AN-1267
应用笔记

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

使用ADSP-CM408F ADC控制器的电机控制反馈采样时序

作者：Dara O'Sullivan、Jens Sorensen和Aengus Murray

简介

本应用笔记介绍ADSP-CM408F模数转换器控制器(ADCC)模块的主要特性，重点讨论该产品在高性能电机控制应用的电流反馈系统中的相关性与可用性。

本应用笔记的目的是为了强调模数转换器(DAC)模块的关键功能，并提供针对电机控制应用的配置指南。本文提供演示ADI ADCC驱动器的代码示例。

有关此ADCC的所有功能、配置寄存器和应用程序接口(API)的更多详细信息，请参阅ADSP-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F产品页面和采用ARM Cortex-M4和16位ADC开发产品的ADSP-CM40x混合信号控制处理器的产品页面上提供的《采用ARM Cortex-M4的ADSP-CM40x混合信号控制处理器硬件参考指南》。

虽然本应用笔记重点讨论电流反馈，类似的配置和应用原理同样适用于其他信号的反馈与测量。

同样，虽然本应用笔记重点讨论ADSP-CM408F，但原理在本质上同样适用于ADSP-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F系列的其他产品。

目录

修订历史

2014年9月 — 修订版0至修订版A

更改“简介”部分 1
更改图2 3
更改图3 4
更改“电流反馈调整”部分 5
更改“ADC操作时序”部分 7
增加“采样时刻调整”部分和图11；重新排序 8
增加“利用触发路由提供增强型精确采样时序”部分 12
更改“示例代码”部分 13

2013年9月—修订版0：初始版

电流反馈系统概述

电机控制应用中的电流反馈示例如图1所示。该配置常用于高性能电机驱动，并针对电机相位绕组电流采样，而非对逆变器低端相位引脚采样。中高电平时，电流传感器或变压器(CT0和CT1)必须用于电流测量路径，因为阻性分流器尺寸过大而低效。

在图1的配置中，处理器位于安全的隔离栅低压侧，而信号隔离通常为CT0和CT1所固有，且微处理器的脉冲宽度调制PWM输出和栅极驱动器之间还存在数字隔离。

通常需要在电流传感器输出和ADC输入之间进行一些信号调理，以便实现范围匹配和高频噪声滤波。随后将调理的电流测量信号施加于ADC输入，用来采样和转换。对每个ADC输入进行一次绕组电流测量将使能电流测量的同步采样以获得更高的控制环路精度，从而增强性能。另外，还可在硬件内直接配置采样时间与PWM sync脉冲的同步。

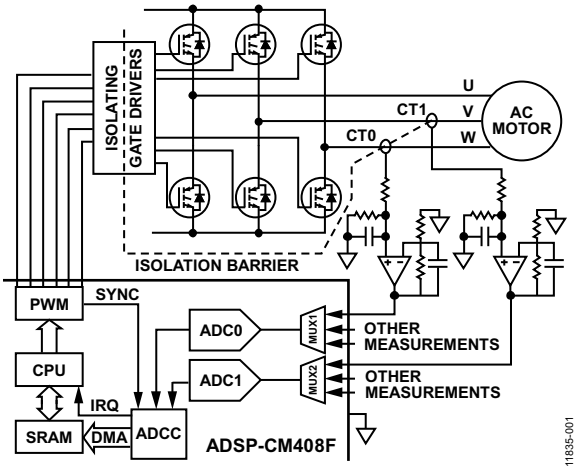

图1. 电机控制中ADSP-CM408F ADC的电流反馈

这些特性可使能PWM周期中相位电流测量点的精密时序。将这些测量时刻与零矢量的中间点或PWM周期的中间点对齐，确保电流采样电平等效于忽略开关纹波的瞬时平均电流。

图2中显示了零矢量中点和PWM周期中点处的同步U相位和V相位采样。

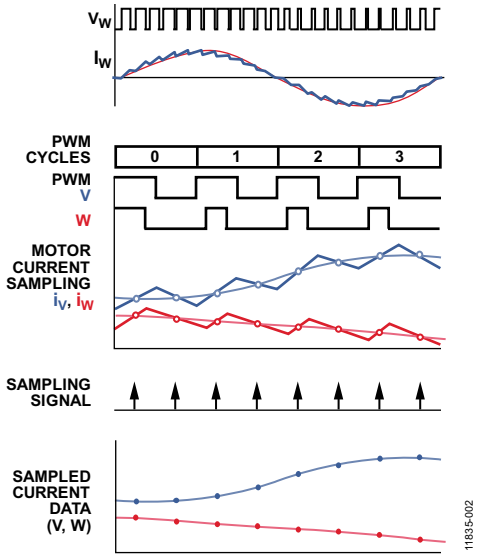

图2. 平均电流采样图解

完成数据转换后，便可将其通过直接存储器访问(DMA)传输至控制器静态随机存取存储器(SRAM)，完成传输后会生成一个中断。在内核模式下，通过存储器映射寄存器，还可实现直接ADC状态和数据读取，但这种方法需消耗更多的处理器开销。

通常还会采样其他模拟信号，例如直流总线电压、隔离式栅极双极性晶体管(IGBT)温度和电机位置正弦与余弦输出。虽然本应用笔记重点讨论电流反馈，但很多信息也与系统中的其他测量参数有关。

ADC模块概述

该ADC采用双通道、16位、高速、低功耗、逐次逼近型寄存器(SAR)设计，精度高达14位。

输入多路复用器最多支持连接两个独立受控ADC的26个模拟输入源组合(每个ADC的12路模拟输入加上1路DAC回送输入)，任意时刻都对两个通道同步采样。ADC转换时间快达380 ns。单端模拟输入所需的电压输入范围为0 V至2.5 V。

多路复用器和ADC之间提供片内缓冲器，无需使用额外的外部信号调理ADSP-CM408F。此外，每个ADC都有一个片内2.5 V基准电压源，当优先选择外部基准电压源时可将其过驱(通过ADCC_CFG寄存器选择该选项)。

ADSP-CM408F中的总模拟子系统的图形概述如图3所示。ADSP-CM408F采用多芯片系统级封装(SiP)，而ADC硅片制造工艺与处理器硅片工艺有所不同，如图3所示。

ADCC负责ADC中与处理器的时序同步，并管理DMA，将采样数据传输到SRAM。

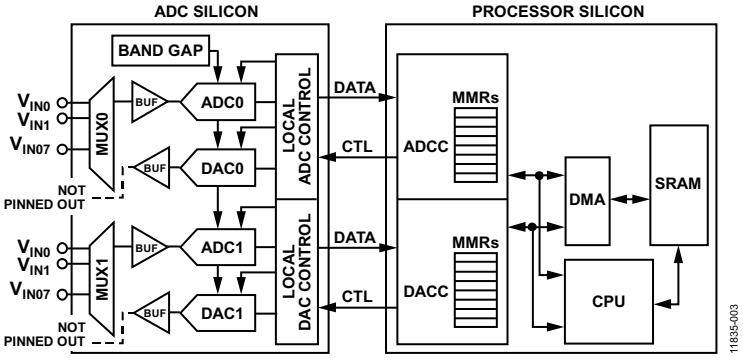

图3. ADSP-CM408F模拟子系统

电流反馈调整

若要最大限度地正确利用ADC的能力，正确调整反馈信号非常重要。信号通过反馈路径处理，如图5所示。双极性相位绕组电流I_W通过电流传感器(或变压器)和信号调理电路的组合功能转换为ADC输入端的单极性电压。

电流传感器的传递函数由下式表示：

$$V_{IW} = K_{CT}I_W + V_{0CT}$$

其中：

V_{IW}为输出电压。

K_{CT}是传感器的线性增益系数。

V_{0CT}是传感器的零电流失调电压。

K_{CT}在不同传感器类型的某些电流水平下表现出非线性，且为了获得更佳的精度，应当与I_W成函数关系，即$K_{CT}(I_W)$。之后，ADC输入电压可表示为：

$$V_{IW\_ADC} = K_{SIG}V_{IW} = K_{SIG}[K_{CT}(I_W)I_W + V_{0CT}]$$

其中，K_{SIG}是信号调理电路的低频增益。

该单极性电压转换为16位无符号整数，并由DMA传输至处理器存储器，然后发出中断，提醒控制程序新数据样本可用。ADC理想化的传递函数如下所示：

$$N_{IW} = K_{ADC}V_{IW\_ADC} = \frac{2^{16}}{2.5}V_{IW\_ADC}$$

其中：

N_{IW}是ADC数字输出字。

K_{ADC}表示ADC的线性增益，等于根据输入电压范围划分的ADC分辨率，如上所示。

ADC输出会产生一些失调；而在软件内进行一些失调补偿(N_{ADC_OFFSET})通常是一个好办法，可将ADC自身的所有失调以及传感器和信号调理电路产生的所有残余失调从ADC输出中去除。该值可在零电流周期(如系统启动或禁用驱动输出)中动态更新。

