

Isolated Motor Control Feedback Using the **ADSP-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F** SINC Filters and the **AD7401A**

By Dara O'Sullivan, Jens Sorensen, and Aengus Murray

INTRODUCTION

This application note introduces the main features of the [ADSP-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F](#) SINC filters, with a focus on high performance motor control applications.

The purpose of this application note is to highlight the key capabilities of the SINC filter module and to provide guidance on the use of the SINC filter drivers provided by Analog Devices, Inc. For more information about the full range of SINC filter features and configuration registers, see the [ADSP-CM40x Mixed-Signal Control Processor with ARM Cortex-M4 Hardware Reference Manual](#) and the documentation within the [ADSP-CM40x Enablement Software package](#).

Each [ADSP-CM402F/ADSP-CM403F/ADSP-CM407F/ADSP-CM408F](#) SINC filter is part of a complete motor current feedback subsystem that includes a current shunt, a modulator to digitize and isolate the signal, and the SINC filter to decode

the bit stream and present it to the controller. This application note describes how to set up the SINC filters.

MOTOR CURRENT CONTROL APPLICATIONS

Figure 1 shows a simplified schematic of an isolated current feedback system for inverter fed motor drives. The system overcomes the difficulty of isolating the analog signal that is generated across the current shunt from the high voltage common signal that is generated by the switching power inverter. It accomplishes this by converting the signal using isolated Σ - Δ modulators and then transmitting a digital signal across the isolation barrier.

The Σ - Δ modulators generate a modulated bit stream as a function of the input voltage and transmit the signal across the isolation barrier to a filter circuit on the low voltage side. The SINC filter filters the bit stream from a second order modulator, such as the [AD7401A](#), to recover a 16-bit digital signal that represents the motor winding current.

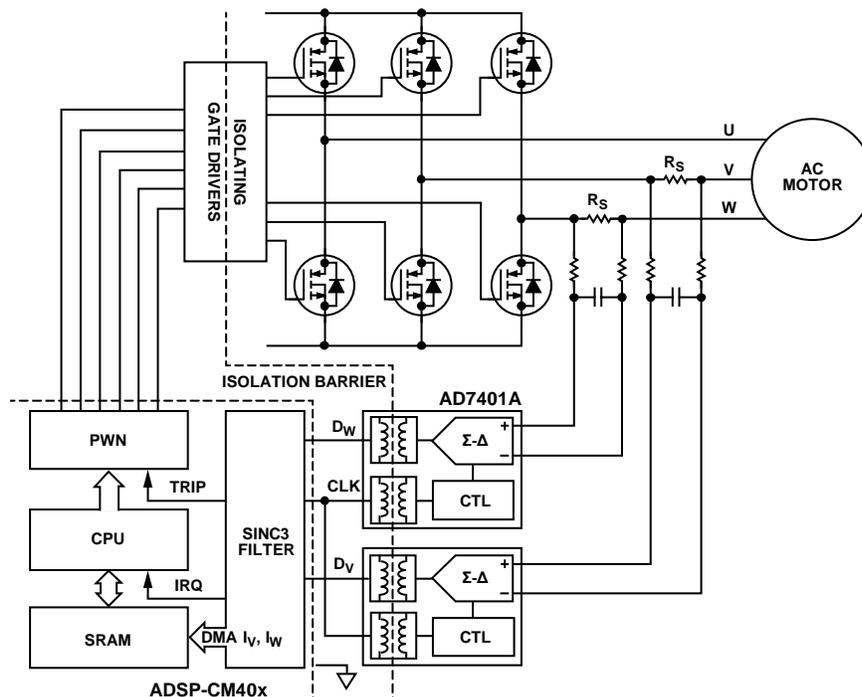


Figure 1. Isolated Current Feedback Using the [AD7401A](#)

11801-001

TABLE OF CONTENTS

Introduction	1	Secondary Filter Scaling and Overload Configuration.....	7
Motor Current Control Applications	1	SINC Module Fault Detection Functions	9
Revision History	2	SINC Filter Setup and Software Driver Functions.....	10
Sinc Filter Module Overview	3	Pin Multiplexer Configuration	10
Current Feedback System Overview.....	4	Data Buffer Memory Allocation	10
Current Shunt Selection	4	Interrupt and Trigger Routing.....	11
Modulator Clock, Primary Filter Decimation, and Data		Primary and Secondary Filter Configurations.....	12
Interrupt Rate Selection.....	5	SINC Filter Software Support.....	13
Primary Filter Scaling	7		

REVISION HISTORY

11/13—Rev. 0 to Rev. A

Changes to Figure 1	1
Changes to Figure 4.....	4
Changes to Table 1	5

9/13—Revision 0: Initial Version

SINC FILTER MODULE OVERVIEW

The SINC filter block performs two functions: it generates a high fidelity feedback signal for the motor control algorithm, and it provides rapid detection of overload currents in the case of fault conditions. Connecting the overload fault signal to the PWM modulator block can shut down the PWM inverter without any software intervention. The SINC filter transfers data directly to memory using DMA, and a processor interrupt can be generated when a preset number of samples is ready. This minimizes the software overhead to service the SINC filter after it is configured. The same feedback circuit applies to isolated dc bus voltage feedback or dc bus current measurements.

Figure 2 shows a block diagram of the SINC filter module. The SINC filter module has four SINC filter pairs that implement feedback signal filtering and overload detection on the digital bit streams connected to the inputs. The filter enable function

assigns SINC filter pairs to one of two configuration register groups to set the filter parameters. The expectation is that the motor drive requires multiple current or voltage filters configured in the same way. The SINC filter module supports control of two motors with one group of two filter pairs assigned to each motor. The primary filter settings are the filter order, decimation rate, offset bias, and gain scaling. The secondary filter settings are the filter order, decimation rate, overload trip levels, and glitch filter settings.

Other configuration functions include modulator clock frequencies, interrupt masking, and DMA data transfer. The other control peripherals required to set up the SINC filter are the port controller that connects external pins to the SINC filter inputs and the trigger routing unit (TRU) that connects SINC output signals to the appropriate peripheral.

