DAQ DIO Register Map

Version 0.1 Tim Ousley

This register map is a combination of the register fields from the DIO rbm file and Vince's internal DIO documentation.

Registers by Address

WindowDataRegister	Write	32 bit	Offset 0x000
WindowAddressRegister	Write	8 bit	Offset 0x004
MasterInterruptControl	Write	8 bit	Offset 0x005
Group1ClearRegisterA	Write	8 bit	Offset 0x006
Group2ClearRegisterA	Write	8 bit	Offset 0x007
Group1FifoWriteARegister	Write	8 bit	Offset 0x008
Group1FifoWriteBRegister	Write	8 bit	Offset 0x009
Group1FifoWriteCRegister	Write	8 bit	Offset 0x00A
Group1FifoWriteDRegister	Write	8 bit	Offset 0x00B
Group2FifoWriteARegister	Write	8 bit	Offset 0x00C
Group2FifoWriteBRegister	Write	8 bit	Offset 0x00D
Group2FifoWriteCRegister	Write	8 bit	Offset 0x00E
Group2FifoWriteDRegister	Write	8 bit	Offset 0x00F
FifoWriteARegister	Write	8 bit	Offset 0x010
FifoWriteBRegister	Write	8 bit	Offset 0x011
FifoWriteCRegister	Write	8 bit	Offset 0x012
FifoWriteDRegister	Write	8 bit	Offset 0x013
Group1TransferCountRegister	Write	32 bit	Offset 0x014
Group2TransferCountRegister	Write	32 bit	Offset 0x018
PortWriteARegister	Write	8 bit	Offset 0x01C
PortWriteBRegister	Write	8 bit	Offset 0x01D
PortWriteCRegister	Write	8 bit	Offset 0x01E
PortWriteDRegister	Write	8 bit	Offset 0x01F
PortDirectionARegister	Write	8 bit	Offset 0x020
PortDirectionBRegister	Write	8 bit	Offset 0x021
PortDirectionCRegister	Write	8 bit	Offset 0x022
PortDirectionDRegister	Write	8 bit	Offset 0x023
PortMaskARegister	Write	8 bit	Offset 0x024
PortMaskBRegister	Write	8 bit	Offset 0x025
PortMaskCRegister	Write	8 bit	Offset 0x026
PortMaskDRegister	Write	8 bit	Offset 0x027
PortPolarityARegister	Write	8 bit	Offset 0x028
PortPolarityBRegister	Write	8 bit	Offset 0x029
PortPolarityCRegister	Write	8 bit	Offset 0x02A
PortPolarityDRegister	Write	8 bit	Offset 0x02B
GroupSynchronization	Write	8 bit	Offset 0x02C
MasterClockRouting	Write	8 bit	Offset 0x02D
Group1ClearRegisterB	Write	8 bit	Offset 0x02E
Group2ClearRegisterB	Write	8 bit	Offset 0x02F
PortPatternARegister	Write	8 bit	Offset 0x030
PortPatternBRegister	Write	8 bit	Offset 0x031
PortPatternCRegister	Write	8 bit	Offset 0x032
PortPatternDRegister	Write	8 bit	Offset 0x033
Group1DataPath	Write	8 bit	Offset 0x040
Group1OperatingMode	Write	8 bit	Offset 0x041
Group1StartupAndClocking	Write	8 bit	Offset 0x042
Group1HandshakeSequence	Write	8 bit	Offset 0x043
Group1ClockSpeedRegister	Write	16 bit	Offset 0x044
Group1ReqManagement	Write	8 bit	Offset 0x046
Group1FrameControl	Write	8 bit	Offset 0x047
Group1FifoControl	Write	8 bit	Offset 0x048

Group1LinePolarities	Write	8 bit	Offset 0x049
Group1SerialAndAckControl	Write	8 bit	Offset 0x04A
Group1InterruptControl	Write	8 bit	Offset 0x04B
Group1DmaLineControl	Write	8 bit	Offset 0x04C
Group1TransferSize	Write	8 bit	Offset 0x04D
Group1ConditioningDelayRegister	Write	8 bit	Offset 0x04E
Group1DaqOptions	Write	8 bit	Offset 0x04F
Group1CounterControl	Write	8 bit	Offset 0x050
Group1PatternDetection	Write	8 bit	Offset 0x051
Group1ReqDelayRegister	Write	8 bit	Offset 0x052
Group1ReqNotDelayRegister	Write	8 bit	Offset 0x053
Group1AckDelayRegister	Write	8 bit	Offset 0x054
Group1AckNotDelayRegister	Write	8 bit	Offset 0x055
Group1Data1DelayRegister	Write	8 bit	Offset 0x056
Group1Data2DelayRegister	Write	8 bit	Offset 0x057
Group1StartDelayRegister	Write	32 bit	Offset 0x058
Group2DataPath	Write	8 bit	Offset 0x060
Group2OperatingMode	Write	8 bit	Offset 0x061
Group2StartupAndClocking	Write	8 bit	Offset 0x062
Group2HandshakeSequence	Write	8 bit	Offset 0x063
Group2ClockSpeedRegister	Write	16 bit	Offset 0x064
Group2ReqManagement	Write	8 bit	Offset 0x066
Group2FrameControl	Write	8 bit	Offset 0x067
Group2FifoControl	Write	8 bit	Offset 0x068
Group2LinePolarities	Write	8 bit	Offset 0x069
Group2SerialAndAckControl	Write	8 bit	Offset 0x06A
Group2InterruptControl	Write	8 bit	Offset 0x06B
Group2DmaLineControl	Write	8 bit	Offset 0x06C
Group2TransferSize	Write	8 bit	Offset 0x06D
Group2ConditioningDelayRegister	Write	8 bit	Offset 0x06E
Group2DaqOptions	Write	8 bit	Offset 0x06F
Group2CounterControl	Write	8 bit	Offset 0x070
Group2PatternDetection	Write	8 bit	Offset 0x071
Group2ReqDelayRegister	Write	8 bit	Offset 0x072
Group2ReqNotDelayRegister	Write	8 bit	Offset 0x073
Group2AckDelayRegister	Write	8 bit	Offset 0x074
Group2AckNotDelayRegister	Write	8 bit	Offset 0x075
Group2Data1DelayRegister	Write	8 bit	Offset 0x076
Group2Data2DelayRegister	Write	8 bit	Offset 0x077
Group2StartDelayRegister	Write	32 bit	Offset 0x078

WindowDataRegister	Read	32 bit	Offset 0x000
InterruptStatus	Read	8 bit	Offset 0x004
GroupStatus	Read	8 bit	Offset 0x005
Group1Flags	Read	8 bit	Offset 0x006
Group2Flags	Read	8 bit	Offset 0x007
Group1FifoReadARegister	Read	8 bit	Offset 0x008
Group1FifoReadBRegister	Read	8 bit	Offset 0x009
Group1FifoReadCRegister	Read	8 bit	Offset 0x00A
Group1FifoReadDRegister	Read	8 bit	Offset 0x00B
Group2FifoReadARegister	Read	8 bit	Offset 0x00C
Group2FifoReadBRegister	Read	8 bit	Offset 0x00D
Group2FifoReadCRegister	Read	8 bit	Offset 0x00E
Group2FifoReadDRegister	Read	8 bit	Offset 0x00F
FifoReadARegister	Read	8 bit	Offset 0x010
FifoReadBRegister	Read	8 bit	Offset 0x011
FifoReadCRegister	Read	8 bit	Offset 0x012
FifoReadDRegister	Read	8 bit	Offset 0x013

FifoStatus	Read	8	bit	Offset 0x014
ChipIdARegister	Read	8	bit	Offset 0x018
ChipIdBRegister	Read	8	bit	Offset 0x019
ChipIdCRegister	Read	8	bit	Offset 0x01A
ChipIdDRegister	Read	8	bit	Offset 0x01B
PortReadARegister	Read	8	bit	Offset 0x01C
PortReadBRegister	Read	8	bit	Offset 0x01D
PortReadCRegister	Read	8	bit	Offset 0x01E
PortReadDRegister	Read	8	bit	Offset 0x01F

Global Registers

ChipIdARegister **Read 0x 018**

Chip Id A	Chip Id A	Chip Id A	Chip Id A	Chip Id A	Chip Id A	Chip Id A	Chip Id A

ChipIdBRegister **Read 0x 019**

Chip Id B	Chip Id B	Chip Id B	Chip Id B	Chip Id B	Chip Id B	Chip Id B	Chip Id B

ChipIdCRegister **Read 0x 01A**

Chip Id C	Chip Id C	Chip Id C	Chip Id C	Chip Id C	Chip Id C	Chip Id C	Chip Id C

ChipIdDRegister **Read 0x 01B**

Chip Id D	Chip Id D	Chip Id D	Chip Id D	Chip Id D	Chip Id D	Chip Id D	Chip Id D

“DIO” in ASCII, then a version number for the DAQ-DIO.

