NI PCIe-6509

NI PCIe-6509 Register Level Programming Reference Manual

January 2012
373665A-01

Important Information

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>.
[ ]	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 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
NI PCIe-6509 Architecture

Chapter 2
NI PCIe-6509 Features

Digital Input 2-1
    Change Detection 2-1
    Debounce Filters 2-2
Digital Output 2-2
    Power-Up State 2-2
    Watchdog Timer 2-2
Interrupts 2-3
    CHInCh Interrupt Control Registers 2-4
    DAQ-STC3 Interrupt Control Registers 2-4
    Enabling Interrupts 2-4
    Acknowledging Interrupts 2-6
    Disabling Interrupts 2-7

Chapter 3
Programming Considerations

EEPROM 3-1
    Capabilities List 3-2
    Interfacing with the EEPROM 3-5
Register Bit Maps 3-5
    RBM File Syntax 3-5
        Global Settings 3-5
        Registers 3-7
        Examples 3-9

Chapter 4
Example Synopses

Examples 4-1

Chapter 5
NI PCIe-6509 Chip Object

Chapter 6
Register Map

List of Static DIO Registers 6-1
List of Filter Registers and Enumerations 6-1
List of Change Detect Registers 6-1
List of Watchdog Timer Registers and Enumerations 6-2
List of Interrupt Registers 6-2
List of RTSI Registers and Enumerations 6-2
List of Miscellaneous Registers 6-3
Static DIO Registers 6-4
Filter Registers 6-9
Change Detect Registers 6-14
Watchdog Timer Registers 6-17
Interrupt Registers 6-24
RTSI Configuration Registers 6-30
Miscellaneous Registers 6-32

Appendix A
Register Maps Appendix

Appendix B
Technical Support and Professional Services

1

NI PCIe-6509 Architecture

The architecture of the NI PCIe-6509 is different than that of the NI PCI-6509, which uses a single tailored ASIC for digital I/O (DIO). The NI PCIe-6509 utilizes two DAQ System Timing Chip 3 (DAQ-STC3) ASICs, developed by National Instruments, in a master/slave configuration. The master DAQ-STC3 is directly connected to the NI PCI Express connector, while the slave DAQ-STC3 is connected to an I/O port on the master DAQ-STC3. The two DAQ-STC3s appear 0x20000 apart in the BAR0 addressable memory space of the NI PCIe-6509. The RTSI lines for each DAQ-STC3 do not terminate at a RTSI connector; they are linked together to allow for the transmission of signals from one DAQ-STC3 to the other.

The Common Host Interface Chip (CHInCh) is the PCI Express interface used on the master DAQ-STC3. This latest generation chip is analogous to the MITE that was present in previous versions of data acquisition hardware. This controller provides both input and output communication to and from the host computer. The basic architecture of PCI Express and PXI Express is very different from that of PCI and PXI, respectively. Each device has a point-to-point link with the switch it is connected to instead of a shared bus between all of the devices.

The only data acquisition function of the DAQ-STC3 used by the NI PCIe-6509 is DIO. Each DAQ-STC3 provides access to 48 lines of DIO for a combined total of the 96 DIO lines accessible on the NI PCIe-6509. The NI PCIe-6509's 96 DIO lines are divided into 12 8-bit ports. Although the DAQ-STC3 allows users to update four DIO ports or two PFI ports simultaneously, only one NI PCIe-6509 port should be updated at a time to prevent crosstalk between lines. Table 1-1 shows the port mappings for NI PCIe-6509 compared with the DIO available on each DAQ-STC3.

Table 1-1. NI PCIe-6509 Port Mappings

NI PCIe-6509 Port	DAQ-STC3 Mapping
Port0	Master DAQ-STC3 Port0 Lines 0–7
Port1	Master DAQ-STC3 Port0 Lines 8–15
Port2	Master DAQ-STC3 Port0 Lines 16–23

Table 1-1. NI PCIe-6509 Port Mappings (Continued)

NI PCIe-6509 Port	DAQ-STC3 Mapping
Port3	Master DAQ-STC3 Port0 Lines 24–31
Port4	Master DAQ-STC3 PFI Lines 0–7
Port5	Master DAQ-STC3 PFI Lines 8–15
Port6	Slave DAQ-STC3 PFI Lines 0–7
Port7	Slave DAQ-STC3 PFI Lines 8–15
Port8	Slave DAQ-STC3 Port0 Lines 0–7
Port9	Slave DAQ-STC3 Port0 Lines 8–15
Port10	Slave DAQ-STC3 Port0 Lines 16–23
Port11	Slave DAQ-STC3 Port0 Lines 24–31

For example, in order to configure Port0 to output mode and output a DIO pattern, you would use the registers related to the Master DAQ-STC3 Port0 lines 0–7. Throughout this document, digital lines are also referred to as DI or PFI lines.

2

NI PCIe-6509 Features

This chapter includes information on the features of the NI PCIe-6509, *Digital Input*, *Digital Output*, and *Interrupts*.

Digital Input

The only digital input (DI) acquisition mode supported by the NI PCIe-6509 is Static DI. Static DI, also referred to as software-timed, is where the rate of acquisition is determined by the speed of the software application.

Some of the features of DI 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 circuit enables detection and reporting of any falling or rising edges. Change detection can be enabled by configuring the *DI_ChangeIrqFE_Register* for falling edges and *DI_ChangeIrqRE_Register* for rising edges on the desired DIO lines. The *PFI_ChangeIrq_Register* can be used to enable change detection for falling and rising edges of the PFI lines. The *ChangeDetectStatusRegister* can be read to determine if a change detection event was detected on any lines configured with the *DI_ChangeIrqFE_Register*, *DI_ChangeIrqRE_Register*, or *PFI_ChangeIrq_Register*. The *DI_ChangeDetectLatched_Register* and *PFI_ChangeDetectLatched_Register* reflect the state of the lines at the time the last change detection event occurred for each DAQ-STC3. The change detection for the master and slave DAQ-STC3s are not linked, so the data lines of each DAQ-STC3 should be cached before software detects a change detection event. Another option would be to only use one DAQ-STC3 for change detection. If these methods do not give the change detection granularity needed, contact your National Instruments field sales engineer.

Debounce Filters

A debounce filter can be enabled on all 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_Filter_Port0and1* and *DI_Filter_Port2and3* registers for the DIO lines. The *PFI_Filter_Port0Lo*, *PFI_Filter_Port0Hi*, *PFI_Filter_Port1Lo*, and *PFI_Filter_Port1Hi* registers are used to set debounce filters for the PFI lines.

Digital Output

The only digital output (DO) generation mode supported by the NI PCIe-6509 is Static DO. Static DO, also referred to as software-timed, is where the rate of generation is determined by the speed of the software application.

Power-Up State

At system startup and reset, the DAQ-STC3 sets all PFI and DIO lines to high-impedance inputs. The power-up behavior of the device can be set by changing the power-up state. The power-up states are user-configurable, using the DAQmx driver.

Watchdog Timer

The onboard Watchdog Timer can be programmed to set the DO outputs to safe states in the event of communication loss between the host computer application and the NI PCIe-6509. The Watchdog Timer can be enabled through the *DO_WDT_ModeSelect_Port0and1_Register*, the *DO_WDT_ModeSelect_Port2and3_Register*, and the *PFI_WDT_ModeSelect_Register*. The per-line safe state value can be specified through the *DO_WDT_SafeStateRegister* and the *PFI_WDT_SafeStateRegister*. In order to set one Watchdog Timer for both DAQ-STC3 chips, the master DAQ-STC3 should use the internal clock, while the slave DAQ-STC3 should use an external signal. The Watchdog Timer event should then be routed from the master DAQ-STC3 to the slave DAQ-STC3.

1. Set the WatchdogExtTrigSel bitfield to the desired RTSI line on the slave DAQ-STC3.

   ```
   WatchdogConfiguration.writeWatchdogExtTrigSel
   (WdtSrc_RTSI0);
   ```

2. Use the *RTSI_OutputSelectRegister_i* and the *RTSI_Trig_Direction_Register* of the master DAQ-STC3 to export the Watchdog Timer event to the RTSI line chosen in Step 2.

   ```
   RTSI_OutputSelectRegister_i[0].writeRTSI_i_Output_
   Select(RTSI_WatchdogExpiredPulse);
   ```

   ```
   RTSI_Trig_Direction.writeRTSIDirection(1);
   ```

3. Set the desired Watchdog Timeout to the *WatchdogTimeoutRegister* of the master DAQ-STC3.

   ```
   WatchdogTimeoutRegister.writeRegister(timeout);
   ```

4. Enable the WatchdogExtTrigEn bitfield in the *WatchdogConfiguration* register of the slave DAQ-STC3.

   ```
   WatchdogConfiguration.writeWatchdogExtTrigEn
   (kTrue);
   ```

5. Enable the WatchdogIntTrigEn bitfield in the *WatchdogConfiguration* register of the master DAQ-STC3.

   ```
   WatchdogConfiguration.writeWatchdogIntTrigEn
   (kTrue);
   ```

Interrupts

The interrupt architecture for the NI PCIe-6509 is different than that of the NI PCI-6509. The largest change is the interrupt routing: the NI PCI-6509 raises interrupts through the MITE while the NI PCIe-6509 uses the CHInCh. The CHInCh provides a simpler interface for enabling, acknowledging, and disabling interrupts. The CHInCh routes all of the interrupts generated by an NI PCIe-6509.