最后，电流传感器零电流失调电压N_{CT_OFFSET}的数字表示从ADC输出信号中去除，使带符号值IW(与实际相位绕组电流有关)的表达式为：

$$IW = K_{ADC}(K_{SIG}[K_{CT}(I_W)I_W + V_{0CT}]) - N_{ADC\_OFFSET} - N_{CT\_OFFSET}$$

其中：

$$N_{CT\_OFFSET} = \frac{2^{16}}{2.5}V_{0CT}$$

这个带符号的16位值可转换为浮点值，或直接使用，具体取决于控制器实现方案。若要最大限度地利用ADC的全范围，则系统中的正峰值受控电流必须与ADC输入电压2.5 V相对应，而负峰值受控电流与ADC输入0 V相对应。

该情况的一个示例如图4所示。该图显示了典型电流波形和相关的各种零电平、峰值电平以及标称电平，图4显示的电流电平将转换为通过信号测量系统传播(如图5所示)的比例量(参见表1)。

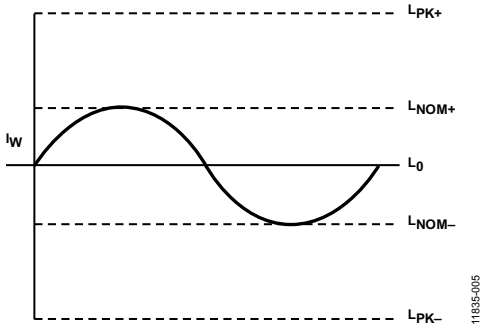

图4. 电流反馈信号幅度

表1. 电流反馈信号幅度

级别	I_W (A)	V_{IW} (V)	V_{IW_ADC} (V)	N_{IW}
L_{PK+}	6.8	4.625	2.313	0xECD9
L_{NOM+}	4	3.75	1.875	0xC000
L_0	0	2.5	1.25	0x8000
L_{NOM-}	−4	+1.25	+0.625	0x4000
L_{PK-}	−6.8	+0.375	+0.188	0x1340

本示例采用连接LEM®,的CAS 6-NP霍尔效应传感器，其初级匝数为3，具有0 V至5 V输出，后接增益为0.5的信号调理电路。

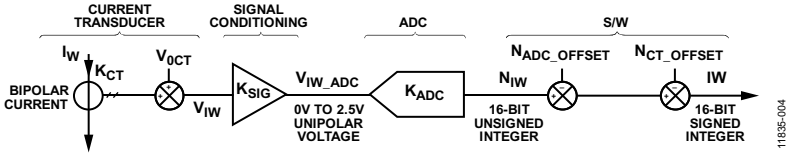

图5. 电流反馈路径的调整关系

ADC时序考虑因素

以PWM周期同步采样事件对于精确电流反馈而言非常重要。ADCC采用PWM周期工作的概念时序如图6所示。下列事件序列由PWM SYNC脉冲触发。

1. PWM sync脉冲触发定时器，以便启动。
2. ADCC不断将事件信息中的采样时间与定时器时间比较。
3. 完成定时器匹配，ADCC计划ADC操作。
4. ADC变为可用后，ADCC便使用事件信息选择适当的通道。
5. ADCC触发ADC转换序列，ADC采样并转换数据。
6. 数据回流至ADCC。
7. ADCC将数据通过DMA传输至存储器位置(LSB优先)。
8. 产生中断(IRQ)，提醒CPU数据样本可用。

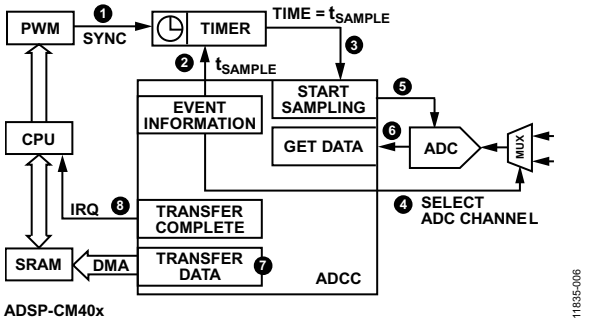

图6. ADCC操作序列

ADCC事件时序

控制器管理最多24个采样事件的配置和时序。这些事件的时序受触发限制，该触发会启动两个定时器(TMR0或TMR1)中的其中一个，并在定时器启动后启动一个事件时间。

如图7所示，触发源可从一系列外设或处理器事件中选取，如PWM SYNC脉冲、定时器或I/O引脚中断。每次事件都与一个事件号码(以Event x表示)、一个事件时间(以TIMEx表示)、控制信息(以CTLx表示)及其结果数据相关联。事件控制信息(图7中以CTLx表示)包含每个采样事件的信息，如ADC接口和通道号、所用的ADC定时器、同步采样选择，以及相关事件ADC数据的存储器失调。ADCC使用该信息对正确的ADC通道(CHx)进行多路复用处理、初始化ADC转换(CVST0/CVST1信号)并将正确的数据传输至适当的事件数据寄存器中。

然后，可设置DMA传输，将每次事件的ADC数据移入SRAM中。完成所有事件和后续DMA传输之后，便产生一个中断，通知主应用代码新ADC数据可用。

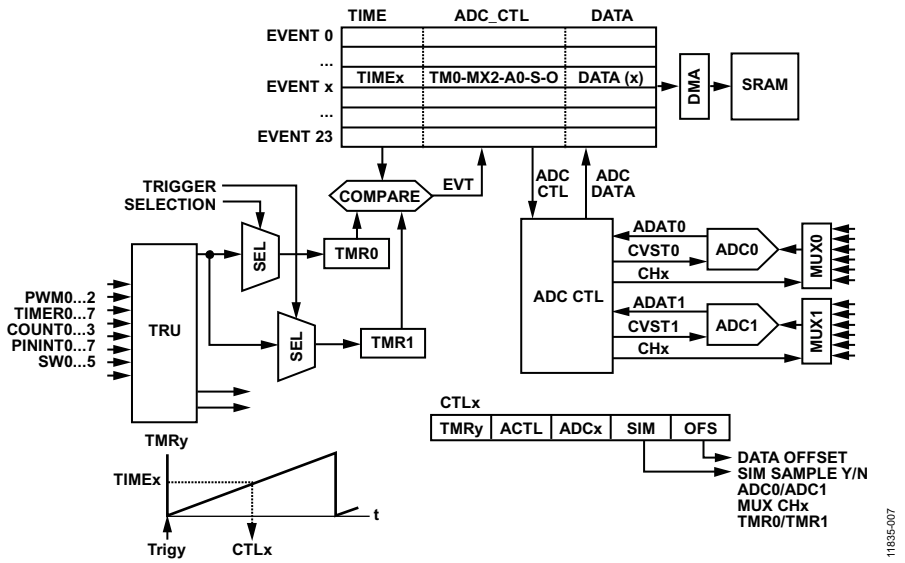

图7. ADCC模块功能框图

例如，图8显示与ADC定时器0有关的3个采样事件。PWM sync脉冲触发定时器，事件时间与每一次事件相关联。事件0和事件1是同步采样事件，事件时间寄存器中的事件时间置零。事件2稍后发生，并同样由事件2时间寄存器确定，表示为ADC时钟周期(t_{ACLK})的倍数。若事件2是与定时器0关联的最终事件，则定时器将在事件处理完成后停止运行，以降低功耗。