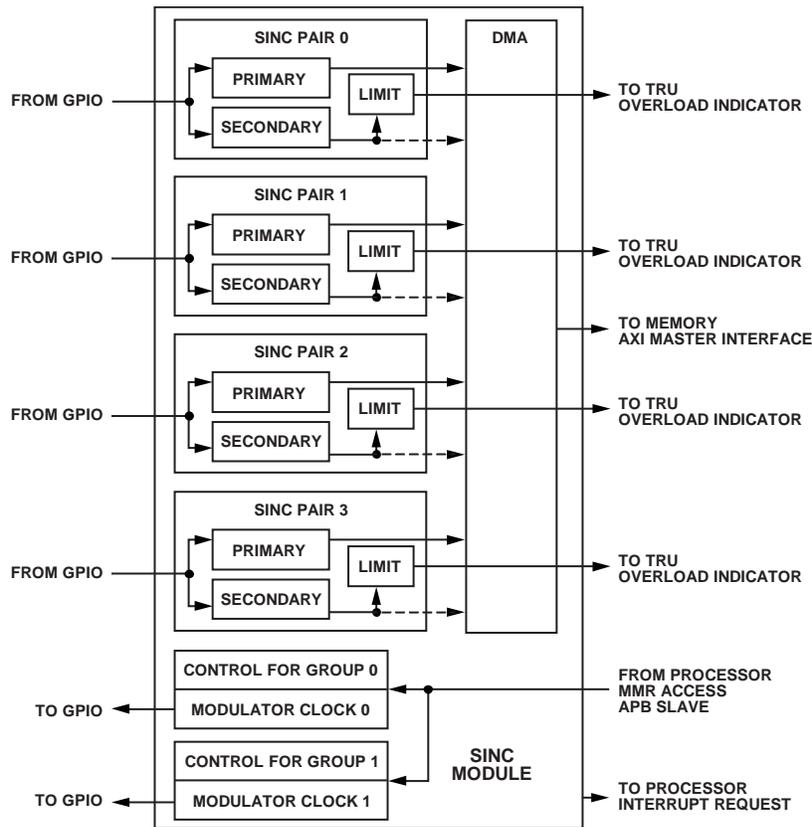


Figure 2. SINC Filter Module Overview

11881-002

CURRENT FEEDBACK SYSTEM OVERVIEW

Figure 4 describes the key elements in the current feedback system. The shunt senses the winding current as a voltage signal that scales according to the shunt resistance. The AD7401A modulator generates an isolated bit stream with a pulse density that scales according to the full-scale input voltage range. The SINC filters extract the pulse density information according to the filter order and decimation rate. The primary filter parameters optimize the filter for precision and additional bias and scaling blocks convert the data into a 16-bit signed integer before it is transferred to memory. The secondary parameters optimize the filter for speed, and the outputs pass the signal to digital comparators that detect overload conditions. Upper and lower limit comparators detect current overloads, and a glitch filter waits for a minimum overload count (LCNT) within a specific window (LWIN) before generating an overload trigger signal. The overload trigger is a trip input signal for the PWM modulator driving the motor inverter. The DMA transfer engine generates an interrupt signal to initiate algorithm execution when the winding current data is ready in memory.

CURRENT SHUNT SELECTION

The system specifications required to define the feedback are the peak control current, $I_{cc}(p)$, and the specified maximum input voltage, $V_{mod}(max)$, for the modulator. The peak current capability of the power inverter typically defines the control current range, but other considerations may apply. The specified maximum operating voltage of the AD7401A modulator is ± 200 mV, which

is the maximum voltage range within which the modulator specifications are valid. This is lower than the ± 320 mV full-scale range (V_{FS}) of the modulator because the linearity and signal-to-noise performance degrades significantly as you get close to the full-scale inputs. The shunt resistance must be less than $V_{mod}(p)/I_{cc}(p)$ to satisfy these constraints, and the closest nominal shunt value is chosen. For the example in Figure 3, given that the power stage peak current rating is 8.5 A, the maximum shunt resistance is 23.5 m Ω . The closest nominal shunt value is 20 m Ω , which yields a specified maximum current of 10 A.

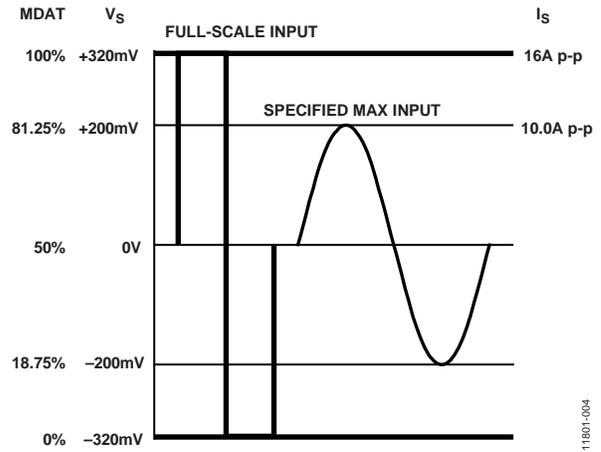


Figure 3. Feedback Current Operating Ranges

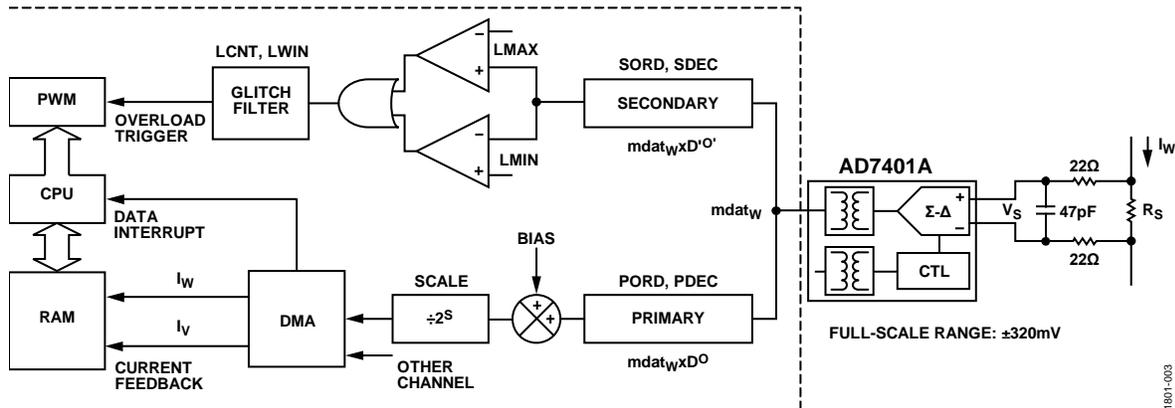


Figure 4. SINC Filter Current Feedback Paths

MODULATOR CLOCK, PRIMARY FILTER DECIMATION, AND DATA INTERRUPT RATE SELECTION

The modulator clock (f_M) and decimation rate (D) are the parameters that define the SINC filter performance. The filter order (O) is typically one order higher than that of the front-end modulator. Therefore, when the AD7401A is used, the filter order is 3. The equations for the filter frequency response and group delay follow. The frequency response shown in Figure 5 has 0s at frequencies that are even multiples of the decimation frequency (f_M/D). Therefore, matching of the decimation frequency to the PWM switching frequency substantially reduces PWM switching harmonics. Other considerations include the increase in the group delay with decimation rate and the maximum decimation limit of the filter.