The NI PCIe-6509 generates interrupts for Watchdog Timer events, Change Detection events, and Change Detection Error events, and the CHInCh then routes those interrupts to the host CPU. The register interface provides increasingly granular control over the interrupts: starting from using one bit to enable/disable all device interrupts, to enabling/disabling every DI and Watchdog Timer interrupt on one DAQ-STC3, to enabling/disabling specific 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 bitfield for the entire device as well as an enable/disable bitfield for the DAQ-STC3. The ISR and VISR report the source of the highest-priority interrupt on the device. The VISR is automatically cleared after the register is read.

DAQ-STC3 Interrupt Control Registers

The DAQ-STC3 has several registers for interrupt control: some offer ASIC-wide control and others are specific to the digital and Watchdog Timer subsystems. The ASIC-wide registers are in the ChpServices chip object, and they provide bitfields to enable/disable all of the interrupts associated with the DI and Watchdog Timer as well as bitfields to determine which specific interrupts have been raised.

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: DI or Watchdog Timer. 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 as follows:

1. OS load.
2. Disable and acknowledge all interrupts.
   a. Strobe the bitfields Clear_CPU_Int and Clear_STC3_Int in the CHInCh IMR, which blocks any device interrupt from signaling the CPU.

      ```
      Interrupt_Mask_Register.writeRegister
      (Clear_CPU_Int|Clear_STC3_Int);
      ```

   b. Acknowledge and disable all interrupts for the DI and Watchdog Timer in the ChpServices for each DAQ-STC3.

      ```
      GlobalInterruptEnable_Register.writeRegister
      (DI_Interrupt_Disable|
      WatchdogTimer_Interrupt_Disable);
      ```

```
WatchdogTimer_Interrupt2_Register.writeRegister
(WDT_TriggerIRQ_Disable|WDT_TriggerIRQ_Ack2);
```

```
ChangeDetectIRQ_Register.writeRegister
(ChangeDetectIRQ_Acknowledge|
ChangeDetectIRQ_Disable|
ChangeDetectErrorIRQ_Acknowledge|
ChangeDetectErrorIRQ_Disable);
```

3. Route the master and slave DAQ-STC3 interrupts to the master CHInCh.

   ```
   ChpServicesLo.IntForwarding_DestinationReg.
   writeRegister(0);
   ```

   ```
   ChpServicesHi.IntForwarding_DestinationReg.
   writeRegister(24);
   ```

4. Register interrupt handlers with the operating system.
5. Enable specific interrupts.
   a. Strobe Set_CPU_Int and Set_STC3_Int in the CHInCh IMR to enable interrupt signaling from the device.

      ```
      Interrupt_Mask_Register.writeRegister
      (Set_CPU_Int|Set_STC3_Int);
      ```

   b. Enable the relevant interrupts needed by the operation in the ChpServices of each DAQ-STC3. For example, to enable the change detection interrupt, perform the following writes:

      ```
      GlobalInterruptEnable_Register.writeRegister
      (DI_Interrupt_Enable);
      ```

      ```
      ChangeDetectIRQ_Register.writeRegister
      (ChangeDetectIRQ_Enable);
      ```

Acknowledging Interrupts

After interrupts have been enabled, the device asserts 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 five or fewer accesses. The order of programming in this situation is as follows:

1. Receive NI PCIe-6509 device interrupt.
2. Determine which NI PCIe-6509 device is interrupting by reading that device's CHInCh (V)ISR.

   ```
   value = Interrupt_Status_Register.readRegister();
   ```

   The External and STC3_Int bitfields will be set since all interrupts pertain to either the DI or Watchdog Timer.
3. Read the *GlobalInterruptStatus_Register* on each DAQ-STC3 to determine the source of the interrupt. Then read each source's Interrupt Status Register to determine which specific interrupts have requested service. Acknowledge any interrupts. For example, if there was a DI change detection event, perform the following accesses to determine that this interrupt was asserted.

   ```
   value =
   GlobalInterruptStatus_Register.readRegister();
   <Check value for source of interrupt>

   value = DI_Interrupt_Status_Register.readRegister();
   <Check value for source of interrupt>

   <if value indicates that ChangeDetectionIrqSt as the
   source>
   ChangeDetectIRQ_Register.writeRegister
   (ChangeDetectIRQ_Acknowledge);
   ```
4. If global or specific interrupts were disabled during the interrupt service routine, as is common in many designs, re-enable them after they have been fully serviced.

   ```
   GlobalInterruptEnable_Register.writeRegister
   (DI_Interrupt_Enable);

   ChangeDetectIRQ_Register.writeRegister
   (ChangeDetectIRQ_Enable);
   ```

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 feature is not in use, it is good practice to disable those corresponding interrupts. The logic of programming in such situations is as follows:

1. To disable all interrupts, strobe the Clear_CPU_Int and Clear_STC3_Int bitfields in the CHInCh's IMR.

   ```
   Interrupt_Mask_Register.writeRegister
   (Clear_CPU_Int|Clear_STC3_Int);
   ```

2. To disable all interrupts from the DI or Watchdog Timer use the relevant bitfields in the *GlobalInterruptEnable_Register* on each DAQ-STC3.

   ```
   GlobalInterruptEnable_Register.writeRegister
   (DI_Interrupt_Disable|
   WatchdogTimer_Interrupt_Disable);
   ```

3. To disable specific interrupts, use the relevant bitfields in the following registers: *ChangeDetectIrq_Register*, *WatchdogTimer_Interrupt1_Register* and *WatchdogTimer_Interrupt2_Register*.

   ```
   WatchdogTimer_Interrupt2_Register.writeRegister
   (WDT_TriggerIRQ_Disable);
   ```

   ```
   ChangeDetectIRQ_Register.writeRegister
   (ChangeDetectIRQ_Disable|
   ChangeDetectErrorIRQ_Disable);
   ```

3 Programming Considerations

This chapter contains sections on the *EEPROM* and *Register Bit Maps*.

EEPROM

The EEPROM contains useful information about a particular device including: the power-up states of each port, programmatic pull-up or pull-down register states, and serial number.

Table 3-1. 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 3-2. 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. There will not be 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 NI PCIe-6509 devices will have at least two nodes in their capabilities list, as described in Tables 3-3 and 3-4.

Table 3-3. 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 the device.

Table 3-4. 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.

Table 3-4. Device-Specific (ID: 0x0001) (Continued)

Address	Value (32-bits)	Description
+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 3-5.
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 power-up states.

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.

Table 3-5. Device Specific Node IDs

ID	Meaning	Description
0x0010	Master P0 Power-Up Value Pointer	Pointer to the 32-bit power-up value of the digital port 0 on the master DAQ-STC3, this value will be copied to the Static_Digital_Output_Register in the master DAQ-STC3 at power-up. The 32-bit word immediately after the power-up value will be the power-up direction. This word will be copied to the DIO_Direction_Register in the master DAQ-STC3 at power-up.
0x0011	Master P1 and P2 Power-Up Value Pointer	Pointer to the 16-bit power-up value of the digital ports 1 and 2 on the master DAQ-STC3, this value will be copied to the PFI_DO_Register in the master DAQ-STC3 at power-up. The 16-bit word immediately after the power-up value will be the power-up direction. This word will be copied to the PFI_Direction_Register in the master DAQ-STC3 at power-up.
0x0012	Master Pull Direction Pointer	Pointer to the 6 bytes of tristate logic level data for digital port Pull-up/Pull-down on the master DAQ-STC3, this value will show the state of the port when the lines are not driven.
0x0020	Slave P0 Power-Up Value Pointer	Pointer to the 32-bit power-up value of the digital port 0 on the slave DAQ-STC3, this value will be copied to the Static_Digital_Output_Register in the slave DAQ-STC3 at power-up. The 32-bit word immediately after the power-up value will be the power up direction. This word will be copied to the DIO_Direction_Register in the slave DAQ-STC3 at power-up.
0x0021	Slave P1 and P2 Power-Up Value Pointer	Pointer to the 16-bit power-up value of the digital ports 1 and 2 on the slave DAQ-STC3, this value will be copied to the PFI_DO_Register in the slave DAQ-STC3 at power-up. The 16-bit word immediately after the power-up value will be the power-up direction. This word will be copied to the PFI_Direction_Register in the slave DAQ-STC3 at power-up.
0x0022	Slave Pull Direction Pointer	Pointer to the 6 bytes tristate logic level data for digital port Pull-up/Pull-down on the slave DAQ-STC3, this value will show the state of the port when the lines are not driven.

