AN-1265
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-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F SINC滤波器和AD7401A实现隔离式电机控制反馈

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

简介

本应用笔记介绍ADSP-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F SINC滤波器的主要特性，重点讨论高性能电机控制应用。

本文的目的是强调SINC滤波器模块的重要功能，并提供ADI公司SINC滤波器驱动程序的使用指南。有关全系列SINC滤波器特性和配置寄存器的更多信息，请参见ADSP-CM40x混合信号控制处理器(含ARM Cortex-M4硬件参考手册)以及ADSP-CM40x Enablement软件包中的文档。

每款ADSP-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F SINC滤波器均为完整电机电流反馈子系统的一部分；该子系统包含分流器、调制器(用于数字化和隔离信号)、SINC滤波器(解码位流并将结果发送至控制器)。本应用笔记描述如何设置SINC滤波器。

电机电流控制应用

图1显示变频电机驱动器的隔离式电流反馈系统的简化原理图。该系统克服了将分流器产生的模拟信号与开关电源逆变器产生的高压通用信号相隔离的难题。该电路使用隔离式Σ-Δ型调制器转换信号，然后将数字信号跨越隔离栅进行传输，解决这一难题。

Σ-Δ型调制器产生与输入电压成函数关系的调制位流，然后将信号越过隔离栅传输至低压侧的滤波器电路。SINC滤波器过滤来自二阶调制器(如AD7401A)的位流，以便恢复表示电机绕组电流的16位数字信号。

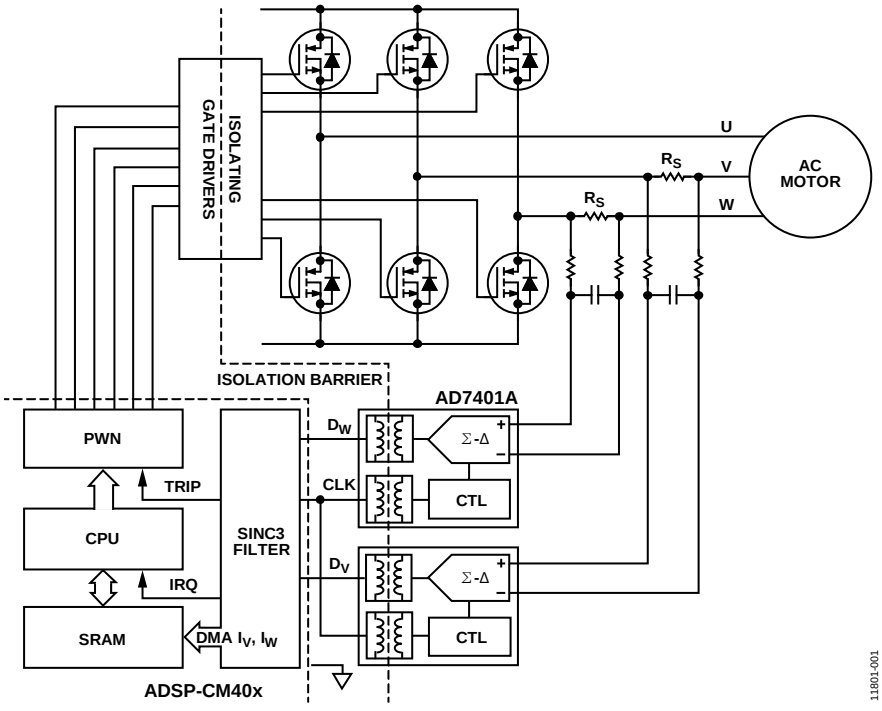

图1. 使用AD7401A的隔离式电流反馈

目录

修订历史

2013年11月—修订版0至修订版A

更改图1 1
更改图4 4
更改表1 5

2013年9月—修订版0：初始版

SINC滤波器模块概述

SINC滤波器模块执行两个功能：为电机控制算法提供高保真反馈信号，并提供故障条件下的过载电流快速检测。将过载故障信号连接至PWM调制器模块可不通过软件干预来关断PWM逆变器。SINC滤波器使用DMA直接将数据传输至存储器，当达到预设的样本数量时便产生处理器中断。这样可以最大限度地减轻SINC滤波器完成配置后的软件服务负担。同样的反馈电路适用于隔离式直流总线电压反馈或直流总线电流测量。

图2显示SINC滤波器模块的框图。SINC滤波器模块由四对SINC滤波器组成，可对连接至输入的数字位流反馈信号实现滤波与过载检测。滤波器使能功能将SINC滤波器对分配至两组配置寄存器的其中之一，以便设置滤波器参数。电机驱动器要求多个电流或电压滤波器采用相同的配置。SINC滤波器模块针对每个电机分配一组(两个)滤波器对，支持对两个电机进行控制。主滤波器的设置包括：滤波器阶数、抽取率、失调偏置和增益调整。次级滤波器的设置包括：滤波器阶数、抽取率、过载跳闸电平和毛刺滤波器设置。

其他配置功能包括：调制器时钟频率、中断屏蔽和DMA数据传输。设置SINC滤波器需要的其他控制外设为端口控制器(将外部引脚连接至SINC滤波器输入)和触发路由单元(即TRU，将SINC输出信号连接至适当外设)。