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

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

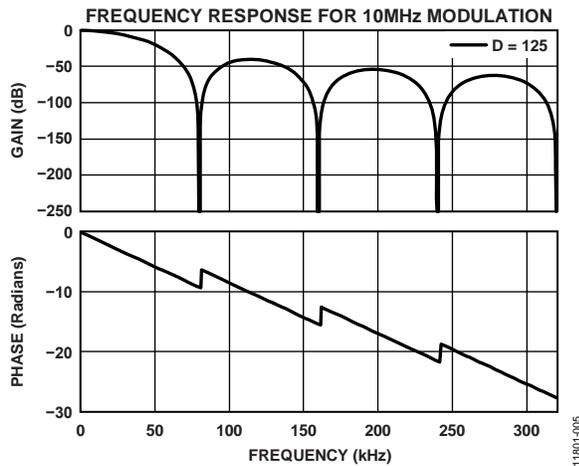


Figure 5. SINC Filter Frequency Response

For a given filter order, the decimation rate and filter order are the filter parameters that define the filter signal-to-noise ratio (SNR) and group delay. Figure 6 and Table 1 show the variation of the SNR, effective number of bits (ENOB), and group delay vs. the decimation rate for a third order filter with a 10 MHz modulator clock. The decimation rate must be in the range of 85 to 210 to achieve an ENOB of 11 bits to 14 bits (and an SNR of 67 dB to 86 dB), which is the filter performance range required for current feedback. The group delay is between 12 μ s and 32 μ s in this decimation rate range.

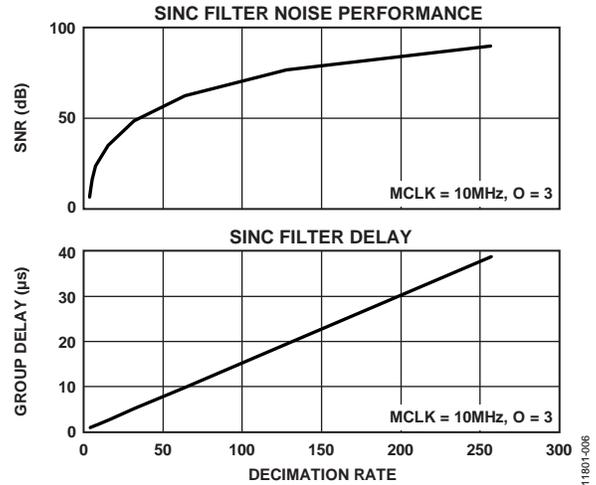


Figure 6. Secondary Filter SNR

Table 1. SNR, ENOB, and Group Delay with Decimation Rate¹

Decimation Rate	SNR (dB)	ENOB (Bits)	Group Delay (μ s)
85	68	11	12.6
113	74	12	16.8
154	80	13	23.0
210	86	14	31.4

¹The test condition is a ± 200 mV sinewave at 1.22 kHz.

It is not possible to match the SINC filter decimation rate with the typical PWM switching frequencies used in motor drives. Matching a switching frequency of 16 kHz would require a decimation rate of 625, and the resultant filter group delay would be 94 μ s. This decimation rate is well above available values, and the group delay would limit the bandwidth of the current loop. Instead, the decimation rate is set to a multiple of the PWM frequency to lower the group delay and still achieve the target filter SNR. The control algorithm samples the data at a submultiple of the decimation frequency matching the PWM switching. This software decimation process involves transferring multiple data samples to a circular buffer in memory and reading the most recent data sample in response to the interrupt generated when the buffer is full. The DMA engine transfers data from the primary SINC filter to data memory, and the SINC control unit generates a trigger every time it transfers a fixed number of samples.

Figure 7 describes the alignment between PWM switching, modulator, decimation, and data sampling. The synchronizing pulse (PWM0_SYNC) from the PWM modulator aligns the startup of the modulator clock with the PWM frequency. The decimation frequency is a submultiple of the modulator clock and a multiple of the PWM frequency. The SINC0_DAT0 trigger rate is at the PWM frequency.

The information in Table 2 illustrates the process of selecting the decimation rate and the PWM switching frequency. The first three entries in the table are chip level settings for the core and peripheral clocks. The maximum core clock rate is 240 MHz, and it is typically an even multiple of the system (peripheral) clock frequency. The SINC filter modulator clock (MCLK) derives from the system clock based on the MDIV register field value, and there are a limited set of values in the 5 MHz to 20 MHz range. The primary decimation rate (PDEC) is 125, which sets the filter SNR at 76 dB (>12-bit ENOB) with a group delay of 18.6 μs. The delay corresponds to a phase lag of only 8° at a typical current control loop bandwidth of 1.25 kHz. The modulator clock is 10 MHz; therefore, the primary decimation clock frequency is 80 kHz, and a software decimation rate (SWDEC) of 5 synchronizes the sample rate with a 16 kHz PWM frequency (PWM). Tuning of the PWM frequency is possible by adjusting the SINC filter decimation rate.

The equation governing the relationship among the modulator clock, the PWM frequency, and the hardware and software decimation rates is

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

where:

PDEC is the hardware decimation rate.
SWDEC is the software decimation rate.

The hardware and software decimation rates must be integers. The PCNT register field value in the SINC filter sets the software decimation rate. The PCNT value loaded in the SINC filter control register is one less than the number of sample delays before an interrupt is generated. The PWM_TM0 register sets the PWM switching frequency and, therefore, sets the sample timing.

Table 2. Decimation Rate Selection

Parameter	Symbol	Value	Unit
Core Clock	CCLK	240	MHz
System Clock Divider	SYSEL	3	
System Clock	SYSCLK	80	MHz
Modulator Clock Divider	MDIV	8	
Modulator Clock (1/T _M)	MCLK	10	MHz
Decimation Rate	PDEC	125	
Filter SNR	SNR	76.0	dB
Filter ENOB	ENOB	12.3	Bits
Decimation Frequency	DCLK	80.0	kHz
Filter Group Delay	τ _d	18.6	μs
Software Decimation Rate	SWDEC	5	
Data Transfer Count	PCNT	4	
PWM Frequency (1/T _s)	PWM	16.00	kHz
PWM Period Count	PWMTM	2500	

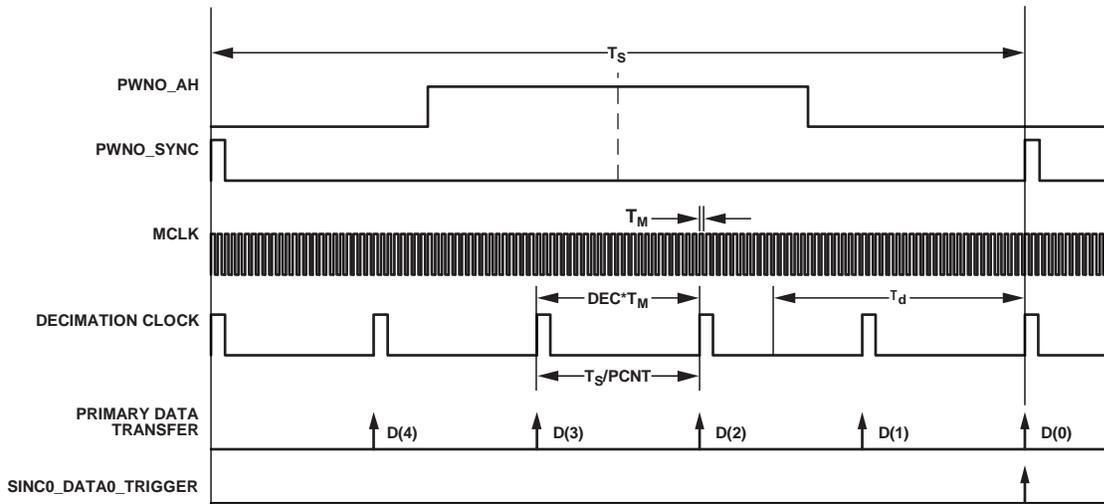


Figure 7. Modulator and Decimation Clock Timing

11801-007

PRIMARY FILTER SCALING

The SINC filter order (O) and decimation rate (D) set the primary filter dc gain, given by

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

The unit has output scaling and bias functions to convert the data to a 16-bit signed integer before it is transferred to memory. The data format is valid as a fractional 16-bit integer (S.15) in the range ±1.0 or as a signed 16-bit integer in the range ±2¹⁵, depending on interpretation.