Chip Id A-
    “D”
Chip Id B-
    “I”
Chip Id C-
    “O”
Chip Id D-
    Version number for the DAQ-DIO

FifoRead*i*Register **Read 0x 010, 0x 011, 0x 012, 0x 013**

Fifo Read i	Fifo Read i	Fifo Read i	Fifo Read i	Fifo Read i	Fifo Read i	Fifo Read i	Fifo Read i

The FifoReadiRegisters, along with the FifoWriteiRegisters, allow you to read or write the DAQ-DIO FIFOs directly. You cannot write a FIFO belonging to an input group or read a FIFO belonging to an output group. For the sake of simplicity, you should choose either the group FIFO registers or the direct FIFO registers, and not intermix the two types of accesses.

The advantages of group FIFO accesses are that the hardware cycles through the FIFOs for you, and you get the same result with programmed I/O that you would get from DMA. The advantage of the direct FIFO registers, besides using fewer addresses within the register map, is that you could potentially mix 8- and 16-bit accesses when handling a 24-bit group.

When using the direct FIFO registers, you must set the ReadyLevel to a value in the 0 to 4 range, and TransferWidth to 0 (32-bit). With this TransferWidth, the Exhausted and TransferReady bits indicate whether the whole group is exhausted or ready, respectively. For example, suppose that with a 32-bit output group and a 16-bit bus you read TransferReady high. You should write to AB and CD before checking TransferReady again, because TransferReady might fall after the write to AB.

FifoStatus **Read 0x 014**

Group 2 Fifo Exhausted	Group 1 Fifo Exhausted	Group 2 Fifo Ready	Group 1 Fifo Ready	Fifo Ready D	Fifo Ready C	Fifo Ready B	Fifo Ready A

Each FIFO ready flag indicates a FIFO is ready with the number of spaces or data specified by ReadyLevel for the group to which the FIFO belongs. The flag has no meaning when the FIFO belongs to both groups, or neither group.

The group ready flags indicate all FIFOs belonging to the group are ready, or, with a ReadyLevel of 6 or 7, that the FIFOs to be used for the next one to two funneling transfers are ready. See ReadyLevel for more information.

Fifo Ready A - 0
ReadyA

Fifo Ready B - 1
ReadyB

Fifo Ready C - 2
ReadyC

Fifo Ready D - 3
ReadyD

Group 1 Fifo Ready - 4
Ready: group 1

Group 2 Fifo Ready - 5
Ready: group 2

Group 1 Fifo Exhausted - 6
Indicates that the group's FIFOs are sufficiently empty (in input mode) or full (in output mode) that they cannot support another CPU or DMA transfer of width DMAWidth.

Group 2 Fifo Exhausted - 7

FifoWrite*i*Register **Write 0x 010, 0x011, 0x012, 0x013**

Fifo Write i	Fifo Write i	Fifo Write i	Fifo Write i	Fifo Write i	Fifo Write i	Fifo Write i	Fifo Write i

See the FifoReadiRegisters for more information.

GroupStatus **Read 0x 005**

Group 2 Serial	Group 2 Req	Group 2 Pattern Detected	Group 2 Data Left	Group 1 Serial	Group 1 Req	Group 1 Pattern Detected	Group 1 Data Left

Group 1 Data Left - 0
Indicates that one or more of the group's FIFOs contains data. In funneling and block modes, it is possible to have data left at the end of a transfer, but not enough data to do another full transfer. The programmer can use the DataLeft flag to help detect this condition.

Group 1 Pattern Detected - 1
Indicates that the programmed pattern has been detected. This bit can generate an interrupt.

Group 1 Req - 2
Current value of the group 1 request line (which can also be used as an extra input, if hardware handshaking is not in use), after any inversion and conditioning

Group 1 Serial - 3
Current value of the group 1 Serial line (which can also be used as an extra input, if serial and cascaded handshaking are not in use), after any inversion and conditioning

Group 2 Data Left - 4
Indicates that one or more of the group's FIFOs contains data. In funneling and block modes, it is possible to have data left at the end of a transfer, but not enough data to do another full transfer. The programmer can use the DataLeft flag to help detect this condition.

Group 2 Pattern Detected - 5
Indicates that the programmed pattern has been detected. This bit can generate an interrupt.

Group 2 Req - 6
Current value of the group 1 request line (which can also be used as an extra input, if hardware handshaking is not in use), after any inversion and conditioning

Group 2 Serial - 7
Current value of the group 1 Serial line (which can also be used as an extra input, if serial and cascaded handshaking are not in use), after any inversion and conditioning

GroupSynchronization **Write 0x 02C**

X	X	X	X	X	Master Run Mode	Master Run Mode	Master Run Mode

Master Run Mode - 0:2
These bits are ANDed with the group RunMode values, allowing you to enable or disabled both groups together. The power-up value is 7 (enabled). However, the group RunMode values power up to 0 (disabled), so that no handshaking takes place at power up. The group synchronization register allows you start and stop the two handshaking groups together.

0	Handshaking disabled for both groups 1 and 2
7	Handshaking determined by RunMode in GroupiOperatingMode

InterruptStatus **Read 0x 004**

X	Window Status	Window Status	Window Status	Window Status	Window Status	Group 2 Interrupting	Group 1 Interrupting

Group 1 Interrupting - 0
Group 1 is requesting interrupt service and, if enabled, the interrupt request pin is active.
Read group 1 flags to identify the cause of the interrupt.

Group 2 Interrupting - 1
Group 2 is requesting interrupt service and, if enabled, the interrupt request pin is active.
Read group 2 flags to identify the cause of the interrupt.

Window Status - 2:6
Bits 2 through 6 of the last address written to the window address register

MasterClockRouting **Write 0x 02D**

X	X	Rtsi Clocking	Rtsi Clocking	X	X	X	X

Rtsi Clocking - 4:5
Treatment of RTSIClock line, an alternate source for the master handshaking clock, which is used as the handshaking clock for any groups having ClockSource set to 0.
[Note: ClockSpeed, if non-zero, should be set after RTSIClocking, protected by RunMode=0 and Config = 1, etc.]

0	Standalone: Ignore the RTSIClock line, instead use the Oscillator line as master handshaking clock. Tristate the RTSIClock line.
2	Slave: Use the RTSIClock line as master handshaking clock.
3	Master: Use the Oscillator line as master handshaking clock and drive it onto the RTSIClock line.

MasterInterruptControl **Write 0x 005**

X	X	X	Restrict TC	Invert Interrupt	Open Collector Interrupt	Interrupt Line Control	Interrupt Line Control

Interrupt Line Control - 0:1

Value	Function
0	Tristate the interrupt line.
1	Drive the value of InvertInt onto the interrupt line.
3	Enable the interrupt line for normal operation, such as for a PCI bus.

Open Collector Interrupt - 2

Makes the interrupt line an open-collector-type output. For glitch-free configuration, Interrupt Line Control should be 0 when OpenInt is modified. Not used for PCI/PXI.

Invert Interrupt - 3

Makes the interrupt line an active-low signal. Not used for PCI/PXI.