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 does not 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/StaticDio.rbm`, `RBM/DioPorts.rbm`, and `RBM/PfiPorts.rbm`.

Global Settings

```
—contains DioPortsLo 0x20000 DioPorts.rbm "tDioPorts.h"
```

This line is in `StaticDio.rbm`, a bit map that aggregates the DIO subsystem bit maps. Specifically, this line says that a variable called `DioPortsLo` at offset `0x20000` is part of this bit map. The bit map for `DioPortsLo` can be found in the file `DioPorts.rbm`, and its declarations are in the file `tDioPorts.h`. The generated code can be found in `ChipObjects/tStaticDio.h` and `ChipObjects/tStaticDio.cpp`.

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

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

Enumeration

```
E DI_Filter_Select_t
V No_Filter     0
V Small_Filter  1
V Medium_Filter 2
V Large_Filter  3
```

This stanza is in `DioPorts.rbm`, and it defines an enumeration named `DI_Filter_Select_t` with unique values for four different DI filter configurations. The generated code can be found in `ChipObjects/tDioPortsValues.h`.

Concrete Register

```
R DI_FilterRegister_Port0and1   32  0x54C Writable
F DI_Filter_Select_Port0_Line0  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 0 Filter Selection
F DI_Filter_Select_Port0_Line1  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 1 Filter Selection
F DI_Filter_Select_Port0_Line2  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 2 Filter Selection
F DI_Filter_Select_Port0_Line3  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 3 Filter Selection
F DI_Filter_Select_Port0_Line4  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 4 Filter Selection
F DI_Filter_Select_Port0_Line5  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 5 Filter Selection
F DI_Filter_Select_Port0_Line6  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 6 Filter Selection
F DI_Filter_Select_Port0_Line7  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 0 line 7 Filter Selection
F DI_Filter_Select_Port1_Line0  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 0 Filter Selection
```

```
F DI_Filter_Select_Port1_Line1  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 1 Filter Selection
F DI_Filter_Select_Port1_Line2  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 2 Filter Selection
F DI_Filter_Select_Port1_Line3  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 3 Filter Selection
F DI_Filter_Select_Port1_Line4  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 4 Filter Selection
F DI_Filter_Select_Port1_Line5  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 5 Filter Selection
F DI_Filter_Select_Port1_Line6  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 6 Filter Selection
F DI_Filter_Select_Port1_Line7  2   nDioPorts::tDI_Filter_Select_t
                                    DI port 1 line 7Filter Selection
```

This stanza is in `DioPorts.rbm`, and it describes a concrete register named `DI_FilterRegister_Port0and1`. The register is 32-bits wide, can be found at offset `0x54C`, and is write-only. The first bitfield is called `DI_Filter_Select_Port0_Line0`. It is 2-bits wide (using bits `1..0`) and requires values from the enumeration `nDioPorts::tDI_Filter_Select_t`. Notice that some fields are marked `Reserved`. The generated code can be found in `ChipObjects/tDioPorts.h`.

Register Template

```
T PFI_OutputSelectRegister_t  8   Writable —-no-hardware-reset true
F PFI_i_Output_Select         7   nPfiPorts::tPFI_Output_Select_t
                                  @PFI line output source selection
F Reserved                    1
```

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

Register Array

```
TRA PFI_OutputSelectRegister_i%d PFI_OutputSelectRegister_t 0xBA
16 PFI_OutSel%d --step 1 --group PFI_OutSelection
```

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

4 Example Synopses

This chapter contains example definitions and descriptions. The examples are as follows:

`boardBringup.cpp`—Self-test an NI PCIe-6509 device.

`dioex1.cpp`—Single-point on-demand digital output on ports 0, 6, and 9.

`dioex2.cpp`—Multi-point on-demand digital input on selected ports.

`dioex3.cpp`—Multi-point on-demand digital output on selected ports.

`dioex4.cpp`—Demonstration of the Watchdog Timer.

`dioex5.cpp`—Demonstration of change detection and digital filters.

Examples

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

- **`boardBringup.cpp`**—Self-test an NI PCIe-6509 device.

  boardBringup performs a self-test on an NI PCIe-6509 device. After verifying the identity of the NI PCIe-6509 device, boardBringup reads the signature registers and prints other hardware information. Finally, boardBringup tests device register I/O by writing to and reading from the scratch pad registers.

- **`dioex1.cpp`**—Single-point on-demand digital output on ports 0, 6, and 9.

  dioex1 performs a software-timed digital output operation and transfers data by way of direct register accesses. After configuring the digital subsystem's channel parameters, dioex1 updates the digital values of ports 0, 6, and 9. Finally, dioex1 restores the hardware's previous state.

- **`dioex2.cpp`**—Multi-point on-demand digital input on selected ports.

  dioex2 performs a software-timed digital input operation and transfers data by way of direct register accesses. After configuring the digital subsystem's channel parameters, dioex2 reads the digital values from the selected ports. Finally, dioex2 restores the hardware's previous state.

- **`dioex3.cpp`**—Multi-point on-demand digital output on selected ports.

  dioex3 performs a software-timed digital output operation and transfers data by way of direct register accesses. After configuring the digital subsystem's channel parameters, dioex3 writes selected digital patterns to the selected ports. Finally, dioex3 restores the hardware's previous state.

- **`dioex4.cpp`**—Demonstration of the Watchdog Timer.

  dioex4 performs a software-timed digital output operation and transfers data by way of direct register accesses. After configuring the digital subsystem's channel parameters, dioex4 writes the selected digital patterns to the selected ports until dioex4 trips the Watchdog. For the next two seconds, dioex4 waits and shows the programmed safe states before restoring the hardware's previous state.
- **`dioex5.cpp`**—Demonstration of change detection and digital filters.

  dioex5 sets the digital filter for the selected ports and performs digital input based on changes detected by the change detection circuits of the device for 10 seconds. Finally, dioex5 restores the hardware's previous state.

5

NI PCIe-6509 Chip Object

This chapter contains information about the chip object for the NI PCIe-6509 devices. Table 5-1 shows sections of the chip object and the corresponding offset.

Table 5-1. NI PCIe-6509 Chip Object

Chip Object	Offset
CHInCh	0x0
ChpServicesLo	0x20000
ChpServicesHi	0x40000
DioPortsLo	0x20000
DioPortsHi	0x40000
PfiPortsLo	0x20000
PfiPortsHi	0x40000

6 Register Map

This chapter consists of *Static DIO Registers*, *Filter Registers*, *Change Detect Registers*, *Watchdog Timer Registers*, *Interrupt Registers*, *RTSI Configuration Registers*, and *Miscellaneous Registers*. Each register is listed by its chip object and name, and labeled as a read (R) or write (W) register.

List of Static DIO Registers

The Static DIO registers are as follows:

DioPorts Static_Digital_Input_Register (R)

PfiPorts Static_Digital_Input_Register (R)

DioPorts Static_Digital_Output_Register (W)

PfiPorts Static_Digital_Output_Register (W)

DioPorts DIO_Direction_Register (W)

PfiPorts PFI_Direction_Register (W)

PfiPorts PFI_OutputSelectRegister_i (W)

List of Filter Registers and Enumerations

The Filter registers are as follows:

DioPorts DI_FilterRegister_Port0and1

DioPorts DI_FilterRegister_Port2and3

PfiPorts PFI_Filter_Register_Port0Lo

PfiPorts PFI_Filter_Register_Port0Hi

PfiPorts PFI_Filter_Register_Port1Lo

PfiPorts PFI_Filter_Register_Port1Hi

The Filter enumerations are as follows:

DI_Filter_Select_t Enumeration

PFI_Filter_Select_t Enumeration

List of Change Detect Registers

The Change Detection registers are as follows:

ChpServices ChangeDetectStatusRegister (R)

DioPorts DI_ChangeIrqRE_Register (W)

DioPorts DI_ChangeIrqFE_Register (W)

PfiPorts PFI_ChangeIrq_Register (W)

DioPorts DI_ChangeDetectLatched_Register (R)

PfiPorts PFI_ChangeDetectLatched_Register (R)

List of Watchdog Timer Registers and Enumerations

The Watchdog Timer registers are as follows:

DioPorts DO_WDT_SafeStateRegister (W)

PfiPorts PFI_WDT_SafeStateRegister (W)

DioPorts DO_WDT_ModeSelect_Port0and1_Register (W)

DioPorts DO_WDT_ModeSelect_Port2and3_Register (W)

PfiPorts PFI_WDT_ModeSelect_Register (W)