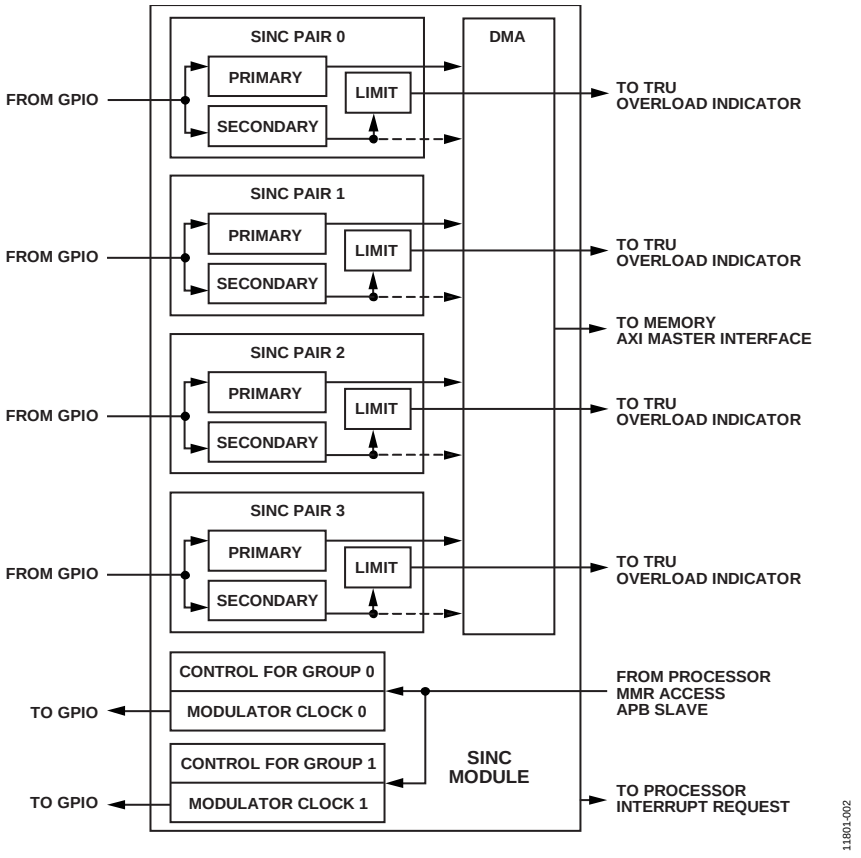

图2. SINC滤波器模块概述

电流反馈系统概述

图4显示电流反馈系统的重要元件。分流器检测绕组电流，将其作为与分流电阻成比例关系的电压信号处理。AD7401A调制器产生隔离式位流，其脉冲密度与满量程输入电压范围成比例。SINC滤波器根据滤波器阶数和抽取率来提取脉冲密度信息。主滤波器参数优化滤波器的精度和额外偏置，定标模块将数据转换为16位带符号整数，然后传输至存储器。次级参数优化滤波器的速度，将输出信号传输至数字比较器，检测过载条件。上限和下限比较器检测电流过载，毛刺滤波器在指定窗口(LWIN)内等待最小过载次数(LCNT)，然后产生过载触发信号。过载触发是PWM调制器驱动电机逆变器的一个跳闸输入信号。当存储器内的绕组电流数据就绪时，DMA传输引擎产生中断信号，触发算法执行。

分流选择

定义反馈所需的系统规格是峰值控制电流$I_{cc}(p)$以及调制器的额定最大输入电压$V_{mod}(max)$。电源逆变器的峰值电流能力通常定义控制电流范围，但可能需要考虑其他因素。AD7401A调制器的额定最大工作电压为±200 mV，该数值为最大电压范围，在其范围内调制器规格数据有效。该值低于调制器的±320 mV满量程范围(V_{FS})，因为满量程输入附近的线性度和信噪比性能下降非常严重。分流电阻值必须低于$V_{mod}(p)/I_{cc}(p)$以满足这些限制条件；此处选择最接近的标称分流电阻值。对于图3中的示例而言，若电源级峰值电流额定值为8.5 A，则最大分流电阻值为23.5 mΩ。最接近该条件的分流电阻标称值为20 mΩ，额定最大电流为10 A。

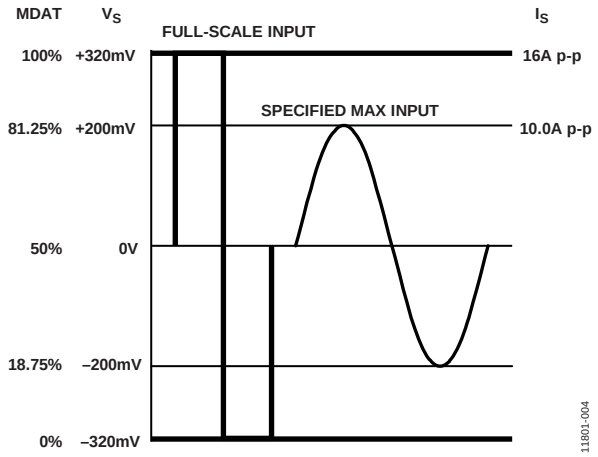

图3. 反馈电流工作范围

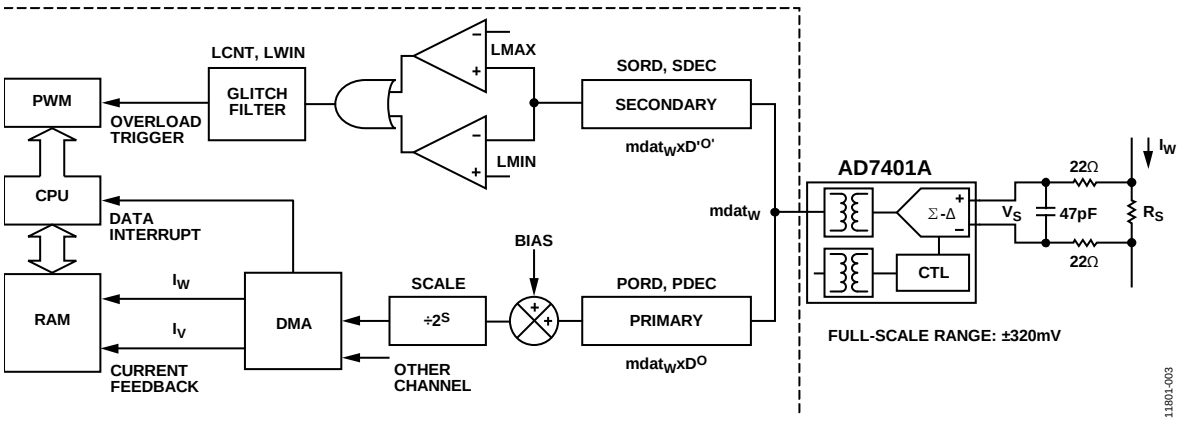

图4. SINC滤波器电流反馈路径

调制器时钟、主滤波器抽取和数据中断速率选择

SINC滤波器性能由调制器时钟(f_M)和抽取率(D)参数确定。滤波器阶数(O)通常比前端调制器高一阶。因此，使用AD7401A时，滤波器阶数为3。滤波器频率响应和群组延迟的计算式如下所示。哪怕频率为抽取频率(f_M/D)的数倍，图5所示的频率响应依然为0秒。因此，将抽取频率与PWM开关频率匹配可显著降低PWM开关谐波。其他考虑因素包括随抽取率上升的群组延迟，以及滤波器的最大抽取限值。

$$H\left(e^{i\frac{f}{f_M}}\right)=\left[\frac{1}{D}\times\frac{\sin\left(D\frac{\pi f}{f_M}\right)}{\sin\left(\frac{\pi f}{f_M}\right)}\times e^{-j(D-1)\frac{\pi f}{f_M}}\right]^O$$

$$\tau_d=\left(\frac{D-1}{2}\right)\frac{0}{f_M}$$

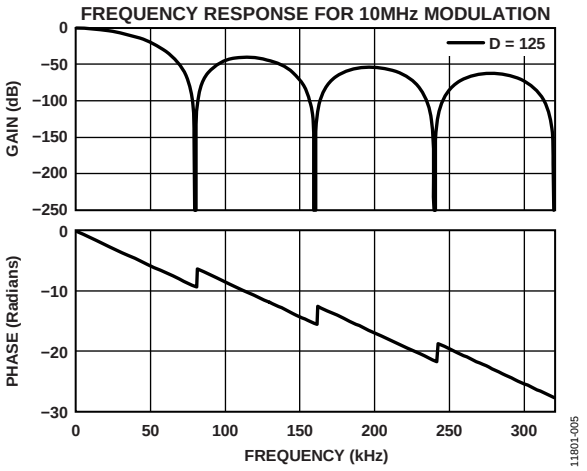

图5. SINC滤波器频率响应

对于给定的滤波器阶数，抽取率和滤波器阶数是决定滤波器性噪比(SNR)和群组延迟的滤波器参数。图6和表1显示三阶滤波器以及10 MHz调制器时钟的不同SNR、有效位数(ENOB)和群组延迟与抽取率的关系。抽取率必须位于85至210范围内，以获得11位至14位ENOB(以及67 dB至86 dB的SNR)；该范围是电流反馈所需的滤波器性能范围。在该抽取率范围内，群组延迟为12 µs至32 µs。