Restrict TC - 4

If set, indicates that DMA terminal count (TC) should only be monitored during a read or write command, with DAck asserted. If cleared, indicates that DMA TC should be monitored any time DAck is asserted. This bit controls the assertion of the various TC flags and their associated interrupts, if enabled, as well as the switching between primary and secondary DMA channels. Not usually used for PCI/PXI.

PortDirection*i*Register **Write 0x 020, 0x021, 0x022, 0x023**

Port Direction i	Port Direction i	Port Direction i	Port Direction i	Port Direction i	Port Direction i	Port Direction i	Port Direction i

Port Direction *i* - 0:31

0	input
1	output

A pin specified for output can still be tristated if it belongs to an input group, or a group that employs tristating or open-collector outputs.

In unstrobed mode, each pin can be individually configured for either input or output.

In group handshaking, the pin directions should be set to match the group direction.

PortMask*i*Register **Write 0x 024, 0x025, 0x026, 0x027**

Port Mask i	Port Mask i	Port Mask i	Port Mask i	Port Mask i	Port Mask i	Port Mask i	Port Mask i

Port Mask *i* - 0:31

For input pins:

0	Significant to change detection
1	Ignored for change detection

For output pins:

0	Active driven output
1	Open collector driven output

Portpattern*i*Register **Write 0x 030, 0x031, 0x032, 0x033**

Port Pattern i	Port Pattern i	Port Pattern i	Port Pattern i	Port Pattern i	Port Pattern i	Port Pattern i	Port Pattern i

PortPatterniRegister contains the user's pattern in pattern detection mode.

PortPolarity*i*Register **Write 0x 028, 0x029, 0x02A, 0x02B**

Port Polarity i	Port Polarity i	Port Polarity i	Port Polarity i	Port Polarity i	Port Polarity i	Port Polarity i	Port Polarity i

Port Polarity i-		0:31
0	Active high	
1	Active low	

PortRead*i*Register **Read 0x 01C, 0x 01D, 0x 01E, 0x 01F**

Port Read i	Port Read i	Port Read i	Port Read i	Port Read i	Port Read i	Port Read i	Port Read i

Direct access registers for unstrobed read access to the the DIO pins not in use for group IO.

PortWrite*i*Register **Write 0x 01C, 0x01D, 0x01E, 0x01F**

Port Write i	Port Write i	Port Write i	Port Write i	Port Write i	Port Write i	Port Write i	Port Write i

If any port is not being used for group I/O, you can read from or write to the port pins directly. Such accesses are called mode 0, or unstrobed accesses, and are modified only by the pin-direction and mask registers.

When you read a port containing output pins, the ASIC reports the current value driven on those pins. In other words, you have "read-back" capability. When you write a port containing input pins, the written value is latched in an output register. However, the value does not appear on the data lines while the pins are directed for input or the port is being used for handshaking I/O.

When a port is being used for group I/O in an output group, the value in the port output register serves as the port's reset value. In other words, resetting the group (see the Operating Mode register) resets the output ports to their mode 0 values.

Group Registers

Group1AckDelayRegister **Write 0x 054**

Group 1 Ack Delay	Group 1 Ack Delay	Group 1 Ack Delay	Group 1 Ack Delay	Group 1 Ack Delay	Group 1 Ack Delay	Group 1 Ack Delay	Group 1 Ack Delay

Group2AckDelayRegister **Write 0x 074**

Group 2 Ack Delay	Group 2 Ack Delay	Group 2 Ack Delay	Group 2 Ack Delay	Group 2 Ack Delay	Group 2 Ack Delay	Group 2 Ack Delay	Group 2 Ack Delay

Delay from sequence start to assertion of Ack, indicating to the peripheral that the ASIC is ready for another transfer.

Group1AckNotDelayRegister **Write 0x 055**

Group 1 Ack Not Delay	Group 1 Ack Not Delay	Group 1 Ack Not Delay	Group 1 Ack Not Delay	Group 1 Ack Not Delay	Group 1 Ack Not Delay	Group 1 Ack Not Delay	Group 1 Ack Not Delay

Group2AckNotDelayRegister **Write 0x 075**

Group 2 Ack Not Delay	Group 2 Ack Not Delay	Group 2 Ack Not Delay	Group 2 Ack Not Delay	Group 2 Ack Not Delay	Group 2 Ack Not Delay	Group 2 Ack Not Delay	Group 2 Ack Not Delay

Delay from sequence start, Req, or Req* to deassertion of Ack, depending on the value of AckNotAfter in the GroupiHandshakeSequence register.

Group1ClearRegisterA **Write 0x 006**

Group 1 Reset Fifo	Group 1 Reset Funneling	Group 1 Clear Secondary TC	Group 1 Clear Primary TC	Group 1 Clear Waited	Group 1 Clear Paused	Group 1 Clear Req Rose	Group 1 Clear Serial Rose

Group2ClearRegisterA **Write 0x 007**

Group 2 Reset Fifo	Group 2 Reset Funneling	Group 2 Clear Secondary TC	Group 2 Clear Primary TC	Group 2 Clear Waited	Group 2 Clear Paused	Group 2 Clear Req Rose	Group 2 Clear Serial Rose

Clear Serial Rose - 0
Clear the SerialRose flag
Clear Req Rose - 1
Clear the ReqRose flag
Clear Paused - 2

Clear the Paused flag
Clear Waited - 3
Clear the Waited flag
Clear Primary TC - 4
Clear the PrimaryTC flag
Clear Secondary TC - 5
Clear the SecondaryTC flag
Reset Funneling - 6
Reset the DMA and CPU-side funneling processes
Reset Fifo - 7
Empty and reset the group's FIFOs

Group1ClearRegisterB **Write 0x 02E**

X	X	X	X	X	X	Group 1 Clear Match Detected	Group 1 Clear Expired

Group2ClearRegisterB **Write 0x 02F**

X	X	X	X	X	X	Group 2 Clear Match Detected	Group 2 Clear Expired

Clear Expired - 0
Clear the Expired flag (and the transfer count, if non-zero)

Clear Match Detected - 1
Clear the MatchDetected flag

Group1ClockSpeedARegister **Write 0x 044**

Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed

Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed	Group 1 Clock Speed

Group2ClockSpeedARegister **Write 0x 064**

Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed

Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed	Group 2 Clock Speed

If the Clock Speed register is not zero, the DAQ-DIO divides the incoming clock from the Oscillator, RTSIClock, or PClock pin by a factor equal to the Clock Speed times two. Clock Speed should not be set until after the Clock Source is written to GroupiStartupAndClocking, and RtsiClocking is written to the MasterClockRouting register.

Using the internal clock source:

0	PCLK = 20 MHz
>0	PCLK = 20 MHz/ (2 * GroupiClockSpeedRegister)

Group1ConditioningDelayRegister **Write 0x 04E**

Group 1 Conditioning Delay	Group 1 Conditioning Delay	Group 1 Conditioning Delay	Group 1 Conditioning Delay	Group 1 Conditioning Delay	Group 1 Conditioning Delay	Group 1 Conditioning Delay	Group 1 Conditioning Delay

Group2ConditioningDelayRegister **Write 0x 06E**

Group 2 Conditioning Delay	Group 2 Conditioning Delay	Group 2 Conditioning Delay	Group 2 Conditioning Delay	Group 2 Conditioning Delay	Group 2 Conditioning Delay	Group 2 Conditioning Delay	Group 2 Conditioning Delay

Group1CounterControl **Write 0x 050**

X	X	X	X	X	X	X	Group 1 Latch Count

Group2CounterControl **Write 0x 070**

X	X	X	X	X	X	X	Group 2 Latch Count

Latch Count - 0

Holds a copy of the transfer count steady so that you can read its value realiably. The actual counter, and therefore the ongoing data-transfer operation, are unaffected. A LatchCount of 1 takes effect on the next deasserting clock edge, and it may be difficult, in the case of slow clocks, to predict when the next such edge will occur. Therefore, it is recommended that you follow the procedure below to read the counter:

1. Write a 0 to LatchCount (if not already low).
2. Write a 1 to LatchCount.
3. Read the Transfer Count Latch twice.
4. If the two readings agree, you are assured the value is correct.
5. Otherwise, read a third time. The third reading gives the correct value.