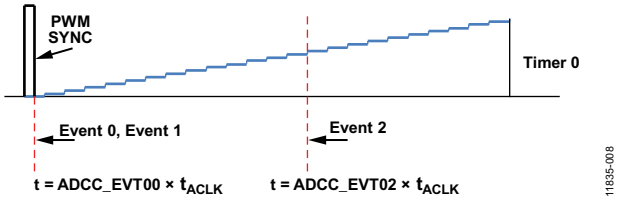

图8. 事件时序

ADC操作时序

ADCC控制器触发采样事件后，ADC操作本身具有一个转换时间延迟。图9显示单次ADC事件与每个ADC接口相关联，且使能两次事件同步采样的情况。

有三个独立的转换周期与ADC操作有关：

1. 写入8位控制字，选择ADC读取通道(ADCC_EVTCTL.CTLWD)。
2. 置位转换脉冲，使能ADC采样和转换。
3. 让16位ADC数据回流至ADCC。

ADCC提供这3个事件相位的片选和选通时钟信号。ADCC与ADC的接口为串行接口，采用双通道位操作。因此，每个CS脉冲期间提供的最小时钟周期数(ADCC时序控制寄存器的NCK段)为8。其他重要的设置有：ADC时钟频率、转换周期片选信号之间的最小延迟(t_{CSCS})(ACLK周期内)，以及CS边沿和ACLK边沿之间的最小延迟(t_{CSCK}和t_{CKCS})。因此，单个同步采样信号对的ADC转换周期时间t_{CONV_ADC}可表示为：

$$t_{CONV\_ADC} = \frac{3}{f_{ACLK}}\left(t_{CSCK} + NCK + t_{CKCS} + t_{CSCS}\right)$$

其中，f_{ACLK}表示ADCC时钟频率。

ADCC时钟由处理器系统时钟(f_{SYSCLK})通过ACKDIV分频(在时序控制寄存器ADCC_TCA中)在内部产生，计算如下：

$$f_{ACLK} = \frac{f_{SYSCLK}}{ACKDIV + 1}$$

其结果是系统时钟来源于处理器内核时钟($f_{CORECLK}$)。当$f_{CORECLK}$为f_{SYSCLK}的整数倍时，获得最佳系统性能。完成ADC转换后，额外延迟是因为ADC数据通过DMA传输至数据存储器，并最终由中断请求服务将数据帧准备就绪，供主应用程序使用。因此，在应用中，从触发(例如，PWM SYNC脉冲)到数据可用的总时间为：

$$t_{CONV\_TOTAL} = t_{CONV\_ADC} + t_{DMA} + t_{IRQ}$$

其中：

t_{DMA}是DMA传输的平均时间。

t_{IRQ}是中断请求服务的平均时间。

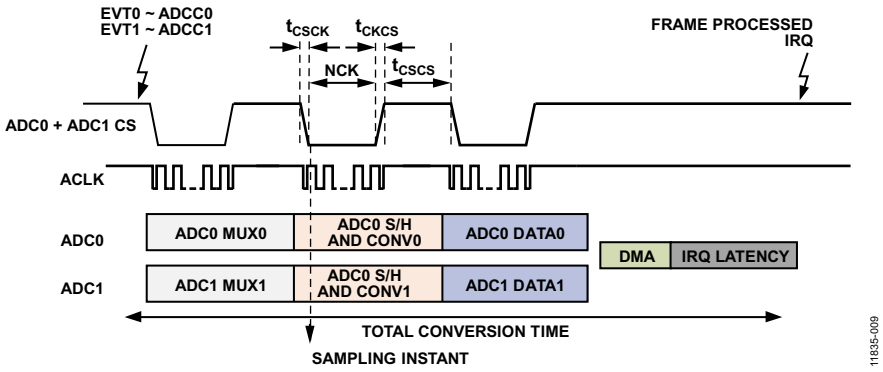

图9. 单次事件同步采样的转换时间

典型时序设置见表2。表中还列出了对时序的一些约束条件。获得正确ADC性能的绝对约束条件是，允许的ADC采样和转换周期(t_{CONV_ADC}/3)必须至少为380 ns。单个同步采样事件的时序结果如图10所示，该结果与电机绕组电流的采样有关(注意，该图为了突出示例而略为夸大)。

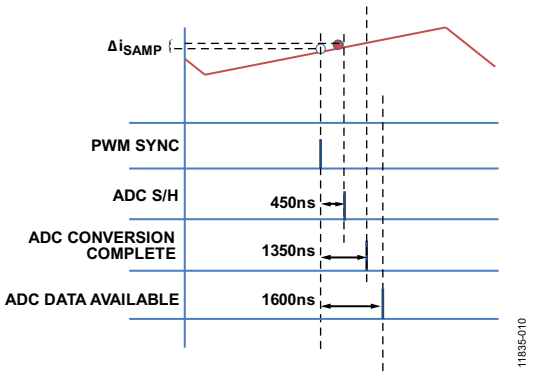

图10. 采样延迟时间

采用这些设置时，在所需的电流波形采样点与实际采样点之间存在450 ns失调。该值等于一个片选脉宽(200 ns + 25 ns + 0 ns)加两次片选之间的脉宽(225 ns)。该结果造成平均电机绕组电流和实际采样电流之间的Δi_{SAMP}产生差异，在时序调度中需加以考虑，虽然在1 kHz的典型电流控制环路带宽环境中，这表示不超过0.2°的相移。此外，对于10 kHz的典型PWM频率，ADC数据从产生PWM SYNC脉冲(表2中的设置)起，在不足2%的可用PWM周期时间内可供应用程序使用。如果在事件发生时ADC处于空闲状态，则4至5个SYSCLK周期的额外延迟将存在于事件激活的时刻与ADC开始工作的时刻之间。

采样时刻调整

可能需要进一步提高电机电流采样时刻的精度并消除所需采样时刻和实际采样时刻之间的450 ns失调。精度提高后对低电感伺服电机等应用案例或者采用较高开关频率的情况特别有益。要消除这一较小的时间偏移量，一种方法是使用通用(GP)定时器在PWM sync脉冲前一个ADCC片选脉宽处创建触发。这可以通过从前一PWM sync脉冲触发GP定时器来实现，如图11所示。

使用此方法时，在PWM周期结束前安排任何采样事件时必须谨慎。所有采样事件必须在下一周期开始前一个片选脉宽处完成(图11中的EVT0标记)。

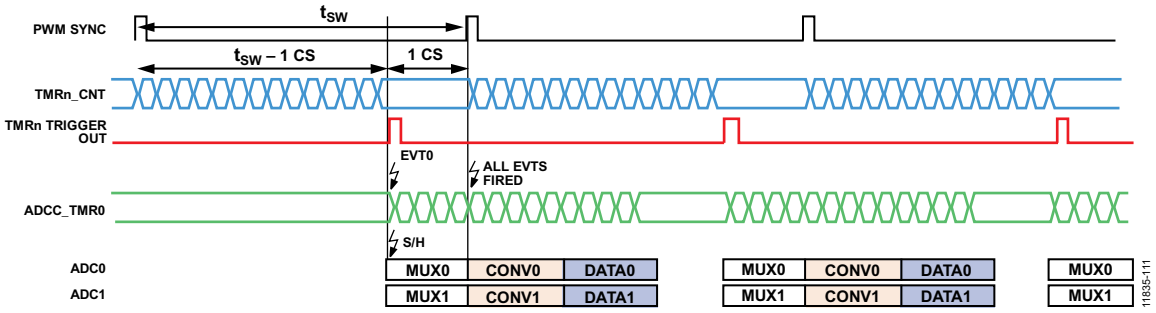

图11. 采样时刻调整的实现

图2. 典型ADC设置的时序设定

参数	数值	备注	设置
$f_{CORECLK}$	240 MHz	允许的最大值	PLL配置
f_{SYSCLK}	80 MHz	最大值为100 MHz	$f_{SYSCLK} = f_{CORECLK}/3$
f_{ACLK}	40 MHz	最大额定值为50 MHz	ADCC_TCA0.CKDIV = 1
CS时间(t_{CSCS})	200 ns	必须允许足够的ACLK周期才能传输CTL字和数据	ADCC_TCA0.NCK = 8
CS边沿至ACLK边沿(t_{CSCK})	25 ns	推荐40 MHz时的最小时间	ADCC_TCB0.TCSCK = 1
ACLK边沿至CS边沿(t_{CKCS})	0 ns	推荐	ADCC_TCB0.TCKCS = 0
CS之间的时间间隔(t_{CSCS})	225 ns	必须高于150 ns以便精确采样	ADCC_TCB0.TCSCS = 9
t_{CONV_ADC}	450 ns		
t_{DMA}	50 ns	平均需要4个SYSCLK周期	
t_{IRQ}	200 ns	平均需要16个SYSCLK周期	