ChpServices WatchdogStatusRegister (R)

ChpServices WatchdogTimeoutRegister (W)

ChpServices WatchdogConfiguration (W)

ChpServices WatchdogControl (W)

The Watchdog Timer enumerations are as follows:

DO_WDT_Mode_t Enumeration

PFI_WDT_Mode_t Enumeration

ChpSrv_WatchdogTimerStateMachineSt_t Enumeration

ChpServ_WatchdogTimerExtSrcSel_t Enumeration

List of Interrupt Registers

The Interrupt registers are as follows:

ChpServices GlobalInterruptStatus_Register (R)

ChpServices GlobalInterruptEnable_Register (W)

ChpServices DI_Interrupt_Status_Register (R)

ChpServices ChangeDetectIRQ_Register (W)

ChpServices WatchdogTimer_Interrupt_Status_Register (R)

ChpServices WatchdogTimer_Interrupt1_Register (W)

ChpServices WatchdogTimer_Interrupt2_Register (W)

CHInCh Interrupt_Mask_Register (R|W)

CHInCh Interrupt_Status_Register (R)

CHInCh Volatile_Interrupt_Status_Register (R)

ChpServices IntForwarding_ControlStatus (R|W)

ChpServices IntForwarding_DestinationReg (R|W)

List of RTSI Registers and Enumerations

The RTSI registers are as follows:

ChpServices RTSI_Trig_Direction_Register (W)

ChpServices RTSI_OutputSelectRegister_i (W)

The RTSI enumeration is shown below:

RTSI_Output_Select_t Enumeration

List of Miscellaneous Registers

The Miscellaneous registers are as follows:

- CHInCh CHInCh_Identification_Register (R)
- CHInCh Scrap_Register (R|W)
- CHInCh PCI_Subsystem_ID_Access_Register (R)
- ChpServices ScratchPadRegister (R|W)
- ChpServices Signature_Register (R)
- ChpServices Joint_Reset_Register (W)
- ChpServices TimeSincePowerUpRegister (R)

Static DIO Registers

DioPorts Static_Digital_Input_Register (R)

Master Offset: 0x20530 (NI PCIe-6509 port0..3)
Slave Offset: 0x40530 (NI PCIe-6509 port8..11)

This register reflects the state of the DI lines. Software should mask lines not being used for static digital input.

Bits	Name
31..24	**DIO_StaticInputValue_Port3**
23..16	**DIO_StaticInputValue_Port2**
15..8	**DIO_StaticInputValue_Port1**
7..0	**DIO_StaticInputValue_Port0**

PfiPorts Static_Digital_Input_Register (R)

Master Offset: 0x200E0 (NI PCIe-6509 port4..5)
Slave Offset: 0x400E0 (NI PCIe-6509 port6..7)

This register reflects the state of the DI lines. Software should mask lines not being used for static digital input.

Bits	Name
15..8	**PFI_StaticInputValue_Port1**
7..0	**PFI_StaticInputValue_Port0**

DioPorts Static_Digital_Output_Register (W)

Master DAQ-STC3 Offset: 0x204B0 (NI PCIe-6509 port0..3)
Slave DAQ-STC3 Offset: 0x404B0 (NI PCIe-6509 port8..11)

The data written in this register will be output to the digital lines when:

- The corresponding line is set as an output (DIODirection_Porti = 1)
- The Watchdog Timer functionality is not enabled or its timer has not triggered

Bits	Name
31..24	**DIO_StaticOutputValue_Port3**
23..16	**DIO_StaticOutputValue_Port2**
15..8	**DIO_StaticOutputValue_Port1**
7..0	**DIO_StaticOutputValue_Port0**

PfiPorts Static_Digital_Output_Register (W)

Master DAQ-STC3 Offset: 0x200E0 (NI PCIe-6509 port4..5)
Slave DAQ-STC3 Offset: 0x400E0 (NI PCIe-6509 port6..7)

The data written in this register will be output to the digital lines when:

- The corresponding line is set as an output (PFIDirection_Porti = 1)
- The corresponding PFI Output Select Register is set to 0x10
- The Watchdog Timer functionality is not enabled or its timer has not triggered

Bits	Name
15..8	**PFI_StaticOutputValue_Port1**
7..0	**PFI_StaticOutputValue_Port0**

DioPorts DIO_Direction_Register (W)

Master DAQ-STC3 Offset: 0x204B4 (NI PCIe-6509 port0..3)
Slave DAQ-STC3 Offset: 0x404B4 (NI PCIe-6509 port8..11)

Setting a bit to 1 will result in the corresponding lines on each port to be used for output. Setting a bit to 0 will result in the corresponding lines to be used for input.

Bits	Name
31..24	**DIODirection_Port3**
23..16	**DIODirection_Port2**
15..8	**DIODirection_Port1**
7..0	**DIODirection_Port0**

PfiPorts PFI_Direction_Register (W)

Master DAQ-STC3 Offset: 0x200A4 (NI PCIe-6509 port4..5)
Slave DAQ-STC3 Offset: 0x400A4 (NI PCIe-6509 port6..7)

Setting a bit to 1 will cause the corresponding line to be used for output. Setting a bit to 0 will cause the corresponding line to be used for input.

Bits	Name
15..8	**PFIDirection_Port1**
7..0	**PFIDirection_Port0**

PfiPorts PFI_OutputSelectRegister_i (W)

These registers must be written with 0x10 to enable digital output on the corresponding line. Since the *PFI_Direction_Register* sets the direction, these registers need only be written to once, which can be done on driver load.

PFI Port	NI PCIe-6509 Port	Line	Offset
Master PFI Port 0	Port4	Line0	0x200BA
		Line1	0x200BB
		Line2	0x200BC
		Line3	0x200BD
		Line4	0x200BE
		Line5	0x200BF
		Line6	0x200C0
		Line7	0x200C1
Master PFI Port 1	Port5	Line0	0x200C2
		Line1	0x200C3
		Line2	0x200C4
		Line3	0x200C5
		Line4	0x200C6
		Line5	0x200C7
		Line6	0x200C8
		Line7	0x200C9
Slave PFI Port 0	Port6	Line0	0x400BA
		Line1	0x400BB
		Line2	0x400BC
		Line3	0x400BD
		Line4	0x400BE
		Line5	0x400BF
		Line6	0x400C0
		Line7	0x400C1

PFI Port	NI PCIe-6509 Port	Line	Offset
Slave PFI Port 1	Port7	Line0	0x400C2
		Line1	0x400C3
		Line2	0x400C4
		Line3	0x400C5
		Line4	0x400C6
		Line5	0x400C7
		Line6	0x400C8
		Line7	0x400C9

PFI_Output_Select_t Enumeration

Value	Meaning
0x10	Set to enable digital output on the corresponding line.

Filter Registers

DioPorts DI_FilterRegister_Port0and1

Master DAQ-STC3 DIO Port 0 and 1 Offset: 0x2054C (NI PCIe-6509 port0..1)
Slave DAQ-STC3 DIO Port 0 and 1 Offset: 0x4054C (NI PCIe-6509 port8..9)

The register will set the DI filter for the associated line.

Bits	Name
31..30	**Port1 Line7 Filter Type**
29..28	**Port1 Line6 Filter Type**
27..26	**Port1 Line5 Filter Type**
25..24	**Port1 Line4 Filter Type**
23..22	**Port1 Line3 Filter Type**
21..20	**Port1 Line2 Filter Type**
19..18	**Port1 Line1 Filter Type**
17..16	**Port1 Line0 Filter Type**
15..14	**Port0 Line7 Filter Type**
13..12	**Port0 Line6 Filter Type**
11..10	**Port0 Line5 Filter Type**
9..8	**Port0 Line4 Filter Type**
7..6	**Port0 Line3 Filter Type**
5..4	**Port0 Line2 Filter Type**
3..2	**Port0 Line1 Filter Type**
1..0	**Port0 Line0 Filter Type**

DioPorts DI_FilterRegister_Port2and3

Master DAQ-STC3 DIO Port 2 and 3 Offset: 0x20550 (NI PCIe-6509 port2..3)
Slave DAQ-STC3 DIO Port 2 and 3 Offset: 0x40550 (NI PCIe-6509 port10..11)

The register will set the DI filter for the associated line.

Bits	Name
31..30	**Port3 Line7 Filter Type**
29..28	**Port3 Line6 Filter Type**
27..26	**Port3 Line5 Filter Type**
25..24	**Port3 Line4 Filter Type**
23..22	**Port3 Line3 Filter Type**
21..20	**Port3 Line2 Filter Type**
19..18	**Port3 Line1 Filter Type**
17..16	**Port3 Line0 Filter Type**
15..14	**Port2 Line7 Filter Type**
13..12	**Port2 Line6 Filter Type**
11..10	**Port2 Line5 Filter Type**
9..8	**Port2 Line4 Filter Type**
7..6	**Port2 Line3 Filter Type**
5..4	**Port2 Line2 Filter Type**
3..2	**Port2 Line1 Filter Type**
1..0	**Port2 Line0 Filter Type**

DI_Filter_Select_t Enumeration

Filter Type Value	Result
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.
2	**Medium_Filter**—Medium Filter rejects less than 6.4 µs ; Accepts greater than 12.8 µs ; Delays 9.6 µs ; Adds 6.4 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.