The raw filter output is an integer between 0 and D^O, where D^O/2 aligns with a 50% pulse density corresponding to 0 A. Adding a bias value of -D^O/2 to the output sets the correct zero level. Dividing the result by D^O/2 would scale the full-scale fractional integer output to ±1. However, for simplicity, the unit has a simple binary scale factor where the user selects S to set the gain near 1.0. Regardless of the scaling, the DMA engine only transfers the 16 least significant bits of the output data; therefore, correct scaling is essential to avoid loss of precision. The output data is saturated to prevent data overflow, which could invert the polarity of the output signal due to incorrect scale factor selection. The filter sets an overflow fault flag when saturation occurs.

Conversion of the data to a floating point simply involves scaling by the inverse of the current shunt gain and adjusting for the mismatch between the filter dc gain and the scale factor.

Feedback Scaling Calculations

The final system gain from the shunt current to the data-word in memory derives from the gains of all the elements in the system as shown in Figure 8. The isolated modulator in this example is the AD7401A.

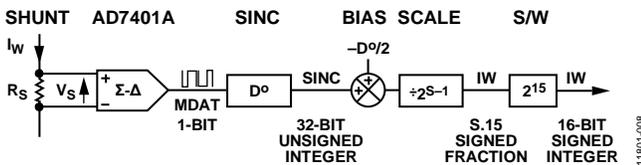


Figure 8. SINC Primary Output Data Scaling

The shunt voltage seen by the modulator is

$$v_s = i_w \times R_s$$

The isolated modulator expects a bipolar input and generates a 50% pulse density for a 0 V input. The pulse density is a function of the ratio of the input voltage (v_s) to the positive full-scale input (V_{FS}):

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

In the case of the AD7401A, the positive full-scale voltage is 320 mV, and the ones density is 81.25% for the specified maximum voltage of 200 mV.

The SINC filter dc gain is D^O; therefore, the raw output as a function of the input voltage is

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

This dc scaling applies to the secondary filter outputs, and the maximum secondary decimation rate restricts the raw output data range to a 16-bit unsigned integer. The secondary output is 0 at the negative full-scale input and D^O at the positive full-scale input.

The bias and scale functions in the primary output path remove the bias on the SINC data and rescale the data to a 16-bit signed integer. The bias value must be -D^O/2 to eliminate the offset in the SINC output for a modulator with a bipolar input range. The rescaling selects the appropriate bit range from the SINC output word.

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

The scale factor (S) must set the maximum fractional integer output at 1.0, which is true when

$$\frac{D^O}{2^S} \ll 1 \therefore S \gg \frac{\ln(D^O)}{\ln 2}$$

The SINC output equation when reading the data as a signed integer adds a scale factor of 2¹⁵.

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

The current reading as a function of the actual winding current (i_w) in this case is

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

SECONDARY FILTER SCALING AND OVERLOAD CONFIGURATION

The secondary SINC filter data outputs connect directly to overload comparators and a glitch filter as shown in Figure 4. The secondary filter decimation rate is set significantly lower than that of the primary filter to achieve fast response to fault conditions. The processor trigger routing unit (TRU) connects the overload trip signal to the PWM modulator shutdown input to clear the fault. The TRU can also connect the overload signal to other sources, such as an external GPIO used to shut down other critical circuit elements.

Typical power inverter switches can withstand a short circuit for a few microseconds; therefore, the overload circuit must have a relatively short detection window. Because the SINC filter can respond to a step input within three decimation cycles, a response within 3 μs is possible using a decimation rate of 10, as shown in Figure 9. The SINC filter also filters out inverter switching noise, as shown in Figure 10. In this figure, a 10 A peak test

waveform has injected 16 A noise pulses of 1.5 μ s duration and 16 A overload pulses of 40 μ s duration. The filter rejects the short noise pulses, but the circuit detects the 16 A overload pulses. The maximum and minimum trip levels in this test are at secondary SINC outputs corresponding to ± 16 A.

A faster response is possible at a lower decimation rate, but as seen from Figure 11, the secondary SINC output exceeds the trip levels even for a simple sinusoidal test current of ± 10 A. The higher SINC filter noise at a decimation rate of 5 generates multiple false trip signals. Figure 12 illustrates the SNR at high (10) and low (5) decimation rates and the noise margin for the trip signal.

The secondary output glitch filter rejects short overload trips by eliminating trips with durations less than a minimum count (LCNT) with a trip count window (WCNT). Figure 13 illustrates how the glitch filter eliminates the spurious overload that is triggered when the decimation rate is 5; however, there is an additional three cycle delay in the response time. Therefore, there is no reduction in response time from the lower decimation rate. The figure illustrates the ability of the filter to reject short noise pulses on the analog input. In this example, the noise pulse is 1.5 μ s in duration.

The secondary SINC filter includes a set of history buffers that capture the eight most recent data samples before a trip is generated for diagnostic purposes. The data in the history registers are accessed directly through the device peripheral memory infrastructure.