ADC流水线

当新事件开始与ADC正在处理的现有事件重叠时，ADCC将新事件作为待处理事件存储在8深先进先出(FIFO)缓冲器中，该缓冲器可用于所有ADC接口。写入激活事件的控制字后，ADCC立即开始写入首个待处理事件的控制字，同时进入激活事件的采样阶段。同样，第一个待处理事件的控制字阶段执行完成后，便进入第二个待处理事件的控制字阶段。以这种方式，ADCC可在每个ADC接口上通过流水线方式交错处理三个并行事件。因此，事件能以紧凑高效的方式排列。

配置事件时序使事件以上述流水线方式处理，可获得最高的ADC吞吐速率。该流水线如图13所示。图中，三对同步采样事件的触发时间间隔很短。

图中，三对同步采样事件的触发时间间隔很短。ADCC开始处理事件0和事件1，同时将事件2至事件5存储在FIFO中。然后，ADC资源变得可用时，便会对这些事件进行处理。

从图7可知，在CS某次置位期间，ADCC会处理全部6个事件以及每个事件的多个阶段，并且两次连续采样之间的时间间隔仅为18个ACLK周期。该时间间隔相当于表2设置中的450 ns，且可通过提高ACLK频率进一步降低。若要在电机控制应用内最大化ADC带宽，最好的方法是有意识地将所有与PWM周期相关的采样事件以流水线方式处理。该方法可以确保新的ADC样本能在PWM周期中可能的最早时刻准备就绪。要实施图13中显示的流水线，则需要所有事件时间均接近零，即紧跟在PWM SYNC脉冲后。

建议在两个事件时间寄存器ADCC_EVTnn(nn表示0到24的寄存器编号)中存储的事件时间之间允许一个最小1 ACLK周期，实现正确调度。图12中显示使用流水线操作时的总转换时间，包括起始延迟、DMA传输和中断服务，可用于采用表2中所示时序设置的各种同步采样对数目。

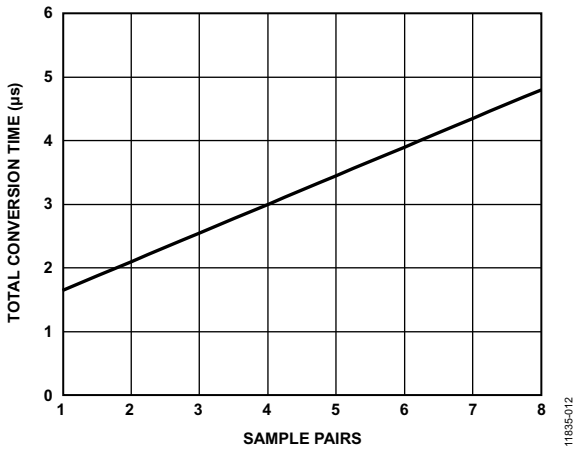

图12. 不同采样对数目的总转换时间

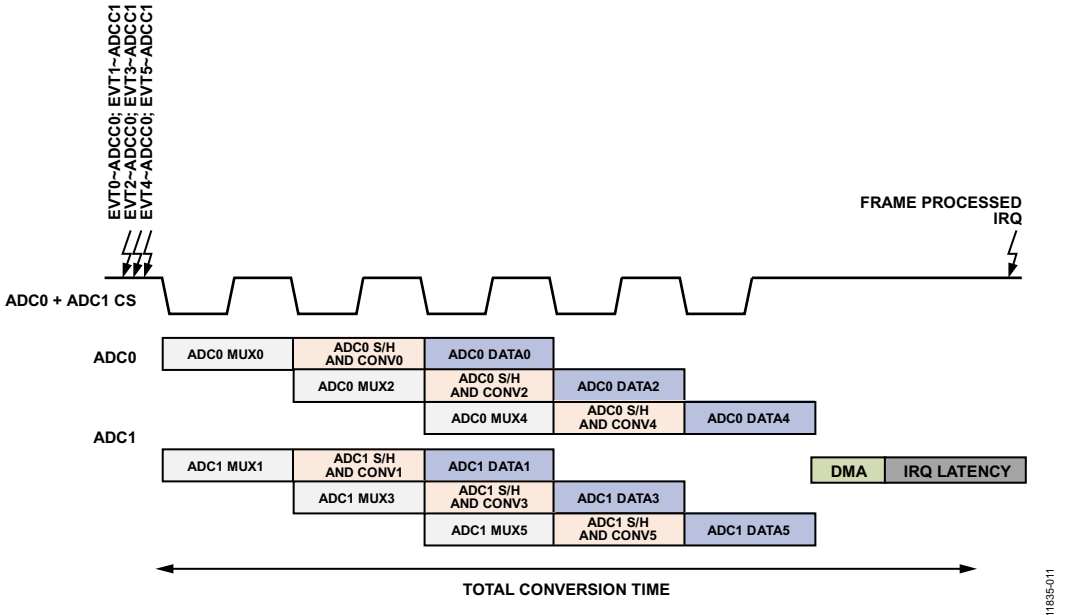

图13. ADC中的流水线事件

ADC数据访问

目前为止，本文中的示例都假定ADC数据采用自动DMA传输从存储器中访问。从ADCC存储器映射寄存器(MMR)的内核读数中直接访问数据是可行的，如图14所示。注意，图14中的ACK表示确认信号，而非模拟时钟。

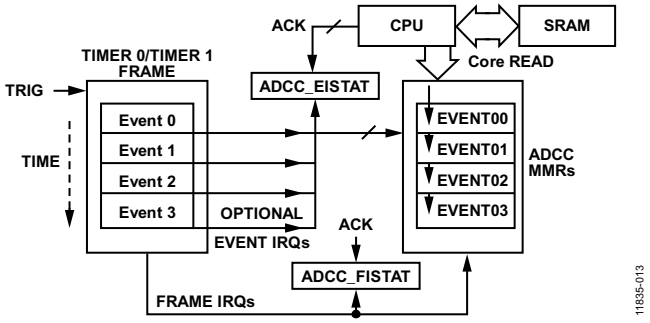

图14. 内核模式的ADC数据访问

在内核模式下，CPU通过事件或帧中断获取有关新数据就绪的信号，这些中断可按需独立屏蔽或解除屏蔽。该模式下还具有额外的灵活性：在整个事件帧结束前，一旦完成独立事件，即可对其进行读取。内核模式的缺点是中断服务以及MMR读访问所需的总延迟比DMA模式下要高。采用优化的内核和时钟分频比设定，则在每个中断服务的最高延迟情况下，每次MMR读操作将花费10至12个SYSCLK周期。

图15显示了DMA模式下的数据访问。在这种情况下，DMA传输仅在完成一个定时器帧后才会发生，且帧中断信号仅在完成DMA传输后才发给CPU。

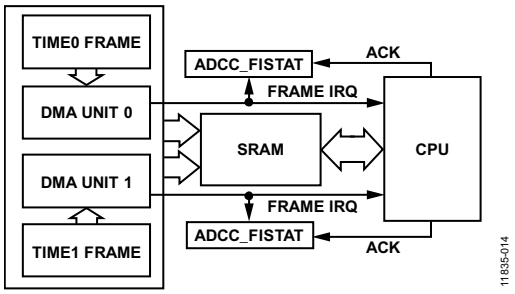

图15. DMA模式下的ADC数据访问

两种情况下，EISTAT和FISTAT寄存器都在激活后提供事件和帧中断的状态指示，且必须在下一次触发前经由CPU清零相关位确认，否则将标记一次触发溢出条件。

ADCC数据故障检测

ADCC提供多个故障状态寄存器位，发生数据故障时可置位；ADCC事件时序和/或不确定事件序列设置不当时，就有可能发生数据故障。这些故障可能使ADCC过载，或造成无效的ADC数据。它们包括：