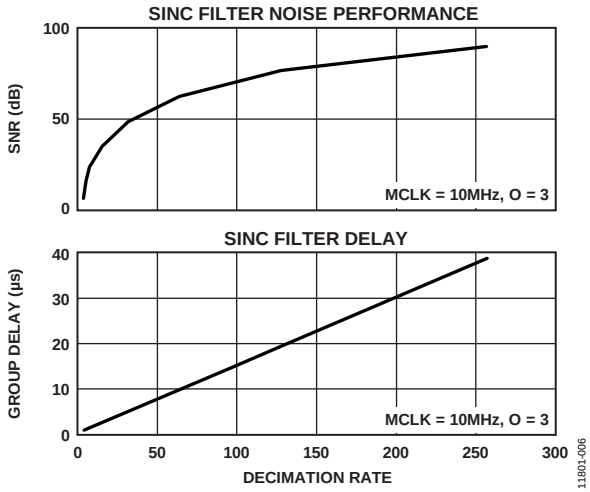

图6. 次级滤波器SNR

表1. 不同抽取率下的SNR、ENOB和群组延迟[1]

抽取率	SNR (dB)	ENOB(位)	群组延迟(µs)
85	68	11	12.6
113	74	12	16.8
154	80	13	23.0
210	86	14	31.4

[1] 测试条件：±200 mV正弦波，1.22kHz。

在电机驱动器中，不可能将SINC滤波器抽取率与典型PWM开关频率相匹配。匹配16 kHz开关频率所需的抽取率为625，而滤波器群组延迟为94 µs。该抽取率远高于可提供的数值，而群组延迟将限制电流环路带宽。相反，抽取率设为PWM频率的倍数，以便降低群组延迟并依然获得目标滤波器SNR。控制算法在抽取频率的约数位置进行数据采样，而抽取频率匹配PWM开关频率。该软件抽取过程包括：将多个数据样本传输至存储器的循环缓冲器中，以及读取最近的数据样本，以便响应缓冲器填满时产生的中断。DMA引擎将数据从主SINC滤波器传输至数据存储器，而SINC控制单元每次传输固定数量的样本后便产生一次触发动作。

图7显示PWM开关、调制器、抽取和数据采样之间的对齐。PWM调制器发出的同步脉冲(PWM0_SYNC)将调制器的启动时钟与PWM频率对齐。抽取频率等于调制器时钟的约数和PWM频率的倍数。SINC0_DAT0触发速率等于PWM频率。

表2中的信息显示选择抽取率和PWM开关频率的过程。表格中的前三条数据是内核和外设时钟的芯片级设置。内核时钟速率最大值为240 MHz，通常等于系统(外设)时钟频率的偶数倍。SINC滤波器调制器的时钟(MCLK)来自系统时钟，数值基于MDIV场寄存器值，且在5 MHz至20 MHz范围内有一组有限值。主抽取率(PDEC)为125，该值将滤波器的SNR设为76 dB(高于12位ENOB)，且群组延迟为18.6 μs。在1.25 kHz典型电流控制环路带宽下，延迟对应的相位滞后仅为8°。调制器时钟为10 MHz；因此，主抽取时钟频率为80 kHz，且数值为5的软件抽取率(SWDEC)将采样速率与16 kHz PWM频率(PWM)同步。调整SINC滤波器抽取率可调节PWM频率。

决定调制器时钟、PWM频率和软硬件抽取率的等式为：

$$\frac{MCLK}{PWM} = PDEC \times SWDEC$$

其中：

*PDEC*表示硬件抽取率。

*SWDEC*表示软件抽取率。

硬件和软件抽取率必须为整数。SINC滤波器中的PCNT场寄存器值确定软件抽取率。载入SINC滤波器控制寄存器中的PCNT值比产生中断前的采样延迟数小1。PWM_TM0寄存器确定PWM开关频率，从而确定采样时序。

表2. 抽取率选择

参数	符号	数值	单位
内核时钟	CCLK	240	MHz
系统时钟分频器	SYSSEL	3	
系统时钟	SYSCLK	80	MHz
调制器时钟分频器	MDIV	8	
调制器时钟($1/T_M$)	MCLK	10	MHz
抽取率	PDEC	125	
滤波器SNR	SNR	76.0	dB
滤波器ENOB	ENOB	12.3	位
抽取频率	DCLK	80.0	kHz
滤波器群组延迟	τ_d	18.6	μs
软件抽取率	SWDEC	5	
数据传输次数	PCNT	4	
PWM频率($1/T_S$)	PWM	16.00	kHz
PWM周期数	PWMTM	2500	

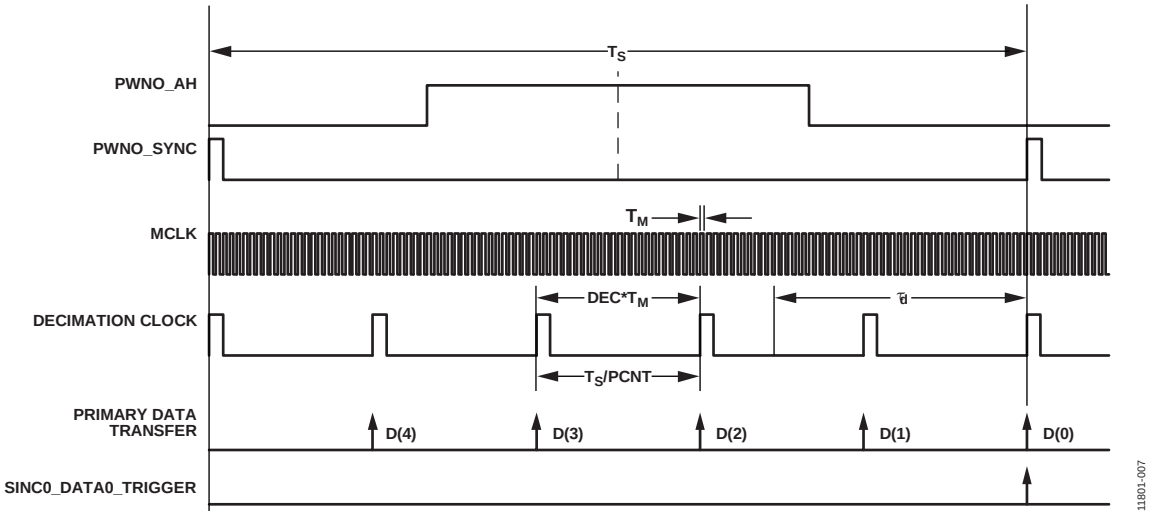

图7. 调制器和抽取时钟时序

主滤波器定标

SINC滤波器阶数(O)和抽取率(D)确定主滤波器直流增益，表达如下：

$$G_{dc} = D^O$$

该单元具有输出定标和偏置功能，可将数据转换为16位带符号整数，然后将其传输至存储器。取决于不同的表述，数据格式在±1.0范围内作为16位分数整数(S.15)有效，或在$\pm 2^{15}$范围内作为带符号16位整数有效。

原始滤波器输出为0至D^O范围内的整数，其中$D^O/2$与50%脉冲密度(相当于0 A)对齐。在输出端加入$-D^O/2$的偏置值可设置正确的零电平。将结果除以$D^O/2$可将满量程分数整数输出定标至±1，但为了简便起见，该单元具有简单的二进制比例因子，用户可选择S，设置1.0附近的增益。无论如何定标，DMA引擎仅传输输出数据的16个最低有效位，因此必须正确定标，避免损失精度。输出数据饱和，防止数据溢出；如果比例因子选择不当，则数据溢出可能会使输出信号的极性反转。发生饱和时，滤波器置位溢出故障标志。