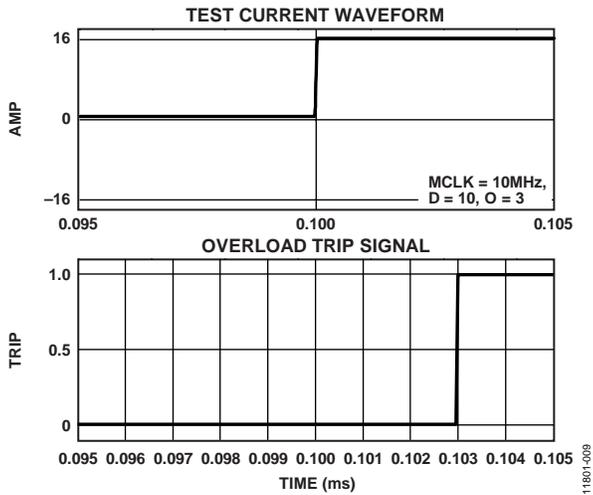


Figure 9. Secondary Filter Overload Detection

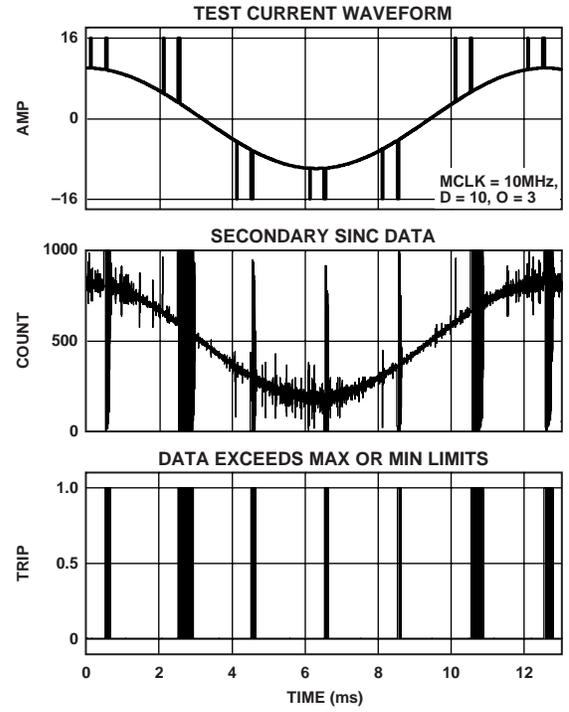


Figure 10. Overload Detection with Decimation Rate of 10

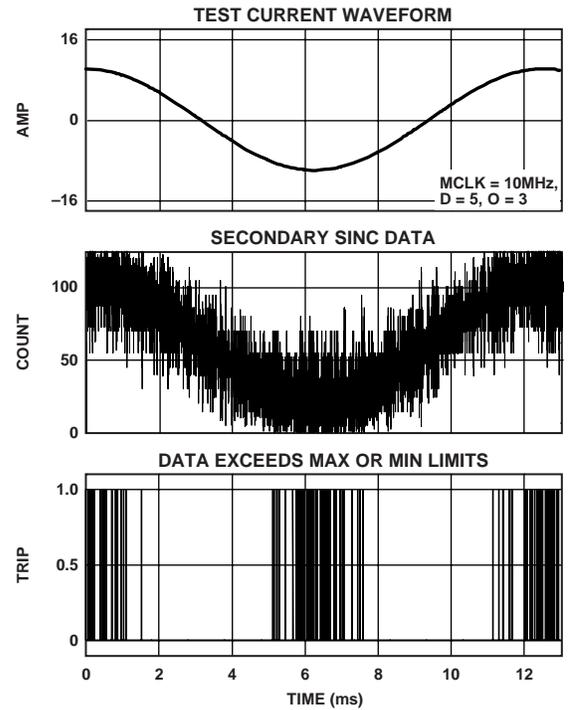


Figure 11. False Overloads Detected with Decimation Rate of 5

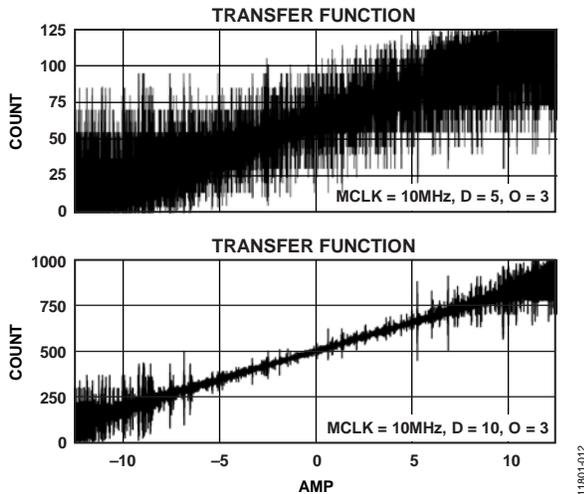


Figure 12. Secondary Filter Gain Curves for Decimations of 5 and 10

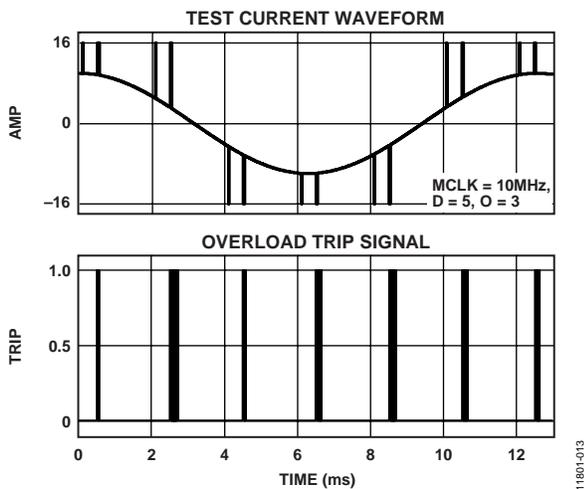


Figure 13. Overload Detection with Decimation Rate of 5 and Glitch Filter with WCNT = 4 and LCNT = 4

Secondary Filter Scaling and Trip Level

There is no extra output scaling on the secondary filters; therefore, valid minimum and maximum values are within the range of 0 to D^O . The negative full-scale current maps to 0, and the positive full-scale current maps to D^O . Setting the minimum and maximum trip levels to 1 and $D^O - 1$ enables the maximum range of the trip function. The transfer function shown in the bottom graph of Figure 12 for a decimation rate of 10 and a 20 mΩ shunt shows that the noise peaks for a 10 A input are within the maximum (1000 counts) and minimum (0 counts) outputs of the filter. Set the LMIN and LMAX trip levels to 1 count and 999 counts to avoid spurious trips for 10 A peak current. The actual current level at which the trip is triggered ranges between 11 A and the full scale of 16 A. The likelihood of a trip increases the closer the current gets to the full-scale limits.

The overload circuit operates a little more precisely within the specified modulator input range. For the previous case, the peak noise at 5 A input is 700 counts, which is equivalent to 6.4 A. Therefore, the trip could be set to operate within the range of 5 A to 6.4 A. The LMAX and LMIN settings in this case would be 700 counts and 300 counts. Attaining precise trip settings using lower decimation rates is more difficult.

SINC MODULE FAULT DETECTION FUNCTIONS

The Secondary Filter Scaling and Overload Configuration section describes the selection of the various filter parameter settings required to achieve the desired primary and secondary filter performance. In addition to overload faults, the SINC module checks for data faults that can arise from incorrect filter settings overloading the chip infrastructure.

The primary filter detects output data saturation when there is an incorrect setting of the output bias and scaling. The filter DMA engine detects a FIFO error if it fails to transfer data before the filter writes new data. The ESATx and EFOVfx bits in the SINC_CTL register mask the SINC0_STAT interrupt generation on saturation and FIFO faults.

SINC FILTER SETUP AND SOFTWARE DRIVER FUNCTIONS

There are several steps to set up the SINC filter module as well as the signal routing and data buffers before the filter is ready for use. Once configured, the DMA engine automatically streams primary filter data to memory, and the secondary limit function shuts down the PWM module in the case of an overload. The system generates an interrupt when data is ready; therefore, the processor can execute the control algorithm and update the PWM modulator registers. Figure 14 outlines the interconnections required between the SINC filter block and the CPU, SRAM, PWM, and external pins to capture motor current feedback signals.

There are four steps to set up current feedback using the SINC filter:

1. Configure the pin multiplexer.
2. Allocate the data buffer memory.
3. Connect the interrupt and trigger routing.
4. Configure the primary and secondary filters.

This section further describes these steps, detailing the setup process and the Analog Devices device driver functions to program the system and SINC filter control registers.