- 触发溢出。当前帧结束前便触发下一次事件。
- DMA带宽。帧结束所需的时间长于用户定义时间。
- 存储器错误。ADC数据写操作失败。
- 事件冲突。处理现有事件时发生新事件。
- 事件错失。事件未处理。

这些错误均可按需配置为内核的中断源，并且它们都会置位ADCC_ERRSTAT寄存器。在电机控制应用中，尤其在电流反馈测量中，与事件错失、存储器和触发溢出有关的错误对于监控核心应用非常重要，因为错误或错失的电流环路数据可能会导致控制环路的不稳定。事件冲突在流水线操作中非常普遍，通常影响不大，除非FIFO已满。

ADCC模块、触发路由和存储器设置

使用ADC之前，需执行一系列步骤来设置ADCC模块以及触发路由单元和数据缓冲器。完成配置后，假定采用DMA数据访问模式，则DMA引擎会自动将主ADC数据传输至存储器，供主应用程序进行内部访问。当数据就绪时，ADCC产生中断，以便处理器执行控制算法并更新PWM调制器寄存器。

图17显示ADCC、CPU、SRAM、PWM和外部信号之间所需的互连，可用于在典型电机控制应用中捕捉电机电流反馈以及其他模拟监控信号。在本例中，编码器正弦和余弦信号、散热片温度和直流总线电压作为额外监控输入示例提供。

设置ADCC以便正确处理信号反馈的三个步骤如下所示：

1. ADCC事件配置。
2. 中断和触发路由。
3. 数据访问和存储器分配。

下列子段落描述正确设置系统所需的步骤与相应的寄存器配置。

ADCC事件配置

ADCC事件的配置示例如图17所示，该示例包括为每个事件分配定时器、ADC接口和通道、时间偏移以及同步采样开关。这可以通过多种方法实现；图16以及表3所示为其中的一种可能情况。该示例采用全部两个定时器，仅供参考。

对于此示例，事件可连接至一个定时器，因为所有事件的定时都与PWM SYNC脉冲有关。在诸如双轴电机控制算法这类用例中，可能必须同时使用两个定时器，因为该用例采用两组PWM输出以及相应的PWM sync脉冲。

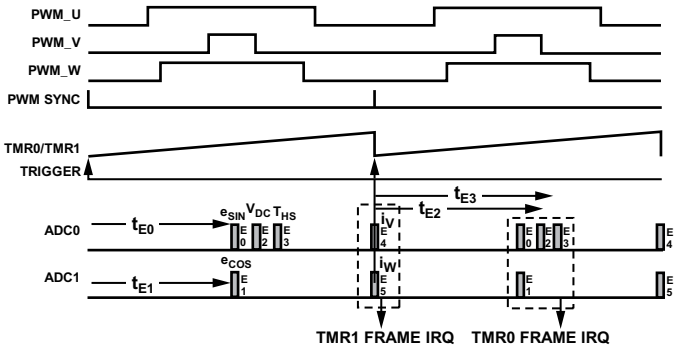

图16. 电机控制应用中的典型ADCC

表3. 示例应用的事件配置

事件	定时器	ADC I/F	ADC 通道	时间	同步采样
E0 (e_{SIN})	TMR0	0	0	t_{E0}	是
E1 (e_{COS})	TMR0	1	0	$t_{E1} = t_{E0}$	是
E2 (V_{DC})	TMR0	0	2	t_{E2}	否
E3 (T_{HS})	TMR0	0	3	t_{E3}	否
E4 (i_V)	TMR1	0	1	0	是
E5 (i_W)	TMR1	1	1	0	是

相位电流i_V和i_W在发生PWM sync脉冲触发后立即同步采样，它们关联至TMR1。定时器1帧立即通过DMA传输至存储器，且新的电流样本可供主应用程序使用。在PWM周期中的较晚时刻，新的事件帧采样并关联至TMR0。编码器正弦和余弦信号同步采样，紧随其后的是直流总线电压和散热片温度信号。对三个ADC0信号进行流水线处理，以最大化吞吐速率。然后，将TMR0帧通过DMA传输至存储器。

这些参数的配置需设置ADCC_EVCTLnn事件控制寄存器和ADCC_EVTnn事件时间寄存器，nn表示每个事件的编号。提供本节内容涉及的驱动程序API，简化该过程。

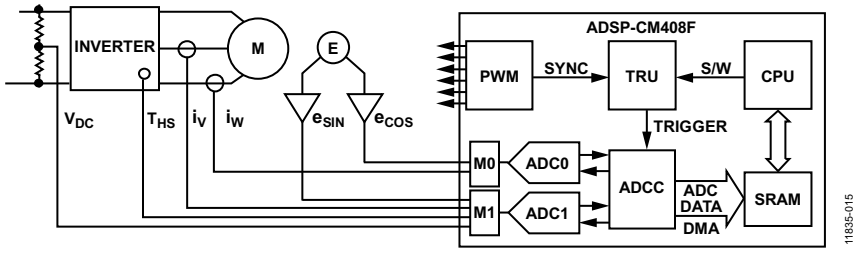

图17. 典型电机控制应用中的系统互连

中断和触发路由

在图17中的示例中，所有事件时间均参考PWM周期；因此，两个定时器都由PWM SYNC脉冲触发。PWM sync脉冲作为硬件触发与ADCC定时器相关联的前提是配置TRU，使PWM sync脉冲作为主触发信号与ADCC触发从机相关联。随后，ADCC定时器必须与ADCC触发相关联。

图18显示相应触发路由的概念图；该路由涉及触发主机19 (PWM0 SYNC)与触发从机24 (ADCC_TRIG0)之间的连接，本例中该连接可通过将主机编号写入适当的从机选择寄存器TRU_SSR24来实现。之后，通过在ADC_CTL寄存器中为TRIGSEL位设置相应的值，将ADCC_TRIG0触发路由至两个定时器。

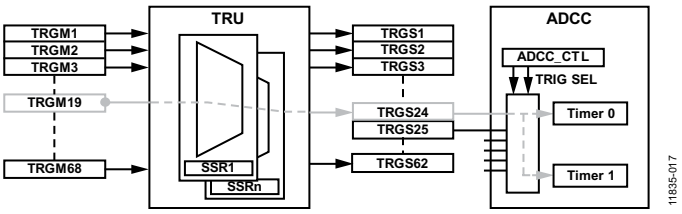

图18. 从PWM SYNC至ADCC定时器的触发路由

此触发路由配置可以提供硬件的直连链路，将PWM时序与ADC采样相关联，而不会在路径上产生软件延迟。触发主机也可从其他源路由，如GPIO引脚中断、定时器和计数器事件。如此配置便可提供精确同步，比如与ADSP-CM408F控制的其他转换器进行采样同步。

此外，完整的ADCC定时器帧能作为触发主机与其他外设或核心从机相关联。

由于本例中采用了DMA传输模式，因此所有事件中断都应当在ADCC_EIMSK寄存器中屏蔽。同样，提供驱动程序API，用来在寄存器中设置适当的中断服务例程，实现DMA模式下的帧中断。

利用触发路由提供增强型精确采样时序

像上文所述那样，要从当前采样时序中移除片选脉宽滞后，需要稍微不同的触发路由配置。在此情况中，ADCC定时器是通过GP定时器触发器触发的，而该触发器本身是通过PWM sync触发的。此序列可参见图11。

数据访问和存储器分配

如图14和图15所示，可通过读取内核MMR访问ADC数据，或通过DMA传输使其能在SRAM中访问。在内核模式下，无需配置特定存储器分配，即可实现除变量外的内核MMR读取数据的写操作。然而在DMA模式下，必须分配特定的存储器区域，然后进行配置，才能实现DMA访问，并且每个定时器都必须配置。所需的存储器大小取决于每个定时器相关的帧尺寸，以及新的帧覆盖之前需要在存储器中存储多少帧。

表19显示SRAM映射概念，以及控制SRAM配置的相关ADCC寄存器。ADCC_BPTR寄存器必须存储指向存储器基址的指针，才能存储ADC样本。若存储器缓冲器中需要存储多个帧，则ADCC_FRINC寄存器中应包含指针的偏移值，使其指向下一帧的基址。在线性缓冲模式下(通过向ADCC_CBSIZ寄存器写入零来激活)，会以持续增加的线性化方式在存储器中存储额外的帧，中间隔开一个帧的增量值。若向ADCC_CBSIZ写入非零值M，则会激活循环缓冲，且在帧的基址指针返回ADCC_BPTR值并开始覆盖现有帧数据之前向存储器写入M帧。