只需反转分流增益用于定标并调节滤波器直流增益和比例因子之间的失配，即可将数据转换为浮点数。

反馈定标计算

存储器中分流电流至数据字的最终系统增益由系统中所有元器件的增益确定，如图8所示。本例中使用的隔离式调制器为AD7401A。

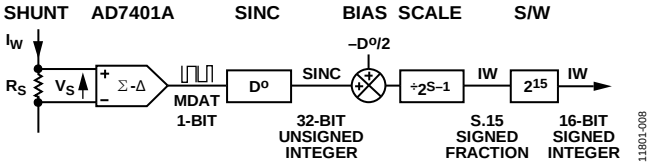

图8. SINC主输出数据定标

调制器得到的分流电压为：

$$v_S = i_W \times R_S$$

隔离式调制器的预期输入为双极性输入，0 V输入的情况下具有50%脉冲密度。脉冲密度与输入电压(v_s)和正满量程输入(V_{FS})之比成函数关系：

$$MDAT = 0.5\left(\frac{v_S}{V_{FS}} + 1\right)$$

对于AD7401A，正满量程电压为320 mV，额定最大电压为200 mV时1的密度为81.25%。

SINC滤波器直流增益为D^O；因此，原始输出与输入电压的函数关系为：

$$SINC = \frac{D^O}{2}\left(\frac{v_S}{V_{FS}} + 1\right)$$

该直流定标适用于次级滤波器输出，最大次级抽取率将原始输出数据范围限制为16位无符号整数。负满量程输入时的次级输出为0，正满量程输入时的次级输出为D^O。

主输出路径上的偏置和定标功能可消除SINC数据偏置，并将数据重新定标为16位带符号整数。偏置值必须等于$-D^O/2$以消除SINC输出失调，并用于双极性输入范围调制器。重新定标功能从SINC输出字中选择适当的位范围。

$$IW = \frac{SINC - \frac{D^O}{2}}{2^{S-1}} = \frac{D^O}{2^S}\left(\frac{v_S}{V_{FS}}\right)$$

比例因子(S)必须将最大分数整数输出设为1.0；当满足下式时该目的可实现：

$$\frac{D^O}{2^S} << 1 \therefore S >> \frac{\ln(D^O)}{\ln 2}$$

读取带符号整数数据时，SINC输出等式加入2^{15}比例因子。

$$IW = \frac{D^O}{2^S}\left(\frac{v_S}{V_{FS}}\right) \times 2^{15}$$

这种情况下，与实际绕组电流(i_w)成函数关系的电流读数为：

$$IW = i_W \times \left(\frac{R_S}{0.32}\right)\left(\frac{D^O}{2^S}\right) \times 2^{15}$$

次级滤波器定标和过载配置

次级SINC滤波器数据输出直接连接过载比较器和毛刺滤波器，如图4所示。相比主滤波器，次级滤波器的抽取率设为一个低得多的值，以便针对故障条件提供更快的响应。处理器触发路由单元(TRU)将过载跳闸信号连接至PWM调制器的关断输入端，清除故障。TRU还可将过载信号连接至其他源，如用于关断其他重要电路元件的外部GPIO。

典型电源逆变器开关可耐受几微秒的短路；因此，过载电路必须具有相对较窄的检测窗口。由于SINC滤波器能够响应三个抽取周期内的步进输入，因此使用抽取率10可实现3 µs以内的响应，如图9所示。SINC滤波器还可滤除逆变器开关噪声，如图10所示。在该图中，将16 A噪声脉冲注入10 A峰值测试波形中，持续1.5 µs；并注入持续时间为40 µs的16 A过载脉冲。滤波器抑制短噪声脉冲，但电路检测16 A过载脉冲。该测试中的最大和最小跳闸电平位于次级SINC输出端，对应±16 A。

在较低的抽取率下可达到更快的响应，但如图11所示，哪怕在±10 A的简单正弦波测试电流下，次级SINC输出电压都会超过跳闸电平。抽取率为5时，较高的SINC滤波器噪声产生多个伪跳闸信号。图12显示高(10)和低(5)抽取率下的SNR以及跳闸信号的噪声裕量。

次级输出毛刺滤波器采用跳闸计数窗口(WCNT)，以低于最小次数(LCNT)的持续时间消除跳闸，从而抑制短过载跳闸。图13显示毛刺滤波器如何消除抽取率为5时触发的杂散过载；但响应时间具有额外三个周期的延迟。因此，较低的抽取率不会降低响应时间。图中显示滤波器抑制模拟输入短噪声脉冲的能力。在该示例中，噪声脉冲持续时间为1.5 μs。