Group1DaqOptions **Write 0x 04F**

Group 1 Prestart	Group 1 Convert Source	Group 1 Pulse Burst	Group 1 Stop Source	Group 1 Stop Source	Group 1 Invert Start	Group 1 Start Source	Group 1 Start Source

Group2DaqOptions **Write 0x 06F**

Group 2 Prestart	Group 2 Convert Source	Group 2 Pulse Burst	Group 2 Stop Source	Group 2 Stop Source	Group 2 Invert Start	Group 2 Start Source	Group 2 Start Source

To select data-acquisition or pattern-generation mode with internal convert/update pulses in non-burst mode, configure a protocol that produces pulses and choose ReqSource = constant, InvertReq = 1, and Convert Source = internal. In burst mode, instead set Pulse Burst = 1 and Convert Source = internal.

To select data acquisition or pattern-generation mode with external convert/update pulses in non-burst mode, configure a protocol that accepts pulses and choose ReqSource = external and, if you plan to use a hardware start trigger, AckLine = tristated. In burst mode, also set PulseBurst = 1.

If you want to use the Ack line for all data acquisition convert pulses and pattern generation update pulses, both external and internal, then set the ExchangePins bit when using external convert pulses.

The DAQ Options register contains the PulseBurst and Convert Source fields, as well as fields that specify the
start and stop triggers for data acquisition or pattern generation. The start trigger begins acquisition. The stop trigger begins the count. The TransferCount register specifies the number of post-trigger data to acquire after the stop trigger.

Start Source - 0:1

Source of the start trigger. Until the start trigger is asserted, the DAQ-DIO transfers no data. With Prestart set, the DAQ-DIO does acquire data prior to the start trigger for purposes of pattern detection, but then discards the data.

0	Software start trigger. Trigger is high unless InvertStart is set, in which case trigger is low. (Note that this is inverted from the operation of software request.)
1	Hardware edge start trigger. An active-going edge on the selected line asserts start. The trigger is the Req or Ack line, whichever is unused for convert pulses. In other words, if ExchangePins in GroupiSerialAndAckControl = ConvertPulse, Req line is the trigger. Otherwise the Ack line is the trigger.
2	Hardware level start trigger. Similar to the hardware edge trigger, except an active level, rather than an active-going edge, asserts start.
3	Pattern match (input groups only). When using this setting with DetectionMethod = 1, you must also set the Prestart bit.

Invert Start - 2

Inverts the polarity of the start signal. Unlike most configuration bits, this bit can be changed at any time, even if RunMode is 7.

Stop Source - 3:4

Source of the stop trigger

0	Software start trigger. Trigger is high unless Invert Start is set, in which case trigger is low. (Note that this is inverted from the operation of software request.)
1	Hardware start edge trigger. The trigger is the Serial line. An active-going edge asserts start.

2	Hardware start level trigger. The trigger is the Serial line. An active level, rather than an active-going edge, asserts start.
3	Pattern match (input groups only). When using this setting with DetectionMethod = 1, you must also set the Prestart bit.

Pulse Burst - 5

Permits burst mode to operate asynchronously, providing or responding to asynchronous strobe (convert) pulses instead of handshaking signals synchronous to a free- running clock. If Convert Source is external, causes the DAQ-DIO to switch from its normal clock source, as identified in the GroupiStartupAndClocking register, to the incoming convert pulses on the Req line (or, with Exchange Pins in GroupiSerialAndAckControl, the Ack line) when the transfer starts. The internal Req signal is constant high. Invert Req in GroupiLinePolarities determines the polarity of the incoming convert signals on the Req or Ack line.

If Convert Source is internal, the DAQ-DIO to generates Ack pulses by ANDing its clock and Ack signals.

The combination of Pulse Burst with Convert Source = internal requires careful configuration order.

Convert Source - 6

Source of strobe (convert) pulses

0	Internal
1	External

This field selects the source of the Start signal when StartSource = 1.

0	Causes the Start signal to be taken from the Ack pin, unless ExchangePins is set. AckLine should therefore be set to tristate.
1	Causes the Start signal to be taken from the Req pin, unless ExchangePins is set. ReqSource should therefore be a source other than the Req pin (normally Req=1).

If BasicSequence = burst and PulseBurst is set, then ConvSource has other effects as well.

0	Forces the internal convert signal (Ack) to become a pulse by ANDing in the handshaking clock
1	Causes the external convert signal (Req) to be taken as a pulse and used as a clock signal.

Prestart - 7

Enables handshaking to occur prior to the Start signal, assuming that the FIFOs meet the programmed BlockSize level. However, no data is acquired into or removed from the FIFOs until Start occurs. Therefore, the BlockSize level, once met, will continue to be met until Start occurs. Should only be used with GroupiDirection = input in the GroupiDataPath register.

Must be set for any mode for StartSource = pattern detection with DetectionMethod = 1.

Group1Data1DelayRegister **Write 0x 056**

Group 1 Data 1 Delay	Group 1 Data 1 Delay	Group 1 Data 1 Delay	Group 1 Data 1 Delay	Group 1 Data 1 Delay	Group 1 Data 1 Delay	Group 1 Data 1 Delay	Group 1 Data 1 Delay

Group2Data1DelayRegister **Write 0x 076**

Group 2 Data 1 Delay	Group 2 Data 1 Delay	Group 2 Data 1 Delay	Group 2 Data 1 Delay	Group 2 Data 1 Delay	Group 2 Data 1 Delay	Group 2 Data 1 Delay	Group 2 Data 1 Delay

Delay from sequence start, Req, or Req* to sampling in standard input mode, or to driving the output lines in tristate output mode. Sequence start, Req, or Req* are chosen by Data 1 After in the GroupiHandshakeSequence register.

Group1Data2DelayRegister **Write 0x 057**

Group 1 Data 2 Delay	Group 1 Data 2 Delay	Group 1 Data 2 Delay	Group 1 Data 2 Delay	Group 1 Data 2 Delay	Group 1 Data 2 Delay	Group 1 Data 2 Delay	Group 1 Data 2 Delay

Group2Data2DelayRegister **Write 0x 077**

Group 2 Data 2 Delay	Group 2 Data 2 Delay	Group 2 Data 2 Delay	Group 2 Data 2 Delay	Group 2 Data 2 Delay	Group 2 Data 2 Delay	Group 2 Data 2 Delay	Group 2 Data 2 Delay

Delay from sequence start, Req, or Req* to the tristating the output lines in tristate output mode. Sequence start, Req, or Req* are chosen by Data 2 After in the GroupiHandshakeSequence register.

Group1DataPath **Write 0x 040**

Group 1 Direction	Group 1 Funneling Order	Group 1 Funneling	Group 1 Funneling	Group 1 Enable Fifo D	Group 1 Enable Fifo C	Group 1 Enable Fifo B	Group 1 Enable Fifo A

Group2DataPath **Write 0x 060**

Group 2 Direction	Group 2 Funneling Order	Group 2 Funneling	Group 2 Funneling	Group 2 Enable Fifo D	Group 2 Enable Fifo C	Group 2 Enable Fifo B	Group 2 Enable Fifo A

Hold the Configuration bit low while configuring the Data Path

Enable Fifo A - 0
    FIFO A belongs to the group.

Enable Fifo B - 1
    FIFO B belongs to the group.

Enable Fifo C - 2
    FIFO C belongs to the group.

Enable Fifo D - 3
    FIFO D belongs to the group.

Funneling - 4:5
    Routing of port data to FIFOs. During serial operation, a group should be set to 8-bitfunneling, but funneling occurs to or from the group's serial I/O register, instead of port A or C. During handshaking, pattern, or burst mode, funneling can be set to 8 bit, 16 bit, or to No Funnelling.

    0 No funneling. Normal operation.

2	8-bit funneling. Only port A (group 1) or C (group 2) is used for I/O. However, transfers take place between the designated ports and each of the group's enabled FIFOs, in succession. If the group contains more than one FIFO (8-to-16, 8-to-24, or 8-to-32 funneling), the Order bit selects which transfer occurs first.
3	16-bit funneling. Ports A and B (group 1) or C and D (group 2) transfer data to and from all the FIFOs in the group, starting from the bottom two FIFOs and moving up if Funnelling Order = 0, but starting from the top two and moving down if Funnelling Order = 1. A group used for 16-bit funneling must contain an even number of FIFOs.

