

Digilent Adept Analog Input/Output (DAIO) Programmer's Reference Manual

Revision: August 16, 2010



1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Introduction

This specification defines the Digilent Adept Analog Input/Output Interface (DAIO) subsystem for version 2 of the Digilent Adept software system. This document describes the capabilities of the DAIO subsystem and the API functions used to access its features.

The DAIO subsystem is used to access analog inputs and analog outputs using Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters on Digilent Adept compatible devices.

DAIO Ports and Port Properties

A DAIO port corresponds to a single A/D or D/A converter. A converter may support multiple channels, but the converter sets the basic conversion properties. All channels on the port will have the same conversion properties. Ports that support continuous, constant rate sampling will divide the available sample rate between all enabled channels.

The port property bits are used to indicate which optional parts of the DAIO interface are supported by a given port.

The following port property bits are defined for DAIO ports: These values are defined in the header file *daio.h*.

dprpAioInput	This bit indicates that the port is an analog input (A/D) port.
dprpAioOutput	This bit indicates that the port is an analog output (D/A) port.
dprpAioSingle	This bit indicates that the port supports single sample mode operation
dprpAioContinuous	This bit indicates that the port supports continuous sampling at a constant rate.
dprpAioWaveOut	This bit indicates that the port supports wave table output. This bit will only be set if the dprpAioContinuous bit is set.
dprpAioOneShot	This bit indicates that the port supports one-shot operation. This bit will only be set if the dprpAioContinuous bit is set.
dprpAioBuffered	This bit indicates that the port supports buffered mode operation when doing continuous sampling.
dprpAioFifo	This bit indicates that the port supports fifo mode operation when doing continuous sampling.
dprpAioSetRate	This bit indicates that the port supports setting the sample rate for continuous sampling.
dprpAioSetFormat	This bit indicates that the port supports setting the sample format, e.g. the number of significant bits per sample.
dprpAioSetGain	This bit indicates that the port supports setting the input gain/output level.
dprpAioSetOffset	This bit indicates that the port supports setting the DC offset level.

dprpAioSetLow	This bit indicates that the port supports setting the low cutoff frequency for the input/output filter.
dprpAioSetHigh	This bit indicates that the port supports setting the high cutoff frequency for the input/output filter.
dprpAioSetRef	This bit indicates that the port supports setting the reference voltage for the converter.
dprpAioExtRef	This bit indicates that the port supports an externally applied reference voltage that can be selected by setting the reference voltage to 0. This bit will only be set if the dprpAioSetRef bit is also set.

Sample Format

The sample format describes the nature of the data making up an individual sample read from an A/D converter to be written to a D/A converter. Some ports on some devices allow setting the sample format, in other cases, the sample format is fixed by the converter.

The following symbols defined in daio.h are used to describe the sample format and when setting the sample format:

sprpAioSmpSigned	Samples are twos-complement signed values
sprpAioSmpUnsigned	Samples are unsigned binary values
sprpAioSmpLeft	The significant bits of the sample are left justified in the sample value
sprpAioSmpRight	The significant bits of the sample are right justified in the sample value.
sprpAioSmpPacked	When operating in continuous sampling mode, this indicates that the samples are packed into the buffer without regard to byte boundaries.

Sample Buffer Mode

Buffering of samples applies to ports/channels that are in a continuous sampling mode. There are two styles of buffering: dprpFifo and dprpBuffered.

The *bfmtAioFifo* buffering style uses the available buffer memory as a first-in-first-out circular queue. In this case, samples can be read from or written to the buffer individually, or in varying sized increments.

The *bfmtAioBuffered* style divides the available buffer memory into some number of equal sized buffers. In this mode, buffers are read or written as integral units. The exact buffer size of samples must be read or written for each input or output operation.

Since channels in *bfmtAioBuffered* mode must read in buffer sized increments, in some cases, such as when the sample rate is very low, it is more desirable to be able to read samples as they become available, rather than having to wait for a buffer to fill. In cases like this, it is preferable to use *bfmtAioFifo* mode instead.

DAIO API Functions

The following API functions make up the DAIO interface.

DaioGetVersion(char * szVersion)

Parameters:

szVersion - pointer to buffer to receive version string

This function returns a version number string identifying the version number of the DAIO DLL.

DaioGetPortCount(HIF hif, INT32 * pcprt)

Parameters

hif - open interface handle on the device
pcprt - pointer to variable to receive count of ports

This function returns the number of AIO ports supported by the device specified by *hif*.

DaioGetPortProperties(HIF hif, INT32 prtReq, DWORD * pdprp)

Parameters:

hif - open interface handle on the device
prtReq - port number to query
pdprp - pointer to variable to return port property bits

This function returns the port property bits for the specified AIO port. The port property bits indicate the specific features of the DAIO port specification implemented by the specified port.

DaioEnable(HIF hif)

Parameters:

hif - open interface handle on the device

This function is used to enable the default DAIO port (port 0) on the specified device. A DAIO port must be enabled before any functions that operate on that port may be called for the specified device.

DaioEnableEx(HIF hif, INT32 prtReq)

Parameters:

hif - open interface handle on the device
prtReq - DAIO port number

This function is used to enable a specific port on devices that support multiple DAIO ports. A DAIO port must be enabled before any functions that operate on the port may be called. The *prtReq* parameter specifies the port number of the AIO port to enable.

DaioDisable(HIF hif)*Parameters:*

hif - open interface handle on the device

This function is used to disable and end access to the currently enabled DAIO port on the specified interface handle.