次级SINC滤波器包含一组历史缓冲器，用于在跳闸前存储8个最新的数据样本，以供诊断。可通过器件外设存储器架构直接访问历史寄存器中的数据。

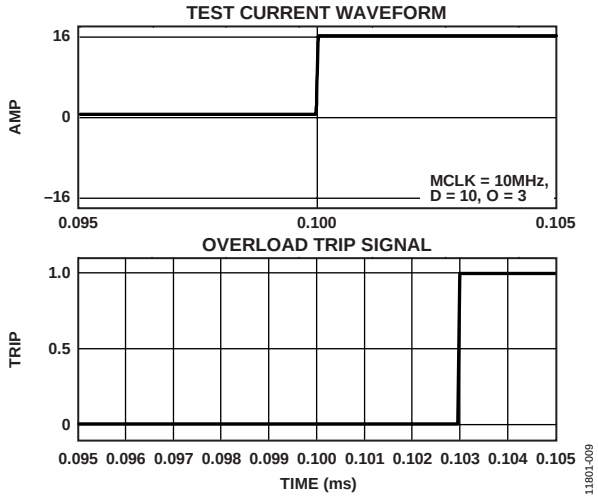

图9. 次级滤波器过载检测

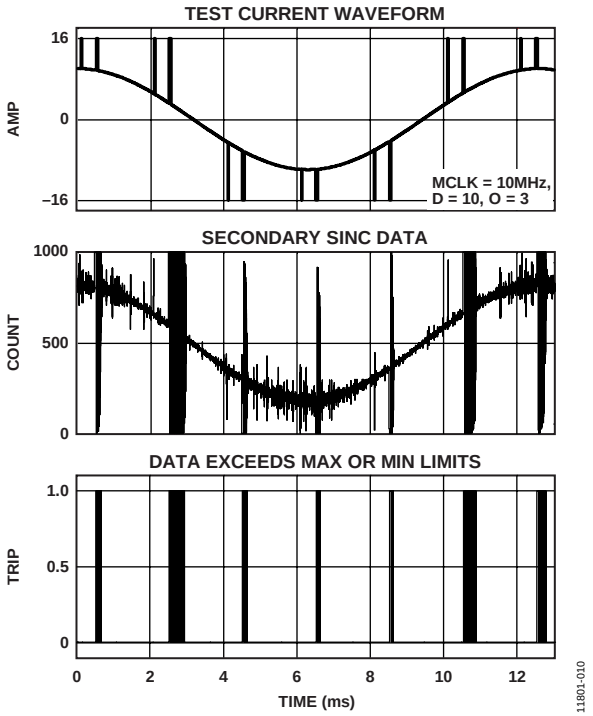

图10. 以抽取率10执行过载检测

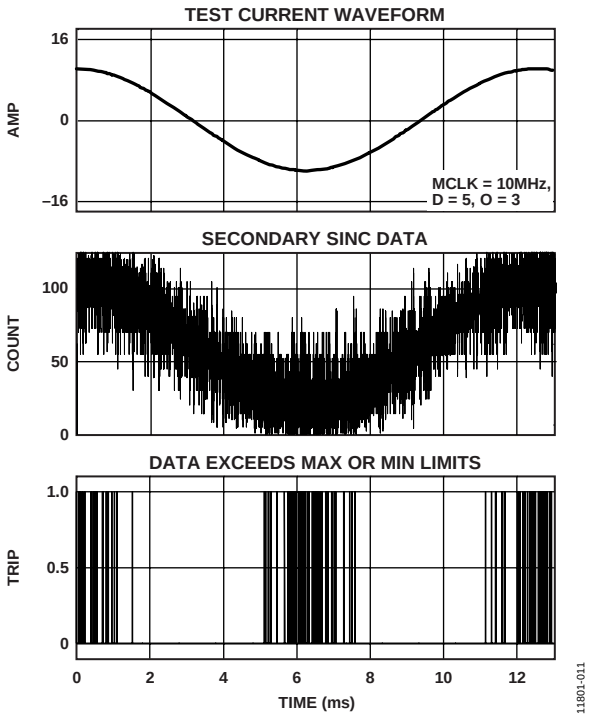

图11. 以抽取率5执行伪过载检测

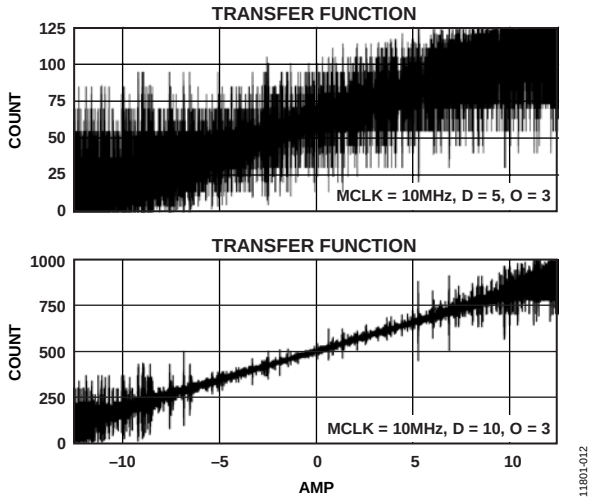

图12. 抽取率为5和10时的次级滤波器增益曲线

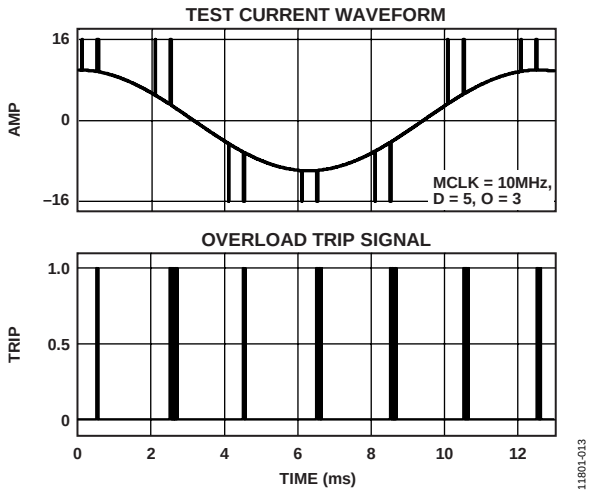

图13. 抽取率为5时的过载检测以及WCNT = 4且LCNT = 4时的毛刺滤波器

次级滤波器定标和跳闸电平

次级滤波器无额外输出定标，但有效的最小和最大值范围为0至D^O。负满量程电流映射至0，正满量程电流映射至D^O。将最小和最大跳闸电平设为1和D^O – 1可让跳闸功能具有最大范围。图12中，底部图形所示的传递函数(抽取率为10，分流电阻为20 mΩ)显示10 A输入的噪声峰值位于滤波器最大(1000次)和最小(0次)输出范围内。将LMIN和LMAX跳闸电平设为1次计数和999次计数可避免10 A峰值电流杂散跳闸信号。触发跳闸的实际电流范围为11 A至16 A满量程电平。电流越接近满量程限值，跳闸的可能性就越大。

过载电路在额定调制器输入范围内的工作精度略高。在精度较高的情况下，5 A输入的峰值噪声为700次计数，等效于6.4 A。因此，可将跳闸设为在5 A至6.4 A范围内工作。这种情况下的LMAX和LMIN设置分别为700次和300次计数。采用较低的抽取率获得精确跳闸设置较为困难。

SINC模块故障检测功能

“次级滤波器定标和过载配置”部分描述了达到所需主和次级滤波器性能的各种滤波器参数设置要求。除过载故障外，SINC模块还检测由于异常滤波器设置导致的数据故障，该故障会使芯片特定的模块过载。

存在输出偏置和定标异常时，主滤波器会检测输出数据饱和。若滤波器DMA引擎无法在滤波器写入新数据前传输数据，则它会检测FIFO故障。发生饱和以及FIFO故障时，SINC_CTL寄存器中的ESATx和EFOVFx位屏蔽SINC0_STAT产生的中断。

SINC滤波器设置和软件驱动程序功能

使用滤波器前，应执行数个步骤，设置SINC滤波器模块以及信号路由和数据缓冲器。完成配置后，DMA引擎便可自动将主滤波器数据流传输至存储器，并且次级限制功能在发生过载时可关断PWM模块。数据就绪时，系统产生中断，以便处理器执行控制算法，更新PWM调制器寄存器。图14显示SINC滤波器模块和CPU、SRAM、PWM以及外部引脚之间的互连布局，用以捕获电机电流反馈信号。

使用SINC滤波器设置电流反馈需执行四个步骤：

1. 配置引脚多路复用器。
2. 分配数据缓冲器存储器。
3. 连接中断并触发路由。
4. 配置主滤波器和次级滤波器。

本小节进一步讨论这些步骤，详细说明设置过程以及ADI器件的驱动程序功能，以便设置系统和SINC滤波器控制寄存器。

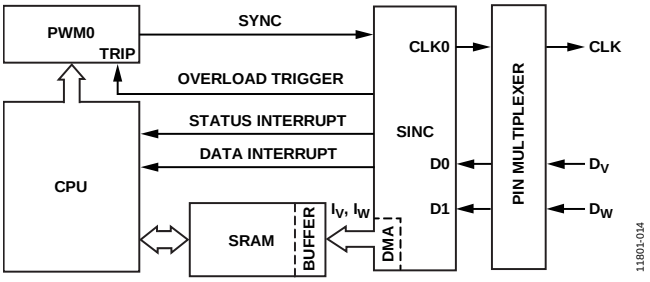

图14. SINC滤波器系统配置

引脚多路复用器配置

引脚多路复用器将前端调制器时钟和数据引脚连接至SINC模块。有两个可用的调制器时钟输出(SINC0_CLK0和SINC0_CLK1)，以及四个可用的SINC数据输入引脚(SINC0_D0、SINC0_D1、SINC0_D2和SINC0_D3)。PORT_MUX寄存器控制这些引脚的选择，可针对每个多路复用引脚从四个交替输入或输出信号中作出选择。PinMux64.jar和PinMux32.jar Java应用程序随ADSP-CM40x Enablement软件包提供，可自动生成C语言代码，使能用户端口选择。图15是PinMux64.jar应用软件窗口快照。