Funneling Order - 6

Byte order, when funneling between unequal path widths.

0	Increment, starting with the lowest port (port A, if used).
1	Decrement, starting with the highest port (port D, if used).

Direction - 7

0	Input
1	Output

Group1DmaLineControl **Write 0x 04C**

X	X	X	X	Group 1 Second Dma Channel	Group 1 Second Dma Channel	Group 1 First Dma Channel	Group 1 First Dma Channel

Group2DmaLineControl **Write 0x 06C**

X	X	X	X	Group 2 Second Dma Channel	Group 2 Second Dma Channel	Group 2 First Dma Channel	Group 2 First Dma Channel

DMA begins on the primary channel and switches between the primary and secondary channels each time the ASIC receives terminal count (TC). If you wish to use only one channel, program First DMA Channel and Second DMA Channel to the same value.

First Dma Channel - 0:1

Primary DMA channel (1 to 2), or 0 for none

Second Dma Channel - 2:3

Secondary DMA channel (1 to 2), or 0 for none

Group1FifoControl **Write 0x 048**

X	X	X	Group 1 Ext Fifo Enable	Group 1 Ext Fifo Direction	Group 1 Fifo Ready Level	Group 1 Fifo Ready Level	Group 1 Fifo Ready Level

Group2FifoControl **Write 0x 068**

X	X	X	Group 2 Ext Fifo Enable	Group 2 Ext Fifo Direction	Group 2 Fifo Ready Level	Group 2 Fifo Ready Level	Group 2 Fifo Ready Level

Fifo Ready Level - 0:2

In handshaking mode, with FIFOs enabled, this field controls the number of transfers for which the ASIC should be prepared before asserting Groupi Fifo Ready in the FifoStatus register. Groupi Fifo Ready is the flag that sets Transfer Ready in the GroupiFlags register and requests bus access from the CPU or DMA controller.

The ASIC asserts Groupi Fifo Ready when prepared for 2 to the power of ReadyLevel transfers, as follows:

Value	FIFO Level for Groupi Fifo Ready
0	1 sample in each FIFO in the group
1	2 samples in each FIFO in the group
2	4 samples in each FIFO in the group
3	8 samples in each FIFO in the group
4	Full FIFO depth.
5	FIFO depth minus one (output) or full FIFO depth (input).
6	1 (target FIFO for the next transfer, with DMAWidth of 8)
7	1 (target FIFOs for the next transfer, with DMAWidth of 16, or target FIFOs for the next two transfers, with DMAWidth of 8 and at least two FIFOs in the group)

The default Ready Level is 0, meaning the ASIC asserts Groupi Fifo Ready when prepared for even one transfer. Raising the level tends to increase bus throughput when using bursts, or multiple transfers for a single bus grant. However, raising the level also reduces the maximum speed of timer-based data acquisition or pattern generation.

In input mode, the ASIC asserts Groupi Fifo Ready when the FIFO depth meets or exceeds the specified level. In output mode, the ASIC asserts Groupi Fifo Ready when the number of empty FIFO slots meets or exceeds the specified level.

You can safely change Ready Level without pausing handshaking. However, you should not change Ready Level while DMA is being performed on the group.

You can use Ready Level and BlockSize fields to disable or reduce the size of the internal FIFOs. In input mode, you can disable the FIFOs by setting Ready Level to 1 and BlockSize to 4. In output mode, you can reduce the effective FIFO depth to two by setting BlockSize to 1 and Ready Level to 5. Disabling the FIFOs (for input) or reducing their depth to two (for output) is appropriate to a control loop, in which software must react to each transfer immediately, without queuing data in the FIFO.

Ext Fifo Direction - 3

External FIFO interface direction. Should be set to the appropriate value during initial configuration if the board makes use of the ASIC's external FIFO interface. If you change the value later, after assigning any FIFOs to groups, the Config bit must be asserted for those groups.

0	Accept data from external FIFOs
1	Drive data to external FIFOs

Ext Fifo Enable - 4

Enable external FIFO interface. Should be set during initial configuration if the board makes use of the ASIC's external FIFO interface. If you change the value later, after assigning any FIFOs to groups, the Config bit must be asserted for those groups. Alters the normal flow of data between the CPU/bus and peripheral interfaces. See "External FIFO Interface" for details.

Group1FifoRead*i*Register **Read 0x 008, 0x009, 0x00A, 0x00B**

Group 1 Fifo Read i	Group 1 Fifo Read i	Group 1 Fifo Read i	Group 1 Fifo Read i	Group 1 Fifo Read i	Group 1 Fifo Read i	Group 1 Fifo Read i	Group 1 Fifo Read i

Group2FifoRead*i*Register **Read 0x 00C, 0x 00D, 0x 00E, 0x 00F**

Group 2 Fifo Read i	Group 2 Fifo Read i	Group 2 Fifo Read i	Group 2 Fifo Read i	Group 2 Fifo Read i	Group 2 Fifo Read i	Group 2 Fifo Read i	Group 2 Fifo Read i

Group1FifoWrite*i*Register **Write 0x 008, 0x009, 0x00A, 0x00B**

Group 1 Fifo Write i	Group 1 Fifo Write i	Group 1 Fifo Write i	Group 1 Fifo Write i	Group 1 Fifo Write i	Group 1 Fifo Write i	Group 1 Fifo Write i	Group 1 Fifo Write i

Group2FifoWrite*i*Register **Write 0x 00C, 0x00D, 0x00E, 0x00F**

Group 2 Fifo Write i	Group 2 Fifo Write i	Group 2 Fifo Write i	Group 2 Fifo Write i	Group 2 Fifo Write i	Group 2 Fifo Write i	Group 2 Fifo Write i	Group 2 Fifo Write i

Reading or writing the group FIFO registers is equivalent to performing DMA upon the group. All accesses should of the same width, equal to the transfer width established by Groupi Transfer Size in the Transfer Size register, and match the group direction. The transfer width should be less than or equal to the group width, determined by the number of FIFOs assigned to the group. If the transfer width is less than the width of the group, each read or write will access a different set of FIFOs.

For example, suppose you program a transfer width of one byte after setting EnableA, EnableB, and EnableD. Suppose you set GroupDirection = 0 (input) and TransferOrder = 1 (descending). Now you can perform a series of 8-bit reads from the first group FIFO register. Your first read will be routed from FIFO D, your next from FIFO B, your next from FIFO A, and then the cycle would repeat. The resulting pattern is the same as if you were using 8-bit DMA.

Group1Flags **Read 0x 006**

Group 1 Secondary TC	Group 1 Primary TC	Group 1 Waited	Group 1 Paused	Group 1 Serial Rose	Group 1 Req Rose	Group 1 Count Expired	Group 1 Transfer Ready

Group2Flags **Read 0x 007**

Group 2 Secondary TC	Group 2 Primary TC	Group 2 Waited	Group 2 Paused	Group 2 Serial Rose	Group 2 Req Rose	Group 2 Count Expired	Group 2 Transfer Ready

Transfer Ready - 0

Indicates that group 1 has data ready and, if DMA is enabled, is requesting DMA service. Operation of this bit is controlled by the group 1 DMASize register, whether or not DMA is enabled. For the Transfer Ready flag to be meaningful, you should set Transfer Width in the GroupiTransferSize register to the full group size and read or write all the group's FIFOs.

Count Expired - 1

In Numbered mode, indicates that the Transfer Count has expired (or has not begun). If the ASIC is not in Numbered mode, Count Expired is 1.

Req Rose - 2

Rising edge of the internal Request signal detected. Typically, Req Rose may also imply that the Req pin rose; however, the implications depend on the Req Conditioning and DAQ Option settings in effect. Clear this flag with the clear register.

Serial Rose - 3

Rising edge of the Serial (extra input) line detected. Clear this flag with the clear register.