PfiPorts PFI_Filter_Register_Port0Lo

Master DAQ-STC3 Register PFI Port0, low bits: 0x200B0 (NI PCIe-6509 port4)
Slave DAQ-STC3 Register PFI Port0, low bits: 0x400B0 (NI PCIe-6509 port6)

The register will set the DI filter for the associated line.

Bits	Name
15	**Reserved**
14..12	**Line3 Filter Type**
11	**Reserved**
10..8	**Line2 Filter Type**
7	**Reserved**
6..4	**Line1 Filter Type**
3	**Reserved**
2..0	**Line0 Filter Type**

PfiPorts PFI_Filter_Register_Port0Hi

Master DAQ-STC3 Register PFI Port0, high bits: 0x200B2 (NI PCIe-6509 port4)
Slave DAQ-STC3 Register PFI Port0, high bits: 0x400B2 (NI PCIe-6509 port6)

The register will set the DI filter for the associated line.

Bits	Name
15	**Reserved**
14..12	**Line7 Filter Type**
11	**Reserved**
10..8	**Line6 Filter Type**
7	**Reserved**
6..4	**Line5 Filter Type**
3	**Reserved**
2..0	**Line4 Filter Type**

PfiPorts PFI_Filter_Register_Port1Lo

Master DAQ-STC3 Register PFI Port1, low bits: 0x200B4 (NI PCIe-6509 port5)
Slave DAQ-STC3 Register PFI Port1, low bits: 0x400B4 (NI PCIe-6509 port7)

The register will set the DI filter for the associated line.

Bits	Name
15	**Reserved**
14..12	**Line3 Filter Type**
11	**Reserved**
10..8	**Line2 Filter Type**
7	**Reserved**
6..4	**Line1 Filter Type**
3	**Reserved**
2..0	**Line0 Filter Type**

PfiPorts PFI_Filter_Register_Port1Hi

Master DAQ-STC3 Register PFI Port1, high bits: 0x200B6 (NI PCIe-6509 port5)
Slave DAQ-STC3 Register PFI Port1, high bits: 0x400B6 (NI PCIe-6509 port7)

The register will set the DI filter for the associated line.

Bits	Name
15	**Reserved**
14..12	**Line7 Filter Type**
11	**Reserved**
10..8	**Line6 Filter Type**
7	**Reserved**
6..4	**Line5 Filter Type**
3	**Reserved**
2..0	**Line4 Filter Type**

PFI_Filter_Select_t Enumeration

Filter Type Value	Result
0	**No_Filter**—No Filter.
2	**Small_Filter**—Small Filter rejects less than 100 ns ; Accepts greater than 200 ns ; Delays 100 ns ; Adds 25 ns jitter.
3	**Medium_Filter**—Medium Filter rejects less than 6.4 µs ; Accepts greater than 6.425 µs ; Delays 6.4 µs ; Adds 25 ns jitter.
4	**Large_Filter**—Large Filter rejects less than 2.54 ms ; Accepts greather than 2.56 ms ; Delays 2.54 ms ; Adds 10.025 µs jitter.

Change Detect Registers

ChpServices ChangeDetectStatusRegister (R)

Master DAQ-STC3 Offset: 0x20540 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40540 (NI PCIe-6509 port6..11)

This register shows the status of the DI change detection on an DAQ-STC3.

Bits	Name	
31..2	**Reserved**	
1	**ChangeDetectError**—This bit reflects the state of the change detection circuit. When the bit is set, it means that the change detection circuit has issued two consecutive events without an acknowledgement from software (with the ChangeDetectIRQ_Acknowledge bitfield). Clear this bit with the ChangeDetectErrorIRQ_Acknowledge bitfield.	
	Value	**Meaning**
	0	**No errors detected**
	1	**Multiple change detection events detected since last acknowledgement**
0	**ChangeDetectStatus**—This bit reflects the state of the change detection circuit. When the bit is set, it means that the change detection circuit has detected one of the registered events. Clear this bit with the ChangeDetectIRQ_Acknowledge bitfield.	
	Value	**Meaning**
	0	**No change detection event detected**
	1	**Change detection event detected**

DioPorts DI_ChangeIrqRE_Register (W)

Master DAQ-STC3 Offset: 0x20540 (NI PCIe-6509 port0..3)
Slave DAQ-STC3 Offset: 0x40540 (NI PCIe-6509 port8..11)

Set to detect rising edges on corresponding DI line.

Bits	Name
31..24	**DI_ChangeIrqRE_Port3**
23..16	**DI_ChangeIrqRE_Port2**
15..8	**DI_ChangeIrqRE_Port1**
7..0	**DI_ChangeIrqRE_Port0**

DioPorts DI_ChangeIrqFE_Register (W)

Master DAQ-STC3 Offset: 0x20544 (NI PCIe-6509 port0..3)
Slave DAQ-STC3 Offset: 0x40544 (NI PCIe-6509 port8..11)

Set to detect falling edges on corresponding DI line.

Bits	Name
31..24	**DI_ChangeIrqFE_Port3**
23..16	**DI_ChangeIrqFE_Port2**
15..8	**DI_ChangeIrqFE_Port1**
7..0	**DI_ChangeIrqFE_Port0**

PfiPorts PFI_ChangeIrq_Register (W)

Master DAQ-STC3 Offset: 0x20548 (NI PCIe-6509 port4..5)
Slave DAQ-STC3 Offset: 0x40548 (NI PCIe-6509 port6..7)

Set to detect rising or falling edges on the corresponding PFI line.

Bits	Name
31..24	**PFI_ChangeIrqFE_Port1**
23..16	**PFI_ChangeIrqFE_Port0**
15..8	**PFI_ChangeIrqRE_Port1**
7..0	**PFI_ChangeIrqRE_Port0**

DioPorts DI_ChangeDetectLatched_Register (R)

Master DAQ-STC3 Offset: 0x20544 (NI PCIe-6509 port0..3)
Slave DAQ-STC3 Offset: 0x40544 (NI PCIe-6509 port8..11)

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 detection circuit on this DAQ-STC3. Software must check the Change Detection Error bit to make sure the data being read is valid and coherent (for the case of reading DI and PFI registers).

Bits	Name
31..24	**DI_ChangeDetectLatched_Port3**
23..16	**DI_ChangeDetectLatched_Port2**
15..8	**DI_ChangeDetectLatched_Port1**
7..0	**DI_ChangeDetectLatched_Port0**

PfiPorts PFI_ChangeDetectLatched_Register (R)

Master DAQ-STC3 Offset: 0x20548 (NI PCIe-6509 port4..5)
Slave DAQ-STC3 Offset: 0x40548 (NI PCIe-6509 port6..7)

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 detection circuit on this DAQ-STC3. Software must check the Change Detection Error bit to make sure the data being read is valid and coherent (for the case of reading DI and PFI registers).

Bits	Name
15..8	**PFI_ChangeDetectLatched_Port1**
7..0	**PFI_ChangeDetectLatched_Port0**

Watchdog Timer Registers

DioPorts DO_WDT_SafeStateRegister (W)

Master DAQ-STC3 Offset: 0x204D0 (NI PCIe-6509 port0..3)
Slave DAQ-STC3 Offset: 0x404D0 (NI PCIe-6509 port8..11)

This is the safe value that the DIO lines will transition to as a result of an expiration of the Watchdog Timer, if the DIO lines have their safe state enabled (through *DO_WDT_ModeSelect_Port0and1_Register* and *DO_WDT_ModeSelect_Port2and3_Register*).

Bits	Name
31..24	**DO_WDT_SafeStateValue_Port3**
23..16	**DO_WDT_SafeStateValue_Port2**
15..8	**DO_WDT_SafeStateValue_Port1**
7..0	**DO_WDT_SafeStateValue_Port0**

PfiPorts PFI_WDT_SafeStateRegister (W)

Master DAQ-STC3 Offset: 0x200E2 (NI PCIe-6509 port4..5)
Slave DAQ-STC3 Offset: 0x400E2 (NI PCIe-6509 port6..7)

This is the safe value that the PFI lines will transition to as a result of an expiration of the Watchdog Timer, if the PFI lines have their safe state enabled (through the *PFI_WDT_ModeSelect_Register*).

