DAQ-STC3™

X Series Driver Development Kit (DDK) Reference Manual

June 2011
372686A-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

Visit ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the Info Code feedback.

Important Information

Warranty

The X Series devices are warranted against defects in materials and workmanship for a period of one year from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

LabVIEW, National Instruments, NI, ni.com, the National Instruments corporate logo, and the Eagle logo are trademarks of National Instruments Corporation. Refer to the *Trademark Information* at `ni.com/trademarks` for other National Instruments trademarks.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at `ni.com/patents`.

Export Compliance Information

Refer to the *Export Compliance Information* at `ni.com/legal/export-compliance` for the National Instruments global trade compliance policy and how to obtain relevant HTS codes, ECCNs, and other import/export data.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Compliance

Electromagnetic Compatibility Information

This hardware has been tested and found to comply with the applicable regulatory requirements and limits for electromagnetic compatibility (EMC) as indicated in the hardware's Declaration of Conformity (DoC)[1]. These requirements and limits are designed to provide reasonable protection against harmful interference when the hardware is operated in the intended electromagnetic environment. In special cases, for example when either highly sensitive or noisy hardware is being used in close proximity, additional mitigation measures may have to be employed to minimize the potential for electromagnetic interference.

While this hardware is compliant with the applicable regulatory EMC requirements, there is no guarantee that interference will not occur in a particular installation. To minimize the potential for the hardware to cause interference to radio and television reception or to experience unacceptable performance degradation, install and use this hardware in strict accordance with the instructions in the hardware documentation and the DoC[1].

If this hardware does cause interference with licensed radio communications services or other nearby electronics, which can be determined by turning the hardware off and on, you are encouraged to try to correct the interference by one or more of the following measures:

- Reorient the antenna of the receiver (the device suffering interference).
- Relocate the transmitter (the device generating interference) with respect to the receiver.
- Plug the transmitter into a different outlet so that the transmitter and the receiver are on different branch circuits.

Some hardware may require the use of a metal, shielded enclosure (windowless version) to meet the EMC requirements for special EMC environments such as, for marine use or in heavy industrial areas. Refer to the hardware's user documentation and the DoC[1] for product installation requirements.

When the hardware is connected to a test object or to test leads, the system may become more sensitive to disturbances or may cause interference in the local electromagnetic environment.

Operation of this hardware in a residential area is likely to cause harmful interference. Users are required to correct the interference at their own expense or cease operation of the hardware.

Changes or modifications not expressly approved by National Instruments could void the user's right to operate the hardware under the local regulatory rules.

[1] The Declaration of Conformity (DoC) contains important EMC compliance information and instructions for the user or installer. To obtain the DoC for this product, visit `ni.com/certification`, search by model number or product line, and click the appropriate link in the Certification column.

Conventions

The following conventions are used in this manual:

<>, [ ]	Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, AO <3..0>, AO [3..0].
[ ]	Square brackets enclose optional items—for example, [`response`].
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **Options»Settings»General** directs you to pull down the **Options** menu, select the **Settings** item, and select **General** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Contents

Chapter 1
Theory of Operation

Chapter 2
Programming Considerations

Chapter 3
Calibration Considerations

Chapter 4
Example Synopses

Chapter 5
X Series Chip Object

Chapter 6
CHInCh Chip Object Registers

Chapter 7
DMA Controller Registers

Chapter 8
SimultaneousControl Registers

Chapter 9
Board Services Registers

Chapter 10
Bus Interface Registers

Chapter 11
Triggers and Timing Registers

Chapter 12
Analog Input Registers

Chapter 13
Counter Registers

Chapter 14
Analog Output Registers

Chapter 15
Digital Output Registers

Chapter 16
Digital Input Registers

Chapter 17
InTimer Registers

Chapter 18
OutTimer Registers

Chapter 19
Stream Circuit Registers

Appendix A
Register Maps Appendix

Appendix B
Technical Support and Professional Services

1

Theory of Operation

Multiplexed X Series devices feature up to 32 analog input (AI) channels, up to four analog output (AO) channels, up to 48 lines of digital input/output (DIO), and four counters.

Functional Overview

The block diagram in Figure 1-1 gives a functional overview of multiplexed X Series devices. Simultaneous X Series devices are similar to the multiplexed devices except for the analog input circuitry involving the analog muxes and ADCs.

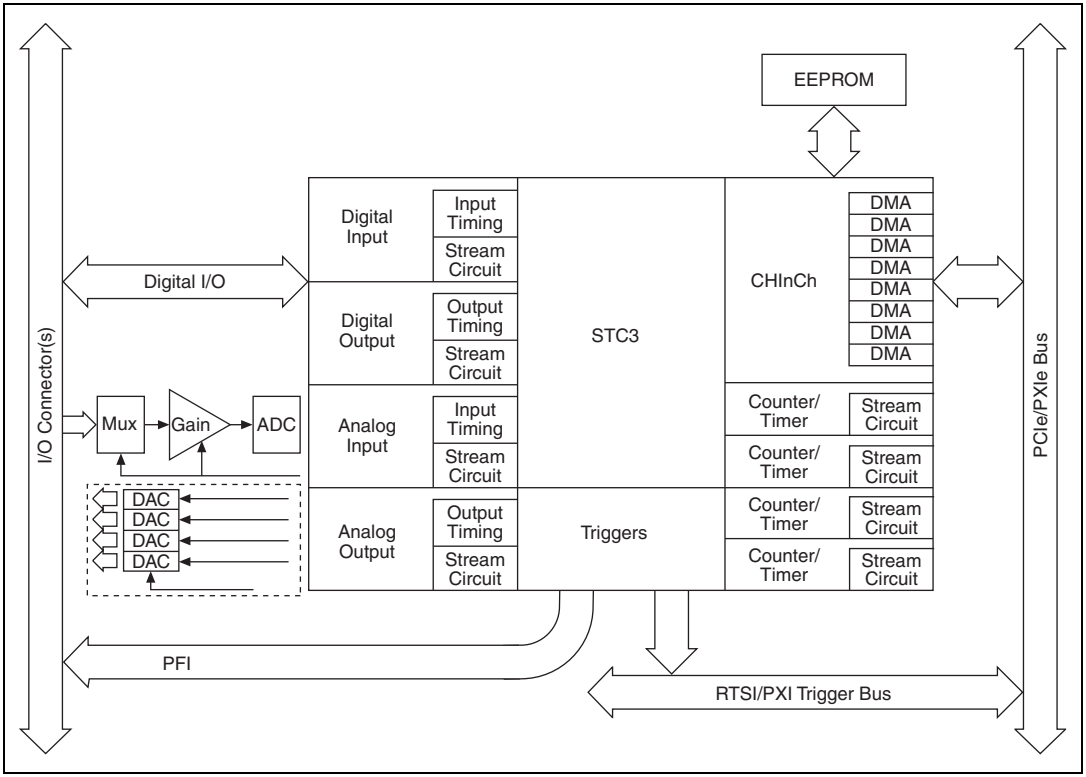


Figure 1-1. Multiplexed X Series Block Diagram

The DAQ System Timing Chip 3 (DAQ-STC3) is the third generation STC ASIC that National Instruments has developed. It provides the majority of digital control for the functions of an X Series device. The Common Host Interface Chip (CHInCh) provides bus communication with PCI Express or PXI Express and is contained within the DAQ-STC3. It contains eight DMA channels that allow you to stream data to and from an X Series device. It also allows for communication with an EEPROM that holds device-specific information, such as calibration information.

An X Series device consists of up to eight subsystems. The digital input and digital output subsystems allow an X Series device to read or write digital signals on the I/O connector of the device, either on-demand or by using a timing engine. The analog input and analog output subsystems allow an X Series device to read or write analog signals on the I/O connector of the device either on-demand or by using a timing engine.

Note Multiplexed X Series devices have varying numbers of analog output DACs depending on the model.

The four counter/timer subsystems allow an X Series device to time signals and produce pulse trains on the I/O connector.

Some other key features of the DAQ-STC3 include the following:

- Flexible AI and AO sample and convert timing
- Many triggering modes
- Independent AI, AO, DI, DO, and counter FIFOs
- Generation and routing of RTSI signals for multi-device synchronization
- Generation and routing of internal and external timing signals
- Four flexible 32-bit counter/timer modules with hardware gating
- Digital waveform acquisition and generation
- Static DIO signals
- True 5 V high current drive DO
- DI change detection
- DO Watchdog Timers
- PLL for clock synchronization
- PCI Express/PXI Express interface
- Independent scatter-gather DMA controllers for all acquisition and generation functions

The interface to the DAQ-STC3 consists of many status and control registers. Table 1-1 shows the address space layout of X Series devices at a per-subsystem level. For more specific information, refer to each subsystems register map and description, Chapters 5 through 19.

Table 1-1. X Series Address Space Layout

Subsystem	Offset	Step (If Multiple)
CHInCh	0x0	—
DMA Channel	0x2000	0x100 (8)
EEPROM	0x5000	—
Simultaneous Control (depending on model)	0x6000	—
DAQ-STC3	0x20000	—
Board Services, Bus Interface, and Triggers	0x20000	—
Analog Input	0x20270	—
Analog Input InTimer	0x202B0	—
Analog Output	0x20400	—
Analog Output OutTimer	0x20470	—
Counter/Timer	0x20300	0x40 (4)
Digital Output	0x204AC	—
Digital Output OutTimer	0x204E0	—
Digital Input	0x20530	—
Digital Input InTimer	0x20560	—
Stream Circuits • AI • Counter 0–3 • DI • AO • DO	0x24000	0x2000 (8)

PCI Interface Circuitry

The Common Host Interface Chip (CHInCh) provides the PCI Express bus interface for X Series data acquisition hardware. This latest generation chip is analogous to the MITE that was present in previous data acquisition hardware. The CHInCh also provides independent scatter-gather DMA controllers for each subsystem to utilize simultaneously. These controllers provide both input and output streaming to and from the host computer.

The basic architecture of PCI Express and PXI Express is very different from that of PCI. Instead of a shared bus between all of the devices, each device has a point-to-point link with the switch it is connected to. A pair of these links (one in each direction) create a lane. Each lane is capable of 250 MB/s of bandwidth in both directions simultaneously. The CHInCh uses one lane of PCI Express, so it can, therefore, transfer up to 250 MB/s of data to and from an X Series data acquisition device simultaneously. In practice however, the limit is about 200 MB/s, and the topology of the system has an impact. Depending on the configuration of the switches involved in a PCI Express system, the performance could be dependent on the other devices in the system. For example, the NI PXIe-1075 chassis has a switch for every four to five slots. This means that data going to or from those devices to the host computer has contention with the other devices near it. Consult your motherboard, controller, and/or chassis documentation for the layout of the PCI Express bus.

Analog Input Circuitry

The multiplexed X Series analog input circuitry consists of 16 or 32 analog channels connected to a multiplexer, followed by a programmable gain instrumentation amplifier (PGIA) and the ADC. The multiplexer connects the positive and negative inputs of the ADC to two different channels for differential mode, a channel and ground for Reference Single-Ended (RSE) mode, or a channel and AI SENSE for Nonreferenced Single-Ended (NRSE) mode. The PGIA amplifies input signals so the ADC can sample small voltages and can handle different ranges on different channels. The input multiplexer and gain adjustment are both controlled by the configuration FIFO, which is incremented after each conversion and will loop continuously until the acquisition terminates.

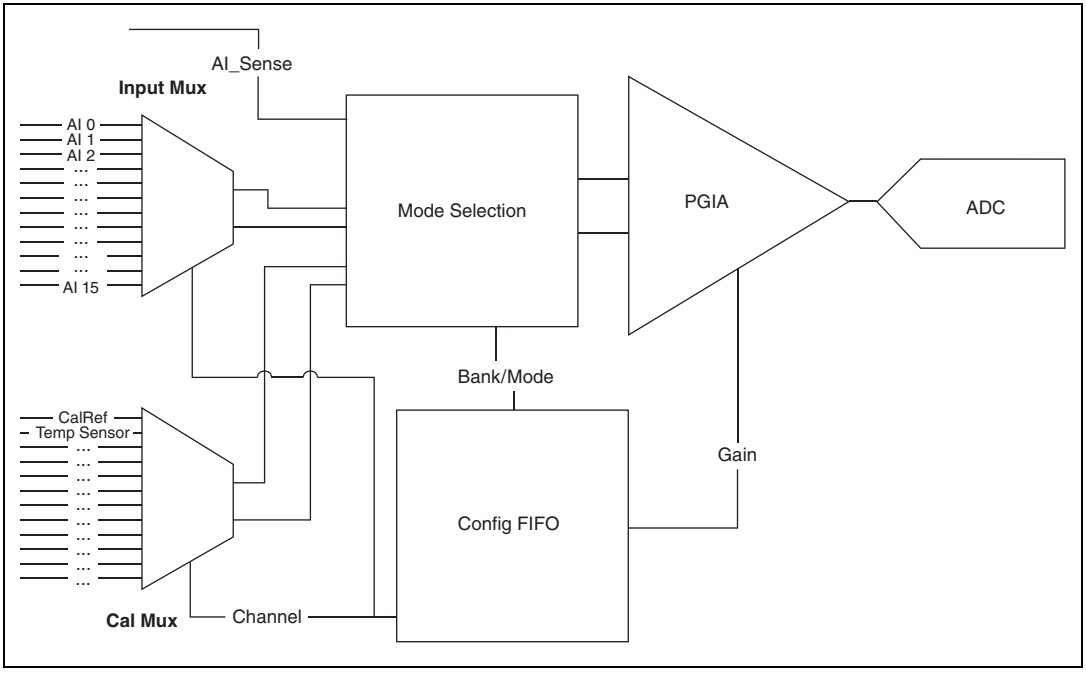


Figure 1-2. Analog Input Circuitry

Since the configuration FIFO value changes after each convert, each channel in the scanlist may have different gain and different terminal configuration (RSE, NRSE, or differential). The configuration FIFO also controls when the current channel scan ends by asserting the last channel signal. The configuration FIFO will start from the beginning during each scan.

In addition to the regular AI 0–AI 15 or AI 0–AI 31 channels, the input multiplexer can also select from calibration sources, as well as loop back the analog output channels on devices that have AO. This allows reading the onboard calibration reference, an AO channel, or the onboard temperature sensor, as well as other auxiliary channels. These channels are listed in Table 1-2.

The Simultaneous MIO (SMIO) X Series analog input circuitry consists of a PGIA for each ADC on the device, and an internal signal multiplexer. The internal signal multiplexer allows selection of calibration sources such as the internal reference, as well as ground versus ground, and AO channels. All ADCs must sample either the external signal or a calibration signal,

since there is only one multiplexer per device. Because each ADC has a separate PGIA, they may have different gains, however SMIO channels are always differential and bipolar.

Analog Output Circuitry

An X Series device consists of zero, two, or four analog output channels. Each channel is driven by a 16-bit digital-to-analog converter (DAC). Each DAC is controlled by AO data lines, a digital sample clock (UPDATE pulse), an analog offset signal and an analog reference signal. In a timed generation, the AO data comes from the FIFO. In a software-driven generation, the AO data comes from the *AO_Direct_Data* register. The AO sample clock (UPDATE) source is controlled by the `AO_UPDATE_Source_Select` field of the *AO_Trigger_Select_Register*. The AO Offset and AO Reference signals are controlled per-channel by the *AO_Config_Bank* registers.

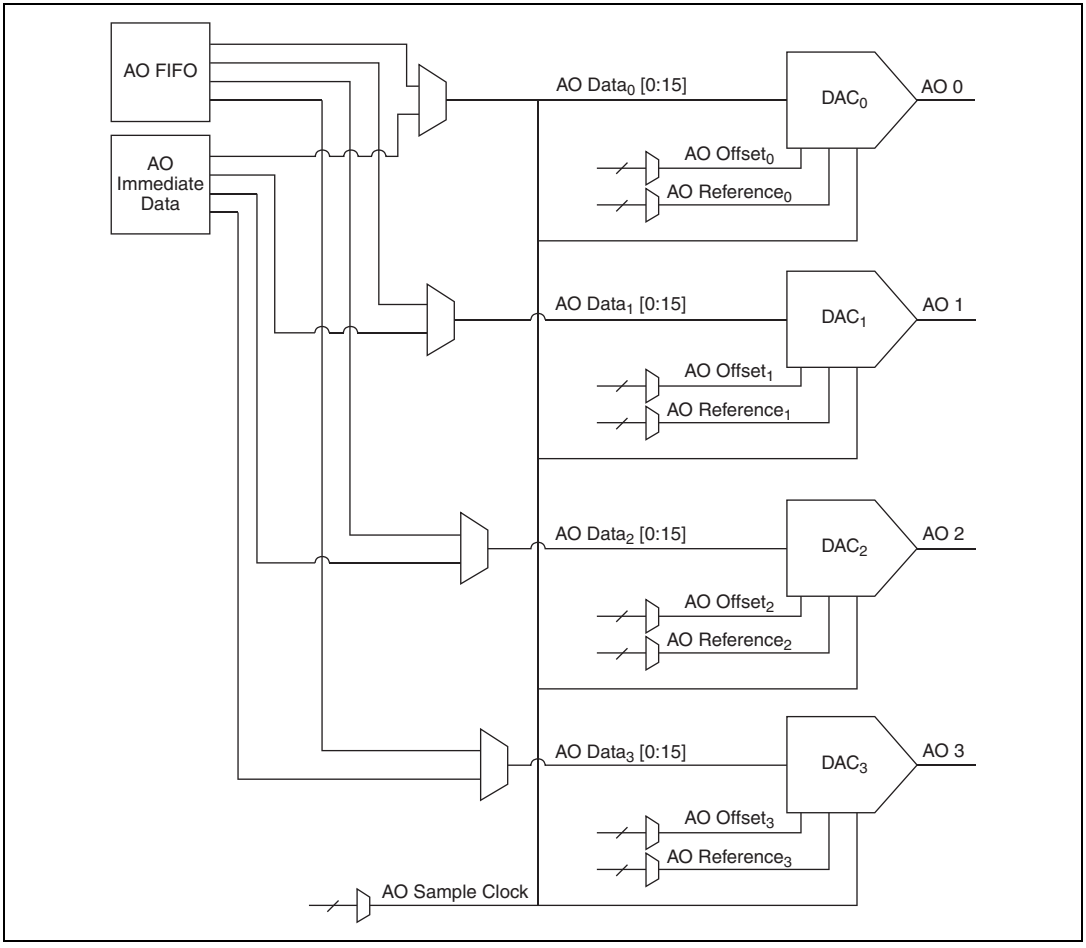


Figure 1-3. Analog Output Circuitry

Digital I/O Circuitry

The DAQ-STC3 ASIC on X Series devices contains most of the hardware required for digital input/output (DIO). The DIO functionality of the DAQ-STC3 is available on three different ports: P0, P1, and P2. P0.<0..31> comprises bidirectional DIO lines, and P1.<0..7> and P2.<0..7> consist of lines that can be used for triggering or static DIO. The P1 and P2 are collectively known as the Programmable Function Interface (PFI). Certain types of X Series devices may have only eight lines of bidirectional I/O P0.<0..7> available through their connector(s).

The P0 circuitry is illustrated in Figure 1-4.

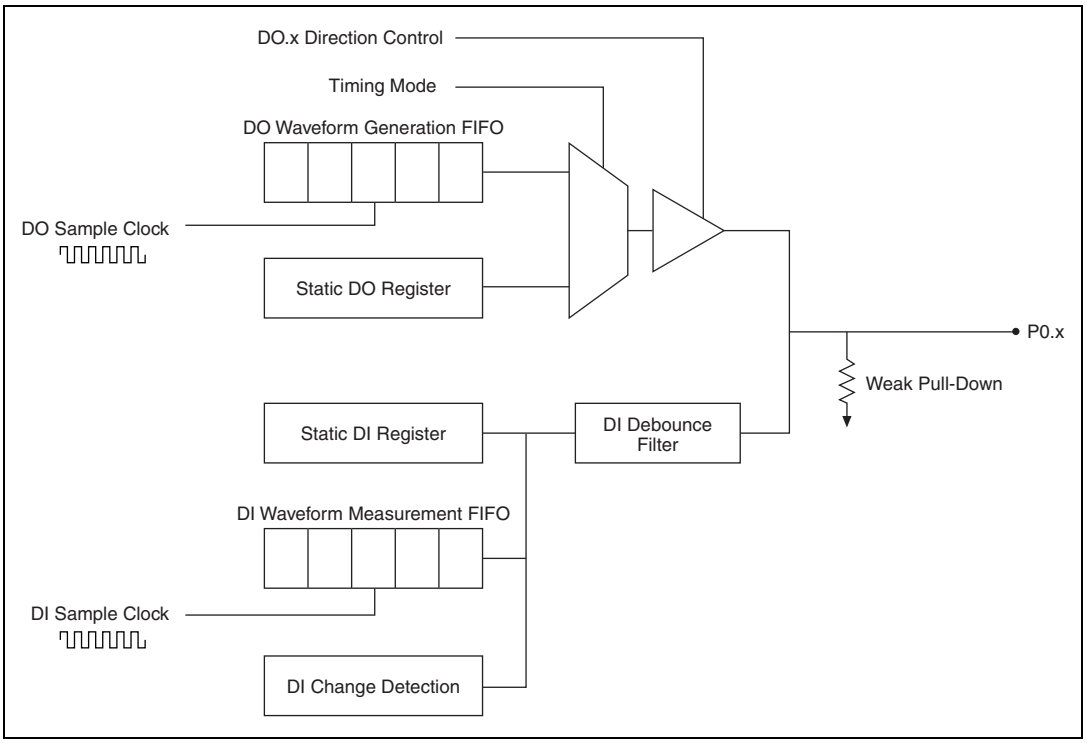


Figure 1-4. Digital Input/Output Circuitry

The PFI DIO functionality through P1 and P2 is discussed in detail in the *Trigger Circuitry* section.

Digital Input

There are two DI acquisition modes, as shown in Figure 1-4:

- Static DI, also referred to as software-timed, where the rate of acquisition is determined by the speed of the software application.
- Timed DI, also referred to as hardware-timed, where a digital hardware signal (DI Sample Clock) controls the rate of the acquisition.

While timed DI is possible only on P0, static DI is possible on P0, P1, and P2. For static DI, the digital values on the P0 lines can be read through the *Static_Digital_Input* register.

In the case of timed DI, digital values on the P0 digital lines are sampled periodically (the rate being determined by the DI Sample Clock) and stored in the DI waveform measurement FIFO, as shown in Figure 1-4, and subsequently transferred to the host computer using DMA.

To program timed DI, refer to the *Input Timing Circuitry* and *Direct Memory Access* sections.

Some of the other features of DIO on the DAQ-STC3 are as follows:

- Direction and function of each line is individually controllable
- DI change detection
- DI debounce filters

Change Detection

The change detection circuitry enables detection and reporting of any falling or rising edges on P0. Change detection can be enabled by configuring the *DI_ChangeIrqFE* register for falling edge and *DI_ChangeIrqRE* register for rising edges on the desired P0 lines.

The *DI_ChangeDetectStatus* register can be read to determine if a change detection event was detected. The *DI_ChangeDetectLatchedDI* register reflects the state of the P0 lines at the time the last change detection event occurred.

Debounce Filters

A debounce filter can be enabled on P0 lines to filter digital signals that do not meet a minimum pulsewidth condition. Filter configuration on a per-line basis can be done through the *DI_FilterLo* and *DI_FilterHigh* registers.

Digital Output

There are two digital output (DO) generation modes:

- Static DO, also referred to as software-timed, where the rate of generation is determined by the speed of the software application.
- Timed DO, also referred to as hardware-timed, where a digital hardware signal (DO Update) controls the rate of the acquisition.

Power-Up State

At system startup and reset, the hardware sets all PFI and DIO lines to high-impedance inputs by default. It is advisable to implement a software routine to statically change the P0 lines to a known state on power-up.

Power-up states are user-configurable, outside the DDK. The states will default to high-impedance before the first configuration.

Watchdog Timer

The onboard Watchdog Timer can be programmed to set the P0 outputs to safe states in the event of no communication between the host computer application and the X Series device.

The Watchdog Timer can be enabled through the *DO_WDT_ModeSelect* registers and the per-line safe state value can be specified through the *DO_WDT_SafeState* register. The Watchdog Timer can be configured by writing to the *Watchdog Timer Registers*, found in Chapter 9, *Board Services Registers*.

When the Watchdog Timer is enabled, the *WatchdogControl Register* register must be pinged with alternating 0xFEED and 0xF00D commands to keep the Watchdog from timing out.

Only a single Watchdog Timer is available for P0 and PFI lines.

Connecting Digital I/O Signals

The DAQ-6202 ASIC on the X Series devices exists to do buffering of the DAQ-STC3 DIO lines. When the DAQ-STC3 is programmed to perform digital input or output, it issues commands to the DAQ-6202 to perform the appropriate change in direction, thus ensuring transparent operation of DIO lines.

Programming the P1 and P2 Digital I/O Registers

1. **pfiDioHelper::reset(powerupStates, numPowerupStates, status)**

   Resets P1 and P2 lines using the corresponding powerup information.

2. **pfiDioHelper::readPresentValue(lineMask, value, status)**

   Returns the current value being sensed on the P1 and P2 lines specified in the lineMask.

3. **pfiDioHelper::writePresentValue(lineMask, newValue, status)**

   Drives the value in newValue onto the P1 and P2 lines specified by the lineMask.

4. **pfiDioHelper::configureLines(lineMask, configuration, status)**

   Configures the P1 and P2 lines specified in the lineMask for the desired direction.

Input Timing Circuitry

The input timing engine of the DAQ-STC3 consists of counters that control the various signals sent to the analog input circuitry. It is responsible for generating the sample and convert clocks, as well as clocking data out of the front end of the subsystem. On simultaneous subsystems, such as DI or AI on SMIO devices, the timing circuitry only controls the sample clock, there is no convert clock or configuration FIFO.

Counters

The DAQ-STC3 input timing engine consists of four main counters. Each counter has start, stop, and reset control lines, plus an output for terminal count. The counters are programmed with a terminal count value before the acquisition begins, and can output the terminal count (TC) condition to another counter. The inputs and outputs of each counter can be routed to RTSI or PFI lines on the DAQ-STC3, allowing for external sample or convert clocks, and external triggers.

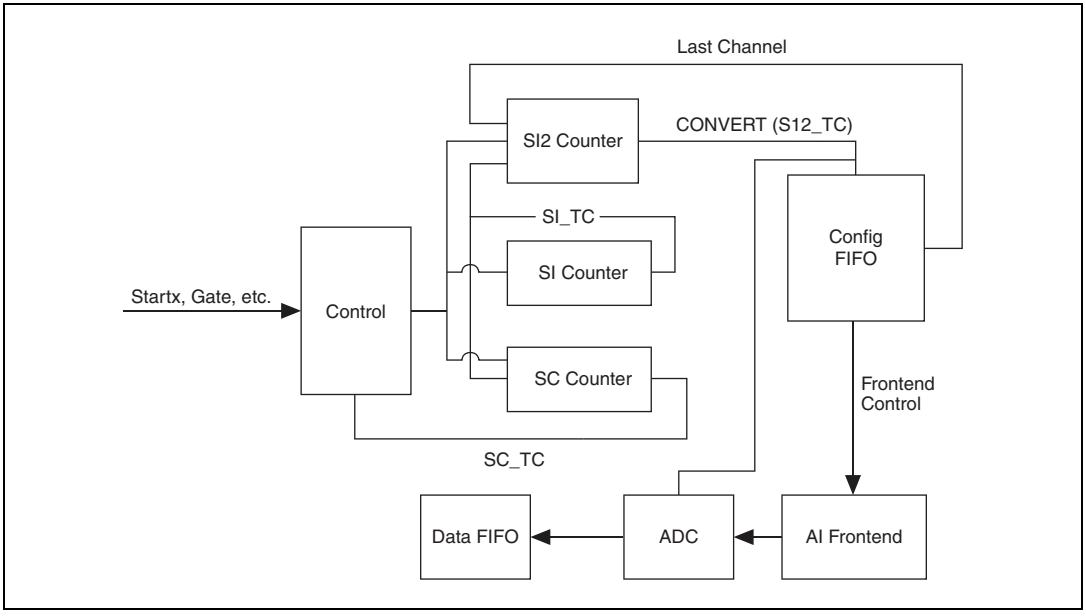


Figure 1-5. AI Timing Circuitry

The sample count (SC) counts down by one for each new sample acquired by the DAQ-STC3 on all the channels. Each sample consists of a scan of the channels in the scanlist on MIO devices, or a conversion of each channel on simultaneous subsystems, such as DI or AI on SMIO devices. When the SC counter reaches terminal count (0), the SC_TC signal is asserted and can be used to stop a finite acquisition. The SC counter can load an initial value from either the A or B load register, and can optionally switch between the two values on SC_TC. This allows for reference triggered acquisitions, where the A register contains a large value and the B register contains the number of post-trigger samples to acquire.

The sample interval (SI) is used to control the sample rate, by generating the internal sample clock. It can use an internal or external timebase to count from. The terminal count on this counter (SI_TC) will cause the start of a new sample, which means a series of convert pulses on MIO devices, a convert for all the ADCs on SMIO devices, or a convert for the digital front end. Like the SC counter, the SI counter has two load registers, A and B. Using these registers and the right load mode, the delay between the start trigger and start of sampling can be controlled.

The sample interval 2 (SI2) counter controls the convert rate by generating the internal convert clock. This counter is not used on simultaneous subsystems. The terminal count of this counter (SI2_TC) is used as the

convert signal and causes the ADC to convert a new sample each time the counter reaches 0. Using the A and B load registers and alternate on stop reload mode allows you to control the delay between the sample clock and the start of conversions. Unlike the SC and SI counters, the SI2 counter does not run free—it uses separate START and STOP signals to start and stop the generation of convert pulses.

The DIV counter is used to repeat the scanlist multiple times per sample on MIO devices. Normally, the SI2 STOP signal comes from the configuration FIFO, however the DIV counter can intercept this signal and cause SI2 to keep generating pulses until N STOP signals have occurred.

Configuration FIFO for Analog Input

On MIO devices, the configuration FIFO is used to set the channel, mode, gain, and to determine whether it is the last channel for each convert during a sample. Two registers, one to write and one to clear, control this FIFO. The FIFO advances on CONVERT (usually SI2_TC), and its output signals drive the front end directly. When the FIFO reaches an item that has last channel set, the STOP signal is asserted and the SI2 counter will stop generating CONVERT pulses.

On SMIO devices, the configuration FIFO is not used. Instead, the *AISetChannelOrder* register must be used to order and configure channels. Once a channel is written, you can write the `AiGain` bitfield in the *AIChanConfigCtrlStat* register to configure the channel before setting AISetChannelOrder to the next channel in the scanlist. Using the CalCtrlStat and CalMuxSel registers, you can change the input source for each channel from the external signal to one of the onboard signals.

Tables 1-2 and 1-3 show how to select some commonly used channels on MIO and SMIO X Series devices.

Table 1-2. FIFO Configuration on MIO Devices

Desired Channel	Channel Type	Bank	Channel
ai0:15	nAI::kDifferential, nAI::kRSE, or nAI::kNRSE	nAI::kBank0	0–15
ai16:31	nAI::kDifferential, nAI::kRSE, or nAI::kNRSE	nAI::kBank1	16–31
_aiGnd_vs_aiGnd	nAI::kLoopback	nAI::kBank0	1
_ao0:1_vs_aoGnd	nAI::kLoopback	nAI::kBank0	2, 3
_ao2:3_vs_aoGnd	nAI::kLoopback	nAI::kBank1	2, 3
_boardTempSensor_vs_aiGnd	nAI::kInternal	nAI::kBank0	4

Table 1-3. Channel Configuration on SMIO Devices

Desired Channel	Channel	CalMuxSel	CalCtrlStat
ai0:15	0–15	0	0xA
_aiGnd_vs_aiGnd	0–15	0	0x5
_ao0:3_vs_aoGnd	Any	4–7	0x5

Triggers

Once armed, the DAQ-STC3 timing engine is controlled by several triggers. START1, the start trigger, will cause all counters to begin counting and start the acquisition. It can be configured to use an external signal or a software pulse as a source. When in pretrigger acquisition mode, the DAQ-STC3 uses the reference trigger (START2) to stop the acquisition. START2 can also be sourced from a software pulse or external signal.

In addition to the START1 and START2 signals, the gate can be used as a pause trigger. This can be controlled by a register or an external signal. When asserted, the gate signal will pause the acquisition until the signal is deasserted.

Programming the Counters and Configuration FIFO

The aiHelper object included with the MHDDK helps with programming the AI timing engine and configuration FIFO. To start using AI, you must create an aiHelper object after creating a tXSeries object. It is recommended that you call `reset()` before configuring acquisitions to be sure hardware and software are in the same state.

When configuration is being performed on the AI timing engine, a 1 must be written to the `Configuration_Start` field of the *Reset_Register*. This will hold the timing engine in reset until configuration is done, at which point the `Configuration_End` field should be written with a 1. Both fields are strobe fields, meaning there is no need to write 0; they are cleared automatically.

Once the AI timing engine is in reset, configuration can begin. Use the aiHelper methods to assist with this. The methods can be called in any order, however they must be called while the timing engine is in reset. The following are the methods used to program timing, in the order typically used to set up a hardware timed acquisition.

1. **aiHelper::programExternalGate(source, polarity, status)**

   This method sets a pause trigger or disables it. The gate may be disabled, or can be an active high or low external signal.

2. **aiHelper::programStart1(source, polarity, edge, status)**

   This method programs START1, the start trigger. Setting source to software pulse allows software to stop the acquisition, otherwise an external signal may be used for a reference trigger. It is recommended that edge be left as True, since most applications will use edge-sensitive and not level-sensitive triggers.

3. **aiHelper::programStart2(source, polarity, edge, status)**

   This method programs START2, the reference trigger. Setting source to software pulse allows software to start the acquisition, otherwise an external signal may be used for a start trigger. It is recommended that edge be left as True, since most applications will use edge-sensitive and not level-sensitive triggers.

4. **aiHelper::programStart(source, polarity, edge, status)**

   This method programs START, the sample clock. Use an external line to configure an external sample clock, otherwise set the source to Internal Timing to have the DAQ-STC3 generate the sample clock. Edge should be True for most applications.

5. **aiHelper::programConvert(source, polarity, status)**

   This method programs the convert clock. For SMIO devices, this should be set to internal timing, with the same polarity as the START signal. For MIO devices, this should be active low, and can be internal or external. When using an external clock, `Start_Stop_Gate_Enable` in the mode 1 register must be programmed to 1 so that START and STOP gate the external convert clock.

6. **inTimerHelper::programTiming(timing, status)**

   This method programs timing according to settings in the inTimerParams object, timing. The following are setting methods used to build a timing configuration.

   a. **timingConfig::setAcqLevelTimingMode(mode, status)**

      This method sets the type of acquisition to perform. Post Trigger mode acquires a finite number of samples after START1, Pre Trigger does the same, but also waits for START2, and acquires a finite amount of samples after it. Continuous mode will run after START1 and not stop until timing is stopped or an error occurs.

   b. **timingConfig::setUseSICounter(useCounter, status)**

      Enables or disables the sample interval counter. The counter can be disabled to use an external sample clock.

   c. **timingConfig::setSamplePeriod(period, status)**

      Sets the sample period, in units of SI counter timebase ticks. The default timebase is 100 MHz. The sample period only applies when SI is enabled.

   d. **timingConfig::setSampleDelay(delay, status)**

      Sets the delay between the start trigger and the start of the acquisition, in units of SI counter timebase ticks. The delay must be set to at least 2.

   e. **timingConfig::setNumberOfSamples(number, status)**

      Sets the number of samples to acquire when Pre Trigger or Post Trigger timing mode is used. This method is programmed into the SC counter.

   f. **timingConfig::setUseSI2Counter(useCounter, status)**

      Enables or disables the convert interval counter, SI2. Use this method on MIO devices when an internal convert clock is desired.

g. **timingConfig::setConvertPeriod(period, status)**

Sets the convert clock period, in units of SI2 timebase ticks. This is 100 MHz by default.

h. **timingConfig::setConvertDelay(delay, status)**

Sets the delay between sample clock and the first convert during each sample period, in units of SI2 counter timebase ticks. The delay must be set to at least 2.

7. **aiHelper::programFIFOWidth(width, status)**

Sets the width of the AI FIFO. This can be two Bytes (16 bits) for most applications.

8. **inTimerHelper::clearFIFO(status)**

Clears the input FIFO. All data is removed and the sample count becomes zero.

9. **inTimerHelper::clearConfigurationMemory(status)**

Clears the channel configuration FIFO.

10. **aiHelper::programChannel(config, status)**

Puts a value in the configuration FIFO. The channel config struct holds the gain, polarity, type of channel, and whether the channel is the last in the list.

11. **inTimerHelper::primeConfigFIFO(status)**

This method advances the config FIFO by 1. This is needed to select the first channel in the list before an acquisition starts.

12. **inTimerHelper::armTiming(status)**

This method arms the timing engine. Once armed, a START1 trigger starts the timing engine and begins the acquisition.

13. **inTimerHelper::strobeStart1(status)**

When software is selected as a START1 source, this starts the acquisition.

Using the InTimer with Digital Input

The generic InTimer is also used with the digital input subsystem. Programming is performed in the same way as simultaneous AI, since all digital lines will be sampled on DI_CONVERT and data sent to the DI FIFO. Start and reference triggers operate in the same manner as for AI. Unlike AI, there is no configuration FIFO for digital input.

To use hardware-timed digital input, the *DI_Mask_Enable_Register* must be set to 1 for each bit that is to be read. Then, the *DI_Mode_Register* must be used to set the width of the DI FIFO data and which lines to retrieve data from. Finally, DI timing programming can be performed in the same way as AI but using the *DI_Trigger_Select_Register* and the DI instance of the InTimer.

A diHelper, similar to the aiHelper discussed in the *Programming the Counters and Configuration FIFO* section, is also available.

Output Timing Circuitry

The analog output timing engine of the DAQ-STC3 consists of counters that control the various signals sent to the analog output circuitry. It is responsible for generating the AO Sample Clock as well as clocking data into the DAC and selecting the channel for each conversion. The analog output timing engine is an extension of the generic OutTimer engine, which is also used for hardware timed digital output.

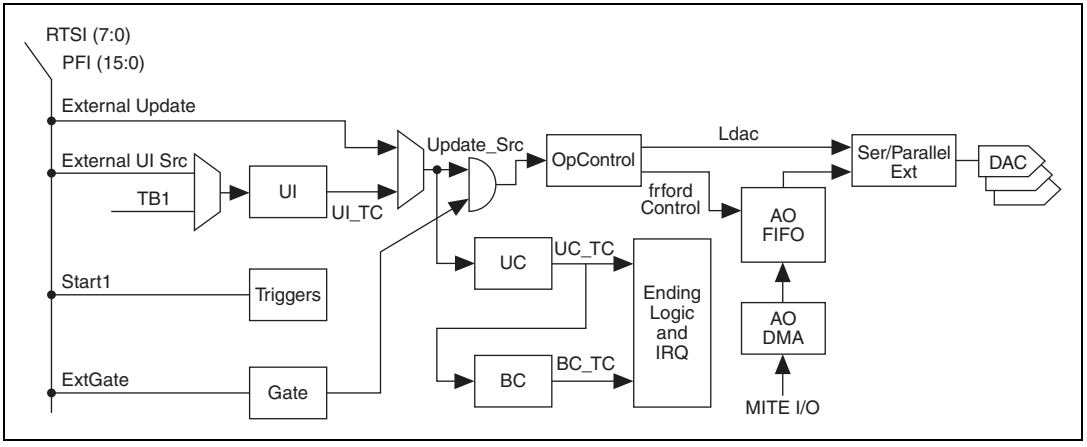


Figure 1-6. Analog Output Timing Circuitry

Counters

The DAQ-STC3 output timing engine consists of three main counters. Each counter has start, stop, and reset control lines, plus an output for terminal count. The counters are programmed with a count value before the generation begins, and can output the terminal count (TC) condition to another counter. The inputs and outputs of each counter can be routed to RTSI or PFI lines or the DAQ-STC3, allowing for external sample clocks and triggers.

The 32-bit update count (UC) counts down by one for each sample generated by the DAQ-STC3 on all channels, that is, once per UPDATE pulse. When the UC counter reaches its terminal count (0), the UC_TC signal is asserted. The UC counter can load an initial value from either the A or B load registers, and can optionally switch between the two values on UC_TC. This allows for the generation of any arbitrary number of points in a single buffer.

The 32-bit buffer counter (BC) counts down by one for each full buffer generated by the DAQ-STC3 on all channels, that is, once per UC_TC. This essentially turns the 32-bit UC and BC counters into a single, collective 64-bit counter. When the BC counter reaches its terminal count (0), the BC_TC signal is asserted and can be used to terminate a finite generation.

The 32-bit update interval (UI) counter is used to control the sample rate by generating an internal sample clock. It can be sourced from an internal or external timebase. The terminal count on this counter (UI_TC) causes an UPDATE pulse, which causes the DACs to latch the digital value and output a new AO value. The UI counter can load an initial value from either the A or B load registers, and has the option of switching between the two values on UI_TC. Using these registers and the right reload mode, you can control the delay between the start trigger and the start of generating.

AO Channel Selection

The DAQ-STC3 can be configured to write to only certain DACs through an order configuration FIFO. Configuring channel selection involves writing to two registers—a single write to the *AO_Config_Control_Register* to clear the FIFO, and then sequential writes to *AO_Order_Config_Data_Register*—to add each of the channels to be written during the generation.

Triggers

The DAQ-STC3 output timing engine, once armed, is controlled by several triggers. START1, the start trigger, will cause all counters to begin counting and start the acquisition. It can be configured to use an external signal or a software pulse as a source. Additionally, the AO Gate can be used as a pause trigger. The pause trigger can be controlled by an external signal or by register writes. When asserted, the gate signal pauses the generation until the signal deasserts.

Programming the Timers, Selecting Channels, and Configuring DACs

Using the aoHelper object included with the MHDDK can assist with programming the AO timing engine. To start using AO, first create a tXSeries object and an aoHelper object. It is recommended that you call `reset()` before configuring the generation to ensure that hardware and software are in the same state.

Begin configuration of the AO timing engine by writing a 1 to the `Configuration_Start` field of the *Reset_Register*. This holds the timing engine in reset until configuration is done, at which point the `Configuration_End` field should be written with a 1. Both fields are strobe fields, meaning there is no need to write 0; they are cleared automatically.

The `AO_Offset` and `AO_Reference` fields in the *AO_Config_Bank Register* select the analog offset and reference signals for each DAC. The offset and reference mapping is device-specific. Refer to Tables 1-4 and 1-5 for X Series offset and reference mapping.

Table 1-4. Offset Value Mapping

Signal	AO_Offset_Value				
AO GND	0				
Internal 5 V	1				
APFI 0	2				
APFI 1	3				
AO x	**AO_Config_Bank Register**	**AO 0**	**AO 1**	**AO 2**	**AO 3**
	AO_Config_Bank_0	—	4	5	6
	AO_Config_Bank_1	4	—	5	6
	AO_Config_Bank_2	4	5	—	6
	AO_Config_Bank_3	4	5	6	—

Table 1-5. Reference Value Mapping

Signal	AO_Reference_Value				
Internal 10 V	0				
Internal 5 V	1				
APFI 0	2				
APFI 1	3				
AO x	**AO_Reference_Value Register**	**AO 0**	**AO 1**	**AO 2**	**AO 3**
	AO_Config_Bank_0	—	4	5	6
	AO_Config_Bank_1	4	—	5	6
	AO_Config_Bank_2	4	5	—	6
	AO_Config_Bank_3	4	5	6	—

Once the AO timing engine is in reset, configuration can begin. Use the aoHelper methods to assist with this. The methods can be called in any order, however they must be called while the timing engine is in reset. The following are the methods used to program timing, in the order typically used to set up a hardware timed acquisition.

1. **aoHelper::programExternalGate(source, sourcePolarity, enable, status)**

   This method configures the external gate (pause trigger). The trigger can be disabled, or can be sourced from an external active-high or active-low signal.

2. **aoHelper::programStart1(source, sourcePolarity, start1Edge, status)**

   This method configures the START1 trigger (start trigger). The trigger can be software-generated, or can be sourced from an external active-high or active-low signal.

3. **aoHelper::programChannels(channels, status)**

   This method enables a list of channels. This method should be called exactly once and the order of the channels passed into the method will control the order that the AO channels get updated.

4. **aoHelper::programConfigBank(channel, reference, updateMode, status)**

   This method configures an individual DAC's reference source, offset source, and update mode (timed or immediate).

5. **aoHelper::programUpdate(source, sourcePolarity, status)**

   This method configures the UPDATE (sample clock) source when the UPDATE mode is Timed. The source can be the UI_TC signal or an external active-high or active-low signal.

6. **outTimerHelper::programUICounter(source, sourcePolarity, continuous, status)**

   This method configures the UI counter. The source of the UI counter can be the TB3 (100 MHz) timebase or an external active-high or active-low signal. The UI counter can be configured for continuous or finite operation.

7. **outTimerHelper::programStopCondition(triggerOnce, continuous, stop_On_BC_TC_Error, stop_on_BC_TC_Trigger_Error, stop_On_Overrun_Error, status)**

   This method configures the conditions under which the counters should stop. The various stop_on_error parameters dictate whether or not the generation should be stopped if various error conditions occur. A BC_TC error occurs if AO_BC_TC_Interrupt_Ack is not set between two AO_BC TCs. A BC_TC trigger error occurs when a START1 trigger is received after the last BC_TC of a staged waveform but before AO_BC_TC_Interrupt_Ack is set to 1. An overrun error occurs when an AO_UPDATE command is issued to a DAC that was not loaded with data.

8. **outTimerHelper::programBufferCount(bufferLoopCount, status)**

   This method sets the BC counter load register A, preloads the BC counter initial value from load register A, and sets the reload mode to No_Change.

9. **outTimerHelper::programUpdateCount(updatesPerBufferPrimary, updatesPerBufferSecondary, status)**

   This method sets both of UC's load registers (A and B), but does not actually load either register into the UC counter. It also sets the UC reload mode to Alternate_First_Period_Every_BC_TC. This facilitates multiple start triggers that generate a finite AO waveform: the UC counter will reload from the *same* load register over and over until a BC_TC is observed. Once observed, the counter reloads from the *other* load register *once*, and then switches back to the original load register and reloads from it repeatedly until another BC_TC.

10. **outTimerHelper::programFIFO(FIFOEnable, FIFOMode, useOnlyOnboardMemory, status)**

    This method enables or disables the FIFO and configures the condition under which a DMA request is issued. The useOnlyOnboardMemory flag can be set to True to cause the device to regenerate the data from its onboard FIFO over and over instead of requesting data from the host.

11. **outTimerHelper::programBCGate(gate, status)**

    This method enables or disables the BC gate. Enabling the BC_GATE allows external AO_UPDATE pulses to pass only when the AO_BC counter is enabled to count. Set this bit to 0 in the internal AO_UPDATE mode (AO_UPDATE_Source_Select is set to 0) or if you are using AO_UPDATE_Pulse. Otherwise, set to 1.

12. **outTimerHelper::programSoftwareGate(gate, status)**

    This method enables or disables the software gate. Enabling the software gate pauses the AO generation, and disabling the software gate resumes the AO generation.

13. **outTimerHelper::loadUI(start1DelayCount, UI_Updates, status)**

    This method loads the UI counter with the value in UI_Updates. The start1DelayCount configures a number of sample clocks between the AO Start trigger and the first actual update to the DACs. The minimum delay count is 2.

14. **outTimerHelper::loadUC(status)**

    This method loads the UC counter with the values programmed during programUpdateCount().

15. **outTimerHelper::armTiming(status)**

    This method arms the various enabled AO timing engine counters (UI, BC, and UC). Once armed, a START1 trigger starts the generation.

16. **outTimerHelper::setArmUI(arm, status)**

    This method enables the arming of the UI counter when outTimer::armTiming() is called.

17. **outTimerHelper::setArmBC(arm, status)**

    This method enables the arming of the BC counter when outTimer::armTiming() is called.

18. **outTimerHelper::setArmUC(arm, status)**

    This method enables the arming of the UC counter when outTimer::armTiming() is called.

19. **outTimerHelper::changeRateWhileRunning(newUICount, status)**

    This method loads a new value into UI's Load B register, which can be used to change the AO sample rate while the timing engine is running.

20. **outTimerHelper::disarmTiming(status)**

    This method disarms all of the AO timing engine counters (UI, BC, and UC) and disables the rearming of those counters when outTimerHelper::armTiming() is called.

21. **outTimerHelper::notAnUpdate(status)**

    This method preloads the DACs with the first sample to be generated once an UPDATE pulse is received.

22. **outTimerHelper::acknowledgeUpdate(acknowledgeUpdate, status)**

    This method acknowledges an UPDATE interrupt.

23. **outTimerHelper::readUpdate(status)**

    This method returns the UPDATE_St bit from the OutTimer's Status_1 register. This bit is set whenever an UPDATE is observed and is cleared by calling outTimerHelper::acknowledgeUpdate().

Using the OutTimer with Digital Output

The generic OutTimer is also used with the digital output subsystem. Programming is performed in the same way as AO. Triggering operates in the same manner as for AO. Unlike AO, there is no configuration FIFO for digital output.

To use hardware-timed digital output, the *DO_Mask_Enable_Register* must be set to 1 for each bit that is to be written. Then, the *DO_Mode_Register* must be used to set the width of the DO FIFO data and which lines to write data to. Finally, DO timing programming can be performed in the same way as AO but using the *DO_Trigger_Select_Register* and the DO instance of the OutTimer. There is a doHelper similar to the aoHelper mentioned in the *Programming the Timers, Selecting Channels, and Configuring DACs* section to help with programming.

Counter Timer I/O Circuitry

Counters perform one function, counting. Other signals modify and control when counting is started, stopped, and latched into or from the FIFO. X Series data acquisition hardware exposes the four counters available on the DAQ-STC3. These counters are improved and enhanced from those available on previous system timing chips.

Signals and Terminology

- **Source**—This is what the counter counts.
- **Gate**—This signal starts, stops, or latches the counting of the source based on how it is configured.
- **Second Gate**—This signal can be configured to modify the Gate signal. With this and the Gate, you can create level behavior between two signals.
- **Sample Clock**—This signal tells the counter to latch data into or from the FIFO.
- **A, B, Z**—These signals are specific to position measurements. B can also be configured to be the count direction (either up or down) for simple event counting.
- **Hardware Arm**—This signal can arm the counter if it needs to be controlled by an external signal.
- **Terminal Count**—When the counter reaches 0, it hits the terminal count, or TC, condition. At this point, it can either reload a value or wrap back around depending on the configuration of the counter. It will also modify its output by either switching state (high to low or low to high) or pulsing its output. In this manner, it is able to produce timed pulses.
- **Aux Counter**—The Aux Counter is new in the DAQ-STC3 and provides a wide array of new available applications. Use the Aux Counter to perform more complicated sample clocked measurements and buffered pulse trains. This is sometimes referred to as the embedded counter.

Features and Common Applications

The *X Series User Manual* contains a chapter on counters that explains in great detail how the counters work on X Series data acquisition hardware. It includes timing diagrams that explain the applications in-depth. Refer to the user manual, the register map, and the examples for more information.

Programming Note

If the Gi_ForceSourceEqualToTimebase bitfield in the *Gi_Mode_Register* is enabled during an acquisition or generation, special care must be taken to ensure that certain registers and bitfields are not communicated with when the Timebase may not be present. In that case, the hardware may be unresponsive, potentially causing a bus error. The registers and bitfields listed in Table 1-6 can be affected.

Table 1-6. Potential Gi_ForceSourceEqualToTimebase Bus Error Registers

Register	Bitfield
Gi_Load_A_Register	All
Gi_Load_B_Register	All
Gi_AutomaticLoad_Register	All
Gi_Command_Register	Gi_Load
Gi_DMA_Config_Register	Gi_DMA_Reset
Gi_Mode_Register	All
Gi_Mode2_Register	All
Gi_AuxCtr_Register	All
Gi_SampleClockRegister	All

In situations where this may be possible, the counter must be forced reset using the Gi_Reset bitfield in the *Gi_Command_Register*. Refer to the documentation for Gi_Disarm in the same register for more information.

Trigger Circuitry

The DAQ-STC3 trigger module on X Series devices supports configuration of programmable function interface (PFI) lines and connection of the PXI_Trig[0:7] bus to various onboard terminals. The PFI lines are divided into two ports P1 and P2.

The PFI circuitry is illustrated in Figure 1-7.

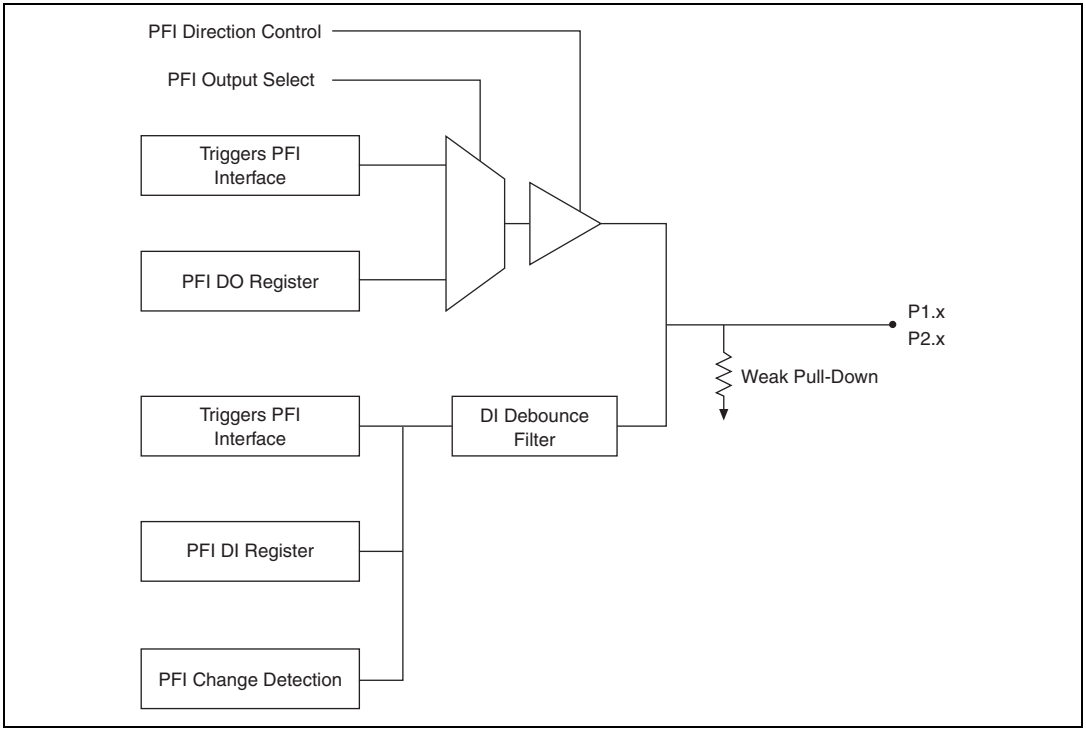


Figure 1-7. PFI Circuitry Block Diagram

PFI

This section includes information on PFI circuitry, including the following sub-sections: *Digital I/O*, *DI Change Detection*, *DI Debounce Filters*, *DO Watchdog Timer*, *PXI_Trig Lines*, *Frequency Out (FOUT)*, *Analog Trigger Control*, *Phase-Locked Loop (PLL)*, and *Internal Trigger Filters*.

Digital I/O

The PFI circuitry supports static DI and DO, where the rate of acquisition and generation of digital values is determined by the speed of the software application, respectively.

Static DI can be performed by setting the *PFI_Direction_Register* to input and by reading the digital value from the *PFI_DI_Register*.

Conversely, static DO can be performed by setting the *PFI_Direction_Register* to output and writing the digital value to the *PFI_DO_Register*.

DI Change Detection

The change detection circuitry enables detection and reporting of any falling or rising edges on the PFI lines. Change detection can be enabled by configuring the *PFI_ChangeIrq_Register* for rising and falling edges independently for the corresponding PFI line.

The *DI_ChangeDetectStatusRegister* can be read to determine if a change detection event was detected. The *DI_ChangeDetectLatchedPFI_Register* reflects the state of the PFI lines at the time the last change detection event occurred.

DI Debounce Filters

A debounce filter can be enabled on PFI lines to filter digital signals that do not meet a minimum pulsewidth condition. Filter configuration on a per-line basis can be done through the PFI_Filter registers listed in the *Triggers and Timing Registers* section of Chapter 11, *Triggers and Timing Registers*.

DO Watchdog Timer

The onboard Watchdog Timer can be programmed to set the PFI outputs to safe states in the event of no communication between the host computer application and the X Series device.

The Watchdog Timer can be enabled through the *PFI_WDT_ModeSelect_Register* and the per-line safe state value can be specified through the *PFI_WDT_SafeStateRegister*. The Watchdog Timer timeout and other settings can be configured through the *Watchdog Timer Registers* in Chapter 9, *Board Services Registers*.

When the Watchdog Timer is enabled, the *WatchdogControl Register* must be pinged with alternating 0xFEED and 0xF00D commands to keep the Watchdog from timing out.

Only a single Watchdog Timer is available for P0 and PFI lines.

PXI_Trig Lines

Digital signals, such as triggers and sample clocks, can be exported to or imported from the PXI_Trig[0:7] bus on the PXI Express chassis backplane. To export to or import from a particular PXI_Trig line, the corresponding line will have to be configured for input or output, respectively, through the *RTSI_Trig_Direction_Register*.

The specific signal—for example: AI_START1, PFI 0—to be exported to or imported from a PXI_Trig line can be specified through the *RTSI_OutputSelectRegister* array.

Frequency Out (FOUT)

The DAQ-STC3 has a digital pulse-train generator that can be programmed through the FOUT register.

Analog Trigger Control

Hardware-timed AI acquisitions on some X Series devices can be triggered based off an analog signal on either one of the AI channels or on one of the APFI lines.

The analog trigger control circuitry on X Series devices is shown in Figure 1-8.

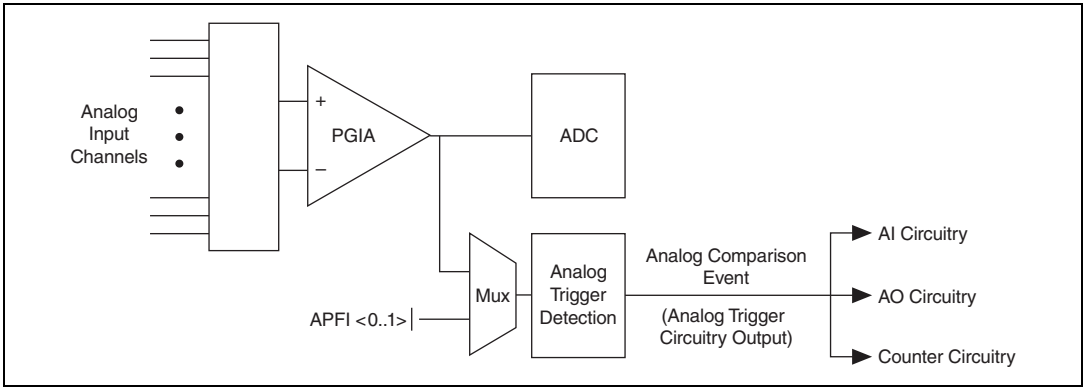


Figure 1-8. Analog Trigger Control Circuitry Block Diagram

The analog trigger detection logic on X Series devices uses two digital inputs into the DAQ-STC3. These digital inputs are sourced by the outputs of two comparators. The references to these two comparators are provided by a pair of trimmer DACs, which are programmable through the *Watchdog Timer Registers* in Chapter 9, *Board Services Registers*.

The inputs to the comparators are selected by programming the `Atrig_Sel` field of the *AnalogTrigControlRegister*. While one comparator receives the high reference voltage level, the other receives the low voltage reference signals. These high and low reference signals are sufficient to realize window and level with hysteresis triggers. The low reference voltage is generated by writing to the *Gen_PWM*[0] register, and

the high reference voltage is generated by writing the corresponding duty cycle to the *Gen_PWM*[1] register.

For a given high and low voltage level, the corresponding PWM duty-cycle setting is obtained through the following conversion:

$$16\text{-bit pwm duty-cycle setting} = (\text{desired voltage level} - \text{min board range}) \times \frac{2^{16}}{(\text{max board range} - \text{min board range})}$$

The analog trigger mode can be specified by programming the `Analog_Trigger_Mode` field of the *AnalogTrigControlRegister*. Strobe the `Analog_Trigger_Reset` bit on the *AnalogTrigControlRegister* before doing a level trigger with hysteresis.

Phase-Locked Loop (PLL)

The DAQ-STC3 contains a PLL that can be employed to lock into a Reference Clock In signal. To select the source signal for the PLL and to use the PLL output signal as TB3, the *Clock_And_Fout2_Register* can be programmed.

The *PLL_Control_Register* can be programmed for the following purposes:

- Enable or disable the PLL
- Configure the PLL loop filter to improve jitter performance
- Control the rate of the PLL input, voltage controlled oscillator (VCO) output, and PLL feedback path

The *PLL_Status_Register* can be read to determine whether the PLL has successfully locked on to the input signal after being enabled.

Refer to the `pllHelper` in the DDK for programming considerations.

Internal Trigger Filters

The IntTrigA registers, found in the *Triggers and Timing Registers* section in Chapter 11, *Triggers and Timing Registers*, can be employed for filtering of a variety of onboard signals. There are a total of eight trigger filters that can be independently configured. The *IntTrigA_FIlter_Register_Lo* and *IntTrigA_FIlter_Register_Hi* registers can be programmed to configure the minimum acceptable digital signal pulse-widths of the different filters. The *IntTriggerA_OutputSelect* registers can be programmed to select a specific signal to be filtered using the corresponding filter.

Direct Memory Access

The DMA architecture for X Series is different than E and M Series. The largest change is the DMA controller: the E and M Series devices use the MITE for DMA transfers, while X Series devices use a new controller, the CHInCh. The CHInCh provides a simpler interface for configuring and controlling DMA transfers.

The premise of the CHInCh architecture is that parsing the DMA linked list, not bus-mastering, is complex. To separate these two tasks, a new subcomponent has been added to each subsystem, the Stream Circuit. The Stream Circuit initiates and throttles DMA transactions and is oblivious to the host address location(s) associated with the transaction. It is simply accessing a single data stream. The CHInCh DMA engine then responds by forwarding the transaction to host address space on the host bus at location(s) determined by the associated DMA linked list.

Just as each subsystem has its own dedicated Stream Circuit, the CHInCh provides a DMA channel for each Stream Circuit. Working together, the Stream Circuit and CHInCh DMA channel controller implement efficient and simple data transfers. The CHInCh fetches and parses the DMA linked list from the host. Using the list, it sends or retrieves the data from the host when prompted by the Stream Circuit. The Stream Circuit then pulls or pushes the data to its subsystem's data FIFO.

DMA Modes

The CHInCh offers the following DMA memory transfer modes:

- Normal mode DMA transfers data to or from ascending address locations in host memory. A starting address is programmed to the *Channel_Memory_Address_Register* (CHMAR) prior to starting the operation. The Stream Circuit is responsible for limiting the operation to the desired amount of data.
- Link chaining mode DMA supports scatter-gather operations in host memory. The CHInCh processes a linked list structure made of chunky links from host memory that describes the buffer. The linked list can be circular or it can end so that a specific amount of data is transferred. The address location and size of the first chunky link are programmed prior to starting the operation in the *Channel_Link_Address_Register* (CHLAR) and *Channel_Link_Size_Register* (CHLSR).

Chunky Link Format

Figure 1-9 describes the format of the DMA linked list used by the CHInCh.

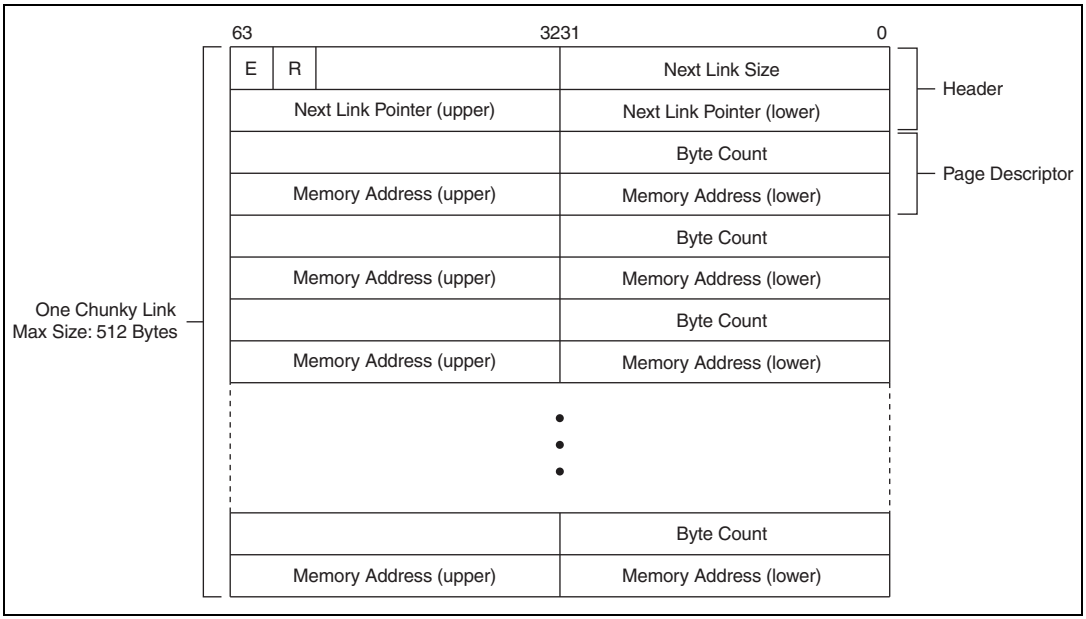


Figure 1-9. Chunky Link Format

Chunky links are limited in size to a maximum number of 512 Bytes each.