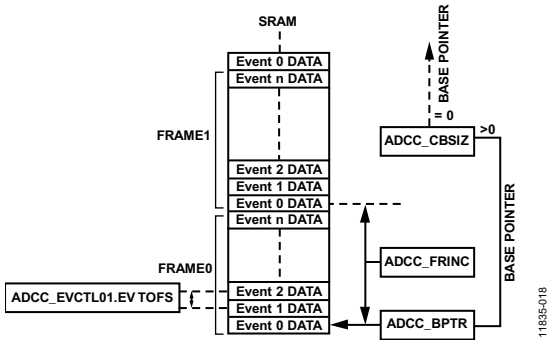

图19. ADC DMA传输的存储器配置

在图17中的电机控制应用示例中，每个PWM周期都会采集ADC样本，并即刻用于控制和监控应用中。因此，将样本以线性方式存储毫无意义，因为存储器将很快过载。在此类应用中，最好在启用循环缓冲时将M限制为1或更小的值，或者将ADCC_FRINC值设为0并在每个PWM周期中覆盖帧。简化这项任务的驱动程序应用编程接口(API)可在“ADCC软件支持”部分找到。

ADCC软件支持

与ADSP-CM40x EZKIT一同提供的ADI Enablement Software软件包内含一系列API函数调用命令，可简化本应用笔记中讨论的ADCC模块设置。这些调用命令监控不同寄存器的正确配置以及需要执行的任何状态确认。

示例代码

本应用笔记中的示例代码逐步说明配置并使用图17中电机控制应用的方法。器件驱动程序会增加额外开销，但极大地简化ADCC模块寄存器编程。

代码的第一部分定义一系列参数和配置常数，用于驱动程序API调用。

第1行至第10行定义每个定时器的帧和关联的数据缓冲器大小。分配样本缓冲器长度时，将包含系数2作为安全措施，用于调试目的。由于ADC样本传输至存储器的操作完全通过硬件触发(包括DMA)，因此在将新的缓冲器提交给驱动程序以及复位ADCC缓冲器指针之前，如果在第122行插入软件断点，则可能导致存储器被覆写。以一个额外缓冲作为裕量可防止出现此类调试相关问题。缓冲器中的帧数目定义为1，这表示每次收到新帧，API便会覆写存储器缓冲器，即每个定时器仅需针对1个帧进行存储器分配。

第11行到第16行以ACLK周期数形式定义每个事件的采样时间，如表3所示。注意，SMP_TIME1、SMP_TIME2和SMP_TIME3仅以一个ACLK周期分隔。该设置使这些事件在ADC0内以流水线方式进行处理。

第17行至第44行定义每个ADC通道的控制字、6个采样事件的通道映射，以及数据缓冲器中每个事件的数组索引。

第45行至第59行声明ADC操作所需的变量和函数原型。ADCC存储器缓冲器和ADCC定时器存储器缓冲器的存储器分配大小由API预定义，且不得进行更改。一个ADCC设置函数、一个TRU设置函数以及两个ISR回调函数(每个ADCC定时器各一个回调函数)对寄存器进行设置。

第60行至91行包含主ADCC配置函数SetupADC()。第一步是设置事件配置表，即含有事件编号、ADC控制字、ADC定时器、同步采样以及每个事件的存储器偏置的struct。

正确完成ADCC事件配置后，必须开启ADCC实例，以及与该实例相关的所有ADCC定时器。然后，必须在寄存器中设置每个定时器帧中断的驱动程序回调函数名(第72行至73行)。随后，使能DMA模式(第74行)，配置ADCC时钟和芯片选择(第75行至第78行)。

定时器以ADCC_TRIG0输入配置为触发源。在SetupTRU()函数中，ADCC_TRIG0触发分别作为触发从机与PWM同步脉冲触发主机相关联(第92行至第97行)，如图18所示。这些函数调用中的数据枚举列于ADI Enablement Software软件包的驱动程序文档中。

第81行将第62行定义的EventCFG指令传递至adi_adcc_ConfigEvent驱动程序函数，随后adi_adcc_SetEventMask驱动程序函数按需使能或屏蔽事件。本例中，使能全部事件。为了获得最高的ADC吞吐速率，使能双位数据接口很重要，代码见第83行，这表示能在8个ADC时钟周期内传输ADC的16位数据。(注意，如果没有使能双位接口，则第76行中的N_{CK}以及第77行和第78行中的t_{CSCS}必须分别设为16和17)。然后进行数据缓冲器的存储器分配，数据缓冲器将提交至ADCC，以便通过adi_adcc_SubmitBuffer调用进行填充。adi_adcc_SubmitBuffer API仅在DMA模式下工作；因此使用该API之前必须先设置DMA模式。该驱动程序函数由主应用程序再次调用(第105行)，以便应用程序完成数据提取后将缓冲器归还给ADCC控制。最终，完成全部配置后，需要使能定时器实例以及ADCC自身。

第92行至第97行包含TRU的设置步骤。它包括开启TRU实例、将触发从PWM sync主机路由至ADCC从机，以及使能TRU。

如前文所述，在应用层处理ADC数据由ADCC定时器回调函数实现，该回调函数后跟一个中断，此中断在完成定时器事件以及相关的DMA传输后发出。

第98至第127行实现回调函数。缓冲数据在缓冲器的相应位置处提取，并保存至适当的全局变量中。本例中，更新后的相位电流数据立即用于电机控制算法中，通过第117行中的算法调用函数MotorControl()从定时器1回调函数中调用。

注意，ADCC事件定时器中断服务是访问ADCC数据时发生的唯一软件调用例程。同步与时序均在硬件层面中实现。

第128行至第136行包含的附加代码片段可插入TRU和ADCC设置函数，以使能图11所示的增强型精确采样时序功能。第128行至第129行中设置了从PWM SYNC至GP定时器TMR7再至ADCC定时器0触发的硬件触发路由路径。第130行至第136行包含的示例代码可插入ADC设置函数，以正确配置和使能GP定时器TMR7，从而提供正确的延迟。

