基于控制和连接外设的 32 位总线。另外还有一些资源可以帮助实现安全特性。R5F 集群可以配置为双模式或锁步模式，为优化计算和安全特性提供了机会。L2 存储器系统支持片上存储器、片外存储器、存储器操作、标准以太网、工业以太网和安全等许多特性。控制外设包括所有流行的特性，例如带影子寄存器的 PWM 以及与 PWM 同步的 SAR-ADC。连接外设提供常用的 I2C、SPI、CAN、LIN 和 UART。

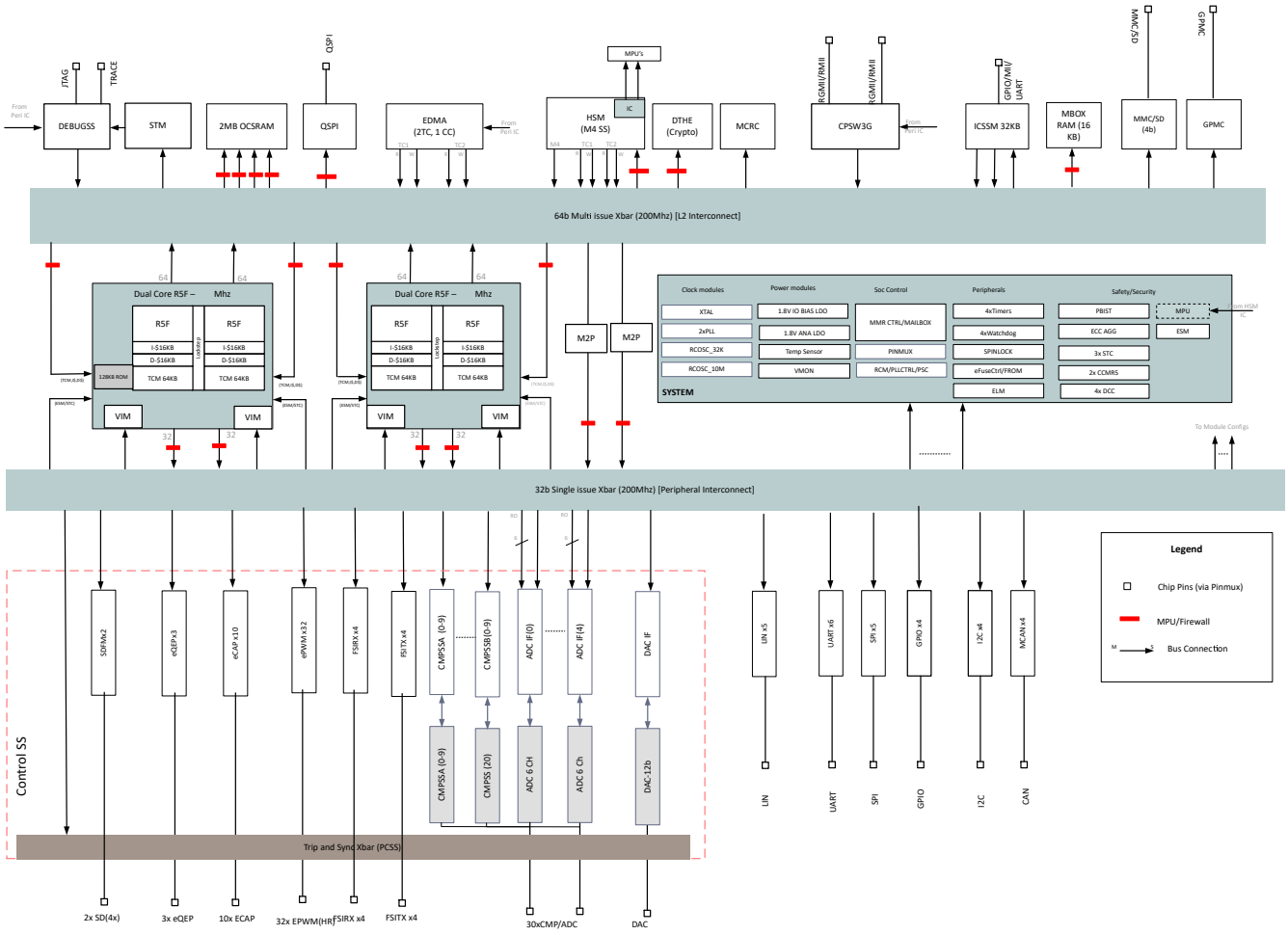

图 3-1. AM263x 方框图

3.2 SDK 资源概览

表 3-1 总结了 SDK 资源概览。在 SDK 安装目录下的“README_FIRST_AM263X.html”中可找到更多详情。默认情况下，目录路径类似于“C:\ti\mcu_plus_sdk_am263x_xx_xx_xx_xx”。“examples/”文件夹是对新手最重要的一个资源。它包括 AM24 匹配连接示例和 C28 控制示例。大多数情况下，在 CCS 中按住 Ctrl 键点击左键，可引导用户进入 API 声明头。由于继承自 C28 和 AM24，控件 API 头包括静态内联的声明和定义，而连接 API 头只显示声明。连接 API 定义必须位于源文件中。头文件和源文件都保存在“source/”文件夹中。如果 CCS 无法获得所需的详细信息，那么这些详细信息很可能保存在“source/”中的某个源文件中。另一个方法是查看“README_FIRST_AM263X.html”的“API Reference”。