The "upper" next link pointer field is only used when the `DMA_LA64` bit in the *Host_Bus_Resource_Control_Register* (HBRCR) is set. The "upper" memory address field is only used when the `DMA_MA64` bit in the HBRCR is set. When these fields are not used, they are reserved and must be zero.

The next link size field indicates the size in Bytes of the next chunky link. The size must be a multiple of eight. The next link pointer field provides the base address of the next chunky link. The base address of a chunky link must be naturally aligned on an 8-Byte boundary.

Each field of the chunky link is assumed to be in little-Endian format.

If bit 63 of the first 8-Byte word (the "E" bit in Figure 1-9) is set, this is the final chunky link in the host memory description. The DMA controller will process the page descriptors in the chunky link that has this bit set but will not fetch any more chunky links.

If bit 62 of the first 8-Byte word (the “R” bit in Figure 1-9) is set, it causes the CHInCh to reuse the linked list rather than fetch more chunky links. It is only legal to set the “R” bit in the second chunky link when the entire linked list contains exactly two chunky links. The result is that the two chunky links will be fetched only once and then repeatedly used to create a circular scatter-gather buffer in system memory. This eliminates the bus bandwidth usage that would normally be required to fetch chunky links.

When using this feature, the next link size field of the second chunky link (the one that has the “R” bit set) must indicate the size of the first chunky link as would be done with a circular linked list. It is illegal to set both the “E” and “R” bits in the same chunky link.

Each chunky link must include at least one page descriptor.

The Byte count field indicates the amount of data to be transferred within the page descriptor. The Byte count field cannot be 0, but any other value is supported. The memory address field provides the base address of the page descriptor in system memory. The memory address can be any value and has no alignment requirements.

DMA Progress Status

DMA progress status information is defined to reflect transfer progress in host memory. In general, progress status is ordered such that it safely reflects the amount of input data available in host memory or output data that has been read from host memory. In other words, software can process input data in host memory or overwrite output data in host memory based on the status information received (the status is coherent).

Readable Status Registers

The only readable registers guaranteed to provide coherent DMA progress status are the 64-bit *Channel_Total_Transfer_Count_Status_Register* (CHTTCSR) and *Channel_Total_Transfer_Count_Latching_Register* (CHTTCLR). For DMA writes to the host, the register only updates once DMA data has made it to a point in the data path in which it is guaranteed to be sent on the host bus before the status read completes on the host bus. Bus ordering rules then ensure coherency throughout the rest of the system.

During DMA reads from the host, the register updates once the associated DMA data has made it to a point in the data path in which it is guaranteed to be sent to the Stream Circuit before any subsequent register access transactions. This timing provides two guarantees for output applications:

- DMA data has been received from the host bus and software can safely overwrite it.
- StreamCircuit receives the DMA data before any register accesses that software might initiate in response to the progress status.

For example, software could write to a Stream Circuit register only after a certain amount of stream data has been sent to it.

The latching version of the register can be read using two 32-bit accesses without concern that the value will change between accesses. This provides support for processors or host buses that cannot perform 64-bit accesses. Note that using two accesses requires the register to be read from a single software thread. Multiple software threads cannot check the status of the same DMA channel unless they can do so with 64-bit accesses to the *Channel_Total_Transfer_Count_Status_Register* (CHTTCSR). The register represents an ample amount of time so that mechanisms to track rollovers should not be necessary.

DMA Progress Control

During normal mode operation, the Stream Circuit controls the amount of data transferred with the *Stream_Additive_Transfer_Count_Register* (SATCR). Writing to this register grants more data allowance for the Stream Circuit to interact with CHInCh DMA channel.

During link chain mode, the buffers described by the linked list control the amount of data transferred. If the tail of the list has the end-of-list bit set, then the CHInCh will transfer a finite amount of data. Conversely, if the tail of the list points to the head, then the CHInCh will continuously transfer data as long as the Stream Circuit continues requesting or providing data. Controlling DMA progress by manipulating the linked list while the DMA channel is active is not recommended. DMA links are pre-fetched in such a way that software cannot know which ones have been fetched and which have not. Rather, the DMA channel should be stopped, the linked list reconfigured, and then the DMA channel restarted.

DMA Interrupts

Each DMA channel controller in the CHInCh can generate an interrupt on several different conditions:

- When errors are encountered during DMA operations.
- When the last link of a link chain has been fully processed.
- When the total Byte count surpasses the value written to *Channel_Total_Transfer_Count_Compare_Register* CHTTCCR.
- When the associated subsystem reports that its operation has completed using its done event, or when software sets the `STOP` bit of the *Channel_Operation_Register* (CHOR).

Interrupts are enabled/disabled by setting the appropriate fields of the *Channel_Control_Register* (CHCR), and they are acknowledged by reading the *Channel_Volatile_Status_Register* (CHVSR) or by writing to the CHOR and the CHCR.

Initializing the DMA Channels

To initialize the `START` bit to 0 for each DMA channel, first write a 0 to the `Notify_On_Done` bit and program the `MODE` field to Normal in the *Channel_Control_Register* (CHCR), then write a 1 to the `STOP` bit of the *Channel_Operation_Register* (CHOR). Clearing the `Notify_On_Done` bit is necessary before writing a 1 to the `STOP` bit to prevent a false Done interrupt from occurring. Clearing `Notify_On_Done` will prevent the false interrupt, but the `Done` status bit may still set in the *Channel_Status_Register* (CHSR) when the `STOP` bit is written with a 1. This is not a problem since the `Done` status bit will clear itself when the DMA channel is started. When starting an operation on the channel, a false interrupt will not be generated if the `Notify_On_Done` bit is written with a 1 while the `Done` status bit is already 1. Notice that even if a channel powers up in the started state, link fetching will not occur as this requires a rising edge on the `START` bit.

Re-Starting a DMA Channel

After a DMA operation has been stopped, care must be taken when freeing system memory buffers or starting a new operation using the same DMA channel. The `Error`, `Last_Link`, and `Done` status bits in the *Channel_Status_Register* (CHSR) each indicate that transfers to or from host memory for a DMA channel are guaranteed to have stopped. This includes link fetches and is sufficient for software to free any memory buffers associated with the DMA operation and linked-list storage.

Each of these three status bits also indicates that the DMA channel in the CHInCh is ready to be programmed for a new operation. However, stream data transfers for the DMA channel could still be progressing through FIFOs or pipeline stages in the Stream Circuit or in the CHInCh. If the DMA channel were re-started, any such data transfers would be considered part of the new operation by the CHInCh. Software should ensure that all data from a previous operation has been flushed prior to starting a new operation with the same DMA channel.

The requirement to wait for one of the three status bits before starting a new operation does not apply to the first use of the DMA channel since the bits reset to unknown values. It is expected that software would be constructed to wait for one of the three bits as part of the DMA shut down procedure. This is likely necessary in order to free all memory buffers associated with the DMA operation. Since waiting is part of the shutdown procedure, it is not necessary to check the status bits as part of the start up procedure. This naturally addresses the difference with the first use since the start up procedure will be executed first after a reset.

DMA Register Programming Efficiency

Note that registers in the DMA Register Group have been strategically arranged to support efficient setup of DMA operations. The registers are arranged such that they can be programmed in ascending address order. This would allow a DMA controller to be set up with a single packet or burst transfer, thereby improving performance on high latency packet-based buses. Key to this point is that the *Channel_Operation_Register* (CHOR) is located at the highest offset of all the control registers. The CHOR would normally be written after all other registers have been initialized. In some modes of DMA operation, it is not necessary to initialize some of the control registers. In these cases, writing to such registers is still allowed and the setup of a DMA operation can still be performed in a single packet by including unnecessary writes to these registers.

Example: Link Chain Mode

The following list demonstrates the order in which registers could be programmed to initiate a link chaining mode DMA operation.

1. *Channel_Link_Address_Register* (CHLAR)
2. *Channel_Link_Size_Register* (CHLSR)
3. *Channel_Control_Register* (CHCR)
4. *Channel_Operation_Register* (CHOR)

Example: Normal Mode

The following list illustrates the order that could be used to initiate a normal mode DMA operation.

1. *Channel_Memory_Address_Register* (CHMAR)
2. Reserved offsets 40–47 (hex)—Not necessary, but write 0s
3. *Channel_Link_Address_Register* (CHLAR)—Not necessary
4. *Channel_Link_Size_Register* (CHLSR)—Not necessary
5. *Channel_Control_Register* (CHCR)
6. *Channel_Operation_Register* (CHOR)

Interrupts

The interrupt architecture for X Series devices is different than E and M Series. The largest change is the interrupt routing: the E and M Series devices raise interrupts via the MITE while X Series devices use the CHInCh. The CHInCh provides a simpler interface for enabling, acknowledging, and disabling interrupts.

The CHInCh routes all of the interrupts generated by an X Series device. Since the CHInCh contains the logic for all of the DMA controllers, it also directly controls their interrupts. Each subsystem on the DAQ-STC3, however, defines and generates its own interrupts, and so the CHInCh then routes those interrupts to the host CPU. The register interface separates DMA interrupts from all DAQ-STC3 interrupts and provides increasingly granular control over the interrupts: starting from using one bit to enable/disable all device interrupts, to enabling/disabling every subsystem's interrupts, to enabling/disabling an individual subsystem's interrupts, to enabling/disabling specific DMA and subsystem interrupts.

CHInCh Interrupt Control Registers

The CHInCh uses two registers for device interrupt routing and control: the *Interrupt_Mask_Register* (IMR), and the register pair *Interrupt_Status_Register* (ISR) and *Volatile_Interrupt_Status_Register* (VISR). The IMR provides device-wide interrupt control, providing a global enable/disable bit field for the entire device as well as an enable/disable bit field for the DAQ-STC3. The ISR and VISR report the source of the highest-priority interrupt on the device as either a DMA interrupt or a DAQ-STC3 interrupt.

If the current interrupt is from a DMA channel, then the CHInCh automatically redirects the (V)ISR register access to the DMA channel's

associated status register (CHISR or CHVISR) and returns that register's contents instead of its own. When the volatile register is read, it also has the side effect of acknowledging that channel's interrupts and clearing its interrupts' flags.

DMA Channel Interrupt Control Registers

Each DMA channel controller uses three registers for DMA interrupt control: the *Channel_Control_Register* (CHCR), the *Channel_Operation_Register* (CHOR), and the register pair *Channel_Status_Register* (CHSR) and *Channel_Volatile_Status_Register* (CHVSR). The CHCR provides bit fields to enable/disable DMA channel interrupts: on error, on last link, on total Byte count, and on done. Refer to the *DMA Interrupts* section for further information on DMA interrupt conditions. The CHOR provides a control bit field to arm the total Byte count interrupt. The CHSR and CHVSR report which channel raised the interrupt as well as the interrupt conditions for that DMA channel. When the volatile register is read, it also has the side effect of acknowledging that channel's interrupts and clearing its interrupts' flags.

DAQ-STC3 Interrupt Control Registers

The DAQ-STC3 has several register sets for interrupt control: some offer ASIC-wide control and others are specific to each subsystem. The ASIC-wide registers are in the BusInterface chip object, and they provide bit fields to enable/disable all of the interrupts associated with each subsystem as well as bit fields to determine which subsystem is interrupting and which specific interrupts have been raised.

In addition to the ASIC-wide registers, each of the eight subsystems have their own interrupt control registers. The registers are called Interrupt_1_Register and Interrupt_2_Register; the interrupt registers for the analog and digital subsystems are in their associated InTimer or OutTimer chip objects, and the interrupt registers for each counter subsystem are in its Counter chip object. Each Interrupt_1_Register has bit fields for enabling and acknowledging individual interrupts for its corresponding subsystem, while the Interrupt_2_Register has bit fields for disabling and acknowledging interrupts.

If the X Series device is a simultaneous MIO device (SMIO), then the AI subsystem interrupt registers are augmented with two more registers that control the AI FIFO interrupts. These registers are in the *SimultaneousControl* chip object and are called *InterruptControl* and *InterruptStatus*, and provide bit fields for enabling, disabling, and acknowledging the AI FIFO interrupt.

The Watchdog Timer (WDT) and PLL circuits can also generate interrupts, and their control registers are in the *BrdServices* chip object as *Gen_Interrupt1_Register* and *Gen_Interrupt2_Register*. Like the subsystem registers, the first provides bit fields for enabling and acknowledging interrupts while the second provides bit fields for disabling and acknowledging interrupts.

Enabling Interrupts

In order for any interrupt to signal the CPU, the CHInCh must be configured to assert interrupts on the host CPU. Further register programming then selects which interrupts to enable: DMA channel, subsystem events, or device events (like WDT or PLL). Typically, when first programming the hardware after booting, the safest way to enable interrupts is by first disabling all of them and then, when under general use, selectively enabling specific interrupts that apply to current operation. The order of programming in this situation would be:

1. OS load.
2. Disable and acknowledge all interrupts.
   a. Strobe the bit fields `Clear_CPU_Int`, `Clear_STC3_Int`, and, if an SMIO device, `Clear_SMIO_FIFO_Int` in the CHInCh IMR, which blocks any device interrupt from signaling the CPU.
   b. Clear `Notify_On_Error`, `Notify_On_Last_Link`, `Notify_On_Total_Count`, and `Notify_On_Done` in each DMA channel's CHCR, which acknowledges and disables all interrupts for that channel. Refer to the *Initializing the DMA Channels* section for additional information about DMA channel initialization.
   c. Strobe every bit field in the Interrupt_2_Register belonging to each subsystem, which acknowledges and disables all interrupts for that subsystem.
3. Register interrupt handlers with the operating system.
4. Enable specific interrupts.
   a. Strobe `Set_CPU_Int` in the CHInCh IMR to enable interrupt signaling from the multiplexed X Series device.
   b. For each DMA channel in the operation, enable the relevant interrupts needed by the operation in each DMA channel's CHCR.
   c. For each subsystem in the operation, enable the relevant interrupts needed by the operation in each subsystem's Interrupt_1_Register.

Acknowledging Interrupts

Once interrupts have been enabled, the device will assert interrupts as their conditions are met. The interrupt handler(s) then need to determine which interrupt(s) have requested service. By starting at the device-wide registers, the interrupt handler can determine which interrupt is requesting service in four or fewer accesses. The order of programming in this situation would be as follows:

1. Receive X Series device interrupt.
2. Determine which X Series device is interrupting.
3. Read that device's CHInCh (V)ISR.
   a. If the `DMA` bit field is asserted, then the read returned the DMA channel's CH(V)SR. If the volatile register was read, all of the signaling DMA channel interrupts were acknowledged in the read; if not, then explicitly acknowledge the interrupts (refer to their bit field descriptions).
   b. If the `DMA` bit is not asserted, the `STC3_Int` or `SMIO_FIFO_Int` bitfield indicates which register to read next.
4. If the `SMIO_FIFO_Int` bit is asserted, read the SimultaneousControl *InterruptStatus* register to verify the interrupt condition and write to the SimultaneousControl *InterruptControl* register to acknowledge the interrupt.
5. If the `STC3_Int` bit is asserted, read the BusInterface *GlobalInterruptStatus_Register* to determine which subsystem(s) have requested service. Then read each asserting subsystem's Interrupt_1_Register to determine which interrupts have requested service. Write to the same register to acknowledge them.
6. If global or specific interrupts were disabled during the interrupt service routine, as is common in many designs, re-enable them once they've been fully serviced.

Disabling Interrupts

Often during an interrupt service routine, it is necessary to prevent subsequent interrupts of the same kind from asserting, and so the interrupt handler disables the interrupts it is currently servicing. Also, when a subsystem or DMA channel is not in use, it is good practice to disable those corresponding interrupts. The logic of programming in such situations would be as follows:

1. To disable all interrupts, strobe the `Clear_CPU_Int` bit-field in the CHInCh's IMR.
2. To disable all DAQ-STC3 interrupts, strobe the `Clear_SMIO_FIFO_Int` and `Clear_STC3_Int` bit-fields in the CHInCh's IMR.
3. To disable DMA interrupts, strobe the relevant bit fields in the channel's CHCR.
4. To disable all interrupts from a subsystem, use the relevant bit fields in the BusInterface's *GlobalInterruptEnable_Register*.
5. To disable specific subsystem interrupts, use the relevant bit fields in the Interrupt_2_Register.

Special Considerations: Maximizing Throughput in Low-Latency Situations

Some applications need to perform a measurement, calculate a new set point, and generate a control signal. Typically control loops, these applications require single-point input and output and thus value low-latency over data streaming to minimize their loop rates. Using interrupts for event notification and device control and using DMA for data transfer provides the most responsive control, maximizes data transport rates, and maximizes the remaining CPU availability to perform analysis.

For analog input operations, the hardware generates a STOP signal to alert the timing engine that a scan has completed. The AI subsystem can also generate an interrupt on this event and does so on the rising edge of the STOP signal. In E and M Series devices, this interrupt was useful for hardware-timed single-point measurements when the control loop application needed to be notified for every new sample. Upon receiving this interrupt, the driver would then query the MITE to determine if that data was available because when the STOP interrupt asserted, the data was not guaranteed to have been transferred to the host. This technique consumed CPU resources as the interrupt handler polled the hardware.

Fortunately for X Series devices, the CHInCh can interrupt on the DMA channel's total transfer count, which occurs once the data has been completely transferred to the host memory. Rather than using the STOP interrupt, if an interrupt handler uses the total transfer count interrupt, it is certain that the new data is immediately available upon receiving the interrupt. The order of programming for this situation (and output operations) is as follows:

1. Program the DMA channel's *Total_Transfer_Count_Compare_Register* (CHTTCCR) with the number of Bytes in a single input/output sample.
2. Set the DMA channel's `Notify on Total Count` flag in the CHCR.
3. Set the DMA channel's `Arm Total Count Interrupt` flag in the CHOR.
4. Start data transfer (through the DMA controller and the subsystem's Stream Circuit).
5. Receive total transfer count interrupt.
6. Increase the CHTTCCR by the number of Bytes in a single input/output sample.
7. Re-arm the total transfer count interrupt in the CHOR.

The latency and throughput balance can be adjusted by using the Stream Circuit's *StreamTransferLimitReg* and the *StreamEvictionReg* for input operations, or by using the *StreamTransactionLimitReg* for output operations. Data transfer can also be throttled in both directions with the *StreamAdditiveTransferCountReg*.

2

Programming Considerations

The Measurements Hardware Driver Development Kit (MHDDK) demonstrates how applications access the X Series hardware subsystems and registers to perform measurements on operating systems not supported by NI-DAQmx. Performing the same register accesses in the same order as an MHDDK example program allows your application/driver to perform the same measurements that are demonstrated in the example program. However, the MHDDK should not be used as a reference architecture for an OS driver. The methods used for accessing the hardware may circumvent operating system security by allowing user-mode applications to directly access hardware registers.

This chapter contains information on *Parts of the MHDDK*, *Initialization*, *EEPROM*, and *Register Bit Maps*.

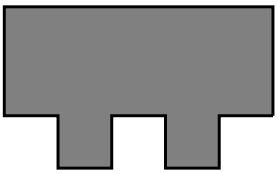
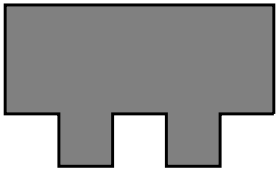
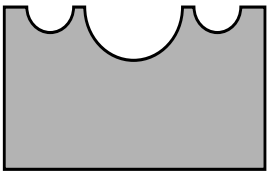
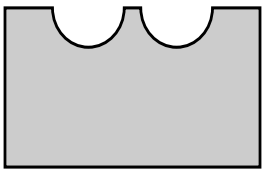
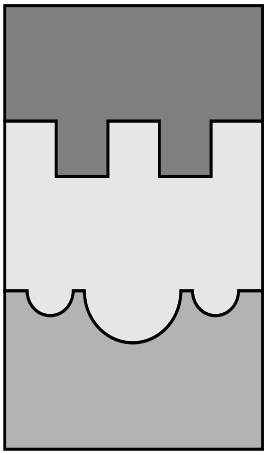
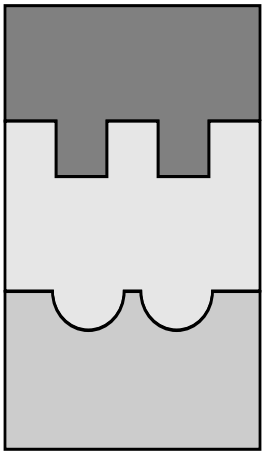
Parts of the MHDDK

The MHDDK is divided into two main layers:

- Operating System Interface[1]—Adapts the hardware support layer to the host OS (QNX, for example)
- Hardware Support—Implements a single device family (X Series, for example)

[1] DMA Interface—Auxiliary OS interface component for DMA I/O on some hardware families (X Series, for example).

Table 2-1. OS Interface and Hardware Support Layers

MHDDK Layer	Windows	Linux
The Hardware Support Layer does not care which operating system is hosting it, and so has a well-defined interface, and expects to have certain calls available to interact with the devices it supports.		
Each operating system implements its own system and kernel calls.		
The operating system Interface layer comes in to match the Hardware Support layer with the operating system kernel.		

Operating System Interface

This component's purpose is to utilize the host OS API to match the upper layer (Hardware Support) to the operating system. Most of the code is generalized for any OS, but when interacting with hardware registers, two key functions are stubbed for platform-specific operations.

Common Files to All Operating Systems

These source files are the same for every OS that the DDK supports. They get compiled into the example programs and run in user-mode (or single-mode if the OS does not have user-mode).

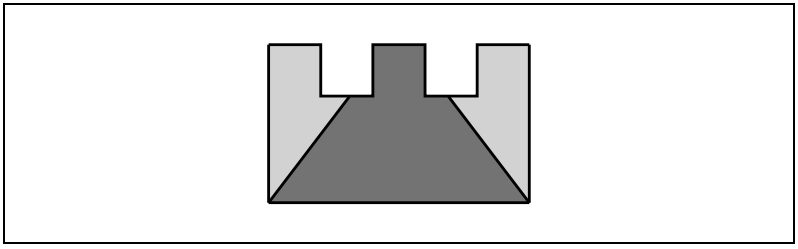


Figure 2-1. Common Operating System Layer

The `osiTypes.h` file contains the basic data types for the target platform (i32, u32, and so on) and macros for converting Endianness.

The `osiBus.h` and `osiBus.cpp` files define three classes:

- `iBus`—Represents a DAQ device
- `tAddressSpace`—Represents an address space (of registers) on the DAQ device
- `tDMAMemory`—Represents a block of DMA memory

They also declare two OS-specific functions:

- `iBus* acquireBoard(tChar *brdLocation)`—Probes the OS for NI hardware and initializes it for programming
- `void releaseBoard(iBus *&busObject)`—Returns resources back to the OS

These two functions are the pieces that know about the specifics of the host OS and are the only pieces that need to be implemented when adding new operating systems to the DDK. This is how code repetition is minimized.

OS-Specific Files

These source files are different for every OS that the DDK supports. They know about the host OS and can do all of the maintenance necessary for preparing a device for programming.

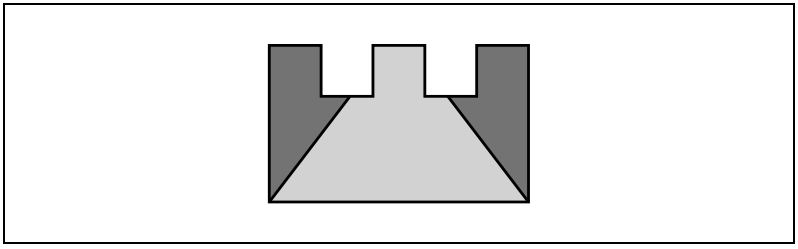


Figure 2-2. Specific OS Layer

The `osiUserCode.cpp` file defines the two function declarations made in `osiBus.h` and may also implement DMA support. This file then gets compiled into the example programs and runs in user-mode (or single-mode).

- `acquireBoard()`—This function performs all of the OS-specific operations needed to prepare a DAQ device for register-level programming. Specifically, there are three tasks that must be completed before RLP can start:
  1. Find and open the device.
  2. Map the registers to (user/single mode) memory.
  3. Create and return the iBus object.
- `releaseBoard()`—This function unwinds all of the setup operations that `acquireBoard()` performed. Namely:
  1. Unmap the registers from memory.
  2. Destroy the iBus object.

Kernel Components

Linux 2.6

This Linux implementation is in the form of a kernel module: `nirlpk.ko`. It follows the recommended module structure as outlined in Linux Device Drivers and implements a `/dev/nirlpkN` entry, DMA buffer allocation, and a `/proc/nirlpk` file tree.

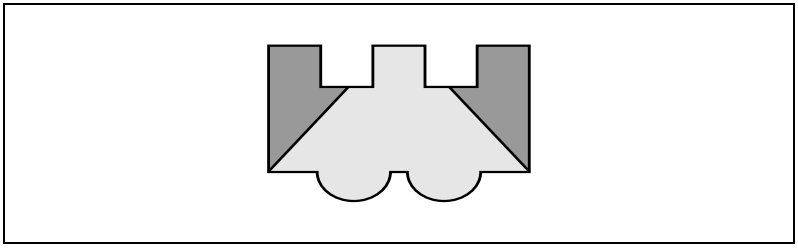


Figure 2-3. Linux 2.6 Kernel Module

Windows WDM

This Windows implementation uses the Windows Driver Model to register NI hardware with the OS.

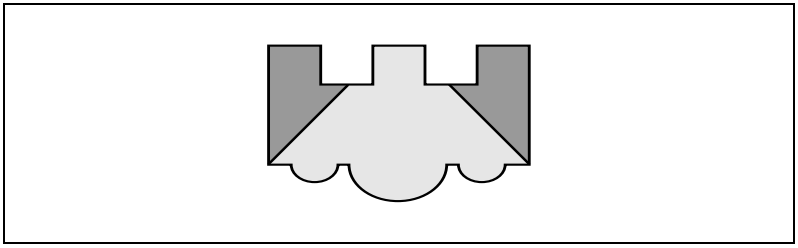


Figure 2-4. Windows WDM Driver

DMA Interface

This component's job is to add DMA support to the OS Interface layer. Most of the code is generalized for any OS. However, when interacting with memory allocation, one class is sub-classed and two methods are implemented to fill in the details.

Common Files to All Operating Systems

These source files are the same for every OS that the DDK supports. They are compiled into the example programs and run in user-mode (or single-mode if the OS does not have user-mode).

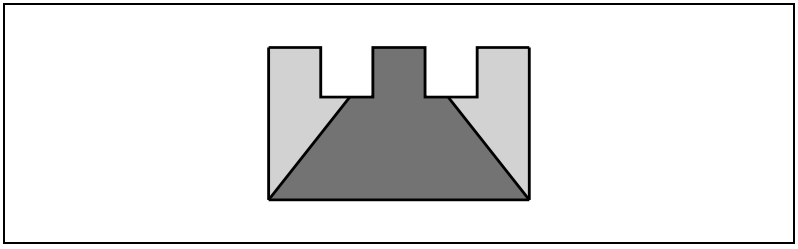


Figure 2-5. Common OS Layer

Part of the DMA component lives in the OS Interface layer. In addition to declaring a generic parent class for DMA memory management (`tDMAMemory`), the `osiBus.h` file also declares two methods in the `iBus` class for interacting with `tDMAMemory`:

- **`tDMAMemory * iBus::allocDMA (u32 size)`**—Responsible for creating a `tDMAMemory` object that allocates a DMA buffer.
- **`void iBus::freeDMA (tDMAMemory *mem)`**—Responsible for cleaning up the `tDMAMemory` object.

This class and two methods are the pieces that know about the specifics of the host OS. They are the only pieces that need to be implemented when adding DMA support to the OS Interface layer. This is how code repetition is minimized.

DMA Objects

- **`tCHInChDMAChannel`**—Object used by the example programs to allocate and initialize the buffer to be used for DMA, configure the CHInCh for DMA, and to provide access for the example program to send/receive the data in the buffer. The CHInCh is the PCI Express interface of the X Series devices. One of its duties includes DMA when configured properly. The `tCHInChDMAChannel` contains a `tCHInChDMAChannelController` and a `tDMABuffer`.
- **`tCHInChDMAChannelController`**—Object used by the `tCHInChDMAChannel` object to access the registers of the CHInCh to configure, start, and stop DMA transfers.

- **tDMABuffer**—Abstract class representing the buffer used in the DMA process. The `tCHInChDMAChannel` will either create a `tLinearDMABuffer` or `tScatterGatherDMABuffer` to store the data for the DMA operation.
- **tLinearDMABuffer**—Class inherits from the `tDMABuffer` class. It is a single contiguous piece of memory that stores the data used during a DMA operation.
- **tScatterGatherDMABuffer**—Class inherits from the `tDMABuffer` class. It is also a contiguous piece of memory, but this memory is pointed to by chunky links that are used in a scatter-gather DMA operation. This object also contains a `tCHInChSGL` object.
- **tCHInChSGL**—Class is a doubly-linked list containing the chunky link nodes to be used by the scatter-gather DMA operation.
- **tCHInCHSGLChunkyLink**—Class manages a single chunky link node used in the DMA operation. It allocates its link's memory and provides an interface for configuring the link. The link has a physical address and size describing its memory in the DMA buffer.

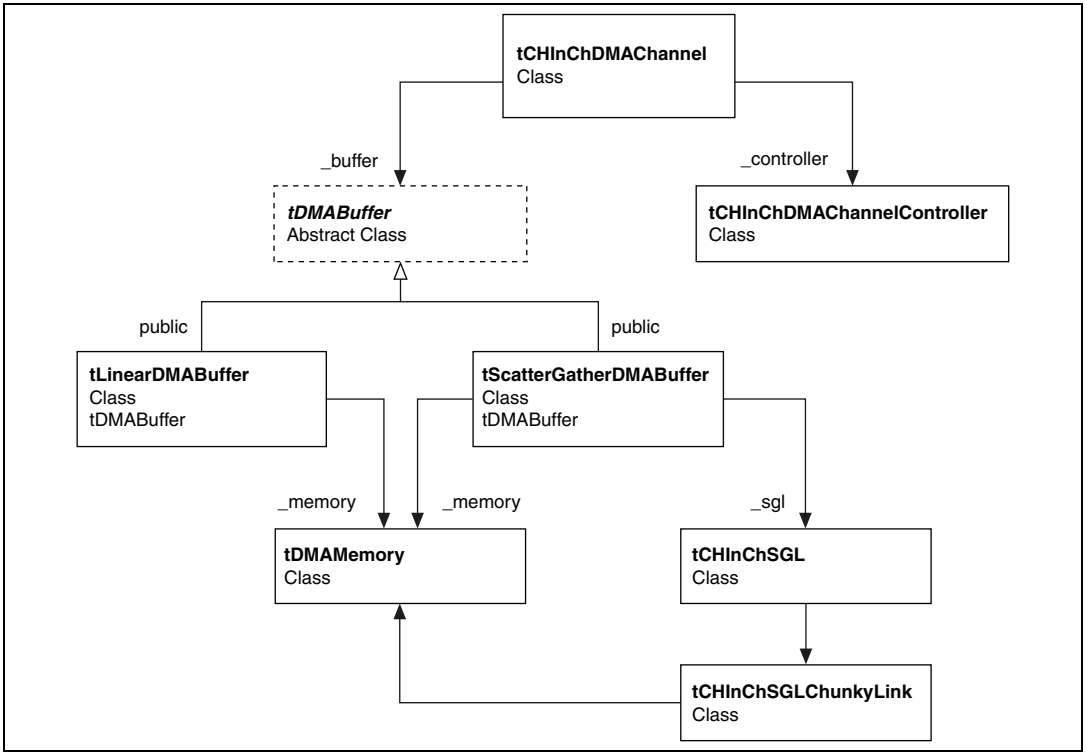


Figure 2-6. Class Diagram for DMA Objects

Operating System-Specific Files

These source files are different for every OS that the DDK supports. The source files know about the host OS and can do all of the specific housekeeping necessary for device DMA.

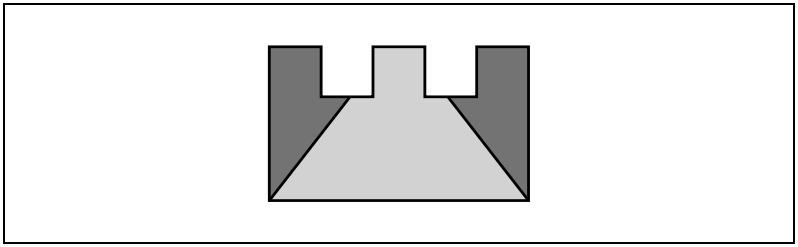


Figure 2-7. Specific Operating System Layer

In addition to implementing device detection (as discussed on the previous page), the `osiUserCode.cpp` file also implements the DMA-related `iBus` methods and OS-specific `tDMAMemory` subclass. This file is compiled into the example programs and runs in user-mode (or single-mode).

- **`tDMAMemory * iBus::allocDMA (u32 size)`**—Method performs all of the OS-specific methods needed to allocate a DMA memory buffer for the device. Specifically, there are three tasks:
  - Allocate a DMA buffer (perhaps with help from a kernel-level component).
  - Map the DMA buffer to user/single mode memory.
  - Create the OS-specific `tDMAMemory` subclass and return it as its parent.
- **`void iBus::freeDMA (tDMAMemory *mem)`**—Function unwinds the DMA setup operations performed by **`iBus::allocDMA()`** (perhaps through the destructor from the `tDMAMemory` subclass). Namely:
  - Unmap the DMA buffer from user/single mode memory.
  - Free the DMA buffer.

Hardware Support

The primary focus of the X Series DDK is the Hardware Support component. The hardware component is split into two parts:

- *Chip Objects*—The Chip Object classes encapsulate the subsystems of the hardware, and provide an API for peeking and poking their registers.

- *Examples*—These programs show how to program the registers for different measurement applications. They demonstrate which bit-fields to use and in which order to perform a given task.

Chip Objects

The X Series chip objects represent parts of the device including specific subsystems and ASICs. They provide an API for accessing the bit-fields and registers of each subsystem. These objects are linked with the example program, so they run in user-mode (or single-mode).

- **tAI**—Represents the analog input subsystem including all of the registers associated with analog input as well as a `tInTimer` object. Use this object to configure analog input measurements.
- **tAO**—Represents the analog output subsystem including all of the registers associated with analog output as well as a `tOutTimer` object. Use this object to configure analog output operations.
- **tBrdServices**—Represents the BrdServices subsystem. It can be used to perform basic sanity tests when accessing the hardware registers because it includes a scratchpad register (read/write) and DAQ-STC3 chip signature register.
- **tBusInterface**—Represents the Interrupt Status Registers associated with each subsystem. While the MHDDK does not demonstrate the use of interrupts with X Series devices, these registers can be used when creating drivers that take advantage of interrupts.
- **tCHInCh**—Represents the PCI Express interface on the X Series devices. Use this object to perform global operations on the device: identification, enabling/disabling interrupts, setting 32-bit or 64-bit addressing mode.
- **tCounter**—Represents the counter subsystems available on the X Series devices. Use this object to configure counter input and output operations.
- **tDI**—Represents the DIO Port 0 input registers as well as the associated `tInTimer` object. Use this object to configure input operations, including timed input, on DIO Port 0.
- **tDMAController**—Represents the DMA channels available on the X Series objects, one for each subsystem. Use this object to control DMA transfers and enable/disable their associated interrupts.
- **tDO**—Represents the DIO Port 0 output registers as well as the associated `tOutTimer` object. Use this object to configure output operations, including timed output, on DIO Port 0.

- **tInTimer**—Represents the input timing subsystems associated with each input subsystem. Use this object to configure the FIFO and timing characteristics of the associated subsystems.
- **tOutTimer**—Represents the output timing subsystems associated with each output subsystem. Use this object to configure the FIFO and timing characteristics of the associated subsystems.
- **tSimultaneousControl**—Used for controlling devices with multiple analog to digital converters.
- **tStreamCircuitRegMap**—Represents the streaming circuitry associated with each subsystem. Use these objects to configure the associated subsystem for operations involving DMA.
- **tTriggers**—Represents the triggers and timing subsystem registers. Use this object to configure the importing and exporting of signals on the RTSI and PFI lines, analog triggers, and digital filters.
- **tXSeries**—Contains all of the subsystems associated with an X Series device.

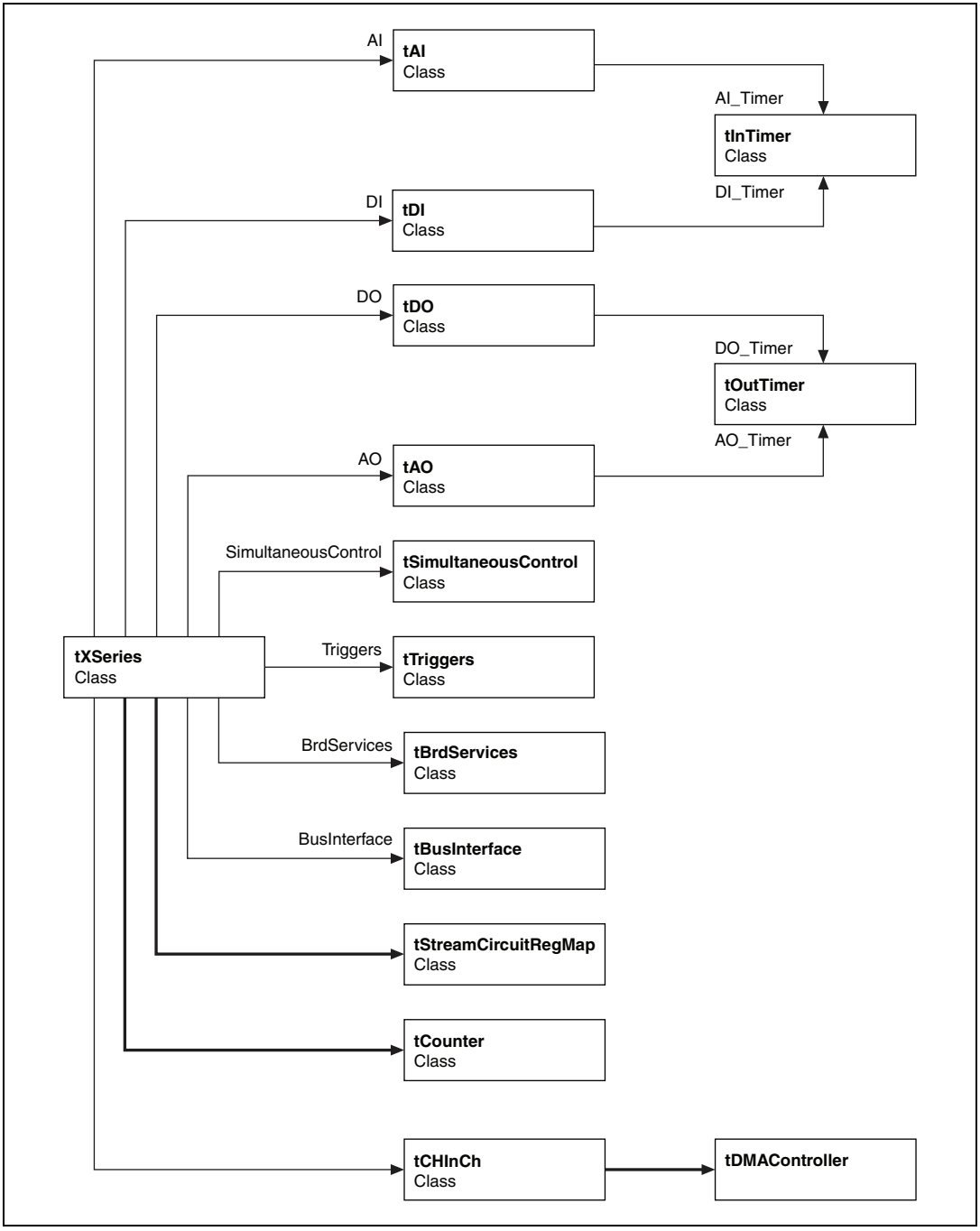


Figure 2-8. Class Diagram for Chip Objects

Examples

The example programs, along with the helper classes, tie all of the DDK code together. The example programs use the chip objects, the OS interface, and the DMA classes to perform measurement operations. Methods from the helper classes are used by the examples to modify multiple bitfields or perform other closely related and common operations on the chip object registers. Some helper objects, such as `tAiHelper`, also contain a `tInTimerHelper` to access registers in the `tInTimer` object associated with the analog input subsystem.

For example, the analog input examples use a `tXSeries` object along with the `tAiHelper` and `tInTimerHelper` to configure the analog input measurements. The `tAiHelper` makes use of the `tSimultaneousControl` and `tAI` objects on the X Series device.

Initialization

While your system is powered off, insert your PCI Express or PXI Express X Series device into an empty PCI Express or PXI Express slot. After powering up your system, the X Series device should be detected by your operating system.

The initialization required for accessing the registers on PCI Express and PXI Express X Series devices is performed in the OS Interface portion of the DDK. In order to access the registers on the board, use the `acquireBoard()` function in the OS Interface that supports your operating system. This function performs three tasks to allow access to the hardware:

1. Find and open the device.
2. Map the registers to user or single mode memory.
3. Create and return the iBus object.

The iBus object is used by the DDK to obtain a handle to the Base Address Register 0 (BAR0) address space of the device. This address space is used by multiple DDK objects to access the device's registers.

The first initialization step is to use the BAR0 address space to create the X Series object. The X Series object constructs and initializes the subsystems of the device that allow access to the device's registers.

The first memory accesses should be to verify that the device is supported in the DDK and retrieve the properties of the device. The tDeviceInfo object demonstrates how your application should access the Subsystem ID of the device and use it to determine the properties of the device such as the

name of the device, the number of analog to digital converters, the number of digital-to-analog converters, and the number of DIO lines on Port 0. Once your application has access to the properties of the device, it can continue to be initialized.

For Simultaneous MIO (SMIO) X Series devices, an additional call is necessary for initialization. The `initializeSimultaneousXSeries()` function performs actions that enable the EEPROM and simultaneous analog input.

EEPROM

The EEPROM contains useful information about a particular device including: the last time that the device was calibrated, the temperature when it was last calibrated, the voltage reference used during that calibration, and the calibration constants. From the EEPROM you can also obtain information about the geographical address (slot number if used in a PXI Express chassis) and the serial number of the device.

The calibration coefficients are used to scale the raw value received from the FIFOs of the analog subsystems into 32-bit floating point numbers.

Table 2-2. EEPROM Contents and Locations

Address	Value (32-Bits)	Description
0x000C	Capabilities List Flag Ptr	EEPROM Pointer to the location of the Capabilities Flag
0x0010	Capabilities List A Ptr	EEPROM Pointer to the start of the Capabilities List A section
0x0014	Capabilities List B Ptr	EEPROM Pointer to the start of the Capabilities List B section

Table 2-3. Capabilities Flag

Address	Value (32-bits)	Description
Capabilities List Flag Ptr	Capabilities List Flag	The value of bit 0 indicates which Capabilities List should be used by the application: 0: Capabilities List A 1: Capabilities List B

Capabilities List

The Capabilities List section consists of a linked list of capabilities nodes. Each node has an ID, a pointer (absolute or relative) to the next node and a body. The structure of the body depends on the ID. You will not find multiple nodes with the same ID in the same list and there is only one ID defined (0x0001) that will store product-specific information in the body. The Next Ptr in each node points to the next node in the link. The two least significant bits (2 LSB) are used to decide what type of pointer it is:

- 0—Absolute address, this pointer is offset from the beginning of the EEPROM address space.
- 2—Relative address, this pointer is offset from the address of the current node.

The DAQ-STC3-based MIO products will have at least three nodes in their capabilities list, as described in Tables 2-4, 2-5, and 2-6.

Table 2-4. Serial Number (ID: 0x0004)

Address	Value (32-bits)	Description
+0	0x0004 | Next Ptr	16-bit ID indicates that this node is a serial number node; the 16-bit Next Ptr is the pointer to the next node. The two LSBs of the pointer can be used to specify a 16-bit absolute or relative address.
+0x04	Serial Number	The 32-bit value is the value of the serial number. Format this number as a hexadecimal number to compare it with the serial number printed on your device.

Table 2-5. Geographical Address Information (ID: 0x4741)

Address	Value (32-bits)	Description
+0	0x4741 | Next Ptr	16-bit ID indicates that this node is a geographical addressing node; the 16-bit Next Ptr is the pointer to the next node. The two LSBs of the pointer can be used to specify a 16-bit absolute or relative address.
+0x04	GA Ptr	The Address in BAR0 memory of a register containing the geographical address bit-field.
+0x08	Shift (Mask: 0x001F0000)	Amount that the register should be right-shifted when extracting the geographical address bit-field. The geographic address bit-field is five bits wide.

Table 2-5. Geographical Address Information (ID: 0x4741) (Continued)

Address	Value (32-bits)	Description
+0x08	ReadWidth (Mask: 0x00000700)	The width of the register that contains the geographical address bit-field. ReadWidth: 1—The register is 8-bit 2—The register is 16-bit 3—The register is 32-bit
+0x08	Space (Mask: 0x0000000F)	Value is 0x1

Table 2-6. Device-Specific (ID: 0x0001)

Address	Value (32-bits)	Description
+0	0x0001 | Next Ptr	16-bit ID indicates that this node is a device specific information node; the 16-bit Next Ptr is the pointer to the next node. The two LSBs of the pointer can be used to specify a 16-bit absolute or relative address.
+0x04	Size	Size of the body of the node, in Bytes, not including this size word, for example, the size of the Body Format plus the group of ID-Value pairs plus the CRC word.
+0x08	Body Format	IDs and values can be 16 bits with the value preceding the ID in offset (value has the lower address and ID has the higher address) (format = 0x01), 32 bits in the same order (format = 0x02), 16 bits with the ID preceding the value (format = 0x03), or 32 bits with the latter order (format = 0x04).
+0x0C	Body	For information specific to the device in the form of ID-value pairs, refer to Table 2-7.
Last 32-bit word	CRC	16-bit CRC value of the whole Body Format and Body. The CRC calculation must include the Body Format but not the Size. The 16 MSBs must be padded with 0s. Since the CHInCh in the DAQ-STC3 is little endian, these 16 0s are in the last two addresses of the node.

The body of the device-specific node will start with a 32-bit format ID followed by a static list of ID-Value pairs where most of the values are pointers to store the location of the other sections such as Calibration.

The format word specifies if the IDs and values will be 16 bits (format = 0x01) or 32 bits (format = 0x02). In the case of 16 bits, the 16 LSBs correspond to the value (pointer) and the 16 MSBs correspond to the ID. Notice that since the CHInCh is little endian, the ID (16 MSBs) will have the higher address (non 32-bit aligned), and the value (16 LSBs) will have the lower address (32-bit aligned). In the case of 32 bits, the first 32-bit word (lower address) will be the ID and the following 32-bit word will be the value paired with that ID.

The decision to use the pointer to Section A or B should be made based on which pointer is not Null or 0x00. If both section pointers are non-Null, use the section with the highest WriteCount.

Table 2-7. Device Specific Node IDs

ID	Meaning	Description
0x0040	External Calibration Section A Pointer	Absolute Pointer to the External Calibration Section A.
0x0041	External Calibration Section B Pointer	Absolute Pointer to the External Calibration Section B.
0x0042	Self Calibration Section A Pointer	Absolute Pointer to the Self Calibration Section A. The self-calibration constants should be used to scale the raw data into voltages.
0x0043	Self Calibration Section B Pointer	Absolute Pointer to the Self Calibration Section B. The self-calibration constants should be used to scale the raw data into voltages.

Calibration Section

Table 2-8. Calibration Section Description

Address	Values	Description
+0	CalDataSize	CalDataSize is a 16-bit value indicating the number of Bytes in the calibration data.
+0x02	CalTimeLower	CalTimeLower is the least significant 32-bits of the 64-bit calibration time.
+0x06	CalTimeUpper	CalTimeUpper is the most significant 32-bits of the 64-bit calibration time.
+0xA	CalTemperature	CalTemperature is a 32-bit floating point number indicating the temperature at the time of calibration.
+0xE	CalVoltageRef	CalVoltageRef is a 32-bit floating point number indicating the reference voltage used during the calibration.
+0x12	AICalSection	The X Series devices have a variable number of ADCs, so this section will be repeated for each ADC on the device.
...	AOCalSection	The AOCalSection immediately follows the AICalSection. The X Series devices have a variable number of DACs, so this section will repeat for each DAC.
Last 32 bit word	CalWriteCount	This value is updated after calibrating the X Series device. If the card has two calibration sections, the one with the highest CalWriteCount has the information from the most recent calibration.

AI Cal Section

Addresses are offset from the AICalSection offset in the Calibration section.

Table 2-9. AI Calibration Section

Address	Values	Description
+0x0	AIMode	The Modes consist of a 9-bit order and four 32-bit floating point mode coefficients. There are four AI Modes in each channel.
+0x44	AI_Interval	The Intervals consist of a 32-bit floating point gain followed by a 32-bit floating point offset. There are seven AI Intervals.
Note: If a device has multiple ADCs, the AIMode and AI_Interval sections will be repeated for each ADC.		

AO Cal Section

Addresses are offset from the AOCalSection offset in the Calibration section.

Table 2-10. AO Calibration Section

Address	Values	Description
+0	AOMode	The Modes consist of an 8-bit order and four 32-bit floating point mode coefficients.
+0x11	AO_Interval	The Intervals consist of a 32-bit floating point gain followed by a 32-bit floating point offset. There are four AO Intervals.
Note: If a board has multiple DACs, the AOMode and AO_Interval sections will be repeated for each DAC.		

For more information about how to obtain and use the calibration coefficients, refer to the eepromHelper class section. Use the following formula to convert the U16 raw value to an F32 voltage.

AI:

$$Voltage = (Coefficient3 \times RawValue^3 + Coefficient2 \times RawValue^2 + Coefficient1 \times RawValue + Coefficient0) \times gain + offset$$

AO:

$$RawValue = (Coefficient1 \times Voltage + Coefficient0) \times gain + offset$$

Interfacing with the EEPROM

The eepromHelper object reads from the EEPROM by using the Window register of the CHInCh. Once a page in the EEPROM is exposed through the Window register, the eepromHelper is able to access the EEPROM data.

Register Bit Maps

A new component in National Instruments' MHDDK is a Register Bit Map (RBM) for a set of device registers. An RBM file is a plain-text file that uses a simple regular syntax to describe the layout of registers for a device, including their offsets, bitfields, and possible values for their bitfields.

Since RBMs are written in a regular syntax, they can be used as input for tailored code generators, which would then translate their high-level register map descriptions into highly optimized code for accessing hardware device registers. Generators could be written to provide C structs, C++ classes, or other data structures in other languages.

RBM File Syntax

An RBM file describes the registers, fields, and enumerated constants that interact with the device or some subset of the device. The RBM file format is a line-oriented, position-dependent format. Each line contains a variable number of whitespace-separated tokens. The first token of a line, called the discriminant, determines what information follows on the rest of the current line and possibly subsequent lines.

The remainder of this section describes the syntax and semantics of each of the discriminants.

Global Settings

These discriminants do not describe individual registers but the overall code structure that the generator should create.

—containable
—containable

The **—containable** flag indicates to the generator that the registers contained in the RBM can be contained by another register map.

—contains

—contains *name offset rbmFile headerString namespace*

The **—contains** statement indicates to the generator that another register map is part of this register map.

- **name**—The variable name that will be used in this register map to refer to the contained register map.
- **offset**—The base offset (in hexadecimal) of the contained register map relative to this register map.
- **rbmFile**—The path to the RBM file for the contained register map.
- **headerString**—The string to use for an `#include` or similar directive to include or import the contained RBM file's generated declarations, assuming the generator creates such code.
- **namespace**—The fully qualified namespace of the contained register map, assuming the generator creates namespaced code.

—generate-include

—generate-include *string*

The **—generate-include** statement instructs the generator to create an `#include` or similar directive using **string** as the parameter.

#

\# *rest*

The **#** discriminant designates a comment line. **rest** should be ignored.

@

@ *rest*

The **@** discriminant designates a documentation line. **rest** should be ignored.

E

E *name*

The **E** discriminant begins a new global enumeration with name **name**. Enumerated values are defined by subsequent uses of the `V` discriminant.

V

V *name integer*

The **V** discriminant defines an enumerated value for a global enumeration with name **name** and value **integer**.

Registers

Registers can be described as individual concrete registers with the **R** discriminant or as parameterized register templates with the **T** discriminant. Both concrete and template registers use the **F** discriminant to describe their bitfields. Register templates are instantiated with the **TRA** discriminant.

R

R *name size offset attributes* [options]

The **R** discriminant begins the definition of a new register.

- **name**—The name of the register.
- **size**—The size of the register in bits.
- **offset**—The address offset (relative to this map) of the register.
- attributes
  - **Readable**—This register is readable.
  - **Writable**—This register is writable.
  - |—The pipe character can OR attributes together.
- options
  - **-force-default [true|false]**—Indicates to the generator that writes to this register must (or must not) be executed.
  - **-initial-value** *value*—Indicates to the generator that it should initialize this register to value if it provides a reset() method.
  - **-no-hardware-reset [true|false]**—Indicates to the generator that it should (or should not) write to this register if it provides a reset() method. However, if it provides a soft copy for the register, it should modify the soft copy regardless.
  - **-no-soft-copy [true|false]**—Indicates to the generator that it should (or should not) create a soft copy of the register's value.

T

T *name size attributes* [options]

The **T** discriminant begins the definition of a new register template. Since it doesn't define a new concrete register, it does not use an offset parameter like the R discriminant.

- **name**—The name of the register.
- **size**—The size of the register in bits.

- attributes
  - **Readable**—This register is readable.
  - **Writable**—This register is writable.
  - **|**—The pipe character can OR attributes together.
- options
  - **-force-default [true|false]**—Indicates to the generator that writes to this register must (or must not) be executed.
  - **-initial-value** *value*—Indicates to the generator that it should initialize this register to **value** if it provides a `reset()` method.
  - **-no-hardware-reset [true|false]**—Indicates to the generator that it should (or should not) write to this register if it provides a `reset()` method. However, if it provides a soft copy for the register, it should modify the soft copy regardless.
  - **-no-soft-copy [true|false]**—Indicates to the generator that it should (or should not) create a soft copy of the register's value.

TRA

TRA *nameFormat typeName offset number* [instanceQualifier [options]]

The **TRA** discriminant defines a new array of registers from the same register template.

- **nameFormat**—The string format that the generator should use when creating individual names for the registers.
- **typeName**—The name of the register template of which each register in the array will be an instance.
- **offset**—The address offset (relative to this map) of the first register in the array.
- **number**—The number of registers in the register array.
- **instanceQualifier**—The string format that the generator should use to differentiate the field names between the registers in this array.
- options
  - **-step** *value*—The address offsets for each register in the array increment by **value** Bytes.

F

F *name size attributes* [type]

The **F** discriminant specifies a field within a register or a register template, starting at the least significant bit.

- **name**—The name of the field. If the name is **Reserved**, then the generator should not create accessors and should always set the field to 0.
- **size**—The size of the field in bits.
- attributes
  - **Strobe**—This attribute indicates to the generator that after writing the register, this field should be reset to 0.
  - **Decoded**—This attribute indicates to the generator that the field can be accessed bit-wise in addition to field-wise.
  - **.**—A period indicates to the generator that there are no attributes specified.
  - **type**—The data type of the field.

Examples

These examples are from three RBM files: RBM/XSeries.rbm, RBM/AI.rbm, and RBM/AO.rbm.

Global Settings

```
—contains AI 0x20270 AI.rbm "tAI.h"
```

This line is in XSeries.rbm, a bit map that aggregates all of the subsystem bit maps. Specifically, this line says that a variable called AI at offset 0x20270 is part of this bit map. The bit map for AI can be found in the file AI.rbm, and its declarations are in the file tAI.h. The generated code can be found in ChipObjects/tXSeries.h and ChipObjects/tXSeries.cpp.

```
—generate-include "tAIValues.h"
```

This line is in AI.rbm, and it says that the generator should add an #include "tAIValues.h" directive to its generated code, which can be found in ChipObjects/tAI.h.

Enumeration

```
E AI_Config_Channel_Type_t
V Differential  1
V NRSE          2
V RSE           3
V Internal      5
```

This stanza is in AI.rbm, and it defines an enumeration named `AI_Config_Channel_Type_t` with unique values for four different AI channel terminal configurations. The generated code can be found in `ChipObjects/tAIValues.h`.

Concrete Register

```
R AI_Trigger_Select_Register  32   0x2C Writable --user-mode-writable
                                   false

F AI_START1_Select            6    nAI::tAI_START1_Select_t

                                   This bitfield selects the AI_START1
                                   trigger.

F AI_START1_Edge              1    This bit enables edge-sensitive
                                   detection of the AI_START1 trigger.

                                   Note: Edge detection is required for
                                   almost all applications. It is also
                                   required when the source of the trigger
                                   is the SW Pulse.

F AI_START1_Polarity          1    nAI::tAI_Polarity_t

                                   This bit determines the polarity of the
                                   AI_START1 trigger.

                                   Note: Set this bit to 0 if
                                   AI_START1_Select is set to 0
                                   (AI_START1_Pulse).

F AI_START2_Select            6    nAI::tAI_START2_Select_t

                                   This bitfield selects the AI_START2
                                   trigger.

F AI_START2_Edge              1    This bit enables edge detection of the
                                   AI_START2 trigger.

                                   Note: Edge detection is required for
                                   almost all applications. It is also
                                   required when the source of the trigger
                                   is the SW Pulse.
```

```
F AI_START2_Polarity          1    nAI::tAI_Polarity_t

                                   This bit determines the polarity of
                                   AI_START2 trigger.

                                   Note: Set this bit to 0 if
                                   AI_START2_Select is set to 0
                                   (AI_START2_Pulse).

F AI_External_Gate_Select     6    nAI::tAI_External_Gate_Select_t

                                   This bitfield enables and selects the
                                   external gate. You can use the external
                                   gate to pause an analog input operation
                                   in progress.

F Reserved                    1

F AI_External_Gate_Polarity   1    nAI::tAI_Polarity_t

                                   This bit selects the polarity of the
                                   external gate signal.

F AI_CONVERT_Source_Select    6    nAI::tAI_StartConvertSelMux_t

                                   Selects the AI_CONVERT source.

                                   When you set this bitfield to 0, the AI
                                   Interface is in internal AI_CONVERT
                                   mode. When you select any other signal
                                   as the AI_CONVERT source, the AI Timer
                                   is in external AI_CONVERT mode.

F Reserved                    1

F AI_Convert_Source_Polarity  1    nAI::tAI_Polarity_t

                                   This bit selects the active edge of the
                                   AI_CONVERT source signal.

                                   Note: The polarity of this bit changed
                                   from the DAQ-STC2 to DAQ-STC3. This
                                   change was made to make it consistent
                                   with all other polarity bits.
```

This stanza is in `AI.rbm`, and it describes a concrete register named `AI_Trigger_Select_Register`. The register is 32-bits wide, can be found at offset `0x2C`, and is write-only. The first bitfield is called `AI_START1_Select`. It is 6-bits wide (using bits `5..0`) and requires values from the enumeration `nAI::tAI_START1_Select_t`. Notice that some fields are marked `Reserved`. The generated code can be found in `ChipObjects/tAI.h`.

Register Template

```
T AO_Config_Bank_t              8    Writable --no-hardware-reset false
                                     --initial-value 0xBF
                                     --user-mode-writablefalse

F AO_Offset                     3    Sets the offset select for Bank s/0/i.

F AO_Reference                  3    Sets the reference select for Bank
                                     s/0/i.

F AO_Update_Mode                1    nAO::tAO_Update_Mode_t

                                     This bit determines the update mode for
                                     Bank s/0/i.

F AO_Bipolar                    1    nAO::tAO_Bipolar_t

                                     This bit sets AO Bank s/0/i in Bipolar
                                     mode.
```

This stanza is in `AO.rbm`, and it describes a register template named `AO_Config_Bank_t`. It is 8-bits wide and write-only. In addition, when `tAO::reset()` is called, only the soft copy of this register is reset. The reset value is **`0xBF`**. The generated code can be found in `ChipObjects/tAO.h` and `ChipObjects/tAO.cpp`.

Register Array

```
TRA AO_Config_Bank%d AO_Config_Bank_t 0x4C  8 Bank%d --step 1 --group
                                            AO_Configuration_Group
```

This stanza is in `AO.rbm`, and it instantiates an array of registers with the name format `AO_Config_Bank%d` from the template `AO_Config_Bank_t`. The first register in this array can be found at offset `0x4C`. There are eight registers in this array, and their fields' names are prefixed with `Bank%d`. These registers are also contiguous on the device, each being one Byte after the preceding element.

3

Calibration Considerations

This chapter contains sections on *Temperature Effects on Device Accuracy* and *Minimizing Offset Error*.

Temperature Effects on Device Accuracy

Ambient temperature has a significant effect on the accuracy of measurement devices. The NI-DAQmx driver contains a self-calibration feature that can be used to calibrate out the effects of ambient temperature variation. This feature is not available in the MHDDK. Therefore, the effects of temperature drift must be taken into account when determining the accuracy of your measurement system. The effect of ambient temperature variations on the measurement accuracy of your X Series device can be calculated using the AI Absolute Accuracy table in the specifications document for your X Series device. Refer to the boardBringup example for how to measure the current temperature of the device, the temperature of the device when externally calibrated, and the temperature of the device when self-calibrated.

NI PXIe-6363 AI Absolute Accuracy Example #1

Consider the AI Absolute Accuracy for an NI PXIe-6363 in ±10 V range as defined in the *NI 6361/6363 Specifications* document. The formulas are defined as follows:

$$Absolute\ Accuracy = (10\ V \times Gain\ Error) + (10\ V \times Offset\ Error) + Noise\ Uncertainty$$

where

$$Gain\ Error = ResidualGainError + (GainTempco \times TempChangeFromLastInternalCal) + (ReferenceTempco \times TempChangeFromLastExternalCal)$$

$$Offset\ Error = ResidualOffsetError + (OffsetTempco \times TempChangeFromLastInternalCal) + INL\_Error$$

$$Noise\ Uncertainty = \frac{Random\ Noise \times 3}{\sqrt{10000}}$$

Consider a device with a TempChangeFromLastInternalCal of 1 degree and a TempChangeFromLastExternalCal of 10 degrees. The absolute accuracy for a 1 degree drift is calculated as follows:

Gain Error = 48 pmm + (13 ppm × **1**) + (1 ppm × 10) = 71 ppm

Offset Error = 13 ppm + (21 ppm × **1**) + 60 ppm = 94 ppm

$$\textit{Noise Uncertainty} = \frac{315\ \mu V \times 3}{\sqrt{10000}} = 9.4\ \mu V$$

Absolute Accuracy = (10 V × 71 ppm) + (10 V × 94 ppm) + 9.4 µV = 1660 µV

Next, keeping the TempChangeFromLastExternalCal the same, set the TempChangeFromLastInternalCal to 10 degrees. The absolute accuracy for a 10 degree drift is calculated as follows:

Gain Error = 48 ppm + (13 ppm × **10)** + (1 ppm × 10) = 188 ppm

Offset Error = 13 ppm + (21 ppm × **10**) + 60 ppm = 283 ppm

$$\textit{Noise Uncertainty} = \frac{315\ \mu V \times 3}{\sqrt{10000}} = 9.4\ \mu V$$

Absolute Accuracy = (10 V × 188 ppm) + (10 V × 283 ppm) + 9.4 µV = 4719.4 µV

The difference between these two absolute accuracies shows the effects of temperature alone on accuracy, since both the noise and offset errors have been normalized. In this example, the effects of 10 degrees of temperature drift are 4719 µV– 1660 µV = 3059 µV of absolute accuracy. You can perform these same calculations for your device using the formulas in the specifications document for your X Series device. Similar formulas are provided for analog output as well.

Minimizing Offset Error

National Instruments X Series devices have internal channels that can be used to minimize offset error. By measuring the _aiGnd_vs_aiGnd channel, you effectively short the ADC to ground, providing a true 0 V reference. By averaging 10,000 samples prior to taking your measurements, you can determine an offset coefficient to reduce your measurement error due to offset.

NI PXIe-6363 AI Absolute Accuracy Example #2

Refer to the calculations for AI absolute accuracy performed in the *NI PXIe-6363 AI Absolute Accuracy Example #1* section. If you are able to completely remove the offset error by performing the described calculations and offset removal, you get better performance.

GainError = 48 ppm + 13 ppm × **10** + 1 ppm × 10 = 188 ppm

OffsetError = ~~13 ppm + 21 ppm × **10** + 60 ppm = 283 ppm~~ = 0 ppm

$$\textit{Noise Uncertainty} = \frac{315\ \mu V \times 3}{\sqrt{10000}} = 9.4\ \mu V$$

AbsoluteAccuracy = 10 V × 188 ppm + 10 V × ~~283 ppm~~ 0 ppm + 9.4 µV = 1889 µV

This is a much better absolute accuracy. The offset error cannot be completely removed by performing this calibration, however it can be minimized.

4

Example Synopses

This chapter contains example definitions and descriptions. The examples are grouped in the following sections:

- *Device*
- *Analog Input*
- *Analog Output*
- *Digital Input*
- *Digital Output*
- *Counter Input*
- *Counter Output*

Example Synopses

The Device examples are as follows:

`boardBringup.cpp`—Self-test an X Series device.