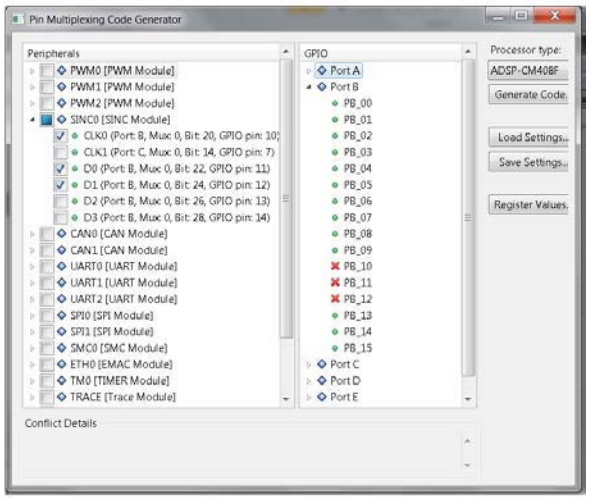

图15. 引脚多路复用器选择工具

数据缓冲器存储器分配

必须定义主滤波器数据缓冲器，并为其分配存储器空间，以便允许控制算法使用数据。软件抽取率和反馈通道数确定存储器尺寸。通道序列中的数据按组排列。指向最近数据组的指针存储在SINC_PPTRx寄存器中。“中断和触发路由”部分描述的器件驱动程序可管理缓冲器和指针。

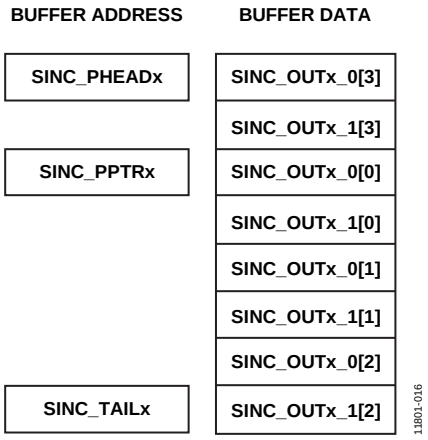

图16. 数据缓冲器结构图

中断和触发路由

图17描述使用中断和触发信号将SINC滤波器与其他外设功能互连的情况。SINC_STAT是SINC滤波器模块的单次处理器中断信号。触发路由单元(TRU)将其他触发信号连接至SINC滤波器模块的外设和处理器中断。在TRU中将触发主机地址载入触发从机寄存器可连接路由。

TRU将SINC滤波器调制器和抽取时钟与PWM调制器频率同步，以满足图7中定义的时序要求。TRU还将SINC滤波器数据传输触发信号与控制软件中断信号相连，开始执行控制算法。

TRU将两个SINC过载触发连接至PWM调制器TRIP1输入，使能过流保护。TRIP0输入仅连接外部跳闸信号。必须配置PWM调制器、TRIP0以及TRIP1输入，以便接受这些触发信号。过载故障可产生两种中断：SINC_STAT中断直接连接CPU，而SINC过载触发产生PWM_TRIP1中断。“SINC滤波器软件支持”结尾部分的代码段包含器件驱动程序的调用，可实现这些连接。

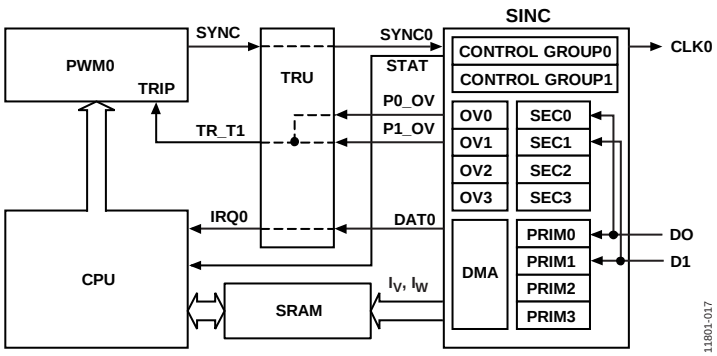

图17. SINC滤波器触发路由

主和次级滤波器配置

对滤波器通道进行分组管理，因为通常2或3个反馈信号需要使用同一个滤波器参数。SINC模块有两组配置寄存器。任意组的通道具有相同的时钟和滤波器参数，如滤波器阶数、抽取率、定标和偏置。例外情况是过载限值和历史寄存器，它们具有独立的通道结构。使能某一滤波器通道即可将其分配至某一配置组。配置寄存器确定调制器时钟、滤波器参数、DMA数据传输和过载检测。

图18描述群组0寄存器的滤波器和系统参数分配。群组1寄存器的结构与之类似。SINC_CTL寄存器使能每个通道，并分配控制组。建议过程为：先配置滤波器组，然后在群组中使能通道。SINC_CTL寄存器同样可以屏蔽SINC_STAT中断。系统状态寄存器SINC_STAT汇报故障和数据触发计数状态。

每组3个寄存器以及时钟寄存器确定主和次级滤波器参数。SINC_RATE0和SINC_RATE1设置主和次级滤波器抽取率(PDEC、SDEC)以及主滤波器相位(通常为0°)。SINC_LEVEL0和SINC_LEVEL1确定主和次级滤波器阶数(PORD、SORD)以及主滤波器定标(PSCALE)。BIAS0和BIAS1确定主滤波器数据失调。SINC_CLK寄存器确定CLK0和CLK1调制器时钟频率，可通过外部触发使能同步。需要的话，该寄存器还可调节时钟相位。

每组3个寄存器支持主滤波器DMA通道。SINC_PHEAD0和SINC_TAIL0确定群组0主滤波器输出数据缓冲器的存储器地址。SINC_PPTR0寄存器将指向最新数据的指针保存在缓冲器中。SINC_LEVEL0寄存器的PCNT位确定每个数据中断的数据传输数量，从而设置软件抽取率(PCNT + 1)。

每通道由5个寄存器支持次级过载检测功能。SEC_LIMIT0确定最大和最小过载阈值，且P0SEC_HIST0、P0SEC_HIST1、P0SEC_HIST2和P0SEC_HIST3在过载跳闸前存储最后8个次级滤波器输出。SINC_LEVEL0和SINC_LEVEL1寄存器设置相关群组中通道的次级滤波器毛刺参数(LWIN、LCNT)。