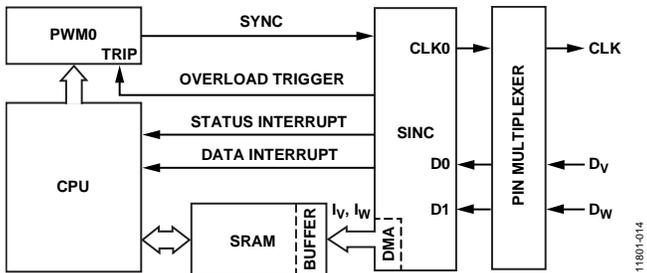


Figure 14. SINC Filter System Configuration

PIN MULTIPLEXER CONFIGURATION

The pin multiplexer connects the front-end modulator clock and data pins to the SINC module. There are two available modulator clock outputs (SINC0_CLK0 and SINC0_CLK1) and four available SINC data input pins (SINC0_D0, SINC0_D1, SINC0_D2, and SINC0_D3). The PORT_MUX registers control the selection of these pins from four alternate input or output signals for each of the multiplexed pins. The PinMux64.jar and PinMux32.jar Java application programs, which are supplied with the [ADSP-CM40x Enablement Software package](#), automatically generate C code to enable the user port selections. Figure 15 is a snapshot of the PinMux64.jar application window.

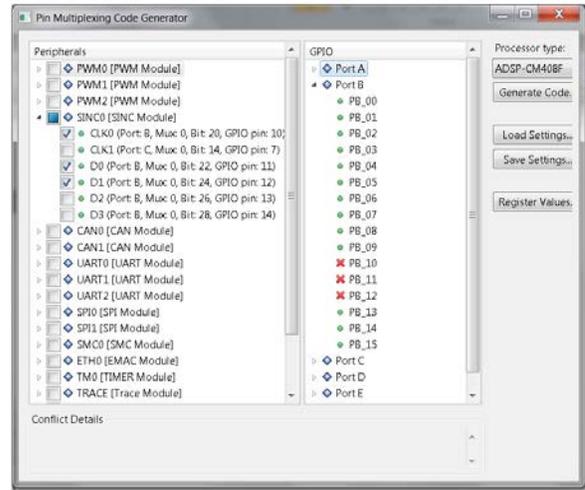


Figure 15. Pin Mux Selection Tool

DATA BUFFER MEMORY ALLOCATION

The primary filter data buffers must be defined and assigned memory space to allow the control algorithm to use the data. The software decimation rate and the number of feedback channels define the buffer size. The data is ordered on a per group basis in channel sequence. The pointer to the most recent data set is stored in the SINC_PPTRx register. Device drivers described in the Interrupt and Trigger Routing section handle the buffer and pointer management.

BUFFER ADDRESS	BUFFER DATA
SINC_PHEADx	SINC_OUTx_0[3]
	SINC_OUTx_1[3]
SINC_PPTRx	SINC_OUTx_0[0]
	SINC_OUTx_1[0]
	SINC_OUTx_0[1]
	SINC_OUTx_1[1]
	SINC_OUTx_0[2]
SINC_TAILx	SINC_OUTx_1[2]

Figure 16. Data Buffer Organization

INTERRUPT AND TRIGGER ROUTING

Figure 17 describes the SINC filter interconnection with other peripheral functions using interrupt and trigger signals. The SINC_STAT is the single processor interrupt signal of the SINC filter module. The trigger routing unit (TRU) connects the other trigger signals to the peripherals and processor interrupts of the SINC filter module. Loading the trigger master address into the trigger slave registers in the TRU connects the routing.

The TRU synchronizes the SINC filter modulator and decimation clocks with the PWM modulator frequency to meet the timing defined in Figure 7. The TRU connects the SINC filter data transfer

trigger to the control software interrupt to start execution of the control algorithm.

The TRU connects both of the SINC overload triggers to the PWM modulator TRIP1 input to enable overcurrent protection. The TRIP0 input connects to the external trip signal only. The PWM modulator as well as TRIP0 and TRIP1 inputs must be configured to accept these triggers. There are two interrupt triggers produced by an overload fault: the SINC_STAT interrupt connected directly to the CPU and the PWM_TRIP1 interrupt generated by the SINC overload trigger. The code segment at the end of the SINC Filter Software Support section includes the device driver calls to implement these connections.

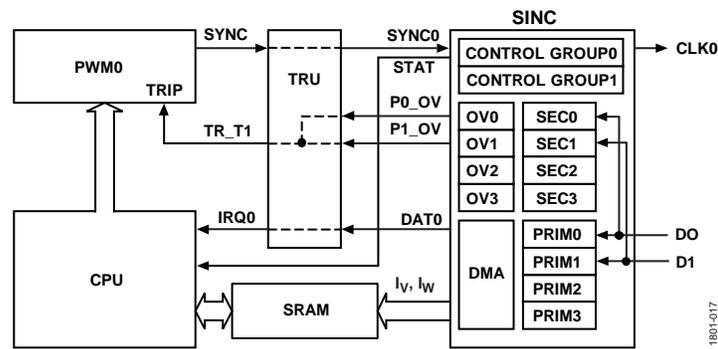


Figure 17. SINC Filter Trigger Routing

11801-017

PRIMARY AND SECONDARY FILTER CONFIGURATIONS

Filter channels are organized in groups because it is typical for two or three feedback signals to need the same filter parameters. The SINC module has two groups of configuration registers. The channels in any one group share the same clock and have common filter parameters, such as filter order, decimation rate, scaling, and bias. The exception is the overload limit and history registers, which have a per channel organization. Enabling a filter channel assigns it to a configuration group. The configuration registers define the modulator clocks, filter parameters, DMA data transfer, and overload detection.

Figure 18 describes the assignment of filter and system parameters to Group0 registers. The Group1 registers organization is the same. The SINC_CTL register enables each channel and assigns the control group. The recommended process is to configure the filter group before enabling the channels in the group. The SINC_CTL register also masks the SINC_STAT interrupt. The system status register, SINC_STAT, reports the fault and data trigger count status.

Three registers per group and the clock register define the primary and secondary filter parameters. SINC_RATE0 and SINC_RATE1

set the primary and secondary filter decimation rates (PDEC, SDEC) and the primary filter phase (normally 0°). SINC_LEVEL0 and SINC_LEVEL1 define the primary and secondary filter order (PORD, SORD) and the primary filter scale (PSCALE). BIAS0 and BIAS1 define the primary filter data offset. The SINC_CLK register defines the CLK0 and CLK1 modulator clock frequency and can enable synchronization with an external trigger. This register also includes a means to adjust the clock phase if required.

Three registers per group support the primary DMA channels. SINC_PHEAD0 and SINC_TAIL0 define the memory addresses for the Group0 primary output data buffer. The SINC_PPTR0 register stores the pointer to the most recent data in the buffer. The PCNT bits in the SINC_LEVEL0 register set the software decimation rate by defining the number of data transfers per data interrupt (PCNT + 1).