The Analog Input examples are as follows:

`aiex1.cpp`—Single-point on-demand analog input.

`aiex2.cpp`—Retriggerable finite hardware-timed analog input.

`aiex3.cpp`—Continuous hardware-timed analog input with optimized DMA.

`aiex4.cpp`—Finite hardware-timed analog input with reference trigger.

`aiex5.cpp`—Continuous single-wire hardware-timed analog input with optimized DMA.

`aiex6.cpp`—Multi-device continuous hardware-timed analog input with optimized DMA.

The Analog Output examples are as follows:

`aoex1.ccp`—Single-point on-demand analog output.

`aoex2.ccp`—Single-point on-demand analog output with simultaneous updates.

`aoex3.ccp`—Retriggerable finite hardware-timed analog output.

`aoex4.ccp`—Finite hardware-timed analog output with external reference clock.

`aoex5.ccp`—Continuous hardware-timed analog output with regeneration.

`aoex6.ccp`—Continuous hardware-timed analog output with optimized DMA.

The Digital Input examples are as follows:

`dioex1.cpp`—Single-point on-demand digital input and output on ports 0, 1, and 2.

`dioex2.cpp`—Finite hardware-timed digital input.

`dioex3.cpp`—Finite hardware-timed digital input with change detection.

`dioex4.cpp`—Continuous hardware-timed digital input with optimized DMA.

The Digital Output examples are as follows:

`dioex5.cpp`—Continuous hardware-timed digital output with regeneration.

`dioex6.cpp`—Continuous hardware-timed digital output with the Watchdog Timer.

The Counter Input examples are as follows:

`gpctex1.cpp`—Single-point on-demand pulse train measurement.

`gpctex2.cpp`—Single-point edge counting with start trigger and hardware controlled direction.

`gpctex3.cpp`—Averaged continuous hardware-timed frequency input with optimized DMA.

`gpctex4.cpp`—Finite hardware-timed position measurement with digital filtering.

`gpctex5.cpp`—Continuous hardware-timed edge counting with optimized DMA.

The Counter Output examples are as follows:

`gpctex6.cpp`—Pulse train output using the frequency generator.

`gpctex7.cpp`—Single-point implicitly-timed pulse train output.

`gpctex8.cpp`—Finite implicitly-timed pulse train output with start trigger.

`gpctex9.cpp`—Continuous hardware-timed pulse train output with regeneration.

Examples

The example synopses below are all defined and then described in more detail.

Device

- **`boardBringup.cpp`**—Self-test an X Series device.

  boardBringup performs a self-test on an X Series device. After identifying the X Series device, boardBringup reads the signature registers and prints other hardware information, including the device's on-board temperature. Next, boardBringup reads the device's EEPROM and prints the calibration meta-information and scaling coefficients. Finally, boardBringup tests device register IO by writing to and from the scratchpad registers.

Analog Input

- **aiex1.cpp**—Single-point on-demand analog input.

  aiex1 reads and scales analog data using software timing and transfers data to the host by reading directly from the FIFO. After configuring the AI subsystem's timing and channel parameters, aiex1 initiates a configurable number of scans and reads, scales and prints the data. Finally, aiex1 restores the hardware's previous state.

- **aiex2.cpp**—Retriggerable finite hardware-timed analog input.

  aiex2 reads and scales analog data using hardware timing and transfers data to the host by reading directly from the FIFO. After configuring the start trigger input and the AI subsystem's timing and channel parameters, aiex2 arms the timing engine. For 10 seconds, data is read, scaled, and printed in chunks on each start trigger received until aiex2 disarms the timing engine. Finally, aiex2 restores the hardware's previous state.

- **aiex3.cpp**—Continuous hardware-timed analog input with optimized DMA.

  aiex3 reads and scales analog data using hardware timing and transfers data to the host using an optimized DMA buffer. After configuring the AI subsystem's timing and channel parameters, aiex3 configures and starts the DMA channel before sending a software start trigger. For 10 seconds, data is read, scaled, and printed in chunks until aiex3 stops the timing engine, shuts down DMA, and flushes the buffer. Finally, aiex3 restores the hardware's previous state.

- **aiex4.cpp**—Finite hardware-timed analog input with reference trigger.

  aiex4 reads and scales analog data using hardware timing and transfers data to the host using DMA. After configuring the reference trigger input and AI subsystem's timing and channel parameters, aiex4 configures and starts the DMA channel before sending a software start trigger. For 10 seconds, aiex4 waits for a reference trigger, counting down how many pre- and post-trigger samples remain in the measurement. Once the measurement is complete, aiex4 shuts down DMA and reads, scales, and prints the data. Finally, aiex4 restores the hardware's previous state.

- **aiex5.cpp**—Continuous single-wire hardware-timed analog input with optimized DMA.

  aiex5 reads and scales analog data using hardware timing and transfers data to the host using an optimized DMA buffer. The sample and convert clock share the same external source, which is known as single-wire mode. After configuring the AI subsystem's timing and channel parameters, aiex5 configures and starts the DMA channel before sending a software start trigger. Once the measurement starts, every four clock ticks will trigger the sample clock as well as the convert clock for each of the four channels. For 10 seconds, data is read, scaled, and printed in chunks until aiex5 stops the timing engine, shuts down DMA, and flushes the buffer. Finally, aiex5 restores the hardware's previous state.

- **aiex6.cpp**—Multi-device continuous hardware-timed analog input with optimized DMA.

  aiex6 reads and scales analog data from two devices using hardware timing and transfers data to the host using optimized DMA buffers. Before configuring the AI subsystem on each device, aiex6 programs the devices' Phase-Lock Loop circuits to lock to a common reference clock and uses backplane (for PXIe) or RTSI (for PCIe) signals to achieve nanosecond synchronization with trigger skew correction. In addition, aiex6 exports both devices' sample clocks on PFI0 to show their phase-alignment. After the AI subsystem's timing and channel parameters have been programmed, aiex6 configures and starts the DMA channel before waiting 10 seconds for an external start trigger. Once triggered, data is read, scaled, and printed in chunks for 10 seconds until aiex6 stops the timing engine, shuts down DMA, and flushes the buffer. Finally, aiex6 restores the hardware's previous state.

Analog Output

- **aoex1.ccp**—Single-point on-demand analog output.

  aoex1 scales and generates analog data using software timing and transfers data to the device by writing directly to the DAC data registers. After configuring the AO subsystem's channels, aoex1 creates ramp waveforms to generate on each channel. Since aoex1 writes directly to the DAC's data registers, it does not need to configure the timing engine or FIFO. After generating the waveforms, aoex1 restores the hardware's previous state.

- **aoex2.ccp**—Single-point on-demand analog output with simultaneous updates.

  aoex2 scales and generates analog data using software timing and transfers data to the device by writing directly to the DAC data registers. After configuring the AO subsystem's timing and channels, aoex2 creates sine waveforms to generate on each channel. Although aoex2 writes directly to the DAC's data registers, it needs to configure the timing engine so that updates will occur simultaneously. After generating the waveforms, aoex2 restores the hardware's previous state.

- **aoex3.ccp**—Retriggerable finite hardware-timed analog output.

  aoex3 scales and generates finite analog data using hardware timing and transfers data to the device using DMA. After configuring the AO subsystem's timing and channels, aoex3 creates a sine wave and a square wave to generate on each channel, then configures and primes the DMA channel. For 10 seconds, the data is generated on each start trigger received until aoex3 stops the timing engine and shuts down DMA. Finally, aoex3 restores the hardware's previous state.

- **aoex4.ccp**—Finite hardware-timed analog output with external reference clock.

  aoex4 scales and generates finite analog data using hardware timing and transfers data to the device using DMA. Before configuring the AO subsystem, aoex4 programs the Phase-Lock Loop circuit to lock to an external reference clock. After configuring the AO

subsystem's timing and channels, aoex4 creates a sine wave to generate on the output channel, then configures and primes the DMA channel. Using the external reference clock as the timebase for the on-board oscillator, aoex4 generates a finite number of analog updates after it asserts the software start trigger. Finally, aoex4 restores the hardware's previous state.

- **aoex5.ccp**—Continuous hardware-timed analog output with regeneration.

  aoex5 scales and generates analog data using hardware timing and transfers data to the device using DMA. After configuring the AO subsystem's timing and channels, aoex5 creates a sine wave and a square wave to generate on each channel, then configures and primes the DMA channel before sending a software start trigger. For 10 seconds, data is regenerated from either the host (using a circular DMA buffer) or the device (using its on-board FIFO) until aoex5 stops the timing engine and shuts down DMA. Finally, aoex5 restores the hardware's previous state.

- **aoex6.ccp**—Continuous hardware-timed analog output with optimized DMA.

  aoex6 scales and generates analog data using hardware timing and transfers data to the device using an optimized DMA buffer. After configuring the AO subsystem's timing and channels, aoex6 creates a sine wave and a square wave to generate on both channels (with a 90-degree phase difference between ao0 and ao1), then configures and primes the DMA channel before sending a software start trigger. For 10 seconds, data is written to the buffer and generated by the device, alternating between sine and square wave waveforms until aoex6 stops the timing engine and shuts down DMA. Finally, aoex6 restores the hardware's previous state.

Digital Input

- **dioex1.cpp**—Single-point on-demand digital input and output on ports 0, 1, and 2.

  dioex1 performs a software-timed digital loop-back operation and transfers data via direct register accesses. After configuring the DI subsystem's channel parameters, dioex1 updates the digital values of ports 0, 1, and 2 and then reads the updates back. Finally, dioex1 restores the hardware's previous state.

- **dioex2.cpp**—Finite hardware-timed digital input.

  dioex2 reads a digital waveform using hardware timing and transfers data to the host by reading directly from the FIFO. After configuring the DI subsystem's timing and channel parameters, dioex2 sends a software start trigger. Once the measurement is complete, dioex2 reads and prints the data. Finally, dioex2 restores the hardware's previous state.

- **dioex3.cpp**—Finite hardware-timed digital input with change detection.

  dioex3 reads a digital waveform using the change detection circuit and transfers data to the host using DMA. After configuring the DI subsystem's timing and channel parameters, dioex3 configures and starts the DMA channel before sending a software start trigger. Once the measurement is complete, dioex3 shuts down DMA data and reads and prints the data. Finally, dioex3 restores the hardware's previous state.

- **dioex4.cpp**—Continuous hardware-timed digital input with optimized DMA.

  dioex4 reads a digital waveform using hardware timing and transfers data to the host using an optimized DMA buffer. After configuring the DI subsystem's timing and channel parameters, dioex4 configures and starts the DMA channel before sending a software start trigger. For 10 seconds, data is read and printed in chunks until dioex4 stops the timing engine, shuts down DMA, and flushes the buffer. Finally, dioex4 restores the hardware's previous state.

Digital Output

- **dioex5.cpp**—Continuous hardware-timed digital output with regeneration.

  dioex5 generates a digital waveform using hardware timing and transfers data to the device using DMA. After configuring the DO subsystem's timing and channels, dioex5 creates a simple incrementing waveform to generate on port0, then configures and primes the DMA channel before sending a software start trigger. For 10 seconds, data is regenerated from either the host (using a circular DMA buffer) or the device (using its on-board FIFO) until dioex5 stops the timing engine and shuts down DMA. Finally, dioex5 restores the hardware's previous state.

- **dioex6.cpp**—Continuous hardware-timed digital output with the Watchdog Timer.

  dioex6 generates a digital waveform using hardware timing and monitors device communication using the Watchdog Timer. After configuring the Watchdog Timer, dioex6 programs the DO subsystem's timing and channel parameters. dioex6 then creates a simple incrementing waveform and configures and starts the DMA channel before sending a software start trigger. For 10 seconds, data is written to an optimized DMA buffer and generated by the device until dioex6 trips the Watchdog. For the next two seconds, dioex6 waits and shows the programmed safe states before stopping the timing engine and shutting down DMA. Finally, dioex6 restores the hardware's previous state.

Counter Input

- **gpctex1.cpp**—Single-point on-demand pulse train measurement.

  gpctex1 measures pulse train characteristics and transfers data to the host by reading directly from the save register or FIFO. After configuring the counter's gating parameters, gpctex1 arms the counter via software. Once started, data is read and printed until the measurement is complete and gpctex1 disarms the counter. Finally, gpctex1 restores the hardware's previous state.

- **gpctex2.cpp**—Single-point edge counting with start trigger and hardware controlled direction.

  gpctex2 counts TTL edges and transfers data to the host by reading directly from the save register. After configuring the counter's gating parameters, gpctex2 arms the counter and waits for an arm-start trigger. Once started, data is read and printed until the measurement

is complete and gpctex2 disarms the counter. Finally, gpctex2 restores the hardware's previous state.

- **gpctex3.cpp**—Averaged continuous hardware-timed frequency input with optimized DMA.

  gpctex3 measures TTL frequency using hardware timing and transfers data to the host using an optimized DMA buffer. The measurement method is configurable to latch the frequency as an average over the clock period or as the most recent pulse. After configuring the counter's timing and gating parameters, gpctex3 configures and starts the DMA channel before arming the counter via software. For 10 seconds, data is read, scaled, and printed in chunks until gpctex3 disarms the counter, shuts down DMA, and flushes the buffer. Finally, gpctex3 restores the hardware's previous state.

- **gpctex4.cpp**—Finite hardware-timed position measurement with digital filtering.

  gpctex4 measures an encoder using hardware timing and transfers data to the host using DMA. Before configuring the counter, gpctex4 programs the digital filter on each input terminal. After configuring the counter's timing and gating parameters, gpctex4 configures and starts the DMA channel before arming the counter via software. Once the measurement is complete, gpctex4 disarms the counter, shuts down DMA, and reads, scales, and prints the data. Finally, gpctex4 restores the hardware's previous state.

- **gpctex5.cpp**—Continuous hardware-timed edge counting with optimized DMA.

  gpctex5 counts TTL edges using hardware timing and transfers data to the host using an optimized DMA buffer. After configuring the counter's timing and gating parameters, gpctex5 configures and starts the DMA channel before arming the counter via software. For 10 seconds, data is read and printed in chunks until gpctex5 disarms the counter, shuts down DMA, and flushes the buffer. Finally, gpctex5 restores the hardware's previous state.

Counter Output

- **gpctex6.cpp**—Pulse train output using the frequency generator.

  gpctex6 generates a pulse train using implicit timing and transfers data to the device by writing directly to the frequency generator registers. After configuring the generator's timebase and divider parameters, gpctex6 enables the output. For 10 seconds, the pulse train generates until gpctex6 disables the generator. Finally, gpctex6 restores the hardware's previous state.

- **gpctex7.cpp**—Single-point implicitly-timed pulse train output.

  gpctex7 scales and generates a pulse train using implicit timing and transfers data to the device by writing directly to the counter registers. After configuring the counter's gating parameters, gpctex7 creates a simple pulse train to generate before sending a software start trigger. The counter generates pulses, following the specified idle count with the specified active count. After the generation is complete, gpctex7 disarms the counter. Finally, gpctex7 restores the hardware's previous state.

- **gpctex8.cpp**—Finite implicitly-timed pulse train output with start trigger.

  gpctex8 scales and generates a finite pulse train using implicit timing and transfers data to the device using DMA. After configuring the counter's timing and gating parameters, gpctex8 creates a simple "chirp" pulse train to generate, then configures and primes the DMA channel before waiting 10 seconds for an external start trigger. Once triggered, the counter generates the pulses, following the specified idle count with the specified active count. After the generation is complete, gpctex8 disarms the counter and shuts down DMA. Finally, gpctex8 restores the hardware's previous state.

- **gpctex9.cpp**—Continuous hardware-timed pulse train output with regeneration.

  gpctex9 scales and generates a continuous pulse train using hardware timing and transfers data to the device using DMA. After configuring the counter's timing and gating parameters, gpctex9 creates a simple "chirp" pulse train to generate, then configures and primes the DMA channel before sending a software start trigger. For 10 seconds, data is regenerated from either the host (using a circular DMA buffer) or the device (using its on-board FIFO). The counter generates pulses, following the specified idle count with the specified active count. The first pulse generated by the counter is not part of the data set and is configurable to allow a custom start delay and an initial pulse train. The first sample clock tick then loads the first pulse train specification from the data set, and the counter will generate that pulse continuously until the next sample clock tick. Each sample clock tick causes the counter to load the next set of pulse train specifications. After the 10 seconds have passed, gpctex9 disarms the counter and shuts down DMA. Finally, gpctex9 restores the hardware's previous state.

5

X Series Chip Object

This chapter contains information about the chip object for the X Series device. Table 5-1 shows sections of the chip object, the offset, and a reference for more information.

Table 5-1. X Series Chip Object

Chip Object	Offset	Reference
CHInCh	0x0	*CHInCh Chip Object Registers*
AI_DMAChannel	0x2000	*DMA Controller Registers*
Counter0DMAChannel	0x2100	*DMA Controller Registers*
Counter1DMAChannel	0x2200	*DMA Controller Registers*
Counter2DMAChannel	0x2300	*DMA Controller Registers*
Counter3DMAChannel	0x2400	*DMA Controller Registers*
DI_DMAChannel	0x2500	*DMA Controller Registers*
AO_DMAChannel	0x2600	*DMA Controller Registers*
DO_DMAChannel	0x2700	*DMA Controller Registers*
SimutaneousControl	0x6000	*SimultaneousControl Registers*
BrdServices	0x20000	*Board Services Registers*
BusInterface	0x20000	*Bus Interface Registers*
Triggers	0x20000	*Triggers and Timing Registers*
AI	0x20270	*Analog Input Registers*
AI InTimer	0x202B0	*InTimer Registers*
Counter0	0x20300	*Counter Registers*
Counter1	0x20340	*Counter Registers*
Counter2	0x20380	*Counter Registers*
Counter3	0x203C0	*Counter Registers*

Table 5-1. X Series Chip Object (Continued)

Chip Object	Offset	Reference
AO	0x20400	*Analog Output Registers*
AO OutTimer	0x20470	*OutTimer Registers*
DO	0x204AC	*Digital Output Registers*
DO OutTimer	0x204E0	*OutTimer Registers*
DI	0x20530	*Digital Input Registers*
DI InTimer	0x20560	*InTimer Registers*
AIStreamCircuit	0x24000	*Stream Circuit Registers*
Counter0StreamCircuit	0x26000	*Stream Circuit Registers*
Counter1StreamCircuit	0x28000	*Stream Circuit Registers*
Counter2StreamCircuit	0x2A000	*Stream Circuit Registers*
Counter3StreamCircuit	0x2C000	*Stream Circuit Registers*
DIStreamCircuit	0x2E000	*Stream Circuit Registers*
AOStreamCircuit	0x30000	*Stream Circuit Registers*
DOStreamCircuit	0x32000	*Stream Circuit Registers*

6

CHInCh Chip Object Registers

CHInCh base address = 0x0

This chapter consists of *DMA Channels*, *CHInCh Registers*, and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

List of CHInCh Chip Object Registers

This section includes the individual channels and their corresponding offsets. You can find more information in Chapter 7, *DMA Controller Registers*.

The CHInCh Chip Object registers are as follows:

Offset 0x0: CHInCh_Identification_Register (R)

Offset 0x10: IO_Port_Resource_Description_Register (R)

Offset 0x5C: Interrupt_Mask_Register (R|W)

Offset 0x60: Interrupt_Status_Register (R)

Offset 0x68: Volatile_Interrupt_Status_Register (R)

Offset 0xA4: Host_Bus_Resource_Control_Register (R|W)

Offset 0xC0: EEPROM_Window_Register (R|W)

Offset 0xC4: Simultaneous_Window_Register (R|W)

Offset 0xE0: Window_Control_Register (R|W)

Offset 0x200: Scrap_Register (R|W)

Offset 0x514: Configuration_Register (R|W)

Offset 0x580: EEPROM_Register_0 (R|W)

Offset 0x584: EEPROM_Register_1 (R|W)

Offset 0x58C: EEPROM_Register_2 (R|W)

Offset 0x590: SMIO_Register_0 (R|W)

Offset 0x594: SMIO_Register_1 (R|W)

Offset 0x598: SMIO_Register_2 (R|W)

Offset 0x59C: SMIO_Register_3 (R|W)

Offset 0x10AC: PCI_SubSystem_ID_Access_Register (R)

DMA Channels

Channel Name	Offset
AI_DMAChannel	0x2000
Counter0DMAChannel	0x2100
Counter1DMAChannel	0x2200
Counter2DMAChannel	0x2300
Counter3DMAChannel	0x2400
DI_DMAChannel	0x2500
AO_DMAChannel	0x2600
DO_DMAChannel	0x2700

List of Enumerated Types

The enumerated types are as follows:

CHInCh_Signature_t

Configuration_Values_t

EEPROM_Register_0_Value_t

EEPROM_Register_1_Value_t

EEPROM_Register_2_Value_t

SMIO_Register_0_Value_t

SMIO_Register_1_Value_t

SMIO_Register_2_Value_t

SMIO_Register_3_Value_t

CHInCh Registers

Offset 0x0: CHInCh_Identification_Register (R)

Absolute Address: 0x00000

Bits	Name
31..0	**ID**—This field is used to identify the CHInCh. The values for this bitfield are in *CHInCh_Signature_t*.

Offset 0x10: IO_Port_Resource_Description_Register (R)

Absolute Address: 0x00010

Bits	Name
31..20	**Reserved**
19..16	**IOMPS**—This field indicates the maximum data payload size in Bytes. The maximum size is 2^{IOMPS}.
15..0	**Reserved**

Offset 0x5C: Interrupt_Mask_Register (R|W)

Absolute Address: 0x0005C

Bits	Name
31	**Set_CPU_Int**—Writing a 1 to this bit enables interrupts to the host bus. This bit will return a 1 when the interrupt is enabled and a 0 when it is disabled. The CPU interrupt is disabled on reset.
30	**Clear_CPU_Int**—Writing a 1 to this bit disables interrupts to the host bus. This bit will return a 0 when the interrupt is enabled and a 1 when it is disabled. The CPU interrupt is disabled on reset.
29..12	**Reserved**
11	**Set_STC3_Int**—Writing a 1 to this bit enables the DAQ-STC3 line to interrupt the host bus. This bit will return a 1 when the interrupt is enabled and a 0 when it is disabled. This interrupt is disabled on reset.
10	**Clear_STC3_Int**—Writing a 1 to this bit disables the DAQ-STC3 line from interrupting the host bus. This bit will return a 0 when the interrupt is enabled and a 1 when it is disabled. This interrupt is disabled on reset.
9	**Set_SMIO_FIFO_Int**—Writing a 1 to this bit enables the SMIO_FIFO line to interrupt the host bus. This bit will return a 1 when the interrupt is enabled and a 0 when it is disabled. This interrupt is disabled on reset.

Bits	Name
8	**Clear_SMIO_FIFO_Int**—Writing a 1 to this bit disables the SMIO_FIFO line to interrupt the host bus. This bit will return a 0 when the interrupt is enabled and a 1 when it is disabled. This interrupt is disabled on reset.
7..0	**Reserved**
Options: No Hardware Reset	

Offset 0x60: Interrupt_Status_Register (R)

Absolute Address: 0x00060

Bits	Name
31	**Int**—This read-only bit returns 1 when the CHInCh has a pending interrupt. This bit will clear when read from the volatile offset if no additional interrupts are pending and the current return value indicates a condition that is internal to the CHInCh (the external bit is clear).
30	**Additional_Int**—This read-only bit returns 1 when at least one additional interrupt is pending beyond the interrupt status word being returned. This allows an interrupt service routine to retrieve all pending notifications by continuously reading this register until this bit returns 0.
29	**External**—This read-only bit returns 1 when the highest priority interrupt is external to the CHInCh. More specific interrupt status must be read from the DAQ-STC3 for external interrupts. External interrupt conditions also must be cleared in the DAQ-STC3–they are not cleared by reading the volatile offset in the CHInCh. The external interrupts should be cleared in the DAQ-STC3 before reading this register again because the external interrupt conditions will prevent this register from returning lower priority conditions–this register will continue to indicate the external condition until the IO interrupts are unasserted or disabled in the *Interrupt_Mask_Register* (IMR).
28	**DMA**—This read-only bit returns 1 when the highest priority notification is from a DMA channel. Refer to the *Channel_Status_Register* for the format of DMA interrupt status words.
27..12	**Reserved**
11	**STC3_Int**—This read-only bit returns 1 when the DAQ-STC3 interrupt line is asserted and external interrupts are the highest priority condition pending.
10	**Reserved**
9	**SMIO_FIFO_Int**—This read-only bit returns 1 when the SMIO_FIFO interrupt line is asserted and external interrupts are the highest priority condition pending.
8..0	**Reserved**
Options: No Soft Copy	

Offset 0x68: Volatile_Interrupt_Status_Register (R)

Absolute Address: 0x00068

Bits	Name
31	**Vol_Int**—This read-only bit returns 1 when the CHInCh has a pending interrupt. This bit clears when read from the volatile offset if no additional interrupts are pending and the current return value indicates a condition that is internal to the CHInCh (the external bit is clear).
30	**Vol_Additional_Int**—This read-only bit returns 1 when at least one additional interrupt is pending beyond the interrupt status word being returned. This allows an interrupt service routine to retrieve all pending notifications by continuously reading this register until this bit returns 0.
29	**Vol_External**—This read-only bit returns 1 when the highest priority interrupt is external to the CHInCh. More specific interrupt status must be read from the DAQ-STC3 for external interrupts. External interrupt conditions also must be cleared in the DAQ-STC3–they are not cleared by reading the volatile offset in the CHInCh. The external interrupts should be cleared in the DAQ-STC3 before reading this register again because the external interrupt conditions will prevent this register from returning lower priority conditions–this register will continue to indicate the external condition until the IO interrupts are unasserted or disabled in the *Interrupt_Mask_Register* (IMR).
28	**Vol_DMA**—This read-only bit returns 1 when the highest priority notification is from a DMA channel. Refer to the *Channel_Status_Register* for the format of DMA interrupt status words.
27..12	**Reserved**
11	**Vol_STC3_Int**—This read-only bit returns 1 when the DAQ-STC3 interrupt line is asserted and external interrupts are the highest priority condition pending.
10	**Reserved**
9	**Vol_SMIO_FIFO_Int**—This read-only bit returns 1 when the SMIO_FIFO interrupt line is asserted and external interrupts are the highest priority condition pending.
8..0	**Reserved**
Options: No Soft Copy	

Offset 0xA4: Host_Bus_Resource_Control_Register (R|W)

Absolute Address: 0x000A4

Bits	Name
31	**IO_Master_Enable**—When this bit is set, the CHInCh processes transactions initiated by the Stream Circuit. When this bit is clear, the CHInCh responds to such transactions with an error. On reset, this bit is clear. Stream transactions can be enabled or disabled with the corresponding DMA channel Start bit.
30..16	**Reserved**
15	**DMA_LA64**—DMA link addresses will be 64-bits wide when this bit is set and 32-bits wide when it is clear. This includes link addresses contained in DMA links. This bit should *not* be modified while any DMA channels are active. On reset, this bit is clear.

Bits	Name
14	**DMA_MA64**—DMA memory addresses will be 64-bits wide when this bit is set and 32-bits wide when it is clear. This bit should *not* be modified while any DMA channels are active. On reset, this bit is clear.
13..0	**Reserved**
Options: No Hardware Reset	

Offset 0xC0: EEPROM_Window_Register (R|W)

Absolute Address: 0x000C0

Bits	Name
31..0	**EEPROM_Window_Field**—This field configures the extended EEPROM window for SMIO X Series devices.
Options: No Hardware Reset; No Soft Copy	

Offset 0xC4: Simultaneous_Window_Register (R|W)

Absolute Address: 0x000C4

Bits	Name
31..0	**Simultaneous_Window_Field**—This field configures the extended register window for SMIO X Series devices.
Options: No Hardware Reset; No Soft Copy	

Offset 0xE0: Window_Control_Register (R|W)

Absolute Address: 0x000E0

Bits	Name
31..0	**Window_Control_Field**—This field enables the extended registers and EEPROM for SMIO X Series devices.
Options: No Hardware Reset; No Soft Copy	

Offset 0x200: Scrap_Register (R|W)

Absolute Address: 0x00200

Bits	Name
31..0	**SDATA**—This field does not affect hardware operations and resets to an unknown value.
Options: No Hardware Reset; No Soft Copy	

Offset 0x514: Configuration_Register (R|W)

Absolute Address: 0x00514

Bits	Name
31..0	**Configuration_Value**—This field enables the extended registers and EEPROM for SMIO X Series devices.
Options: No Hardware Reset; No Soft Copy	

Offset 0x580: EEPROM_Register_0 (R|W)

Absolute Address: 0x00580

Bits	Name
31..0	**EEPROM_Register_0_Value**—This field configures the EEPROM for SMIO X Series devices. The values for this bitfield are in *EEPROM_Register_0_Value_t*.
Options: No Hardware Reset; No Soft Copy	

Offset 0x584: EEPROM_Register_1 (R|W)

Absolute Address: 0x00584

Bits	Name
31..0	**EEPROM_Register_1_Value**—This field configures the EEPROM for SMIO X Series devices. The values for this bitfield are in *EEPROM_Register_1_Value_t*.
Options: No Hardware Reset; No Soft Copy	

Offset 0x58C: EEPROM_Register_2 (R|W)

Absolute Address: 0x0058C

Bits	Name
31..0	**EEPROM_Register_2_Value**—This field configures the EEPROM for SMIO X Series devices. The values for this bitfield are in *EEPROM_Register_2_Value_t*.
Options: No Hardware Reset; No Soft Copy	

Offset 0x590: SMIO_Register_0 (R|W)

Absolute Address: 0x00590

Bits	Name
31..0	**SMIO_Register_0_Value**—This field configures the registers for SMIO X Series devices. The values for this bitfield are in *SMIO_Register_0_Value_t*.
Options: No Hardware Reset; No Soft Copy	

Offset 0x594: SMIO_Register_1 (R|W)

Absolute Address: 0x00594

Bits	Name
31..0	**SMIO_Register_1_Value**—This field configures the registers for SMIO X Series devices. The values for this bitfield are in *SMIO_Register_1_Value_t*.
Options: No Hardware Reset; No Soft Copy	

Offset 0x598: SMIO_Register_2 (R|W)

Absolute Address: 0x00598

Bits	Name
31..0	**SMIO_Register_2_Value**—This field configures the registers for SMIO X Series devices. The values for this bitfield are in *SMIO_Register_2_Value_t*.
Options: No Hardware Reset; No Soft Copy	

Offset 0x59C: SMIO_Register_3 (R|W)

Absolute Address: 0x0059C

Bits	Name
31..0	**SMIO_Register_3_Value**—This field configures the registers for SMIO X Series devices. The values for this bitfield are in *SMIO_Register_3_Value_t*.
Options: No Hardware Reset; No Soft Copy	

Offset 0x10AC: PCI_SubSystem_ID_Access_Register (R)

Absolute Address: 0x010AC

Bits	Name
31..16	**SubSystem_Product_ID**—This field returns the PID of X Series devices; use this field to determine your specific X Series model.
15..0	**SubSystem_Vendor_ID**—This field returns the VID of the sub-system; 0x1093 is for National Instruments.

Enumerated Types

CHInCh_Signature_t

Value	Name
0xC0107AD0	**CHInChSignature**

Configuration_Values_t

Value	Name
0x4	**EEPROMConfig**
0xC	**WindowSize**
0xF	**EEPROMConfigMask**
0x40	**SMIOConfig**
0x80	**WindowEnable**
0xF0	**SMIOConfigMask**
0x200	**MaxLinkSize**
0x1000	**PageSize**
0x5000	**EEPROMOffset**
0x6000	**SMIOOffset**

EEPROM_Register_0_Value_t

Value	Name
0x14044410	**EEPROMSettingsRegister0Value**

EEPROM_Register_1_Value_t

Value	Name
0x8188008	**EEPROMSettingsRegister1Value**

EEPROM_Register_2_Value_t

Value	Name
0x30006	**EEPROMSettingsRegister2Value**

SMIO_Register_0_Value_t

Value	Name
0x404440C	**SimultaneousRegister0Value**

SMIO_Register_1_Value_t

Value	Name
0x8060102	**SimultaneousRegister1Value**

SMIO_Register_2_Value_t

Value	Name
0x8060002	**SimultaneousRegister2Value**

SMIO_Register_3_Value_t

Value	Name
0x3	**SimultaneousRegister3Value**

7

DMA Controller Registers

This chapter consists of *DMA Channels*, *DMA Controller Registers* registers and *Enumerated Types* enumerations. Each register is labeled as a read (R) or write (W) register.

List of DMAController Registers

The DMAController registers are as follows:

Offset 0x38: Channel_Memory_Address_Register_LSW (R|W)

Offset 0x3C: Channel_Memory_Address_Register_MSW (R|W)

Offset 0x48: Channel_Link_Address_Register_LSW (R|W)

Offset 0x4C: Channel_Link_Address_Register_MSW (R|W)

Offset 0x50: Channel_Link_Size_Register (R|W)

Offset 0x54: Channel_Control_Register (R|W)

Offset 0x58: Channel_Operation_Register (R|W)

Offset 0x60: Channel_Status_Register (R)

Offset 0x68: Channel_Volatile_Status_Register (R)

Offset 0x90: Channel_Total_Transfer_Count_Compare_Register_LSW (R|W)

Offset 0x94: Channel_Total_Transfer_Count_Compare_Register_MSW (R|W)

Offset 0xA0: Channel_Total_Transfer_Count_Status_Register_LSW (R)

Offset 0xA4: Channel_Total_Transfer_Count_Status_Register_MSW (R)

Offset 0xA8: Channel_Total_Transfer_Count_Latching_Register_LSW (R)

Offset 0xAC: Channel_Total_Transfer_Count_Latching_Register_MSW (R)

List of DMAController Enumerated Types

The enumerated type from the DMA controller is *DMA_Mode_t*.

DMA Channels

This section includes the individual channels and their corresponding offsets.

Channel Name	Offset
AI_DMAChannel	0x2000
Counter0DMAChannel	0x2100
Counter1DMAChannel	0x2200
Counter2DMAChannel	0x2300
Counter3DMAChannel	0x2400
DI_DMAChannel	0x2500
AO_DMAChannel	0x2600
DO_DMAChannel	0x2700

DMA Controller Registers

Offset 0x38: Channel_Memory_Address_Register_LSW (R|W)

Absolute Addresses:
AI_DMAChannel: 0x2038
Counter0DMAChannel: 0x2138
Counter1DMAChannel: 0x2238
Counter2DMAChannel: 0x2338
Counter3DMAChannel: 0x2438
DI_DMAChannel: 0x2538
AO_DMAChannel: 0x2638
DO_DMAChannel: 0x2738

Bits	Name
31..0	**Memory_Address_LSW**—For Normal mode (NormalDmaMode), this field stores the address location of the next DMA data to transfer on the host bus. This is used in both modes. In Link Chaining mode, it is not necessary to initialize this field since the Link process will do so. When the DMA_MA64 bit in the *Host_Bus_Resource_Control_Register* (HBRCR) is clear, it is not necessary to write the upper half of this register. The value returned when reading this register is not guaranteed to be coherent with DMA data transfers. This field resets to an unknown value. When writing to this register using 8-bit writes, software must start with the least significant Byte and must write at least 3 Bytes.
Options: No Soft Copy	

Offset 0x3C: Channel_Memory_Address_Register_MSW (R|W)

Absolute Addresses:
AI_DMAChannel: 0x203C
Counter0DMAChannel: 0x213C
Counter1DMAChannel: 0x223C
Counter2DMAChannel: 0x233C
Counter3DMAChannel: 0x243C
DI_DMAChannel: 0x253C
AO_DMAChannel: 0x263C
DO_DMAChannel: 0x273C

Bits	Name
31..0	**Memory_Address_MSW**—This 64-bit register has been divided into two 32-bit registers to support those architectures. 64-bit architectures can access the register atomically.
Options: No Soft Copy	

Offset 0x48: Channel_Link_Address_Register_LSW (R|W)

Absolute Addresses:
AI_DMAChannel: 0x2048
Counter0DMAChannel: 0x2148
Counter1DMAChannel: 0x2248
Counter2DMAChannel: 0x2348
Counter3DMAChannel: 0x2448
DI_DMAChannel: 0x2548
AO_DMAChannel: 0x2648
DO_DMAChannel: 0x2748

Bits	Name
31..3	**Link_Address_LSW**—For Link Chain mode (LinkChainDmaMode), this field stores the address location of the next DMA link to be fetched. This is only used in Link Chaining mode. This field should be initialized to point to the first DMA link before the START bit is written with a 1 in the *Channel_Operation_Register* (CHOR). When the DMA_LA64 bit in the *Host_Bus_Resource_Control_Register* (HBRCR) is clear, it is not necessary to write the upper half of this register. The link processor updates this field as links are processed. Note that this register only accepts values that are naturally aligned to 8 Bytes. This field resets to an unknown value.
2..0	**Reserved**
Options: No Soft Copy	

Offset 0x4C: Channel_Link_Address_Register_MSW (R|W)

Absolute Addresses:
AI_DMAChannel: 0x204C
Counter0DMAChannel: 0x214C
Counter1DMAChannel: 0x224C
Counter2DMAChannel: 0x234C
Counter3DMAChannel: 0x244C
DI_DMAChannel: 0x254C
AO_DMAChannel: 0x264C
DO_DMAChannel: 0x274C

Bits	Name
31..0	**Link_Address_MSW**—This 64-bit register has been divided into two 32-bit regsiters to support those architectures. 64-bit architectures can access the register automatically.
Options: No Soft Copy	

Offset 0x50: Channel_Link_Size_Register (R|W)

Absolute Addresses:
AI_DMAChannel: 0x2050
Counter0DMAChannel: 0x2150
Counter1DMAChannel: 0x2250
Counter2DMAChannel: 0x2350
Counter3DMAChannel: 0x2450
DI_DMAChannel: 0x2550
AO_DMAChannel: 0x2650
DO_DMAChannel: 0x2750

Bits	Name
31..3	**Link_Size**—This field stores the size of the next DMA link to be fetched. This is only used in Link Chaining mode. This field should be initialized to the size of the first DMA link before the START bit is written with a 1 in the *Channel_Operation_Register* (CHOR). The link processor updates this field as links are processed. Note that this register only accepts values that are integer multiples of 8 Bytes. This field resets to an unknown value.
2..0	**Reserved**

Offset 0x54: Channel_Control_Register (R|W)

Absolute Addresses:
AI_DMAChannel: 0x2054
Counter0DMAChannel: 0x2154
Counter1DMAChannel: 0x2254
Counter2DMAChannel: 0x2354
Counter3DMAChannel: 0x2454
DI_DMAChannel: 0x2554
AO_DMAChannel: 0x2654
DO_DMAChannel: 0x2754

Bits	Name
31..28	**Reserved**
27	**Notify_On_Error**—Setting this bit causes the DMA channel to generate interrupts when errors are encountered during DMA operations. Writing a 0 to this bit also clears the Error bit in the *Channel_Status_Register* (CHSR) so that the interrupt condition is not reported when it is disabled. This bit resets to an unknown value.
26	**Notify_On_Last_Link**—Setting this bit causes the DMA channel to generate an interrupt after processing the last link of a chain. Writing a 0 to this bit also clears the Last Link bit in the *Channel_Status_Register* (CHSR) so that the interrupt condition is not reported when it is disabled. This bit resets to an unknown value.

Bits	Name
25	**Notify_On_Total_Count**—Setting this bit causes the DMA channel to generate an interrupt when the Total_Count bit in the *Channel_Status_Register* (CHSR) changes from 0 to 1. Writing a 0 to this bit also clears the Total_Count bit in the CHSR so that the interrupt condition is not reported when it is disabled. This bit resets to an unknown value.
24..16	**Reserved**
15	**Notify_On_Done**—Setting this bit causes the DMA channel to generate an interrupt when the Done bit in the *Channel_Status_Register* (CHSR) changes from 0 to 1. Writing a 0 to this bit also clears the Done bit in the CHSR so that the interrupt condition is not reported when it is disabled. This bit resets to an unknown value.
14..2	**Reserved**
1..0	**MODE**—This field controls the DMA mode. This field resets to an unknown value. The values for this bit field are in *DMA_Mode_t*.
Options: No Hardware Reset	

Offset 0x58: Channel_Operation_Register (R|W)

Absolute Addresses:
AI_DMAChannel: 0x2058
Counter0DMAChannel: 0x2158
Counter1DMAChannel: 0x2258
Counter2DMAChannel: 0x2358
Counter3DMAChannel: 0x2458
DI_DMAChannel: 0x2558
AO_DMAChannel: 0x2658
DO_DMAChannel: 0x2758

Bits	Name
31..26	**Reserved**
25	**Arm_Total_Count_Int**—Writing a 1 to this bit arms the total transfer count interrupt for the Byte count specified in the *Channel_Total_Transfer_Count_Compare_Register* (CHTTCCR). If the *Channel_Total_Transfer_Count_Status_Register* (CHTTCSR) has already been reached or passed the specified Byte count, the interrupt will occur immediately when this bit is written with a 1. It is not necessary to clear this bit after setting it. When read, this bit indicates if a Total Transfer Count interrupt is currently armed. The interrupt is armed from the time this bit is written with a 1 until the interrupt occurs. The behavior is undefined if this bit is written with a 1 in the same register access that any of the CLR_TTC, STOP, or START bits are written with a 1.
24..3	**Reserved**
2	**CLR_TTC**—Writing a 1 to this bit clears the *Channel_Total_Transfer_Count_Status_Register* (CHTTCSR). Note that this bit can be written with a 1 in the same register access that the START bit is written with a 1. It is not necessary to clear this bit after setting it. This bit returns 0 when read.

Bits	Name
1	**STOP**—Writing a 1 to this bit stops the DMA channel. The STOP bit takes priority when both the START and STOP bits are set in the same write. Reading this bit returns 0 when the DMA channel is started and 1 when it is stopped
0	**START**—Writing a 1 to this bit starts the DMA channel. After DMA channel is started, DMA requests from the Stream Circuit the corresponding stream are serviced. Starting a DMA channel also causes link fetching to begin if using Link Chaining mode. Writing a 1 to this bit also clears all interrupt conditions in the *Channel_Status_Register* (CHSR). The interrupt conditions can be cleared by writing a 1 to this bit even if the DMA channel is already started. Reading this bit returns 1 when the DMA channel is started and 0 when it is stopped. On reset, the state of Start is unknown and the DMA channel must be stopped and then started before it is operational. Except for the first use of a DMA channel, Start should *not* be written with a 1 until one of the Error, Last_Link, or Done bits of the CHSR returns 1.
Options: No Hardware Reset	

Offset 0x60: Channel_Status_Register (R)

Absolute Addresses:
AI_DMAChannel: 0x2060
Counter0DMAChannel: 0x2160
Counter1DMAChannel: 0x2260
Counter2DMAChannel: 0x2360
Counter3DMAChannel: 0x2460
DI_DMAChannel: 0x2560
AO_DMAChannel: 0x2660
DO_DMAChannel: 0x2760

Bits	Name
31	**Int**—This read-only bit returns 1 when the DMA channel has a pending interrupt. This bit clears when read from the volatile offset because all interrupt conditions are also in this register.
30	**Additional_Int**—This read-only bit returns 0 when read from the *Channel_Status_Register* (CHSR) or Channel Volatile Status Register (CHVSR). This bit is only effective when read from the Interrupt Status Register (ISR) or Volatile Interrupt Status Register (VISR).
29..28	**Reserved**
27	**Error**—This read-only bit sets when the DMA channel experiences an error condition while accessing host memory. This also causes the DMA channel to be stopped in the *Channel_Operation_Register* (CHOR). An interrupt is also generated when this bit sets if the Notify_On_Error bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_ On_Error bit in the CHCR is written with a 0. Once this bit is set it is safe to free system memory buffers that are used to store the application data and linked-list. It is also safe to start a new operation on the DMA channel after this bit sets.

Bits	Name
26	**Last_Link**—This read-only bit sets after the linked-list has been fully processed. This bit sets after the DMA channel transfers the data of the last page descriptor in a chain to or from host memory. The conditions that set this bit also cause the DMA channel to be stopped in the *Channel_Operation_Register* (CHOR). An interrupt will also be generated when this bit sets if the Notify_ On_Last_Link bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_On_Last_Link bit in the CHCR is written with a 0. Once this bit is set it is safe to free system memory buffers that are used to store the application data and linked-list. It is also safe to start a new operation on the DMA channel once this bit sets. Note that the Last_Link bit can return an invalid value when the Done bit in this register is set. Software should ignore the Last_Link bit when the Done bit is set.
25	**Total_Count**—After having been armed with the Arm_Total_Count_Int bit in the *Channel_Operation_Register* (CHOR), this read-only bit sets after the DMA channel transfers the data to or from host memory that causes the *Channel_Total_Transfer_Count_Status_Register* (CHTTCSR) to reach or pass the value in the *Channel_Total_Transfer_Count_Compare_Register* (CHTTCCR). An interrupt will also be generated when this bit sets if the Notify_On_Total_Count bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_On_Total_Count bit in the CHCR is written with a 0. Once this bit returns a 1, the application data represented by the value in the CHTTCCR is guaranteed to have been moved to or from system memory.
24..16	**Reserved**
15	**Done**—This read-only bit indicates that the DMA operation has ended due to the receipt of a Done indication from the associated subsystem or because the Stop bit in the *Channel_Operation_Register* (CHOR) was written with a 1. The conditions that set this bit also cause the DMA channel to be stopped in the CHOR. An interrupt is also generated when this bit sets if the Notify_On_Done bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_On_Done bit in the CHCR is written with a 0. Once this bit is set, it is safe to free system memory buffers that are used to store the application data and linked-list. It is also safe to start a new operation on the DMA channel once this bit sets.
14	**Link_Ready**—This read-only bit returns a 1 when the DMA channel has a link that is ready to be processed. This status could be useful for optimizing the start-up of an acquisition or generation operation. Some applications might need to hold off acquisition or generation until the DMA channel has fetched a link.
13..12	**Reserved**
11..0	**STREAM**—This read-only field returns the Stream Circuit number that the DMA channel is associated with.
Options: No Soft Copy	

Offset 0x68: Channel_Volatile_Status_Register (R)

Absolute Addresses:
AI_DMAChannel: 0x2068
Counter0DMAChannel: 0x2168
Counter1DMAChannel: 0x2268
Counter2DMAChannel: 0x2368
Counter3DMAChannel: 0x2468
DI_DMAChannel: 0x2568
AO_DMAChannel: 0x2668
DO_DMAChannel: 0x2768

Bits	Name
31	**Vol_Int**—This read-only bit returns 1 when the DMA channel has a pending interrupt. This bit will clear when read from the volatile offset because all interrupt conditions are also in this register.
30	**Vol_Additional_Int**
29..28	**Reserved**
27	**Vol_Error**—This read-only bit sets when the DMA channel experiences an error condition while accessing host memory. This also causes the DMA channel to be stopped in the *Channel_Operation_Register* (CHOR). An interrupt is also generated when this bit sets if the Notify_On_Error bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_ On_Error bit in the CHCR is written with a 0. After this bit is set it is safe to free system memory buffers that are used to store the application data and linked-list. It is also safe to start a new operation on the DMA channel once this bit sets.
26	**Vol_Last_Link**—This read-only bit sets after the linked-list has been fully processed. This bit sets after the DMA channel transfers the data of the last page descriptor in a chain to or from host memory. The conditions that set this bit also cause the DMA channel to be stopped in the *Channel_Operation_Register* (CHOR). An interrupt will also be generated when this bit sets if the Notify_ On_Last_Link bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_On_Last_Link bit in the CHCR is written with a 0. After this bit is set it is safe to free system memory buffers that are used to store the application data and linked-list. It is also safe to start a new operation on the DMA channel once this bit sets. Note that the Last_Link bit can return an invalid value when the Done bit in this register is set. Software should ignore the Last_Link bit when the Done bit is set.
25	**Vol_Total_Count**—After having been armed with the Arm_Total_Count_Int bit in the *Channel_Operation_Register* (CHOR), this read-only bit sets after the DMA channel transfers the data to or from host memory that causes the *Channel_Total_Transfer_Count_Status_Register* (CHTTCSR) to reach or pass the value in the *Channel_Total_Transfer_Count_Compare_Register* (CHTTCCR). An interrupt is also generated when this bit sets if the Notify_On_Total_Count bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_On_Total_Count bit in the CHCR is written with a 0. After this bit returns a 1, the application data represented by the value in the CHTTCCR is guaranteed to have been moved to or from system memory.

Bits	Name
24..16	**Reserved**
15	**Vol_Done**—This read-only bit indicates that the DMA operation has ended due to the receipt of a Done indication from the associated subsystem or because the Stop bit in the *Channel_Operation_Register* (CHOR) was written with a 1. The conditions that set this bit also cause the DMA channel to be stopped in the CHOR. An interrupt will also be generated when this bit sets if the Notify_On_Done bit of the *Channel_Control_Register* (CHCR) is set. The value of this bit is unknown on reset but clears when the START bit in the CHOR is written with a 1. This bit also clears when read from the volatile offset and when the Notify_On_Done bit in the CHCR is written with a 0. Once this bit is set, it is safe to free system memory buffers that are used to store the application data and linked-list. It is also safe to start a new operation on the DMA channel once this bit sets.
14	**Vol_Link_Ready**—This read-only bit returns a 1 when the DMA channel has a link that is ready to be processed. This status could be useful for optimizing the start-up of an acquisition or generation operation. Some applications might need to hold off acquisition or generation until the DMA channel has fetched a link.
13..12	**Reserved**
11.0	**Vol_STREAM**—This read-only field returns the Stream Circuit number that the DMA channel is associated with.
Options: No Soft Copy	

Offset 0x90: Channel_Total_Transfer_Count_Compare_Register_LSW (R|W)

Absolute Addresses:
AI_DMAChannel: 0x2090
Counter0DMAChannel: 0x2190
Counter1DMAChannel: 0x2290
Counter2DMAChannel: 0x2390
Counter3DMAChannel: 0x2490
DI_DMAChannel: 0x2590
AO_DMAChannel: 0x2690
DO_DMAChannel: 0x2790

Bits	Name
31..0	**TCC_LSW**—This register specifies a total Byte count at which an interrupt is desired. This entire register should be at the desired value before the interrupt is armed in the *Channel_Operation_Register* (CHOR). After being armed, the interrupt will occur once the *Channel_Total_Transfer_Count_Status_Register* (CHTTCSR) reaches or passes the programmed value. When any Byte of this register is written, the interrupt is disarmed. This prevents a bad comparison from occurring when multiple writes are required to provide the whole value. If a previous value had been armed, writing a new value to this register disarms it. A write to any Byte of this register also clears the Total_Count bit of the *Channel_Status_Register* (CHSR). Note that the previously armed value might have triggered an interrupt just before it gets disarmed by a write of the new value to this register. Clearing the Total_Count status bit on any write of this field prevents software from seeing an interrupt condition from any previously armed value. This field resets to an unknown value so it should be initialized before the interrupt is armed.
Options: No Soft Copy	

Offset 0x94: Channel_Total_Transfer_Count_Compare_Register_MSW (R|W)

Absolute Addresses:
AI_DMAChannel: 0x2094
Counter0DMAChannel: 0x2194
Counter1DMAChannel: 0x2294
Counter2DMAChannel: 0x2394
Counter3DMAChannel: 0x2494
DI_DMAChannel: 0x2594
AO_DMAChannel: 0x2694
DO_DMAChannel: 0x2794

Bits	Name
31..0	**TCC_MSW**—This 64-bit register has been divided into two 32-bit regsiters to support those architectures. 64-bit architectures can access the register atomically.
Options: No Soft Copy	

Offset 0xA0: Channel_Total_Transfer_Count_Status_Register_LSW (R)

Absolute Addresses:
AI_DMAChannel: 0x20A0
Counter0DMAChannel: 0x21A0
Counter1DMAChannel: 0x22A0
Counter2DMAChannel: 0x23A0
Counter3DMAChannel: 0x24A0
DI_DMAChannel: 0x25A0
AO_DMAChannel: 0x26A0
DO_DMAChannel: 0x27A0

Bits	Name
31..0	**TTCS_LSW**—This read-only field indicates the total number of Bytes that have been transferred to or from host memory by the DMA channel since the last time the CLR_TTC bit in the *Channel_Operation_Register* (CHOR) was written with a 1. Reads of this register are coherent with DMA data transfers—any data reported as moved by this register is guaranteed to have moved to or from system memory by the time software could act on the status. Assuming a system data width of 64-bits at a system clock rate of 133 MHz, this register represents over 548 years of uninterrupted data transfer. Note that special measures are required to read an atomic value from the register with a processor that cannot perform a 64-bit operation. In such a situation, software might read the upper half, then the lower half, then the upper half again. If the upper half changed between the two reads, software could safely use the larger value and assume a value of 0 for the lower half. Another option is to use the latching version of the register described below. Note also that this field resets to an unknown value, so it should be cleared with the CLR_TTC bit in the *Channel_Operation_Register* before use.
Options: No Soft Copy	

Offset 0xA4: Channel_Total_Transfer_Count_Status_Register_MSW (R)

Absolute Addresses:
AI_DMAChannel: 0x20A4
Counter0DMAChannel: 0x21A4
Counter1DMAChannel: 0x22A4
Counter2DMAChannel: 0x23A4
Counter3DMAChannel: 0x24A4
DI_DMAChannel: 0x25A4
AO_DMAChannel: 0x26A4
DO_DMAChannel: 0x27A4

Bits	Name
31..0	**TTCS_MSW**—This 64-bit register has been divided into two 32-bit regsiters to support those architectures. 64-bit architectures can access the register atomically.
Options: No Soft Copy	

Offset 0xA8: Channel_Total_Transfer_Count_Latching_Register_LSW (R)

Absolute Addresses:
AI_DMAChannel: 0x20A8
Counter0DMAChannel: 0x21A8
Counter1DMAChannel: 0x22A8
Counter2DMAChannel: 0x23A8
Counter3DMAChannel: 0x24A8
DI_DMAChannel: 0x25A8
AO_DMAChannel: 0x26A8
DO_DMAChannel: 0x27A8

Bits	Name
31..0	**TTCL_LSW**—This read-only field provides the same status as the previous register except it provides a latch for half the register so that 32-bit processors can read the value. This 32-bit access will return the value of the upper half at the same time as the last read of the lower half. This allows processors that cannot perform 64-bit accesses to read the 64-bit value without concern that it will change between reads. Note that multiple software threads cannot reliably access the same *Channel_Total_Transfer_Count_Latching_Register* unless a mutex is used.
Options: No Soft Copy	

Offset 0xAC: Channel_Total_Transfer_Count_Latching_Register_MSW (R)

Absolute Addresses:
AI_DMAChannel: 0x20AC
Counter0DMAChannel: 0x21AC
Counter1DMAChannel: 0x22AC
Counter2DMAChannel: 0x23AC
Counter3DMAChannel: 0x24AC
DI_DMAChannel: 0x25AC
AO_DMAChannel: 0x26AC
DO_DMAChannel: 0x27AC

Bits	Name
31..0	**TTCL_MSW**—This 64-bit register has been divided into two 32-bit registers to support those architectures. 64-bit architectures can access the register atomically.
Options: No Soft Copy	

Enumerated Types

DMA_Mode_t

Value	Name
1	**Normal DmaMode**
2	**LinkChainDmaMode**

8

SimultaneousControl Registers

SimultaneousControl base address = 0x6000

Each SimultaneousControl register is labeled as a read (R) or write (W) register.

List of SimultaneousControl Registers

The SimultaneousControl registers are as follows:

SimultaneousControl Registers

Offset 0x0: SignatureYear (R)

Absolute Address: 0x6000

Bits	Name
7..0	**SignatureYear_Default**

Offset 0x01: SignatureMonth (R)

Absolute Address: 0x6001

Bits	Name
7..0	**SignatureMonth_Default**

Offset 0x02: SignatureDay (R)

Absolute Address: 0x6002

Bits	Name
7..0	**SignatureDay_Default**

Offset 0x03: SignatureHour (R)

Absolute Address: 0x6003

Bits	Name
7..0	**SignatureHour_Default**

Offset 0x0C: Scratch (R|W)

Absolute Address: 0x600C

Bits	Name
7..0	**Scratch_Default**

Offset 0x0D: AISetChannelOrder (W)

Absolute Address: 0x600D

Bits	Name
7..4	**Reserved**
3..0	**AiChannel**—Current channel active for configuration.
Options: No Soft Copy	

Offset 0x0E: AIChanConfigCtrlStat (R|W)

Absolute Address: 0x600E

Bits	Name	
7..2	**Reserved**	
1..0	**AiGain**—Write to set the gain for the active AI Channel.	
	Value	**Meaning**
	00	**5 V Range**
	01	**10 V Range**
	10	**2 V Range**
	11	**1 V Range**
Options: No Soft Copy		

Offset 0x0F: AIClearChannelOrder (W)

Absolute Address: 0x600F

Bits	Name
7..1	**Reserved**
0	**AiClearConfig** (Strobe)—The AI Clear Channel Configuration Register clears the channel ordering/selection and the gain for all AI channels. This must always be performed at the start of each acquisition task. Writing a 1 to this bit clears the channel order and voltage ranges. This must be done at the start of each acquisition task. This bit clears itself.
Options: No Soft Copy	

Offset 0x10: AITriggerConfigCtrlStat (R|W)

Absolute Address: 0x6010

Bits	Name	
7..5	**Reserved**	
4..0	**AiTrigSel**—Write to set the MUX for the AI Trigger.	
	Value	**Meaning**
	00xxx	**Main**
	01xxx	**Ext**
	10xxx	**APFI0**
	11xxx	**APFI1**
	xx000	**Channel 0 (Main), Channel 8 (Ext)**
	xx001	**Channel 1 (Main), Channel 9 (Ext)**
	xx010	**Channel 2 (Main), Channel 10 (Ext)**
	xx011	**Channel 3 (Main), Channel 11 (Ext)**
	xx100	**Channel 4 (Main), Channel 12 (Ext)**
	xx101	**Channel 5 (Main), Channel 13 (Ext)**
	xx110	**Channel 6 (Main), Channel 14 (Ext)**
	xx111	**Channel 7 (Main), Channel 15 (Ext)**

Offset 0x14: AiFifoCtrlStat (R|W)

Absolute Address: 0x6014

Bits	Name
7	**ResetAiFifo**—Write Only. Write to reset the FIFO. Software needs to clear this signal; it does not clear itself.
6..4	**Reserved**
3	**ClrFifoIn16BitMode**—Write/Read. Write a 1 to set the FIFO in 32-bit wide mode. Writing a 0 has no effect. This bit clears itself.
2	**SetFifoIn16BitMode**—Write/Read. Write a 1 to set the FIFO into 16-bit wide mode. Writing a 0 has no effect. This bit clears itself.
1	**NotEmpty**—Read Only. Read this bit to return a 1 if the AI FIFO has data in it.
0	**SampleStranded**—Read Only. Reading this bit as a 1 indicates that a sample exists that has not been written into the AI FIFO.
Options: No Hardware Reset; No Soft Copy	

Offset 0x15: LoopbackCtrlStat (R|W)

Absolute Address: 0x6015

Bits	Name
7..4	**Reserved**
3	**ClrLoopPos**—Write a 1 to disable LoopPos. Reading a 1 means Loopback is disabled.
2	**SetLoopPos**—Write a 1 to enable LoopPos. Reading a 1 means Loopback is enabled.
1	**ClrLoopNeg**—Write a 1 to disable LoopNeg. Reading a 1 means Loopback is disabled.
0	**SetLoopNeg**—Write a 1 to enable LoopNeg. Reading a 1 means Loopback is enabled.
Options: No Soft Copy	

Offset 0x16: LoopbackSourceSel (R|W)

Absolute Address: 0x6016

Bits	Name	
7..5	**Reserved**	
4..0	**LoopbackMuxSel**—Write to select the source of the Loopback Mux. A read returns its current channel selected.	
	Value	**Meaning**
	00000	**_aiGnd_vs_aiGnd**
	00100	**_ao0_vs_aoGnd**
	00101	**_ao1_vs_aoGnd**
	00110	**_ao2_vs_aoGnd**
	00111	**_ao3_vs_aoGnd**
Options: No Hardware Reset		

Offset 0x17: TempSensorCtrlStat (R|W)

Absolute Address: 0x6017

Bits	Name
7..2	**Reserved**
1	**TempReady**—Read to check when new Temp Sensor data is available.
0	**TempStart**—Write to start a read to the serial Temp Sensor.
Options: No Hardware Reset, No Soft Copy	

Offset 0x18: TempSensorDataHi (R)

Absolute Address: 0x6018

Bits	Name
7..0	**TempUpperByte**—Binary Two's complement temperature.

Offset 0x19: TempSensorDataLo (R)

Absolute Address: 0x6019

Bits	Name
7..0	**TempLowerByte**—Binary Two's complement temperature.

Offset 0x1A: InterruptControl (W)

Absolute Address: 0x601A

Bits	Name
7	**GlobalInterruptEnable** (Strobe)—Write a 1 to enable the corresponding interrupt. Writing a 0 has no effect.
6	**GlobalInterruptDisable** (Strobe)—Write a 1 to disable the corresponding interrupt. Writing a 0 has no effect.
5	**Reserved**
4	**AiFifoOverflowInterruptEnable** (Strobe)—Write a 1 to enable the corresponding interrupt. Writing a 0 has no effect.
3	**AiFifoOverflowInterruptDisable** (Strobe)—Write a 1 to disable the corresponding interrupt. Writing a 0 has no effect.
2	**AiFifoOverflowInterruptAck** (Strobe)—Write a 1 to acknowledge the interrupt. The interrupt will not deassert if the condition that caused it is still pending. Writing a 0 has no effect.
1..0	**Reserved**
Options: No Soft Copy	

Offset 0x1A: InterruptStatus (R)

Absolute Address: 0x601A

Bits	Name
7	**GlobalInterruptEnabled**—Reading this bit returns the status of the interrupt enable.
6	**GlobalInterruptStatus**—Reading this bit returns a 1 if pin aIntOut is asserted.
5	**GlobalInterruptCondition**—Reading this bit returns a 1 if the corresponding condition is True. The aIntOut pin may or may not be asserted depending on whether the corresponding Interrupt Enable is True. Writing to these bits has no effect.
4	**AiFifoOverflowInterruptEnabled**—Reading this bit returns the status of the interrupt enable.
3..2	**Reserved**

Bits	Name
1	**AiFifoOverflowInterruptStatus**—Reading these bits returns a 1 if the corresponding interrupt is actively causing the aIntOut pin to assert.
0	**AiFifoOverflowInterruptCondition**—Reading these bit returns a 1 if the corresponding condition is True. The aIntOut pin may or may not be asserted depending on whether the corresponding Interrupt Enable is True. Writing to these bits has no effect.
Options: No Soft Copy	

Offset 0x1B: AcquisitionCtrl (R|W)

Absolute Address: 0x601B

Bits	Name
7..2	**Reserved**
1	**ExtendedChannelsPresent**—Read only. Reflects the status of the ExtendedChannelsPresent signal.
0	**AcquisitionDone**—Write Only. Software writes to this bit to indicate the last CONVERT.
Options: No Soft Copy	

Offset 0x1C: DcmCtrlStat (R|W)

Absolute Address: 0x601C

Bits	Name
7	**Stc3DcmIsLocked**—Read to this bit to verify the lock status of the DCM.
6	**Stc3DcmResetStatus**—Reading this bit indicates the current status of the DCM Reset signal.
5	**ClrStc3DcmReset**—Write to this bit to clear the DCM Reset.
4	**SetStc3DcmReset**—Write to this bit to assert the DCM Reset.
3	**ExtDcmIsLocked**—Read to this bit to verify the lock status of the DCM.
2	**ExtDcmResetStatus**—Reading this bit indicates the current status of the DCM Reset signal.
1	**ClrExtDcmReset**—Write to this bit to clear the DCM Reset.
0	**SetExtDcmReset**—Write to this bit to assert the DCM Reset.
Options: No Hardware Reset; No Soft Copy	

9

Board Services Registers

BrdServices base address = 0x20000

The board services sections are as follows:

- *SCXI Registers*
- *PWM Registers*
- *Global Registers*
- *Watchdog Timer Registers*
- *GenIRQ Registers*
- *Enumerated Types*

Each register is labeled as a read (R) or write (W) register.

List of Board Services Registers

The SCXI registers are as follows:

Offset 0x10: SCXI_Serial_Data_In_Register (R)

Offset 0x10: SCXI_Serial_Data_Out_Register (W)

Offset 0x12: SCXI_Control_Register (R|W)

Offset 0x14: SCXI_Output_Enable_Register (R|W)

Offset 0x16: SCXI_Status_Register (R)

Offset 0x16: SCXI_Mux_Clock_Register (W)

The PWM registers are as follows:

Offset 0x18: GenPwmPageSpec_i (i) Array (W)

Offset 0x24: Gen_PWM_i (i) Array (W)

The Global registers are as follows:

Offset 0x04: ScratchPadRegister (R|W)

Offset 0x60: Signature_Register (R)

Offset 0x64: TimeSincePowerUpRegister (R)

Offset 0x64: Joint_Reset_Register (W)

The Watchdog Timer registers are as follows:

Offset 0x68: WatchdogStatusRegister (R)

Offset 0x68: WatchdogTimeoutRegister (W)

Offset 0x6C: WatchdogConfiguration Register (W)

Offset 0x6E: WatchdogControl Register (W)

The GenIRQ registers are as follows:

Offset 0x70: Gen_Interrupt1_Register (W)

Offset 0x74: Gen_Interrupt2_Register (W)

List of Enumerated Types

The enumerated types are as follows:

STC3_Signature_t

BrdSrv_SCXI_Force_AI_EXTMUX_CLK_Width_t

BrdSrv_SCXI_HW_Serial_Timebase_t

BrdSrv_SCXI_Trig1_Output_Select_t

BrdSrv_WatchdogTimerExtSrcSel_t

BrdSrv_WatchdogTimerStateMachineSt_t

SCXI Registers

Offset 0x10: SCXI_Serial_Data_In_Register (R)

Absolute Address: 0x20010

Bits	Name
7..0	**SCXI_Data_In** (Strobe)—This register represents the last eight bits of data shifted in through the SCXI MISO line. Data is shifted in on the rising edge of SPIClk regardless of whether it is in hardware- or software-timed mode.