无论何时，都必须在调用SetupADC函数前调用SetupTRU函数。

```
/***********************************************
          ADCC Module Setup Code Example
***********************************************/

/*******************Defines*********************/
1.  #define ADCC_DEVICE_NUM         0
2.  #define TRU_DEV_NUM             0
3.  #define ADI_TRU_REQ_MEMORY
4.  #define NUM_SAMPLES0       4
5.  #define NUM_SAMPLES1       2           /*
    Length of ADC buffers */
6.  #define FRAME_INC0
    2*NUM_SAMPLES0*sizeof(short)
7.  #define FRAME_INC1
    2*NUM_SAMPLES1*sizeof(short)  /* Frame
    increment in number of bytes for each buffer*/
8.  #define FRAMES_IN_BUFFER  1   /*Number of
    frames in buffer */
9.  #define NO_OF_EVENTS     6         /* Total
    number of events */
10. #define EVENT_MASK              0xFFFF

/*Event Times in ACLK Cycles*/
11. #define SMP_TIME0            950
12. #define SMP_TIME1            950
13. #define SMP_TIME2            951
14. #define SMP_TIME3            952
15. #define SMP_TIME4            0
16. #define SMP_TIME5            0

/* Control Words for All ADC Channels */
/*Upper Nibble = Chan No. Lower Nibble = 0xF for
Sim Sampling, 0xD Otherwise*/
17. #define ADC0_VIN00_CTL          0x0F
18. #define ADC0_VIN01_CTL          0x1F
19. #define ADC0_VIN02_CTL          0x2D
20. #define ADC0_VIN03_CTL          0x3D
21. #define ADC0_VIN04_CTL          0x4D
22. #define ADC0_VIN05_CTL          0x5D
23. #define ADC0_VIN06_CTL          0x6D
24. #define ADC0_VIN07_CTL          0x7D

25. #define ADC1_VIN00_CTL          0x0F
26. #define ADC1_VIN01_CTL          0x1F
27. #define ADC1_VIN02_CTL          0x2D
28. #define ADC1_VIN03_CTL          0x3D
29. #define ADC1_VIN04_CTL          0x4D
30. #define ADC1_VIN05_CTL          0x5D
31. #define ADC1_VIN06_CTL          0x6D
32. #define ADC1_VIN07_CTL          0x7D

/*Mapping the Signals to the Appropriate ADC
Channels*/
33. #define ES_CTL                  ADC0_VIN00_CTL
34. #define EC_CTL                  ADC1_VIN00_CTL
35. #define VDC_CTL                 ADC0_VIN02_CTL
36. #define THS_CTL                 ADC0_VIN03_CTL
37. #define IV_CTL                  ADC0_VIN01_CTL
38. #define IW_CTL                  ADC1_VIN01_CTL

/*Locations of ADC Signals in Data Buffer Index*/
39. #define IV_ADC                   0
40. #define IW_ADC                   1
41. #define ES_ADC                   0
42. #define EC_ADC                   1
43. #define VDC_ADC                  2
44. #define THS_ADC                  3

/*******************Variables*******************/
45. static ADI_ADCC_HANDLE hADCC;             /*
    ADCC Handle */
46. static ADI_ADCC_HANDLE hADCCTimer0,
    hADCCTimer1;   /*ADCC Timer Handles*/
47. static uint8_t ADCCMemory[ADI_ADCC_MEMORY];
    /* Memory buffer for the ADCC device -
    predefined */
48. static uint8_t
    ADCCTmr0Memory[ADI_ADCC_TMR_MEMORY];
49. static uint8_t
    ADCCTmr1Memory[ADI_ADCC_TMR_MEMORY];    /*
    Memory buffer for the ADCC Timers -
    predefined*/
50. static uint16_t SampleBuffer0[NUM_SAMPLES0];
51. static uint16_t SampleBuffer1[NUM_SAMPLES1];
    /* Memory buffer for the ADC samples */
52. static uint16_t Iv_adc, Iw_adc;
53. static uint16_t Es_adc, Ec_adc, Vdc_adc,
    Ths_adc;
    /*Variables for ADC data*/
54. static uint8_t
    TruDevMemory[ADI_TRU_REQ_MEMORY];
55. static ADI_TRU_HANDLE hTru;
   /*TRU Device Memory and Handle*/

/*************Function Prototypes***************/
56. void SetupADC(void);
57. void SetupTRU(void);
58. static void AdccTmr0Callback(void *pCBParam,
    uint32_t Event, void *pArg);
59. static void AdccTmr1Callback(void *pCBParam,
    uint32_t Event, void *pArg);

/******Function to Configure ADCC***************/
60. void SetupADC(void) {
61. static ADI_ADCC_RESULT result;

/*Set Up Event Configuration Table*/
62. ADI_ADCC_EVENT_CFG  EventCFG[NO_OF_EVENTS] = {
63. {0,  ES_CTL, ADI_ADCC_ADCIF0, ADI_ADCC_TIMER0,
    true,  0, SMP_TIME0},
64. {1,  EC_CTL, ADI_ADCC_ADCIF1, ADI_ADCC_TIMER0,
    true,  2, SMP_TIME1},
65. {2,  VDC_CTL, ADI_ADCC_ADCIF0,
    ADI_ADCC_TIMER0, false,  4, SMP_TIME2 },
66. {3,  THS_CTL, ADI_ADCC_ADCIF0,
    ADI_ADCC_TIMER0, false,  6, SMP_TIME3 },
67. {4,  IV_CTL, ADI_ADCC_ADCIF0, ADI_ADCC_TIMER1,
    true,  8, SMP_TIME4 },
68. {5,  IW_CTL, ADI_ADCC_ADCIF1, ADI_ADCC_TIMER1,
    true, 10, SMP_TIME5 }}; /*Event#, CTL_WORD,
    ADC Interface, Timer ID, sim. samp, Mem offset
    in frame, Event time */
```

```
/*ADCC Setup API Functions*/
69. result = adi_adcc_OpenDevice(ADCC_DEVICE_NUM,
    ADCCMemory, &hADCC);
70. result = adi_adcc_OpenTimer(hADCC,
    ADI_ADCC_TIMER0, ADCCTmr0Memory,
    &hADCCTimer0);
71. result = adi_adcc_OpenTimer(hADCC,
    ADI_ADCC_TIMER1, ADCCTmr1Memory,
    &hADCCTimer1); /* ADCC Device handle, Timer to
    open, Timer memory, Pointer to the timer
    handle */
72. result = adi_adcc_RegisterTmrCallback
    (hADCCTimer0, AdccTmr0Callback, hADCCTimer0);
73. result = adi_adcc_RegisterTmrCallback
    (hADCCTimer1, AdccTmr1Callback,
    hADCCTimer1);/*Register callback functions*/
74. result = adi_adcc_EnableDMAMode(hADCC,true);

75. result = adi_adcc_ConfigADCCClock(hADCC,
    ADI_ADCC_ADCIF0, false,1u, 8u );
76. result = adi_adcc_ConfigADCCClock(hADCC,
    ADI_ADCC_ADCIF1, false,1u, 8u ); /*For each
    ADC interface: ADCC handle, ADC Interface
    number, falling edge, ACLK Clock divide, NCK*/
77. result = adi_adcc_ConfigChipSelect(hADCC,
    ADI_ADCC_ADCIF0, false, 1u, 0u, 9);
78. result = adi_adcc_ConfigChipSelect(hADCC,
    ADI_ADCC_ADCIF1, false, 1u, 0u, 9);/*For each
    interface: ADCC handle, ADC interface, active
    low, TCSCK, TCKCS, TCSCS*/
79. result = adi_adcc_ConfigTimer(hADCCTimer0,
    ADI_ADCC_TRIG0, true, false);
80. result = adi_adcc_ConfigTimer(hADCCTimer1,
    ADI_ADCC_TRIG0, true, false); /*For each
    timer: Timer handle, Timer trigger source,
    falling edge trigger, No trigger output */
81. result = adi_adcc_ConfigEvent(hADCC,
    &EventCFG[0], NO_OF_EVENTS);  /*ADCC handle,
    Pointer to the event configuration table,
    Number of events in the table */
82. result = adi_adcc_SetEventMask(hADCC,
    EVENT_MASK);                              /*
    Handle to the device, Enable all events */
83. adi_adcc_EnableDualBitDataIF(hADCC, true);
    /*Dual bit interface allows highest
    throughput*/
84. memset((void *)SampleBuffer0, 0, NUM_SAMPLES0
    * sizeof(short));
85. memset((void *) SampleBuffer1, 0, NUM_SAMPLES1
    * sizeof(short));
86. result = adi_adcc_SubmitBuffer(hADCCTimer0,
    SampleBuffer0, FRAME_INC0, FRAMES_IN_BUFFER);
87. result = adi_adcc_SubmitBuffer(hADCCTimer1,
    SampleBuffer1, FRAME_INC1, FRAMES_IN_BUFFER);
/*For each timer: timer handle, Pointer to the
    buffer, Frame increment,  Number of frames
    that fits into the given buffer */

88. result = adi_adcc_EnableTimer(hADCCTimer0,
    true);
89. result = adi_adcc_EnableTimer(hADCCTimer1,
    true);
90. result = adi_adcc_EnableDevice(hADCC, true);
    /*Enable everything*/
91. }

/*******Function to Configure TRU***************/
92. void SetupTRU(void){
93. ADI_TRU_RESULT result;
94. result = adi_tru_Open (TRU_DEV_NUM,
    &TruDevMemory[0], ADI_TRU_REQ_MEMORY, &hTru);
    /* Setup TRU for ADCC. Slave is ADCC0 trig 1
    and master is PWM0 SYNC pulse*/
95. result = adi_tru_TriggerRoute (hTru,
    TRGS_ADCC0_TRIG0, TRGM_PWM0_SYNC);  /*TRU
    device, slave, master*/
96. result = adi_tru_Enable (hTru, true); /*Enable
    TRU*/
97. }

/***********ADCC Timer Callbacks****************/
98. static void AdccTmr0Callback(void *pCBParam,
    uint32_t Event, void *pArg){
99.   switch(Event){
100.         case ADI_ADCC_EVENT_FRAME_PROCESSED:
101.           Es_adc= SampleBuffer0[ES_ADC];
102.           Ec_adc = SampleBuffer0[EC_ADC];
103.           Vdc_adc = SampleBuffer0[VDC_ADC];
104.           Ths_adc = SampleBuffer0[THS_ADC];
    /*Store all of the data sampled in appropriate
    global variables*/
105.    _adcc_SubmitBuffer(hADCCTimer0,
    SampleBuffer0, FRAME_INC0, FRAMES_IN_BUFFER);
    /*Return the buffer to the ADCC for use in the
    next events*/
106.          break;
107.      case ADI_ADCC_EVENT_BUFFER_PROCESSED:
108.          break;
109.      default:
110.         break;
111.     }

112.   static void AdccTmr1Callback(void
    *pCBParam, uint32_t Event, void *pArg){

113.     switch(Event){
114.        case ADI_ADCC_EVENT_FRAME_PROCESSED:
115.          Iv_adc = SampleBuffer1[IV_ADC];
116.          Iw_adc = SampleBuffer1[IW_ADC];
117.          MotorControl();     /*Run the
    current control algorithm*/
118.
119.
120.          break;

121.      case ADI_ADCC_EVENT_BUFFER_PROCESSED:
122.   adi_adcc_SubmitBuffer(hADCCTimer1,
    SampleBuffer1, FRAME_INC1, FRAMES_IN_BUFFER);
123.          break;
124.      default:
125.         break;
126.     }
127.   return;
   }
```

```
/************************************************
          Enhanced Precision Timing Code
************************************************/
/*Setup TRU for ADCC enhanced timing precision.
Slave is ADCC0 trig 1 and master is GP timer 7
Added to SetpTRU() function in place of line 95 */

128.    result = adi_tru_TriggerRoute(hTru,
    TRGS_ADCC0_TRIG0, TRGM_TIMER0_TMR7);  // TRU
    device, slave, master
129.    result = adi_tru_TriggerRoute(hTru,
    TRGS_TIMER0_TMR7, TRGM_PWM0_SYNC);  // TRU
    device, slave, master

/*Setup GP timer 7 timer used to advance frame by
one CS. Add to SetupADC() function after line 91*/

130.    *pREG_TIMER0_STOP_CFG_SET =
    BITM_TIMER_STOP_CFG_TMR07;
131.    *pREG_TIMER0_RUN_CLR =
    BITM_TIMER_RUN_SET_TMR07; /*Disable Timer
    First*/
132.    *pREG_TIMER0_TMR7_CFG =
    ENUM_TIMER_TMR_CFG_PWMSING_MODE|ENUM_TIMER_TMR
    _CFG_IRQMODE1 |ENUM_TIMER_TMR_CFG_TRIGSTART |
    ENUM_TIMER_TMR_CFG_POS_EDGE|ENUM_TIMER_TMR_CFG
    _PADOUT_EN | ENUM_TIMER_TMR_CFG_EMU_CNT;
133.    *pREG_TIMER0_TMR7_DLY = (uint32_t)(fsysclk
    / F_SW - 0.00000045 * fsysclk); /* Delay must
    be Tsw minus one ADC chip-select. Chip select
    is 18 ACLKs*/
134.    *pREG_TIMER0_TMR7_WID = 16; /*Be careful
    here... DLY+WID must be smaller than one PWM
    period. In other words, WID must be smaller
    than one ADC chip select. If WID>CS, trigger
    pulse stretches into next PWM period. */
135.    *pREG_TIMER0_TRG_MSK &=
    ~(BITM_TIMER_TRG_MSK_TMR07);
136.    *pREG_TIMER0_TRG_IE |=
    BITM_TIMER_TRG_IE_TMR07; /*Enable TMR7*/
```