Five registers per channel support the secondary overload detection function. SEC_LIMIT0 defines the maximum and minimum overload threshold, and POSEC_HIST0, POSEC_HIST1, POSEC_HIST2, and POSEC_HIST3 store the last eight secondary filter outputs before an overload trip. The SINC_LEVEL0 and SINC_LEVEL1 registers set the secondary filter glitch parameters (LWIN, LCNT) for the channels in the associated group.

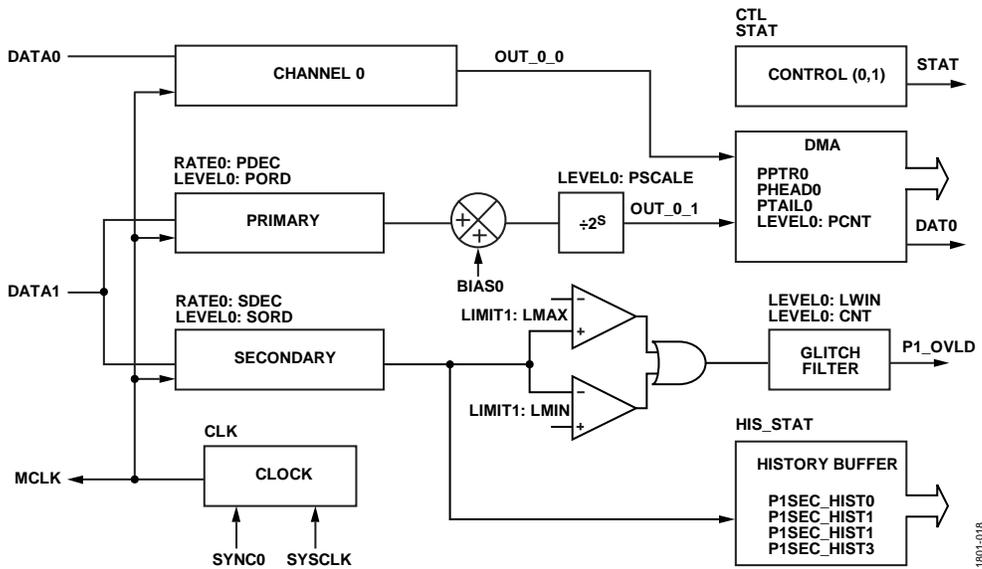


Figure 18. SINC Register Mapping

SINC FILTER SOFTWARE SUPPORT

The code segment that follows is an example of how to set up the primary and secondary filters for two channels of current feedback. These code snippets are extracts from working code tested on a closed-loop motor control evaluation platform. The device driver adds some overhead but significantly simplifies the programming of the SINC module registers. The function call constant names match the parameter names used in this document; therefore, most of the code should be self-explanatory.

The first block of code (Lines[1:21]) defines a number of parameter constants. Line 9 and Line 10 define the size of the data buffers for the primary SINC data. The next block of code (Lines[22:36]) sets up prototype functions and allocates memory. The SINC callback functions defined on Line 28 and Line 29 handle the SINC_DATA0 and SINC_STAT interrupts.

The SetupTRU code block (Lines[37:46]) includes all the trigger routing. The SetupPWM code block (Lines[47:74]) includes the PWM timer frequency, sync pulse, and trip function setup. The external hardware trip connects to TRIP0, and the internal SINC_Px_OVLD triggers connect to TRIP1. The TRIP1 interrupt is one of the interrupts generated by a SINC overload. The overload also generates a SINC_STAT interrupt.

The SetupSINC code block (Lines[75:106]) configures the SINC filter parameters. Lines[78:80] open the device driver and set up the callback functions. Lines[81:85] set up various group parameters, including order and decimation rate. Lines[87:89] control the initial setting of the overload limits to their full range to avoid a spurious trip when the filter starts. Line 86 sets up the circular buffer for the primary SINC data, and Line 94 and Line 95 assign data channels to the buffers. Line 91 and Line 92 set up the modulator clock. On Line 91, the driver calculates the divide ratio from the system clock and modulator frequencies. On Line 92, the driver call enables the clock and sets the synchronization mode. Line 97 enables the SINC_STAT interrupt masks. Line 98 and Line 99 enable the Filter Channel 0 and Filter Channel 1 that are assigned to Group0. Line 100 and Line 101 introduce a short delay before specifying the SINC interrupt masks on Line 103 and setting the working secondary filter overload limits on Line 104 and Line 105.

The final block of code (Lines[106:129]) includes the callback functions for the SINC data and overload interrupt. The SincDataCallback function copies data from the buffer to the motor control variables and calls the control function. SincStatusCallback calls the fault handler routines.

```

1. /*****
2. SINC FILTER SETUP CODE SNIPPETS
3. *****/
4. /**** Include file #define code ****/
5. /* SINC definitions */
6. #define SINC_DEV                0
7. #define SINC_NUM_SAMPLES        4
   /* this determines how often a data
   interrupt is generated */
8. #define SINC_NUM_PAIRS          2
9. #define SINC_DATA_SIZE
   (SINC_NUM_PAIRS * SINC_NUM_SAMPLES)
10. #define CIRC_BUF_SIZE
   (SINC_NUM_SAMPLES*20)
   /* size for the device circular buffer */
11. #define SINC_MODCLK              (5000000)
   /* modulator clock freq */
12. #define PDEC                    125
   /* primary decimation */
13. #define PSCALE                   24
   /* Primary scale */
14. #define SDEC                     5
   /* secondary decimation */
15. #define LWIN                     4
   /* Glitch window */
16. #define LCNT                     4
   /* Glitch count */
17. #define LMAX                     124
   /* Overload max limit */
18. #define LMIN                     1
   /* Overload min limit */
19. /* TRU definitions*/
20. #define TRU_DEV_NUM              0
21. #define ADI_TRU_REQ_MEMORY        4u

22. /* SINC related P R O T O T Y P E S and
   memory allocation */
23. /* Function prototypes */
24. void SetupPWM(void);
25. void SetupTRU(void);
26. void SetupSINC(void);
27. /* Prototype for callback functions */
28. static void SincDataCallback(void
   *pHandle, uint32_t event, void *pArg);
29. static void SincStatusCallback(void*
   pHandle, uint32_t event, void* pArg);
30. /* SINC handler and data buffers */
31. static uint8_t
   SincMemory[ADI_SINC_MEMORY_SIZE];
32. static ADI_SINC_HANDLE hSINC;
33. static int16_t sincData1[SINC_DATA_SIZE];
34. static int16_t sincData2[SINC_DATA_SIZE];

35. static int16_t
   sincCircBuffer[CIRC_BUF_SIZE];
36. /*****
37. void SetupTRU(void){
38. /***** Function: SetupTRU (code
   snippet for SINC related setup) *****/
39. ADI_TRU_RESULT result;
40. result = adi_tru_Open (TRU_DEV_NUM,
   &TruDevMemory[0], ADI_TRU_REQ_MEMORY,
   &hTru);
41. // Setup TRU for SINC. Slave is SINC0
   SYNC0, master is PWM0 sync pulse
42. result = adi_tru_TriggerRoute (hTru,
   TRGS_SINC0_SYNC0, TRGM_PWM0_SYNC); //
   TRU device, slave, master
43. result = adi_tru_TriggerRoute (hTru,
   TRGS_PWM0_TRIP_TRIG1,
   TRGM_SINC0_P0_OVLD); /* connect
   SINC_Px_OVLD trigger to PWM0_TRIP_TRIG1.
   slave, master */
44. result = adi_tru_TriggerRoute (hTru,
   TRGS_PWM0_TRIP_TRIG1,
   TRGM_SINC0_P1_OVLD); /* Both overload
   detection on TRIP1. TRIP0 used by HW */
45. result = adi_tru_Enable (hTru, true); //
   Enable TRU
46. }

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

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

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

54. temp = (uint32_t)(fsysclk *
   SYNC_PULSE_WIDTH); // Calculate
   sync pulse width as number of sys clocks
   (up-down counter)
55. result = adi_pwm_SetSyncWidth(hPWM,
   temp);

56. result = adi_pwm_ExtSyncEnable(hPWM,
   false, false); // Internal sync
   used