Offset 0x10: SCXI_Serial_Data_Out_Register (W)

Absolute Address: 0x20010

Bits	Name
7..0	**SCXI_Data_Out** (Strobe)—When the hardware communications is enabled, this register should be written with the data to be shifted. When hardware communications is disabled, bit 0 of this register is output on the SCXI MOSI line.
Options: No Soft Copy	

Offset 0x12: SCXI_Control_Register (R|W)

Absolute Address: 0x20012

Bits	Name
7	**SCXI_HW_Serial_Enable**—Set this bit to enable the HW_Serial logic. When enabled, the SPI_Clk comes from HW_Serial logic.
6	**SCXI_HW_Serial_Start** (Strobe)—Writing 1 to this bit starts an SCXI Shift Out as long as HW_Serial_Enable is set. The SCXI_Shift_In_Prog bit should be read to make sure there is no Serial Shift occurring before writing to this bit.
5	**SCXI_HW_Serial_Timebase**—Determines the HW_Serial_Timebase. The values for this bitfield are in *BrdSrv_SCXI_HW_Serial_Timebase_t*.
4	**SCXI_D_A**—This register is output onto the SCXI D_A line.
3	**SCXI_Intr**—This register is output onto the SCXI Intr line.
2	**SCXI_Front_Panel_MISO_Enable**—When this bit is set to 1, the front panel MISO signal is used to shift data into the SCXI Input register. When both this bit and the SCXI_Back_Plane_MISO_Enable bit are set, the Input register shifts in the logical AND of the Front and Back MISO signals.

Bits	Name
1	**SCXI_Back_Plane_MISO_Enable**—When this bit is set to 1, the back plane MISO signal is used to shift data into the SCXI Input register. When both this bit and the SCXI_Front_Panel_MISO_Enable bit are set, the Input register shifts in the logical AND of the Front and Back MISO signals.
0	**SCXI_SW_SPIClk**—This register is output onto the SCXI SPIClk line when the HW_Serial_Enable bit is written to 0.
Options: Initial Value = 0xA1	

Offset 0x14: SCXI_Output_Enable_Register (R|W)

Absolute Address: 0x20014

Bits	Name
7..3	**SCXI_Trig1_Output_Select**—This bitfield selects the source for SCXI_Trig1. The values for this bitfield are in *BrdSrv_SCXI_Trig1_Output_Select_t*.
2	**SCXI_Trig0_Output_Enable**—This bit enables the SCXI_Trig0 signal for output.
1	**SCXI_Trig1_Output_Enable**—This bit enables the SCXI_Trig1 signal for output.
0	**SCXI_Dedicated_Output_Enable**—This bit enables the output for the SCXI D_A, Intr, and MOSI lines that go to the dedicated SCXI lines on the PXI backplane.

Offset 0x16: SCXI_Status_Register (R)

Absolute Address: 0x20016

Bits	Name
7..1	**Reserved**
0	**SCXI_Shift_In_Prog** (Strobe)—This bit reads a 1 when the Hardware SCXI Communications Engine is shifting data. This bit should be polled for 0 before starting another SCXI shift operation with the SCXI_HW_Serial_Start bit.

Offset 0x16: SCXI_Mux_Clock_Register (W)

Absolute Address: 0x20016

Bits	Name
7..1	**Reserved**
0	**SCXI_Force_AI_EXTMUX_CLK_Width**—This bit affects the AI_ExternalMux_Clk signal when the AI_EXTMUX_CLK_Pulse_Width bit is set to 0. The values for this bitfield are in *BrdSrv_SCXI_Force_AI_EXTMUX_CLK_Width_t*.

PWM Registers

Offset 0x18: GenPwmPageSpec_i (i) Array (W)

Absolute Address: 0x20018

Array Offsets = 0x18 + (i × 1) [7 registers]

Bits	Name
7..0	**GenPwmNumPagesSpec_i**—This field sets the number of page bits that the current PWM is going to operate with. This number is programmable from 0 to 10. Programming a number outside this range is invalid and could cause wrong behavior. The number of page bits determines the number of windows the PWM uses to distribute the ones in a normal PWM period. The number of windows is $2^{NumPages}$.

Offset 0x24: Gen_PWM_i (i) Array (W)

Absolute Address: 0x20024

Array Offsets = 0x24 + (i × 2) [7 registers]

Bits	Name
15..0	**Gen_PWM_i_Duty_Cycle**—These PWMs have programmable frequency and page bits. The fundamental frequency is 50 MHz. PWM *n* Duty Cycle—The duty cycle of the PWM output is expressed by the formula: DutyCycle = 100% × (PWM *n* Duty Cycle/65536) **Note**: 0 is reset value and represents 0% duty cycle, and 65535 represents the maximum duty cycle possible (99.9985%). The number of page bits (determined by the GenPwmNumPagesSpec_i bitfield) control how the high time is spread on the period of the signal. The higher the page bits, the more spread the high time will be. This increments the effective frequency of the signal making it easier to filter. The trade-off is that it makes the PWM less linear due to the difference in transition times between rising and falling edges.

Global Registers

Offset 0x04: ScratchPadRegister (R|W)

Absolute Address: 0x20004

Bits	Name
31..0	**Scratch_Pad**—This is a simple readable and writable register that is included for self-test.

Offset 0x60: Signature_Register (R)

Absolute Address: 0x20060

Bits	Name
31..0	**STC3_Revision**—This bitfield contains the revision data of the chip in the form YYMMDDHH. YY = year (20YY). MM = month. DD = day. HH = Hour (24 hour time). The value of the register is 0x08050509 for revision A STC3 ASICs and 0x08050501 for revision B STC3 ASICs.

Offset 0x64: TimeSincePowerUpRegister (R)

Absolute Address: 0x20064

Bits	Name
31..0	**TimeSincePowerUpValue**—This number indicates the time since the last reset on the bus. The time can be calculated as $2^{16} \times (Osc\ Period) \times N$. With a 100 MHz oscillator, this number comes out to be 0.65536 ms × *N*.

Offset 0x64: Joint_Reset_Register (W)

Absolute Address: 0x20064

Bits	Name
15..1	**Reserved**
0	**Software_Reset** (Strobe)—Setting this bit to 1 resets the DAQ-STC3 endpoint. One difference between a hardware reset and a software reset is that personality bits are *not* cleared on a software reset. This bit is a strobe and clears itself.
Options: No Soft Copy	

Watchdog Timer Registers

Offset 0x68: WatchdogStatusRegister (R)

Absolute Address: 0x20068

Bits	Name
31..16	**Reserved**
15..8	**WatchdogExpiredCnt**—This bitfield reflects how many times the Watchdog Timer has expired since the last reset. This includes internal and external triggers, if enabled.
7..3	**Reserved**
2..0	**WatchdogSM_State**—This bitfield reflects the status of the Watchdog state machine. The values for this bitfield are in *BrdSrv_WatchdogTimerStateMachineSt_t*.

Offset 0x68: WatchdogTimeoutRegister (W)

Absolute Address: 0x20068

Bits	Name
31..0	**WatchdogTimeoutValue**—This bitfield sets the timeout value for the Watchdog Timer in the DAQ-STC3. The value represents the number of bus clock periods that the counter counts before expiring if it does not get acknowledged from software. When the timer expires, it can generate a pulse that is exported to I/O resources (such as AO or DO), and to RTSI. The modules receiving this pulse can take action, such as placing the outputs in a safe state.

Offset 0x6C: WatchdogConfiguration Register (W)

Absolute Address: 0x2006C

Bits	Name
15..10	**Reserved**
9	**WatchdogIntTrigEn**—This bit enables the generation of Expired trigger from the internal counter. When this bit is set, any time the timer gets to zero, the Watchdog Timer will generate an expired trigger pulse.
8	**WatchdogExtTrigEn**—This bit enables the generation of an Expired Trigger pulse from the external trigger source. When this bit is set, the Watchdog Timer generates a trigger any time that it detects a rising edge in the selected external trigger (after polarity selection).
7	**WatchdogExtTrigPol**—This bit selects the polarity of the external trigger signal. Set to 0 for active high, and to 1 for active low.
6..3	**Reserved**
2..0	**WatchdogExtTrigSel**—This bitfield selects the source for the external trigger source. The values for this bitfield are in *BrdSrv_WatchdogTimerExtSrcSel_t*.

Offset 0x6E: WatchdogControl Register (W)

Absolute Address: 0x2006E

Bits	Name
15..0	**WatchdogCommand** (Strobe)—This register must be pinged with alternating 0xFEED and 0xF00D commands to keep the Watchdog Timer from expiring. The expiration interval is set by the *WatchdogTimeout Register*. After reset, 0x5678 should be used to start the sequence. Writing 0xFEED or 0xF00D out of sequence causes an immediate expiration of the timer. 0x1234 can be written to pause the Watchdog. 0x5678 can be used to resume the Watchdog. After resuming from the pause, the device expects the 0xFEED/0xF00D sequence to restart. That is, 0xFEED first. You can force the timer to expire immediately by writing 0xDEAD. After the Watchdog Timer expires, there are several ways to acknowledge and continue. If in the expired state, the WDT receives the 0x1234 command, and then goes to the paused state. From there, it can be restarted by writing 0x5678. Alternatively, one can write the 0xACED command to immediately restart the WDT. In either case, the WDT starts with a fresh count that would restart the 0xFEED/0xF00D sequence, expecting 0xFEED first.
Options: No Soft Copy	

GenIRQ Registers

Offset 0x70: Gen_Interrupt1_Register (W)

Absolute Address: 0x20070

Bits	Name
31..18	**Reserved**
17	**PLL_OutOfLockIRQ_Ack** (Strobe)—This bit clears the PLL_OutOfLockIRQ Interrupt event and acknowledges the interrupt.
16	**WDT_TriggerIRQ_Ack** (Strobe)—This bit clears the WDT_TriggerIRQ Interrupt event and acknowledges the interrupt.
15..2	**Reserved**
1	**PLL_OutOfLockIRQ_Enable** (Strobe)—This strobe bit enables the PLL_OutOfLockIRQ interrupt.
0	**WDT_TriggerIRQ_Enable** (Strobe)—This strobe bit enables the WDT_TriggerIRQ interrupt.
Options: No Soft Copy	

Offset 0x74: Gen_Interrupt2_Register (W)

Absolute Address: 0x20074

Bits	Name
31..18	**Reserved**
17	**PLL_OutOfLockIRQ_Ack2** (Strobe)—This bit clears the PLL_OutOfLockIRQ Interrupt event and acknowledges the interrupt.
16	**WDT_TriggerIRQ_Ack2** (Strobe)—This bit clears the WDT_TriggerIRQ Interrupt event and acknowledges the interrupt.
15..2	**Reserved**
1	**PLL_OutOfLockIRQ_Disable** (Strobe)—This strobe bit disables the PLL_OutOfLockIRQ interrupt.
0	**WDT_TriggerIRQ_Disable** (Strobe)—This strobe bit disables the WDT_TriggerIRQ interrupt.
Options: No Soft Copy	

Enumerated Types

STC3_Signature_t

Value	Name
0x08050501	**STC3_RevBSignature**
0x08050509	**STC3_RevASignature**

BrdSrv_SCXI_Force_AI_EXTMUX_CLK_Width_t

Value	Name
0	**AI_ExtMux_Clk_250ns**—Pulsewidth is 250 ns.
1	**AI_ExtMux_Clk_500ns**—Pulsewidth is 500 ns.

BrdSrv_SCXI_HW_Serial_Timebase_t

Value	Name
0	**SxciSrc_100kHz**
1	**ScxiSrc_FastTB**—The Fast timebase for SCXI communication is 1200 ns (833.3 kHz).

BrdSrv_SCXI_Trig1_Output_Select_t

Value	Name	Value	Name
0	**SxciTrig1_Out_PFI0**	13	**SxciTrig1_Out_PFI13**
1	**SxciTrig1_Out_PFI1**	14	**SxciTrig1_Out_PFI14**
2	**SxciTrig1_Out_PFI2**	15	**SxciTrig1_Out_PFI15**
3	**SxciTrig1_Out_PFI3**	16	**SxciTrig1_Out_RTSI0**
4	**SxciTrig1_Out_PFI4**	17	**SxciTrig1_Out_RTSI1**
5	**SxciTrig1_Out_PFI5**	18	**SxciTrig1_Out_RTSI2**
6	**SxciTrig1_Out_PFI6**	19	**SxciTrig1_Out_RTSI3**
7	**SxciTrig1_Out_PFI7**	20	**SxciTrig1_Out_RTSI4**
8	**SxciTrig1_Out_PFI8**	21	**SxciTrig1_Out_RTSI5**
9	**SxciTrig1_Out_PFI9**	22	**SxciTrig1_Out_RTSI6**
10	**SxciTrig1_Out_PFI10**	23	**SxciTrig1_Out_RTSI7**
11	**SxciTrig1_Out_PFI11**	24	**SxciTrig1_Out_AI_Start**
12	**SxciTrig1_Out_PFI12**	31	**SxciTrig1_Out_Low**

BrdSrv_WatchdogTimerExtSrcSel_t

Value	Name	Value	Name
0	**WdtSrc_RTSI0**	4	**WdtSrc_RTSI4**
1	**WdtSrc_RTSI1**	5	**WdtSrc_RTSI5**
2	**WdtSrc_RTSI2**	6	**WdtSrc_RTSI6**
3	**WdtSrc_RTSI3**	7	**WdtSrc_RTSI7**

BrdSrv_WatchdogTimerStateMachineSt_t

Value	Name	Value	Name
0	**WdtSt_SynchReset**	3	**WdtSt_Sleeping**
1	**WdtSt_CountDownFeed**	4	**WdtSt_ExpiredPulse**
2	**WdtSt_CountDownF00D**	5	**WdtSt_Expired**

10

Bus Interface Registers

BusInterface base address = 0x20000

This chapter consists of *I_O_Bus Registers*. Each register is labeled as a read (R) or write (W) register.

List of Bus Interface Registers

The I_O_Bus registers are as follows:

I_O_Bus Registers

Offset 0x70: GlobalInterruptStatus_Register (R)

Absolute Address: 0x20070

Bits	Name
15..11	**Reserved**
10	**Gen_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the Generic Group occurs and is enabled, therefore causing a CPU interrupt.
9..8	**Reserved**
7	**DO_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the DO subsystem occurs and is enabled, therefore causing a CPU interrupt.
6	**DI_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the DI subsystem occurs and is enabled, therefore causing a CPU interrupt.
5	**G3_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the Counter 3 subsystem occurs and is enabled, therefore causing a CPU interrupt.
4	**G2_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the Counter 2 subsystem occurs and is enabled, therefore causing a CPU interrupt.
3	**G1_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the Counter 1 subsystem occurs and is enabled, therefore causing a CPU interrupt.
2	**G0_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the Counter 0 subsystem occurs and is enabled, therefore causing a CPU interrupt.
1	**AO_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the AO subsystem occurs and is enabled, therefore causing a CPU interrupt.
0	**AI_Interrupt_Status**—This bitfield is set to 1 when any enabled interrupt in the AI subsystem occurs and is enabled, therefore causing a CPU interrupt.

Offset 0x72: AI_Interrupt_Status_Register (R)

Absolute Address: 0x20072

Bits	Name
15	**AI_FifoIrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_FIFO Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
14	**AI_OverrunIrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_Overrun Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.

Bits	Name
13	**AI_StopIrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_STOP Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
12	**AI_StartIrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_START Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
11	**AI_Start2IrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_START2 Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
10	**AI_Start1IrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_START1 Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
9	**AI_SC_TC_IrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_SC_TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
8	**AI_ScanOverrunIrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_ScanOverrun Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
7	**AI_SC_PreWaitRollOverSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_SC_PreWaitRollOver Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
6	**AI_OverflowIrqSt**—This bitfield reflects the component of the AI IRQ vector that corresponds to the AI_Overflow Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
5..0	**Reserved**

Offset 0x74: AO_Interrupt_Status_Register (R)

Absolute Address: 0x20074

Bits	Name
15	**AO_FifoIrqSt**—This bitfield reflects the component of the AO IRQ vector that corresponds to the AO_FIFO Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
14	**AO_UC_TC_IrqSt**—This bitfield reflects the component of the AO IRQ vector that corresponds to the AO_UC_TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
13	**AO_ErrorIrqSt**—This bitfield reflects the component of the AO IRQ vector that corresponds to the AO_Error Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.

Bits	Name
12	**AO_UpdateIrqSt**—This bitfield reflects the component of the AO IRQ vector that corresponds to the AO_UPDATE Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
11	**AO_Start1IrqSt**—This bitfield reflects the component of the AO IRQ vector that corresponds to the AO_START1 Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
10	**AO_BC_TC_IrqSt**—This bitfield reflects the component of the AO IRQ vector that corresponds to the AO_BC_TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
9	**AO_WriteTooFastIrqSt**—This bitfield reflects the component of the AO IRQ vector that corresponds to the AO_WriteTooFast Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
8..0	**Reserved**

Offset 0x76: TIO_Interrupt_Status_Register (i) Array (R)

Absolute Address: 0x20076

Array Offsets = 0x76 + (i × 2) [4 registers]

Bits	Name
15..13	**Reserved**
12	**AuxCtrTC_ErrorIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the Aux Ctr TC Error Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
11	**AuxCtrTC_IrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the Aux Ctr TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
10	**DisarmEventIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the Disarm Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
9	**SampleClockIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the Sample Clock Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
8	**TIO_GateSwitchErrorIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the GateSwitchError Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
7	**TIO_WritesTooFast**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the WritesTooFast Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.

Bits	Name
6	**TIO_DMA_ErrorIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the DMA_Error Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
5	**TIO_TC_ErrorIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the TC_Error Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
4	**TIO_GateErrorIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the GateError Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
3	**TIO_SampleClkErrorSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the SampleClkError Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
2	**TIO_TC_IrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
1	**TIO_GateIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the Gate Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
0	**TIO_DmaIrqSt**—This bitfield reflects the component of the TIO IRQ vector that corresponds to the DMA Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.

Offset 0x78: GlobalInterruptEnable_Register (W)

Absolute Address: 0x20078

Bits	Name
31..27	**Reserved**
26	**Gen_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the Generic Group from propagating to the CHInCh.
25..24	**Reserved**
23	**DO_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the DO subsystem from propagating to the CHInCh.
22	**DI_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the DI subsystem from propagating to the CHInCh.
21	**G3_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the Counter 3 subsystem from propagating to the CHInCh.
20	**G2_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the Counter 2 subsystem from propagating to the CHInCh.

Bits	Name
19	**G1_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the Counter 1 subsystem from propagating to the CHInCh.
18	**G0_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the Counter 0 subsystem from propagating to the CHInCh.
17	**AO_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the AO subsystem from propagating to the CHInCh.
16	**AI_Interrupt_Disable** (Strobe)—Write 1 to this field to block all of the interrupts in the AI subsystem from propagating to the CHInCh.
15..11	**Reserved**
10	**Gen_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the Generic Group to propagate to the CHInCh.
9..8	**Reserved**
7	**DO_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the DO subsystem to propagate to the CHInCh.
6	**DI_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the DI subsystem to propagate to the CHInCh.
5	**G3_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the Counter 3 subsystem to propagate to the CHInCh.
4	**G2_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the Counter 2 subsystem to propagate to the CHInCh.
3	**G1_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the Counter 1 subsystem to propagate to the CHInCh.
2	**G0_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the Counter 0 subsystem to propagate to the CHInCh.
1	**AO_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the AO subsystem to propagate to the CHInCh.
0	**AI_Interrupt_Enable** (Strobe)—Write 1 to this field to allow all of the interrupts in the AI subsystem to propagate to the CHInCh.
Options: No Soft Copy	

Offset 0x7E: DI_Interrupt_Status_Register (R)

Absolute Address: 0x2007E

Bits	Name
15	**DI_FifoIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_FIFO Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
14	**DI_OverrunIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_Overrun Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
13	**DI_StopIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_STOP Interrupt. This means that this bitfield is hasn't only if the interrupt occurred, the interrupt is enabled, and the interrupt hasn’t been acknowledged.
12	**DI_StartIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_START Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
11	**DI_Start2IrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_START2 Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
10	**DI_Start1IrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_START1 Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
9	**DI_SC_TC_IrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_SC_TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
8	**DI_ScanOverrunIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_ScanOverrun Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
7	**DI_SC_PreWaitRollOverSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_SC_PreWaitRollOver Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
6	**DI_OverflowIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_Overflow Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
5..2	**Reserved**
1	**DI_ChangeDetectionErrorIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_ChangeDetectError Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
0	**DI_ChangeDetectionIrqSt**—This bitfield reflects the component of the DI IRQ vector that corresponds to the DI_ChangeDetectStatus Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.

Offset 0x80: DO_Interrupt_Status_Register (R)

Absolute Address: 0x20080

Bits	Name
15	**DO_FifoIrqSt**—This bitfield reflects the component of the DO IRQ vector that corresponds to the DO_FIFO Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
14	**DO_UC_TC_IrqSt**—This bitfield reflects the component of the DO IRQ vector that corresponds to the DO_UC_TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
13	**DO_ErrorIrqSt**—This bitfield reflects the component of the DO IRQ vector that corresponds to the DO_Error Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
12	**DO_UpdateIrqSt**—This bitfield reflects the component of the DO IRQ vector that corresponds to the DO_UPDATE Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
11	**DO_Start1IrqSt**—This bitfield reflects the component of the DO IRQ vector that corresponds to the DO_START1 Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
10	**DO_BC_TC_IrqSt**—This bitfield reflects the component of the DO IRQ vector that corresponds to the DO_BC_TC Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
9	**DO_WriteTooFastIrqSt**—This bitfield reflects the component of the DO IRQ vector that corresponds to the DO_WriteTooFast Interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
8..0	**Reserved**

Offset 0x86: Gen_Interrupt_Status_Register (R)

Absolute Address: 0x20086

Bits	Name
15..2	**Reserved**
1	**PLL_OutOfLockEventSt**—This bitfield reflects the status of the PLL_OutOfLock interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.
0	**WatchdogTimerTriggerSt**—This bitfield reflects the status of the WatchdogTimer interrupt. This means that this bitfield is 1 only if the interrupt occurred, is enabled, and has not been acknowledged.

11

Triggers and Timing Registers

Triggers base address = 0x20000

This chapter consists of *Triggers and Timing Registers* and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

List of Triggers and Timing Registers

The triggers and timing registers are as follows:

List of Enumerated Types

The enumerated types are as follows:

PFI_WDT_Mode_t

Trig_Analog_Trigger_Mode_t

Trig_Atrig_Sel_t

Trig_Filter_Custom_Timebase_t

Trig_Filter_Ext_Signal_Select_t

Trig_Filter_Select_t

Trig_FOUT_Enable_t

Trig_FOUT_FastTB_DivideBy2_t

Trig_FOUT_Timebase_Select_t

Trig_IntTriggerA_i_Output_Select_t

Trig_PFI_i_Pin_Dir_t

Trig_PFI_Output_Select_t

Trig_PLL_Filter_Range_t

Trig_PLL_In_Source_Select_t

Trig_RTSI_i_Output_Select_t

Trig_RTSI_i_Pin_Dir_t

Trig_StarTrig_Output_Select_t

Trig_TB3_Select_t

TrigPllLockedStatus_t

Triggers and Timing Registers

Offset 0xA0: AnalogTrigControlRegister (W)

Absolute Address:
Triggers: 0x200A0

Bits	Name
15..10	**Reserved**
9..8	**Atrig_Sel**—This bitfield sets the value of the Atrig_Sel pins. This is used to select between the output of the PGIA or the APFI pins. The values for this bitfield are in *Trig_Atrig_Sel_t*.
7..6	**Reserved**
5	**Analog_Trigger_Reset** (Strobe)—This bit clears the hysteresis registers in the analog trigger circuit. Set this bit to 1 when you arm a timing engine with which you want to use the analog trigger circuit. Before setting this bit to 1, make sure that the analog trigger is not being used by any other timing engine in the DAQ-STC3.
4..3	**Reserved**
2..0	**Analog_Trigger_Mode**—This bit selects the analog trigger mode of operation if the analog trigger circuitry is enabled. The values for this bitfield are in *Trig_Analog_Trigger_Mode_t*.
Options: No Hardware Reset	

Offset 0xA2: FOUT_Register (W)

Absolute Address:
Triggers: 0x200A2

Bits	Name
15	**FOUT_Enable**—Setting this bit to 1 enables and starts frequency output on the FOUT signal, which can be routed to PFI pins. To change the frequency divider value, first clear this bit, then change FOUT_Divider, and set this bit again. If this bit is clear, FOUT is disabled. The values for this bitfield are in *Trig_FOUT_Enable_t*.
14	**FOUT_Timebase_Select**—This bit selects the timebase used for FOUT, also known as FOUT_TIMEBASE. The values for this bitfield are in *Trig_FOUT_Timebase_Select_t*.
13	**Reserved**

Bits	Name
12	**FOUT_FastTB_DivideBy2**—This bit determines whether to divide the fast timebase used in the FOUT pin. The values for this bitfield are in *Trig_FOUT_FastTB_DivideBy2_t*.
11..4	**Reserved**
3..0	**FOUT_Divider**—This bit selects the divide ratio for the FOUT output signal. Set this bitfield to 0, for a division ratio of 16: FOUT = FOUT_TIMEBASE divided by 16. Otherwise, the division ratio will be the value of this bitfield (1 to 15).

Offset 0xA4: PFI_Direction_Register (W)

Absolute Address:
Triggers: 0x200A4

Bits	Name
15	**PFI15_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI15. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
14	**PFI14_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI14. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
13	**PFI13_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI13. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
12	**PFI12_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI12. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
11	**PFI11_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI11. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.

Bits	Name
10	**PFI10_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI10. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
9	**PFI9_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI9. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
8	**PFI8_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI8. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
7	**PFI7_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI7. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
6	**PFI6_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI6. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
5	**PFI5_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI5. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
4	**PFI4_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI4. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
3	**PFI3_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI3. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.

Bits	Name
2	**PFI2_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI2. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
1	**PFI1_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI1. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
0	**PFI0_Pin_Dir**—This bit selects the direction of the bidirectional pin PFI0. **Note**: Writing to this register also initiates a command transaction to the DAQ-6202, which sets the PFI pin to the right direction. This transaction holds the bus until complete. The values for this bitfield are in *Trig_PFI_i_Pin_Dir_t*.
Options: No Hardware Reset	

Offset 0xA6: RTSI_Trig_Direction_Register (W)

Absolute Address:
Triggers: 0x200A6

Bits	Name
15	**RTSI7_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI7. The values for this bitfield are in *Trig_RTSI_i_Pin_Dir_t*.
14	**RTSI6_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI6. The values for this bitfield are in *Trig_RTSI_i_Pin_Dir_t*.
13	**RTSI5_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI5. The values for this bitfield are in *Trig_RTSI_i_Pin_Dir_t*.
12	**RTSI4_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI4. The values for this bitfield are in *Trig_RTSI_i_Pin_Dir_t*.
11	**RTSI3_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI3. The values for this bitfield are in *Trig_RTSI_i_Pin_Dir_t*.
10	**RTSI2_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI2. The values for this bitfield are in *Trig_RTSI_i_Pin_Dir_t*.
9	**RTSI1_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI1. The values for this bitfield are in *Trig_RTSI_i_Pin_Dir_t*.

Bits	Name
8	**RTSI0_Pin_Dir**—This bit selects the output of the bidirectional pin RTSI0. The values for this bitfield are in the *Trig_RTSI_i_Pin_Dir_t*.
7..0	**Reserved**

Offset 0xA8: RTSI_OutputSelectRegister_i (i) Array (W)

Absolute Address:
Triggers: 0x200A8

Array Offsets = 0xA8 + (i × 1) [8 registers]

Bits	Name
7	**Reserved**
6..0	**RTSI_i_Output_Select**—RTSI line output source selection. The values for this bitfield are in *Trig_RTSI_i_Output_Select_t*.
Options: No Hardware Reset	

Offset 0xB0: PFI_Filter_Register_0 (W)

Absolute Address:
Triggers: 0x200B0

Bits	Name
15	**Reserved**
14..12	**PFI3_Filter_Select**—PFI line 3 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
11	**Reserved**
10..8	**PFI2_Filter_Select**—PFI line 2 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
7	**Reserved**
6..4	**PFI1_Filter_Select**—PFI line 1 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
3	**Reserved**
2..0	**PFI0_Filter_Select**—PFI line 0 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.

Offset 0xB2: PFI_Filter_Register_1 (W)

Absolute Address:
Triggers: 0x200B2

Bits	Name
15	**Reserved**
14..12	**PFI7_Filter_Select**—PFI line 7 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
11	**Reserved**
10..8	**PFI6_Filter_Select**—PFI line 6 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
7	**Reserved**
6..4	**PFI5_Filter_Select**—PFI line 5 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
3	**Reserved**
2..0	**PFI4_Filter_Select**—PFI line 4 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.

Offset 0xB4: PFI_Filter_Register_2 (W)

Absolute Address:
Triggers: 0x200B4

Bits	Name
15	**Reserved**
14..12	**PFI11_Filter_Select**—PFI line 11 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
11	**Reserved**
10..8	**PFI10_Filter_Select**—PFI line 10 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
7	**Reserved**
6..4	**PFI9_Filter_Select**—PFI line 9 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.

Bits	Name
3	**Reserved**
2..0	**PFI8_Filter_Select**—PFI line 8 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.

Offset 0xB6: PFI_Filter_Register_3 (W)

Absolute Address:
Triggers: 0x200B6

Bits	Name
15	**Reserved**
14..12	**PFI15_Filter_Select**—PFI line 15 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
11	**Reserved**
10..8	**PFI14_Filter_Select**—PFI line 14 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
7	**Reserved**
6..4	**PFI13_Filter_Select**—PFI line 13 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
3	**Reserved**
2..0	**PFI12_Filter_Select**—PFI line 12 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.

Offset 0xB8: STAR_Trig_Register (W)

Absolute Address:
Triggers: 0x200B8

Bits	Name
15	**Star_Trig_Pin_Dir**—This bit selects the direction of the Star_Trig pin.
14..4	**Reserved**
3..0	**Star_Trig_Output_Select**—This bit selects the signal routed to the Star_Trig pin when it is used as an output. The values for this bitfield are in *Trig_StarTrig_Output_Select_t*.

Offset 0xBA: PFI_OutputSelectRegister_i (i) Array (W)

Absolute Address:
Triggers: 0x200BA

Bits	Name
7	**Reserved**
6..0	**PFI_i_Output_Select**—PFI line output source selection. The values for this bitfield are in *Trig_PFI_Output_Select_t*.
Options: No Hardware Reset	

Offset 0xCA: DStarC_Trig_Register (W)

Absolute Address:
Triggers: 0x200CA

Bits	Name
15	**DStarC_Enable**—This bit enables DStarC to drive signals out to the system.
14..4	**Reserved**
3..0	**DStarC_Output_Select**—This bit selects the signal routed to the DStarC pin when enabled. The values for this bitfield are in *Trig_StarTrig_Output_Select_t*.

Offset 0xDA: Clock_And_Fout2_Register (W)

Absolute Address:
Triggers: 0x200DA

Bits	Name
15..7	**Reserved**
6	**TB3_Select**—This bit controls the mux that selects between the two possible sources for the internal clock TB3. The values for this bitfield are in *Trig_TB3_Select_t*.
5	**Reserved**
4..0	**PLL_In_Source_Select**—These bits select the Reference Clock In for the PLL on the DAQ-STC3 ASIC. The values for this bitfield are in *Trig_PLL_In_Source_Select_t*.

Offset 0xDC: PLL_Status_Register (R)

Absolute Address:
Triggers: 0x200DC

Bits	Name
15..2	**Reserved**
1	**HW_Pll_Locked**—This bit shows the status of the PLL Lock indicator. This signal indicates a Lock has occurred in the PLL. The Lock bit will go high when the phase margin between the input clock and the feedback clock is within 0–5% of the input period. This is the main lock indication for the PLL in the DAQ-STC3 ASIC. The values for this bitfield are in *TrigPllLockedStatus_t*.
0	**PLL_TimerExpired**—This bit shows the status of the PLL Lock timer. Any time any of the parameters of the PLL changes (PLL_Divisor, PLL_Multiplier Reference Clock In Source, or PLL_Enable), a timer starts counting and this bit is cleared. The counter counts for 1.1 mS, which is considerably more than the lock time for the PLL. When the timer expires, this bit sets. This bit does not check the actual locked status of the PLL. This bit can be used with the HW_Pll_Locked to infer a problem with the connections or operation of the PLL. When this bit asserts, but the HW_Pll_Locked is still 0, it means the PLL has not locked within the expected period of time. The values for this bitfield are in *TrigPllLockedStatus_t*.

Offset 0xDC: PLL_Control_Register (W)

Absolute Address:
Triggers: 0x200DC

Bits	Name
31	**PLL_Enable**—When this bit is set, it enables the PLL to start locking to the Reference Clock In signal. When this bit is cleared, the PLL is disabled, and the output of the PLL is forced to 0.
30..28	**PLL_Filter_Range**—These bits set the PLL loop filter to work with the post-reference divider frequency. Choose the highest valid range for best jitter performance. The values for this bitfield are in *Trig_PLL_Filter_Range_t*.
27..21	**PLL_RefDivisor**—Reference Divider Value (Binary value + 1, so 0 ≥ div by 1). These bits set the division ratio applied to the Reference Clock In before being used in the Phase comparator.
20..14	**PLL_Multiplier**—PLL Multiplier or Feedback Divider Value (Binary value +1, so 0 ≥ div by 1). These bits set the division ratio for the feedback path of the PLL.
13..8	**PLL_OutputDivider**—PLL Output Divider (binary value +1, so 0 ≥ div by 1). These bits set the division ratio for the output of the VCO.
7..0	**Reserved**
Options: Initial Value = 0x6000	

Offset 0xE0: PFI_DI_Register (R)

Absolute Address:
Triggers: 0x200E0

Bits	Name
15	**PFI_15_DI_Bitfield**—This bitfield will represent the status of PFI pin 15 when it is configured as an input.
14	**PFI_14_DI_Bitfield**—This bitfield will represent the status of PFI pin 14 when it is configured as an input.
13	**PFI_13_DI_Bitfield**—This bitfield will represent the status of PFI pin 13 when it is configured as an input.
12	**PFI_12_DI_Bitfield**—This bitfield will represent the status of PFI pin 12 when it is configured as an input.
11	**PFI_11_DI_Bitfield**—This bitfield will represent the status of PFI pin 11 when it is configured as an input.
10	**PFI_10_DI_Bitfield**—This bitfield will represent the status of PFI pin 10 when it is configured as an input.
9	**PFI_9_DI_Bitfield**—This bitfield will represent the status of PFI pin 9 when it is configured as an input.
8	**PFI_8_DI_Bitfield**—This bitfield will represent the status of PFI pin 8 when it is configured as an input.
7	**PFI_7_DI_Bitfield**—This bitfield will represent the status of PFI pin 7 when it is configured as an input.
6	**PFI_6_DI_Bitfield**—This bitfield will represent the status of PFI pin 6 when it is configured as an input.
5	**PFI_5_DI_Bitfield**—This bitfield will represent the status of PFI pin 5 when it is configured as an input.
4	**PFI_4_DI_Bitfield**—This bitfield will represent the status of PFI pin 4 when it is configured as an input.
3	**PFI_3_DI_Bitfield**—This bitfield will represent the status of PFI pin 3 when it is configured as an input.
2	**PFI_2_DI_Bitfield**—This bitfield will represent the status of PFI pin 2 when it is configured as an input.
1	**PFI_1_DI_Bitfield**—This bitfield will represent the status of PFI pin 1 when it is configured as an input.
0	**PFI_0_DI_Bitfield**—This bitfield will represent the status of PFI pin 0 when it is configured as an input.

Offset 0xE0: PFI_DO_Register (W)

Absolute Address:
Triggers: 0x200E0

Bits	Name
15..0	**PFI_DO_Bf**—These bitfields represent the value that will be output on the PFI pins when they are configured to output the PFI_DO mode.
Options: No Hardware Reset	

Offset 0xE2: PFI_WDT_SafeStateRegister (W)

Absolute Address:
Triggers: 0x200E2

Bits	Name
15..0	**PFI_WDT_SafeStateValue**—This is the safe value that the PFI lines will go to if the Watchdog Timer is enabled, and the PFI lines have the safe state enabled (through the *PFI_WDT_ModeSelect_Register*).

Offset 0xE4: PFI_WDT_ModeSelect_Register (W)

Absolute Address:
Triggers: 0x200E4

Bits	Name
31..30	**PFI_WDT_ModeD15**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
29..28	**PFI_WDT_ModeD14**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
27..26	**PFI_WDT_ModeD13**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
25..24	**PFI_WDT_ModeD12**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
23..22	**PFI_WDT_ModeD11**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.

Bits	Name
21..20	**PFI_WDT_ModeD10**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
19..18	**PFI_WDT_ModeD9**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
17..16	**PFI_WDT_ModeD8**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
15..14	**PFI_WDT_ModeD7**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
13..12	**PFI_WDT_ModeD6**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
11..10	**PFI_WDT_ModeD5**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
9..8	**PFI_WDT_ModeD4**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
7..6	**PFI_WDT_ModeD3**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
5..4	**PFI_WDT_ModeD2**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
3..2	**PFI_WDT_ModeD1**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.
1..0	**PFI_WDT_ModeD0**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *PFI_WDT_Mode_t*.

Offset 0xE8: IntTriggerA_OutputSelectRegister_i (i) Array (W)

Absolute Address:
Triggers: 0x200E8

Array Offsets = 0xE8 + (i × 1) [8 registers]

Bits	Name
7	**Reserved**
6..0	**IntTriggerA_i_Output_Select**—IntTriggerA line output source selection. The values for this bitfield are in *Trig_IntTriggerA_i_Output_Select_t*.
Options: No Hardware Reset	

Offset 0xF8: IntTrigA_Filter_Register_Lo (W)

Absolute Address:
Triggers: 0x200F8

Bits	Name
15	**Reserved**
14..12	**ITA3_Filter_Select**—Internal Trigger A line 3 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
11	**Reserved**
10..8	**ITA2_Filter_Select**—Internal Trigger A line 2 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
7	**Reserved**
6..4	**ITA1_Filter_Select**—Internal Trigger A line 1 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
3	**Reserved**
2..0	**ITA0_Filter_Select**—Internal Trigger A line 0 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.

Offset 0xFA: IntTrigA_Filter_Register_Hi (W)

Absolute Address:
Triggers: 0x200FA

Bits	Name
15	**Reserved**
14..12	**ITA7_Filter_Select**—Internal Trigger A line 7 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
11	**Reserved**
10..8	**ITA6_Filter_Select**—Internal Trigger A line 6 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
7	**Reserved**
6..4	**ITA5_Filter_Select**—Internal Trigger A line 5 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.
3	**Reserved**
2..0	**ITA4_Filter_Select**—Internal Trigger A line 4 filter selection. The values for this bitfield are in *Trig_Filter_Select_t*.

Offset 0x100: Trig_Filter_Settings1_Register (W)

Absolute Address:
Triggers: 0x20100

Bits	Name
15..13	**Trig_Filter_Ext_Signal_Select**—If External_Signal is selected as filter for a determined trigger line, that line must be stable for two assertions of this signal for a transition or pulse to be detected. The values for this bitfield are in *Trig_Filter_Ext_Signal_Select_t*.
12	**Reserved**
11..10	**Trig_Filter_Custom_Timebase_1**—If Custom_Filter_1 is selected as filter for a determined trigger line, this timebase is used for the dedicated counter of the line that waits for the signal to be stable a custom period of time before a transition or a pulse is detected. The values for this bitfield are in *Trig_Filter_Custom_Timebase_t*.
9	**Reserved**
8..0	**Trig_Filter_Custom_Period_1**—If Custom_Filter_1 is selected as filter for a determined trigger line, this timebase is used for the dedicated counter of the line that waits for the signal to be stable a custom period of time before a transition or a pulse is detected.

Offset 0x102: Trig_Filter_Settings2_Register (W)

Absolute Address:
Triggers: 0x20102

Bits	Name
15..12	**Reserved**
11..10	**Trig_Filter_Custom_Timebase_2**—If Custom_Filter_2 is selected as filter for a determined trigger line, this timebase is used for the dedicated counter of the line that waits for the signal to be stable a custom period of time before a transition or a pulse is detected. The values for this bitfield are in *Trig_Filter_Custom_Timebase_t*.
9	**Reserved**
8..0	**Trig_Filter_Custom_Period_2**—If Custom_Filter_2 is selected as filter for a determined trigger line, that line must be stable for this number of clock periods of the selected timebase for a transition or pulse to be detected.

Offset 0x104: PLL_LockCount_Register (W)

Absolute Address:
Triggers: 0x20104

Bits	Name
15..0	**PLL_LockCount**—This bitfield determines the delay that the PLL_TimerExpired status bit waits before expiring after the PLL has been enabled. The idea is to provide a minimum time as a lock period before the PLL is used, or a reference time to know when the PLL Locked bit will no longer assert because the max locked period has been exceeded. The PLL minimum lock time is specified at 50 µs at 10 MHz. The default value for this register is 12500. This counter runs on BusClk. So this value will translate to the following: • 100 µs when BusClock is 125 MHz • 325 µs when BusClock is 40.0 MHz • 400 µs when BusClock is 31.25 MHz • 650 µs when BusClock is 20.0 MHz This is only the default value. This register can be updated by software.
Options: Initial Value = 0xD6D8	

Offset 0x106: Sync100_Repeat_Count_Register (W)

Absolute Address:
Triggers: 0x20106

Bits	Name
7..0	**Sync100_Repeat_Count**—This bit sets the delay value for the Sync100 repeater circuit. The value written should be the value desired minus two (so the default value of 8 means the delay is 10 cycles).
Options: Initial Value = 0x08	

Enumerated Types

PFI_WDT_Mode_t

Value	Name
0	**WdtMode_Disabled**—When set to Disabled, there will be no action on a Watchdog Timer event. This is the default.
1	**WdtMode_Freeze**—When set to Freeze, the DO subsystem freezes all the routing registers, the static DO register, and the direction register. This enforces keeping the current state even if software does "bad" writes to the hardware (with the exception of the reset bits). This freezing of the registers happens on any other mode except for Disabled. This state does not impose a "safe value" on the line. Note that if the line was routed to something other than static DO (for example Timed DO), the value could change. The freeze does *not* stop any subsystems; it only preserves the routing and the static DO values. Software must acknowledge the WDT before the hardware will accept any changes to configuration, including this bit.
2	**WdtMode_Tristate**—When set to Tristate, the DO line goes to High Impedance whenever there is a WDT event. It also freezes all the routing registers. Software must acknowledge the WDT before the hardware will accept any changes to configuration, including this bit.
3	**WdtMode_SafeValue**—When set to SafeValue, the DO line goes to the predefined safe value whenever there is a WDT event. It also freezes all the routing registers. Software must acknowledge the WDT before the hardware will accept any changes to configuration, including this bit.

Trig_Analog_Trigger_Mode_t

Value	Name
0	**ATrigMode_Low_Window**
1	**ATrigMode_High_Window**
2	**ATrigMode_Middle_Window**
4	**ATrigMode_High_Hysteresis**
6	**ATrigMode_Low_Hysteresis**

Trig_Atrig_Sel_t

Value	Name
0	**ATrigSel_AI_Chan**—The output of the PGIA, which is the currently-selected AI channel.
1	**ATrigSel_APFI0**
2	**ATrigSel_APFI1**
3	**ATrigSel_Ground**

Trig_Filter_Custom_Timebase_t

Value	Name
0	**Filter_Timebase_TB3**—Timebase used to generate the filters' Internal Clock Signal is TB3.
1	**Filter_Timebase_TB2**—Timebase used to generate the filters' Internal Clock Signal is TB2.
2	**External_Signal**—Timebase used to generate the filters' Internal Clock Signal is selected with the Trig_Filter_Ext_Signal_Select bitfield in the *Trig_Filter_Settings1_Register*.

Trig_Filter_Ext_Signal_Select_t

Value	Name
0	**Filter_Ext_Signal_IntTriggerA0**
1	**Filter_Ext_Signal_IntTriggerA1**
2	**Filter_Ext_Signal_IntTriggerA2**
3	**Filter_Ext_Signal_IntTriggerA3**
4	**Filter_Ext_Signal_IntTriggerA4**
5	**Filter_Ext_Signal_IntTriggerA5**
6	**Filter_Ext_Signal_IntTriggerA6**
7	**Filter_Ext_Signal_IntTriggerA7**

Trig_Filter_Select_t

Value	Name
0	**No_Filter**—No filter.
1	**Sync_To_TB3**—Synchronize signal to TB3.
2	**Small_Filter**—Small Filter rejects less than 100 ns : Accepts greater than 100 ns : Delays 100 ns : Adds 25 ns Jitter.
3	**Medium_Filter**—Medium Filter rejects less than 6.4 µs : Accepts greater than 6.425 us : Delays 6.4 us : Adds 25 ns Jitter.
4	**Large_Filter**—Large Filter rejects less than 2.54 ms : Accepts greater than 2.56 ms : Delays 2.54 ms : Adds 10.025 µs Jitter.
5	**Custom_Filter_1**—Uses Trig_Filter_Custom_Timebase_1 and Trig_Filter_Custom_Period_1 selections from *Trig_Filter_Settings1_Register* to filter the signal.
6	**Custom_Filter_2**—Uses Trig_Filter_Custom_Timebase_2 and Trig_Filter_Custom_Period_2 selections from *Trig_Filter_Settings2_Register* to filter the signal.

Trig_FOUT_Enable_t

Value	Name
0	**FOUT_Disabled**
1	**FOUT_Enabled**

Trig_FOUT_FastTB_DivideBy2_t

Value	Name
0	**FOUT_FastTB_isTB1**—No, the timebase is TB1 = 20 MHz.
1	**FOUT_FastTB_isTB1_DivBy2**—Yes, the timebase is TB1 / 2 = 10 MHz.

Trig_FOUT_Timebase_Select_t

Value	Name
0	**FOUT_Src_IsFastTB**—FOUT_TIMEBASE = TB1 if FOUT_FastTB_DivideBy2 is 0. FOUT_TIMEBASE = TB1/2 if FOUT_FastTB_DivideBy2 is 1.
1	**FOUT_Src_IsTB2**—FOUT_TIMEBASE = TB2.

Trig_IntTriggerA_i_Output_Select_t

Value	Name	Value	Name
0	**IntTriggerA_PFI0**	42	**IntTriggerA_AI_Gate**
1	**IntTriggerA_PFI1**	43	**IntTriggerA_DI_Convert**
2	**IntTriggerA_PFI2**	44	**IntTriggerA_DI_Start1**
3	**IntTriggerA_PFI3**	45	**IntTriggerA_DI_Start2**
4	**IntTriggerA_PFI4**	46	**IntTriggerA_DI_Gate**
5	**IntTriggerA_PFI5**	55	**IntTriggerA_AO_UPDATE**
6	**IntTriggerA_PFI6**	56	**IntTriggerA_AO_START1**
7	**IntTriggerA_PFI7**	57	**IntTriggerA_AO_Gate**
8	**IntTriggerA_PFI8**	58	**IntTriggerA_DO_Update**
9	**IntTriggerA_PFI9**	59	**IntTriggerA_DO_Start1**
10	**IntTriggerA_PFI10**	60	**IntTriggerA_DO_Gate**
11	**IntTriggerA_PFI11**	61	**IntTriggerA_DIO_ChangeDetect**

Value	Name	Value	Name
12	**IntTriggerA_PFI12**	62	**IntTriggerA_G0_SRC**
13	**IntTriggerA_PFI13**	63	**IntTriggerA_G0_GATE**
14	**IntTriggerA_PFI14**	64	**IntTriggerA_G0_OUT**
15	**IntTriggerA_PFI15**	65	**IntTriggerA_G0_SampleClk**
16	**IntTriggerA_RTSI0**	66	**IntTriggerA_G1_SRC**
17	**IntTriggerA_RTSI1**	67	**IntTriggerA_G1_GATE**
18	**IntTriggerA_RTSI2**	68	**IntTriggerA_G1_OUT**
19	**IntTriggerA_RTSI3**	69	**IntTriggerA_G1_SampleClk**
20	**IntTriggerA_RTSI4**	70	**IntTriggerA_G2_SRC**
21	**IntTriggerA_RTSI5**	71	**IntTriggerA_G2_GATE**
22	**IntTriggerA_RTSI6**	72	**IntTriggerA_G2_OUT**
23	**IntTriggerA_RTSI7**	73	**IntTriggerA_G2_SampleClk**
32	**IntTriggerA_PXI_StarTrig**	74	**IntTriggerA_G3_SRC**
33	**IntTriggerA_PXIe_DStarA**	75	**IntTriggerA_G3_GATE**
34	**IntTriggerA_PXIe_DStarB**	76	**IntTriggerA_G3_OUT**
35	**IntTriggerA_ReferenceClock**	77	**IntTriggerA_G3_SampleClk**
36	**IntTriggerA_FOUT**	78	**IntTriggerA_WatchdogExpiredPulse**
37	**IntTriggerA_ATrig**	103	**IntTriggerA_G0_HwArm**
38	**IntTriggerA_AI_CONVERT**	104	**IntTriggerA_G1_HwArm**
39	**IntTriggerA_AI_START**	105	**IntTriggerA_G2_HwArm**
40	**IntTriggerA_AI_START1**	106	**IntTriggerA_G3_HwArm**
41	**IntTriggerA_AI_START2**		

Trig_PFI_i_Pin_Dir_t

Value	Name
0	**PFI_Input**
1	**PFI_Output**

Trig_PFI_Output_Select_t

Value	Name	Value	Name
0	**PFI_WatchdogExpired**	31	**PFI_G2_SRC**
1	**PFI_AI_Start1**	32	**PFI_G2_Gate**
2	**PFI_AI_Start2**	33	**PFI_G2_Out**
3	**PFI_AI_Convert**	34	**PFI_G3_SRC**
4	**PFI_G1_SRC**	35	**PFI_G3_Gate**
5	**PFI_G1_Gate**	36	**PFI_G3_Out**
6	**PFI_AO_Update**	37	**PFI_DO_Start1**
7	**PFI_AO_Start1**	38	**PFI_DO_Gate**
8	**PFI_AI_Start**	39	**PFI_DI_Start1**
9	**PFI_G0_SRC**	40	**PFI_DI_Start2**
10	**PFI_G0_Gate**	41	**PFI_DI_Gate**
11	**PFI_SCXI_SpiClk**	42	**PFI_PXIe_DStarA**
12	**PFI_AI_ExternalMux_Clk**	43	**PFI_PXIe_DStarB**
13	**PFI_G0_Out**	44	**PFI_G0_SampleClk**
14	**PFI_G1_Out**	45	**PFI_G1_SampleClk**
15	**PFI_Freq_Out**	46	**PFI_G2_SampleClk**
16	**PFI_DO**	47	**PFI_G3_SampleClk**
17	**PFI_Atrig**	56	**PFI_AI_Gate**
18	**PFI_RTSI0**	57	**PFI_AO_Gate**
19	**PFI_RTSI1**	66	**PFI_IntTriggerA0**
20	**PFI_RTSI2**	67	**PFI_IntTriggerA1**
21	**PFI_RTSI3**	68	**PFI_IntTriggerA2**
22	**PFI_RTSI4**	69	**PFI_IntTriggerA3**
23	**PFI_RTSI5**	70	**PFI_IntTriggerA4**
24	**PFI_RTSI6**	71	**PFI_IntTriggerA5**
25	**PFI_RTSI7**	72	**PFI_IntTriggerA6**
26	**PFI_Star_Trig**	73	**PFI_IntTriggerA7**
27	**PFI_SCXI_Trig1_in**	74	**PFI_G0_HwArm**

Value	Name	Value	Name
28	**PFI_DIO_Change_Detect**	75	**PFI_G1_HwArm**
29	**PFI_DI_Convert**	76	**PFI_G2_HwArm**
30	**PFI_DO_Update**	77	**PFI_G3_HwArm**

Trig_PLL_Filter_Range_t

Value	Name
6	**PLL_100MHz**—100 MHz.

Trig_PLL_In_Source_Select_t

Value	Name	Value	Name
1	**RefClkSrc_PXIe_Clk100**	15	**RefClkSrc_PFI2**
2	**RefClkSrc_PXIe_DStarA**	16	**RefClkSrc_PFI3**
3	**RefClkSrc_PXIe_DStarB**	17	**RefClkSrc_PFI4**
4	**RefClkSrc_Star_Trigger**	18	**RefClkSrc_PFI5**
5	**RefClkSrc_RTSI0**	19	**RefClkSrc_PFI6**
6	**RefClkSrc_RTSI1**	20	**RefClkSrc_PFI7**
7	**RefClkSrc_RTSI2**	21	**RefClkSrc_PFI8**
8	**RefClkSrc_RTSI3**	22	**RefClkSrc_PFI9**
9	**RefClkSrc_RTSI4**	23	**RefClkSrc_PFI10**
10	**RefClkSrc_RTSI5**	24	**RefClkSrc_PFI11**
11	**RefClkSrc_RTSI6**	25	**RefClkSrc_PFI12**
12	**RefClkSrc_RTSI7**	26	**RefClkSrc_PFI13**
13	**RefClkSrc_PFI0**	27	**RefClkSrc_PFI14**
14	**RefClkSrc_PFI1**	28	**RefClkSrc_PFI15**

Trig_RTSI_i_Output_Select_t

Value	Name	Value	Name
0	**RTSI_AI_START1**	33	**RTSI_G3_SRC**
1	**RTSI_AI_START2**	34	**RTSI_G3_Z**
2	**RTSI_AI_CONVERT**	35	**RTSI_DI_Convert**
3	**RTSI_AO_UPDATE**	36	**RTSI_DI_Start1**
4	**RTSI_AO_START1**	37	**RTSI_DI_Start2**
5	**RTSI_G0_SRC**	38	**RTSI_DI_Gate**
6	**RTSI_G0_GATE**	39	**RTSI_DO_Update**
7	**RTSI_G0_OUT**	40	**RTSI_DO_Start1**
8	**RTSI_G0_Z**	41	**RTSI_DO_Gate**
9	**RTSI_AO_Gate**	42	**RTSI_PFI6**
10	**RTSI_AI_Gate**	43	**RTSI_PFI7**
11	**RTSI_FOUT**	44	**RTSI_PFI8**
12	**RTSI_RefClkOut**	45	**RTSI_PFI9**
13	**RTSI_DIO_ChangeDetect**	46	**RTSI_PFI10**
14	**RTSI_WatchdogExpiredPulse**	47	**RTSI_PFI11**
15	**RTSI_PFI0**	48	**RTSI_PFI12**
16	**RTSI_PFI1**	49	**RTSI_PFI13**
17	**RTSI_PFI2**	50	**RTSI_PFI14**
18	**RTSI_PFI3**	51	**RTSI_PFI15**
19	**RTSI_PFI4**	52	**RTSI_PXIe_DStarA**
20	**RTSI_PFI5**	53	**RTSI_PXIe_DStarB**
21	**RTSI_G1_OUT**	54	**RTSI_G0_SampleClk**
22	**RTSI_G1_GATE**	55	**RTSI_G1_SampleClk**
23	**RTSI_G1_SRC**	56	**RTSI_G2_SampleClk**
24	**RTSI_G1_Z**	57	**RTSI_G3_SampleClk**
25	**RTSI_Atrig**	72	**RTSI_IntTriggerA0**
26	**RTSI_AI_START**	73	**RTSI_IntTriggerA1**
27	**RTSI_G2_OUT**	74	**RTSI_IntTriggerA2**

Value	Name	Value	Name
28	**RTSI_G2_GATE**	75	**RTSI_IntTriggerA3**
29	**RTSI_G2_SRC**	76	**RTSI_IntTriggerA4**
30	**RTSI_G2_Z**	77	**RTSI_IntTriggerA5**
31	**RTSI_G3_OUT**	78	**RTSI_IntTriggerA6**
32	**RTSI_G3_GATE**	79	**RTSI_IntTriggerA7**

Trig_RTSI_i_Pin_Dir_t

Value	Name
0	**RTSI_Input**
1	**RTSI_Output**

Trig_StarTrig_Output_Select_t

Value	Name	Value	Name
8	**Star_IntTriggerA0**	12	**Star_IntTriggerA4**
9	**Star_IntTriggerA1**	13	**Star_IntTriggerA5**
10	**Star_IntTriggerA2**	14	**Star_IntTriggerA6**
11	**Star_IntTriggerA3**	15	**Star_IntTriggerA7**

Trig_TB3_Select_t

Value	Name
0	**TB3_From_OSC**
1	**TB3_From_PLL**

TrigPllLockedStatus_t

Value	Name
0	**PLL_NotLocked**
1	**PLL_Locked**

12

Analog Input Registers

AI base address = 0x20270

This chapter consists of *Analog Input Registers* and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

The analog input registers contain an InTimer object, offset: 0x202B0. Refer to Chapter 17, *InTimer Registers*, for more information.

List of Analog Input Registers

The analog input registers are as follows:

Offset 0x0: AI_Config_FIFO_Status_Register (R)

Offset 0x4: AI_Data_FIFO_Status_Register (R)

Offset 0x8: AI_FIFO_Data_Register (R)

Offset 0x8: AI_FIFO_Data_Register16 (R)

Offset 0x1E: AI_Config_FIFO_Data_Register (W)

Offset 0x28: AI_Data_Mode_Register (W)

Offset 0x2C: AI_Trigger_Select_Register (W)

Offset 0x30: AI_Trigger_Select_Register2 (W)

List of Enumerated Types

The enumerated types are as follows:

AI_Config_Bank_t

AI_Config_Channel_Type_t

AI_Disabled_Enabled_t

AI_External_Gate_Select_t

AI_FifoWidth_t

AI_Polarity_t

AI_START1_Select_t

AI_START2_Select_t

AI_StartConvertSelMux_t

Analog Input Registers

Offset 0x0: AI_Config_FIFO_Status_Register (R)

Absolute Address:
AI: 0x20270

Bits	Name
31..0	**AI_Config_FIFO_Status**—This bitfield represents the amount of data that remains in the AI configuration FIFO.

Offset 0x4: AI_Data_FIFO_Status_Register (R)

Absolute Address:
AI: 0x20274

Bits	Name
31..0	**AI_Data_FIFO_Status**—This bitfield represents the amount of data that remains in the AI data FIFO.

Offset 0x8: AI_FIFO_Data_Register (R)

Absolute Address:
AI: 0x20278

Bits	Name
31..0	**AI_FIFO_Data**—This bitfield reads from the AI data FIFO through the Programmed I/O. **Note**: This register should be read as 32 bits. This register should be used when the AI_FifoWidth is set for 4 Byte width. Note that the resolution could be any value.
Options: No Soft Copy	

Offset 0x8: AI_FIFO_Data_Register16 (R)

Absolute Address:
AI: 0x20278

Bits	Name
15..0	**AI_FIFO_Data1**—This bitfield reads from the AI data FIFO through the Programmed I/O. **Note**: This register should be read as 16 bits. This register should be used when the AI_FifoWidth is set for 2 Byte width. Note that the resolution should be 16 bits or less.
Options: No Soft Copy	

Offset 0x1E: AI_Config_FIFO_Data_Register (W)

Absolute Address:
AI: 0x2028E

Bits	Name
15	**Reserved**
14	**AI_Config_Last_Channel**—This bit should be set in the last entry of the scan sequence loaded into the channel configuration memory.
13	**AI_Config_Dither**—This bit enables dithering on the ADC, which improves accuracy. The values for this bitfield are in *AI_Disabled_Enabled_t*.
12	**Reserved**
11..9	**AI_Config_Gain**—These three bits control the gain settings of the input PGIA for the selected analog channel.
8..6	**AI_Config_Channel_Type**—These bits indicate which type of resource is active for the current entry in the scan list. The values for this bitfield are in *AI_Config_Channel_Type_t*.
5..4	**AI_Config_Bank**—These bits indicate which bank is active for the current entry in the scan list. The values for this bitfield are in *AI_Config_Bank_t*.
3..0	**AI_Config_Channel**—These bits indicate which channel is active for the current resource in the scan list.

Offset 0x28: AI_Data_Mode_Register (W)

Absolute Address:
AI: 0x20298

Bits	Name
31..9	**Reserved**
8	**AI_FifoWidth**—This bitfield allows for selection of the AI FIFO width, with a 0 indicating 2 Bytes wide, and a 1 indicating 4 Bytes wide. Any change to this bitfield must be followed by a reset of the AI FIFO. This bitfield usually remains constant and is based on the resolution of the ADC (2 Bytes for 12-, 14-, and 16-bits; 4 Bytes for 18 and up). The extended data is sign-extended for bipolar measurements, and it is zero-extended for unipolar measurements. The width selected corresponds to the expected read size of the direct FIFO data register in programmed I/O, as listed in *AI_FIFO_Data_Register* or *AI_FIFO_Data_Register16*. Programmed I/O reads to the FIFO of different size than the current FIFO size are *not* allowed (for example, a 32-bit register to a FIFO configured as 16 bits). The values for this bitfield are in *AI_FifoWidth_t*.
7	**AI_DoneNotificationEnable**—This bit enables the generation of the done notification on the input stream. This notification happens after a LastShiftIn event. After this notification happens, no more data can be written until a FIFO reset happens.
6..0	**Reserved**

Offset 0x2C: AI_Trigger_Select_Register (W)

Absolute Address:
AI: 0x2029C

Bits	Name
31	**AI_Convert_Source_Polarity**—This bit selects the active edge of the AI_CONVERT source signal. **Note**: The polarity of this bit changed from DAQ-STC2 to DAQ-STC3. This change was made to make it consistent with all other polarity bits. The values for this bitfield are in *AI_Polarity_t*.
30	**Reserved**
29..24	**AI_CONVERT_Source_Select**—Selects the AI_CONVERT source. When you set this bitfield to 0, the AI Interface is in internal AI_CONVERT mode. When you select any other signal as the AI_CONVERT source, the AI Timer is in external AI_CONVERT mode. The values for this bitfield are in *AI_StartConvertSelMux_t*.

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>23</td><td colspan="2">AI_External_Gate_Polarity—This bit selects the polarity of the external gate signal.

The values for this bitfield are in AI_Polarity_t.</td></tr>
<tr><td>22</td><td colspan="2">Reserved</td></tr>
<tr><td>21..16</td><td colspan="2">AI_External_Gate_Select—This bitfield enables and selects the external gate. You can use the external gate to pause an analog input operation in progress.

The values for this bitfield are in AI_External_Gate_Select_t.</td></tr>
<tr><td>15</td><td colspan="2">AI_START2_Polarity—This bit determines the polarity of AI_START2 trigger.

Note: Set this bit to 0 if AI_START2_Select is set to 0 (AI_START2_Pulse).

The values for this bitfield are in AI_Polarity_t.</td></tr>
<tr><td>14</td><td colspan="2">AI_START2_Edge—This bit enables edge detection of the AI_START2 trigger.

Note: Edge detection is required for almost all applications. It is also required when the source of the trigger is the software pulse.

<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled (level-sensitive trigger)</td></tr><tr><td>1</td><td>Enabled (edge-sensitive trigger)</td></tr></table></td></tr>
<tr><td>13..8</td><td colspan="2">AI_START2_Select—This bitfield selects the AI_START2 trigger.

The values for this bitfield are in AI_START2_Select_t.</td></tr>
<tr><td>7</td><td colspan="2">AI_START1_Polarity—This bit determines the polarity of the AI_START1 trigger.

Note: Set this bit to 0 if AI_START1_Select is set to 0 (AI_START1_Pulse).

The values for this bitfield are in AI_Polarity_t.</td></tr>
<tr><td>6</td><td colspan="2">AI_START1_Edge—This bit enables edge-sensitive detection of the AI_START1 trigger.

Note: Edge detection is required for almost all applications. Is is also required when the source of the trigger is the software pulse.

<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Disabled (level-sensitive trigger)</td></tr><tr><td>1</td><td>Enabled (edge-sensitive trigger)</td></tr></table></td></tr>
<tr><td>5..0</td><td colspan="2">AI_START1_Select—This bitfield selects the AI_START1 trigger.