Bits	Name
15..8	**PFI_WDT_SafeStateValue_Port1**
7..0	**PFI_WDT_SafeStateValue_Port0**

DioPorts DO_WDT_ModeSelect_Port0and1_Register (W)

Master DAQ-STC3 Register Port 0 and 1: 0x204D4 (NI PCIe-6509 port0..1)
Slave DAQ-STC3 Register Port 0 and 1: 0x404D4 (NI PCIe-6509 port8..9)

The register sets the mode that each line should enter when the Watchdog Timer is triggered.

Bits	Name
31..30	**Port1 Line7 Watchdog Mode**
29..28	**Port1 Line6 Watchdog Mode**
27..26	**Port1 Line5 Watchdog Mode**
25..24	**Port1 Line4 Watchdog Mode**
23..22	**Port1 Line3 Watchdog Mode**
21..20	**Port1 Line2 Watchdog Mode**
19..18	**Port1 Line1 Watchdog Mode**
17..16	**Port1 Line0 Watchdog Mode**
15..14	**Port0 Line7 Watchdog Mode**
13..12	**Port0 Line6 Watchdog Mode**
11..10	**Port0 Line5 Watchdog Mode**
9..8	**Port0 Line4 Watchdog Mode**
7..6	**Port0 Line3 Watchdog Mode**
5..4	**Port0 Line2 Watchdog Mode**
3..2	**Port0 Line1 Watchdog Mode**
1..0	**Port0 Line0 Watchdog Mode**

DioPorts DO_WDT_ModeSelect_Port2and3_Register (W)

Master DAQ-STC3 Register Port 2 and 3: 0x204D8 (NI PCIe-6509 port2..3)
Slave DAQ-STC3 Register Port 2 and 3: 0x404D8 (NI PCIe-6509 port10..11)

The register sets the mode that each line should enter when the Watchdog Timer is triggered.

Bits	Name
31..30	**Port3 Line7 Watchdog Mode**
29..28	**Port3 Line6 Watchdog Mode**
27..26	**Port3 Line5 Watchdog Mode**
25..24	**Port3 Line4 Watchdog Mode**
23..22	**Port3 Line3 Watchdog Mode**
21..20	**Port3 Line2 Watchdog Mode**
19..18	**Port3 Line1 Watchdog Mode**
17..16	**Port3 Line0 Watchdog Mode**
15..14	**Port2 Line7 Watchdog Mode**
13..12	**Port2 Line6 Watchdog Mode**
11..10	**Port2 Line5 Watchdog Mode**
9..8	**Port2 Line4 Watchdog Mode**
7..6	**Port2 Line3 Watchdog Mode**
5..4	**Port2 Line2 Watchdog Mode**
3..2	**Port2 Line1 Watchdog Mode**
1..0	**Port2 Line0 Watchdog Mode**

DO_WDT_Mode_t Enumeration

Watchdog Mode Value	Result
0	**WDT_Disabled**
1	**WDT_Freeze**
2	**WDT_Tristate**
3	**WDT_SafeValue**

PfiPorts PFI_WDT_ModeSelect_Register (W)

Master DAQ-STC3 Offset: 0x200E4 (NI PCIe-6509 port4..5)
Slave DAQ-STC3 Offset: 0x400E4 (NI PCIe-6509 port6..7)

The register sets the mode that each line should enter when the Watchdog Timer is triggered.

Bits	Name
31..30	**Port1 Line7 Watchdog Mode**
29..28	**Port1 Line6 Watchdog Mode**
27..26	**Port1 Line5 Watchdog Mode**
25..24	**Port1 Line4 Watchdog Mode**
23..22	**Port1 Line3 Watchdog Mode**
21..20	**Port1 Line2 Watchdog Mode**
19..18	**Port1 Line1 Watchdog Mode**
17..16	**Port1 Line0 Watchdog Mode**
15..14	**Port0 Line7 Watchdog Mode**
13..12	**Port0 Line6 Watchdog Mode**
11..10	**Port0 Line5 Watchdog Mode**
9..8	**Port0 Line4 Watchdog Mode**
7..6	**Port0 Line3 Watchdog Mode**
5..4	**Port0 Line2 Watchdog Mode**
3..2	**Port0 Line1 Watchdog Mode**
1..0	**Port0 Line0 Watchdog Mode**

PFI_WDT_Mode_t Enumeration

Watchdog Mode Value	Result
0	**WDT_Disabled**
1	**WDT_Freeze**
2	**WDT_Tristate**
3	**WDT_SafeValue**

ChpServices WatchdogStatusRegister (R)

Master DAQ-STC3 Offset: 0x20068 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40068 (NI PCIe-6509 port6..11)

This register shows the current state of the Watchdog Timer for each DAQ-STC3.

Bits	Name
31..16	**Reserved**
15..8	**WatchdogExpiredCnt**—This bitfield reflects how many times the Watchdog Timer has expired since the last reset. This will include internal and external triggers if enabled.
7..3	**Reserved**
2..0	**WatchdogSM_State**—This bitfield reflects the status of the Watchdog state machine.

ChpSrv_WatchdogTimerStateMachineSt_t Enumeration

Watchdog State Value	Result
0	**WdtSt_SynchReset**
1	**WdtSt_CountDownFEED**
2	**WdtSt_CountDownF00D**
3	**WdtSt_Sleeping**
5	**WdtSt_ExpiredPulse**
6	**WdtSt_Expired**

ChpServices WatchdogTimeoutRegister (W)

Master DAQ-STC3 Offset: 0x20068 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40068 (NI PCIe-6509 port6..1)

Bits	Name
31..0	**WatchdogTimeoutValue**—This bitfield sets a 32-bit timeout value for the Watchdog Timer in the DAQ-STC3. The value represents the number of bus clock periods that the counter will count before expiring if it does not get acknowledged from software. The bus clock frequency is 31.25 MHz. When the timer expires, it can generate a pulse that can be exported to RTSI. The modules receiving this pulse can take action such as placing the outputs on a safe state.

ChpServices WatchdogConfiguration (W)

Master DAQ-STC3 Offset: 0x2006C (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x4006C (NI PCIe-6509 port6..11)

This register sets the expiration for the Watchdog Timer to come from the internal timer or an external pulse.

Bits	Name	
15..10	**Reserved**	
9	**IntTrigEn**—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.	
	Value	**Meaning**
	0	**Disabled**
	1	**Use internal timer for the Watchdog Timer**
8	**ExtTrigEn**—This bit enables the generation of an Expired Trigger pulse from the external trigger source. When this bit is set, the Watchdog Timer will generate a trigger any time that it detects a rising edge in the selected external trigger (after polarity selection).	
	Value	**Meaning**
	0	**Disabled**
	1	**Use external trigger specified by ExtTrigSel for the Watchdog Timer**
7	**ExtTrigPol**—This bit selects the polarity of the external trigger signal.	
	Value	**Meaning**
	0	**Active high**
	1	**Active low**
6..3	**Reserved**	
2..0	**ExtTrigSel**—This bitfield selects the source for the external trigger source.	

ChpServ_WatchdogTimerExtSrcSel_t Enumeration

ExtTrigSel Value	Result
0	**RTSI0**
1	**RTSI1**
2	**RTSI2**
3	**RTSI3**
4	**RTSI4**
5	**RTSI5**
6	**RTSI6**
7	**RTSI7**

ChpServices WatchdogControl (W)

Master DAQ-STC3 Offset: 0x2006E (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x4006E (NI PCIe-6509 port6..11)

This register is used to change the state of the Watchdog Timer.

Bits	Name	
15..0	**WatchdogCommand**	
	Watchdog Control	**Result**
	0x5678	Used to start the Watchdog Timer sequence. Must write alternating 0xFEED and 0xF00D to restart the timer. Writing 0xFEED and 0xF00D out of order will also result in the timer expiring.
	0xFEED	Restarts the Watchdog Timer when written in sequence with 0xF00D.
	0xF00D	Restarts the Watchdog Timer when written in sequence with 0xFEED.
	0x1234	Pauses the Watchdog Timer. After restarting with 0x5678, 0xFEED and 0xF00D should resume in sequence.
	0xDEAD	Causes the Watchdog Timer to expire immediately. The Watchdog Timer can be restarted by pausing and starting the timer or by using the restart command.
	0xACED	Restarts the Watchdog Timer.

Interrupt Registers

ChpServices GlobalInterruptStatus_Register (R)

Master DAQ-STC3 Offset: 0x20070 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40070 (NI PCIe-6509 port6..11)

The bitfields in this register indicate if any enabled interrupts in the applicable DI or Watchdog Timer groups have occurred.