Paused - 4

EndOfCycle (Paused). Indicates that a complete handshaking cycle has finished since the last clearing of this flag. If you performed the last such clear after setting Run Mode bit 0, then assertion of EndOfCycle indicates that handshaking has paused.

Waited - 5

Indicates that the handshaking sequence had to stop and wait for ASIC ready. This might be a cause for concern in data acquisition or pattern generation, if you cannot tolerate added delays in a precisely timed sequence. Use the clear register to clear this flag.

Primary TC - 6

DMA terminal count has occurred on the group's primary DMA channel.

Secondary TC - 7

DMA terminal count has occurred on the group's secondary DMA channel.

Group1FrameControl **Write 0x 047**

X	Group 1 Frame Size	Group 1 Frame Size	Group 1 Frame Size	Group 1 Frame Size	Group 1 Frame Size	Group 1 Frame Tristate	Group 1 Frame Tristate

Group2FrameControl **Write 0x 067**

X	Group 2 Frame Size	Group 2 Frame Size	Group 2 Frame Size	Group 2 Frame Size	Group 2 Frame Size	Group 2 Frame Tristate	Group 2 Frame Tristate

Frame Tristate - 0:1

Frame and tristate mode control. Tristating applies only if the group direction is output.

0	Normal. DataDelay2 is ignored.
1	Frame mode. Transfer data from DataDelay1 to DataDelay2.
2	Tristate mode. Drive the output lines at DataDelay2. Tristate the outputs and advance to new data, if available, at DataDelay1.

Frame Size - 2:6

Indicates in exponential form the number of transfers the ASIC's FIFOs should be prepared for before the ASIC raises the DataReady status flag, begins the handshaking sequence, and asserts

Ack. Peripheral-side equivalent of ReadyLevel. This register does not control the number of transfers done in a block, which is determined by the BlockTristate, Data1After,
Data1Delay, Data2After, and Data2Delay registers. Usually, however, the BlockSize should equal DataDelay2 minus DataDelay1, rounded up to a power of two. However, when using data funneling, divide by the multiplexing factor to get a good block size (again, round up)

For all modes:

0	1 sample for each FIFO in the group
1	2 samples for each FIFO in the group
2	4 samples for each FIFO in the group
3	8 samples for each FIFO in the group
4	Full FIFO depth.

For control-loop applications only:

5	FIFO depth minus one (input) or full FIFO depth (output).

For funneling only:

6	1 (1 FIFO in a group of 2 or more)
7	1 (2 FIFOs out of 4)

Group1HandshakeSequence **Write 0x 043**

Group 1 Data 2 After	Group 1 Data 2 After	Group 1 Data 1 After	Group 1 Data 1 After	Group 1 Ack Not After	Group 1 Ack Not After	Group 1 Basic Sequence	Group 1 Basic Sequence

Group2HandshakeSequence **Write 0x 063**

Group 2 Data 2 After	Group 2 Data 2 After	Group 2 Data 1 After	Group 2 Data 1 After	Group 2 Ack Not After	Group 2 Ack Not After	Group 2 Basic Sequence	Group 2 Basic Sequence

Basic Sequence - 0:1

Basic handshaking sequence, starting when the ASIC is ready for a transfer.

0	Burst mode. Counter L holds the time since Ready and Req.
1	Ready -> Req. (Use this sequence if only one edge of Req affects the timing of other handshaking events.)
2	Ready -> Req -> Req*.
3	Ready -> Req* -> Req.

Ack Not After - 2:3

Event after which the ASIC should deassert Ack. Does not apply to BasicSequence 0, burst mode.
Note: There is no AckAfter register. If sequenced handshaking is used, Ack always occurs after ASIC ready for transfer.

0	ASIC ready for transfer
1	Req
2	Req*

Data 1 After - 4:5

Event after which the ASIC should sample data (input mode). In block mode (input or output), a sequence of transfers begins at this time. In tristate mode, the output lines exit high-impedance state at this time.

0	ASIC ready for transfer
1	Req
2	Req*
3	None. Data1Delay must be 0. Use this value to handshake without acquiring data.

Data 2 After - 6:7

Event after which the ASIC should quit sampling or driving data, in block or tristate mode. Does not apply to sequence 0, burst mode.

0	ASIC ready for transfer
1	Req
2	Req*

Group1InterruptControl **Write 0x 04B**

Group 1 Enable Secondary T C	Group 1 Enable Primary T C	Group 1 Enable Waited	Group 1 Enable Paused	Group 1 Enable Serial Rose	Group 1 Enable Req Rose	Group 1 Enable Count Expired	Group 1 Enable Transfer Ready

Group2InterruptControl **Write 0x 06B**

Group 2 Enable Secondary TC	Group 2 Enable Primary TC	Group 2 Enable Waited	Group 2 Enable Paused	Group 2 Enable Serial Rose	Group 2 Enable Req Rose	Group 2 Enable Count Expired	Group 2 Enable Transfer Ready

See also the GroupiPatternDetection register, which has an additional interrupt enable, Groupi Enable Match Detected.

Enable Transfer Ready - 0
Interrupt on TransferReady

Enable Count Expired - 1
Interrupt on CountExpired

Enable Req Rose - 2
Interrupt on ReqRose

Enable Serial Rose - 3
Interrupt on SerialRose

Enable Paused - 4
Interrupt on EndOfCycle (Paused)

Enable Waited - 5
Interrupt on Waited

Enable Primary TC - 6
Interrupt on PrimaryTC

Enable Secondary TC - 7
Interrupt on SecondaryTC

Group1LinePolarities **Write 0x 049**

X	X	Group 1 Open Clock	Group 1 Open Ack	Group 1 Invert Serial	Group 1 Invert Clock	Group 1 Invert Req	Group 1 Invert Ack

Group2LinePolarities **Write 0x 069**

X	X	Group 2 Open Clock	Group 2 Open Ack	Group 2 Invert Serial	Group 2 Invert Clock	Group 2 Invert Req	Group 2 Invert Ack

Handshaking line polarities. To invert port data polarities, use the port polarity registers.

Invert Ack - 0
Inverts the polarity of the Ack signal (could be on the Req pin if ExchangePins set)

Invert Req - 1
Inverts the polarity of the Req signal (could be on the Ack pin if ReqSource = Ack pin)

Invert Clock - 2
Inverts the polarity of the Clock signal. If the line is used to carry the handshaking clock, causes the ASIC to drive data on the falling clock edge and sample on the rising edge.

Invert Serial - 3
Inverts the polarity of the Serial signal

Open Ack - 4
Makes the Ack line, if configured as an output, an open-collector type signal. To assert Ack (or, with InvertAck = 1, to deassert Ack), the ASIC drives the Ack line to a low voltage. Otherwise, the ASIC tristates the Ack line. For glitch-free configuration, AckLine should be 0 whenever OpenAck is modified.

Open Clock - 5
Makes the PClock line, if configured as an output, an open-collector type signal. This option is only useful when using the PClock line as an extra output, rather than a clock signal. To assert PClock (or, with InvertClock = 1, to deassert PClock), the ASIC drives the PClock line to a low voltage. Otherwise, the ASIC tristates the PClock line. For glitch-free configuration, ClockLine should be 0 whenever OpenClock is modified.

Group1OperatingMode **Write 0x 041**

Group 1 Configurat ion	Group 1 Data Latching	Group 1 Data Latching	Group 1 Burst Master	Group 1 Numbered	Group 1 Run Mode	Group 1 Run Mode	Group 1 Run Mode

Group2OperatingMode **Write 0x 061**

Group 2 Configurat ion	Group 2 Data Latching	Group 2 Data Latching	Group 2 Burst Master	Group 2 Numbered	Group 2 Run Mode	Group 2 Run Mode	Group 2 Run Mode

Run Mode - 0:2

Enables for the handshaking process. Run Mode starts, pauses, and resets the transfer.