The values for this bitfield are in AI_START1_Select_t.</td></tr>
</table>

Offset 0x30: AI_Trigger_Select_Register2 (W)

Absolute Address:
AI: 0x202A0

Bits	Name	
31..24	**Reserved**	
23	**AI_START_Polarity**—This bit determines the polarity of AI_START trigger. The values for this bitfield are in *AI_Polarity_t*.	
22	**AI_START_Edge**—This bit enables edge-sensitive detection of the AI_START trigger. **Note**: Always set this bit.	
	Value	**Meaning**
	0	**Disabled** (level-sensitive trigger)
	1	**Enabled** (edge-sensitive trigger)
21..16	**AI_START_Select**—This bit selects the AI_START trigger. When you set this bit to 0, the AI Timer is in internal AI_START mode. When you select any other signal as the AI_START trigger, the AI Timer is in external AI_START mode. The values for this bitfield are in *AI_StartConvertSelMux_t*.	
15..0	**Reserved**	

Enumerated Types

AI_Config_Bank_t

Value	Name
0	**Bank0**
1	**Bank1**

AI_Config_Channel_Type_t

Value	Name
0	**Loopback**
1	**Differential**
2	**NRSE**
3	**RSE**
5	**Internal**

AI_Disabled_Enabled_t

Value	Name
0	**Disabled**
1	**Enabled**

AI_External_Gate_Select_t

Value	Name	Value	Name
0	**Gate_Disabled**	25	**Gate_PFI14**
1	**Gate_PFI0**	26	**Gate_PFI15**
2	**Gate_PFI1**	27	**Gate_RTSI7**
3	**Gate_PFI2**	30	**Gate_Analog_Trigger**
4	**Gate_PFI3**	31	**Gate_Low**
5	**Gate_PFI4**	32	**Gate_G0_Out**
6	**Gate_PFI5**	33	**Gate_G1_Out**
7	**Gate_PFI6**	34	**Gate_G2_Out**

Value	Name	Value	Name
8	**Gate_PFI7**	35	**Gate_G3_Out**
9	**Gate_PFI8**	36	**Gate_G0_Gate**
10	**Gate_PFI9**	37	**Gate_G1_Gate**
11	**Gate_RTSI0**	38	**Gate_G2_Gate**
12	**Gate_RTSI1**	39	**Gate_G3_Gate**
13	**Gate_RTSI2**	40	**Gate_DI_Gate**
14	**Gate_RTSI3**	43	**Gate_AO_Gate**
15	**Gate_RTSI4**	44	**Gate_DO_Gate**
16	**Gate_RTSI5**	53	**Gate_IntTriggerA0**
17	**Gate_RTSI6**	54	**Gate_IntTriggerA1**
18	**Gate_PXIe_DStarA**	55	**Gate_IntTriggerA2**
19	**Gate_PXIe_DStarB**	56	**Gate_IntTriggerA3**
20	**Gate_Star_Trigger**	57	**Gate_IntTriggerA4**
21	**Gate_PFI10**	58	**Gate_IntTriggerA5**
22	**Gate_PFI11**	59	**Gate_IntTriggerA6**
23	**Gate_PFI12**	60	**Gate_IntTriggerA7**
24	**Gate_PFI13**		

AI_FifoWidth_t

Value	Name
0	**TwoByteFifo**
1	**FourByteFifo**

AI_Polarity_t

Value	Name
0	**Active_High_Or_Rising_Edge**
1	**Active_Low_Or_Falling_Edge**

AI_START1_Select_t

Value	Name	Value	Name
0	**Start1_SW_Pulse**	23	**Start1_PFI12**
1	**Start1_PFI0**	24	**Start1_PFI13**
2	**Start1_PFI1**	25	**Start1_PFI14**
3	**Start1_PFI2**	26	**Start1_PFI15**
4	**Start1_PFI3**	27	**Start1_RTSI7**
5	**Start1_PFI4**	28	**Start1_DIO_ChgDetect**
6	**Start1_PFI5**	30	**Start1_Analog_Trigger**
7	**Start1_PFI6**	31	**Start1_Low**
8	**Start1_PFI7**	36	**Start1_G0_Out**
9	**Start1_PFI8**	37	**Start1_G1_Out**
10	**Start1_PFI9**	38	**Start1_G2_Out**
11	**Start1_RTSI0**	39	**Start1_G3_Out**
12	**Start1_RTSI1**	40	**Start1_DI_Start1**
13	**Start1_RTSI2**	43	**Start1_AO_Start1**
14	**Start1_RTSI3**	44	**Start1_DO_Start1**
15	**Start1_RTSI4**	53	**Start1_IntTriggerA0**
16	**Start1_RTSI5**	54	**Start1_IntTriggerA1**
17	**Start1_RTSI6**	55	**Start1_IntTriggerA2**
18	**Start1_PXIe_DStarA**	56	**Start1_IntTriggerA3**
19	**Start1_PXIe_DStarB**	57	**Start1_IntTriggerA4**
20	**Start1_Star_Trigger**	58	**Start1_IntTriggerA5**
21	**Start1_PFI10**	59	**Start1_IntTriggerA6**
22	**Start1_PFI11**	60	**Start1_IntTriggerA7**

AI_START2_Select_t

Value	Name	Value	Name
0	**Start2_SW_Pulse**	23	**Start2_PFI12**
1	**Start2_PFI0**	24	**Start2_PFI13**
2	**Start2_PFI1**	25	**Start2_PFI14**
3	**Start2_PFI2**	26	**Start2_PFI15**
4	**Start2_PFI3**	27	**Start2_RTSI7**
5	**Start2_PFI4**	28	**Start2_DIO_ChgDetect**
6	**Start2_PFI5**	30	**Start2_Analog_Trigger**
7	**Start2_PFI6**	31	**Start2_Low**
8	**Start2_PFI7**	36	**Start2_G0_Out**
9	**Start2_PFI8**	37	**Start2_G1_Out**
10	**Start2_PFI9**	38	**Start2_G2_Out**
11	**Start2_RTSI0**	39	**Start2_G3_Out**
12	**Start2_RTSI1**	40	**Start2_DI_Start2**
13	**Start2_RTSI2**	43	**Start2_AO_Start1**
14	**Start2_RTSI3**	44	**Start2_DO_Start1**
15	**Start2_RTSI4**	53	**Start2_IntTriggerA0**
16	**Start2_RTSI5**	54	**Start2_IntTriggerA1**
17	**Start2_RTSI6**	55	**Start2_IntTriggerA2**
18	**Start2_PXIe_DStarA**	56	**Start2_IntTriggerA3**
19	**Start2_PXIe_DStarB**	57	**Start2_IntTriggerA4**
20	**Start2_Star_Trigger**	58	**Start2_IntTriggerA5**
21	**Start2_PFI10**	59	**Start2_IntTriggerA6**
22	**Start2_PFI11**	60	**Start2_IntTriggerA7**

AI_StartConvertSelMux_t

Value	Name	Value	Name
0	**StartCnv_InternalTiming**	26	**StartCnv_PFI15**
1	**StartCnv_PFI0**	27	**StartCnv_RTSI7**
2	**StartCnv_PFI1**	28	**StartCnv_G1_Out**
3	**StartCnv_PFI2**	29	**StartCnv_SCXI_Trig1**
4	**StartCnv_PFI3**	30	**StartCnv_Atrig**
5	**StartCnv_PFI4**	31	**StartCnv_Low**
6	**StartCnv_PFI5**	32	**StartCnv_PXIe_DStarA**
7	**StartCnv_PFI6**	33	**StartCnv_PXIe_DStarB**
8	**StartCnv_PFI7**	34	**StartCnv_G2_Out**
9	**StartCnv_PFI8**	35	**StartCnv_G3_Out**
10	**StartCnv_PFI9**	36	**StartCnv_G0_SampleClk**
11	**StartCnv_RTSI0**	37	**StartCnv_G1_SampleClk**
12	**StartCnv_RTSI1**	38	**StartCnv_G2_SampleClk**
13	**StartCnv_RTSI2**	39	**StartCnv_G3_SampleClk**
14	**StartCnv_RTSI3**	40	**StartCnv_DI_Convert**
15	**StartCnv_RTSI4**	43	**StartCnv_AO_Update**
16	**StartCnv_RTSI5**	44	**StartCnv_DO_Update**
17	**StartCnv_RTSI6**	53	**StartCnv_IntTriggerA0**
18	**StartCnv_DIO_ChgDetect**	54	**StartCnv_IntTriggerA1**
19	**StartCnv_G0_Out**	55	**StartCnv_IntTriggerA2**
20	**StartCnv_Star_Trigger**	56	**StartCnv_IntTriggerA3**
21	**StartCnv_PFI10**	57	**StartCnv_IntTriggerA4**
22	**StartCnv_PFI11**	58	**StartCnv_IntTriggerA5**
23	**StartCnv_PFI12**	59	**StartCnv_IntTriggerA6**
24	**StartCnv_PFI13**	60	**StartCnv_IntTriggerA7**
25	**StartCnv_PFI14**		

13

Counter Registers

G0 base address = 0x20300
G1 base address = 0x20340
G2 base address = 0x20380
G3 base address = 0x203C0

This chapter consists of *Counter Registers* and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

List of Counter Registers

The Counter registers are as follows:

List of Enumerated Types

The enumerated types are as follows:

Gi_A_Select_t

Gi_Armed_St_t

Gi_AuxCtrMode_t

Gi_B_Select_t

Gi_Bank_Switch_Enable_t

Gi_Bank_Switch_Mode_t

Gi_Counting_Once_t

Gi_CountingMode_t

Gi_Disabled_Enabled_t

Gi_Gate_Select_t

Gi_GatingMode_t

Gi_HW_Arm_Select_t

Gi_HwArmSyncMode_t

Gi_Index_Mode_t

Gi_Index_Phase_t

Gi_Load_Source_Select_t

Gi_Loading_On_Gate_t

Gi_Loading_On_TC_t

Gi_Output_Mode_t

Gi_Output_St_t

Gi_Polarity_t

Gi_Reload_Source_Switching_t

Gi_SampleClkSampleMode_t

Gi_SampleClockMode_t

Gi_SampleClockSelect_t

Gi_Second_Gate_Mode_t

Gi_Second_Gate_Select_t

Gi_Source_Select_t

Gi_Stop_Mode_t

Gi_Trigger_Mode_For_Edge_Gate_t

Gi_Up_Down_t

Gi_Z_Select_t

Counter Registers

Offset 0x0: Gi_Command_Register (W)

Absolute Addresses:
G0: 0x20300
G1: 0x20340
G2: 0x20380
G3: 0x203C0

Bits	Name
15	**Gi_Disarm_Paired_Counter** (Strobe)—Setting this bit to 1 disarms this counter's pair. For example, setting Gi_Disarm_Paired_Counter of counter 0 disarms counter 1, and setting Gi_Disarm_Paired_Counter of counter 1 disarms counter 0. This bit is cleared automatically.
14	**Gi_Reset** (Strobe)—Setting this bit to 1 resets the counter, clears Gi_Arm and Gi_Arm_Copy, clears the *Gi_Mode_Register*, and clears the appropriate bits of the *Gi_Input_Select_Register*. This bit is cleared automatically.
13	**Gi_Arm_Paired_Counter** (Strobe)—Setting this bit to 1 arms this counter's pair. For example, setting Gi_Arm_Paired_Counter of counter 1 arms counter 0, and setting Gi_Arm_Paired_Counter of counter 0 arms counter 1. The paired counter remains armed (and the bit remains set) until it is disarmed, either by hardware or by setting its Gi_Disarm to 1.
12..10	**Reserved**
9	**Gi_Bank_Switch_Start** (Strobe)—Setting this bit to 1 indicates load register bank switching on the condition selected by Gi_Bank_Switch_Mode. You can use this bit in an interrupt service program. This bit is cleared automatically.
8	**Reserved**
7	**Gi_WrLoadRegsFromFifo** (Strobe)—This bit forces a write to the selected bank of load registers from the FIFO. This bit only works when the Output FIFO is enabled and the counter is not armed. Use this bit to Write Bank X and Y from the FIFO before arming. The counter still needs to be preloaded. The Gi_ForcedWrFromFifoInProgSt bit sets when this command is received and clears once the operation is completed.
6..5	**Reserved**

Bits	Name
4	**Gi_Disarm** (Strobe)—Setting this bit to 1 disarms the general-purpose counter. This bit is cleared automatically. The counter has three basic modes of operation: • Src=TB=TB3. In this mode the TB does not change from not armed to armed. The Disarm process should happen cleanly and does not require any further software intervention. • Src != TB, TB=TB3. (Gi_ForceSourceEqualToTimebase = 0) This mode emulates an external source. In this mode the TB is also TB3 both before and after arm, so again, disarm should happen cleanly and does not require any further software intervention. • Src=TB != TB3. (Gi_ForceSourceEqualToTimebase = 1) In this mode, you can replace the timebase of the counter by using the clean clk mux. In this mode, you have no way to know if the disarm can happen cleanly or not because the source of the clock may not be there anymore. For this case, the disarm should happen cleanly if the source is there. The disarm may be slow or not happen if the timebase is too slow or not present. In that case, software must reset the counter if the disarm process times out. This forces a disarm and forces the TB3 clock to become the timebase again.
3	**Reserved**
2	**Gi_Load** (Strobe)—Setting this bit to 1 loads the contents of the selected load register into the general-purpose counter. This bit should only be set when the counter is disarmed. It will be ignored otherwise. This bit is cleared automatically.
1	**Reserved**
0	**Gi_Arm** (Strobe)—Setting this bit to 1 arms the general-purpose counter. The counter remains armed (and Gi_Armed_St remains set) until it is disarmed, either by hardware or by setting Gi_Disarm to 1.

Offset 0x2: Gi_Mode_Register (W)

Absolute Addresses:
G0: 0x20302
G1: 0x20342
G2: 0x20382
G3: 0x203C2

Bits	Name
15	**Gi_Reload_Source_Switching**—If Gi_Gate_Select_Load_Source is set to 0, this bit enables load register selection. The values for this bitfield are in *Gi_Reload_Source_Switching_t*.
14	**Gi_Loading_On_Gate**—This bit determines whether the gate signal causes counter reload: • Gate signal does not cause counter reload. • Counter is reloaded on gate edge that stops the counter, unless edge gating is used and Gi_Trigger_Mode_For_Edge_Gate is set to GateLoads. If Gi_Trigger_Mode_For_Edge_Gate is set to GateLoads, the counter is reloaded on every Selected Gate edge. **Note**: Reloading occurs on active source edge. The values for this bitfield are in *Gi_Loading_On_Gate_t*.
13	**Gi_ForceSourceEqualToTimebase**—When this bit is cleared, the counter operates with the TB3 signal (normally 100 MHz) as the main timebase of the counter. In the case of an external source, the counter samples the external source with the timebase, and then re-syncs the outputs. This works fine for frequencies up to the timebase frequency divided by 4 (25 MHz in the case of 100 MHz timebase). If a higher source signal is needed, the counter can be set to use the external signal as the timebase. This is done by setting this bit. The counter assumes its timebase is a free-running signal. **Note**: When this bit is set, the counter relies on the user-provided source to operate. If that source is very slow or not present, the counter may not respond to normal operations (such as disarm). Refer to the Disarm bit in the *Gi_Command_Register* for more details on how to recover from this situation.
12	**Gi_Loading_On_TC**—This bit determines the counter behavior on TC. The values for this bitfield are in *Gi_Loading_On_TC_t*.
11..10	**Gi_Counting_Once**—This bit determines whether the hardware disarms the counter when the counter stops because of a hardware condition. The values for this bitfield are in *Gi_Counting_Once_t*.
9..8	**Gi_Output_Mode**—This bit selects the mode for the G_OUT signal. **Note**: The Toggle modes (on TC or Gate) are not reset when disarmed. This allows the counter to keep the value of the output after the operation is complete. The logic involved in generating the toggle values is reset when the bitfield is set to TC. Therefore, when the DAQ-STC3 is being configured to use one of the toggle bits, the Gi_Output_Mode bitfield should be set to TC, then the DAQ-STC3 should be configured, and finally the Gi_Output_Mode field set to the toggle mode of choice. This prevents glitches on the output and guarantees that the output will start at the right value. The values for this bitfield are in *Gi_Output_Mode_t*.

Bits	Name
7	**Gi_Load_Source_Select**—If the general-purpose counter is disarmed, this bit selects the initial counter load register: • Load register A • Load register B **Note**: The source for subsequent loads depends on the Gi_Reload_Source_Switching bitfield in the *Gi_Mode_Register*. If the general-purpose counter is armed, writing to this bit has no effect. The values for this bitfield are in *Gi_Load_Source_Select_t*.
6..5	**Gi_Stop_Mode**—This bit selects the condition on which the counter stops. Notice that regardless of this bitfield setting, you can always use the software disarm command, Gi_Disarm, to stop the counter. The gate condition that stops the counter is determined by Gi_Gating_Mode (in case of level gating) or by a combination of Gi_Gating_Mode and Gi_Trigger_Mode_For_Edge_Gate (in case of edge gating). Selections StopAtGateOrFirstTC and StopAtGateOrSecondTC are valid only if Gi_Trigger_Mode_For_Edge_Gate is set to GateEdgeStarts (no hardware limitation, therefore if the selections are invalid, the acquisition is invalid). The values for this bitfield are in *Gi_Stop_Mode_t*.
4..3	**Gi_Trigger_Mode_For_Edge_Gate**—This bit selects the triggering mode, if gating is not disabled. Selections FirstGateStartSecondGateStops and FirstGateStopsSecondGateStarts are valid only if Gi_Stop_Mode is set to StopOnGateCondition (no hardware limit on this). Selections FirstGateStartSecondGateStops, FirstGateStopsSecondGateStarts, and GateEdgeStarts are valid only if Gi_Gating_Mode is set to AssertingEdgeGating or DeassertingEdgeGating. Selection GateLoads is valid only if Gi_Gating_Mode is not set to GateDisabled. The values for this bitfield are in *Gi_Trigger_Mode_For_Edge_Gate_t*.
2	**Gi_Gate_On_Both_Edges**—This bit enables you to use both gate edges to generate the gate interrupt and/or to control counter operation. **Note**: This bit also affects where interrupts are generated. The values for this bitfield are in *Gi_Disabled_Enabled_t*.
1..0	**Gi_Gating_Mode**—This bit enables and selects the counter gating mode. When Gi_Gating_Mode is GateDisabled, gate level is available only for control of counting direction (up/down), and for no other purpose. The values for this bitfield are in *Gi_GatingMode_t*.

Offset 0x4: Gi_HW_Save_Register (R)

Absolute Addresses:
G0: 0x20304
G1: 0x20344
G2: 0x20384
G3: 0x203C4

Bits	Name
31..0	**Gi_HW_Save_Value**—This register contains the last value of the counter saved by the gate or sample clock based on their configuration. This register should not be necessary because the counters have a FIFO for input data.

Offset 0x4: Gi_Load_A_Register (W)

Absolute Addresses:
G0: 0x20304
G1: 0x20344
G2: 0x20384
G3: 0x203C4

Bits	Name
31..0	**Gi_Load_A**—This bitfield is the load value A for the general-purpose counter. If load register A is the selected load register, the counter loads the value contained in this bitfield on Gi_Load, on the counter TC, and on the G_GATE induced counter reload condition (if G_GATE reloading is enabled).

Offset 0x8: Gi_Save_Register (R)

Absolute Addresses:
G0: 0x20308
G1: 0x20348
G2: 0x20388
G3: 0x203C8

Bits	Name
31..0	**Gi_Save_Value**—This register allows a safe read of the current value of the counter.

Offset 0x8: Gi_Load_B_Register (W)

Absolute Addresses:
G0: 0x20308
G1: 0x20348
G2: 0x20388
G3: 0x203C8

Bits	Name
31..0	**Gi_Load_B**—This bitfield is the load value B for the general-purpose counter. If load register B is the selected load register, the counter loads the value contained in this bitfield on Gi_Load, on the counter TC, and on the G_GATE induced counter reload condition (if G_GATE reloading is enabled).

Offset 0xC: Gi_Status_Register (R)

Absolute Addresses:
G0: 0x2030C
G1: 0x2034C
G2: 0x2038C
G3: 0x203CC

Bits	Name
31	**Reserved**
30	**Gi_AuxTC_EventSt**—Aux Counter TC event status. This event can be useful for finite dynamic pulse train generation, or to generate interrupts every number of pulse train generation waveform switches.
29	**Gi_AuxTC_ErrorEventSt**—Aux Counter TC Error event status. This event occurs when an Aux TC event happens before the previous one has been acknowledged.
28	**Gi_WritesTooFastErrorSt**—A “Writes too fast error” happens when software writes twice to the same bank before a switch event occurs. The event can be the switching of the banks by the counter (when Gi_WriteOnSwitchRequest = 0) or the scheduling of the switch, as in a gate event (when Gi_WriteOnSwitchRequest = 1). It is possible to get an interrupt on this error. Clear the error and the status by writing 1 to Gi_WritesTooFastErrorAck.
27	**Gi_ForcedWrFromFifoInProgSt**—This bit indicates that a set of writes to the load registers initiated by writing to Gi_WrLoadRegsFromFifo is in progress. It is guaranteed by hardware that if this bit is 0 after writing the Gi_WrLoadRegsFromFifo bit, the hardware is done writing the load registers.
26	**Gi_DisarmEventInterruptSt**—This bit indicates a Disarm Event Interrupt. This interrupt, when enabled, asserts when a Disarm Event is detected. Clear this bit using the Gi_Disarm_Interrupt_Ack bit.

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>25</td><td colspan="2">Gi_GateSwitchError_St—This bit indicates whether a gate switch error occurred in an output generation, for example, a new gate requests the counter to switch banks when the counter still has a switch pending from the previous gate. This reflects that the frequency of the output is less than the frequency of the gate. This bit clears with the Gi_GateSwitchError_Ack command bitfield and can generate an interrupt when enabled by Gi_GateSwitchErrorInt_En mode bit.</td></tr>
<tr><td>24</td><td colspan="2">Gi_TC_St—This bit indicates whether the general-purpose counter has reached TC. This bit is cleared by setting Gi_TC_Interrupt_Ack to 1.</td></tr>
<tr><td rowspan="4">23</td><td colspan="2">Gi_Gate_Interrupt_St—This bit indicates whether a gate interrupt has occurred.
Note: This bit can be cleared by setting Gi_Gate_Interrupt_Ack to 1.</td></tr>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No interrupt</td></tr>
<tr><td>1</td><td>Interrupt request generated</td></tr>
<tr><td>22..21</td><td colspan="2">Reserved</td></tr>
<tr><td rowspan="4">20</td><td colspan="2">Gi_Gate_St—This bit indicates status of the general-purpose counter gate.
Note: This bit can only be used in the level-gating mode.
Note: The state of this bit reflects the gate value after polarity compensation.</td></tr>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Inactive gate</td></tr>
<tr><td>1</td><td>Active gate</td></tr>
<tr><td>19</td><td colspan="2">Reserved</td></tr>
<tr><td rowspan="3">18</td><td colspan="2">Gi_Bank_St—This bit indicates the load register bank used by the counter.</td></tr>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Bank X</td></tr>
<tr><td>1</td><td>Bank Y</td></tr>
<tr><td rowspan="3">17</td><td colspan="2">Gi_SampleClockInterruptSt—This bit indicates a Sample Clock Interrupt. This interrupt, when enabled, asserts with a valid sample clock that will result in writing new data to the FIFO: either a sample clock with no error, or a sample clock overrun when not in stop on error mode. Clear this bit using the Gi_SampleClockInterruptAck bit.</td></tr>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>No Error</td></tr>
<tr><td>1</td><td>A Sample Clock Interrupt error has occurred</td></tr>
</table>

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>16</td><td colspan="2">Gi_SampleClockOverrun_St—This bit indicates a Sample Clock overrun error. Clear this bit using the Gi_SampleClockOverrunErrorAck bit.

Note: If the Gi_SampleClockStopOnError bit is set, the counter stops acquiring data when the error occurs.</td></tr>
<tr><td></td><td>Value</td><td>Meaning</td></tr>
<tr><td></td><td>0</td><td>No Error</td></tr>
<tr><td></td><td>1</td><td>A Sample Clock Overrun error has occurred</td></tr>
<tr><td>15</td><td colspan="2">Gi_DRQ_St—DMA Request Status. This bit is set when the counter requires DMA service and clears automatically when the request is serviced.</td></tr>
<tr><td>14</td><td colspan="2">Gi_Gate_Error_St—This bit indicates the detection of a counter gate acknowledge latency error.

Note: Gi_Gate_Error_St is set when a second gate interrupt occurs before the first gate interrupt is acknowledged. To clear this bit, set Gi_Gate_Error_Confirm.</td></tr>
<tr><td>13</td><td colspan="2">Reserved</td></tr>
<tr><td>12</td><td colspan="2">Gi_TC_Error_St—This bit indicates the detection of a TC latency error.

Note: A TC latency error is detected when Gi_TC_Interrupt_Ack is not set between two counter TCs. This allows you to detect large interrupt latencies and potential problems associated with them. To clear this bit, set Gi_TC_Error_Confirm to 1.</td></tr>
<tr><td>11</td><td colspan="2">Gi_DRQ_Error—DMA Request Error. This bit is set when a DMA overflow error occurs on a read operation or when an underflow error occurs on a write operation. When reading, this bit sets if a save request occurs when the FIFO is full and another DMA read follows. The save request does not corrupt the data when the FIFO is received after completion.

When writing, an error is set when the bank switches to a new bank before the registers on that bank have been written. This is usually a FIFO underflow error.</td></tr>
<tr><td>10</td><td colspan="2">Gi_No_Load_Between_Gates_St—This bit indicates that a counter reload did not occur between two relevant G_GATE edges.</td></tr>
<tr><td>9</td><td colspan="2">Reserved</td></tr>
<tr><td>8</td><td colspan="2">Gi_Armed_St—This bit indicates whether the counter is armed. Software must poll on this bit after issuing arm/disarm commands until the command takes effect.

The values for this bitfield are in Gi_Armed_St_t.</td></tr>
<tr><td>7..5</td><td colspan="2">Reserved</td></tr>
<tr><td>4</td><td colspan="2">Gi_Next_Load_Source_St—This bit indicates the next load source of the counter.</td></tr>
<tr><td></td><td>Value</td><td>Meaning</td></tr>
<tr><td></td><td>0</td><td>Load register A</td></tr>
<tr><td></td><td>1</td><td>Load register B</td></tr>
<tr><td>3</td><td colspan="2">Reserved</td></tr>
</table>

Bits	Name
2	**Gi_Counting_St**—If the counter is armed, this bit indicates whether the counter is counting. **Note**: If the counter is not armed, this bit should be ignored.
1	**Reserved**
0	**Gi_Output_St**—This bit indicates the current G_OUT state for the counter (after the polarity selection). The values for this bitfield are in *Gi_Output_St_t*.

Offset 0xC: Gi_Input_Select_Register (W)

Absolute Addresses:
G0: 0x2030C
G1: 0x2034C
G2: 0x2038C
G3: 0x203CC

Bits	Name	
15	**Gi_Source_Polarity**—This bit selects the active edge of the general-purpose counter source.	
	Value	**Meaning**
	0	**Rising edge**
	1	**Falling edge**
14	**Gi_Output_Polarity**—This bit selects the polarity of the G_OUT pulse (in the TC mode) or the initial G_OUT level (in the toggle output mode).	
	Value	**Meaning**
	0	**Active high pulse or initial low level**
	1	**Active low pulse or initial high level**
13	**Gi_Gate_Select_Load_Source**—This bit enables the selection of the load register bank by the counter gate. **Note**: When this bit is set to 1, an active gate level selects load register bank X, and an inactive gate level selects load register bank Y. Also, Gi_Reload_Source_Switching is ignored. This feature can be used only in conjunction with level gating. The values for this bitfield are in *Gi_Disabled_Enabled_t*.	
12..7	**Gi_Gate_Select**—This bitfield selects the G_GATE source for the general- purpose counter. The values for this bitfield are in *Gi_Gate_Select_t*.	

Bits	Name
6..1	**Gi_Source_Select**—This bitfield selects the general-purpose counter source. The values for this bitfield are in *Gi_Source_Select_t*.
0	**Gi_Gate_Polarity**—This bit selects the polarity of the G_GATE input signal. The values for this bitfield are in *Gi_Polarity_t*.

Offset 0xE: Gi_Autoincrement_Register (W)

Absolute Addresses:
G0: 0x2030E
G1: 0x2034E
G2: 0x2038E
G3: 0x203CE

Bits	Name
15..8	**Reserved**
7..0	**Gi_Auotincrement**—This 8-bit register holds a fixed value that is added to the contents of load register A after each counter reload, so that on the next reload the counter loads the incremented value. You should use the autoincrement feature in pulse-train generation for ETS to automatically increase the pulse delay after each trigger.

Offset 0x10: Gi_FifoStatusRegister (R)

Absolute Addresses:
G0: 0x20310
G1: 0x20350
G2: 0x20390
G3: 0x203D0

Bits	Name
31..0	**Gi_FifoStatus**—This bitfield indicates the FIFO full status of the input FIFO in units of samples.

Offset 0x10: Gi_Counting_Mode_Register (W)

Absolute Addresses:
G0: 0x20310
G1: 0x20350
G2: 0x20390
G3: 0x203D0

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>15</td><td colspan="2">Gi_Prescale_Div_2—When prescaling is enabled, set this bit to change the prescale value from 8 to 2.</td></tr>
<tr><td></td><td>Value</td><td>Meaning</td></tr>
<tr><td></td><td>0</td><td>Division Ratio</td></tr>
<tr><td></td><td>1</td><td>Division Ratio</td></tr>
<tr><td>14</td><td colspan="2">Gi_Prescale—When this bit is enabled, a high-speed counter divides the selected source by eight (or two) before clocking the counter. In prescale mode, you can measure signal frequencies that exceed the 100 MHz limit of the counters. The division ratio is selectable using the Gi_Prescale_Div_2 bitfield.</td></tr>
<tr><td>13..8</td><td colspan="2">Gi_HW_Arm_Select—This bitfield selects the source for the HW arm signal.

The values for this bitfield are in Gi_HW_Arm_Select_t.</td></tr>
<tr><td>7</td><td colspan="2">Gi_HW_Arm_Enable—Setting this bit enables the counter to be armed by a signal from the source specified by Gi_HW_Arm_Select. After configuring the HW_Arm settings, the Gi_Arm bit should be set.</td></tr>
<tr><td>6..5</td><td colspan="2">Gi_Index_Phase—This bitfield determines the state of the A and B quadrature signals, and is where the index (or Z signal) is acted upon. The Z index can span multiple phases of the quadrature, but you should reload the counter in only one of the phases.

The values for this bitfield are in Gi_Index_Phase_t.</td></tr>
<tr><td>4</td><td colspan="2">Gi_Index_Mode—The effect of setting this bit depends on the counting mode. This bit can be written at run time when the counter is programmed for Quadrature or Two-Pulse mode to enable or disable Z index loading. For any other mode, it should be set before arming and remain constant.

The values for this bitfield are in Gi_Index_Mode_t.</td></tr>
<tr><td>3</td><td colspan="2">Gi_HW_Arm_Polarity—This bit sets the polarity of the HW Arm signal. Writing a 1 to this bit sets the polarity to active low. 0 means active high.

The values for this bitfield are in Gi_Polarity_t.</td></tr>
<tr><td>2..0</td><td colspan="2">Gi_Counting_Mode—This field determines the counting mode to use to interface to different encoders and applications.

The values for this bitfield are in Gi_CountingMode_t.</td></tr>
</table>

Offset 0x12: Gi_Second_Gate_Register (W)

Absolute Addresses:
G0: 0x20312
G1: 0x20352
G2: 0x20392
G3: 0x203D2

Bits	Name
15..14	**Reserved**
13	**Gi_Second_Gate_Polarity**—Setting this bit inverts the selected second gate, changing the polarity. The values for this bitfield are in *Gi_Polarity_t*.
12..7	**Gi_Second_Gate_Select**—This field selects the signal used as the second gate for the counter. The values for this bitfield are in *Gi_Second_Gate_Select_t*.
6..1	**Reserved**
0	**Gi_Second_Gate_Mode**—The second gate feature allows one signal to start the counter and another signal to stop the counter for two-edge separation measurements or for pulse-width measurements. When 0, the second gate is disabled. When 1, The actual gate signal is modified with a combination of the first and second gate. An assertion of the second gate signals asserts the counter gate, and an assertion of the gate signal deasserts the gate. You can use this new gate signal for start and stop operations. Gi_Second_Gate_Mode is also useful in pulse-width measurements. The level gating mode starts counting when the counter is armed while the gate is high, so the first measurement may be incorrect (too short). Using the selected gate input to the second gate and inverting both gate and second gate polarity forces a rising edge to occur before the gate asserts. As a result, the first pulse is measured. Gi_Second_Gate_Mode is also used for two-edge separation measurements. The values for this bitfield are in *Gi_Second_Gate_Mode_t*.

Offset 0x14: Gi_SampleClockCountRegister (R)

Absolute Addresses:
G0: 0x20314
G1: 0x20354
G2: 0x20394
G3: 0x203D4

Bits	Name
31..24	**Reserved**
23..0	**Gi_SampleClockCount**—This bitfield reflects the value of the sample clock count. This count resets to 0 when arming the counter and counts down, so the value after the first sample clock is 0xFFFFFF. Note that when doing a software disarm, the value read remains as the last valid value registered until the counter is armed again reflecting the reset value of 0.

Offset 0x14: Gi_DMA_Config_Register (W)

Absolute Addresses:
G0: 0x20314
G1: 0x20354
G2: 0x20394
G3: 0x203D4

Bits	Name
15	**Reserved**
14	**Gi_DoneNotificationEnable**—This bit enables the generation of the done notification on the input stream. This notification occurs after the last sample after an SC_TC event is written to the FIFO. After this notification occurs, no more data can be written until a FIFO reset occurs. **Note**: This feature only works then the counter is paired with a timing engine for generation of the sample clock.
13	**Gi_WrFifoEnable**—This bit enables the FIFO for output operations.You must enable this bit for DMA operation. When cleared, the data for the load registers can be written directly to the load registers or through the *Gi_AutomaticLoadRegister*, which directs the write to the correct load register.
12	**Reserved**

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>11..10</td><td colspan="2">Gi_WaitForFirstEventOnGate—This bit forces the hardware to wait for a first event on the gate before starting to save data in the FIFO. This can be used to prevent the “bad first point” problem. Depending on how the operation is set up, waiting for a RE or FE of the gate ensures that the first point returned is the first possible valid measurement. For example, for pulse-width measurement that saves data on the FE of the gate, setting this bit to wait for RE ensures that the first measurement is valid.</td></tr>
<tr><td></td><td>Value</td><td>Meaning</td></tr>
<tr><td></td><td>0</td><td>Don’t Wait</td></tr>
<tr><td></td><td>1</td><td>Wait for a RE on the gate</td></tr>
<tr><td></td><td>2</td><td>Wait for an FE on the gate</td></tr>
<tr><td>9..4</td><td colspan="2">Reserved</td></tr>
<tr><td>3</td><td colspan="2">Gi_DMA_Reset (Strobe)—Direct Memory Access Reset. Setting this bit resets the pointer used to track. This bit should be set as a part of initializing DMA. It is cleared automatically. Setting this bit also clears the active FIFO (as determined by DMA_Write). After setting this bit, a read or bus flush should follow to ensure that the reset is complete before any direct or DMA writes to the output FIFO occur (not necessary to flush when using TIO FIFO as input).</td></tr>
<tr><td>2</td><td colspan="2">Reserved</td></tr>
<tr><td>1</td><td colspan="2">Gi_DMA_Write—Direct Memory Access Write. This bit indicates the direction of the DMA operation. The counter FIFO must be reset with Gi_DMA_Reset when the value of this field changes.</td></tr>
<tr><td></td><td>Value</td><td>Meaning</td></tr>
<tr><td></td><td>0</td><td>DMA controller reads data from the save registers</td></tr>
<tr><td></td><td>1</td><td>DMA controller writes data to the load registers</td></tr>
<tr><td>0</td><td colspan="2">Gi_DMA_Enable—Direct Memory Access Enable. When this bit is set, an additional DMA mode is added for streaming counts into or out of the counters.
• The read mode uses the counter FIFO to increase the rate at which pulse or period measurements are made. If a DMA channel (stream) is assigned to the counter, the data transfer occurs automatically. If the data transfer occurs through PIO, then the FIFO can be read from the Gi_RdFifoRegister.
• For pulse generation, the DMA can reload the load register when a bank switch occurs. DRQ asserts on the bank switch and clears when the last load register (register B) is written. When using PIO, the data is written to the Gi_AutomaticLoadRegister, which automatically writes register A and then register B as it receives data. When using the FIFO (Gi_WrFifoRegister), the counter loads the data automatically from the FIFO. Data may be added to the FIFO either by PIO (write to Gi_WrFifoRegister), or through DMA when a DMA channel (stream) is assigned to the counter.</td></tr>
<tr><td colspan="3">Options: Initial Value = 0x0000</td></tr>
</table>

Offset 0x18: Gi_RdFifoRegister (R)

Absolute Addresses:
G0: 0x20318
G1: 0x20358
G2: 0x20398
G3: 0x203D8

Bits	Name
31..10	**Gi_RdFifoData**—This bitfield reads from the Counter data FIFO through programmed I/O. To enable the FIFO, Gi_DMA_Enable must be set, and Gi_DMA_Write must be cleared (set to not write).

Offset 0x18: Gi_WrFifoRegister (W)

Absolute Addresses:
G0: 0x20318
G1: 0x20358
G2: 0x20398
G3: 0x203D8

Bits	Name
31..0	**Gi_WrFifoData**—This bitfield writes to the A and B load registers of the appropriate load bank in a FIFO fashion through programmed I/O. To enable this register, Gi_DMA_Enable and Gi_DMA_Write must be set.
Options: No Soft Copy	

Offset 0x1C: Gi_SampleClockRegister (W)

Absolute Addresses:
G0: 0x2031C
G1: 0x2035C
G2: 0x2039C
G3: 0x203DC

Bits	Name	
15	**Gi_SampleClockGateIndependent**—When this bit is set, the sample clock saves data independently of the gate. This is useful for event counting or position measurements. This also enables the dual counting mode, in which the main counter counts source edges and the aux counter counts gate edges.	
14..13	**Reserved**	
12	**Gi_SampleClockSampleMode**—This bitfield selects when the sample clock saves data. The options trade off latency for accurate chronology between the sample clock and the sample returned. The values for this bitfield are in *Gi_SampleClockMode_t*.	
11	**Gi_SampleClockPulse**—When set, this bit generates a pulse on the sample clock signal. The SampleClockSelect field must be set to the pulse bit enumeration to work.	
10..8	**Gi_SampleClockMode**—Selects the mode of operation for the sample clock. The values for this bitfield are in *Gi_SampleClockMode_t*.	
7	**Gi_SampleClockLevelMode**—This bit enables level mode for the sample clock. The counter saves measurements while the sample clock is active.	
	Value	**Meaning**
	0	**Level Mode is Disabled.** Sample clock operates in Edge mode.
	1	**Level Mode is Enabled.**
6	**Gi_SampleClockPolarity**—This bit selects the polarity for the Sample clock signal.	
	Value	**Meaning**
	0	**Works off rising edge of incoming signal.**
	1	**Works off falling edge of incoming signal.**
5..0	**Gi_SampleClockSelect**—This bitfield selects the sample clock signal for general-purpose counter. The values for this bitfield are in *Gi_SampleClockSelect_t*.	

Offset 0x1E: Gi_AuxCtrRegister (W)

Absolute Addresses:
G0: 0x2031E
G1: 0x2035E
G2: 0x2039E
G3: 0x203DE

Bits	Name
15..9	**Reserved**
8	**Gi_AuxCtrLoad**—Setting this bit to 1 loads the contents of the load register A into the Aux counter. This bit is cleared automatically.
7..3	**Reserved**
2..0	**Gi_AuxCtrMode**—This bitfield selects the operation mode of the Aux counter. Based on this mode, the Aux counter automatically performs the appropriate routing and settings for the operation. The values for this bitfield are in *Gi_AuxCtrMode_t*.

Offset 0x20: Gi_AuxCtrLoadA_Register (W)

Absolute Addresses:
G0: 0x20320
G1: 0x20360
G2: 0x203A0
G3: 0x203E0

Bits	Name
31..0	**Gi_AuxCtrLoadA**—Value for loadA register of the Aux Counter.

Offset 0x24: Gi_AuxCtrLoadB_Register (W)

Absolute Addresses:
G0: 0x20324
G1: 0x20364
G2: 0x203A4
G3: 0x203E4

Bits	Name
31..0	**Gi_AuxCtrLoadB**—Value for loadB register for the Aux Counter.

Offset 0x28: Gi_AutomaticLoadRegister (W)

Absolute Addresses:
G0: 0x20328
G1: 0x20368
G2: 0x203A8
G3: 0x203E8

Bits	Name
31..0	**Gi_AutoLdRegister**—This field writes to the appropriate load register, determined by internal pointers, when DMA mode is enabled and the Write mode bit is set. It is similar to the write switch functionality on the original DAQ-STC.
Options: No Soft Copy	

Offset 0x2C: Gi_Interrupt1_Register (W)

Absolute Addresses:
G0: 0x2032C
G1: 0x2036C
G2: 0x203AC
G3: 0x203EC

Bits	Name
31	**Gi_Gate_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears Gi_Gate_Interrupt_St and acknowledges the gate interrupt request for the counter if the gate interrupt is enabled. This bit is cleared automatically. **Note**: Do not use gate interrupts on the DAQ-STC3 for interrupt-driven data transfer. Use the DMA interrupts instead.
30	**Gi_TC_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears Gi_TC_St and acknowledges the TC interrupt request for the counter if the TC interrupt is enabled. This bit is cleared automatically.
29	**Gi_DMA_Error_Ack** (Strobe)—Setting this bit to 1 clears Gi_DRQ_Error. This bit is cleared automatically.
28	**Gi_GateSwitchError_Ack** (Strobe)—Setting this bit to 1 clears Gi_GateSwitchError_St. This bit is cleared automatically.
27	**Gi_WritesTooFastErrorAck** (Strobe)—Setting this bit to 1 clears Gi_WritesTooFastErrSt. This bit is cleared automatically.
26	**Gi_Disarm_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears Gi_Disarm_Interrupt_St and acknowledges the interrupt. This bit is cleared automatically.
25	**Gi_Aux_Ctr_TC_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears Gi_Aux_Ctr_TC_Interrupt_St and acknowledges the interrupt. This bit is cleared automatically.

Bits	Name
24	**Gi_Aux_Ctr_TC_Error_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears Aux Ctr TC Error Interrupt_St and acknowledges the interrupt. This bit is cleared automatically.
23	**Reserved**
22	**Gi_TC_Error_Confirm** (Strobe)—Setting this bit to 1 clears Gi_TC_Error_St. This bit is cleared automatically.
21	**Gi_Gate_Error_Confirm** (Strobe)—Setting this bit to 1 clears Gi_Gate_Error_St. This bit is cleared automatically.
20	**Gi_SampleClockInterruptAck** (Strobe)—Setting this bit to 1 clears Gi_SampleClockInterruptSt. This bit is cleared automatically.
19	**Gi_SampleClockOverrunErrorAck** (Strobe)—When set, this bit clears the overrun error status bit and the overrun interrupt if enabled.When stop on error is set, this also enables the counter to start saving data again.
18..16	**Reserved**
15	**Gi_DMA_Error_Interrupt_Enable** (Strobe)—This bit enables the counter's DMA Error Interrupt. This error is the equivalent of a FIFO overflow error for input operations, and an underflow error for Output operations.
14	**Gi_TC_Error_Interrupt_Enable** (Strobe)—This bit enables the counter's TC error interrupt. This interrupt sets when the latency for acknowledging the TC interrupt is too long and another TC is received.
13	**Gi_Gate_Error_Interrupt_Enabled** (Strobe)—This bit enables the counter's Gate error interrupt. This interrupt sets when the latency for acknowledging the Gate interrupt is too long and another Gate is received.
12	**Gi_SampleClockOverrunIntEn** (Strobe)—This bit enables the generation of an interrupt on an overrun error from the Sample Clock signal.
11	**Gi_TC_Interrupt_Enable** (Strobe)—The Gi_TC interrupt occurs on the rising edge of counter's TC. This bit enables the counter's TC interrupt.
10	**Gi_Gate_Interrupt_Enable** (Strobe)—The relevant gate edge is the stop edge in case of level gating, or the active edge (both start and stop) in the case of edge gating. This bit enables the counter's gate interrupt. **Note**: Do not use this interrupt for data transfer operations with the save registers or the FIFO. Use the Gi DMA interrupt instead.
9	**Gi_DMA_Int_Enable** (Strobe)—Direct Memory Access Interrupt. This bit enables the generation of interrupts for data transfer using the DMA mode circuit. As when DMA is enabled, it uses the internal FIFO, and generates the interrupt based on its status. This mode of generating interrupts should be used to transfer data using interrupts instead of using the gate interrupt.
8	**Gi_WritesTooFastErrorEn** (Strobe)—This bit enables the generation of interrupts when a WritesTooFast error occurs.
7	**Gi_GateSwitchErrorInt_En** (Strobe)—This bit enables the generation of interrupts when a GateSwitchError occurs.

Bits	Name
6	**Gi_SampleClockInterruptEn** (Strobe)—This strobe bit enables the Sample Clock to generate an interrupt only if that sample clock writes data to the FIFO; this means either an error-free sample clock, or an overrun when not in “stop on error” mode.
5	**Gi_Disarm_Interrupt_En** (Strobe)—This strobe bit enables the Disarm Event to generate an interrupt.
4	**Gi_Aux_Ctr_TC_Interrupt_En** (Strobe)—This strobe bit enables the Gi_Aux_Ctr_TC to generate an interrupt.
3	**Gi_Aux_Ctr_TC_Error_Interrupt_En** (Strobe)—This strobe bit enables the Gi_Aux_Ctr_TC_Error event to generate an interrupt.
2..0	**Reserved**
Options: No Soft Copy	

Offset 0x30: Gi_Interrupt2_Register (W)

Absolute Addresses:
G0: 0x20330
G1: 0x20370
G2: 0x203B0
G3: 0x203F0

Bits	Name
31	**Gi_Gate_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears Gi_Gate_Interrupt_St and acknowledges the gate interrupt request for the counter when the gate interrupt is enabled. This bit is cleared automatically. **Note**: Do not use gate interrupts on the DAQ-STC3 for interrupt-driven data transfer. Use the DMA interrupts instead.
30	**Gi_TC_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears Gi_TC_St and acknowledges the TC interrupt request for the counter when the TC interrupt is enabled. This bit is cleared automatically.
29	**Gi_DMA_Error_Ack2** (Strobe)—Setting this bit to 1 clears Gi_DRQ_Error. This bit is cleared automatically.
28	**Gi_GateSwitchError_Ack2** (Strobe)—Setting this bit to 1 clears Gi_GateSwitchError_St. This bit is cleared automatically.
27	**Gi_WritesTooFastErrorAck2** (Strobe)—Setting this bit to 1 clears Gi_WritesTooFastErrSt. This bit is cleared automatically.
26	**Gi_Disarm_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears Gi_Disarm_Interrupt_St and acknowledges the interrupt. This bit is cleared automatically.
25	**Gi_Aux_Ctr_TC_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears Gi_Aux_Ctr_TC_Interrupt_St and acknowledges the interrupt. This bit is cleared automatically.

Bits	Name
24	**Gi_Aux_Ctr_TC_Error_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears Aux Ctr TC Error Interrupt_St and acknowledges the interrupt. This bit is cleared automatically.
23	**Reserved**
22	**Gi_TC_Error_Confirm2** (Strobe)—Setting this bit to 1 clears Gi_TC_Error_St. This bit is cleared automatically.
21	**Gi_Gate_Error_Confirm2** (Strobe)—Setting this bit to 1 clears Gi_Gate_Error_St. This bit is cleared automatically.
20	**Gi_SampleClockInterruptAck2** (Strobe)—Setting this bit to 1 clears Gi_SampleClockInterruptSt. This bit is cleared automatically.
19	**Gi_SampleClockOverrunErrorAck2** (Strobe)—When set, this bit clears the overrun error status bit and the overrun interrupt if enabled. If stop on error is set, this also enables the counter to start saving data again.
18..16	**Reserved**
15	**Gi_DMA_Error_Interrupt_Disable** (Strobe)—This strobe bit disables the counter's DMA Error Interrupt. This error is the equivalent of a FIFO overflow error for input operations, and an underflow error for Output operations.
14	**Gi_TC_Error_Interrupt_Disable** (Strobe)—This strobe bit disables the counter's TC error interrupt. This interrupt sets when the latency for acknowledging the TC interrupt is too long and another TC is received.
13	**Gi_Gate_Error_Interrupt_Disable** (Strobe)—This strobe bit disables the counter's Gate error interrupt. This interrupt sets when the latency for acknowledging the Gate interrupt is too long and another Gate is received.
12	**Gi_SampleClockOverrunIntDis** (Strobe)—This strobe bit disables the generation of an interrupt on an overrun error from the Sample Clock signal.
11	**Gi_TC_Interrupt_Disable** (Strobe)—This strobe bit disables the counter's TC interrupt. **Note**: The Gi_TC interrupt occurs on the rising edge of counter's TC.
10	**Gi_Gate_Interrupt_Disable** (Strobe)—This strobe bit disables the counter's gate interrupt. **Note**: The relevant gate edge is the stop edge in case of level gating, or the active edge (both start and stop) in the case of edge gating. **Note**: Do not use this bitfield to read data of the save registers or the FIFO. Use the Gi DMA interrupt instead.
9	**Gi_DMA_Int_Disable** (Strobe)—Direct Memory Access Interrupt. This strobe bit disables the generation of interrupts for data transfer using the DMA mode circuit. As when DMA is enabled, it uses internal FIFO, and generates the interrupt based on its status. **Note**: This mode of generating interrupts should be used to transfer data using interrupts, instead of using the gate interrupt.
8	**Gi_WritesTooFastErrorDis** (Strobe)—This bit disables the generation of interrupts when a WritesTooFast error occurs.

Bits	Name
7	**Gi_GateSwitchErrorInt_Dis** (Strobe)—This bit disables the generation of interrupts when a GateSwitchError occurs.
6	**Gi_SampleClockInterruptDis** (Strobe)—This strobe bit disables the Sample Clock to generate an interrupt if, and only if, that sample clock writes data to the FIFO; this means either an error-free sample clock, or an overrun when not in “stop on error” mode.
5	**Gi_Disarm_Interrupt_Dis** (Strobe)—This strobe bit disables the Disarm Event to generate an interrupt.
4	**Gi_Aux_Ctr_TC_Interrupt_Dis** (Strobe)—This strobe bit disables the Gi_Aux_Ctr_TC to generate an interrupt.
3	**Gi_Aux_Ctr_TC_Error_Interrupt_Dis** (Strobe)—This strobe bit disables the Gi_Aux_Ctr_TC_Error event to generate an interrupt.
2..0	**Reserved**
Options: No Soft Copy	

Offset 0x38: Gi_ABZ_Select_Register (W)

Absolute Addresses:
G0: 0x20338
G1: 0x20378
G2: 0x203B8
G3: 0x203F8

Bits	Name
31..22	**Reserved**
21..16	**Gi_A_Select**—This field selects the A signal source during position measurements. The values for this bitfield are in *Gi_A_Select_t*.
15..14	**Reserved**
13..8	**Gi_B_Select**—This field selects the B signal source during position measurements. **Note**: This mux can also be used to select the source for the G_UP_DOWN signal when the Gi_Up_Down bitfield is set to Gi_B_High_Up. The values for this bitfield are in *Gi_B_Select_t*.
7..6	**Reserved**
5..0	**Gi_Z_Select**—This field selects the Z signal source during position measurements. The values for this bitfield are in *Gi_Z_Select_t*.

Offset 0x3C: Gi_Mode3_Register (W)

Absolute Addresses:
G0: 0x2033C
G1: 0x2037C
G2: 0x203BC
G3: 0x203FC

Bits	Name
15..3	**Reserved**
2	**Gi_TimeCoherentSemiperiod**—This bit affects how the counter performs semiperiod measurements with the sample clock. The counter measures a consecutive low and high semiperiod, and returns them in the order requested (for example, high, and then low). When the bit is cleared, the high and low semiperiods can be acquired in any chronological order. This is the fastest measurement but there is uncertainty of the order of the two semiperiods. For example, counter gets a sample clock while measuring the low semiperiod, finishes that measurement, and then measures the high semiperiod. It will return the data then in the requested order: high, then low. When the bit is set, the counter forces the chronological order of the measurements to be the same as what was requested. In the example listed above, if the counter gets the sample clock while measuring the low semiperiod, it discards the measurement and waits until a high semiperiod measurement begins. After that is completed, it performs a low semiperiod measurement and finally returns the data as requested (high and then low). The latency in this mode could be larger when the bit is not set.
1..0	**Reserved**

Offset 0x3E: Gi_Mode2_Register (W)

Absolute Addresses:
G0: 0x2033E
G1: 0x2037E
G2: 0x203BE
G3: 0x203FE

Bits	Name
15..14	**Gi_Up_Down**—This bit selects the up/down mode: • Software-selected down counting. • Software-selected up counting. • Hardware-selected up/down signal selected by the Gi_B signal Mux (refer to the *Gi_B_Select* enumeration). Logic low counts down and logic high counts up. • Hardware-selected up/down counting controlled by the internal gate value (refer to the *Gi_Polarity* enumeration), where the active gate level counts up and the inactive gate level counts down. The selection can be safely changed while the counter is counting. The values for this bitfield are in *Gi_Up_Down_t*.
13	**Reserved**
12	**Gi_Bank_Switch_Enable**—When the general-purpose counter is not armed, this bit selects the bank to which you can write: Bank X if the bit is 0, Bank Y if the bit is 1. If the general-purpose counter is armed, the value of this bit just before arming enables bank switching: disabled if this bit is 0 before arming, enabled if this bit is 1. It is recommended that after loading all banks with data, but before arming, the desired bank switching behavior is written to this register. The values for this bitfield are in *Gi_Bank_Switch_Enable_t*.
11..10	**Gi_Bank_Switch_Mode**—This bit selects the source that controls general-purpose counter load register bank switching, if bank switching is enabled. The values for this bitfield are in *Gi_Bank_Switch_Mode_t*.
9	**Reserved**
8	**Gi_WriteOnSwitchRequest**—When this mode is enabled, writes to the counter banks are enabled until a switch is requested (normally using the gate as a sample clock). This way, if a write from the CPU occurs after a switch is requested but before it happens, this write will not invalidate the pending switch. In this mode, an underflow error occurs if at the moment of the switch request, the data written to the load registers is not sufficient or not coherent. Also, a Writes Too Fast error happens when data is written twice to the load registers before a new switch request is received.
7..4	**Reserved**

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>3</td><td colspan="2">Gi_StopOnError—For Counter Input Operations—This bit forces the counter to stop saving data on an overrun error condition. If not set, the counter ignores the error and continues saving data on valid sample clock signals. The counter starts saving data again if the Overrun error is acknowledged.

For Counter Output Operations—This bit prevents the counter to generate further switches of waveforms after an underflow error has been detected. It does not stop the counter. It only stops the waveform switching.</td></tr>
<tr><td></td><td>Value</td><td>Meaning</td></tr>
<tr><td></td><td>0</td><td>Do not stop on error.</td></tr>
<tr><td></td><td>1</td><td>Stop on error.</td></tr>
<tr><td>2</td><td colspan="2">Gi_CtrOutFifoRegenerationEn—This bit enables the Counter Output FIFO for regeneration when empty. When this bit is enabled, the FIFO regenerates all the data in it upon becoming empty.</td></tr>
<tr><td>1..0</td><td colspan="2">Gi_HwArmSyncMode—This bitfield determines the mode of operation of the HW arm trigger synchronization.

The values for this bitfield are in Gi_HwArmSyncMode_t.</td></tr>
</table>

Enumerated Types

Gi_A_Select_t

Value	Name	Value	Name
1	**A_PFI0**	20	**A_Star_Trig**
2	**A_PFI1**	21	**A_PFI10**
3	**A_PFI2**	22	**A_PFI11**
4	**A_PFI3**	23	**A_PFI12**
5	**A_PFI4**	24	**A_PFI13**
6	**A_PFI5**	25	**A_PFI14**
7	**A_PFI6**	26	**A_PFI15**
8	**A_PFI7**	27	**A_RTSI7**
9	**A_PFI8**	30	**A_Analog_trig**
10	**A_PFI9**	31	**A_LogicLow**
11	**A_RTSI0**	46	**A_IntTrigA_0**
12	**A_RTSI1**	47	**A_IntTrigA_1**
13	**A_RTSI2**	48	**A_IntTrigA_2**
14	**A_RTSI3**	49	**A_IntTrigA_3**
15	**A_RTSI4**	50	**A_IntTrigA_4**
16	**A_RTSI5**	51	**A_IntTrigA_5**
17	**A_RTSI6**	52	**A_IntTrigA_6**
18	**A_PXIe_DStarA**	53	**A_IntTrigA_7**
19	**A_PXIe_DStarB**		

Gi_Armed_St_t

Value	Name
0	**Not_Armed**
1	**Armed**

Gi_AuxCtrMode_t

Value	Name
0	**Aux_Disabled**—Default mode. Aux counter is not used.
1	**Aux_FrequencyMeasurement**—Frequency measurement mode. In this mode, the aux counter is used with the main counter to perform a reciprocal frequency measurement. The input signal goes in the Ctr gate. The counter counts the number of periods on the signal at the gate with the Main counter, and counts the ticks of the source that occurred on that period of time with the Aux counter. The period of time is defined by the sample clock. The counter starts counting on the first edge of the gate and accumulates counts until the sample clock. At the end of the period after the sample clock the counters reload and start a new measurement.
2	**Aux_FinitePulseTrain**—Finite pulse train mode. This mode allows a finite pulse train generation at the output of the counter with a programmable initial delay. This initial delay can optionally use the auto-increment feature to make it variable. In this mode, the main counter is set up to perform a delayed pulse generation (which can be triggered). The initial delay is from the moment the counter starts counting (arm, or gate if triggered), until the TC of the counter. This period of time is measured in ticks of the source. At that point, the Main counter reloads, flips the output, enables the Aux counter to count and switches from counting ticks of the source to count TC's of the Aux counter. On the second TC of Main, the Aux counter is disabled again. If retriggered, the counter again loads the delay from trigger to pulse train. If this delay is in load register A, then auto-increment can be used. The Aux counter should be set up for the actual pulse train specs. The first change on the output is caused by the Main counter. This takes the output from its idle to its active state. Therefore, the first count of the Aux counter (pre-load value) will be its active state time. At its TC, the counter flips the output (to its inactive state) and starts counting the inactive time of the waveform. In other words, if the output polarity is high, the output will go "high until TC" to "low until TC," and so on. The minimum inactive pulse width is two ticks for all cases. The minimum active pulse with is: • Two ticks when the source is not TB3 or when the Src != TB, TB=TB3 (Gi_ForceSourceEqualToTimebase = 0). • Three ticks when the source is TB3 or when Src = TB (Gi_ForceSourceEqualToTimebase = 1). **Note**: It is possible to generate waveforms with two ticks active in these modes by programming the main counter to count one extra TC of the Aux counter. This works around the counter limitation.

Value	Name
3	**Aux_MinFinitePulseTrain**—The Min Finite Pulse Train mode differs from the normal Finite pulse train case because the operation is optimized for minimum trigger to output pulse time. In this mode, the trigger (gate) causes the first toggle of the output and enables the Aux counter to count. The main counter only counts TCs of the Aux counter. When the Main counter generates a TC, the Aux counter is disabled again. The Aux counter should be set up for the actual pulse train specs. The first change on the output is caused by the Gate of the counter. This takes the output from its idle to its active state. Therefore, the first count of the Aux counter (pre-load value) is its active state time. At its TC, the counter flips the output (to its inactive state) and starts counting the inactive time of the waveform. In other words, if the output polarity is high, the output will go “high until TC” to “low until TC,” and so on. The minimum inactive pulse width is two ticks for all cases. The minimum active pulse with is: • Two ticks when the source is not TB3 or when the Src != TB, TB=TB3 (Gi_ForceSourceEqualToTimebase = 0). • Three ticks when the source is TB3 or when Src = TB (Gi_ForceSourceEqualToTimebase = 1). **Note**: It is *not* possible to work around the counter limitation in this case. Because the first pulse is caused by the gate of the counter, there could be a pulse width distortion of up to one source period.
4	**Aux_GateEventCounting**—This mode enables the aux counter for event counting on the gate terminal. This can be used to implement two parallel event counting mode in which the Main counter is counting events on the source terminal and the Aux counter is counting events on the gate, and the sampling is done with the sample clock (in MainAndAux mode). Another possible application is event counting in parallel with position measurement. This works in a similar way as the two parallel event counting.
5	**Aux_FinitePwmGeneration**—This mode enables the Aux counter to count when the Main counter switches banks. It is intended to work with the Main counter programmed for normal counting mode in output generation. When this mode is selected, the options for the main counter to stop/disarm on TC refer to the Aux counter’s TC, enabling this option to perform a finite PWM generation (where finite refers to the number of bank switches). Software can use the Disarm interrupt or the Aux TC interrupt to know when a finite operation has completed. Both finite or continuous PWM generation can generate a minimum of two ticks for both the active and inactive pulses. When the timing of the PWM is implicit, and the source is TB3 or when Src = TB (Gi_ForceSourceEqualToTimebase = 1), the counter has a limit of five ticks minimum for the period. That is, the sum of the inactive and active pairs should be five or more.

Gi_B_Select_t

Value	Name	Value	Name
0	**B_LogicLow0**	19	**B_PXIe_DStarB**
1	**B_PFI0**	20	**B_Star_Trig**
2	**B_PFI0**	21	**B_PFI10**
3	**B_PFI2**	22	**B_PFI11**
4	**B_PFI3**	23	**B_PFI12**
5	**B_PFI4**	24	**B_PFI13**
6	**B_PFI5**	25	**B_PFI14**
7	**B_PFI6**	26	**B_PFI15**
8	**B_PFI7**	27	**B_RTSI7**
9	**B_PFI8**	30	**B_Analog_Trig**
10	**B_PFI9**	31	**B_LogicLow**
11	**B_RTSI0**	46	**B_IntTrigA_0**
12	**B_RTSI1**	47	**B_IntTrigA_1**
13	**B_RTSI2**	48	**B_IntTrigA_2**
14	**B_RTSI3**	49	**B_IntTrigA_3**
15	**B_RTSI4**	50	**B_IntTrigA_4**
16	**B_RTSI5**	51	**B_IntTrigA_5**
17	**B_RTSI6**	52	**B_IntTrigA_6**
18	**B_PXIe_DStarA**	53	**B_IntTrigA_7**

Gi_Bank_Switch_Enable_t

Value	Name
0	**Disabled_If_Armed_Else_Write_To_X**
1	**Enabled_If_Armed_Else_Write_To_Y**

Gi_Bank_Switch_Mode_t

Value	Name
0	**Gate**—The gate of the counter controls the switching of the banks. If DMA mode is on, the gate event schedules the bank switching. If DMA mode is off, the gate schedules a bank switching but only after software allows it by writing to the Gi_Bank_Switch_Start bit.
1	**Software**—Implicit timing. If DMA mode is on, each bank is generated once. If DMA is off, the bank is generated until software schedules a switch with the Gi_Bank_Switch_Start bit.
2	**SampleClk**—Same as Gate, but uses the Sample Clock signal instead. The advantage of this mode is that it allows for separate start trigger and sample clock signals on the generation.

Gi_Counting_Once_t

Value	Name
0	**NoHardwareDisarm**—No hardware disarm.
1	**DisarmAtTcThatStops**—Disarm at the TC that stops counting.
2	**DisarmAtGateThatStops**—Disarm at the G_Gate that stops counting.
3	**DisarmAtTcOrGateThatStops**—Disarm at the TC or G_Gate that stops counting, whichever comes first.

Gi_CountingMode_t

Value	Name
0	**NormalCounting**—Default mode.
1	**QuadEncoderX1**—Enables the main counter in Quadrature encoder X1 mode.
2	**QuadEncoderX2**—Enables the main counter in Quadrature encoder X2 mode.
3	**QuadEncoderX4**—Enables the main counter in Quadrature encoder X4 model.
4	**TwoPulseEncoder—**Enables the main counter in Two Pulse position measurement mode.
6	**FrequencyMeasure**—Enables the Aux and Main counters to do two counter, reciprocal mode frequency measurement.
7	**FinitePulseTrain**—Enables the Aux counter to do finite pulse train generation.

Gi_Disabled_Enabled_t

Value	Name
0	**Disabled**
1	**Enabled**

Gi_Gate_Select_t

Value	Name	Value	Name
0	**Gate_DioChgDetect**	23	**Gate_PFI12**
1	**Gate_PFI0**	24	**Gate_PFI13**
2	**Gate_PFI1**	25	**Gate_PFI14**
3	**Gate_PFI2**	26	**Gate_PFI15**
4	**Gate_PFI3**	27	**Gat_RTSI7**
5	**Gate_PFI4**	28	**Gate_AI_START1**
6	**Gate_PFI6**	29	**Gate_G_PairedSrc**
7	**Gate_PFI6**	30	**Gate_Atrig**
8	**Gate_PFI7**	31	**Gate_PXIe_DStarA**
9	**Gate_PFI8**	32	**Gate_PXIe_DStarB**
10	**Gate_PFI9**	41	**Gate_IntTrigA_0**
11	**Gate_RSTI0**	42	**Gate_IntTrigA_1**
12	**Gate_RSTI1**	43	**Gate_IntTrigA_2**
13	**Gate_RSTI2**	44	**Gate_IntTrigA_3**
14	**Gate_RSTI3**	45	**Gate_IntTrigA_4**
15	**Gate_RSTI4**	46	**Gate_IntTrigA_5**
16	**Gate_RSTI5**	47	**Gate_IntTrigA_6**
17	**Gate_RSTI6**	48	**Gate_IntTrigA_7**
18	**Gate_AI_START2**	59	**Gate_DI_START1**
19	**Gate_StarTrig**	60	**Gate_DI_START2**
20	**Gate_G_PairedOut**	61	**Gate_AO_START1**
21	**Gate_PFI10**	62	**Gate_DO_START1**
22	**Gate_PFI11**	63	**Gate_LogicLow**

Gi_GatingMode_t

Value	Name
0	**GateDisabled**
1	**LevelGating**
2	**AssertingEdgeGating**—Rising edge if Gi_Gating_Polarity is set to 0, and falling edge if Gi_Gating_Polarityis set to 1.
3	**DeassertingEdgeGating**—Falling edge if Gi_Gating_Polarity is set to 0, and rising edge if Gi_Gating_Polarity is set to 1.