Bits	Name
15..11	**Reserved**
10	**WatchdogTimer_Interrupt_Status**—This bitfield is set to 1 if any enabled interrupt on the Watchdog Timer Group occurs and is enabled.
9..7	**Reserved**
6	**DI_Interrupt_Status**—This bitfield is set to 1 if any enabled interrupt on the DI subsystem occurs and is enabled.
5..0	**Reserved**

ChpServices GlobalInterruptEnable_Register (W)

Master DAQ-STC3 Offset: 0x20078 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40078 (NI PCIe-6509 port6..11)

The bitfields in this register allow for the enabling and disabling of interrupts for the DI or Watchdog Timer groups.

Bits	Name
31..27	**Reserved**
26	**WatchdogTimer_Interrupt_Disable**—Write 1 to this field to block all interrupts in the Watchdog Timer group from propagating to the CHInCh.
25..23	**Reserved**
22	**DI_Interrupt_Disable**—Write 1 to this field to block interrupts in the DI subsystem from propagating to the CHInCh.
21..11	**Reserved**
10	**WatchdogTimer_Interrupt_Enable**—Write 1 to this field to allow the interrupts in the Watchdog Timer group to propagate to the CHInCh.
9..7	**Reserved**

Bits	Name
6	**DI_Interrupt_Enable**—Write 1 to this field to allow the interrupts in the DI subsystem to propagate to the CHInCh.
5..0	**Reserved**

ChpServices DI_Interrupt_Status_Register (R)

Master DAQ-STC3 Offset: 0x2007E (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x4007E (NI PCIe-6509 port6..11)

The bitfields in this register reflect the status of the individual interrupts associated with the DI change detection group. These bitfields are set to 1 if the interrupt is enabled, has occurred, and has not been acknowledged.

Bits	Name
15..2	**Reserved**
1	**ChangeDetectionErrorIrqSt**—This bitfield is 1 only if the interrupt occurred, the interrupt is enabled, and the interrupt has not been acknowledged.
0	**ChangeDetectionIrqSt**—This bitfield is 1 only if the interrupt occurred, the interrupt is enabled, and the interrupt has not been acknowledged.

ChpServices ChangeDetectIRQ_Register (W)

Master DAQ-STC3 Offset: 0x20554 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40554 (NI PCIe-6509 port6..11)

The bitfields in this register allow for the acknowledging, enabling, and disabling of the change detection interrupts and errors. These bits also affect the *ChangeDetectStatusRegister*.

Bits	Name
31..8	**Reserved**
7	**ChangeDetectErrorIRQ_Enable**—This interrupt fires when two Change Detect interrupts are generated without an acknowledge.
6	**ChangeDetectErrorIRQ_Disable**—Disables the Change Detect interrupt.
5	**ChangeDetectIRQ_Enable**—This interrupt fires when the Change Detect circuit detects one of the registered events.
4	**ChangeDetectIRQ_Disable**—Disables the Change Detect interrupt.
3..2	**Reserved**

Bits	Name
1	**ChangeDetectErrorIRQ_Acknowledge**—Acknowledges the Change Detect Error interrupt and clears the status bit.
0	**ChangeDetectIRQ_Acknowledge**—Acknowledges the Change Detect interrupt and clears the status bit.

ChpServices WatchdogTimer_Interrupt_Status_Register (R)

Master DAQ-STC3 Offset: 0x20086 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40086 (NI PCIe-6509 port6..11)

The bitfields in this register reflect the status of the individual interrupts in the Watchdog Timer group.

Bits	Name
15..1	**Reserved**
0	**WatchdogTimerTriggerSt**—This bitfield is 1 only if the interrupt is enabled, has occurred, and has not been acknowledged.

ChpServices WatchdogTimer_Interrupt1_Register (W)

Master DAQ-STC3 Offset: 0x20070 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40070 (NI PCIe-6509 port6..11)

The bitfields in this register allow for the enabling and acknowledging of interrupts for the Watchdog Timer.

Bits	Name
31..17	**Reserved**
16	**WDT_TriggerIRQ_Ack**—This bit clears the WDT_TruggerIRQ Interrupt event and acknowledges the interrupt.
15..1	**Reserved**
0	**WDT_TriggerIRQ_Enable**—This strobe bit enables the WDT_TriggerIRQ interrupt.

ChpServices WatchdogTimer_Interrupt2_Register (W)

Master DAQ-STC3 Offset: 0x20074 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40074 (NI PCIe-6509 port6..11)

The bitfields in this register allow for the disabling and acknowledging of interrupts for the Watchdog Timer.

Bits	Name
31..17	**Reserved**
16	**WDT_TriggerIRQ_Ack2**—This bit clears the WDT_TriggerIRQ Interrupt event and acknowledges the interrupt.
15..1	**Reserved**
0	**WDT_TriggerIRQ_Disable**—This strobe bit disables the WDT_TriggerIRQ interrupt.

CHInCh Interrupt_Mask_Register (R|W)

Offset: 0x5C

This register controls the board level interrupts.

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 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 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..0	**Reserved**

CHInCh Interrupt_Status_Register (R)

Offset: 0x60

This register returns the interrupt status word for the highest priority interrupt pending.

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 bitfield returns 1 when at least one additional interrupt is pending beyond the interrupt status word being returned.
29	**External**—This bitfield returns 1 when the highest priority interrupt is external to the CHInCh. A more specific interrupt status must be read from the DAQ-STC3. 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.
28..12	**Reserved**
11	**DAQ-STC3 Int**—This bitfield returns 1 when the DAQ-STC3 interrupt line and external interrupts are the highest priority condition pending.
10..0	**Reserved**

CHInCh Volatile_Interrupt_Status_Register (R)

Offset: 0x68

This register returns the interrupt status word for the highest priority interrupt pending. After reading this register, its contents automatically clear.

Bits	Name
31	**Vol_Int**—This bitfield returns 1 when the CHInCh has a pending interrupt. The 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	**Vol_Additional_Int**—This bitfield returns 1 when at least one additional interrupt is pending beyond the interrupt status word being returned.
29	**Vol_External**—This bitfield returns 1 when the highest priority interrupt is external to the CHInCh. A more specific interrupt status must be read from the DAQ-STC3. 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.
28..12	**Reserved**
11	**Vol_STC3_Int**—This bitfield returns 1 when the DAQ-STC3 interrupt line and external interrupts are the highest priority condition pending.
10..0	**Reserved**

ChpServices IntForwarding_ControlStatus (R|W)

Master DAQ-STC3 Offset: 0x22204 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x42204 (NI PCIe-6509 port6..11)

This register allows for the enabling and resetting of the DAQ-STC3 to forward interrupts to the CHInCh.

Bits	Name
31..2	**Reserved**
1	**IntForwarding_Reset**—When this bit is written with a 1, the interrupt forwarding will be synchronously reset. The interrupt forwarding is not completely reset until the IntForwarding_Enable bit returns a 0.
0	**IntForwarding_Enable**—Writing a 1 to this bit enables the interrupt forwarding to the CHInCh. This bit will return a 1 when interrupt forwarding is enabled and a 0 when it is disabled. This bit clears on reset and when the IntForwarding_Reset bit is written with a 1.

ChpServices IntForwarding_DestinationReg (R|W)

Master DAQ-STC3 Offset: 0x22208 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x42208 (NI PCIe-6509 port6..11)

This register configures the destination on the CHInCh to which the DAQ-STC3 should forward interrupts. The master DAQ-STC3 should set IntForwarding_Destination to 0, while the slave DAQ-STC3 should set it to decimal 24.

Bits	Name
31..8	**Reserved**
7..0	**IntForwarding_Destination**

RTSI Configuration Registers

ChpServices RTSI_Trig_Direction_Register (W)

Master DAQ-STC3 Offset: 0x200A6 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x400A6 (NI PCIe-6509 port6..11)

These bits select the output of the bidirectional RTSI pins.

Bits	Name
15..8	**RTSIDirection**—Corresponds to the RTSI0..7 (bit 15 = RTSI7; bit 8 = RTSI0). Setting a bit to 1 results in the corresponding line to be used for output. Setting a bit to 0 results in the corresponding line being used for input.
7..0	**Reserved**

ChpServices RTSI_OutputSelectRegister_i (W)

The register must be set when the corresponding RTSI line is set to output.

STC3	RTSI Line	Offset
Master	Line0	0x200A8
	Line1	0x200A9
	Line2	0x200AA
	Line3	0x200AB
	Line4	0x200AC
	Line5	0x200AD
	Line6	0x200AE
	Line7	0x200AF
Slave	Line0	0x400A8
	Line1	0x400A9
	Line2	0x400AA
	Line3	0x400AB
	Line4	0x400AC
	Line5	0x400AD
	Line6	0x400AE
	Line7	0x400AF

RTSI_Output_Select_t Enumeration

Value	Result
13	Export DIO Change Detection Signal
14	Export Watchdog Timer Expired Signal

Miscellaneous Registers

CHInCh CHInCh_Identification_Register (R)

Offset: 0x00

This register is used to identify the CHInCh. The value read is 0xC0107AD0 (–1072661808).