0	Stops the current transfer, resetting the group handshaking logic, conditioning, serial conditioning, and conditioning delay counters
6	Pauses the current transfer at the end of the current cycle through the handshaking sequence, setting the Paused status flag
7	Starts or restarts the transfer

Numbered - 3

Causes the group to handshake for only for as many transfers as are specified by the Transfer Count register

Burst Master - 4

Designates the group as master for burst-mode operations

Data Latching - 5:6

Controls the precise signal edge that causes data to be latched into or out of the ASIC

For input groups, data is sampled on the rising clock edge specified by the DataAfter, DataDelay, and BlockTristate registers. However, data can be latched ahead of time as follows:

0	Sample data on the falling clock edge.
1	Sample data on the rising clock edge.
2	Sample data on the active-going Req edge.
3	Sample data on the inactive-going Req edge.

For output groups, new data is normally driven on the first rising clock of the cycle. In block protocols, new data is also driven on every rising clock edge from the data 1 control event to the data 2 control event, as specified by the DataAfter and DataDelay registers.

0	Drive data on the rising clock edge.
1	Reserved.
2	Delay new data until the next active-going Req edge.
3	Delay new data until the next inactive-going Req edge.

Configuration - 7

Inhibits the group from writing or resetting its FIFOs and ports, so that you can safely configure the DataPath register, as well as the ExtDirection and ExtEnable fields. Lower the Configuration bit before raising RunMode bit 2, using a separate write operation, and before writing to the group.

Group1PatternDetection **Write 0x 051**

X	X	X	X	X	Group 1 Enable Match Detected	Group 1 Invert Match	Group 1 Detection Method

Group2PatternDetection **Write 0x 071**

X	X	X	X	X	Group 2 Enable Match Detected	Group 2 Invert Match	Group 2 Detection Method

Detection Method - 0

0 Compare the input pins directly to the pattern at all times

1 Compare acquired data to the pattern. A pattern is ignored unless it occurs at the sample point specified by the handshaking registers and DAQ Option registers. This mode inserts a latch into the input data path before the FIFOs.

Ignored for output. Also ignored for burst mode; only DetectionMethod 1 can be used. Must be set to 1 if ReqSource is change detection; therefore, change detection is compatible only with DetectionMethod 1.

Invert Match - 1

If set, match is asserted when the unmasked data lines do not match the input pattern

Enable Match Detected - 2

Interrupt on match-detected flag

Group1ReqDelayRegister **Write 0x 052**

Group 1 Req Delay	Group 1 Req Delay	Group 1 Req Delay	Group 1 Req Delay	Group 1 Req Delay	Group 1 Req Delay	Group 1 Req Delay	Group 1 Req Delay

Group2ReqDelayRegister **Write 0x 072**

Group 2 Req Delay	Group 2 Req Delay	Group 2 Req Delay	Group 2 Req Delay	Group 2 Req Delay	Group 2 Req Delay	Group 2 Req Delay	Group 2 Req Delay

Delay from sequence start or Req* until the state machine searches for active Req . If a request is pending, the Req counter resets and begins counting. Otherwise, the ASIC waits for a request.

Group1ReqManagement **Write 0x 046**

Group 1 Req Edge	Group 1 Req Edge	Group 1 Req Condition ing	Group 1 Req Condition ing	Group 1 Req Condition ing	Group 1 Req Source	Group 1 Req Source	Group 1 Req Source

Group2ReqManagement **Write 0x 066**

Group 2 Req Edge	Group 2 Req Edge	Group 2 Req Condition ing	Group 2 Req Condition ing	Group 2 Req Condition ing	Group 2 Req Source	Group 2 Req Source	Group 2 Req Source

The Req line is also configured by Exchange Pins in GroupiSerialAndAckControl, DataLatching in GroupiOperatingMode, and InvertReq in GroupiLinePolarities.

Req Source - 0:2

0 Request line.

3 Change detection. Request is asserted when the current data on the unmasked input lines do not match the last-acquired data. DetectionMethod must be set to 1. Not applicable to block or burst modes.

Be sure to set the sequence delay registers to allow at least one cycle, plus any required conditioning time, between sampling data and checking Req. Recommended values: Seq = 1, Data1After = 1, ReqConditioning = 0 (synchronize), StartDelay = 1, ReqDelay = 2, ClockSpeed = 50ns. With these values, change detection will work reliabily for remain that remain on the input lines at least 175ns, being three cycles plus a 25ns margin to account for any difference between minimum and maximum input data paths.

4 The Req signal is constant, equal to the value of InvertReq. (You may change the value of InvertReq without pausing the group.)

Req Conditioning - 3:5

(In a synchronous protocol, you can use the unsynchronized, undebounced Req signal to generate control signals by specifying sequence delays of zero.)

0	Synchronize the Req line.
1	Do not condition the Req line at all. Use Req without synchronization.
2	Debounce the Req line using the clock. Req transitions are considered glitches unless the new value is held for three consecutive clock cycles.
3	Debounce the Req line using ConditioningDelay. Req transitions are considered glitches unless the new value remains constant over three samplings, with sampling occurring every DataDelay clock cycles. Applies only to sequences 0 and 1.
5	Freeze Req for a delay of ConditioningDelay after any Req transition, to allow settling time after the transition. Invalid if SerialConditioning specifies debounce.
6	Req is considered present if sampled high at least ConditioningDelay after ASIC ready. Invalid if SerialConditioning specifies debounce.
7	Req is considered present if sampled high at least ConditioningDelay after recognition of the previous Req (sequence 1 only). Invalid if SerialConditioning specifies debounce.

Req Edge - 6:7

0	Level. Request is present if Req is sampled high at the time specified by Req Delay. ReqBar is present if Req is sampled low at the time specified by Req* Delay.
1	Edge. Request is present if the Req line has shown an active-going edge prior to the time specified by Req Delay. Req* is present if the Req line has shown a negative-going edge prior to the time specified by Req* Delay. Both are synchronized to the group clock.

Group1ReqNotDelayRegister **Write 0x 053**

Group 1 Req Not Delay	Group 1 Req Not Delay	Group 1 Req Not Delay	Group 1 Req Not Delay	Group 1 Req Not Delay	Group 1 Req Not Delay	Group 1 Req Not Delay	Group 1 Req Not Delay

Group2ReqNotDelayRegister **Write 0x 073**

Group 2 Req Not Delay	Group 2 Req Not Delay	Group 2 Req Not Delay	Group 2 Req Not Delay	Group 2 Req Not Delay	Group 2 Req Not Delay	Group 2 Req Not Delay	Group 2 Req Not Delay

Delay from sequence start or Req until the state machine searches for inactive Req, in sequence 2 or 3. If Req is already inactive, the Req* counter resets and begins counting. Otherwise, the ASIC waits for Req*.

Group1SerialAndAckControl **Write 0x 04A**

Group 1 Exchange Pins	Group 1 Ack Line	Group 1 Ack Line	Group 1 Bit Order	Group 1 Serial Conditioning	Group 1 Serial Conditioning	Group 1 Serial Line	Group 1 Serial Line

Group2SerialAndAckControl **Write 0x 06A**

Group 2 Exchange Pins	Group 2 Ack Line	Group 2 Ack Line	Group 2 Bit Order	Group 2 Serial Conditioning	Group 2 Serial Conditioning	Group 2 Serial Line	Group 2 Serial Line

The Serial line, also called the Stop Trigger, can be used as an extra input, and therefore has some of the same conditioning options available for the Req line; can carry serial data; can participate in cascading; or can carry a trigger signal to a timing chip, such as an STC.

Serial Line - 0:1

Serial data line control and serial mode enable. In serial mode, the group's normal data lines are not used. Instead, the group's Serial data line is used. An 8-bit shift register holds data during I/O. The data is funneled to the group's funnel registers, according to which Select bits are set, as in 8-to-anything funneling. (Most serial ports employ a block mode, in which handshaking takes place once per byte, word, or double word, rather than once per bit.)

0	Tristate the serial data line and use it as an extra input.
1	Drive the trigger signal onto the group's serial line.
3	Enable serial mode. Use the serial data line for data transfer. [Serial protocols are synchronous, and the serial data line is permitted to glitch.]