Gi_HW_Arm_Select_t

Value	Name	Value	Name
0	**HwArm_DIO_ChgDetect**	23	**HwArm_PFI1**
1	**HWArm_PFI0**	24	**HwArm_PFI2**
2	**HWArm_PFI1**	25	**HwArm_PFI3**
3	**HWArm_PFI2**	26	**HwArm_PFI4**
4	**HWArm_PFI3**	27	**HwArm_RTSI7**
5	**HWArm_PFI4**	28	**HwArm_AI_START1**
6	**HWArm_PFI5**	29	**HwArm_Analog_Trig**
7	**HWArm_PFI6**	30	**HwArm_DI_Start1**
8	**HWArm_PFI7**	31	**HwArm_AO_Start1**
9	**HWArm_PFI8**	32	**HwArm_DO_Start1**
10	**HWArm_PFI9**	33	**HwArm_PXIe_DStarA**
11	**HwArm_RTSI0**	34	**HwArm_PXIe_DStarB**
12	**HwArm_RTSI1**	35	**HwArm_IT0_Start1**
13	**HwArm_RTSI2**	36	**HwArm_IT1_Start1**
14	**HwArm_RTSI3**	48	**HwArm_IntTrigA_0**
15	**HwArm_RTSI4**	49	**HwArm_IntTrigA_1**
16	**HwArm_RTSI5**	50	**HwArm_IntTrigA_2**
17	**HwArm_RTSI6**	51	**HwArm_IntTrigA_3**
18	**HwArm_AI_START2**	52	**HwArm_IntTrigA_4**
19	**HwArm_Star_Trig**	53	**HwArm_IntTrigA_5**

Value	Name	Value	Name
20	**HwArm_G_PairedOut**	54	**HwArm_IntTrigA_6**
21	**HwArm_PFI0**	55	**HwArm_IntTrigA_7**
22	**HwArm_PFI1**		

Gi_HwArmSyncMode_t

Value	Name
0	**SyncDefault**—This is the default mode of operation. The trigger is synchronized to the source of the timer for internal use only.
1	**SyncSlave**—In this mode, the timer is set up to receive the exported trigger from a master device. The transmission and reception of this trigger depends on the SyncPulse signal (PXIe_Sync100 on PXI Express systems).
2	**SyncMaster**—In this mode, the timer transmits the trigger and then waits to use it so that all devices (master and slaves) trigger at the same time.

Gi_Index_Mode_t

Value	Name
0	**IndexModeCleared** • Quadrature Mode: The index (Z) is disabled. • Two Pulse Mode: The index (Z) is disabled. • Normal Counting Mode: Enables edge detection on the source pin.
1	**IndexModeSet** • Quadrature Mode: The index (Z) reloads the counter in the phase determined by the value of Gi_Index_Phase when the Z input is high. • Two Pulse Mode: The counter reloads while Z is active. • Normal Counting Mode: Disables edge detection on the source pin. Edge detection allows the signal to be narrower pulses but introduces more delay.

Gi_Index_Phase_t

Value	Name
0	**A_low_B_low**
1	**A_low_B_high**
2	**A_high_B_low**
3	**A_high_B_low**

Gi_Load_Source_Select_t

Value	Name
0	**Load_From_Register_A**
1	**Load_From_Register_B**

Gi_Loading_On_Gate_t

Value	Name
0	**NoCounterReloadOnGate**
1	**ReloadOnStopGate**

Gi_Loading_On_TC_t

Value	Name
0	**RolloverOnTC**
1	**ReloadOnTC**

Gi_Output_Mode_t

Value	Name
1	**TC_mode**—The counter TC signal appears on G_OUT.
2	**Toggle_Output_On_TC**—G_OUT changes state on the trailing edge of counter TC.
3	**Toggle_Output_On_TC_or_Gate**—G_OUT changes state on the trailing edge of counter TC and on the active gate edge. This mode can be used for sequential scanning.

Gi_Output_St_t

Value	Name
0	**Low**
1	**High**

Gi_Polarity_t

Value	Name
0	**ActiveHigh**
1	**ActiveLow**

Gi_Reload_Source_Switching_t

Value	Name
0	**UseSameLoadRegister**
1	**UseAlternatingLoadRegisters**

Gi_SampleClkSampleMode_t

Value	Name
0	**SC_NextSaved**—In NextSaved mode, the counter starts a new valid measurement after receiving the sample clock. This mode guarantees that the measurement reflects the state of the signal after the sample clock, but has higher and unknown latency impact.
1	**SC_LastSaved**—In LastSaved, the counter returns the last completed acquisition, even if this started or even completed before the sample clock is received. This has a latency advantage, since the data is returned immediately, though the data could be arbitrarily old.

Gi_SampleClockMode_t

Value	Name
0	**SC_Disabled**—The Sample Clock is disabled. All measurements are saved.
1	**SC_SingleSample**—The Sample Clock is enabled in single mode. After the sample clock, the counter saves one measurement corresponding to the saving mode of the counter (gate mode or the value of the counter if gate independent is enabled).
2	**SC_DoubleSampleRF**—The Sample Clock is enabled in double mode. After the sample clock, the counter saves two measurements corresponding to Rising to Falling Edge.
3	**SC_DoubleSampleFR**—The Sample Clock is enabled in double mode. After the sample clock, the counter saves two measurements corresponding to Falling to Rising Edge.
4	**SC_MainAndAux**—The sample clock is enabled and saves the values of the Main and the Aux counter in a coherent way. This mode can be used for frequency mode or double event counting mode.

Gi_SampleClockSelect_t

Value	Name	Value	Name
0	**SampleClk_SwPulse**	24	**SampleClk_PFI13**
1	**SampleClk_PFI0**	25	**SampleClk_PFI14**
2	**SampleClk_PFI1**	26	**SampleClk_PFI15**
3	**SampleClk_PFI2**	27	**SampleClk_RTSI7**
4	**SampleClk_PFI3**	28	**SampleClk_AI_START**
5	**SampleClk_PFI4**	30	**SampleClk_Atrig**
6	**SampleClk_PFI5**	31	**SampleClk_DI_Convert**
7	**SampleClk_PFI6**	32	**SampleClk_DI_Start1**
8	**SampleClk_PFI7**	33	**SampleClk_AI_Convert**
9	**SampleClk_PFI8**	34	**SampleClk_AI_Start1**
10	**SampleClk_PFI9**	35	**SampleClk_AI_Start2**
11	**SampleClk_RTSI0**	36	**SampleClk_AO_Update**
12	**SampleClk_RTSI1**	37	**SampleClk_DO_Update**
13	**SampleClk_RTSI2**	38	**SampleClk_PXIe_DStarA**
14	**SampleClk_RTSI3**	39	**SampleClk_PXIe_DStarB**
15	**SampleClk_RTSI4**	48	**SampleClk_IntTrigA_0**
16	**SampleClk_RTSI5**	49	**SampleClk_IntTrigA_1**
17	**SampleClk_RTSI6**	50	**SampleClk_IntTrigA_2**
19	**SampleClk_Star_Trig**	51	**SampleClk_IntTrigA_3**
20	**SampleClk_DIO_ChgDetect**	52	**SampleClk_IntTrigA_4**
21	**SampleClk_PFI10**	53	**SampleClk_IntTrigA_5**
22	**SampleClk_PFI11**	54	**SampleClk_IntTrigA_6**
23	**SampleClk_PFI12**	55	**SampleClk_IntTrigA_7**

Gi_Second_Gate_Mode_t

Value	Name
0	**DisabledSecondGate**
1	**SecondGateAssertFirstDeassertsGate**

Gi_Second_Gate_Select_t

Value	Name	Value	Name
0	**Gate2_LogicLow**	22	**Gate2_PFI11**
1	**Gate2_PFI0**	23	**Gate2_PFI12**
2	**Gate2_PFI1**	24	**Gate2_PFI13**
3	**Gate2_PFI2**	25	**Gate2_PFI14**
4	**Gate2_PFI3**	26	**Gate2_PFI15**
5	**Gate2_PFI4**	27	**Gate2_RTSI7**
6	**Gate2_PFI5**	28	**Gate2_G_PairedGate**
7	**Gate2_PFI6**	29	**Gate_G_PairedSrc**
8	**Gate2_PFI7**	30	**Gate2_G_Gate1**
9	**Gate2_PFI8**	31	**Gate2_PXIe_DStarA**
10	**Gate2_PFI9**	32	**Gate2_PXIe_DStarB**
11	**Gate2_RTSI0**	33	**Gate2_Atrig**
12	**Gate2_RTSI1**	34	**Gate2_DioChgDetect**
13	**Gate2_RTSI2**	35	**Gate2_AI_START2**
14	**Gate2_RTSI3**	46	**Gate2_IntTrigA_0**
15	**Gate2_RTSI4**	47	**Gate2_IntTrigA_1**
16	**Gate2_RTSI5**	48	**Gate2_IntTrigA_2**
17	**Gate2_RTSI6**	49	**Gate2_IntTrigA_3**
18	**Gate2_AI_START1**	50	**Gate2_IntTrigA_4**
19	**Gate2_Star_Trig**	51	**Gate2_IntTrigA_5**
20	**Gate2_G_PairedOut**	52	**Gate2_IntTrigA_6**
21	**Gate2_PFI10**	53	**Gate2_IntTrigA_7**

Gi_Source_Select_t

Value	Name	Value	Name
1	**Src_PFI0**	23	**Src_PFI12**
2	**Src_PFI1**	24	**Src_PFI13**
3	**Src_PFI2**	25	**Src_PFI14**
4	**Src_PFI3**	26	**Src_PFI15**
5	**Src_PFI4**	27	**Src_RTSI7**
6	**Src_PFI5**	28	**Src_TB1**
7	**Src_PFI6**	29	**Src_PXIClk10**
8	**Src_PFI7**	30	**Src_TB3**
9	**Src_PFI8**	32	**Src_StarTrig**
10	**Src_PFI9**	33	**Src_PXIe_DStarA**
11	**Src_RTSI0**	34	**Src_PXIe_DStarB**
12	**Src_RTSI1**	43	**Src_IntTrigA_0**
13	**Src_RTSI2**	44	**Src_IntTrigA_1**
14	**Src_RTSI3**	45	**Src_IntTrigA_2**
15	**Src_RTSI4**	46	**Src_IntTrigA_3**
16	**Src_RTSI5**	47	**Src_IntTrigA_4**
17	**Src_RTSI0**	48	**Src_IntTrigA_5**
18	**Src_TB2**	49	**Src_IntTrigA_6**
19	**Src_G_PAiredTC**	50	**Src_IntTrigA_7**
20	**Src_G_PairedGate**	59	**Src_DIO_ChgDetect**
21	**Src_PFI10**	60	**Src_Atrig**
22	**Src_PFI11**		

Gi_Stop_Mode_t

Value	Name
0	**StopOnGateCondition**—Stop on gate condition.
1	**StopAtGateOrFirstTC**—Stop on gate condition or at the first TC, whichever comes first.
2	**StopAtGateOrSecondTC**—Stop on gate condition or at the second TC, whichever comes first.

Gi_Trigger_Mode_For_Edge_Gate_t

Value	Name
0	**FirstGateStartsSecondGateStops**—The first gate edge starts the counting, and the next gate edge stops the counting.
1	**FirstGateStopsSecondGateStarts**—The first gate edge stops the counting, and the next gate edge starts the counting.
2	**GateEdgeStarts**—The gate edge always starts the counting, unless counting is already in progress, in which case the edge is ignored. The valid Gi_Stop_Mode settings for this selection are 1 and 2, but only the TC (not the gate) stops the counting.
3	**GateLoads**—The gate is used for reload, save, or load select only (if any of those options is enabled). The gate is not used for stopping.

Gi_Up_Down_t

Value	Name
0	**CountDown**
1	**CountUp**
2	**Gi_B_High_Up**
3	**Gi_Gate_Active_Up**

Gi_Z_Select_t

Value	Name	Value	Name
1	**Z_PFI0**	20	**Z_Star_Trig**
2	**Z_PFI1**	21	**Z_PFI10**
3	**Z_PFI2**	22	**Z_PFI11**
4	**Z_PFI3**	23	**Z_PFI12**
5	**Z_PFI4**	24	**Z_PFI13**
6	**Z_PFI5**	25	**Z_PFI14**
7	**Z_PFI6**	26	**Z_PFI15**
8	**Z_PFI7**	27	**Z_RTSI7**
9	**Z_PFI8**	30	**Z_Analog_Trig**
10	**Z_PFI9**	31	**Z_LogicLow**
11	**Z_RTSI0**	46	**Z_IntTrigA_0**
12	**Z_RTSI1**	47	**Z_IntTrigA_1**
13	**Z_RTSI2**	48	**Z_IntTrigA_2**
14	**Z_RTSI3**	49	**Z_IntTrigA_3**
15	**Z_RTSI4**	50	**Z_IntTrigA_4**
16	**Z_RTSI5**	51	**Z_IntTrigA_5**
17	**Z_RTSI6**	52	**Z_IntTrigA_6**
18	**Z_PXIE_DStarA**	53	**Z_IntTrigA_7**
19	**Z_PXIe_DStarB**		

14

Analog Output Registers

AO base address = 0x20400

This chapter consists of *Analog Output Registers* and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

The analog output registers contain an OutTimer object, offset: 0x20470. Refer to Chapter 18, *OutTimer Registers*, for more information.

List of Analog Output Registers

The analog output registers are as follows:

Offset 0x0: AO_DacShadow (i) Register Array (R)

Offset 0x0: AO_Direct_Data (i) Register Array (W)

Offset 0x40: AO_Order_Config_Data_Register (W)

Offset 0x44: AO_Config_Control_Register (W)

Offset 0x48: AO_Trigger_Select_Register (W)

Offset 0x4C: AO_Config_Bank (i) Register Array (W)

Offset 0x58: AO_FIFO_Status_Register (R)

Offset 0x58: AO_FIFO_Data_Register (W)

List of Enumerated Types

The enumerated types are as follows:

AO_Bipolar_t

AO_External_Gate_Select_t

AO_Polarity_t

AO_START1_Select_t

AO_Update_Mode_t

AO_UPDATE_Source_Select_t

Analog Output Registers

Offset 0x0: AO_DacShadow (i) Register Array (R)

Absolute Address:
AO: 0x20400

Array Offsets = 0x0 + (i × 4) [8 registers]

Bits	Name
31..0	**AO_DacShadow_Bitfield**—This register returns the last value that was updated on the DACi. The value returned by this register needs to be masked by the resolution of the DAC. The data is LSB aligned. So, for a 16-bit DAC, only bits 0–15 should be considered.

Offset 0x0: AO_Direct_Data (i) Register Array (W)

Absolute Address:
AO: 0x20400

Array Offsets = 0x0 + (i × 4) [16 registers]

Bits	Name
31..0	**AO_Direct_Data_Bitfield** (Strobe)—This register is used to write data directly to DACi.
Options: No Soft Copy, No Hardware Reset	

Offset 0x40: AO_Order_Config_Data_Register (W)

Absolute Address:
AO: 0x20440

Bits	Name
31..4	**Reserved**
3..0	**AO_Waveform_Bitfield_Order**—This bitfield sets the order of all the channels in the generation. This field acts as the data input for an order configuration FIFO. Sequential writes to this register sets the order of the waveform, with the first channel written being the first one on the waveform, and so on. Make sure the order configuration FIFO is cleared before starting the sequence of writes. All writes to this register must be performed during a configuration cycle and no writes are allowed outside configuration.
Options: No Soft Copy	

Offset 0x44: AO_Config_Control_Register (W)

Absolute Address:
AO: 0x20444

Bits	Name
31..1	**Reserved**
0	**AO_Waveform_Order_Clear** (Strobe)—This bitfield clears the configuration FIFO for the AO, which holds the order for the waveform channels. This bit should be set before writing the order channels for a new waveform generation. This bit should only be set as part of a configuration cycle.
Options: No Soft Copy	

Offset 0x48: AO_Trigger_Select_Register (W)

Absolute Address:
AO: 0x20448

Bits	Name
31..26	**AO_UPDATE_Source_Select**—This bitfield selects the AO_UPDATE source. When you set this bit to 0, the DAQ-STC3 is in the internal AO_UPDATE mode. When you select any other signal as the AO_UPDATE source, the DAQ-STC3 is in the external AO_UPDATE mode. The values for this bitfield are in *AO_UPDATE_Source_Select_t*.
25	**AO_UPDATE_Source_Polarity**—This bit selects the active edge of the AO_UPDATE source (the signal that is selected by AO_UPDATE_Source_Select). You must set this bit to 0 in the internal AO_UPDATE mode. The values for this bitfield are in *AO_Polarity_t*.
24	**Reserved**
23..18	**AO_START1_Select**—This bitfield selects the AO_START1 trigger. The values for this bitfield are in *AO_START1_Select_t*.
17	**AO_START1_Polarity**—This bit determines the polarity of AO_START1 trigger. The values for this bitfield are in *AO_Polarity_t*.
16	**AO_START1_Edge**—This bit enables edge detection of the AO_START1 trigger. This bit should normally be set to 1. Set this bit to 1 if AO_START1_Select is 0.
15..10	**AO_External_Gate_Select**—This bit enables and selects the external gate. The values for this bitfield are in *AO_External_Gate_Select_t*.
9	**AO_External_Gate_Polarity**—This bit selects the polarity of the analog output external gate signal. When set to 0, the gate is Active high (high enables operation). When set to 1, the gate is Active low (low enables operation).

Bits	Name
8	**AO_External_Gate_Enable**—Setting this bit to 1 enables external gating for the analog output group.
7..0	**Reserved**

Offset 0x4C: AO_Config_Bank (i) Register Array (W)

Absolute Address:
AO: 0x2044C

Array Offsets = 0x4C + (i × 1) [8 registers]

Bits	Name
7	**AO_Bipolar**—This bit sets AO_Bank_i in Bipolar mode. The values for this bitfield are in *AO_Bipolar_t*.
6	**AO_Update_Mode**—This bit determines the update mode for Bank_i. The values for this bitfield are in *AO_Update_Mode_t*.
5..3	**AO_Reference**—This bit configures the Reference for Bank_i.
2..0	**AO_Offset**—This bit configures the Offset for Bank_i.
Options: Initial Value = 0xBF	

Offset 0x58: AO_FIFO_Status_Register (R)

Absolute Address:
AO: 0x20458

Bits	Name
31..0	**AO_FIFO_Status**—This bitfield represents the amount of data that remains in the AO data FIFO.

Offset 0x58: AO_FIFO_Data_Register (W)

Absolute Address:
AO: 0x20458

Bits	Name
31..0	**AO_FIFO_Data** (Strobe)—This register is used to load data into the AO data FIFO.
Options: No Soft Copy	

Enumerated Types

AO_Bipolar_t

Value	Name
1	**Bipolar**

AO_External_Gate_Select_t

Value	Name	Value	Name
0	**Gate_Disabled**	25	**Gate_PFI14**
1	**Gate_PFI0**	26	**Gate_PFI15**
2	**Gate_PFI1**	27	**Gate_RTSI7**
3	**Gate_PFI2**	30	**Gate_Analog_Trigger**
4	**Gate_PFI3**	31	**Gate_Low**
5	**Gate_PFI4**	32	**Gate_G0_Out**
6	**Gate_PFI5**	33	**Gate_G1_Out**
7	**Gate_PFI6**	34	**Gate_G2_Out**
8	**Gate_PFI7**	35	**Gate_G3_Out**
9	**Gate_PFI8**	36	**Gate_G0_Gate**
10	**Gate_PFI9**	37	**Gate_G1_Gate**
11	**Gate_RTSI0**	38	**Gate_G2_Gate**
12	**Gate_RTSI1**	39	**Gate_G3_Gate**
13	**Gate_RSTI2**	40	**Gate_AI_Gate**
14	**Gate_RTSI3**	41	**Gate_DI_Gate**
15	**Gate_RTSI4**	44	**Gate_DO_Gate**
16	**Gate_RTSI5**	53	**Gate_IntTriggerA0**
17	**Gate_RSTI6**	54	**Gate_IntTriggerA1**
18	**Gate_PXIe_DStarA**	55	**Gate_IntTriggerA2**
19	**Gate_PXIe_DStarB**	56	**Gate_IntTriggerA3**
20	**Gate_Star_Trigger**	57	**Gate_IntTriggerA4**
21	**Gate_PFI10**	58	**Gate_IntTriggerA5**

Value	Name	Value	Name
22	**Gate_PFI11**	59	**Gate_IntTriggerA6**
23	**Gate_PFI12**	60	**Gate_IntTriggerA7**
24	**Gate_PFI13**		

AO_Polarity_t

Value	Name
0	**Rising_Edge**
1	**Falling_Edge**

AO_START1_Select_t

Value	Name	Value	Name
0	**Start1_Pulse**	25	**Start1_PFI14**
1	**Start1_PFI0**	26	**Start1_PFI15**
2	**Start1_PFI0**	27	**Start1_RTSI7**
3	**Start1_PFI0**	28	**Start1_PXIe_DStarA**
4	**Start1_PFI0**	29	**Start1_PXIe_DStarB**
5	**Start1_PFI0**	30	**Start1_Analog_Trigger**
6	**Start1_PFI0**	31	**Start1_Low**
7	**Start1_PFI0**	32	**Start1_G0_Out**
8	**Start1_PFI0**	33	**Start1_G0_Out**
9	**Start1_PFI0**	34	**Start1_G0_Out**
10	**Start1_PFI0**	35	**Start1_G0_Out**
11	**Start1_RTSI0**	36	**Start1_DIO_ChgDetect**
12	**Start1_RTSI1**	37	**Start1_DI_Start1**
13	**Start1_RTSI2**	38	**Start1_DI_Start2**
14	**Start1_RTSI3**	43	**Start1_DO_Start1**
15	**Start1_RTSI4**	53	**Start1_IntTriggerA0**
16	**Start1_RTSI5**	54	**Start1_IntTriggerA1**
17	**Start1_RTSI6**	55	**Start1_IntTriggerA2**

Value	Name	Value	Name
18	**Start1_AI_Start2**	56	**Start1_IntTriggerA3**
19	**Start1_AI_Start1**	57	**Start1_IntTriggerA4**
20	**Start1_Star_Trigger**	58	**Start1_IntTriggerA5**
21	**Start1_PFI10**	59	**Start1_IntTriggerA6**
22	**Start1_PFI11**	60	**Start1_IntTriggerA7**
23	**Start1_PFI12**	61	**Start1_FifoCondition**
24	**Start1_PFI13**		

AO_Update_Mode_t

Value	Name
0	**Immediate**—Configures the Bank for Immediate mode.
1	**Timed**—Configures the Bank for Timed mode.

AO_UPDATE_Source_Select_t

Value	Name	Value	Name
0	**Update_UI_TC**	26	**Update_PFI15**
1	**Update_PFI0**	27	**Update_RSTI7**
2	**Update_PFI1**	28	**Update_G2_Out**
3	**Update_PFI2**	29	**Update_G3_Out**
4	**Update_PFI3**	30	**Update_Analog_Trigger**
5	**Update_PFI4**	31	**Update_Low**
6	**Update_PFI5**	32	**Update_PXIe_DStarA**
7	**Update_PFI6**	33	**Update_PXIe_DStarB**
8	**Update_PFI7**	34	**Update_DIO_ChgDetect**
9	**Update_PFI8**	35	**Update_G0_SampleClk**
10	**Update_PFI9**	36	**Update_G1_SampleClk**
11	**Update_RTSI0**	37	**Update_G2_SampleClk**
12	**Update_RTSI1**	38	**Update_G3_SampleClk**
13	**Update_RTSI2**	39	**Update_AI_Convert**

Value	Name	Value	Name
14	**Update_RTSI3**	40	**Update_AI_Start**
15	**Update_RTSI4**	41	**Update_DI_Convert**
16	**Update_RTSI5**	44	**Update_DO_Update**
17	**Update_RTSI6**	53	**Update_IntTriggerA0**
18	**Update_G0_Out**	54	**Update_IntTriggerA1**
19	**Update_G1_Out**	55	**Update_IntTriggerA2**
20	**Update_Star_Trigger**	56	**Update_IntTriggerA3**
21	**Update_PFI10**	57	**Update_IntTriggerA4**
22	**Update_PFI11**	58	**Update_IntTriggerA5**
23	**Update_PFI12**	59	**Update_IntTriggerA6**
24	**Update_PFI13**	60	**Update_IntTriggerA7**
25	**Update_PFI14**	61	**Update_AutoUpdate**

15

Digital Output Registers

DO base address = 0x204AC

This chapter consists of *Digital Output Registers* and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

The digital output registers contain an OutTimer object, offset: 0x204E0. Refer to Chapter 18, *OutTimer Registers*, for more information.

List of Digital Output Registers

The digital output registers are as follows:

Offset 0x0: DO_FIFO_St_Register (R)

Offset 0x0: SCXI_DIO_Enable_Register (W)

Offset 0x4: Static_Digital_Output_Register (W)

Offset 0x8: DIO_Direction_Register (W)

Offset 0xC: CDO_FIFO_Data_Register (W)

Offset 0x10: DO_Mask_Enable_Register (W)

Offset 0x14: DO_Mode_Register (W)

Offset 0x18: DO_Trigger_Select_Register (W)

Offset 0x1C: DO_DirectDataRegister (W)

Offset 0x24: DO_WDT_SafeStateRegister (W)

Offset 0x28: DO_WDT_ModeSelect1_Register (W)

Offset 0x2C: DO_WDT_ModeSelect2_Register (W)

List of Enumerated Types

The enumerated types are as follows:

DO_Data_Lane_t

DO_DataWidth_t

DO_External_Gate_Select_t

DO_LineDir_t

DO_Polarity_t

DO_START1_Select_t

DO_TimedMask_t

DO_UPDATE_Source_Select_t

DO_WDT_Mode_t

Digital Output Registers

Offset 0x0: DO_FIFO_St_Register (R)

Absolute Address:
DO: 0x204AC

Bits	Name
31..16	**Reserved**
15..0	**DO_FIFO_FullCount**—This bitfield represents the number of samples that remain in the DO data FIFO.

Offset 0x0: SCXI_DIO_Enable_Register (W)

Absolute Address:
DO: 0x204AC

Bits	Name
7..3	**Reserved**
2	**SCXI_DIO_Intr_Enable**—Set this bit to enable the SCXI_Intr signal to be output onto DIO2.
1	**SCXI_DIO_D_A_Enable**—Set this bit to enable the SCXI_D_A signal to be output onto DIO1.
0	**SCXI_DIO_MOSI_Enable**—Set this bit to enable the SCXI_MOSI signal to be output onto DIO0.

Offset 0x4: Static_Digital_Output_Register (W)

Absolute Address:
DO: 0x204B0

Bits	Name
31..0	**DO_StaticValue**—Static digital output data. The data written in this register is output to the digital lines when: • The corresponding line is set as an output (kDIODirection_Pini = 1). • The DO_Mask_Enable Register is set for static output (CDO_Mask(i) = 0). • The SCXI_DIO_Enable_Register is set to output static DIO on lines 2:0. • The Watchdog Timer functionality is not enabled or its timer has not triggered.
Options: No Hardware Reset	

Offset 0x8: DIO_Direction_Register (W)

Absolute Address:
DO: 0x204B4

Bit “i” determines the direction of DIO line “i”.

Bits	Name
31	**kDIODirection_Pin31**—The values for this bitfield are in *DO_LineDir_t*.
30	**kDIODirection_Pin30**—The values for this bitfield are in *DO_LineDir_t*.
29	**kDIODirection_Pin29**—The values for this bitfield are in *DO_LineDir_t*.
28	**kDIODirection_Pin28**—The values for this bitfield are in *DO_LineDir_t*.
27	**kDIODirection_Pin27**—The values for this bitfield are in *DO_LineDir_t*.
26	**kDIODirection_Pin26**—The values for this bitfield are in *DO_LineDir_t*.
25	**kDIODirection_Pin25**—The values for this bitfield are in *DO_LineDir_t*.
24	**kDIODirection_Pin24**—The values for this bitfield are in *DO_LineDir_t*.
23	**kDIODirection_Pin23**—The values for this bitfield are in *DO_LineDir_t*.
22	**kDIODirection_Pin22**—The values for this bitfield are in *DO_LineDir_t*.
21	**kDIODirection_Pin21**—The values for this bitfield are in *DO_LineDir_t*.
20	**kDIODirection_Pin20**—The values for this bitfield are in *DO_LineDir_t*.
19	**kDIODirection_Pin19**—The values for this bitfield are in *DO_LineDir_t*.
18	**kDIODirection_Pin18**—The values for this bitfield are in *DO_LineDir_t*.
17	**kDIODirection_Pin17**—The values for this bitfield are in *DO_LineDir_t*.
16	**kDIODirection_Pin16**—The values for this bitfield are in *DO_LineDir_t*.
15	**kDIODirection_Pin15**—The values for this bitfield are in *DO_LineDir_t*.
14	**kDIODirection_Pin14**—The values for this bitfield are in *DO_LineDir_t*.
13	**kDIODirection_Pin13**—The values for this bitfield are in *DO_LineDir_t*.
12	**kDIODirection_Pin12**—The values for this bitfield are in *DO_LineDir_t*.
11	**kDIODirection_Pin11**—The values for this bitfield are in *DO_LineDir_t*.
10	**kDIODirection_Pin10**—The values for this bitfield are in *DO_LineDir_t*.
9	**kDIODirection_Pin9**—The values for this bitfield are in *DO_LineDir_t*.
8	**kDIODirection_Pin8**—The values for this bitfield are in *DO_LineDir_t*.
7	**kDIODirection_Pin7**—The values for this bitfield are in *DO_LineDir_t*.

Bits	Name
6	**kDIODirection_Pin6**—The values for this bitfield are in *DO_LineDir_t*.
5	**kDIODirection_Pin5**—The values for this bitfield are in *DO_LineDir_t*.
4	**kDIODirection_Pin4**—The values for this bitfield are in *DO_LineDir_t*.
3	**kDIODirection_Pin3**—The values for this bitfield are in *DO_LineDir_t*.
2	**kDIODirection_Pin2**—The values for this bitfield are in *DO_LineDir_t*.
1	**kDIODirection_Pin1**—The values for this bitfield are in *DO_LineDir_t*.
0	**kDIODirection_Pin0**—The values for this bitfield are in *DO_LineDir_t*.
Options: No Hardware Reset	

Offset 0xC: CDO_FIFO_Data_Register (W)

Absolute Address:
DO: 0x204B8

Bits	Name
31..0	**CDO_FIFO_Data**—This register is used to load data in to the CDO FIFO.
Options: No Soft Copy	

Offset 0x10: DO_Mask_Enable_Register (W)

Absolute Address:
DO: 0x204BC

Bits	Name
31..0	**CDO_Mask**—Sets on a per-bit basis whether the data routed to the output pins on P0 are from the CDO engine or from static DO. The values for this bitfield are in *DO_TimedMask_t*.

Offset 0x14: DO_Mode_Register (W)

Absolute Address:
DO: 0x204C0

Bits	Name
31..16	**Reserved**
15..14	**DO_DataWidth**—Use to determine the width of the data removed from the FIFO. When the FIFO width is programmable, the FIFO width changes with this bitfield. If the FIFO is of a set width, this bitfield, in combination with CDO_Data_Lane, determines how the data is smeared across the FIFO data lines. If 0, data saved is 8-bits. If 1, data saved is 16-bits. If 2, data saved is 32-bits. **Note:** The FIFO must be reset when this bitfield is changed. The values for this bitfield are in *DO_DataWidth_t*.
13..12	**DO_Data_Lane**—Use to determine which Byte, word lanes to use for output data. When the DO_DataWidth bitfield is 16 bits: • If 0, then CDIO_15_0. • If 1, then CDIO_31_16. When the DO_DataWidth bitfield is 8 bits: • If 0, then CDIO_7_0. • If 1, then CDIO_15_8. • If 2, then CDIO_23_16. • If 3, then CDIO_31_24. The values for this bitfield are in *DO_Data_Lane_t*.
11..0	**Reserved**
Options: Inital Value = 0x80	

Offset 0x18: DO_Trigger_Select_Register (W)

Absolute Address:
DO: 0x204C4

Bits	Name
31..26	**DO_UPDATE_Source_Select**—This bitfield selects the DO_UPDATE source. When you set this bit to 0, the DO Timer is in the internal DO_UPDATE mode. When you select any other signal as the DO_UPDATE source, the DO Timer is in the external DO_UPDATE mode. The values for this bitfield are in *DO_UPDATE_Source_Select_t*.
25	**DO_UPDATE_Source_Polarity**—This bit selects the active edge of the DO_UPDATE source (the signal that is selected by DO_UPDATE_Source_Select). You must set this bit to 0 in the internal DO_UPDATE mode. The values for this bitfield are in *DO_Polarity_t*.
24	**Reserved**
23..18	**DO_START1_Select**—This bitfield selects the DO_START1 trigger. The values for this bitfield are in *DO_START1_Select_t*.
17	**DO_START1_Polarity**—This bit determines the polarity of DO_START1 trigger. The values for this bitfield are in *DO_Polarity_t*.
16	**DO_START1_Edge**—This bit enables edge detection of the DO_START1 trigger. This bit should normally be set to 1. Set this bit to 1 if DO_START1_Select is Start1_Pulse.
15..10	**DO_External_Gate_Select**—This bit enables and selects the external gate. The values for this bitfield are in *DO_External_Gate_Select_t*.
9	**DO_External_Gate_Polarity**—This bit selects the polarity of the DO external gate signal. When set to 0 the gate is Active high (high enables operation). When set to 1, the gate is Active low (low enables operation).
8	**DO_External_Gate_Enable**—Setting this bit to 1 enables external gating for the digital output group.
7..0	**Reserved**

Offset 0x1C: DO_DirectDataRegister (W)

Absolute Address:
DO: 0x204C8

Bits	Name
31..0	**CDO_Direct_data**—This register is used to load data into the CDO Engine when the FIFO is disabled.
Options: No Soft Copy	

Offset 0x24: DO_WDT_SafeStateRegister (W)

Absolute Address:
DO: 0x204D0

Bits	Name
31..0	**DO_WDT_SafeStateValue**—This is the safe value that the DIO lines will go to if the Watchdog Timer is enabled, and the DIO lines have the safe state enabled (through the DO_WDT_ModeSelect[i]_Register).

Offset 0x28: DO_WDT_ModeSelect1_Register (W)

Absolute Address:
DO: 0x204D4

Bits	Name
31..30	**DO_WDT_ModeD15**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
29..28	**DO_WDT_ModeD14**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
27..26	**DO_WDT_ModeD13**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
25..24	**DO_WDT_ModeD12**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
23..22	**DO_WDT_ModeD11**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
21..20	**DO_WDT_ModeD10**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
19..18	**DO_WDT_ModeD9**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.

Bits	Name
17..16	**DO_WDT_ModeD8**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
15..14	**DO_WDT_ModeD7**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
13..12	**DO_WDT_ModeD6**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
11..10	**DO_WDT_ModeD5**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
9..8	**DO_WDT_ModeD4**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
7..6	**DO_WDT_ModeD3**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
5..4	**DO_WDT_ModeD2**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
3..2	**DO_WDT_ModeD1**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
1..0	**DO_WDT_ModeD0**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.

Offset 0x2C: DO_WDT_ModeSelect2_Register (W)

Absolute Address:
DO: 0x204D8

Bits	Name
31..30	**DO_WDT_ModeD31**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
29..28	**DO_WDT_ModeD30**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
27..26	**DO_WDT_ModeD29**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
25..24	**DO_WDT_ModeD28**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
23..22	**DO_WDT_ModeD27**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
21..20	**DO_WDT_ModeD26**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
19..18	**DO_WDT_ModeD25**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
17..16	**DO_WDT_ModeD24**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
15..14	**DO_WDT_ModeD23**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
13..12	**DO_WDT_ModeD22**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.

Bits	Name
11..10	**DO_WDT_ModeD21**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
9..8	**DO_WDT_ModeD20**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
7..6	**DO_WDT_ModeD19**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
5..4	**DO_WDT_ModeD18**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
3..2	**DO_WDT_ModeD17**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.
1..0	**DO_WDT_ModeD16**—This bitfield determines what action will be taken by the corresponding DO line in the event of a Watchdog Timer event. The values for this bitfield are in *DO_WDT_Mode_t*.

Enumerated Types

DO_Data_Lane_t

Value	Name
0	**DO_DataLane0**
1	**DO_DataLane1**
2	**DO_DataLane2**
3	**DO_DataLane3**

DO_DataWidth_t

Value	Name
0	**DO_OneByte**
1	**DO_TwoBytes**
2	**DO_FourBytes**

DO_External_Gate_Select_t

Value	Name	Value	Name
0	**Gate_Disabled**	25	**Gate_PFI14**
1	**Gate_PFI0**	26	**Gate_PFI15**
2	**Gate_PFI1**	27	**Gate_RTSI7**
3	**Gate_PFI2**	30	**Gate_Analog_Trigger**
4	**Gate_PFI3**	31	**Gate_Low**
5	**Gate_PFI4**	32	**Gate_G0_Out**
6	**Gate_PFI5**	33	**Gate_G1_Out**
7	**Gate_PFI6**	34	**Gate_G2_Out**
8	**Gate_PFI7**	35	**Gate_G3_Out**
9	**Gate_PFI8**	36	**Gate_G0_Gate**
10	**Gate_PFI9**	37	**Gate_G1_Gate**
11	**Gate_RTSI0**	38	**Gate_G2_Gate**
12	**Gate_RTSI1**	39	**Gate_G3_Gate**

Value	Name	Value	Name
13	**Gate_RTSI2**	40	**Gate_AI_Gate**
14	**Gate_RTSI3**	41	**Gate_DI_Gate**
15	**Gate_RTSI4**	44	**Gate_AO_Gate**
16	**Gate_RTSI5**	53	**Gate_IntTriggerA0**
17	**Gate_RTSI6**	54	**Gate_IntTriggerA1**
18	**Gate_PXIe_DStarA**	55	**Gate_IntTriggerA2**
19	**Gate_PXIe_DStarB**	56	**Gate_IntTriggerA3**
20	**Gate_Star_Trigger**	57	**Gate_IntTriggerA4**
21	**Gate_PFI10**	58	**Gate_IntTriggerA5**
22	**Gate_PFI11**	59	**Gate_IntTriggerA6**
23	**Gate_PFI12**	60	**Gate_IntTriggerA7**
24	**Gate_PFI13**		

DO_LineDir_t

Value	Name
0	**Input**
1	**Output**

DO_Polarity_t

Value	Name
0	**Rising_Edge**
1	**Falling_Edge**

DO_START1_Select_t

Value	Name	Value	Name
0	**Start1_Pulse**	25	**Start1_PFI14**
1	**Start1_PFI0**	26	**Start1_PFI15**
2	**Start1_PFI1**	27	**Start1_RTSI7**
3	**Start1_PFI2**	28	**Start1_PXIe_DStarA**
4	**Start1_PFI3**	29	**Start1_PXIe_DStarB**
5	**Start1_PFI4**	30	**Start1_Analog_Trigger**
6	**Start1_PFI5**	31	**Start1_Low**
7	**Start1_PFI6**	32	**Start1_G0_Out**
8	**Start1_PFI7**	33	**Start1_G1_Out**
9	**Start1_PFI8**	34	**Start1_G2_Out**
10	**Start1_PFI9**	35	**Start1_G3_Out**
11	**Start1_RTSI0**	36	**Start1_DIO_ChgDetect**
12	**Start1_RTSI1**	37	**Start1_DI_Start1**
13	**Start1_RTSI2**	38	**Start1_DI_Start2**
14	**Start1_RTSI3**	43	**Start1_AO_Start1**
15	**Start1_RTSI4**	53	**Start1_IntTriggerA0**
16	**Start1_RTSI5**	54	**Start1_IntTriggerA1**
17	**Start1_RTSI6**	55	**Start1_IntTriggerA2**
18	**Start1_AI_Start2**	56	**Start1_IntTriggerA3**
19	**Start1_AI_Start1**	57	**Start1_IntTriggerA4**
20	**Start1_Star_Trigger**	58	**Start1_IntTriggerA5**
21	**Start1_PFI10**	59	**Start1_IntTriggerA6**
22	**Start1_PFI11**	60	**Start1_IntTriggerA7**
23	**Start1_PFI12**	61	**Start1_FifoCondition**
24	**Start1_PFI13**		

DO_TimedMask_t

Value	Name
0	**StaticDIO**
1	**CDOEngine**

DO_UPDATE_Source_Select_t

Value	Name	Value	Name
0	**Update_UI_TC**	26	**Update_PFI15**
1	**Update_PFI0**	27	**Update_RTSI7**
2	**Update_PFI1**	28	**Update_G2_Out**
3	**Update_PFI2**	29	**Update_G3_Out**
4	**Update_PFI3**	30	**Update_Analog_Trigger**
5	**Update_PFI4**	31	**Update_Low**
6	**Update_PFI5**	32	**Update_PXIe_DStarA**
7	**Update_PFI6**	33	**Update_PXIe_DStarB**
8	**Update_PFI7**	34	**Update_DIO_ChgDetect**
9	**Update_PFI8**	35	**Update_G0_SampleClk**
10	**Update_PFI9**	36	**Update_G1_SampleClk**
11	**Update_RTSI0**	37	**Update_G2_SampleClk**
12	**Update_RTSI1**	38	**Update_G3_SampleClk**
13	**Update_RTSI2**	39	**Update_AI_Convert**
14	**Update_RTSI3**	40	**Update_AI_Start**
15	**Update_RTSI4**	41	**Update_DI_Convert**
16	**Update_RTSI5**	44	**Update_AO_Update**
17	**Update_RTSI6**	53	**Update_IntTriggerA0**
18	**Update_G0_Out**	54	**Update_IntTriggerA1**
19	**Update_G1_Out**	55	**Update_IntTriggerA2**
20	**Update_Star_Trigger**	56	**Update_IntTriggerA3**
21	**Update_PFI10**	57	**Update_IntTriggerA4**
22	**Update_PFI11**	58	**Update_IntTriggerA5**

Value	Name	Value	Name
23	**Update_PFI12**	59	**Update_IntTriggerA6**
24	**Update_PFI13**	60	**Update_IntTriggerA7**
25	**Update_PFI14**	61	**Update_FreqOut**

DO_WDT_Mode_t

Value	Name
0	**WDT_Disabled**—When set to Disabled, there is no action on a Watchdog Timer event. This is the default.
1	**WDT_Freeze**—When set to Freeze, the DO subsystem freezes all the routing registers, the static DO register, and the direction register. This has the effect of enforcing the current state even if software does “bad” writes to the hardware (except when the writes hit the reset bits). This freezing of the registers happens on any other mode except for Disabled. This state does not impose a “safe value” on the line. Note that if the line was routed to something other than static DO (for example, Timed DO), the value would change. The freeze does not stop any subsystems, only preserves the routing and static DO values. Software must acknowledge the Watchdog Timer before the hardware accepts any changes to configuration, including this bit.
2	**WDT_Tristate**—When set to Tristate, the DO line goes to High Impedance whenever there is a Watchdog Timer event. It also freezes all the routing registers. Software must acknowledge the Watchdog Timer before the hardware accepts any changes to configuration, including this bit.
3	**WDT_SafeValue**—When set to SafeValue, the DO line goes to the predefined safe value whenever there is a Watchdog Timer event. It also freezes all the routing registers. Software must acknowledge the Watchdog Timer before the hardware accepts any changes to configuration, including this bit.

16

Digital Input Registers

DI base address = 0x20530

This chapter consists of *Digital Input Registers*, *Change Detect Registers* and *Enumerated Types*.

The digital input registers contain an InTimer object, offset: 0x20560. Refer to Chapter 17, *InTimer Registers*, for more information.

List of Digital Input Registers

The digital input registers are as follows:

Offset 0x0: Static_Digital_Input_Register (R)

Offset 0x00: DI_DMA_Select_Register (W)

Offset 0x04: DI_FIFO_St_Register (R)

Offset 0x04: DI_Mode_Register (W)

Offset 0x08: DI_FIFO_Data_Register (R)

Offset 0x08: DI_FIFO_Data_Register16 (R)

Offset 0x08: DI_FIFO_Data_Register8 (R)

Offset 0x08: DI_Mask_Enable_Register (W)

Offset 0x0C: DI_Trigger_Select_Register (W)

Offset 0x1C: DI_FilterRegisterLo (W)

Offset 0x20: DI_FilterRegisterHi (W)

The change detect registers are as follows:

Offset 0x10: DI_ChangeDetectStatusRegister (R)

Offset 0x10: DI_ChangeIrqRE_Register (W)

Offset 0x14: DI_ChangeDetectLatchedDI_Register (R)

Offset 0x14: DI_ChangeIrqFE_Register (W)

Offset 0x18: DI_ChangeDetectLatchedPFI_Register (R)

Offset 0x18: PFI_ChangeIrq_Register (W)

Offset 0x24: DI_ChangeDetectIRQ_Register (W)

List of Enumerated Types

The enumerated types are as follows:

DI_Data_Lane_t

DI_DataWidth_t

DI_External_Gate_Select_t

DI_Filter_Select_t

DI_FilterMode_t

DI_Mask_t

DI_Polarity_t

DI_START1_Select_t

DI_START2_Select_t

DI_StartConvertSelMux_t

Digital Input Registers

Offset 0x0: Static_Digital_Input_Register (R)

Absolute Address:
DI: 0x20530

Bits	Name
31..0	**DI_StaticValue**—This register reflects the state of the DI lines. Software should mask lines not being used for static digital input.

Offset 0x00: DI_DMA_Select_Register (W)

Absolute Address:
DI: 0x20530

Bits	Name
7	**DI_DoneNotificationEnable**—This bit enables the generation of the done notification on the input stream. This notification happens after the last poins is written after a LastSC_TC event. After this notification occurs, no more data can be written until a FIFO reset happens.
6..0	**Reserved**

Offset 0x04: DI_FIFO_St_Register (R)

Absolute Address:
DI: 0x20534

Bits	Name
31..16	**Reserved**
15..0	**CDI_FIFO_FullCount**—Indicates the number of samples available to read in the CDI FIFO.

Offset 0x04: DI_Mode_Register (W)

Absolute Address:
DI: 0x20534

Bits	Name
31..16	**Reserved**
15..14	**DI_DataWidth**—Use to determine the width of the data to be saved to the FIFO. When the FIFO width is programmable, the FIFO width changes with this bitfield. If the FIFO is of a set width, this bitfield, in combination with CDI_Data_Lane, determines how the data is smeared across the FIFO data lines. If 0, then data saved is 8-bits. If 1, then 16-bits. If 2 or more, then 32-bits. **Note**: The FIFO must be reset when this bitfield is changed. The values for this bitfield are in *DI_DataWidth_t*.
13..12	**DI_Data_Lane**—Use to determine which Byte or word lanes to read data from. When the DI_DataWidth bitfield is 16-bits: • If 0, then CDIO_15_0. • If 1, then CDIO_31_16. When the DI_DataWidth bitfield is 8-bits: • If 0, then CDIO_7_0. • If 1, then CDIO_15_8. • If 2, then CDIO_23_16. • If 3, then CDIO_31_24. The values for this bitfield are in *DI_Data_Lane_t*.
11	**DI_DigitalFiltersMode**—If in correlated mode, the filters keep the output correlated as long as the skew in the input lines is less than the filter setting and the valid input lines remain stable for two times the filter setting before changing again. If disabled, the digital filters in DI will behave like the NI 6509. Input lines need to be stable for twice the filter setting for the change to be guaranteed in the output; this will have better latency and jitter specs, but will not guarantee correlation between lines. The values for this bitfield are in *DI_FilterMode_t*.
10..8	**Reserved**
7..0	**Reserved**

Offset 0x08: DI_FIFO_Data_Register (R)

Absolute Address:
DI: 0x20538

Bits	Name
31..0	**CDI_FIFO_Data**—This register is used to read data from the DI FIFO 32 bits at a time. This register should be used any time that DI_DataWidth is set to 32-bits.

Offset 0x08: DI_FIFO_Data_Register16 (R)

Absolute Address:
DI: 0x20538

Bits	Name
15..0	**CDI_FIFO_Data16**—This register is used to read data from the DI FIFO 16 bits at a time. This register should be used any time that DI_DataWidth is set to 16-bits.

Offset 0x08: DI_FIFO_Data_Register8 (R)

Absolute Address:
DI: 0x20538

Bits	Name
7..0	**CDI_FIFO_Data8**—This register is used to read data from the DI FIFO eight bits at a time. This register should be used any time that DI_DataWidth is set to 8-bits.

Offset 0x08: DI_Mask_Enable_Register (W)

Absolute Address:
DI: 0x20538

Bits	Name
31..0	**CDI_Mask**—Sets on a per-bit basis whether the data is written to the FIFO, or the bit is masked and a zero is written to the FIFO. The values for this bitfield are in *DI_Mask_t*.

Offset 0x0C: DI_Trigger_Select_Register (W)

Absolute Address:
DI: 0x2053C

Bits	Name
31	**DI_CONVERT_Source_Polarity**—This bit selects the active edge of the DI_CONVERT source signal. The values for this bitfield are in *DI_Polarity_t*.
30	**Reserved**

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>29..24</td><td colspan="2">DI_CONVERT_Source_Select—Selects the DI_CONVERT source.

When you set this bitfield to 0, the DI Timer is in internal DI_CONVERT mode. When you select any other signal as the DI_CONVERT source, the DI Timer is in external DI_CONVERT mode.

The values for this bitfield are in DI_StartConvertSelMux_t.</td></tr>
<tr><td>23</td><td colspan="2">DI_External_Gate_Polarity—This bit selects the polarity of the external gate signal.

The values for this bitfield are in DI_Polarity_t.</td></tr>
<tr><td>22</td><td colspan="2">Reserved</td></tr>
<tr><td>21..16</td><td colspan="2">DI_External_Gate_Select—This bitfield enables and selects the external gate. You can use the external gate to pause a digital input operation in progress.

The values for this bitfield are in DI_External_Gate_Select_t.</td></tr>
<tr><td>15</td><td colspan="2">DI_START2_Polarity—This bit determines the polarity of DI_START2 trigger.

Note: Set this bit to 0 if DI_START2_Select is set to 0 (DI_START2_Pulse).

The values for this bitfield are in DI_Polarity_t.</td></tr>
<tr><td rowspan="4">14</td><td colspan="2">DI_START2_Edge—This bit enables edge detection of the DI_START2 trigger.

Note: Set this bit to 0 if DI_START2_Select is set to 0 (DI_START2_Pulse).</td></tr>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0</td><td>Disabled (level-sensitive trigger)</td></tr>
<tr><td>1</td><td>Enabled (edge-sensitive trigger)</td></tr>
<tr><td>13..8</td><td colspan="2">DI_START2_Select—This bitfield selects the DI_START2 trigger.

The values for this bitfield are in DI_START2_Select_t.</td></tr>
<tr><td>7</td><td colspan="2">DI_START1_Polarity—This bit determines the polarity of the DI_START1 trigger.

Note: Set this bit to 0 if DI_START1_Select is set to 0 (DI_START1_Pulse).

The values for this bitfield are in DI_Polarity_t.</td></tr>
<tr><td rowspan="4">6</td><td colspan="2">DI_START1_Edge—This bit enables edge-sensitive detection of the DI_START1 trigger:

Note: Set this bit to 0 if DI_START1_Select is set to 0 (DI_START1_Pulse).</td></tr>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0</td><td>Disabled (level-sensitive trigger)</td></tr>
<tr><td>1</td><td>Enabled (edge-sensitive trigger)</td></tr>
<tr><td>5..0</td><td colspan="2">DI_START1_Select—This bitfield selects the DI_START1 trigger.

The values for this bitfield are in DI_START1_Select_t.</td></tr>
</table>

Offset 0x1C: DI_FilterRegisterLo (W)

Absolute Address:
DI: 0x2054C

Bits	Name
31..30	**DI15_Filter_Select**—DI line 15 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
29..28	**DI14_Filter_Select**—DI line 14 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
27..26	**DI13_Filter_Select**—DI line 13 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
25..24	**DI12_Filter_Select**—DI line 12 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
23..22	**DI11_Filter_Select**—DI line 11 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
21..20	**DI10_Filter_Select**—DI line 10 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
19..18	**DI9_Filter_Select**—DI line 9 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
17..16	**DI8_Filter_Select**—DI line 8 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
15..14	**DI7_Filter_Select**—DI line 7 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
13..12	**DI6_Filter_Select**—DI line 6 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
11..10	**DI5_Filter_Select**—DI line 5 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
9..8	**DI4_Filter_Select**—DI line 4 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
7..6	**DI3_Filter_Select**—DI line 3 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.

Bits	Name
5..4	**DI2_Filter_Select**—DI line 2 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
3..2	**DI1_Filter_Select**—DI line 1 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
1..0	**DI0_Filter_Select**—DI line 0 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.

Offset 0x20: DI_FilterRegisterHi (W)

Absolute Address:
DI: 0x20550

Bits	Name
31..30	**DI31_Filter_Select**—DI line 31 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
29..28	**DI30_Filter_Select**—DI line 30 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
27..26	**DI29_Filter_Select**—DI line 29 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
25..24	**DI28_Filter_Select**—DI line 28 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
23..22	**DI27_Filter_Select**—DI line 27 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
21..20	**DI26_Filter_Select**—DI line 26 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
19..18	**DI25_Filter_Select**—DI line 25 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
17..16	**DI24_Filter_Select**—DI line 24 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
15..14	**DI23_Filter_Select**—DI line 23 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
13..12	**DI22_Filter_Select**—DI line 22 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.

Bits	Name
11..10	**DI21_Filter_Select**—DI line 21 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
9..8	**DI20_Filter_Select**—DI line 20 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
7..6	**DI19_Filter_Select**—DI line 19 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
5..4	**DI18_Filter_Select**—DI line 18 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
3..2	**DI17_Filter_Select**—DI line 17 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.
1..0	**DI16_Filter_Select**—DI line 16 Filter Selection. The values for this bitfield are in *DI_Filter_Select_t*.

Change Detect Registers

Offset 0x10: DI_ChangeDetectStatusRegister (R)

Absolute Address:
DI: 0x20540

Bits	Name
31..2	**Reserved**
1	**DI_ChangeDetectError**—This bit reflects the state of the Change Detect Error Circuit. When the bit is on, the change detect circuit has issued two consecutive events without an acknowledge from software (with the ChangeDetectIRQ_Acknowledge). Clear this bit with the ChangeDetectErrorIRQ_Acknowledge bitfield.
0	**DI_ChangeDetectStatus**—This bit reflects the state of the Change Detect Circuit. When the bit is on, the change detect circuit has detected one of the registered events. Clear this bit with the ChangeDetectIRQ_Acknowledge bitfield.

Offset 0x10: DI_ChangeIrqRE_Register (W)

Absolute Address:
DI: 0x20540

Bits	Name
31..0	**DI_ChangeIrqRE**—Set to check for rising edges on corresponding DI line.

Offset 0x14: DI_ChangeDetectLatchedDI_Register (R)

Absolute Address:
DI: 0x20544

Bits	Name
31..0	**DI_ChangeDetectLatchedDI**—This register reflects the state of the DI lines at the time the last masked change was detected including the change that caused it. Software should mask lines not being used for static digital input. **Note**: This register is written at all the events detected by the change detect circuit. SW must check the Change Detect Error bit to make sure the data being read is valid and coherent (for the case of reading DI and PFI registers).

Offset 0x14: DI_ChangeIrqFE_Register (W)

Absolute Address:
DI: 0x20544

Bits	Name
31..0	**DI_ChangeIrqFE**—Set to check for falling edges on corresponding DI line.

Offset 0x18: DI_ChangeDetectLatchedPFI_Register (R)

Absolute Address:
DI: 0x20548

Bits	Name
15..0	**DI_ChangeDetectLatchedPFI**—This register reflects the state of the PFI lines at the time the last masked change was detected including the change that caused it. Software should mask lines not being used for static digital input. **Note**: This register is written at all the events detected by the change detect circuit. SW must check the Change Detect Error bit to make sure the data being read is valid and coherent (for the case of reading DI and PFI registers).

Offset 0x18: PFI_ChangeIrq_Register (W)

Absolute Address:
DI: 0x20548

Bits	Name
31..16	**PFI_ChangeIrqFE**—Set to check for falling edges on corresponding PFI line.
15..0	**PFI_ChangeIrqRE**—Set to check for rising edges on corresponding PFI line.

Offset 0x24: DI_ChangeDetectIRQ_Register (W)

Absolute Address:
DI: 0x20554

Bits	Name
31..8	**Reserved**
7	**ChangeDetectErrorIRQ_Enable** (Strobe)—Enables the Change Detect Error Interrupt. This interrupt fires when two Change Detect Interrupts are generated without an acknowledge from software.
6	**ChangeDetectErrorIRQ_Disable** (Strobe)—Disables the Change Detect Interrupt.
5	**ChangeDetectIRQ_Enable** (Strobe)—Enables the Change Detect Interrupt. This interrupt fires any time the change detect circuit detects one of the registered events.
4	**ChangeDetectIRQ_Disable** (Strobe)—Disables the Change Detect Interrupt.
3..2	**Reserved**
1	**ChangeDetectErrorIRQ_Acknowledge** (Strobe)—Acknowledges the Change Detect Error Interrupt and clears the status bit.
0	**ChangeDetectIRQ_Acknowledge** (Strobe)—Acknowledges the Change Detect Interrupt and clears the status bit.
Options: No Soft Copy	

Enumerated Types

DI_Data_Lane_t

Value	Name
0	**DI_DataLane0**
1	**DI_DataLane1**
2	**DI_DataLane2**
3	**DI_DataLane3**

DI_DataWidth_t

Value	Name
0	**DI_OneByte**
1	**DI_TwoBytes**
2	**DI_FourBytes**

DI_External_Gate_Select_t

Value	Name	Value	Name
0	**Gate_Disabled**	25	**Gate_PFI14**
1	**Gate_PFI0**	26	**Gate_PFI15**
2	**Gate_PFI1**	27	**Gate_RTSI7**
3	**Gate_PFI2**	30	**Gate_Analog_Trigger**
4	**Gate_PFI3**	31	**Gate_Low**
5	**Gate_PFI4**	32	**Gate_G0_Out**
6	**Gate_PFI5**	33	**Gate_G1_Out**
7	**Gate_PFI6**	34	**Gate_G2_Out**
8	**Gate_PFI7**	35	**Gate_G3_Out**
9	**Gate_PFI8**	36	**Gate_G0_Gate**
10	**Gate_PFI9**	37	**Gate_G1_Gate**
11	**Gate_RTSI0**	38	**Gate_G2_Gate**
12	**Gate_RTSI1**	39	**Gate_G3_Gate**
13	**Gate_RTSI2**	40	**Gate_AI_Gate**
14	**Gate_RTSI3**	43	**Gate_AO_Gate**
15	**Gate_RTSI4**	44	**Gate_DO_Gate**
16	**Gate_RTSI5**	53	**Gate_IntTriggerA0**
17	**Gate_RTSI6**	54	**Gate_IntTriggerA1**
18	**Gate_PXIe_DStarA**	55	**Gate_IntTriggerA2**
19	**Gate_PXIe_DStarB**	56	**Gate_IntTriggerA3**
20	**Gate_Star_Trigger**	57	**Gate_IntTriggerA4**
21	**Gate_PFI10**	58	**Gate_IntTriggerA5**
22	**Gate_PFI11**	59	**Gate_IntTriggerA6**
23	**Gate_PFI12**	60	**Gate_IntTriggerA7**
24	**Gate_PFI13**		

DI_Filter_Select_t

Value	Name
0	**No_Filter**—No filter.
1	**Small_Filter**—Small Filter rejects less than 100 ns : Accepts greater than 200 ns : Delays 150 ns : Adds 100 ns Jitter (In Correlated Mode—Delays 250 ns : Adds 200 ns Jitter).
2	**Medium_Filter**—Medium Filter rejects less than 6.4 µs : Accepts greater than 12.8 us : Delays 9.6 us : Adds 6.4 us Jitter (In Correlated Mode—Delays 16 us : Adds 12.8 µs Jitter).
3	**Large_Filter**—Large Filter rejects less than 2.54 ms : Accepts greater than 5.1 ms : Delays 3.8 ms : Adds 2.54 ms Jitter (In Correlated Mode—Delays 6.4 ms : Adds 5.1 ms Jitter).

DI_FilterMode_t

Value	Name
0	**6509Mode**—Filters behave like the NI 6509 filter.
1	**CorrelatedMode**—Attempts to keep correlation after the filters.

DI_Mask_t

Value	Name
0	**ForceZero**—0 in FIFO.
1	**KeepData**—P0(i) in FIFO.