```

```

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

58.result = adi_pwm_SetTripEnable(hPWM,
    ADI_PWM_CHANNEL_A, ADI_PWM_TRIP0_SRC,
    true); // Enable Trip0 and trip on all
    channels

59.result = adi_pwm_SetTripEnable(hPWM,
    ADI_PWM_CHANNEL_B, ADI_PWM_TRIP0_SRC,
    true);

60.result = adi_pwm_SetTripEnable(hPWM,
    ADI_PWM_CHANNEL_C, ADI_PWM_TRIP0_SRC,
    true);

61.result = adi_pwm_SetTripEnable(hPWM,
    ADI_PWM_CHANNEL_A, ADI_PWM_TRIP1_SRC,
    true); // Enable Trip1 and trip on all
    channels

62.result = adi_pwm_SetTripEnable(hPWM,
    ADI_PWM_CHANNEL_B, ADI_PWM_TRIP1_SRC,
    true);

63.result = adi_pwm_SetTripEnable(hPWM,
    ADI_PWM_CHANNEL_C, ADI_PWM_TRIP1_SRC,
    true);

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

65.result = adi_pwm_SetTripMode(hPWM,
    ADI_PWM_CHANNEL_B, ADI_PWM_TRIP0_SRC,
    false);

66.result = adi_pwm_SetTripMode(hPWM,
    ADI_PWM_CHANNEL_C, ADI_PWM_TRIP0_SRC,
    false);

67.result = adi_pwm_SetTripMode(hPWM,
    ADI_PWM_CHANNEL_A, ADI_PWM_TRIP1_SRC,
    false);

68.result = adi_pwm_SetTripMode(hPWM,
    ADI_PWM_CHANNEL_B, ADI_PWM_TRIP1_SRC,
    false);

69.result = adi_pwm_SetTripMode(hPWM,
    ADI_PWM_CHANNEL_C, ADI_PWM_TRIP1_SRC,
    false);

70.result = adi_pwm_InterruptEnable(hPWM,
    ADI_PWM_INTERRUPT_TIMER0, true);
    // Enable sync irq

71.result = adi_pwm_InterruptEnable(hPWM,
    ADI_PWM_INTERRUPT_TRIP0, true);
    // Enable trip0 irq

72.result = adi_pwm_InterruptEnable(hPWM,
    ADI_PWM_INTERRUPT_TRIP1, true);
    // Enable trip1 irq

73. /** other PWM setup code **/

74. }

75. void SetupSINC(void){
76. /**          Function: SetupSINC
    *****/
77. static ADI_SINC_RESULT result;

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

79. result = adi_sinc_RegisterDataCallback
    (hSINC, SincDataCallback, 0);

80. result = adi_sinc_RegisterStatusCallback
    (hSINC, SincStatusCallback, 0);

81. /* Specify Group Parameters */
82. result = adi_sinc_SetRateControl (hSINC,
    ADI_SINC_GROUP_0,
    ADI_SINC_FILTER_PRIMARY, PDEC, 0);

83. result = adi_sinc_SetRateControl (hSINC,
    ADI_SINC_GROUP_0,
    ADI_SINC_FILTER_SECONDARY, SDEC, 0);

84. result = adi_sinc_SetLevelControl (hSINC,
    ADI_SINC_GROUP_0, LWIN, LCNT,
    SINC_NUM_SAMPLES, PSCALE);

85. result = adi_sinc_SetFilterOrder (hSINC,
    ADI_SINC_GROUP_0,
    ADI_SINC_FILTER_THIRD_ORDER,
    ADI_SINC_FILTER_THIRD_ORDER);

86. result = adi_sinc_SetCircBuffer(hSINC,
    ADI_SINC_GROUP_0, sincCircBuffer,
    CIRC_BUF_SIZE);

87. /* Reset overload amplitude detection
    limits to 0 - FullScale */
88. result = adi_sinc_SetAmplitudeLimit
    (hSINC, ADI_SINC_PAIR_0, 0x0000, 0xFFFF);
89. result = adi_sinc_SetAmplitudeLimit
    (hSINC, ADI_SINC_PAIR_1, 0x0000, 0xFFFF);

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

92. result = adi_sinc_EnableModClock (hSINC,
    ADI_SINC_GROUP_0,
    ADI_SINC_MOD_CLK_PWM_SYNC);

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

```

```

95. result = adi_sinc_SubmitBuffer(hSINC,
    ADI_SINC_GROUP_0, sincData2, 16);

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

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

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

102. /* Enable & assign used SINC filter
    pair, and specify interrupt masks */
103. 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);

104. result = adi_sinc_SetAmplitudeLimit
    (hSINC, ADI_SINC_PAIR_0, LMIN, LMAX);
105. result = adi_sinc_SetAmplitudeLimit
    (hSINC, ADI_SINC_PAIR_1, LMIN, LMAX);
106.  }

107.  /**          Function:
    SincDataCallback          ****/
108.  static void SincDataCallback(void*
    pHandle, uint32_t event, void* pArg){
109.  static uint16_t *bufferPtr;
110.  bufferPtr = (uint16_t*)pArg; /*
    pointer to sincData1 or sincData2 */
111.  switch((ADI_SINC_EVENT)event){
112.  case ADI_SINC_EVENT_DATA0:
113.  Mctrl_U.ibc_sinc[1] = *bufferPtr;
114.  Mctrl_U.ibc_sinc[0] = *(bufferPtr+1);
115.  MotorControl(); /* Algorithm
    call */
116.  break;
117.  case ADI_SINC_EVENT_STATUS:
118.  break;
119.  default:
120.  break;
121.  }
122.  }
123.  /* Function: SincStatusCallback
    */
124.  static void SincStatusCallback(void*
    pHandle, uint32_t event, void* pArg){
125.  ADI_SINC_EVENT eEvent =
    (ADI_SINC_EVENT)event;
126.  uint32_t status = (uint32_t)pArg;
127.  if (status & ADI_SINC_STATUS_GLIM0){
128.  SINC_TRIP_Fault_handler();
129.  }
130.  }

```