示例实验结果

“示例代码”部分提供的电流采样代码部分已在闭环永磁同步电机控制应用电路中进行了测试。应用电路采用通用交流线输入以及−6.8 A至+6.8 A的受控电机电流范围，并利用了电流传感器；该电流传感器参数图4中的电流调整数据。图20至图23还显示了应用电路的采样结果。

图20显示了参考速度为1500 rpm且电机空载时测得的电机相位电流。电机电流水平极低，并且高度不连续。

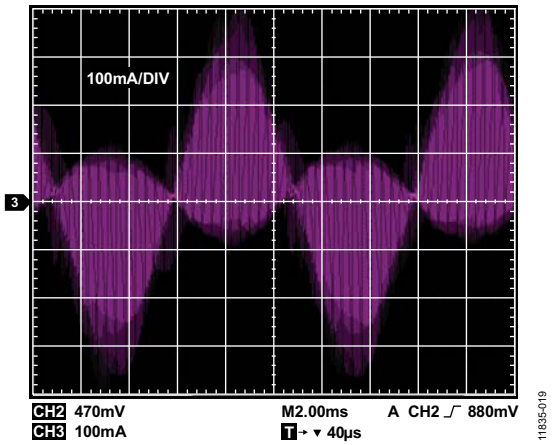

图20. 测量电机相位电流

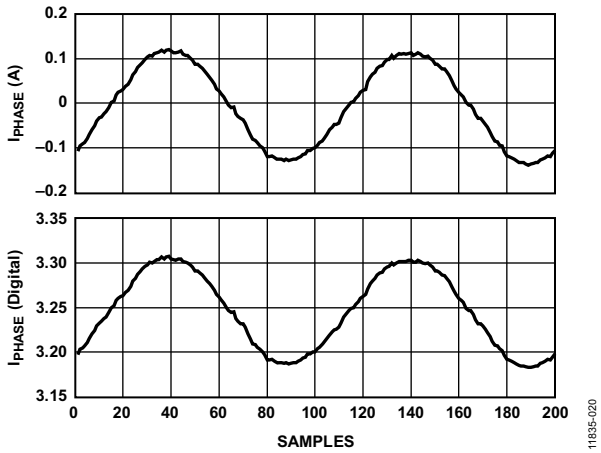

图21. ADC采样电机相位电流：上图为调整至真实值；下图为数字字输出

图21显示采用正确同步采样方法的平均效应，由图中可见电机相位电流具有平滑的正弦平均波形，即便电流水平低于最大值的2%时亦是如此。图21和图22(即跟踪I_Q参考电流的控制环路工作曲线)均通过ADSP-CM408F产生的数据流获得，该产品通过RS-232连接MATLAB®接口。

在图23中，PWM sync脉冲位置以及后续的采样触发显示在相位电流PWM周期的中央，该处电流等于瞬时平均值。为便于说明，该图显示的是较高的负载。

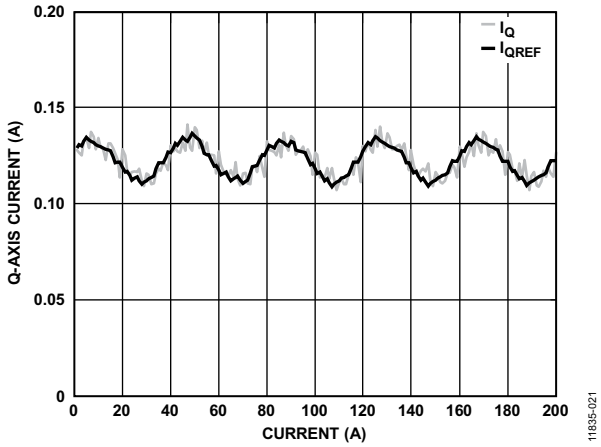

图22. Q轴参考电流和实际电流

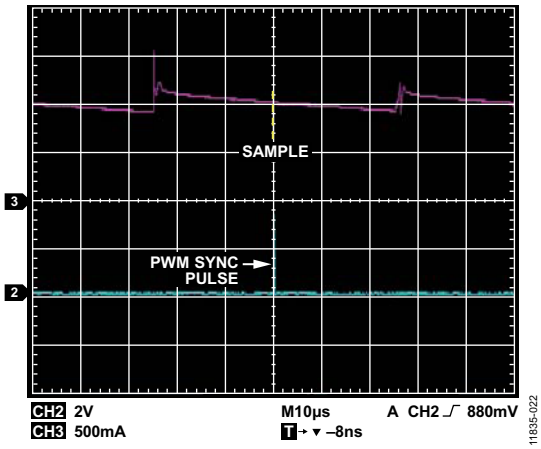

图23. 与相位电流有关的采样

AN11835sc-0-9/14(A)