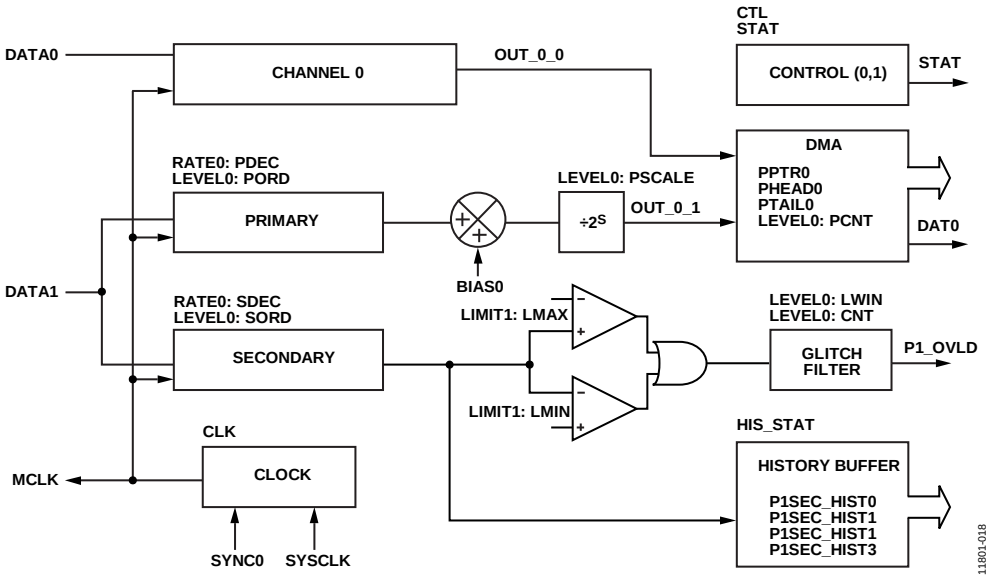

图18. SINC寄存器映射

SINC滤波器软件支持

下文代码片段给出一个如何设置双通道电流反馈中主滤波器和次级滤波器的示例。这些代码片段从工作代码中提取，并在闭环电机控制评估平台上测试。器件驱动程序增加了一些开销，但极大简化了SINC模块寄存器的编程。该函数调用与本文所用参数名相匹配的常量名，因此大部分代码无需注释。

第一段代码(行[1:21])定义一系列参数常量。第9行和第10行定义主SINC数据的数据缓冲器尺寸。下一段代码(行[22:36])设置原型功能并分配存储器。第28行和第29行定义的SINC回调函数处理SINC_DATA0和SINC_STAT中断。

SetupTRU代码段(行[37:46])包含所有触发路由。SetupPWM代码段(行[47:74])包含PWM定时器频率、同步脉冲和跳闸功能设置。外部硬件跳闸信号连接TRIP0，内部SINC_Px_OVLD触发连接TRIP1。TRIP1中断是SINC过载产生的中断之一。过载还产生SINC_STAT中断。

SetupSINC代码段(行[75:106])配置SINC滤波器参数。行[78:80]打开器件驱动程序并设置回调函数。行[81:85]设置各种群组参数，包括阶数和抽取率。行[87:89]控制过载限值的初始设置，将其设为整个范围以避免滤波器启动时的杂散跳闸信号。第86行设置循环缓冲器，用于主SINC数据；第94行和第95行分配数据通道到缓冲器。第91行和第92行设置调制器时钟。第91行驱动程序计算系统时钟和调制器频率的分频比。第92行调用驱动程序使能时钟并设置同步模式。第97行使能SINC_STAT中断屏蔽。第98行和第99行使能分配至群组0的滤波器通道0和滤波器通道1。第100行和第101行在指定SINC中断屏蔽(第103行)之前引入短延迟，并设置次级滤波器过载限值(第104行和第105行)。