Serial Conditioning - 2:3

Synchronization to apply to the Serial line, if an input. Values other than 2 are mainly useful when the Serial Line is being used as an extra input.

0	No conditioning
1	Latch on the negative clock edge
2	Synchronize to the positive clock edge
3	Latch on the negative clock edge and then synchronize to the positive clock edge

Bit Order - 4

Bit order, for serial mode transfers.

0	Most significant bit first
1	Least significant bit first

Ack Line - 5:6

Acknowledge line control

0	Tristate the Ack line
1	Use the Ack line as an extra output. Drive the value of InvertAck onto the line.
3	Use Ack for handshaking

Exchange Pins - 7

Drive the Ack signal onto the Req pin instead of the Ack pin. If Req Source in GroupiReqManagement = Ack pin, then the Req and Ack pins exchange functions. This would allow, for example, direct connection of two identical boards.

Group1StartDelay*i*Register **Write 0x 058, 0x059, 0x05A, 0x05B**

Group 1 Start Delay i	Group 1 Start Delay i	Group 1 Start Delay i	Group 1 Start Delay i	Group 1 Start Delay i	Group 1 Start Delay i	Group 1 Start Delay i	Group 1 Start Delay i

Group2StartDelay*i*Register **Write 0x 078, 0x079, 0x07A, 0x07B**

Group 2 Start Delay i	Group 2 Start Delay i	Group 2 Start Delay i	Group 2 Start Delay i	Group 2 Start Delay i	Group 2 Start Delay i	Group 2 Start Delay i	Group 2 Start Delay i

Delay from the end of the handshaking sequence to the restarting of the sequence. In fixed-rate I/O (data acquisition or pattern generation), set Start Delay to the sample interval, less Req Delay and Req* Delay in sequence 2 or 3.

Start Delay = board clock freq/sampling freq - Req Delay - Req Not Delay

Group1StartupAndClocking **Write 0x 042**

Group 1 Invert Stop	Group 1 Clock Line	Group 1 Clock Line	Group 1 Clock Source	Group 1 Idle Drive	Group 1 Idle Ack	Group 1 Begin Event	Group 1 Begin Event

Group2StartupAndClocking **Write 0x 062**

Group 2 Invert Stop	Group 2 Clock Line	Group 2 Clock Line	Group 2 Clock Source	Group 2 Idle Drive	Group 2 Idle Ack	Group 2 Begin Event	Group 2 Begin Event

Begin Event - 0:1

Event on which to begin handshaking. If you do not start handshaking on ASIC ready for transfer, you must ensure the ASIC is ready before entering run mode. This register does not apply to burst mode.

0	ASIC ready for transfer.
1	Req.
2	Req* (not available in sequence 1)

Idle Ack - 2

The value of Ack to assert at startup

Idle Drive - 3

Causes the ASIC to drive output data while idle (ignored in input mode)

Clock Source - 4

Source of the group handshaking clock, up to 20MHz, which runs all handshaking state machines.

0	Master handshaking clock, either from the Oscillator pin or from the RTSIClock pin, depending on the value of the RTSIClocking register.
1	Input clock from the group's peripheral clock (PClock) line. If ClockSource is 1, ClockLine should be 0 (PClock line undriven).

Clock Line - 5:6

Clock signal to drive on the peripheral clock line, if any.

0	Tristate the peripheral's Clock line and use it, if ClockSource=1, as the handshaking clock input.
1	Use the Clock line as an extra output. Drive the value of InvertClock onto the line.
3	Drive the clock signal used by the handshaking state onto the Clock line.

Invert Stop - 7

Invert the polarity of the stop trigger. The Stop Trigger Source is selected by Stop Source in the GroupiDaqOptions register.

Group1TransferCount*i*Register **Write 0x 014, 0x015, 0x016, 0x017**

Group 1 Transfer Count i	Group 1 Transfer Count i	Group 1 Transfer Count i	Group 1 Transfer Count i	Group 1 Transfer Count i	Group 1 Transfer Count i	Group 1 Transfer Count i	Group 1 Transfer Count i

Group2TransferCount*i*Register **Write 0x 018, 0x019, 0x020, 0x021**

Group 2 Transfer Count i	Group 2 Transfer Count i	Group 2 Transfer Count i	Group 2 Transfer Count i	Group 2 Transfer Count i	Group 2 Transfer Count i	Group 2 Transfer Count i	Group 2 Transfer Count i

Transfer Count *i* - 0:31

In Numbered mode, write the requested number of transfers to the transfer count register, or zero for the maximum number of counted transfers. Transfer Count is ignored when the group is not in Numbered mode. Numbered mode is set in the GroupiOperatingMode Register.

Group1TransferSize **Write 0x 04D**

X	X	Group 1 Require Ready Level	Group 1 Transfer Length	Group 1 Transfer Length	Group 1 Transfer Order	Group 1 Transfer Width	Group 1 Transfer Width

Group2TransferSize **Write 0x 06D**

X	X	Group 2 Require Ready Level	Group 2 Transfer Length	Group 2 Transfer Length	Group 2 Transfer Order	Group 2 Transfer Width	Group 2 Transfer Width

Transfer Width - 0:1

In the case of DMA or group-FIFO reads and writes, each DMA cycle or FIFO access performed should match the specified width. In the case of direct-FIFO transfers, you should set TransferWidth to 0 (see the section on direct FIFO registers).

0 (or 1)	32-bit DMA or group-FIFO transfers, or direct-FIFO transfers.
2	8-bit DMA or group-FIFO transfers. Only the low-order 8 bus lines perform DMA and group-FIFO transfers. However, transfers take place between these pins and each of the group's FIFOs in succession. If the group contains more than one FIFO, the TransferOrder bit determines which FIFO transfers data first.
3	16-bit DMA or group-FIFO transfers. Only the low-order 16 bus lines perform DMA. However, transfers take place between these pins and all of the group's FIFOs, two at a time, in succession. If the group contains more than two

FIFOs, the TransferOrder bit determines which ports transfer data first. A group performing 16-bit DMA must contain two or four ports.

Transfer Order - 2

DMA or group-FIFO byte order, when the number of FIFOs in the group does not match the TransferWidth.

0	Increment, starting with the lowest FIFO (FIFO A, if used)
1	Decrement, starting with the highest FIFO (FIFO D, if used)

Transfer Length - 3:4

Maximum TransferReady (DMA request) duration. Useful for DMA on some platforms. Provides a time-out mechanism, required by certain DMA controllers, to prevent the DMA process from monopolizing the bus. When the time-out expires, the ASIC holds DMAReq low until DMAAck deactivates.

0	No limit
1	Maximum of 1transfer (deassert DRQ after every transfer)
2	Maximum of 8 transfers
3	Maximum of 16 transfers

Require Ready Level - 5

0	DMAReq asserts when the FIFOs meet the ReadyLevel and stays asserted as long as data (in input mode) or spaces (in output mode) remain. In a Numbered input-mode transfer, DMAReq goes high after the expiration of TransferCount, to indicate that the FIFOs should be drained.
1	DMAReq simply reflects the current state of the FIFOs, indicating whether they meet the programmed ReadyLevel

WindowAddressRegister **Write 0x 004**

Address	Address	Address	Address	Address	Address	Address	Address

WindowDataRegister **Read/Write 0x 000**

Data	Data	Data	Data	Data	Data	Data	Data

Data	Data	Data	Data	Data	Data	Data	Data

Data	Data	Data	Data	Data	Data	Data	Data

Data	Data	Data	Data	Data	Data	Data	Data

The WindowAddressRegister and WindowDataRegister can access any other register on the DAQ-DIO. To access a given register, first write the given register's address to the WindowAddressRegister, then either read or write to the WindowDataRegister. The WindowAddressRegister acts like a multiplexer and connects the WindowDataRegister to the given register. This technique is used on AT-DIO-32HS boards to reduce the amount system IO resources required by the board. These registers are usually not used on PCI or PXI DIO boards.

Address 0:7

Address of the DIO register that will be accessed for the next WindowDataRegister read or write.

Data 0:31

Data written or read from the DIO register which is at the address specified by the WindowAddressRegister.