DI_Polarity_t

Value	Name
0	**Active_High_Or_Rising_Edge**
1	**Active_Low_Or_Falling_Edge**

DI_START1_Select_t

Value	Name	Value	Name
0	**Start1_SW_Pulse**	23	**Start1_PFI12**
1	**Start1_PFI0**	28	**Start1_DIO_ChgDetect**
2	**Start1_PFI1**	30	**Start1_Analog_Trigger**
3	**Start1_PFI2**	31	**Start1_Low**
4	**Start1_PFI3**	24	**Start1_PFI13**
5	**Start1_PFI4**	25	**Start1_PFI14**
6	**Start1_PFI5**	26	**Start1_PFI15**
7	**Start1_PFI6**	27	**Start1_RTSI7**
8	**Start1_PFI7**	36	**Start1_G0_Out**
9	**Start1_PFI8**	37	**Start1_G1_Out**
10	**Start1_PFI9**	38	**Start1_G2_Out**
11	**Start1_RTSI0**	39	**Start1_G3_Out**
12	**Start1_RTSI1**	40	**Start1_AI_Start1**
13	**Start1_RTSI2**	43	**Start1_AO_Start1**
14	**Start1_RTSI3**	44	**Start1_DO_Start1**
15	**Start1_RTSI4**	53	**Start1_IntTriggerA0**
16	**Start1_RTSI5**	54	**Start1_IntTriggerA1**
17	**Start1_RTSI6**	55	**Start1_IntTriggerA2**
18	**Start1_PXIe_DStarA**	56	**Start1_IntTriggerA3**
19	**Start1_PXIe_DStarB**	57	**Start1_IntTriggerA4**
20	**Start1_Star_Trigger**	58	**Start1_IntTriggerA5**
21	**Start1_PFI10**	59	**Start1_IntTriggerA6**
22	**Start1_PFI11**	60	**Start1_IntTriggerA7**

DI_START2_Select_t

Value	Name	Value	Name
0	**Start2_SW_Pulse**	23	**Start2_PFI12**
1	**Start2_PFI0**	24	**Start2_PFI13**
2	**Start2_PFI1**	25	**Start2_PFI14**
3	**Start2_PFI2**	26	**Start2_PFI15**
4	**Start2_PFI3**	27	**Start2_RTSI7**
5	**Start2_PFI4**	28	**Start2_DIO_ChgDetect**
6	**Start2_PFI5**	30	**Start2_Analog_Trigger**
7	**Start2_PFI6**	31	**Start2_Low**
8	**Start2_PFI7**	36	**Start2_G0_Out**
9	**Start2_PFI8**	37	**Start2_G1_Out**
10	**Start2_PFI9**	38	**Start2_G2_Out**
11	**Start2_RTSI0**	39	**Start2_G3_Out**
12	**Start2_RTSI1**	40	**Start2_AI_Start2**
13	**Start2_RTSI2**	43	**Start2_AO_Start1**
14	**Start2_RTSI3**	44	**Start2_DO_Start1**
15	**Start2_RTSI4**	53	**Start2_IntTriggerA0**
16	**Start2_RTSI5**	54	**Start2_IntTriggerA1**
17	**Start2_RTSI6**	55	**Start2_IntTriggerA2**
18	**Start2_PXIe_DStarA**	56	**Start2_IntTriggerA3**
19	**Start2_PXIe_DStarB**	57	**Start2_IntTriggerA4**
20	**Start2_Star_Trigger**	58	**Start2_IntTriggerA5**
21	**Start2_PFI10**	59	**Start2_IntTriggerA6**
22	**Start2_PFI11**	60	**Start2_IntTriggerA7**

DI_StartConvertSelMux_t

Value	Name	Value	Name
0	**SampleClk_Internal**	26	**SampleClk_PFI15**
1	**SampleClk_PFI0**	27	**SampleClk_RTSI7**
2	**SampleClk_PFI1**	28	**SampleClk_G1_Out**
3	**SampleClk_PFI2**	29	**SampleClk_AI_Start**
4	**SampleClk_PFI3**	30	**SampleClk_Atrig**
5	**SampleClk_PFI4**	31	**SampleClk_Low**
6	**SampleClk_PFI5**	32	**SampleClk_PXIe_DStarA**
7	**SampleClk_PFI6**	33	**SampleClk_PXIe_DStarB**
8	**SampleClk_PFI7**	34	**SampleClk_G2_Out**
9	**SampleClk_PFI8**	35	**SampleClk_G3_Out**
10	**SampleClk_PFI9**	36	**SampleClk_G0_SampleClk**
11	**SampleClk_RTSI0**	37	**SampleClk_G1_SampleClk**
12	**SampleClk_RTSI1**	38	**SampleClk_G2_SampleClk**
13	**SampleClk_RTSI2**	39	**SampleClk_G3_SampleClk**
14	**SampleClk_RTSI3**	40	**SampleClk_AI_Convert**
15	**SampleClk_RTSI4**	43	**SampleClk_AO_Update**
16	**SampleClk_RTSI5**	44	**SampleClk_DO_Update**
17	**SampleClk_RTSI6**	53	**SampleClk_IntTriggerA0**
18	**SampleClk_DIO_ChgDetect**	54	**SampleClk_IntTriggerA1**
19	**SampleClk_G0_Out**	55	**SampleClk_IntTriggerA2**
20	**SampleClk_Star_Trigger**	56	**SampleClk_IntTriggerA3**
21	**SampleClk_PFI10**	57	**SampleClk_IntTriggerA4**
22	**SampleClk_PFI11**	58	**SampleClk_IntTriggerA5**
23	**SampleClk_PFI12**	59	**SampleClk_IntTriggerA6**
24	**SampleClk_PFI13**	60	**SampleClk_IntTriggerA7**
25	**SampleClk_PFI14**	61	**SampleClk_FreqOut**

17

InTimer Registers

AI base address = 0x202B0
DI base address = 0x20560

This chapter consists of *InTimer Registers* and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

List of InTimer Registers

The InTimer registers are as follows:

List of Enumerated Types

The enumerated types are as follows:

InTimer_Disabled_Enabled_t

InTimer_Disarmed_Armed_t

InTimer_Error_t

InTimer_External_Gate_Mode_t

InTimer_External_MUX_Present_t

InTimer_EXTMUX_CLK_Pulse_Width_t

InTimer_FIFO_Empty_St_t

InTimer_FIFO_Full_St_t

InTimer_FIFO_Mode_t

InTimer_FIFO_Request_St_t

InTimer_Idle_Counting_t

InTimer_Load_A_Load_B_t

InTimer_Polarity_t

InTimer_Pre_Trigger_t

InTimer_SC_Q_St_t

InTimer_SC_Reload_Mode_t

InTimer_SI_Load_Switch_Pending_t

InTimer_SI_Reload_Mode_t

InTimer_SI_Source_Polarity_t

InTimer_SI_Source_Select_t

InTimer_SI2_Reload_Mode_t

InTimer_SI2_Source_Select_t

InTimer_START_Output_Select_t

InTimer_Start_Trigger_Length_t

InTimer_Start1_Export_Mode_t

InTimer_Start2_Export_Mode_t

InTimer_SyncMode_t

InTimer Registers

Offset 0x00: Status_1_Register (R)

Absolute Addresses:
AI: 0x202B0
DI: 0x20560

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>31</td><td colspan="2">SC_PreWaitCountTC_ErrorSt—Indicates that the SC_TC rolled over at least twice without an acknowledge while waiting for the START2 trigger in a pretrigger acquisition. Clear this bit with the SC_PreWaitCountTC_ErrorAck.</td></tr>
<tr><td>30</td><td colspan="2">Last_Shiftin_St—This bit indicates that the data corresponding to the last convert after the last SC_TC has been written to the FIFO. It is cleared by setting AI_SC_TC_Interrupt_Ack to 1.</td></tr>
<tr><td rowspan="4">29</td><td colspan="2">External_Gate_St—This bit indicates whether the external gate and the software gate are set to enable analog input operation.</td></tr>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0</td><td>Pause analog input operation.</td></tr>
<tr><td>1</td><td>Enable analog input operation.</td></tr>
<tr><td>28</td><td colspan="2">Scan_In_Progress_St—This bit indicates whether a scan is currently in progress. The bit is set when a valid AI_START is received and the bit is cleared when a valid AI_STOP is received.</td></tr>
<tr><td>27</td><td colspan="2">SC_PreWaitCountTC_St—Indicates that the SC_TC counter rolled over while waiting for the START2 trigger. Clear this bit using the SC_PreWaitCountTC_Interrupt_Ack bit.</td></tr>
<tr><td rowspan="4">26</td><td colspan="2">Start_Stop_Gate_St—This bit indicates the status of the start/stop gate, if start/stop gating is enabled:</td></tr>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0</td><td>External AI_CONVERTs are blocked because a valid AI_START has not been received.</td></tr>
<tr><td>1</td><td>External AI_CONVERTs are allowed to pass.</td></tr>
<tr><td rowspan="4">25</td><td colspan="2">SC_Gate_St—This bit indicates the status of the AI_SC gate if the AI_SC gate is enabled.</td></tr>
<tr><td>Value</td><td>Meaning</td></tr>
<tr><td>0</td><td>AI_SC gate blocks external AI_CONVERTs.</td></tr>
<tr><td>1</td><td>AI_SC gate allows external AI_CONVERTs to pass.</td></tr>
<tr><td>24</td><td colspan="2">DIV_Armed_St—This bit indicates whether the AI_DIV counter is armed.
The values for this bitfield are in InTimer_Disarmed_Armed_t.</td></tr>
</table>

<table>
<tr><th>Bits</th><th>Name</th></tr>
<tr><td>23</td><td>SI2_Next_Load_Source_St—This bit indicates the next load source of the AI_SI2 counter.

The values for this bitfield are in InTimer_Load_A_Load_B_t.</td></tr>
<tr><td>22</td><td>SI2_Armed_St—This bit indicates whether the AI_SI2 counter is armed.

The values for this bitfield are in InTimer_Disarmed_Armed_t.</td></tr>
<tr><td>21</td><td>SI_Counting_St—If the AI_SI counter is armed, this bit indicates whether the AI_SI counter is enabled to count. If the AI_SI counter is disarmed, this bit should be ignored.</td></tr>
<tr><td>20</td><td>SI_Next_Load_Source_St—This bit indicates the next load source of the AI_SI counter.

The values for this bitfield are in InTimer_Load_A_Load_B_t.</td></tr>
<tr><td>19</td><td>SI_Armed_St—This bit indicates whether the AI_SI counter is armed.

The values for this bitfield are in InTimer_Disarmed_Armed_t.</td></tr>
<tr><td>18</td><td>Reserved</td></tr>
<tr><td>17</td><td>SC_Next_Load_Source_St—This bit indicates the next load source of the AI_SC counter.

The values for this bitfield are in InTimer_Load_A_Load_B_t.</td></tr>
<tr><td>16</td><td>SC_Armed_St—This bit indicates whether the AI_SC counter is armed.

The values for this bitfield are in InTimer_Disarmed_Armed_t.</td></tr>
<tr><td>15</td><td>ScanOverrun_St—This bit indicates the detection of a ScanOverrun error. A ScanOverrun error is detected if an AI sample clock pulse comes in the middle of a scan interval (time between the sample clock and the convert pulse of the last channel for that sample).

The values for this bitfield are in InTimer_Error_t.</td></tr>
<tr><td>14</td><td>FIFO_Full_St—This bit reflects the state of the AI FIFO Full internal signal (after the polarity selection), which indicates the AI data FIFO status.

The values for this bitfield are in InTimer_FIFO_Full_St_t.</td></tr>
<tr><td>13</td><td>FIFO_Half_Full_St—This bit reflects the state of the AI FIFO Half-full internal signal (after the polarity selection), which indicates the AI data FIFO status:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Less than half-full.</td></tr>
<tr><td>1</td><td>Half-full or more than half-full.</td></tr>
</table></td></tr>
<tr><td>12</td><td>FIFO_Empty_St—This bit reflects the state of the AI FIFO Empty internal signal (after the polarity selection), which indicates the AI data FIFO status.

The values for this bitfield are in InTimer_FIFO_Empty_St_t.</td></tr>
<tr><td>11</td><td>Overrun_St—This bit indicates the detection of an ADC overrun error. The overrun error indicates that the AI_CONVERT interval is not long enough to complete a conversion. Clear this bit by setting AI_Overrun_Interrupt_Ack to 1.

The values for this bitfield are in InTimer_Error_t.</td></tr>
</table>

Bits	Name
10	**Overflow_St**—This bit indicates the detection of an ADC overflow error. The overflow error indicates that an attempt was made to write the ADC result to a full AI data FIFO; that is, the reading from the FIFO is too slow to match the writing to the FIFO. If the overflow error occurs, at least one point of data has been lost. This bit is cleared by setting AI_Overflow_Interrupt_Ack to 1. The values for this bitfield are in *InTimer_Error_t*.
9	**SC_TC_Error_St**—This bit indicates the detection of an SC_TC error. An SC_TC error is detected if AI_SC_TC_Interrupt_Ack is not set between two AI_SC TCs. This allows you to detect large interrupt latencies and potential problems associated with them. To clear this bit, set SC_TC_Error_Confirm to 1. The values for this bitfield are in *InTimer_Error_t*.
8	**START2_St**—This bit indicates whether a valid AI_START2 trigger has been received by the AI_SC counter in the pretrigger acquisition mode. A valid AI_START2 signal is one that is received while the AI_SC counter is in the Waiting for AI_START2 state. Clear this bit by setting AI_START2_Interrupt_Ack to 1. Refer to the *DMA Interrupts* section in Chapter 1, *Theory of Operation*, for more information.
7	**START1_St**—This bit indicates that a valid AI_START1 trigger has been received by the DAQ-STC3. A valid AI_START1 trigger is one that is received while the AI_SC counter is armed and in the Idle state. Clear this bit by setting AI_START1_Interrupt_Ack to 1. Refer to the *DMA Interrupts* section in Chapter 1, *Theory of Operation*, for more information.
6	**SC_TC_St**—This bit indicates whether the AI_SC counter has reached TC. Clear this bit by setting AI_SC_TC_Interrupt_Ack to 1. Refer to the *DMA Interrupts* section in Chapter 1, *Theory of Operation*, for more information.
5	**START_St**—This bit indicates that a valid AI_START trigger has been received by the AI timing engine. A valid AI_START trigger is one that is received while the AI_SC counter is enabled to count. Clear this bit by setting AI_START_Interrupt_Ack to 1. Refer to the *DMA Interrupts* section in Chapter 1, *Theory of Operation*, for more information.
4	**STOP_St**—This bit indicates that a valid AI_STOP signal has been received by the AI timing engine. A valid AI_STOP trigger is one that is received while the AI_SC counter is enabled to count after a valid AI_START. Clear this bit by setting AI_STOP_Interrupt_Ack to 1. Refer to the *DMA Interrupts* section in Chapter 1, *Theory of Operation*, for more information.
3..2	**Reserved**
1	**FIFO_Request_St**—This bit indicates the status of the DMA request (internal signal AIFREQ) and the FIFO interrupt. AI_FIFO_Mode selects the condition on which to generate the DMA request and FIFO interrupt. The values for this bitfield are in *InTimer_FIFO_Request_St_t*.
0	**Reserved**

Offset 0x00: Command_Register (W)

Absolute Addresses:
AI: 0x202B0
DI: 0x20560

Bits	Name
31	**End_On_SC_TC** (Strobe)—Setting this bit to 1 disarms the AI_SC, AI_SI, AI_SI2, and AI_DIV counters at the next SC_TC. You can use this bit to stop the acquisition in continuous acquisition mode. This bit is cleared automatically.
30	**End_On_End_Of_Scan** (Strobe)—Setting this bit to 1 disarms the AI_SC, AI_SI, AI_SI2, and AI_DIV counters at the next AI_STOP. You can use this bit to stop the acquisition in continuous acquisition mode. This bit is cleared automatically. **Note**: If you set this bit as part of a single point AI acquisition on one channel in single-wire mode, you will actually acquire two points. You can discard the second point.
29	**SC_PreWaitCountTC_ErrorAck** (Strobe)—This bit clears the SC_PreWaitCountTC_Error status.
28..26	**Reserved**
25	**SI_Switch_Load_On_SC_TC** (Strobe)—Setting this bit to 1 causes the AI_SI counter to switch load registers at the next SC_TC. This action is internally synchronized to the falling edge of the internal signal SI_CLK. You use this bit for scan rate change during an acquisition for staged analog input. This bit is cleared automatically. This bit has no effect if the SI_Reload_Mode is set to a value different than “No_Change”.
24	**SI_Switch_Load_On_STOP** (Strobe)—Setting this bit to 1 causes the AI_SI counter to switch load registers upon receiving a AI_STOP trigger. This action is internally synchronized to the falling edge of the internal signal SI_CLK. This bit is cleared automatically. This bit has no effect if the SI_Reload_Mode is set to a value different than “No_Change”.
23	**SI_Switch_Load_On_TC** (Strobe)—Setting this bit to 1 causes the AI_SI counter to switch load registers at its next TC. This action is internally synchronized to the falling edge of the internal signal SI_CLK. You can use this bit for scan rate change during an acquisition. This bit is cleared automatically. This bit has no effect if the SI_Reload_Mode is set to a value different than “No_Change”.
22..21	**Reserved**
20	**SC_Switch_Load_On_TC** (Strobe)—Setting this bit to 1 causes the AI_SC counter to switch load registers at the next SC_TC. You can use this bit for staged analog input. This bit is cleared automatically. This bit has no effect if the SC_Reload_Mode is set to a value different than “SC_Reload_No_Change”.
19	**Reserved**
18	**START_Pulse** (Strobe)—Setting this bit to 1 sends an AI_START trigger to the counters if the AI_START software strobe is selected. You select the software strobe by setting AI_START_Select to 18. This bit is cleared automatically.

Bits	Name
17	**START2_Pulse** (Strobe)—Setting this bit to 1 sends a AI_START2 trigger to the AI_SC counter if the AI_START2 software strobe is selected. The AI_START2 software strobe is selected when AI_START2_Select is set to 0. This bit is cleared automatically.
16	**START1_Pulse** (Strobe)—Setting this bit to 1 sends an AI_START1 trigger to the counters if the AI_START1 software strobe is selected. The AI_START1 software strobe is selected when AI_START1_Select is set to 0. This bit is cleared automatically.
15..14	**Reserved**
13	**Disarm** (Strobe)—Setting this bit to 1 asynchronously disarms the AI_SC, AI_SI, AI_SI2, and AI_DIV counters. This bit is cleared automatically.
12	**SI2_Arm** (Strobe)—Setting this bit to 1 arms the AI_SI2 counter. The counter remains armed and the bit remains set until it is disarmed, either by hardware or by setting AI_Disarm to 1. **Note**: All Arm bits used for a given acquisition should be set on the same register write.
11	**SI2_Load** (Strobe)—This bitfield is load register A for the AI_SI2 counter. If load register A is the selected AI_SI2 load register, the AI_SI2 counter loads the value contained in this bitfield on AI_SI2_Load and on SI2_TC.
10	**SI_Arm** (Strobe)—Setting this bit to 1 arms the AI_SI counter. The counter remains armed (and the bit remains set) until it is disarmed, either by hardware or by setting AI_Disarm to 1. **Note**: All Arm bits used for a given acquisition should be set on the same register write.
9	**SI_Load** (Strobe)—If the AI_SI counter is disarmed, this bit loads the AI_SI counter with the contents of the selected AI_SI load register (A or B). If the AI_SI counter is armed, writing to this bit has no effect. This bit is cleared automatically.
8	**DIV_Arm** (Strobe)—This bit arms the AI_DIV counter. The counter remains armed (and the bit remains set) until it is disarmed, either by hardware or by setting AI_Disarm to 1. **Note**: All Arm bits used for a given acquisition should be set on the same register write.
7	**DIV_Load** (Strobe)—If the AI_DIV counter is disarmed, this bit loads the AI_DIV counter with the contents of the AI_DIV load register. If the AI_DIV counter is armed, writing to this bit has no effect. This bit is cleared automatically.
6	**SC_Arm** (Strobe)—This bit arms the AI_SC counter. The counter remains armed (and the bit remains set) until it is disarmed, either by hardware or by setting AI_Disarm to 1. **Note**: All Arm bits used for a given acquisition should be set on the same register write.
5	**SC_Load** (Strobe)—If the AI_SC counter is disarmed, this bit loads the AI_SC counter with the contents of the selected AI_SC load register (A or B). If the AI_SC counter is armed, writing to this bit has no effect. This bit is cleared automatically.
4	**Reserved**
3	**EXTMUX_CLK_Pulse** (Strobe)—Setting this bit to 1 produces a pulse on the AI_EXTMUX_CLK output signal if the output is enabled. The pulsewidth is determined by AI_EXTMUX_CLK_Pulse_Width. This bit is cleared automatically.

Bits	Name
2	**LOCALMUX_CLK_Pulse** (Strobe)—Setting this bit to 1 produces a pulse on the internal signal AI_LOCALMUX_CLK. This pulse advances the configuration FIFO. This is useful for priming the analog front end prior to an acquisition start.
1	**SI_Cancel_Load_Switch** (Strobe)—Setting this bit to 1 causes any switch pending in the load source for the SI counter to be cancelled. This only works if the switch pending was caused by another strobe bit in this register.
0	**CONVERT_Pulse** (Strobe)—Setting this bit to 1 produces a pulse on the p_AI_Convert and AI_CONVERT output signals if the signals are enabled for output and if AI_CONVERT pulses are not blocked. AI_CONVERT pulses can be blocked by the external gate, the software gate, the start/stop gate, or the AI_SC gate. The pulsewidths of the output signals are determined by AI_CONVERT_Pulse_Width. This bit is cleared automatically. This bit is disabled when AI_Configuration_Start is set to 1.
Options: No Soft Copy	

Offset 0x04: Status_2_Register (R)

Absolute Addresses:
AI: 0x202B4
DI: 0x20564

Bits	Name	
31..16	**Reserved**	
15	**DIV_Q_St**—This bit reflects the state of the AI_DIV control state machine. The values for this bitfield are in *InTimer_Idle_Counting_t*.	
14	**Reserved**	
13..12	**SI_Q_St**—This bitfield reflects the state of the AI_SI control state machine.	
	Value	**Meaning**
	0	**Idle**
	1	**Waiting for AI_START**
	2	**Counting**
	3	**Counting Until TC**
11..10	**Reserved**	

<table>
<tr><th>Bits</th><th colspan="2">Name</th></tr>
<tr><td>9..8</td><td colspan="2">SI2_Q_St—This bitfield reflects the state of the AI_SI2 control state machine.</td></tr>
<tr><td></td><td>Value</td><td>Meaning</td></tr>
<tr><td></td><td>0</td><td>Idle</td></tr>
<tr><td></td><td>1</td><td>Counting</td></tr>
<tr><td></td><td>2</td><td>Waiting for AI_START</td></tr>
<tr><td>7..5</td><td colspan="2">Reserved</td></tr>
<tr><td>4..2</td><td colspan="2">SC_Q_St—This bitfield reflects the state of the AI_SC control state machine.
The values for this bitfield are in InTimer_SC_Q_St_t.</td></tr>
<tr><td>1</td><td colspan="2">Reserved</td></tr>
<tr><td>0</td><td colspan="2">SI_Load_Switch_Pending_St—This bit indicates if there is any switch pending in the load source for the SI counter. This only reflects a switch pending if the switch is not automatic (the switch was requested using the bitfields in the command register).
The values for this bitfield are in InTimer_SI_Load_Switch_Pending_t.</td></tr>
</table>

Offset 0x04: Mode_1_Register (W)

Absolute Addresses:
AI: 0x202B4
DI: 0x20564

Bits	Name
31	**SCAN_IN_PROG_Pulse**—Set this bit to 1 to begin a pulse on the SCAN_IN_PROG output signal, if the output is enabled. Set this bit to 0 to end the pulse. Scan In Progress output signal is the Start (sample clock) when programmed for that mode.
30..27	**Reserved**
26..22	**SI_Source_Select**—Selects the AI_SI source. The SI Source will be used to clock the SI counter. In SMIO mode, converts will be generated from this signal. In MIO mode, the sample clock (Start) is generated from this signal. The values for this bitfield are in *InTimer_SI_Source_Select_t*.
21	**Reserved**
20	**SI_Source_Polarity**—This bit selects the active edge of the AI_SI source signal. Set this bit to 0 if an internal timebase is used. The values for this bitfield are in *InTimer_SI_Source_Polarity_t*.
19..18	**Reserved**

Bits	Name
17	**Continuous**—This bit determines the behavior of the AI_SC, AI_SI, AI_SI2, and AI_DIV counters during SC_TC: If this bit is set to 0, when AI_Pre_Trigger is 0, the counters return to idle on the first SC_TC. If AI_Pre_Trigger is 1, the counters return to idle on the second SC_TC. If this bit is set to 1, the counters ignore SC_TC. Set this bit to 0 if you want to acquire a predetermined number of scans. You also need to use AI_Pre_Trigger to select the pretrigger or posttrigger acquisition mode. Set this bit to 1 if you want to continuously acquire data or to perform staged analog input. You can use AI_End_On_End_Of_Scan and AI_End_On_SC_TC to stop an analog input operation in the continuous acquisition mode.
16	**Trigger_Once**—This bit controls the retriggerability of the AI_SC, AI_SI, AI_SI2 and AI_DIV counters. When this bit is 0, the counters remain armed and retriggerable after generating a timing sequence. When the bit is 1, the counters disarm after one AI timing sequence. Set this bit to 0 for a single, finite-pretrigger-infinite-posttrigger AI operation. Set this bit to 1 if you want to retrigger an acquisition. If AI_Continuous is set to 0, you may not set this bit to 1. This restriction means you are not allowed to perform a retriggerable, continuous AI acquisition. **Note**: If the operation is halted by AI_End_On_End_Of_Scan or AI_End_On_SC_TC, the counters are disarmed, regardless of the state of this bit.
15	**Reserved**
14	**Start_Stop_Gate_Enable**—This bit enables the start/stop gate (STST_GATE) When enabled, external AI_CONVERT pulses pass through the DAQ-STC3 only during the interval between the assertion of AI_START and the assertion of AI_STOP. Only enable in the external AI_CONVERT mode. Disable this bitfield in the internal AI_CONVERT mode. The values for this bitfield are in *InTimer_Disabled_Enabled_t*.
13	**Pre_Trigger**—If AI_Continuous is 0, this bit selects between the posttrigger acquisition mode and the pretrigger acquisition mode. If AI_Continuous is 1, this bit is not used. The values for this bitfield are in *InTimer_Pre_Trigger_t*.
12	**External_MUX_Present**—This bit determines when the AI_LOCALMUX_CLK output signal pulses. This bit allows you to use the AI_DIV counter for AI_LOCALMUX_CLK signal control. This is useful if one or more external multiplexers, such as an AMUX-64T or SCXI, are connected to the device the DAQ-STC3 is on. You should set this bit to 0 if no external multiplexers are present or if each external channel corresponds to one internal channel. You should set this bit to 1 if one or more external multiplexers are present and you are multiplexing more than one external channel onto each internal channel. If this bit is set to 1, the AI_DIV counter must be used to determine the number of AI_EXTMUX_CLK pulses that correspond to one AI_LOCALMUX_CLK pulse. The values for this bitfield are in *InTimer_External_MUX_Present_t*.
11..10	**Reserved**

Bits	Name
9	**SI2_Initial_Load_Source**—This bit selects the initial AI_SI2 load register. Do not change this bit while the counter is counting. The values for this bitfield are in *InTimer_Load_A_Load_B_t*.
8	**SI2_Reload_Mode**—This bit selects the reload mode for the AI_SI2 counter. Set this bit to 1 in the internal AI_CONVERT mode to make the time intervals between the AI_START trigger and the first AI_CONVERT different from the time interval between AI_CONVERTs. The values for this bitfield are in *InTimer_SI2_Reload_Mode_t*.
7	**SI_Initial_Load_Source**—If the AI_SI counter is disarmed, this bit selects the initial AI_SI load register. If the AI_SI counter is armed, writing to this bit has no effect. The values for this bitfield are in *InTimer_Load_A_Load_B_t*.
6..4	**SI_Reload_Mode**—This bit selects the reload mode for the AI_SI counter. The values for this bitfield are in *InTimer_SI_Reload_Mode_t*.
3	**Reserved**
2	**SC_Initial_Load_Source—**If the AI_SC counter is disarmed, this bit selects the initial AI_SC load register. If the AI_SC counter is armed, this bit has no effect. The values for this bitfield are in *InTimer_Load_A_Load_B_t*.
1	**SC_Reload_Mode**—This bitfield selects the reload mode for the AI_SC counter. You can use 1 for pretriggered acquisition mode and for staged analog input. The values for this bitfield are in *InTimer_SI_Reload_Mode_t*.
0	**ExportedConvertPolarity**—This bit determines the polarity of the Convert signal that gets exported to PFI or RTSI or to other resources on the DAQ-STC3. Its default value is 1, which means Active Low signals. The values for this bitfield are in *InTimer_Polarity_t*.
Options: Initial Value = 0x0001	

Offset 0x08: SI_Save_Register (R)

Absolute Addresses:
AI: 0x202B8
DI: 0x20568

Bits	Name
31..0	**SI_Save_Value**—This bitfield reflects the contents of the AI_SI counter. Reading from this bitfield while the AI_SI counter is counting can result in an erroneous value.

Offset 0x08: Mode_2_Register (W)

Absolute Addresses:
AI: 0x202B8
DI: 0x20568

Bits	Name	
31	**Reserved**	
30	**HaltOnError**—When set, the timing engine prevents any more converts to be generated or any more data to be written to the FIFO after an error condition.	
29	**Software_Gate**—This bit controls the software gate, which you can use to pause an analog input operation:	
	Value	**Meaning**
	0	**Enable operation**
	1	**Pause operation**
28	**Reserved**	
27	**SI2_Source_Select**—This bit selects the AI_SI2 source. In MIO mode, the AI_SI2 signal clocks the SI2 counter, and in internal timing, it generates the convert signal. The values for this bitfield are in *InTimer_SI2_Source_Select_t*.	
26..25	**Reserved**	
24	**External_Gate_Mode**—This bit determines the gating mode, if gating is enabled. The values for this bitfield are in *InTimer_External_Gate_Mode_t*.	
23..22	**FIFO_Mode**—This bit selects the AI data FIFO condition on which to generate the FIFO interrupt (if the FIFO interrupt is enabled). The values for this bitfield are in *InTimer_FIFO_Mode_t*.	
21..20	**SyncMode**—This bitfield determines the mode of operation of trigger synchronization. The values for this bitfield are in *InTimer_SyncMode_t*.	
19	**Start_Trigger_Length**—This bit determines the length of the AI_START signal when it is output on one of the PFI pins and AI_START_Output_Select is set to 0 (AI_START). The values for this bitfield are in *InTimer_Start_Trigger_Length_t*.	
18	**Start1_Export_Mode**—This bit determines the AI_START1 signal when it is output on one of the PFI pins. The values for this bitfield are in *InTimer_Start1_Export_Mode_t*.	
17	**Start2_Export_Mode**—This bit determines the AI_START2 signal when it is output on one of the PFI pins. The values for this bitfield are in *InTimer_Start2_Export_Mode_t*.	

Bits	Name
16..11	**Reserved**
10	**START_Output_Select**—This bit selects the source for the AI_START signal output on PFI. The values for this bitfield are in *InTimer_START_Output_Select_t*.
9	**SCAN_IN_PROG_Polarity**—When AI_START is output on a PFI pin and AI_START_Output_Select is set to 1 (output SCAN_IN_PROG), this bitfield selects the polarity of the SCAN_IN_PROG signal. The values for this bitfield are in *InTimer_Polarity_t*.
8	**Reserved**
7	**EXTMUX_CLK_Polarity**—When AI_EXTMUX_CLK is output on a PFI pin, this bit enables and selects the polarity of the AI_EXTMUX_CLK output signal. The values for this bitfield are in *InTimer_Polarity_t*.
6	**Reserved**
5	**EXTMUX_CLK_Pulse_Width**—This bit selects the pulsewidth and assertion time of the AI_EXTMUX_CLK output signal. The values for this bitfield are in *InTimer_EXTMUX_CLK_Pulse_Width_t*.
4..0	**Reserved**

Offset 0x0C: SC_Save_Register (R)

Absolute Addresses:
AI: 0x202BC
DI: 0x2056C

Bits	Name
31..0	**SC_Save_Value**—Reflects the state of the SC counter. It is a safe read even when the counter is running.

Offset 0x0C: SI_Load_A_Register (W)

Absolute Addresses:
AI: 0x202BC
DI: 0x2056C

Bits	Name
31..0	**SI_Load_A**—This bitfield is the load value A for the AI_SI counter. If load register A is the selected AI_SI load register, the AI_SI counter loads the value contained in this bitfield on AI_SI_Load and on SI_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AI_SI_Load (in the Command register).

Offset 0x10: SI2_Save_Register (R)

Absolute Addresses:
AI: 0x202C0
DI: 0x20570

Bits	Name
31..0	**SI2_Save_Value**—This bitfield reflects the contents of the AI_SI2 counter. Reading from this bitfield while the AI_SI2 counter is counting can result in an erroneous value.

Offset 0x10: SI_Load_B_Register (W)

Absolute Addresses:
AI: 0x202C0
DI: 0x20570

Bits	Name
31..0	**SI_Load_B**—This bitfield is the load value B for the AI_SI counter. If load register B is the selected AI_SI load register, the AI_SI counter loads the value contained in this bitfield on AI_SI_Load and on SI_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AI_SI_Load (in the Command register).

Offset 0x14: DIV_Save_Register (R)

Absolute Addresses:
AI: 0x202C4
DI: 0x20574

Bits	Name
15..0	**DIV_Save_Value**—This bitfield reflects the contents of the AI_DIV counter. Reading from this bitfield while the AI_DIV counter is counting can result in an erroneous value.

Offset 0x14: SC_Load_A_Register (W)

Absolute Addresses:
AI: 0x202C4
DI: 0x20574

Bits	Name
31..0	**SC_Load_A**—This bitfield is the load value A for the AI_SC counter. If load register A is the selected AI_SC load register, the AI_SC counter loads the value contained in this bitfield on AI_SC_Load and on SC_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AI_SC_Load (in the Command register).

Offset 0x18: SC_PreWaitCntRegister (R)

Absolute Addresses:
AI: 0x202C8
DI: 0x20578

Bits	Name
31..0	**SC_PreWaitCount**—This register reflects the number of points that were acquired between the end of the pre-trigger count of samples and the arrival of the START2 trigger. This register should be read after the Start2 trigger (at the end of the acquisition, for example). For very long (more than 2^{32} samples in the Waiting state) acquisitions, software must keep track of the DI_SC_PreWaitCountTC event in order to keep an accurate count.

Offset 0x18: SC_Load_B_Register (W)

Absolute Addresses:
AI: 0x202C8
DI: 0x20578

Bits	Name
31..0	**SC_Load_B**—This bitfield is the load value B for the AI_SC counter. If load register B is the selected AI_SC load register, the AI_SC counter loads the value contained in this bitfield on AI_SC_Load and on SC_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AI_SC_Load (in the Command register).

Offset 0x1C: SI2_Load_A_Register (W)

Absolute Addresses:
AI: 0x202CC
DI: 0x2057C

Bits	Name
31..0	**SI2_Load_A**—This bitfield is the load value A for the AI_SI2 counter. If load register A is the selected AI_SI2 load register, the AI_SI2 counter loads the value contained in this bitfield on AI_SI2_Load and on SI2_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AI_SI_Load (in the Command register).

Offset 0x20: SI2_Load_B_Register (W)

Absolute Addresses:
AI: 0x202D0
DI: 0x20580

Bits	Name
31..0	**SI2_Load_B**—This bitfield is the load value B for the AI_SI2 counter. If load register B is the selected AI_SI2 load register, the AI_SI2 counter loads the value contained in this bitfield on AI_SI2_Load and on SI2_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AI_SI2_Load (in the Command register).

Offset 0x24: DIV_Load_A_Register (W)

Absolute Addresses:
AI: 0x202D4
DI: 0x20584

Bits	Name
15..0	**DIV_Load_A**—This bitfield is the load register for the AI_DIV counter. The AI_DIV counter loads the value contained in this bitfield on AI_DIV_Load and on DIV_TC. The DIV Counter can be used to repeat *N* times each entry in the configuration FIFO, where *N* is the value programmed in this register.

Offset 0x2C: Interrupt1_Register (W)

Absolute Addresses:
AI: 0x202DC
DI: 0x2058C

Bits	Name
31	**Overflow_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_Overflow_St and acknowledges the Overflow interrupt request if the Overflow interrupt is enabled. This bit is cleared automatically.
30	**SC_PreWaitCountTC_Interrupt_Ack** (Strobe)—This bit clears the SC_PreWaitCountTC_Interrupt event and acknowledges the interrupt.
29	**Overrun_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_Overrun_St and acknowledges the Overrun interrupt request if the Overrun interrupt is enabled. This bit is cleared automatically.
28	**STOP_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_STOP_St and acknowledges the AI_STOP interrupt request if the AI_STOP interrupt is enabled. This bit is cleared automatically.
27	**START_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_START_St and acknowledges the AI_START interrupt request if the AI_START interrupt is enabled. This bit is cleared automatically.
26	**START2_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_START2_St and acknowledges the AI_START2 interrupt request if the AI_START2 interrupt is enabled. This bit is cleared automatically.
25	**START1_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_START1_St and acknowledges the AI_START1 interrupt request if the AI_START1 interrupt is enabled. This bit is cleared automatically.
24	**SC_TC_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_Last_Shiftin_St, AI_SC_TC_St, and the SC_TC interrupt request (in either interrupt bank) if the SC_TC interrupt is enabled. This bit is cleared automatically.
23	**SC_TC_Error_Confirm** (Strobe)—Setting this bit to 1 clears AI_SC_TC_Error_St. This bit is cleared automatically.
22	**ScanOverrun_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AI_ScanOverrun_St. This bit is cleared automatically.

Bits	Name
21..10	**Reserved**
9	**SC_PreWaitCountTC_Interrupt_Enable** (Strobe)—This bit enables the Interrupt on the event that the SC_TC rolls over when the device is in the PreWait state. This is the state after the pretrigger count has been satisfied but before the START2 trigger has been received. This event is useful to keep track of how many points have been received. If another of these events happens before the original is acknowledged the error will be signaled by the SC_PreWaitCountTC_ErrorSt bit.
8	**ScanOverrun_Interrupt_Enable** (Strobe)—This strobe bit enables the AI ScanOverrun interrupt. A ScanOverrun error is generated if an AI sample clock pulse comes in the middle of a scan interval (time between the sample clock and the convert pulse of the last channel for that sample).
7	**FIFO_Interrupt_Enable** (Strobe)—This strobe bit enables the AI FIFO interrupt. The FIFO interrupt is generated on the FIFO condition indicated by AI_FIFO_Mode.
6	**Overflow_Interrupt_Enable** (Strobe)—This strobe bit enables the AI Overflow interrupt.
5	**Overrun_Interrupt_Enable** (Strobe)—This strobe bit enables the AI Overrun interrupt.
4	**STOP_Interrupt_Enable** (Strobe)—This strobe bit enables the AI_STOP interrupt. The AI_STOP interrupt is generated on valid AI_STOP triggers recognized by the DAQ-STC3. A valid AI_STOP trigger is one that is received while the AI_SC counter is enabled to count after a valid AI_START.
3	**START_Interrupt_Enable** (Strobe)—This strobe bit enables the AI_START interrupt. The AI_START interrupt is generated on valid AI_START triggers received by the DAQ-STC3. A valid AI_START trigger is one that is received while the AI_SC counter is enabled to count.
2	**START2_Interrupt_Enable** (Strobe)—This strobe bit enables the AI_START2 interrupt. The AI_START2 interrupt is generated on valid AI_START2 triggers received by the DAQ-STC3. A valid AI_START2 trigger is one that is received while the AI_SC counter is in the Waiting for AI_START2 state.
1	**START1_Interrupt_Enable** (Strobe)—This strobe bit enables the AI_START1 interrupt. The AI_START1 interrupt is generated on valid AI_START1 triggers received by the DAQ-STC3. A valid AI_START1 trigger is one that is received while the AI_SC counter is armed and in the Idle state.
0	**SC_TC_Interrupt_Enable** (Strobe)—This strobe bit enables the SC_TC interrupt. SC_TC interrupts are generated on every SC_TC falling edge unless the pretrigger acquisition mode is selected. In the pretrigger acquisition mode, the first SC_TC falling edge does not generate an interrupt, but subsequent SC_TC falling edges do.
Options: No Soft Copy	

Offset 0x30: Interrupt2_Register (W)

Absolute Addresses:
AI: 0x202E0
DI: 0x20590

Bits	Name
31	**Overflow_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_Overflow_St and acknowledges the Overflow interrupt request if the Overflow interrupt is enabled. This bit is cleared automatically.
30	**SC_PreWaitCountTC_Interrupt_Ack2** (Strobe)—This bit clears the SC_PreWaitCountTC_Interrupt event and acknowledges the interrupt.
29	**Overrun_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_Overrun_St and acknowledges the Overrun interrupt request if the Overrun interrupt is enabled. This bit is cleared automatically.
28	**STOP_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_STOP_St and acknowledges the AI_STOP interrupt request if the AI_STOP interrupt is enabled. This bit is cleared automatically.
27	**START_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_START_St and acknowledges the AI_START interrupt request if the AI_START interrupt is enabled. This bit is cleared automatically.
26	**START2_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_START2_St and acknowledges the AI_START2 interrupt request if the AI_START2 interrupt is enabled. This bit is cleared automatically.
25	**START1_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_START1_St and acknowledges the AI_START1 interrupt request if the AI_START1 interrupt is enabled. This bit is cleared automatically.
24	**SC_TC_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_Last_Shiftin_St, AI_SC_TC_St, and the SC_TC interrupt request if the SC_TC interrupt is enabled. This bit is cleared automatically.
23	**SC_TC_Error_Confirm2** (Strobe)—Setting this bit to 1 clears AI_SC_TC_Error_St. This bit is cleared automatically.
22	**ScanOverrun_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AI_ScanOverrun_St. This bit is cleared automatically.
21..10	**Reserved**
9	**SC_PreWaitCountTC_Interrupt_Disable** (Strobe)—This bit disables the Interrupt on the event that the SC_TC rolls over when the device is in the PreWait state. This is the state after the pretrigger count has been satisfied but before the START2 trigger has been received. This event is useful to keep track of how many points have been received. If another of these events happens before the original is acknowledged the error will be signaled by the SC_PreWaitCountTC_ErrorSt bit.
8	**ScanOverrun_Interrupt_Disable** (Strobe)—This bit disables the AI ScanOverrun interrupt. A ScanOverrun error is generated when an AI sample clock pulse comes in the middle of a scan interval (time between the sample clock and the convert pulse of the last channel for that sample).
7	**FIFO_Interrupt_Disable** (Strobe)—This bit disables the AI FIFO interrupt. The FIFO interrupt is generated on the FIFO condition indicated by AI_FIFO_Mode.
6	**Overflow_Interrupt_Disable** (Strobe)—This strobe bit disables the AI Overflow interrupt.

Bits	Name
5	**Overrun_Interrupt_Disable** (Strobe)—This strobe bit disables the AI Overrun interrupt.
4	**STOP_Interrupt_Disable** (Strobe)—This bit disables the AI_STOP interrupt. The AI_STOP interrupt is generated on valid AI_STOP triggers recognized by the DAQ-STC3. A valid AI_STOP trigger is one that is received while the AI_SC counter is enabled to count after a valid AI_START.
3	**START_Interrupt_Disable** (Strobe)—This bit disables the AI_START interrupt. The AI_START interrupt is generated on valid AI_START triggers received by the DAQ-STC3. A valid AI_START trigger is one that is received while the AI_SC counter is enabled to count.
2	**START2_Interrupt_Disable** (Strobe)—This bit disables the AI_START2 interrupt. The AI_START2 interrupt is generated on valid AI_START2 triggers received by the DAQ-STC3. A valid AI_START2 trigger is one that is received while the AI_SC counter is in the Waiting for AI_START2 state.
1	**START1_Interrupt_Disable** (Strobe)—This bit disables the AI_START1 interrupt. The AI_START1 interrupt is generated on valid AI_START1 triggers received by the DAQ-STC3. A valid AI_START1 trigger is one that is received while the AI_SC counter is armed and in the Idle state.
0	**SC_TC_Interrupt_Disable** (Strobe)—This bit disables the SC_TC interrupt. SC_TC interrupts are generated on every SC_TC falling edge unless the pretrigger acquisition mode is selected. In the pretrigger acquisition mode, the first SC_TC falling edge does not generate an interrupt, but subsequent SC_TC falling edges will.
Options: No Soft Copy	

Offset 0x38: Reset_Register (W)

Absolute Addresses:
AI: 0x202E8
DI: 0x20598

Bits	Name
15..5	**Reserved**
4	**FIFO_Clear** (Strobe)—Writing to this bit clears the AI Data FIFO.
3	**Configuration_Memory_Clear** (Strobe)—Writing to this bit clears the configuration FIFO.
2	**Configuration_End** (Strobe)—This bit clears Configuration_Start, which holds the analog input circuitry in reset to prevent glitches on the output pins during configuration. You should set this bit to 1 when ending the configuration of the analog input circuitry. This bit is cleared automatically.
1	**Configuration_Start** (Strobe)—This bit holds the analog input circuitry in reset to prevent glitches on the output pins during configuration. You should set this bit to 1 when beginning the configuration process of the analog input circuitry. By doing this you ensure that no spurious glitches appear on the output pins and on the internal circuit components. If you do not set this bit to 1, the DAQ-STC3 can behave erroneously. This bit is cleared by setting Configuration_End to 1.
0	**Reset** (Strobe)—Setting this bit to 1 resets the InTimer. Setting this bit to 1 also clears all the status bits and interrupts related to the subsystem, except those associated with the data FIFO. This bit is cleared automatically. **Note**: Using this bitfield to reset the timer also activates the ConfigReset. A strobe write to the Configuration_End bitfield must be executed after resetting the timer with this bitfield in a separate register access.
Options: No Soft Copy	

Enumerated Types

InTimer_Disabled_Enabled_t

Value	Name
0	**Disabled**
1	**Enabled**

InTimer_Disarmed_Armed_t

Value	Name
0	**Disarmed**
1	**Armed**

InTimer_Error_t

Value	Name
0	**NO_ERROR**
1	**ERROR**

InTimer_External_Gate_Mode_t

Value	Name
0	**Free_Run**—Free-run gating mode.
1	**Halt_Gating**—Halt-run gating mode.

InTimer_External_MUX_Present_t

Value	Name
0	**Every_Convert**—Pulse on every AI_CONVERT.
1	**DIV_TC_Converts**—Pulse only on AI_CONVERTs that occur during DIV_TC.

InTimer_EXTMUX_CLK_Pulse_Width_t

Value	Name
0	**ExtMuxPulseLong**—Pulsewidth depends on SCXI_Force_AI_EXTMUX_CLK_Width. AI_EXTMUX_CLK trails the AI_LOCALMUX_CLK pulse by 25–75 ns (1–3 TB4 periods).
1	**ExtMuxPulseShort**—Pulsewidth is 75–100 ns (3–4 TB4 periods). AI_EXTMUX_CLK and AI_LOCALMUX_CLK assert at the same time.

InTimer_FIFO_Empty_St_t

Value	Name
0	**Not_Empty**
1	**Empty**

InTimer_FIFO_Full_St_t

Value	Name
0	**Not_Full**
1	**Full**

InTimer_FIFO_Mode_t

Value	Name
0	**FifoMode_Not_Empty**—Generate on FIFO not empty. Keep the request and interrupt asserted while the FIFO is not empty.
1	**FifoMode_Half_Full**—Generate on FIFO more than half-full. Keep the request and interrupt asserted while the FIFO is half-full.
2	**FifoMode_Full**—Generate on FIFO full. Keep the request and interrupt asserted while the FIFO is full.
3	**FifoMode_Half_Full_Until_Empty**—Generate on FIFO more than half-full. Keep the request and interrupt asserted while the FIFO is not empty. At the end of an acquisition, the request and interrupt assert and remain asserted until the FIFO empties.

InTimer_FIFO_Request_St_t

Value	Name
0	**Not_Asserted**
1	**Asserted**

InTimer_Idle_Counting_t

Value	Name
0	**Idle**
1	**Counting**

InTimer_Load_A_Load_B_t

Value	Name
0	**Load_A**—Load register A.
1	**Load_B**—Load register B.

InTimer_Polarity_t

Value	Name
0	**Active_High**
1	**Active_Low**

InTimer_Pre_Trigger_t

Value	Name
0	**Posttrigger**
1	**Pretrigger**

InTimer_SC_Q_St_t

Value	Name
0	**SC_Idle**
1	**SC_WaitforStart1**
2	**SC_PreCounting**—Counting pretriggered samples.
3	**SC_PreWait**—Waiting for Start2 Trigger.
4	**SC_WaitForStartOfSample**—Master/Slave sync state.
5	**SC_Counting**—Counting Post-trigger samples.

InTimer_SC_Reload_Mode_t

Value	Name
0	**SC_Reload_No_Change**—No automatic change of the AI_SC load register.
1	**SC_Reload_Switch**—The AI_SC counter switches load registers on every SC_TC.

InTimer_SI_Load_Switch_Pending_t

Value	Name
0	**No_Switch_Pending**—There is no load switch pending in the SI counter.
1	**Switch_Pending**—There is a load switch pending in the SI counter.

InTimer_SI_Reload_Mode_t

Value	Name
0	**SI_Reload_No_Change**—No_Change.
4	**SI_Reload_Alt_First_Period_Every_STOP**—Alternate first period on every AI_STOP.
5	**SI_Reload_Switch_Every_STOP**—Switch on every AI_STOP.
6	**SI_Reload_Alt_First_Period_Every_SCTC**—Alternate first period every SC_TC.
7	**SI_Reload_Switch_Every_SCTC**—Switch on every SC_TC.

InTimer_SI_Source_Polarity_t

Value	Name
0	**Rising_Edge**
1	**Falling_Edge**

InTimer_SI_Source_Select_t

Value	Meaning	Value	Meaning
0	**SI_Src_TB3**	16	**SI_Src_RTSI5**
1	**SI_Src_PFI0**	17	**SI_Src_RTSI6**
2	**SI_Src_PFI1**	18	**SI_Src_TB2**
3	**SI_Src_PFI2**	19	**SI_Src_DStarA**
4	**SI_Src_PFI3**	20	**SI_Src_Star_Trigger**
5	**SI_Src_PFI4**	21	**SI_Src_PFI10**
6	**SI_Src_PFI5**	22	**SI_Src_PFI11**
7	**SI_Src_PFI6**	23	**SI_Src_PFI12**
8	**SI_Src_PFI7**	24	**SI_Src_PFI13**
9	**SI_Src_PFI8**	25	**SI_Src_PFI14**
10	**SI_Src_PFI9**	26	**SI_Src_PFI15**
11	**SI_Src_RTSI0**	27	**SI_Src_RTSI7**
12	**SI_Src_RTSI1**	28	**SI_Src_TB1**
13	**SI_Src_RTSI2**	29	**SI_Src_PXI_Clk10**
14	**SI_Src_RTSI3**	30	**SI_Src_Analog_Trigger**
15	**SI_Src_RTSI4**	31	**SI_Src_DStarB**

InTimer_SI2_Reload_Mode_t

Value	Name
0	**SI2_Reload_NoChange**—No automatic change of the AI_SI2 load register.
1	**SI2_Reload_Alt_First_Period_Every_STOP**—Alternate first period on every AI_STOP.

InTimer_SI2_Source_Select_t

Value	Name
0	**SI2_Src_Is_SI_Src**—The same signal selected as the AI_SI source.
1	**SI2_Src_IsTB3**—TB3.

InTimer_START_Output_Select_t

Value	Name
0	**AI_Start**—AI_START is output. The pulsewidth of the signal depends on AI_Start_Trigger_Length.
1	**SCAN_IN_PROG**—SCAN_IN_PROG is output. The pulse starts at the same time as sample clock, but remains asserted until the last convert of the scan.

InTimer_Start_Trigger_Length_t

Value	Name
0	**ExportSynchronizedStart**—Output the normal internal version of the signal. If in Single Wire mode, it is the edge detected convert signal, if not, then it is the start signal synchronized to SI Src (if SI and SI2 sources are external) or a four TB3 clks pulse extended version of the filtered start (synchronous to SI2 source and only when SC counter is ready to count starts).
1	**ExportExtendedStart**—In this case, the signal asserts synchronous to SI2 source and is extended to 16 TB3 clocks. Only valid starts are exported (SC counter ready to count starts).

InTimer_Start1_Export_Mode_t

Value	Name
0	**ExportSynchronizedStart1**—Output the internal signal. This version asserts synchronized to SI2 timebase and deasserts synchronized to the SI timebase, with a minimum pulse with of 16 TB3 clocks if SI2 timebase is TB3 or one SI timebase period otherwise.
1	**ExportEdgeDetectedStart1**—In this case, the signal is edge detected and pulse stretched. The width of the pulse can be 17–18 TB3 clock cycles.

InTimer_Start2_Export_Mode_t

Value	Name
0	**ExportUnmaskedStart2**—Output the internal signal. This version asserts synchronized to SI2 timebase and deasserts synchronized to the SI timebase, with a minimum pulse with of 16 TB3 clocks if SI timebase is TB3 or one SI timebase period otherwise.
1	**ExportMaskedStart2**—In this case, the signal is also synchronized to TB3; but it is only exported if the SC counter is ready to receive a START2 trigger. The length is 15 TB3 clock cycles.

InTimer_SyncMode_t

Value	Name
0	**SyncDefault**—This is the default mode of operation. The trigger is synchronized to the source of the timer for internal use only.
1	**SyncSlave**—In this mode, the timer is setup to receive the exported trigger from a master device. The transmission and reception of this trigger depends on the SyncPulse signal (PXIe_Sync100 on PXI Express systems).
2	**SyncMaster—**In this mode, the timer transmits the trigger and then waits to use it so that all devices (master and slaves) trigger at the same time.

18

OutTimer Registers

AO base address = 0x20470
DO base address = 0x204E0

This chapter consists of *OutTimer Registers* and *Enumerated Types*. Each register is labeled as a read (R) or write (W) register.

List of OutTimer Registers

The OutTimer registers are as follows:

List of Enumerated Types

The enumerated types are as follows:

OutTimer_Armed_t

OutTimer_BC_Gate_t

OutTimer_BC_Q_t

OutTimer_BC_Reload_Mode_t

OutTimer_Continuous_t

OutTimer_Disabled_Enabled_t

OutTimer_Error_t

OutTimer_External_Gate_t

OutTimer_FIFO_Empty_t

OutTimer_FIFO_Full_t

OutTimer_FIFO_Half_Full_t

OutTimer_FIFO_Mode_t

OutTimer_Load_Source_t

OutTimer_Mute_t

OutTimer_Polarity_t

OutTimer_Software_Gate_t

OutTimer_Start1_Export_Mode_t

OutTimer_Stop_On_Error_t

OutTimer_TMRDACWRs_In_Progress_t

OutTimer_Trigger_t

OutTimer_UC_Q_t

OutTimer_UC_Reload_Mode_t

OutTimer_UI_Count_Enabled_t

OutTimer_UI_Load_Switch_Pending_t

OutTimer_UI_Q_t

OutTimer_UI_Reload_Mode_t

OutTimer_UI_Source_Select_t

OutTimer_Write_Switch_t

OutTimerSyncMode_t

OutTimer Registers

Offset 0x00: Status_1_Register (R)

Absolute Addresses:
AO: 0x20470
DO: 0x204E0

Bits	Name
31	**Reserved**
30	**FIFO_Full_St**—This bit reflects the state of data FIFO Full flag. The values for this bitfield are in *OutTimer_FIFO_Full_t*.
29	**FIFO_Half_Full_St**—This bit reflects the state of the data FIFO Half Full flag. **Note**: The operation of this bit is similar to AI_FIFO_Half_Full as described in the *InTimer Registers* section of Chapter 17, *InTimer Registers*. For input operations, however, the FIFO requires service when it is *more* than half-full. For output operations, the FIFO requires service when it is *half-full or less*. For this reason, the input and output ISRs must check for opposite values when deciding on interrupt servicing. The values for this bitfield are in *OutTimer_FIFO_Half_Full_t*.
28	**FIFO_Empty_St**—This bit reflects the state of the AO empty flag from the data FIFO. The values for this bitfield are in *OutTimer_FIFO_Empty_t*.
27	**BC_TC_Error_St**—This bit indicates the detection of a BC_TC error. **Note**: A BC_TC error occurs if AO_BC_TC_Interrupt_Ack is not set between two AO_BC TCs. This allows you to detect large interrupt latencies and potential problems associated with them. To clear this bit, set AO_BC_TC_Error_Confirm to 1. The values for this bitfield are in *OutTimer_Error_t*.
26	**Underflow_St**—This bit indicates the detection of an underflow error. The values for this bitfield are in *OutTimer_Error_t*.
25	**Overrun_St**—This bit indicates the detection of an overrun error. **Note**: An overrun error occurs when an AO_UPDATE command is issued to a DAC that was not loaded with data. This bit can be cleared by setting AO_Error_Interrupt_Ack to 1. **Note**: This bit can incorrectly indicate that an error occurred after the end of a waveform generation sequence if there is no more data in the buffer. You can avoid this false error by transferring one more point of data to the device than the waveform generation requires. The values for this bitfield are in *OutTimer_Error_t*.

Bits	Name
24	**START1_St**—This bit indicates that a valid AO_START1 trigger has been received by the DAQ-STC3. **Note**: A valid AO_START1 trigger is one that is received while the AO_BC counter is armed and in the WAIT1 state. Clear this bit by setting AO_START1_Interrupt_Ack to 1. The values for this bitfield are in *OutTimer_Trigger_t*.
23	**BC_TC_St**—This bit indicates whether the AO_BC counter has reached TC. This bit is set on the trailing edge of BC_TC. Clear this bit by setting AO_BC_TC_Interrupt_Ack to 1.
22	**UC_TC_St**—This bit indicates whether the AO_UC counter has reached TC. To clear this bit, set AO_UC_TC_Interrupt_Ack to 1.
21	**UPDATE_St**—This bit indicates whether an AO_UPDATE has occurred. **Note**: Clear this bit by setting AO_UPDATE_Interrupt_Ack to 1. The values for this bitfield are in *OutTimer_Trigger_t*.
20..18	**Reserved**
17	**FIFO_Request_St**—This bit indicates the status of the DMA request (output signal AOFREQ) and FIFO interrupt. AO_FIFO_Mode selects the condition on which to generate the FIFO interrupt.
16	**Reserved**
15	**UC_Next_Load_Source_St**—This bit indicates the next load source of the AO_UC counter. The values for this bitfield are in *OutTimer_Load_Source_t*.
14	**UC_Armed_St**—This bit indicates whether the AO_UC counter is armed. The values for this bitfield are in *OutTimer_Armed_t*.
13	**UC_Q_St**—This bit reflects state of the AO_UC control state machine. The values for this bitfield are in *OutTimer_UC_Q_t*.
12	**External_Gate_St**—This bit indicates whether the external gate and software gate are set to enable waveform generation. The values for this bitfield are in *OutTimer_External_Gate_t*.
11	**BC_Gate_St**—When the BC_GATE is enabled (refer to the AO_BC_Gate_Enable bitfield in the *Mode_2_Register*), this bit indicates the state of the BC_GATE. **Note**: The BC_GATE is active only when the AO_BC counter is enabled to count. When the BC_GATE is disabled, this bit is undefined. Disable the BC_GATE in the internal AO_UPDATE mode. The values for this bitfield are in *OutTimer_BC_Gate_t*.

Bits	Name
10	**TMRDACWRs_In_Progress_St**—This bit indicates whether the TMRDACWR sequence initiated by an AO_UPDATE or by setting AO_Not_An_UPDATE to 1 has completed. **Note**: You can poll this bit if you want to wait for the DAC to load before arming the analog-output counters. The hardware ensures that if this bit reads 0 after writing AO_Not_An_UPDATE to 1, it means the hardware is done writing to all DACs. The values for this bitfield are in *OutTimer_TMRDACWRs_In_Progress_t*.
9	**UI_Q_St**—This field reflects the state of the AO_UI control circuit. The values for this bitfield are in *OutTimer_UI_Q_t*.
8	**UI_Count_Enabled_St**—If the AO_UI counter is armed, this bit indicates whether the AO_UI counter is enabled to count. If the counter is disarmed, this bit should be ignored. The values for this bitfield are in *OutTimer_UI_Count_Enabled_t*.
7	**UI_Load_Switch_Pending_St**—This bit indicates whether there is any switch pending in the load source for the UI counter. This only reflects a switch pending if the switch is not automatic (the switch was requested using the bitfields in the command register). The values for this bitfield are in *OutTimer_UI_Load_Switch_Pending_t*.
6	**UI_Next_Load_Source_St**—Reflects the status of the next load of the UI counter.
5	**UI_Armed_St**—This bit indicates whether the AO_UI counter is armed. The values for this bitfield are in *OutTimer_Armed_t*.
4	**BC_TC_Trigger_Error_St**—This bit indicates the detection of a BC_TC trigger error. A BC_TC trigger error occurs when a AO_START1 trigger is received after the last BC_TC of a staged waveform but before AO_BC_TC_Interrupt_Ack is set to 1. This allows you to detect triggers which arrive before completion of the ISR. You can clear this bit by setting AO_BC_TC_Trigger_Error_Confirm to 1. The values for this bitfield are in *OutTimer_Error_t*.
3	**BC_Q_St**—This field reflects the state of the AO_BC control circuit. The values for this bitfield are in *OutTimer_BC_Q_t*.
2	**Write_Too_Fast_St**—This bit indicates the detection of a Write_Too_Fast error, when a direct write is attempted before the previous write is updated by the timed update signal.
1	**BC_Next_Load_Source_St**—This bit reflects the state of the load source for the BC counter.
0	**BC_Armed_St**—This bit indicates whether the AO_BC counter is armed. The values for this bitfield are in *OutTimer_Armed_t*.

Offset 0x00: Command_1_Register (W)

Absolute Addresses:
AO: 0x20470
DO: 0x204E0

Bits	Name
31..30	**Reserved**
29	**Disarm** (Strobe)—Setting this bit to 1 asynchronously disarms the AO_BC, AO_UC, and AO_UI counters. This bit is cleared automatically.
28..27	**Reserved**
26	**UI_Arm** (Strobe)—Setting this bit to 1 arms the AO_UI counter. The counter remains armed, and the bit remains set, until it is disarmed either by hardware or by setting AO_Disarm to 1. **Note**: All Arm bits used for a given acquisition should be set on the same register write.
25	**UI_Load** (Strobe)—If the AO_UI counter is disarmed, this bit loads the AO_UI counter with the contents of the selected AO_UI load register (A or B). If the AO_UI counter is armed, writing to this bit has no effect. This bit is cleared automatically.
24	**UC_Arm** (Strobe)—This bit arms the AO_UC counter. The counter remains armed, and the bit remains set, until it is disarmed either by hardware or by setting AO_Disarm to 1. **Note**: All Arm bits used for a given acquisition should be set on the same register write.
23	**UC_Load** (Strobe)—If the AO_UC counter is disarmed, this bit loads the AO_UC counter with the contents of the selected AO_UC load register (A or B). If the AO_UC counter is armed, writing to this bit has no effect. This bit is cleared automatically.
22	**BC_Arm** (Strobe)—This bit arms the AO_BC counter. The counter remains armed, and the bit remains set, until it is disarmed either by hardware or by setting AO_Disarm to 1. **Note**: All Arm bits used for a given acquisition should be set on the same register write.
21	**BC_Load** (Strobe)—If the AO_BC counter is disarmed, this bit loads the AO_BC counter with the contents of the selected AO_BC load register. If the AO_BC counter is armed, writing to this bit has no effect. This bit is cleared automatically.
20..17	**Reserved**
16	**Update_Pulse** (Strobe)—Setting this bit to 1 produces a pulse on the AO_UPDATE output signals if the signals are enabled for output and if AO_UPDATE pulses are not blocked. AO_UPDATE pulses can be blocked by the external gate or by AO_Software_Gate. The pulsewidth of the output signals is determined by AO_UPDATE_Pulse_Width. This bit is cleared automatically. The AO timing engine must be armed for this bit to work.
15	**End_On_BC_TC** (Strobe)—Setting this bit to 1 causes the AO_BC, AO_UC, and AO_UI counters to be stopped, but not disarmed, at the next BC_TC. You can set this bit to stop waveform generation in the continuous mode so that the AO timing engine ends up in a retriggerable state. This action is internally synchronized to the falling edge of the AO_UC source. This bit is cleared automatically.

Bits	Name
14	**End_On_UC_TC** (Strobe)—Setting this bit to 1 causes the AO_BC, AO_UC, and AO_UI counters to be disarmed at the next UC_TC. You can set this bit to stop waveform generation in the continuous mode. This action is internally synchronized to the falling edge of the AO_UC source. This bit is cleared automatically.
13	**Reserved**
12	**UI_Cancel_Load_Switch** (Strobe)—Setting this bit to 1 causes any switch pending in the load source for the UI counter to be cancelled. This only works if the switch pending was caused by another strobe bit in this register (refer to related bitfields: UI_Switch_Load_On_BC_TC, UI_Switch_Load_On_UC_TC, UI_Switch_Load_On_TC, UC_Switch_Load_On_BC_TC, UC_Switch_Load_On_TC, BC_Switch_Load_On_TC).
11..10	**Reserved**
9	**UI_Switch_Load_On_BC_TC** (Strobe)—Setting this bit to 1 causes the AO_UI counter to switch load registers at the next BC_TC. You can use this bit to change the update rate during waveform generation at the end of the current MISB. This bit is cleared automatically. This bit has no effect if the UI_Reload_Mode is set to a value different than UI_Reload_No_Change.
8	**UI_Switch_Load_On_UC_TC** (Strobe)—Setting this bit to 1 causes the AO_UI counter to switch load registers on the next UC_TC. This bit is cleared automatically. You can use this bit to change the update rate during waveform generation at the end of the current buffer. This bit has no effect if the UI_Reload_Mode is set to a value different than UI_Reload_No_Change.
7	**UI_Switch_Load_On_TC** (Strobe)—Setting this bit to 1 causes the AO_UI counter to switch the load registers at the next UI_TC. You can use this bit to change the update rate during waveform generation at the next update event. This bit is cleared automatically. This bit has no effect if the UI_Reload_Mode is set to a value different than UI_Reload_No_Change.
6	**UC_Switch_Load_On_BC_TC** (Strobe)—Setting this bit to 1 causes the AO_UC counter to switch load registers at the next BC_TC. This bit is cleared automatically. This bit has no effect if the UC_Reload_Mode is set to a value different than No_Change.
5	**UC_Switch_Load_On_TC** (Strobe)—Setting this bit to 1 causes the AO_UC counter to switch load registers at the next UC_TC. This bit is cleared automatically. This bit has no effect if the UC_Reload_Mode is set to a value different than No_Change.
4	**BC_Switch_Load_On_TC** (Strobe)—Setting this bit to 1 causes the AO_BC counter to switch load registers at the next BC_TC. This bit is cleared automatically. This bit has no effect if the BC_Reload_Mode is set to a value different than BC_Reload_No_Change.
3..1	**Reserved**
0	**START1_Pulse** (Strobe)—Setting this bit to 1 sends a AO_START1 trigger to the AO_BC, AO_UC, and AO_UI counters if the AO_START1 software strobe is selected (AO_START1_Select is set to 0). This bit is cleared automatically.
Options: No Soft Copy	

Offset 0x04: UI_Save_Register (R)

Absolute Addresses:
AO: 0x20474
DO: 0x204E4

Bits	Name
31..0	**UI_Save**—This bitfield reflects the contents of the AO_UI counter. Reading from this bitfield while the AO_UI counter is counting can result in an erroneous value.

Offset 0x04: UI_Load_A_Register (W)

Absolute Addresses:
AO: 0x20474
DO: 0x204E4

Bits	Name
31..0	**UI_Load_A**—This bitfield is the load value A for the AO_UI counter. If load register A is the selected AO_UI load register, the AO_UI counter loads the value contained in this bitfield on AO_UI_Load and on UI_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AO_UI_Load (in the Command register).

Offset 0x08: UI_Load_B_Register (W)

Absolute Addresses:
AO: 0x20478
DO: 0x204E8

Bits	Name
31..0	**UI_Load_B**—This bitfield is the load value B for the AO_UI counter. If load register B is the selected AO_UI load register, the AO_UI counter loads the value contained in this bitfield on AO_UI_Load and on UI_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AO_UI_Load (in the Command register).

Offset 0x0C: UC_Save_Register (R)

Absolute Addresses:
AO: 0x2047C
DO: 0x204EC

Bits	Name
31..0	**UC_Save**—This bitfield reflects the contents of the UC counter. It is always a safe read. When AO_Hold_BC_On_UC_Read is zero, the read reflects the current state of the UC counter with no side effects. When AO_Hold_BC_On_UC_Read is one, reading this field will always reflect the state of the UC counter but will also have the side effect of freezing the BC counter, allowing a true snapshot of the state of the counters.

Offset 0x0C: UC_Load_A_Register (W)

Absolute Addresses:
AO: 0x2047C
DO: 0x204EC

Bits	Name
31..0	**UC_Load_A**—This bitfield is the load value A for the AO_UC counter. If load register A is the selected AO_UC load register, the AO_UC counter loads the value contained in this bitfield on AO_UC_Load and on UC_TC. **Caution:** To prevent register ready violations, this register should be written only once before the value is loaded through AO_UC_Load (in the Command register).

Offset 0x10: UC_Load_B_Register (W)

Absolute Addresses:
AO: 0x20480
DO: 0x204F0

Bits	Name
31..0	**UC_Load_B**—This bitfield is the load value B for the AO_UC counter. If load register B is the selected AO_UC load register, the AO_UC counter loads the value contained in this bitfield on AO_UC_Load and on UC_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AO_UC_Load (in the Command register).