最后一段代码(行[106:129])包含回调函数，用于SINC数据和过载中断。SincDataCallback函数将缓冲器的数据复制到电机控制变量中，并调用控制函数。SincStatusCallback调用故障处理程序。

```
/*************************************
SINC FILTER SETUP CODE SNIPPETS
*************************************/
/****   Include file #define code    ***/
/* SINC definitions */
#define SINC_DEV                0
#define SINC_NUM_SAMPLES        4
   /* this determines how often a data
   interrupt is generated */
#define SINC_NUM_PAIRS          2
#define SINC_DATA_SIZE
   (SINC_NUM_PAIRS * SINC_NUM_SAMPLES)
#define CIRC_BUF_SIZE
   (SINC_NUM_SAMPLES*20)
   /* size for the device circular buffer */
#define SINC_MODCLK             (5000000)
   /* modulator clock freq */
#define PDEC                125
   /* primary decimation */
#define PSCALE              24
   /* Primary scale */
#define SDEC                5
   /* secondary decimation */
#define LWIN                4
   /* Glitch window    */
#define LCNT                4
   /* Glitch count */
#define LMAX           124
   /* Overload max limit */
#define LMIN           1
   /* Overload min limit */
/* TRU definitions*/
#define TRU_DEV_NUM             0
#define ADI_TRU_REQ_MEMORY      4u


/*  SINC related P R O T O T Y P E S  and
   memory allocation                */
/* Function prototypes */
void SetupPWM(void);
void SetupTRU(void);
void SetupSINC(void);
/* Prototype for callback functions */
static void SincDataCallback(void
   *pHandle, uint32_t event, void *pArg);
static void SincStatusCallback(void*
   pHandle, uint32_t event, void* pArg);
/* SINC handler and data buffers */
static uint8_t
   SincMemory[ADI_SINC_MEMORY_SIZE];
static ADI_SINC_HANDLE hSINC;
static int16_t sincData1[SINC_DATA_SIZE];
static int16_t sincData2[SINC_DATA_SIZE];
static int16_t
   sincCircBuffer[CIRC_BUF_SIZE];
/***********************************/


void SetupTRU(void){
/*****       Function: SetupTRU  (code
   snippet for SINC related setup)  *******/
ADI_TRU_RESULT result;
result = adi_tru_Open (TRU_DEV_NUM,
   &TruDevMemory[0], ADI_TRU_REQ_MEMORY,
   &hTru);
// Setup TRU for SINC. Slave is SINC0
   SYNC0, master is PWM0 sync pulse
result = adi_tru_TriggerRoute (hTru,
   TRGS_SINC0_SYNC0, TRGM_PWM0_SYNC);  //
   TRU device, slave, master
result = adi_tru_TriggerRoute (hTru,
   TRGS_PWM0_TRIP_TRIG1,
   TRGM_SINC0_P0_OVLD); /* connect
   SINC_Px_OVLD trigger to PWM0_TRIP_TRIG1.
   slave, master */
result = adi_tru_TriggerRoute (hTru,
   TRGS_PWM0_TRIP_TRIG1,
   TRGM_SINC0_P1_OVLD); /* Both overload
   detection on TRIP1. TRIP0 used by HW */
result = adi_tru_Enable (hTru, true); //
   Enable TRU
}


void SetupPWM(void){
/****   Function: SetupPWM  (code snippet
   for SINC related setup)  *******/
static ADI_PWM_RESULT result;
uint32_t temp = 0;


result = adi_pwm_Open(PWM_DEV,
   &PwmMemory, ADI_PWM_MEMORY_SIZE, &hPWM,
   PwmCallback, NULL); // Open driver


temp = (uint32_t)(fsysclk / (2u * FPWM));
   // Calculate switching period as number
   of sys clocks (up-down counter)
result = adi_pwm_SetReferencePeriod(hPWM,
   temp);


temp = (uint32_t)(fsysclk *
   SYNC_PULSE_WIDTH);       // Calculate
   sync pulse width as number of sys clocks
   (up-down counter)
result = adi_pwm_SetSyncWidth(hPWM,
   temp);
result = adi_pwm_ExtSyncEnable(hPWM,
   false, false);          // Internal sync
   used
```

```
result =
adi_pwm_SetIntSyncTimerMode(hPWM,
ADI_PWM_TIMER0);  // Use timer 0 to
generate sync.


result = adi_pwm_SetTripEnable(hPWM,
ADI_PWM_CHANNEL_A, ADI_PWM_TRIP0_SRC,
true);    // Enable Trip0 and trip on all
channels
result = adi_pwm_SetTripEnable(hPWM,
ADI_PWM_CHANNEL_B, ADI_PWM_TRIP0_SRC,
true);
result = adi_pwm_SetTripEnable(hPWM,
ADI_PWM_CHANNEL_C, ADI_PWM_TRIP0_SRC,
true);
result = adi_pwm_SetTripEnable(hPWM,
ADI_PWM_CHANNEL_A, ADI_PWM_TRIP1_SRC,
true);    // Enable Trip1 and trip on all
channels
result = adi_pwm_SetTripEnable(hPWM,
ADI_PWM_CHANNEL_B, ADI_PWM_TRIP1_SRC,
true);
result = adi_pwm_SetTripEnable(hPWM,
ADI_PWM_CHANNEL_C, ADI_PWM_TRIP1_SRC,
true);


result = adi_pwm_SetTripMode(hPWM,
ADI_PWM_CHANNEL_A, ADI_PWM_TRIP0_SRC,
false);     // Stop PWM and report fault
at trip. Do not restart
result = adi_pwm_SetTripMode(hPWM,
ADI_PWM_CHANNEL_B, ADI_PWM_TRIP0_SRC,
false);
result = adi_pwm_SetTripMode(hPWM,
ADI_PWM_CHANNEL_C, ADI_PWM_TRIP0_SRC,
false);
result = adi_pwm_SetTripMode(hPWM,
ADI_PWM_CHANNEL_A, ADI_PWM_TRIP1_SRC,
false);
result = adi_pwm_SetTripMode(hPWM,
ADI_PWM_CHANNEL_B, ADI_PWM_TRIP1_SRC,
false);
result = adi_pwm_SetTripMode(hPWM,
ADI_PWM_CHANNEL_C, ADI_PWM_TRIP1_SRC,
false);


result = adi_pwm_InterruptEnable(hPWM,
ADI_PWM_INTERRUPT_TIMER0, true);
// Enable sync irq
result = adi_pwm_InterruptEnable(hPWM,
ADI_PWM_INTERRUPT_TRIP0, true);
// Enable trip0 irq
result = adi_pwm_InterruptEnable(hPWM,
ADI_PWM_INTERRUPT_TRIP1, true);
// Enable trip1 irq


/*** other PWM setup code **/
}


void SetupSINC(void){
/***           Function: SetupSINC
******/


static ADI_SINC_RESULT result;


result = adi_sinc_Open(SINC_DEV,
SincMemory, ADI_SINC_MEMORY_SIZE,
&hSINC);


result = adi_sinc_RegisterDataCallback
(hSINC, SincDataCallback, 0);
result = adi_sinc_RegisterStatusCallback
(hSINC, SincStatusCallback, 0);


/* Specify Group Parameters */
result = adi_sinc_SetRateControl (hSINC,
ADI_SINC_GROUP_0,
ADI_SINC_FILTER_PRIMARY, PDEC, 0);
result = adi_sinc_SetRateControl (hSINC,
ADI_SINC_GROUP_0,
ADI_SINC_FILTER_SECONDARY, SDEC, 0);
result = adi_sinc_SetLevelControl (hSINC,
ADI_SINC_GROUP_0, LWIN, LCNT,
SINC_NUM_SAMPLES, PSCALE);
result = adi_sinc_SetFilterOrder (hSINC,
ADI_SINC_GROUP_0,
ADI_SINC_FILTER_THIRD_ORDER,
ADI_SINC_FILTER_THIRD_ORDER);
result = adi_sinc_SetCircBuffer(hSINC,
ADI_SINC_GROUP_0, sincCircBuffer,
CIRC_BUF_SIZE);


/* Reset overload amplitude detection
limits to 0 – FullScale  */
result = adi_sinc_SetAmplitudeLimit
(hSINC, ADI_SINC_PAIR_0, 0x0000, 0xFFFF);
result = adi_sinc_SetAmplitudeLimit
(hSINC, ADI_SINC_PAIR_1, 0x0000, 0xFFFF);


/* Specify Modulator Clock frequency,
phase & startup synchronization  */
result = adi_sinc_ConfigModClock (hSINC,
ADI_SINC_GROUP_0, fsysclk, SINC_MODCLK,
0, false);
result = adi_sinc_EnableModClock (hSINC,
ADI_SINC_GROUP_0,
ADI_SINC_MOD_CLK_PWM_SYNC);


/* submit buffers to receive SINC data */
result = adi_sinc_SubmitBuffer(hSINC,
ADI_SINC_GROUP_0, sincData1, 16);
```

```
result = adi_sinc_SubmitBuffer(hSINC,
ADI_SINC_GROUP_0, sincData2, 16);


/* route the TRU interrupt */
result = adi_sinc_EnableDataInterrupt
(hSINC, ADI_SINC_GROUP_0,
ADI_SINC_DATA_INT_0, true);


result = adi_sinc_EnablePair(hSINC,
ADI_SINC_PAIR_0, ADI_SINC_GROUP_0, true);
result = adi_sinc_EnablePair(hSINC,
ADI_SINC_PAIR_1, ADI_SINC_GROUP_0, true);


  for (int i=0; i<500; i++) // Wait 10us
to let data propagate through the filter
before setting trip limits.
  asm("nop;");


  /* Enable & assign used SINC filter
pair, and specify interrupt masks */
  result = adi_sinc_SetControlIntMask
(hSINC,
ADI_SINC_INT_EPCNT0|ADI_SINC_INT_EFOVF0 |
ADI_SINC_INT_EPCNT1|ADI_SINC_INT_EFOVF1 |
ADI_SINC_INT_ELIM0);


  result = adi_sinc_SetAmplitudeLimit
(hSINC, ADI_SINC_PAIR_0, LMIN, LMAX);
  result = adi_sinc_SetAmplitudeLimit
(hSINC, ADI_SINC_PAIR_1, LMIN, LMAX);
  }
  /***        Function:
SincDataCallback             ****/
  static void SincDataCallback(void*
pHandle, uint32_t event, void* pArg){
  static uint16_t *bufferPtr;
  bufferPtr = (uint16_t*)pArg;  /*
pointer to sincData1 or sincData2 */
  switch((ADI_SINC_EVENT)event){
  case ADI_SINC_EVENT_DATA0:
  Mctrl_U.ibc_sinc[1] = *bufferPtr;
  Mctrl_U.ibc_sinc[0] = *(bufferPtr+1);
  MotorControl();       /* Algorithm
call */
  break;
  case ADI_SINC_EVENT_STATUS:
  break;
  default:
  break;
  }
  }

*/
  static void SincStatusCallback(void*
pHandle, uint32_t event, void* pArg){
  ADI_SINC_EVENT eEvent =
(ADI_SINC_EVENT)event;
  uint32_t status = (uint32_t)pArg;
  if (status & ADI_SINC_STATUS_GLIM0){
  SINC_TRIP_Fault_handler();
  }
  }
```


AN11801sc-0-11/13(A)