表 3-1. SDK 目录结构

文件夹/文件	说明
${SDK_INSTALL_PATH}/	
README_FIRST_AM263X.html	在 Web 浏览器中打开此文件，以获取 SDK 用户指南
makefile	使用“make”构建整个 SDK 的出色 makefile
imports.mak	列出相关工具路径的出色 makefile
docs/	离线 HTML 文档
examples/	跨多个电路板、CPU、NO-RTOS、RTOS 的 AM263X 应用示例
source/	设备驱动程序、中间件库和 API
tools/	工具和实用程序，如 CCS 加载脚本、初始化脚本。
${SDK_INSTALL_PATH}/source/	
board/	电路板外围设备驱动程序
驱动器/	SOC 外围设备驱动程序
industrial_comms/	工业通信协议栈和工业协议 FW HAL（固件和硬件抽象层）
kernel/	这些环境的 NO RTOS 和 RTOS 内核与驱动程序移植层 (DPL)
${SDK_INSTALL_PATH}/examples/	
驱动器/	以 SOC 和电路板为中心的设备驱动程序示例。这些示例基于 NO-RTOS 和 RTOS。
empty/	复制到工作区中的模板项目，并根据应用需求进行定制
industrial_comms/	EtherCAT 从属设备示例

3.3 从 AM24 迁移代码

AM24 与 AM263x 有相似的架构和连接外设。但控制外设完全不同。一般来说，与连接相关的程序可以在很少或没有修改的情况下进行迁移，而与控制外设相关的程序必须针对 AM263x 技术参考手册中的细节进行更新。至于 SDK，每个版本都是独特的。但是 AM24 和 AM263x 在驱动程序移植层的 API 几乎是相同的。不同之处可参见“SOC 特定的设备驱动程序”。

值得注意的是，AM24 与 AM263x 尽管有相似之处，但在架构和连接方面也存在差异。例如，AM24 为工业以太网、千兆位工业通信子系统、更灵活的外部存储器扩展 DDR4 子系统提供更强大的支持。AM263x 中具备 100 兆位工业以太网和 16 位/32 位并行总线特性。与此类特性相关的程序必须重新设计，才能从 AM24 迁移到 AM263x。

另一点是，AM24 R5F 内核频率最高 800MHz，而 AM263x R5F 内核频率为 400MHz。在代码迁移期间，必须料想到并妥当处理执行时间的重大变化。务必确保执行时间保持在要求范围内。但是，对于牵引逆变器，考虑到 400MHz 和 800MHz 内核远高于经典的 MCU，因此大多数情况下，它们的执行时间应该不构成问题。

3.4 从 C28 迁移代码

C28 与 AM263x 有相似的控制外设。但架构和连接外设完全不同。一般来说，与控制外设相关的程序可以在很少或没有修改的情况下进行迁移，而与 CPU、内存管理和连接外设相关的程序必须针对 AM263x 技术参考手册中的细节进行更新。

众所周知，直接操作寄存器在过去的 C28 程序中得到广泛应用。最近几年，从寄存器操作改为了 API 调用。从寄存器操作改为 API 调用可以简化较复杂的 MCU 的采用。但是，从寄存器用户转为 API 用户需要完成一些工作。对于 C28 和 AM263x，这项工作都是不可避免的。完成这项工作后，不难使用 AM263x 控制子系统，因为来自 ADC 和 PWM 等模块的概念非常相似。表 3-2 中提供了有关控制 API 相似性的一些示例。另外，AM263x SDK 还提供强大的 Sysconfig。它提供直观的系统配置用户接口。终端用户可以直接将他们对控制外设的想法应用到配置中，而无需担心 API 细节。控制环路中广泛使用的 API 已经在框架中提供并在节 2 中介绍。

表 3-2. API 定义相似性示例

API 函数	AM263x 定义	C28 定义
获取 ADC 结果	static inline uint16_t ADC_readResult (uint32_t resultBase, ADC_SOCNumber socNumber)	static inline uint16_t ADC_readResult (uint32_t resultBase, ADC_SOCNumber socNumber)
设置 PWM 占空比	static inline void EPWM_setCounterCompareValue (uint32_t base, EPWM_CounterCompareModule compModule, uint16_t compCount)	static inline void EPWM_setCounterCompareValue (uint32_t base, EPWM_CounterCompareModule compModule, uint16_t compCount)

另一方面，尽管二者有相似之处，但在 SDK 和名称相似的某些设计上也有些不同。如节 3.2 所示，AM263x 的 SDK 结构与 C28 的 SDK 非常不同。尽管它们都有相似的控制外设和相似的 API，但仍然必须了解 SDK 结构的不同，以便在开发过程中轻松查找细节。对于 XBAR 等某些特性，C28 和 AM263x 都用 XBAR 同步模块之间的操作，但 AM263x 中的 XBAR 远比 C28 中的 XBAR 更加强大。这也带来一项挑战，必须充分理解和正确配置它。C28 中的 XBAR 程序无法直接应用于 AM263x 项目。

4 总结

牵引逆变器是一种典型的应用，既需要出色的 CPU 吞吐量，又需要灵活的控制外设。本文简要介绍一种牵引逆变器参考设计的架构，其中使用了牵引逆变器的 AM263x 软件框架。然后，重点介绍如何配置、更改和调试使用该软件框架的牵引逆变器参考设计。此处提供分步指导，以帮助用户了解框架细节。另一方面，提供了简要指南，总结了从 AM24 和 C28 系列迁移代码的资源、提示和技巧。在系统设计期间，务必了解特性并进行调整。毕竟，本文演示的是软件框架，可缩短学习曲线，并加速 AM263x 在牵引系统中的采用。

5 参考文献

- *经过 ASIL D 等级功能安全认证的高速牵引和双向直流/直流转换参考设计*
- *AM263x Sitara™ 微控制器数据表*
- *AM263x Sitara 处理器技术参考手册*
- *AM263x 控制卡用户指南*

重要声明和免责声明

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

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

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

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

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265