Offset 0x14: BC_Save_Register (R)

Absolute Addresses:
AO: 0x20484
DO: 0x204F4

Bits	Name
31..0	**BC_Save**—This bitfield reflects the contents of the BC counter. It is always a safe read. When AO_Hold_BC_On_UC_Read is zero, the read reflects the current state of the BC counter. When AO_Hold_BC_On_UC_Read is one, the field will freeze after each AO_UC_Save read, allowing a true snapshot of the state of the counters.

Offset 0x14: BC_Load_A_Register (W)

Absolute Addresses:
AO: 0x20484
DO: 0x204F4

Bits	Name
31..0	**BC_Load_A**—This bitfield is the load value A for the AO_BC counter. If load register A is the selected AO_BC load register, the AO_BC counter loads the value contained in this bitfield on AO_BC_Load and on BC_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AO_BC_Load (in the Command register).

Offset 0x18: BC_Load_B_Register (W)

Absolute Addresses:
AO: 0x20488
DO: 0x204F8

Bits	Name
31..0	**BC_Load_B**—This bitfield is the load value B for the AO_BC counter. If load register B is the selected AO_BC load register, the AO_BC counter loads the value contained in this bitfield on the AO_BC_Load and on BC_TC. **Caution**: To prevent register ready violations, this register should be written only once before the value is loaded through AO_BC_Load (in the Command register).

Offset 0x1C: Mode_1_Register (W)

Absolute Addresses:
AO: 0x2048C
DO: 0x204FC

Bits	Name
31..29	**Reserved**
28..27	**UC_Reload_Mode**—This bit selects the reload mode for the AO_UC counter. The values for this bitfield are in *OutTimer_UC_Reload_Mode_t*.
26..22	**UI_Source_Select**—This bitfields selects the AO_UI source. The values for this bitfield are in *OutTimer_UI_Source_Select_t*.
21..20	**Reserved**
19	**UI_Source_Polarity**—This bit selects the active edge of the AO_UI source (the signal that is selected by AO_UI_Source_Select). The values for this bitfield are in *OutTimer_Polarity_t*.
18	**Reserved**
17	**Continuous**—This bit determines the behavior of the AO_BC, AO_UC, and AO_UI counters during BC_TC. The values for this bitfield are in *OutTimer_Continuous_t*.
16	**Trigger_Once**—Setting this bit to 1 causes the analog output timing sequence to stop on BC_TC. The AO_BC, AO_UC, and AO_UI counters are disarmed at this time.
15..14	**FIFO_Mode**—This bitfield selects the data FIFO condition on which to generate the FIFO interrupt. The values for this bitfield are in *OutTimer_FIFO_Mode_t*.
13	**FIFO_Retransmit_Enable**—This bit enables the local buffer mode. In the local buffer mode, the contents of the data FIFO are regenerated when the FIFO empties. The AO timing engine accomplishes this by pulsing the AOFFRT signal when the FIFO empty condition is indicated the AOFEF. You can use the local buffer mode when the FIFO is large enough to hold the whole waveform to be generated and the waveform does not vary in time. The values for this bitfield are in *OutTimer_Disabled_Enabled_t*.
12	**Reserved**
11	**UC_Initial_Load_Source**—If the AO_UC counter is disarmed, this bit selects the initial AO_UC load register. If the AO_UC counter is armed, writing to this bit has no effect. The values for this bitfield are in *OutTimer_Load_Source_t*.
10	**UC_Write_Switch**—This bit enables the write switch feature of the AO_UC load registers. Writes to AO_UC load register A are directed to the next available load register for writing (either A or B). The values for this bitfield are in *OutTimer_Write_Switch_t*.

Bits	Name
9..8	**Reserved**
7	**UI_Initial_Load_Source**—If the AO_UI counter is disarmed, this bit selects the initial AO_UI load register. If the AO_UI counter is armed, writing to this bit has no effect. The values for this bitfield are in *OutTimer_Load_Source_t*.
6..4	**UI_Reload_Mode**—This bitfield selects the reload mode for the AO_UI counter. The values for this bitfield are in *OutTimer_UI_Reload_Mode_t*.
3	**UI_Write_Switch**—This bit enables the write switch feature of the AO_UI load registers. Writes to AO_UI load register A are directed to the next available load register for writing (either A or B). The values for this bitfield are in *OutTimer_Write_Switch_t*.
2	**BC_Initial_Load_Source**—If the AO_BC counter is disarmed, this bit selects the initial AO_BC load register. If the AO_BC counter is armed, writing to this bit has no effect. The values for this bitfield are in *OutTimer_Load_Source_t*.
1	**BC_Reload_Mode**—This bit selects the reload mode for the AO_BC counter. You can use setting 1 in waveform staging to obtain a new buffer repetition count for each MISB. The values for this bitfield are in *OutTimer_BC_Reload_Mode_t*.
0	**BC_Write_Switch**—This bit enables the write switch feature of the AO_BC load registers. Writes to AO_BC load register A are directed to the next available load register for writing (either A or B). The values for this bitfield are in *OutTimer_Write_Switch_t*

Offset 0x20: Mode_2_Register (W)

Absolute Addresses:
AO: 0x20490
DO: 0x20500

Bits	Name
31	**Reserved**
30	**BC_Gate_Enable**—This bit enables the BC_GATE. Enabling the BC_GATE allows external AO_UPDATE pulses to pass only when the AO_BC counter is enabled to count. Set this bit to 0 in the internal AO_UPDATE mode (AO_UPDATE_Source_Select is set to 0) or if you are using AO_UPDATE_Pulse. Otherwise, set to 1. The values for this bitfield are in *OutTimer_Disabled_Enabled_t*.
29	**FIFO_Enable**—This bit enables the TMRDACWR signal to generate pulses after each AO_UPDATE. This pulses generate FIFO reads as well as DacWr signals at the pins or the serial interfaces. The values for this bitfield are in *OutTimer_Disabled_Enabled_t*.
28	**Reserved**
27	**Start1_Export_Mode**—This bit selects the signal appearing on the PFI pin when routed with the START1 signal for the correspondent instance of the Out timer. The values for this bitfield are in *OutTimer_Start1_Export_Mode_t*.
26	**Hold_BC_On_UC_Read**—When this bit is 1, every time the UC counter is read, the value of the BC counter is held until it is read in order to keep correlation between counter readings.
25..23	**Reserved**
22	**AOFREQ_Enable**—This bit enables the FIFO Request signal. This bitfield has no effect, since the request condition is determined directly by the Stream Circuit. The values for this bitfield are in *OutTimer_Disabled_Enabled_t*.
21	**Stop_On_Overrun_Error**—This bit determines whether analog output timing will stop when an overrun error occurs. If this bit is set and an overrun error is detected, the AO_UPDATE pulses will be masked off until the overrun error is cleared by the AO_Error_Interrupt_Ack bit. AO_Overrun_St is set in either case. The values for this bitfield are in *OutTimer_Stop_On_Error_t*.
20	**Stop_On_BC_TC_Trigger_Error**—This bit determines whether output timing stops when a BC_TC trigger error occurs. AO_BC_TC_Trigger_Error_St is set in either case. The values for this bitfield are in *OutTimer_Stop_On_Error_t*.

Bits	Name
19	**Stop_On_BC_TC_Error**—This bit determines whether output timing stops when a BC_TC error occurs. AO_BC_TC_Error_St is set in either case. The values for this bitfield are in *OutTimer_Stop_On_Error_t*.
18	**Not_An_UPDATE** (Strobe)—Setting this strobe bit causes the generation of an appropriate number of TMRDACWR pulses without generating any AO_UPDATE pulses. Use this bit during the AO configuration phase in the programming sequence to write the first point of the buffer into the DACs. For the TMRDACWR pulses to be generated, AO_FIFO_Enable must be set to 1 and the data FIFO must contain data.
17	**Software_Gate**—This bit controls the software gate, which can be used to pause an analog output operation. The values for this bitfield are in *OutTimer_Software_Gate_t*.
16..9	**Reserved**
8..7	**SyncMode**—This bitfield determines the mode of operation of trigger synchronization. The values for this bitfield are in *OutTimerSyncMode_t*.
6..4	**Reserved**
3	**Mute_B**—This bit determines whether the programmed buffer is a mute buffer. Set this bit to 0 if you want AO_UPDATE and related signals to be generated while the AO_BC counter is using load register B as the active load register. Set this bit to 1 if you want the DAQ-STC3 to suppress AO_UPDATE and related signals while the AO_BC counter is using load register B as the active load register. You can use the mute operation to obtain a pause between two real waveforms. You must set the AO_Mute_B bit to the correct value before the AO_BC counter begins using load register B. The values for this bitfield are in *OutTimer_Mute_t*.
2	**Mute_A**—This bit determines whether the programmed buffer is a mute buffer. Set this bit to 0 if you want AO_UPDATE and related signals to be generated while the AO_BC counter is using load register A as the active load register. Set this bit to 1 if you want the DAQ-STC3 to suppress AO_UPDATE and related signals while the AO_BC counter is using load register A as the active load register. You can use the mute operation to obtain a pause between two real waveforms. You must set the AO_Mute_A bit to the correct value before the AO_BC counter begins using load register A. The values for this bitfield are in *OutTimer_Mute_t*.
1..0	**Reserved**

Offset 0x26: Output_Control_Register (W)

Absolute Addresses:
AO: 0x20496
DO: 0x20506

Bits	Name
15..8	**Number_Of_Channels**—This bitfield determines the number of analog output channels that are written. The channel order is defined by the configuration FIFO.
7..1	**Reserved**
0	**ExportedUpdatePolarity**—This bitfield determines the polarity of the update exported to PFI, RTSI and all other subsystems. If active high, the rising edge is the one that updates. If active low, the falling edge is the significant one. The values for this bitfield are in *OutTimer_Polarity_t*.
Options: Initial Value = 0x1	

Offset 0x28: Interrupt1_Register (W)

Absolute Addresses:
AO: 0x20498
DO: 0x20508

Bits	Name
31..30	**Reserved**
29	**Error_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AO_Overrun_St and acknowledges the Error interrupt request if the Error interrupt is enabled. This bit is cleared automatically.
28..27	**Reserved**
26	**UPDATE_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AO_UPDATE_St and acknowledges the AO_UPDATE interrupt request if the AO_UPDATE interrupt is enabled. This bit is cleared automatically.
25	**START1_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AO_START1_St and acknowledges the AO_START1 interrupt request if the AO_START1 interrupt is enabled. This bit is cleared automatically.
24	**BC_TC_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AO_BC_TC_St and acknowledges the BC_TC interrupt request if the BC_TC interrupt is enabled. This bit is cleared automatically.
23	**UC_TC_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears AO_UC_TC_St and acknowledges the UC_TC interrupt request if the UC_TC interrupt is enabled. This bit is cleared automatically.
22..21	**Reserved**
20	**BC_TC_Error_Confirm** (Strobe)—Setting this bit to 1 clears AO_BC_TC_Error_St. This bit is cleared automatically.

Bits	Name
19	**BC_TC_Trigger_Error_Confirm** (Strobe)—Setting this bit to 1 clears AO_BC_TC_Trigger_Error_St. This bit is cleared automatically.
18..17	**Reserved**
16	**Write_Too_Fast_Interrupt_Ack** (Strobe)—Setting this bit to 1 clears the AO_Write_Too_Fast_St. This bit is cleared automatically.
15..10	**Reserved**
9	**Write_Too_Fast_Interrupt_Enable** (Strobe)—This bit strobe enables the AO_Write_Too_Fast interrupt. The Error AO_Write_Too_Fast interrupt is generated when a direct write is attempted before the previous write is updated by the timed update signal.
8	**FIFO_Interrupt_Enable** (Strobe)—This bit strobe enables the FIFO interrupt. The FIFO interrupt is generated on the FIFO condition indicated by AO_FIFO_Mode.
7	**Reserved**
6	**UC_TC_Interrupt_Enable** (Strobe)—This bit strobe enables the UC_TC interrupt. UC_TC interrupts are generated on the leading edge of UC_TC.
5	**Error_Interrupt_Enable** (Strobe)—This bit strobe enables the Error interrupt. The Error interrupt is generated on the detection of an overrun error condition.
4..3	**Reserved**
2	**UPDATE_Interrupt_Enable** (Strobe)—This bit strobe enables the AO_UPDATE interrupt. AO_UPDATE interrupts are generated on the trailing edge of AO_UPDATE.
1	**START1_Interrupt_Enable** (Strobe) This bit strobe enables the AO_START1 interrupt. The AO_START1 interrupt is generated on valid AO_START1 triggers received by the DAQ-STC3. A valid AO_START1 trigger is one that is received while the AO_BC counter is armed and in the WAIT1 state.
0	**BC_TC_Interrupt_Enable** (Strobe)—This bit strobe enables the BC_TC interrupt. BC_TC interrupts are generated on the trailing edge of BC_TC.
Options: No Soft Copy	

Offset 0x2C: Interrupt2_Register (W)

Absolute Addresses:
AO: 0x2049C
DO: 0x2050C

Bits	Name
31..30	**Reserved**
29	**Error_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AO_Overrun_St and acknowledges the Error interrupt request if the Error interrupt is enabled. This bit is cleared automatically.
28..27	**Reserved**
26	**UPDATE_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AO_UPDATE_St and acknowledges the AO_UPDATE interrupt request if the AO_UPDATE interrupt is enabled. This bit is cleared automatically.
25	**START1_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AO_START1_St and acknowledges the AO_START1 interrupt request if the AO_START1 interrupt is enabled. This bit is cleared automatically.
24	**BC_TC_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AO_BC_TC_St and acknowledges the BC_TC interrupt request if the BC_TC interrupt is enabled. This bit is cleared automatically.
23	**UC_TC_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears AO_UC_TC_St and acknowledges the UC_TC interrupt request if the UC_TC interrupt is enabled. This bit is cleared automatically.
22..21	**Reserved**
20	**BC_TC_Error_Confirm2** (Strobe)—Setting this bit to 1 clears AO_BC_TC_Error_St. This bit is cleared automatically.
19	**BC_TC_Trigger_Error_Confirm2** (Strobe)—Setting this bit to 1 clears AO_BC_TC_Trigger_Error_St. This bit is cleared automatically.
18..17	**Reserved**
16	**Write_Too_Fast_Interrupt_Ack2** (Strobe)—Setting this bit to 1 clears the AO_Write_Too_Fast_St. This bit is cleared automatically.
15..10	**Reserved**
9	**Write_Too_Fast_Interrupt_Disable** (Strobe)—This bit strobe disables the AO_Write_Too_Fast interrupt. The Error AO_Write_Too_Fast interrupt is generated when a direct write is attempted before the previous write is updated by the timed update signal.
8	**FIFO_Interrupt_Disable** (Strobe)—This bit strobe disables the FIFO interrupt. The FIFO interrupt is generated on the FIFO condition indicated by AO_FIFO_Mode.
7	**Reserved**
6	**UC_TC_Interrupt_Disable** (Strobe)—This bit strobe disables the UC_TC interrupt. UC_TC interrupts are generated on the leading edge of UC_TC.

Bits	Name
5	**Error_Interrupt_Disable** (Strobe)—This bit strobe disables the Error interrupt. The Error interrupt is generated on the detection of an overrun error condition.
4..3	**Reserved**
2	**UPDATE_Interrupt_Disable** (Strobe)—This bit strobe disables the AO_UPDATE interrupt. AO_UPDATE interrupts are generated on the trailing edge of AO_UPDATE.
1	**START1_Interrupt_Disable** (Strobe)—This bit strobe disables the AO_START1 interrupt. The AO_START1 interrupt is generated on valid AO_START1 triggers received by the DAQ-STC3. A valid AO_START1 trigger is one that is received while the AO_BC counter is armed and in the WAIT1 state.
0	**BC_TC_Interrupt_Disable** (Strobe)—This bit strobe disables the BC_TC interrupt. BC_TC interrupts are generated on the trailing edge of BC_TC.
Options: No Soft Copy	

Offset 0x34: Reset_Register (W)

Absolute Addresses:
AO: 0x204A4
DO: 0x20514

Bits	Name
15..4	**Reserved**
3	**FIFO_Clear** (Strobe)—Writing to this register pulses the DAC_FIFO_Clear signal. Following a write to this bitfield a read (or bus flush) must occur to ensure that the FIFO reset has completed before passing data into the FIFO.
2	**Configuration_End** (Strobe)—This bit clears Configuration_Start, which holds the analog output circuitry in reset to prevent glitches on the output pins during configuration. Set this bit to 1 at the end of the configuration process of the analog output circuitry. This bit is cleared automatically.
1	**Configuration_Start** (Strobe)—This bit holds the analog output circuitry in reset to prevent glitches on the output pins during configuration. You should set this bit to 1 when beginning the configuration process of the analog output circuitry. By doing this, you ensure that no spurious glitches appear on the output pins and on the internal circuit components. If you do not set this bit to 1, the DAQ-STC3 can behave erroneously. You can clear this bit by setting Configuration_End to 1.
0	**Reset** (Strobe)—Setting this bit to 1 resets the OutTimer. Setting this bit to 1 also clears all the status bits and interrupts related to the subsystem except those associated with the data FIFO. This bit is cleared automatically. **Caution**: Using this bitfield to reset the timer will also activate the ConfigReset. A strobe write to the Configuration_End bitfield must be executed after resetting the timer with this bitfield (in a separate register access).
Options: No Soft Copy	

Enumerated Types

OutTimer_Armed_t

Value	Name
0	**Disarmed**
1	**Armed**

OutTimer_BC_Gate_t

Value	Name
0	**BC_Gate_Inactive**—External AO_UPDATEs are blocked.
1	**BC_Gate_Active**—External AO_UPDATEs are allowed to pass.

OutTimer_BC_Q_t

Value	Name
0	**BC_St_WAIT**
1	**BC_St_CNT**

OutTimer_BC_Reload_Mode_t

Value	Name
0	**BC_Reload_No_Change**—No automatic change of the AO_BC load register.
1	**BC_Reload_Switch_On_BC_TC**—The AO_BC counter switches load registers on BC_TC.

OutTimer_Continuous_t

Value	Name
0	**FiniteOp**—Counters stop on BC_TC.
1	**ContinuousOp**—Counters ignore BC_TC. The counters remain armed and generate AO_UPDATE pulses until an AO_End_On_BC_TC or AO_End_On_UC_TC command is given, until the AO timing engine is reset using AO_Reset or until an AO_Trigger_Once command is issued.

OutTimer_Disabled_Enabled_t

Value	Name
0	**Disabled**
1	**Enabled**

OutTimer_Error_t

Value	Name
0	**No_error**
1	**Error**

OutTimer_External_Gate_t

Value	Name
0	**ExtGate_Pause_Operation**
1	**ExtGate_Enable_Operation**

OutTimer_FIFO_Empty_t

Value	Name
0	**Not_empty**
1	**Empty**

OutTimer_FIFO_Full_t

Value	Name
0	**Not_full**
1	**Full**

OutTimer_FIFO_Half_Full_t

Value	Name
0	**Half_full_or_less**
1	**More_than_half_full**

OutTimer_FIFO_Mode_t

Value	Name
0	**FifoMode_Empty**—On empty FIFO.
1	**FifoMode_Less_Than_Half_Full**—On half-full or less FIFO.
2	**FifoMode_Less_Than_Full**—On less than full FIFO.
3	**FifoMode_Less_Than_Half_Full_to_Full**—Generate on half-full or less FIFO, but keep asserted until FIFO is full.

OutTimer_Load_Source_t

Value	Name
0	**Reg_A**—Load register A.
1	**Reg_B**—Load register B.

OutTimer_Mute_t

Value	Name
0	**Normal_buffer**
1	**Mute_buffer**

OutTimer_Polarity_t

Value	Name
0	**Rising_Edge**
1	**Falling_Edge**

OutTimer_Software_Gate_t

Value	Name
0	**SwGate_Enable_operation**
1	**SwGate_Pause_operation**

OutTimer_Start1_Export_Mode_t

Value	Name
0	**ExportSynchronizedTriggers**—Output the internal signal. This version will be synchronized to the UI source, with a pulse with of 16 cycles of TB3 if the UI source is TB3, and one UI source period otherwise.
1	**ExportEdgeDetectedTriggers**—In this case, the signal is edge detected and pulse stretched. The width of the pulse can be 17–18 TB3 clock cycles.

OutTimer_Stop_On_Error_t

Value	Name
0	**Continue_on_Error**
1	**Stop_on_Error**

OutTimer_TMRDACWRs_In_Progress_t

Value	Name
0	**WritesCompleted**
1	**WritesInProgress**

OutTimer_Trigger_t

Value	Name
0	**Has_Not_Happened**
1	**Has_Happened**

OutTimer_UC_Q_t

Value	Name
0	**UC_Idle**
1	**UC_Counting**

OutTimer_UC_Reload_Mode_t

Value	Name
0	**No_Change**—No_Change.
1	**Switch_Every_UC_TC**—Switch on every UC_TC.
2	**Switch_Every_BC_TC**—Switch on every BC_TC.
3	**Alternate_First_Period_Every_BC_TC**—Alternate first period every BC_TC.

OutTimer_UI_Count_Enabled_t

Value	Name
0	**UI_CountNotEnabled**
1	**UI_CoundIsEnabled**

OutTimer_UI_Load_Switch_Pending_t

Value	Name
0	**No_Switch_Pending**—There is no load switch pending in the UI counter.
1	**Switch_Pending**—There is a load switch pending in the UI counter.

OutTimer_UI_Q_t

Value	Name
0	**UI_St_WAIT**
1	**UI_St_CNT**

OutTimer_UI_Reload_Mode_t

Value	Name
0	**UI_Reload_No_Change**—No automatic change of the AO_UI load register.
4	**UI_Reload_Switch_On_UC_TC_First**—Alternate first period on UC_TC. Use this setting to make the time interval between the AO_START trigger and the first AO_UPDATE pulse different from the remaining AO_UPDATE intervals.
5	**UI_Reload_Switch_On_UC_TC**—Switch load register on UC_TC. Use this setting to synchronously change the AO_UPDATE interval at each UC_TC.
6	**UI_Reload_Switch_On_BC_TC_First**—Alternate first period on BC_TC. Use this setting to make the time interval between the AO_START1 trigger and the first AO_UPDATE pulse different from the remaining AO_UPDATE intervals.
7	**UI_Reload_Switch_On_BC_TC**—Switch load register on BC_TC. Use this setting to synchronously change the AO_UPDATE interval at each BC_TC. This is convenient for staged analog output operation.

OutTimer_UI_Source_Select_t

Value	Meaning	Value	Meaning
0	**UI_Src_TB3**	16	**UI_Src_RTSI5**
1	**UI_Src_PFI0**	17	**UI_Src_RTSI6**
2	**UI_Src_PFI1**	18	**UI_Src_DStarA**
3	**UI_Src_PFI2**	19	**UI_Src_TB2**
4	**UI_Src_PFI3**	20	**UI_Src_Star_Trigger**
5	**UI_Src_PFI4**	21	**UI_Src_PFI10**
6	**UI_Src_PFI5**	22	**UI_Src_PFI11**
7	**UI_Src_PFI6**	23	**UI_Src_PFI12**
8	**UI_Src_PFI7**	24	**UI_Src_PFI13**
9	**UI_Src_PFI8**	25	**UI_Src_PFI14**
10	**UI_Src_PFI9**	26	**UI_Src_PFI15**
11	**UI_Src_RTSI0**	27	**UI_Src_RTSI7**
12	**UI_Src_RTSI1**	28	**UI_Src_TB1**
13	**UI_Src_RTSI2**	29	**UI_Src_PXI_Clk10**
14	**UI_Src_RTSI3**	30	**UI_Src_Analog_Trigger**
15	**UI_Src_RTSI4**	31	**UI_Src_DStarB**

OutTimer_Write_Switch_t

Value	Name
0	**Write_A**—Unconditionally directed to load register A.
1	**Write_Inactive_Register**—Directed to the inactive load register.

OutTimerSyncMode_t

Value	Name
0	**SyncDefault**—This is the default mode of operation. The trigger is synchronized to the source of the timer for internal use only.
1	**SyncSlave**—In this mode, the timer is setup to receive the exported trigger from a master device. The transmission and reception of this trigger depends on the SyncPulse signal (PXIe_Sync100 on PXI Express systems).
2	**SyncMaster**—In this mode, the timer transmits the trigger and then waits to use it so that all devices (master and slaves) trigger at the same time.

19

Stream Circuit Registers

AI base address = 0x24000
G0 base address = 0x26000
G1 base address = 0x28000
G2 base address = 0x2A000
G3 base address = 0x2C000
DI base address = 0x2E000
AO base address = 0x30000
DO base address = 0x32000

Each register is labeled as a read (R) or write (W) register.

List of Stream Circuit Registers

The Stream Circuit registers are as follows:

Stream Circuit Registers

Offset 0x800: StreamControlStatusReg (R|W)

Absolute Addresses:
AI: 0x24800
G0: 0x26800
G1: 0x28800
G2: 0x2A800
G3: 0x2C800
DI: 0x2E800
AO: 0x30800
DO: 0x32800

Bits	Name
31..19	**Reserved**
18	**FifoEmpty**—This read-only register returns a 1 when the FIFO is empty.
17..16	**Reserved**
15	**InvalidPacketClear** (Strobe)—Writing a 1 to this bit clears the InvalidPacketFlag.
14	**InvalidPacketFlag**—This read-only bit returns a 1 when a packet received by the Stream Circuit is invalid. This happens when an incoming packet references the wrong Byte lane or transfers too much data for the resources available in the FIFO. For Input Stream Circuits, an incoming Split Read Request has a Read Response returned with a Completion Status of Failure, plus no payload. For Output Stream Circuits, an incoming Posted Write Request or Read Response is received but ignored. This bit clears on reset or when any of the StreamCircuit Reset, StreamCircuitResetNotify, or RejectedPacketClear bits is written with a 1.
13	**DoneFlagSet** (Strobe)—This bit is only implemented for output Stream Circuits. Writing a 1 to this bit sets the Done flag, which indicates that no more data should be received. The Done flag is also be set when a stream packet is received with the Done bit set. The Done flag allows software to know when it is safe to reset the Stream Circuit. This bit returns a 1 when the done flag is set and a 0 when it is clear. The Done Flag is cleared on reset and when either of the StreamCircuitReset or StreamCircuitResetNotify bits is written with a 1. This bit is only implemented for Output Stream Circuits.
12	**Reserved**
11	**StreamCircuitResetNotify** (Strobe)—Writing a 1 to this bit performs the same function as the StreamCircuitReset bit. The difference is this bit generates a stream Done event after the Stream Circuit is reset. This allows software to be notified when the entire reset process is complete. Use of this bit with a CHInCh DMA channel requires the associated DMA channel in the CHInCh to be in the started state so that the done event can be processed. This bit clears itself.

Bits	Name
10	**StreamCircuitResetComplete**—This read-only bit indicates when a stream reset operation is complete. After either of the StreamCircuitReset or StreamCircuitResetNotify bits is written with a 1, this status bit does not set until all the following conditions are true: • The Stream Circuit in the Endpoint Interface has been reset. • All transactions associated with the Stream Circuit have been completed by the targets. This bit sets on reset to indicate that the Stream Circuit is reset. This bit clears whenever the DataTransferEnable bit is written with a 1.
9	**StreamCircuitReset** (Strobe)—When this bit is written with a 1, the entire Stream Circuit is synchronously reset. This can be used to initialize the Stream Circuit to a known state. The FIFO and application logic is not reset. This bit clears itself.
8	**DataTransferEnable** (Strobe)—Writing a 1 to this bit enables the Stream Circuit to initiate data transfers. This bit returns a 1 when data transfers are enabled and a 0 when they are disabled. This bit clears on reset and when either of the StreamCircuitReset or StreamCircuitResetNotify bits is written with a 1. For input streams, writes are not initiated before this bit is written with a 1. For output streams, writing a 1 to this bit causes the Stream Circuit to begin sending out request packets. When data transfers are not enabled, both input and output streams can still service requests as long as the Stream Circuit is not in the process of begin reset.
7..4	**Reserved**
3	**CISATCR_AccessIgnored**—This read-only bit indicates when a write to the StreamAdditiveTransferCountReg (CISATCR) has been ignored. Writes to CISATCR result in 32-bit signed additions. If the result of an access is negative, the register is not affected by the write and this status bit sets. This bit updates on each write to CISATCR so it always reflects the last access.
2	**CISTCR_Clear** (Strobe)—Writing a 1 to this bit clears the StreamTransferCountReg (CISTCR). Generally, this should be done when initializing a new operation. This bit is the only way to get the CISTCR into a known state since there is no way to write to it directly. The count value can only be written additively.
1	**CISTCR_Disable** (Strobe)—Writing a 1 to this bit disables the StreamTransferCountReg (CISTCR). This bit returns a 0 when the CISTCR is enabled and 1 when it is disabled.
0	**CISTCR_Enable** (Strobe)—Writing a 1 to this bit enables the StreamTransferCountReg (CISTCR). This bit returns a 1 when the CISTCR is enabled and a 0 when it is disabled. The CISTCR is disabled at reset. When the CISTCR is enabled, data transfers are limited by the count. When the CISTCR is disabled, the circuit ignores its value and data transfers are initiated indefinitely. Disabling CISTCR can be useful for finite transfers of an unknown size or when pre-triggering in host memory. However, when the CISTCR is disabled, pack network congestion can result.
Options: No Soft Copy	

Offset 0x810: StreamTransferCountReg (R)

Absolute Addresses:
AI: 0x24810
G0: 0x26810
G1: 0x28810
G2: 0x2A810
G3: 0x2C810
DI: 0x2E810
AO: 0x30810
DO: 0x32810

Bits	Name
31..0	**StreamTransferCount**—This register limits the number of Bytes that are transferred with the Stream Circuit. As the Endpoint Interface initiates transfers in Stream space, the value in this register is decreased by the amount of data requested. The device never requests more Bytes than the amount indicated by the current value of the register. When the register reaches 0, the data stream is paused. The Endpoint Interface is designed to transfer data until this register reaches 0. It will not pause a few Bytes sooner due to data alignment or any other simplification. This is intended for DMA progress control. The Endpoint Interface is designed to allow writes to this field to occur while the data stream is active. When reading this field, the value returned is not guaranteed to be coherent with DMA data transfers in host memory. This register resets to an unknown value and should be initialized with the CISTCR_Clear bit prior to or in the same access in which the DataTransferEnable bit is set.
Options: No Soft Copy	

Offset 0x810: StreamAdditiveTransferCountReg (W)

Absolute Addresses:
AI: 0x24810
G0: 0x26810
G1: 0x28810
G2: 0x2A810
G3: 0x2C810
DI: 0x2E810
AO: 0x30810
DO: 0x32810

Bits	Name
31..0	**StreamAdditiveTransferCount**—The value written to this field is added to the current value of the Stream Transfer Count Register (CISTCR). This field is interpreted as a 32-bit signed value. The units are Bytes. Results of writing to this register are accurate regardless of the timing of the write relative to the processing of data in the stream. Therefore, the Endpoint Interface handles the case that the write occurs at the same time that hardware is decreasing CISTCR due to the issuing of a new stream request. Software is responsible for ensuring that any writes to this field do not cause the CISTCR to overflow. In most applications software maintains status about DMA progress. This same status can be used for ensuring that writes to this register do not overflow CISTCR. This field should always be written with a 32-bit access.
Options: No Soft Copy	

Offset 0x814: StreamFifoSizeReg (R)

Absolute Addresses:
AI: 0x24814
G0: 0x26814
G1: 0x28814
G2: 0x2A814
G3: 0x2C814
DI: 0x2E814
AO: 0x30814
DO: 0x32814

Bits	Name
31..0	**StreamFifoSize**—This is the size of the Stream Circuit's data FIFO in Bytes. It is an actual size in Bytes, though the last few Bytes may not be usable by a Stream Circuit depending on the sample width. To find the number of samples that the FIFO can store, divide this value by the number of Bytes per sample and round down.

Offset 0x81C: StreamTransferLimitReg (R|W)

Absolute Addresses:
AI: 0x2481C
G0: 0x2681C
G1: 0x2881C
G2: 0x2A81C
G3: 0x2C81C
DI: 0x2E81C
AO: 0x3081C
DO: 0x3281C

Bits	Name
31..16	**StreamMaxPayloadSize**—The value of this field is the largest payload size that can be used by the Stream Circuit when sending out request packets. This field defaults to the maximum on power-up.
15..0	**StreamMinPayloadSize**—The value of this field is the smallest payload size that can be used by the Stream Circuit when sending out request packets. The exception to this is when the StreamTransferCountReg (CISTCR) is below this value or eviction occurs. When this happens, the remainder of the available data is transferred. To ensure that data transfers occur, program this value with a value less than or equal to the StreamMaxPayloadSize value so that requests will actually happen. This field defaults to the maximum on power-up for high throughput.
Options: No Hardware Reset	

Offset 0x820: StreamEvictionReg (R|W)

Absolute Addresses:
AI: 0x24820
G0: 0x26820
G1: 0x28820
G2: 0x2A820
G3: 0x2C820
DI: 0x2E820
AO: 0x30820
DO: 0x32820

Bits	Name
31	**DisableEviction**—When set, the eviction timer is disabled, and StreamMinPayloadSize is always considered. When cleared, the eviction timer triggers data communications even less than StreamMinPayloadSize Bytes can be transferred.
30..10	**Reserved**
9..0	**EvictionTime**—The value to preload into the eviction timer. This defaults to 0x20, or about one microsecond for a 31.25 MHz bus clock.
Options: No Hardware Reset	

Offset 0x824: StreamTransactionLimitReg (R|W)

Absolute Addresses:
AI: 0x24824
G0: 0x26824
G1: 0x28824
G2: 0x2A824
G3: 0x2C824
DI: 0x2E824
AO: 0x30824
DO: 0x32824

Bits	Name
31..16	**Reserved**
15..8	**MaxTransactionLimit**—This read-only field tells how many transactions the Output Stream Circuit is capable of simultaneously generating.
7..0	**TransactionLimit**—This field controls the number of transactions the Output Stream Circuit is capable of simultaneously generating. Valid values for this field range from 1 to MaxTransactionLimit. Do not write to this field while data transfers are enabled with DataTransferEnable.
Options: No Hardware Reset	

Offset 0x904: DMAChannel (R|W)

Absolute Addresses:
AI: 0x24904
G0: 0x26904
G1: 0x28904
G2: 0x2A904
G3: 0x2C904
DI: 0x2E904
AO: 0x30904
DO: 0x32904

Bits	Name
31..22	**Reserved**
21..12	**DMAChannelNumber**—This field provides the DMA Channel Number to use.
11..0	**Reserved**

A

Register Maps Appendix

This appendix contains all of the register maps mentioned in this manual.

CHInCh Chip Object

0x00000	CHInCh	CHInCh_Identification_Register
0x00010	CHInCh	IO_Port_Resource_Description_Register
0x0005C	CHInCh	Interrupt_Mask_Register
0x00060	CHInCh	Interrupt_Status_Register
0x00068	CHInCh	Volatile_Interrupt_Status_Register
0x000A4	CHInCh	Host_Bus_Resource_Control_Register
0x000C0	CHInCh	EEPROM_Window_Register
0x000C4	CHInCh	Simultaneous_Window_Register
0x000E0	CHInCh	Window_Control_Register
0x00200	CHInCh	Scrap_Register
0x00514	CHInCh	Configuration_Register
0x00580	CHInCh	EEPROM_Register_0
0x00584	CHInCh	EEPROM_Register_1
0x0058C	CHInCh	EEPROM_Register_2
0x00590	CHInCh	SMIO_Register_0
0x00594	CHInCh	SMIO_Register_1
0x00598	CHInCh	SMIO_Register_2

0x0059C	CHInCh	SMIO_Register_3
0x010AC	CHInCh	PCI_SubSystem_ID_Access_Register

AI DMAController

0x02038	AI DMAController	Channel_Memory_Address_Register_LSW
0x0203C	AI DMAController	Channel_Memory_Address_Register_MSW
0x02048	AI DMAController	Channel_Link_Address_Register_LSW
0x0204C	AI DMAController	Channel_Link_Address_Register_MSW
0x02050	AI DMAController	Channel_Link_Size_Register
0x02054	AI DMAController	Channel_Control_Register
0x02058	AI DMAController	Channel_Operation_Register
0x02060	AI DMAController	Channel_Status_Register
0x02068	AI DMAController	Channel_Volatile_Status_Register
0x02090	AI DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02094	AI DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW
0x020A0	AI DMAController	Channel_Total_Transfer_Count_Status_Register_LSW
0x020A4	AI DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x020A8	AI DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x020AC	AI DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

Counter 0 DMAController

0x02138	G0 DMAController	Channel_Memory_Address_Register_LSW
0x0213C	G0 DMAController	Channel_Memory_Address_Register_MSW
0x02148	G0 DMAController	Channel_Link_Address_Register_LSW
0x0214C	G0 DMAController	Channel_Link_Address_Register_MSW

0x02150	G0 DMAController	Channel_Link_Size_Register
0x02154	G0 DMAController	Channel_Control_Register
0x02158	G0 DMAController	Channel_Operation_Register
0x02160	G0 DMAController	Channel_Status_Register
0x02168	G0 DMAController	Channel_Volatile_Status_Register
0x02190	G0 DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02194	G0 DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW
0x021A0	G0 DMAController	Channel_Total_Transfer_Count_Status_Register_LSW
0x021A4	G0 DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x021A8	G0 DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x021AC	G0 DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

Counter 1 DMAController

0x02238	G1 DMAController	Channel_Memory_Address_Register_LSW
0x0223C	G1 DMAController	Channel_Memory_Address_Register_MSW
0x02248	G1 DMAController	Channel_Link_Address_Register_LSW
0x0224C	G1 DMAController	Channel_Link_Address_Register_MSW
0x02250	G1 DMAController	Channel_Link_Size_Register
0x02254	G1 DMAController	Channel_Control_Register
0x02258	G1 DMAController	Channel_Operation_Register
0x02260	G1 DMAController	Channel_Status_Register
0x02268	G1 DMAController	Channel_Volatile_Status_Register
0x02290	G1 DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02294	G1 DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW
0x022A0	G1 DMAController	Channel_Total_Transfer_Count_Status_Register_LSW

0x022A4	G1 DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x022A8	G1 DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x022AC	G1 DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

Counter 2 DMAController

0x02338	G2 DMAController	Channel_Memory_Address_Register_LSW
0x0233C	G2 DMAController	Channel_Memory_Address_Register_MSW
0x02348	G2 DMAController	Channel_Link_Address_Register_LSW
0x0234C	G2 DMAController	Channel_Link_Address_Register_MSW
0x02350	G2 DMAController	Channel_Link_Size_Register
0x02354	G2 DMAController	Channel_Control_Register
0x02358	G2 DMAController	Channel_Operation_Register
0x02360	G2 DMAController	Channel_Status_Register
0x02368	G2 DMAController	Channel_Volatile_Status_Register
0x02390	G2 DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02394	G2 DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW
0x023A0	G2 DMAController	Channel_Total_Transfer_Count_Status_Register_LSW
0x023A4	G2 DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x023A8	G2 DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x023AC	G2 DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

Counter 3 DMAController

0x02438	G3 DMAController	Channel_Memory_Address_Register_LSW
0x0243C	G3 DMAController	Channel_Memory_Address_Register_MSW
0x02448	G3 DMAController	Channel_Link_Address_Register_LSW

0x0244C	G3 DMAController	Channel_Link_Address_Register_MSW
0x02450	G3 DMAController	Channel_Link_Size_Register
0x02454	G3 DMAController	Channel_Control_Register
0x02458	G3 DMAController	Channel_Operation_Register
0x02460	G3 DMAController	Channel_Status_Register
0x02468	G3 DMAController	Channel_Volatile_Status_Register
0x02490	G3 DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02494	G3 DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW
0x024A0	G3 DMAController	Channel_Total_Transfer_Count_Status_Register_LSW
0x024A4	G3 DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x024A8	G3 DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x024AC	G3 DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

DI DMAController

0x02538	DI DMAController	Channel_Memory_Address_Register_LSW
0x0253C	DI DMAController	Channel_Memory_Address_Register_MSW
0x02548	DI DMAController	Channel_Link_Address_Register_LSW
0x0254C	DI DMAController	Channel_Link_Address_Register_MSW
0x02550	DI DMAController	Channel_Link_Size_Register
0x02554	DI DMAController	Channel_Control_Register
0x02558	DI DMAController	Channel_Operation_Register
0x02560	DI DMAController	Channel_Status_Register
0x02568	DI DMAController	Channel_Volatile_Status_Register
0x02590	DI DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02594	DI DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW

0x025A0	DI DMAController	Channel_Total_Transfer_Count_Status_Register_LSW
0x025A4	DI DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x025A8	DI DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x025AC	DI DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

AO DMAController

0x02638	AO DMAController	Channel_Memory_Address_Register_LSW
0x0263C	AO DMAController	Channel_Memory_Address_Register_MSW
0x02648	AO DMAController	Channel_Link_Address_Register_LSW
0x0264C	AO DMAController	Channel_Link_Address_Register_MSW
0x02650	AO DMAController	Channel_Link_Size_Register
0x02654	AO DMAController	Channel_Control_Register
0x02658	AO DMAController	Channel_Operation_Register
0x02660	AO DMAController	Channel_Status_Register
0x02668	AO DMAController	Channel_Volatile_Status_Register
0x02690	AO DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02694	AO DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW
0x026A0	AO DMAController	Channel_Total_Transfer_Count_Status_Register_LSW
0x026A4	AO DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x026A8	AO DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x026AC	AO DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

DO DMAController

0x02738	DO DMAController	Channel_Memory_Address_Register_LSW
0x0273C	DO DMAController	Channel_Memory_Address_Register_MSW

0x02748	DO	DMAController	Channel_Link_Address_Register_LSW
0x0274C	DO	DMAController	Channel_Link_Address_Register_MSW
0x02750	DO	DMAController	Channel_Link_Size_Register
0x02754	DO	DMAController	Channel_Control_Register
0x02758	DO	DMAController	Channel_Operation_Register
0x02760	DO	DMAController	Channel_Status_Register
0x02768	DO	DMAController	Channel_Volatile_Status_Register
0x02790	DO	DMAController	Channel_Total_Transfer_Count_Compare_Register_LSW
0x02794	DO	DMAController	Channel_Total_Transfer_Count_Compare_Register_MSW
0x027A0	DO	DMAController	Channel_Total_Transfer_Count_Status_Register_LSW
0x027A4	DO	DMAController	Channel_Total_Transfer_Count_Status_Register_MSW
0x027A8	DO	DMAController	Channel_Total_Transfer_Count_Latching_Register_LSW
0x027AC	DO	DMAController	Channel_Total_Transfer_Count_Latching_Register_MSW

SimultaneousControl

0x06000		SimultaneousControl	SignatureYear
0x06001		SimultaneousControl	SignatureMonth
0x06002		SimultaneousControl	SignatureDay
0x06003		SimultaneousControl	SignatureHour
0x0600C		SimultaneousControl	Scratch
0x0600D		SimultaneousControl	AISetChannelOrder
0x0600E		SimultaneousControl	AIChanConfigCtrlStat
0x0600F		SimultaneousControl	AIClearChannelOrder
0x06010		SimultaneousControl	AITriggerConfigCtrlStat
0x06014		SimultaneousControl	AiFifoCtrlStat

0x06015	SimultaneousControl	LoopbackCtrlStat
0x06016	SimultaneousControl	LoopbackSourceSel
0x06017	SimultaneousControl	TempSensorCtrlStat
0x06018	SimultaneousControl	TempSensorDataHi
0x06019	SimultaneousControl	TempSensorDataLo
0x0601A	SimultaneousControl	InterruptControl
0x0601A	SimultaneousControl	InterruptStatus
0x0601B	SimultaneousControl	AcquisitionCtrl
0x0601C	SimultaneousControl	DcmCtrlStat

Board Services

0x20004	Board Services	ScratchPadRegister
0x20010	Board Services	SCXI_Serial_Data_In_Register
0x20010	Board Services	SCXI_Serial_Data_Out_Register
0x20012	Board Services	SCXI_Control_Register
0x20014	Board Services	SCXI_Output_Enable_Register
0x20016	Board Services	SCXI_Status_Register
0x20016	Board Services	SCXI_Mux_Clock_Register
0x20018	Board Services	GenPwmPageSpec_i (i) Array
0x20024	Board Services	Gen_PWM_i (i) Array
0x20060	Board Services	Signature_Register
0x20064	Board Services	TimeSincePowerUpRegister
0x20064	Board Services	Joint_Reset_Register
0x20068	Board Services	WatchdogStatusRegister
0x20068	Board Services	WatchdogTimeoutRegister

0x2006C	Board Services	WatchdogConfiguration Register
0x2006E	Board Services	WatchdogControl Register
0x20070	Board Services	Gen_Interrupt1_Register

Bus Interface

0x20070	Bus Interface	GlobalInterruptStatus_Register
0x20072	Bus Interface	AI_Interrupt_Status_Register
0x20074	Board Services	Gen_Interrupt2_Register
0x20074	Bus Interface	AO_Interrupt_Status_Register
0x20076	Bus Interface	TIO_Interrupt_Status_Register (i) Array
0x20078	Bus Interface	GlobalInterruptEnable_Register
0x2007E	Bus Interface	DI_Interrupt_Status_Register
0x20080	Bus Interface	DO_Interrupt_Status_Register
0x20086	Bus Interface	Gen_Interrupt_Status_Register

Triggers

0x200A0	Triggers	AnalogTrigControlRegister
0x200A2	Triggers	FOUT_Register
0x200A4	Triggers	PFI_Direction_Register
0x200A6	Triggers	RTSI_Trig_Direction_Register
0x200A8	Triggers	RTSI_OutputSelectRegister_i (i) Array
0x200B0	Triggers	PFI_Filter_Register_0
0x200B2	Triggers	PFI_Filter_Register_1
0x200B4	Triggers	PFI_Filter_Register_2
0x200B6	Triggers	PFI_Filter_Register_3

0x200B8	Triggers	STAR_Trig_Register
0x200BA	Triggers	PFI_OutputSelectRegister_i (i) Array
0x200CA	Triggers	DStarC_Trig_Register
0x200DA	Triggers	Clock_And_Fout2_Register
0x200DC	Triggers	PLL_Status_Register
0x200DC	Triggers	PLL_Control_Register
0x200E0	Triggers	PFI_DI_Register
0x200E0	Triggers	PFI_DO_Register
0x200E2	Triggers	PFI_WDT_SafeStateRegister
0x200E4	Triggers	PFI_WDT_ModeSelect_Register
0x200E8	Triggers	IntTriggerA_OutputSelectRegister_i (i) Array
0x200F8	Triggers	IntTrigA_Filter_Register_Lo
0x200FA	Triggers	IntTrigA_Filter_Register_Hi
0x20100	Triggers	Trig_Filter_Settings1_Register
0x20102	Triggers	Trig_Filter_Settings2_Register
0x20104	Triggers	PLL_LockCount_Register
0x20106	Triggers	Sync100_Repeat_Count_Register

Analog Input

0x20270	Analog Input	AI_Config_FIFO_Status_Register
0x20274	Analog Input	AI_Data_FIFO_Status_Register
0x20278	Analog Input	AI_FIFO_Data_Register
0x20278	Analog Input	AI_FIFO_Data_Register16
0x2028E	Analog Input	AI_Config_FIFO_Data_Register
0x20298	Analog Input	AI_Data_Mode_Register

0x2029C	Analog Input	AI_Trigger_Select_Register
0x202A0	Analog Input	AI_Trigger_Select_Register2

AI InTimer

0x202B0	AI InTimer	Status_1_Register
0x202B0	AI InTimer	Command_Register
0x202B4	AI InTimer	Status_2_Register
0x202B4	AI InTimer	Mode_1_Register
0x202B8	AI InTimer	SI_Save_Register
0x202B8	AI InTimer	Mode_2_Register
0x202BC	AI InTimer	SC_Save_Register
0x202BC	AI InTimer	SI_Load_A_Register
0x202C0	AI InTimer	SI2_Save_Register
0x202C0	AI InTimer	SI_Load_B_Register
0x202C4	AI InTimer	DIV_Save_Register
0x202C4	AI InTimer	SC_Load_A_Register
0x202C8	AI InTimer	SC_PreWaitCntRegister
0x202C8	AI InTimer	SC_Load_B_Register
0x202CC	AI InTimer	SI2_Load_A_Register
0x202D0	AI InTimer	SI2_Load_B_Register
0x202D4	AI InTimer	DIV_Load_A_Register
0x202DC	AI InTimer	Interrupt1_Register
0x202E0	AI InTimer	Interrupt2_Register
0x202E8	AI InTimer	Reset_Register

Counter 0

0x20300	G0 Counter	Gi_Command_Register
0x20302	G0 Counter	Gi_Mode_Register
0x20304	G0 Counter	Gi_HW_Save_Register
0x20304	G0 Counter	Gi_Load_A_Register
0x20308	G0 Counter	Gi_Save_Register
0x20308	G0 Counter	Gi_Load_B_Register
0x2030C	G0 Counter	Gi_Status_Register
0x2030C	G0 Counter	Gi_Input_Select_Register
0x2030E	G0 Counter	Gi_Autoincrement_Register
0x20310	G0 Counter	Gi_FifoStatusRegister
0x20310	G0 Counter	Gi_Counting_Mode_Register
0x20312	G0 Counter	Gi_Second_Gate_Register
0x20314	G0 Counter	Gi_SampleClockCountRegister
0x20314	G0 Counter	Gi_DMA_Config_Register
0x20318	G0 Counter	Gi_RdFifoRegister
0x20318	G0 Counter	Gi_WrFifoRegister
0x2031C	G0 Counter	Gi_SampleClockRegister
0x2031E	G0 Counter	Gi_AuxCtrRegister
0x20320	G0 Counter	Gi_AuxCtrLoadA_Register
0x20324	G0 Counter	Gi_AuxCtrLoadB_Register
0x20328	G0 Counter	Gi_AutomaticLoadRegister
0x2032C	G0 Counter	Gi_Interrupt1_Register
0x20330	G0 Counter	Gi_Interrupt2_Register

0x20338	G0 Counter	Gi_ABZ_Select_Register
0x2033C	G0 Counter	Gi_Mode3_Register
0x2033E	G0 Counter	Gi_Mode2_Register

Counter 1

0x20340	G1 Counter	Gi_Command_Register
0x20342	G1 Counter	Gi_Mode_Register
0x20344	G1 Counter	Gi_HW_Save_Register
0x20344	G1 Counter	Gi_Load_A_Register
0x20348	G1 Counter	Gi_Save_Register
0x20348	G1 Counter	Gi_Load_B_Register
0x2034C	G1 Counter	Gi_Status_Register
0x2034C	G1 Counter	Gi_Input_Select_Register
0x2034E	G1 Counter	Gi_Autoincrement_Register
0x20350	G1 Counter	Gi_FifoStatusRegister
0x20350	G1 Counter	Gi_Counting_Mode_Register
0x20352	G1 Counter	Gi_Second_Gate_Register
0x20354	G1 Counter	Gi_SampleClockCountRegister
0x20354	G1 Counter	Gi_DMA_Config_Register
0x20358	G1 Counter	Gi_RdFifoRegister
0x20358	G1 Counter	Gi_WrFifoRegister
0x2035C	G1 Counter	Gi_SampleClockRegister
0x2035E	G1 Counter	Gi_AuxCtrRegister
0x20360	G1 Counter	Gi_AuxCtrLoadA_Register
0x20364	G1 Counter	Gi_AuxCtrLoadB_Register

0x20368	G1 Counter	Gi_AutomaticLoadRegister
0x2036C	G1 Counter	Gi_Interrupt1_Register
0x20370	G1 Counter	Gi_Interrupt2_Register
0x20378	G1 Counter	Gi_ABZ_Select_Register
0x2037C	G1 Counter	Gi_Mode3_Register
0x2037E	G1 Counter	Gi_Mode2_Register

Counter 2

0x20380	G2 Counter	Gi_Command_Register
0x20382	G2 Counter	Gi_Mode_Register
0x20384	G2 Counter	Gi_HW_Save_Register
0x20384	G2 Counter	Gi_Load_A_Register
0x20388	G2 Counter	Gi_Save_Register
0x20388	G2 Counter	Gi_Load_B_Register
0x2038C	G2 Counter	Gi_Status_Register
0x2038C	G2 Counter	Gi_Input_Select_Register
0x2038E	G2 Counter	Gi_Autoincrement_Register
0x20390	G2 Counter	Gi_FifoStatusRegister
0x20390	G2 Counter	Gi_Counting_Mode_Register
0x20392	G2 Counter	Gi_Second_Gate_Register
0x20394	G2 Counter	Gi_SampleClockCountRegister
0x20394	G2 Counter	Gi_DMA_Config_Register
0x20398	G2 Counter	Gi_RdFifoRegister
0x20398	G2 Counter	Gi_WrFifoRegister
0x2039C	G2 Counter	Gi_SampleClockRegister

0x2039E	G2 Counter	Gi_AuxCtrRegister
0x203A0	G2 Counter	Gi_AuxCtrLoadA_Register
0x203A4	G2 Counter	Gi_AuxCtrLoadB_Register
0x203A8	G2 Counter	Gi_AutomaticLoadRegister
0x203AC	G2 Counter	Gi_Interrupt1_Register
0x203B0	G2 Counter	Gi_Interrupt2_Register
0x203B8	G2 Counter	Gi_ABZ_Select_Register
0x203BC	G2 Counter	Gi_Mode3_Register
0x203BE	G2 Counter	Gi_Mode2_Register

Counter 3

0x203C0	G3 Counter	Gi_Command_Register
0x203C2	G3 Counter	Gi_Mode_Register
0x203C4	G3 Counter	Gi_HW_Save_Register
0x203C4	G3 Counter	Gi_Load_A_Register
0x203C8	G3 Counter	Gi_Save_Register
0x203C8	G3 Counter	Gi_Load_B_Register
0x203CC	G3 Counter	Gi_Status_Register
0x203CC	G3 Counter	Gi_Input_Select_Register
0x203CE	G3 Counter	Gi_Autoincrement_Register
0x203D0	G3 Counter	Gi_FifoStatusRegister
0x203D0	G3 Counter	Gi_Counting_Mode_Register
0x203D2	G3 Counter	Gi_Second_Gate_Register
0x203D4	G3 Counter	Gi_SampleClockCountRegister
0x203D4	G3 Counter	Gi_DMA_Config_Register

0x203D8	G3 Counter	Gi_RdFifoRegister
0x203D8	G3 Counter	Gi_WrFifoRegister
0x203DC	G3 Counter	Gi_SampleClockRegister
0x203DE	G3 Counter	Gi_AuxCtrRegister
0x203E0	G3 Counter	Gi_AuxCtrLoadA_Register
0x203E4	G3 Counter	Gi_AuxCtrLoadB_Register
0x203E8	G3 Counter	Gi_AutomaticLoadRegister
0x203EC	G3 Counter	Gi_Interrupt1_Register
0x203F0	G3 Counter	Gi_Interrupt2_Register
0x203F8	G3 Counter	Gi_ABZ_Select_Register
0x203FC	G3 Counter	Gi_Mode3_Register
0x203FE	G3 Counter	Gi_Mode2_Register

Analog Output

0x20400	Analog Output	AO_DacShadow (i) Register Array
0x20400	Analog Output	AO_Direct_Data (i) Register Array
0x20440	Analog Output	AO_Order_Config_Data_Register
0x20444	Analog Output	AO_Config_Control_Register
0x20448	Analog Output	AO_Trigger_Select_Register
0x2044C	Analog Output	AO_Config_Bank (i) Register Array
0x20458	Analog Output	AO_FIFO_Status_Register
0x20458	Analog Output	AO_FIFO_Data_Register

AO OutTimer

0x20470	AO OutTimer	Status_1_Register
0x20470	AO OutTimer	Command_1_Register
0x20474	AO OutTimer	UI_Save_Register
0x20474	AO OutTimer	UI_Load_A_Register
0x20478	AO OutTimer	UI_Load_B_Register
0x2047C	AO OutTimer	UC_Save_Register
0x2047C	AO OutTimer	UC_Load_A_Register
0x20480	AO OutTimer	UC_Load_B_Register
0x20484	AO OutTimer	BC_Save_Register
0x20484	AO OutTimer	BC_Load_A_Register
0x20488	AO OutTimer	BC_Load_B_Register
0x2048C	AO OutTimer	Mode_1_Register
0x20490	AO OutTimer	Mode_2_Register
0x20496	AO OutTimer	Output_Control_Register
0x20498	AO OutTimer	Interrupt1_Register
0x2049C	AO OutTimer	Interrupt2_Register
0x204A4	AO OutTimer	Reset_Register

Digital Output

0x204AC	Digital Output	DO_FIFO_St_Register
0x204AC	Digital Output	SCXI_DIO_Enable_Register
0x204B0	Digital Output	SCXI_DIO_Enable_Register
0x204B4	Digital Output	DIO_Direction_Register

0x204B8	Digital Output	CDO_FIFO_Data_Register
0x204BC	Digital Output	DO_Mask_Enable_Register
0x204C0	Digital Output	DO_Mode_Register
0x204C4	Digital Output	DO_Trigger_Select_Register
0x204C8	Digital Output	DO_DirectDataRegister
0x204D0	Digital Output	DO_WDT_SafeStateRegister
0x204D4	Digital Output	DO_WDT_ModeSelect1_Register
0x204D8	Digital Output	DO_WDT_ModeSelect2_Register

DO OutTimer

0x204E0	DO OutTimer	Status_1_Register
0x204E0	DO OutTimer	Command_1_Register
0x204E4	DO OutTimer	UI_Save_Register
0x204E4	DO OutTimer	UI_Load_A_Register
0x204E8	DO OutTimer	UI_Load_B_Register
0x204EC	DO OutTimer	UC_Save_Register
0x204EC	DO OutTimer	UC_Load_A_Register
0x204F0	DO OutTimer	UC_Load_B_Register
0x204F4	DO OutTimer	BC_Save_Register
0x204F4	DO OutTimer	BC_Load_A_Register
0x204F8	DO OutTimer	BC_Load_B_Register
0x204FC	DO OutTimer	Mode_1_Register
0x20500	DO OutTimer	Mode_2_Register
0x20506	DO OutTimer	Output_Control_Register
0x20508	DO OutTimer	Interrupt1_Register

0x2050C	DO OutTimer	Interrupt2_Register
0x20514	DO OutTimer	Reset_Register

Digital Input

0x20530	Digital Input	Static_Digital_Input_Register
0x20530	Digital Input	DI_DMA_Select_Register
0x20534	Digital Input	DI_FIFO_St_Register
0x20534	Digital Input	DI_Mode_Register
0x20538	Digital Input	DI_FIFO_Data_Register
0x20538	Digital Input	DI_FIFO_Data_Register16
0x20538	Digital Input	DI_FIFO_Data_Register8
0x20538	Digital Input	DI_Mask_Enable_Register
0x2053C	Digital Input	DI_Trigger_Select_Register
0x20540	Digital Input	DI_ChangeDetectStatusRegister
0x20540	Digital Input	DI_ChangeIrqRE_Register
0x20544	Digital Input	DI_ChangeDetectLatchedDI_Register
0x20544	Digital Input	DI_ChangeIrqFE_Register
0x20548	Digital Input	DI_ChangeDetectLatchedPFI_Register
0x20548	Digital Input	PFI_ChangeIrq_Register
0x2054C	Digital Input	DI_FilterRegisterLo
0x20554	Digital Input	DI_ChangeDetectIRQ_Register

DI InTimer

0x20560	DI InTimer	Status_1_Register
0x20560	DI InTimer	Command_Register
0x20564	DI InTimer	Status_2_Register
0x20564	DI InTimer	Mode_1_Register
0x20568	DI InTimer	SI_Save_Register
0x20568	DI InTimer	Mode_2_Register
0x2056C	DI InTimer	SC_Save_Register
0x2056C	DI InTimer	SI_Load_A_Register
0x20570	DI InTimer	SI2_Save_Register
0x20570	DI InTimer	SI_Load_B_Register
0x20574	DI InTimer	DIV_Save_Register
0x20574	DI InTimer	SC_Load_A_Register
0x20578	DI InTimer	SC_PreWaitCntRegister
0x20578	DI InTimer	SC_Load_B_Register
0x2057C	DI InTimer	SI2_Load_A_Register
0x20580	DI InTimer	SI2_Load_B_Register
0x20584	DI InTimer	DIV_Load_A_Register
0x2058C	DI InTimer	Interrupt1_Register
0x20590	DI InTimer	Interrupt2_Register
0x20598	DI InTimer	Reset_Register

AI StreamCircuit

0x24800	AI StreamCircuit	StreamControlStatusReg
0x24810	AI StreamCircuit	StreamTransferCountReg
0x24810	AI StreamCircuit	StreamAdditiveTransferCountReg
0x24814	AI StreamCircuit	StreamFifoSizeReg
0x2481C	AI StreamCircuit	StreamTransferLimitReg
0x24820	AI StreamCircuit	StreamEvictionReg
0x24824	AI StreamCircuit	StreamTransactionLimitReg
0x24904	AI StreamCircuit	DMAChannel

Counter 0 StreamCircuit

0x26800	G0 StreamCircuit	StreamControlStatusReg
0x26810	G0 StreamCircuit	StreamTransferCountReg
0x26810	G0 StreamCircuit	StreamAdditiveTransferCountReg
0x26814	G0 StreamCircuit	StreamFifoSizeReg
0x2681C	G0 StreamCircuit	StreamTransferLimitReg
0x26820	G0 StreamCircuit	StreamEvictionReg
0x26824	G0 StreamCircuit	StreamTransactionLimitReg
0x26904	G0 StreamCircuit	DMAChannel

Counter 1 StreamCircuit

0x28800	G1 StreamCircuit	StreamControlStatusReg
0x28810	G1 StreamCircuit	StreamTransferCountReg
0x28810	G1 StreamCircuit	StreamAdditiveTransferCountReg
0x28814	G1 StreamCircuit	StreamFifoSizeReg

0x2881C	G1 StreamCircuit	StreamTransferLimitReg
0x28820	G1 StreamCircuit	StreamEvictionReg
0x28824	G1 StreamCircuit	StreamTransactionLimitReg
0x28904	G1 StreamCircuit	DMAChannel

Counter 2 StreamCircuit

0x2A800	G2 StreamCircuit	StreamControlStatusReg
0x2A810	G2 StreamCircuit	StreamTransferCountReg
0x2A810	G2 StreamCircuit	StreamAdditiveTransferCountReg
0x2A814	G2 StreamCircuit	StreamFifoSizeReg
0x2A81C	G2 StreamCircuit	StreamTransferLimitReg
0x2A820	G2 StreamCircuit	StreamEvictionReg
0x2A824	G2 StreamCircuit	StreamTransactionLimitReg
0x2A904	G2 StreamCircuit	DMAChannel

Counter 3 StreamCircuit

0x2C800	G3 StreamCircuit	StreamControlStatusReg
0x2C810	G3 StreamCircuit	StreamTransferCountReg
0x2C810	G3 StreamCircuit	StreamAdditiveTransferCountReg
0x2C814	G3 StreamCircuit	StreamFifoSizeReg
0x2C81C	G3 StreamCircuit	StreamTransferLimitReg
0x2C820	G3 StreamCircuit	StreamEvictionReg
0x2C824	G3 StreamCircuit	StreamTransactionLimitReg
0x2C904	G3 StreamCircuit	DMAChannel

DI StreamCircuit

0x2E800	DI StreamCircuit	StreamControlStatusReg
0x2E810	DI StreamCircuit	StreamTransferCountReg
0x2E810	DI StreamCircuit	StreamAdditiveTransferCountReg
0x2E814	DI StreamCircuit	StreamFifoSizeReg
0x2E81C	DI StreamCircuit	StreamTransferLimitReg
0x2E820	DI StreamCircuit	StreamEvictionReg
0x2E824	DI StreamCircuit	StreamTransactionLimitReg
0x2E904	DI StreamCircuit	DMAChannel

AO StreamCircuit

0x30800	AO StreamCircuit	StreamControlStatusReg
0x30810	AO StreamCircuit	StreamTransferCountReg
0x30810	AO StreamCircuit	StreamAdditiveTransferCountReg
0x30814	AO StreamCircuit	StreamFifoSizeReg
0x3081C	AO StreamCircuit	StreamTransferLimitReg
0x30820	AO StreamCircuit	StreamEvictionReg
0x30824	AO StreamCircuit	StreamTransactionLimitReg
0x30904	AO StreamCircuit	DMAChannel

DO StreamCircuit

0x32800	DO StreamCircuit	StreamControlStatusReg
0x32810	DO StreamCircuit	StreamTransferCountReg
0x32810	DO StreamCircuit	StreamAdditiveTransferCountReg
0x32814	DO StreamCircuit	StreamFifoSizeReg

0x3281C	DO	StreamCircuit	StreamTransferLimitReg
0x32820	DO	StreamCircuit	StreamEvictionReg
0x32824	DO	StreamCircuit	StreamTransactionLimitReg
0x32904	DO	StreamCircuit	DMAChannel

B

Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Technical support at ni.com/support includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at forums.ni.com. The specific DDK Forum can be found at forums.ni.com/t5/Driver-Development-Kit-DDK/bd-p/90. NI Applications Engineers make sure every question submitted online receives an answer.
  - For information about other technical support options in your area, visit ni.com/services, or contact your local office at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.
- **Declaration of Conformity (DoC)**—A DoC is our claim of compliance with the Council of the European Communities using the manufacturer's declaration of conformity. This system affords the user protection for electromagnetic compatibility (EMC) and product safety. You can obtain the DoC for your product by visiting ni.com/certification.

- **Calibration Certificate**—If your product supports calibration, you can obtain the calibration certificate for your product at `ni.com/calibration`.

You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.