Bits	Name
31..0	**ID**—This field is used to identify the CHInCh.

CHInCh Scrap_Register (R|W)

Offset: 0x200

Bits	Name
31..0	**SDATA**—This field does not affect hardware operations and resets to an unknown value. It can be used to verify read/write access to the NI PCIe-6509 registers.

CHInCh PCI_Subsystem_ID_Access_Register (R)

Offset: 0x10AC

This register returns the VID of the sub-system: 0x1093 for National Instruments and returns the PID of the subsystem: 0x7326 for NI PCIe-6509.

Bits	Name
31..16	**SubSystem_Product_ID**—This field returns the PID of the NI PCIe device. Use this field to determine which specific X Series model is being used.
15..0	**SubSystem_Vendor_ID**—This field returns the VID of the subs-system: 0x1093 for National Instruments.

ChpServices ScratchPadRegister (R|W)

Master DAQ-STC3 Offset: 0x20004 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40004 (NI PCIe-6509 port6..11)

This is a simple readable and writable register included for self-test.

Bits	Name
31..0	**Scratch_Pad**—This is a register included for self-test.

ChpServices Signature_Register (R)

Master DAQ-STC3 Offset: 0x20060 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40060 (NI PCIe-6509 port6..11)

Bits	Name
31..0	**STC3**—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 this register is 0x08050509 for revision A DAQ-STC3 ASICs and 0x08050501 for revision B DAQ-STC3 ASICs.

ChpServices Joint_Reset_Register (W)

Master DAQ-STC3 Offset: 0x20064 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40064 (NI PCIe-6509 port6..11)

Bits	Name
15..1	**Reserved**
0	**Software_Reset**—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.

ChpServices TimeSincePowerUpRegister (R)

Master DAQ-STC3 Offset: 0x20064 (NI PCIe-6509 port0..5)
Slave DAQ-STC3 Offset: 0x40064 (NI PCIe-6509 port6..11)

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 is 0.65536 ms $\times n$.

A Register Maps Appendix

This appendix contains all of the register maps mentioned in this manual with their offsets (hi and lo, where applicable), a reference to the appropriate section containing the register, and the exact register name.

CHInCh Chip Object

Offset	Section Reference	Register Name
0x00000	Miscellaneous Registers	CHInCh_Identification_Register
0x0005C	Interrupt Registers	Interrupt_Mask_Register
0x00060	Interrupt Registers	Interrupt_Status_Register
0x00068	Interrupt Registers	Volatile_Interrupt_Status_Register
0x00200	Miscellaneous Registers	Scrap_Register
0x010AC	Miscellaneous Registers	PCI_Subsystem_ID_Access_Register

ChpServices Chip Object

LoOffset	HiOffset	Section Reference	Register Name
0x20004	0x40004	Miscellaneous Registers	ScratchPadRegister
0x20060	0x40060	Miscellaneous Registers	Signature_Register
0x20064	0x40064	Miscellaneous Registers	Joint_Reset_Register
0x20064	0x40064	Miscellaneous Registers	TimeSincePowerUpRegister
0x20068	0x40068	Watchdog Timer Registers	WatchdogStatusRegister
0x20068	0x40068	Miscellaneous Registers	WatchdogTimeoutRegister

LoOffset	HiOffset	Section Reference	Register Name
0x2006C	0x4006C	Miscellaneous Registers	WatchdogConfiguration
0x2006E	0x4006E	Miscellaneous Registers	WatchdogControl
0x20070	0x40070	Interrupt Registers	GlobalInterruptStatus_Register
0x20070	0x40070	Interrupt Registers	WatchdogTimer_Interrupt1_Register
0x20074	0x40074	Interrupt Registers	WatchdogTimer_Interrupt2_Register
0x20078	0x40078	Interrupt Registers	GlobalInterruptEnable_Register
0x2007E	0x4007E	Interrupt Registers	DI_Interrupt_Status_Register
0x20086	0x40086	Interrupt Registers	WatchdogTimer_Interrupt_Status_Register
0x200A6	0x400A6	RTSI Configuration Registers	RTSI_Trig_Direction_Register
0x200A8	0x400A8	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[0]
0x200A9	0x400A9	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[1]
0x200AA	0x400AA	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[2]
0x200AB	0x400AB	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[3]
0x200AC	0x400AC	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[4]
0x200AD	0x400AD	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[5]
0x200AE	0x400AE	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[6]
0x200AF	0x400AF	RTSI Configuration Registers	RTSI_OutputSelectRegister_i[7]
0x20540	0x40540	Change Detect Registers	ChangeDetectStatusRegister
0x22204	0x42204	Interrupt Registers	IntForwarding_ControlStatus
0x22208	0x42208	Interrupt Registers	IntForwarding_DestinationReg

DioPorts Chip Object

LoOffset	HiOffset	Section Reference	Register Name
0x204B0	0x404B0	Static DIO Registers	Static_Digital_Output_Register
0x204B4	0x404B4	Static DIO Registers	DIO_Direction_Register
0x204D0	0x404D0	Watchdog Timer Registers	DO_WDT_SafeStateRegister
0x204D4	0x404D4	Watchdog Timer Registers	DO_WDT_ModeSelect_Port0and1_Register
0x204D8	0x404D8	Watchdog Timer Registers	DO_WDT_ModeSelect_Port2and3_Register
0x20530	0x40530	Static DIO Registers	Static_Digital_Input_Register
0x20540	0x40540	Change Detect Registers	DI_ChangeIrqRE_Register
0x20544	0x40544	Change Detect Registers	DI_ChangeIrqFE_Register
0x20544	0x40544	Change Detect Registers	DI_ChangeDetectLatched_Register
0x2054C	0x4054C	Filter Registers	DI_FilterRegister_Port0and1
0x20550	0x40550	Filter Registers	DI_FilterRegister_Port2and3

PfiPorts Chip Object

LoOffset	HiOffset	Section Reference	Register Name
0x200A4	0x400A4	Static DIO Registers	PFI_Direction_Register
0x200B0	0x400B0	Filter Registers	PFI_Filter_Register_Port0Lo
0x200B2	0x400B2	Filter Registers	PFI_Filter_Register_Port0Hi
0x200B4	0x400B4	Filter Registers	PFI_Filter_Register_Port1Lo
0x200B6	0x400B6	Filter Registers	PFI_Filter_Register_Port1Hi
0x200BA	0x400BA	Static DIO Registers	PFI_OutputSelectRegister_i[0]

LoOffset	HiOffset	Section Reference	Register Name
0x200BB	0x400BB	Static DIO Registers	PFI_OutputSelectRegister_i[1]
0x200BC	0x400BC	Static DIO Registers	PFI_OutputSelectRegister_i[2]
0x200BD	0x400BD	Static DIO Registers	PFI_OutputSelectRegister_i[3]
0x200BE	0x400BE	Static DIO Registers	PFI_OutputSelectRegister_i[4]
0x200BF	0x400BF	Static DIO Registers	PFI_OutputSelectRegister_i[5]
0x200C0	0x400C0	Static DIO Registers	PFI_OutputSelectRegister_i[6]
0x200C1	0x400C1	Static DIO Registers	PFI_OutputSelectRegister_i[7]
0x200C2	0x400C2	Static DIO Registers	PFI_OutputSelectRegister_i[8]
0x200C3	0x400C3	Static DIO Registers	PFI_OutputSelectRegister_i[9]
0x200C4	0x400C4	Static DIO Registers	PFI_OutputSelectRegister_i[10]
0x200C5	0x400C5	Static DIO Registers	PFI_OutputSelectRegister_i[11]
0x200C6	0x400C6	Static DIO Registers	PFI_OutputSelectRegister_i[12]
0x200C7	0x400C7	Static DIO Registers	PFI_OutputSelectRegister_i[13]
0x200C8	0x400C8	Static DIO Registers	PFI_OutputSelectRegister_i[14]
0x200C9	0x400C9	Static DIO Registers	PFI_OutputSelectRegister_i[15]
0x200E0	0x400E0	Static DIO Registers	Static_Digital_Input_Register
0x200E0	0x400E0	Static DIO Registers	Static_Digital_Output_Register
0x200E2	0x400E2	Watchdog Timer Registers	PFI_WDT_SafeStateRegister
0x200E4	0x400E4	Watchdog Timer Registers	PFI_WDT_ModeSelect_Register
0x20548	0x40548	Change Detect Registers	PFI_ChangeIrq_Register
0x20548	0x40548	Change Detect Registers	PFI_ChangeDetectLatched_Register

B Technical Support and Professional Services

Log in to your National Instruments `ni.com` User Profile to get personalized access to your services. Visit the following sections of `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 `ni.com/forums`. 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.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support, as well as exclusive access to eLearning training modules at `ni.com/elearning`. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.

    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 training and certification program information. You can also 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.