DaioGetChannelCount(HIF hif, INT32 * pcchnCnt)*Parameters:*

hif - open interface handle on the device
pcchnCnt - variable to receive the channel count

This function is used to return the number of channels supported by the currently enabled DAIO port on the specified interface handle.

DaioChannelEnable(HIF hif, INT32 chn)*Parameters:*

hif - open interface handle on the device
chn - channel number of the channel to enable

This function is used to enable a channel on the currently enabled DAIO port on the specified interface handle.

DaioChannelDisable(HIF hif, INT32 chn)*Parameters:*

hif - open interface handle on the device
chn - channel number of the channel to disable

This function is used to disable a channel on the currently enabled DAIO port on the specified interface handle.

DaioGetSample(HIF hif, INT32 chn, INT32 * psmp)*Parameters:*

hif - open interface handle on the device
chn - channel number
smp - sample value to write

This function is used to read a single sample from an input channel that is in single sample mode.

DaioPutSample(HIF hif, INT32 chn, INT32 smp)*Parameters:*

hif - open interface handle on the device
chn - channel number
smp - sample value to write

This function is used to write a single sample to an output channel that is in single sample mode.

DaioGetBuffer(HIF hif, INT32 chn, DWORD cbRcv, void * rgbRcv, BOOL fOverlap)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
cbRcv	- number of bytes of sample data to return
rgbRcv	- pointer to buffer to receive sample data
fOverlap	- TRUE for overlapped operation, FALSE for blocking operation

This function is used to read a buffer of samples from an input channel that is in a continuous sampling mode.

If the buffer mode for the port is in *bfmdAioBuffered* mode, the number of bytes read must be the exact size of the buffer. If there are an insufficient number of samples available to be read, the behavior depends on the wait mode specified using the *DaioSetBufferMode* function. If the wait mode is *bfmdAioWait*, the function will wait until data becomes available to be returned. If the wait mode is *bfmdAioFail*, the function will fail immediately without returning any data.

If the port is in *bfmdAioFifo* mode, any number of bytes can be read, except that the number of bytes read must be a multiple of the number of bytes per sample. If more bytes are requested to be read than are currently available in the fifo, the behavior depends on the wait mode. If the wait mode is *bfmdAioWait*, the function will wait until the requested bytes are available to be read. If the wait mode is *bfmdAioFail*, the function will return whatever data is currently available in the buffer.

If the wait mode is *bfmdAioWait*, the port should be running (run mode set to TRUE) or else the call will time out, as no data will ever become available to be read.

DaioPutBuffer(HIF hif, INT32 chn, DWORD cbSnd, void * rgbSnd, BOOL fOverlap)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
cbSnd	- number of sample values to send
rgbSnd	- pointer to buffer of sample data to send
fOverlap	- TRUE for overlapped operation, FALSE for blocking operation

This function is used to write a buffer of samples to an output channel that is in a continuous sampling mode.

If the buffer mode for the port is in *bfmdAioBuffered* mode, the number of bytes written must be the exact size of the buffer. If there isn't enough buffer space available to contain the number of bytes being written, the behavior depends on the wait mode specified using the *DaioSetBufferMode* function. If the wait mode is *bfmdAioWait*, the function will wait until buffer space becomes available to contain the data being written. If the wait mode is *bfmdAioFail*, the function will fail at the point where there isn't sufficient buffer space to contain the remaining data to be written.

If the port is in *bfmdAioFifo* mode, any number of bytes can be written, except that the number of bytes written must be a multiple of the number of bytes per sample. If more bytes are written than there is buffer space available, the behavior depends on the wait mode. If the wait mode is *bfmdAioWait*, the function will wait until all bytes can be written. If the wait mode is *bfmdAioFail*, the function will fail at the point where the buffer becomes full.

If the wait mode is `bfmdAioWait`, the port should be running (run mode set to `TRUE`) or else the call will time out and fail as it is waiting for buffer space that will never become available.

DaioGetReference(HIF hif, double * pvltCur);

Parameters:

hif - open interface handle on the device
pvltCur - variable to receive the current reference value

This function returns the current reference voltage for the converter associated with this port. The reference voltage determines the maximum conversion range for the converter associated with the port. The returned value is in volts.

DaioSetReference(HIF hif, double vltReq double * pvltSet)

Parameters:

hif - open interface handle on the device
vltReq - requested reference voltage
pvltSet - variable to receive the reference voltage obtained

This function sets the reference voltage for the converter associated with the port. The requested reference voltage is specified in `vltReq`. The reference voltage will be set to the nearest supported value that doesn't exceed the requested value. The actual value set is returned in `pvltSet`. The reference value is specified in volts. A value of 0 selects an externally applied voltage as the reference on devices that support this feature.

DaioGetLowCutoff(HIF hif, INT32 chn, double * pfrqCur)

Parameters:

hif - open interface handle on the device
chn - channel number
pfrqCur - variable to receive the current low cutoff frequency

This function returns the low cutoff frequency for the input/output filter. The returned value is in Hz. The return value will be 0 if the device doesn't have an input/output filter with a low cutoff frequency.

DaioSetLowCutoff(HIF hif, INT32 chn, double frqReq, double * pfrqSet)

Parameters:

hif - open interface handle on the device
chn - channel number
frqReq - requested value for low cutoff frequency
pfrqSet - variable to receive the frequency obtained

This function is used to set the low cutoff frequency for the input/output filter. The requested value is specified in Hz. The low cutoff frequency will be set to the nearest supported value that doesn't exceed the requested value. The actual value set is returned in `pfrqSet`. This function is ignored if the devices doesn't support setting the low cutoff frequency on the input/output filter.

DaioGetHighCutoff(HIF hif, INT32 chn, double * pfrqCur)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
pfrqCur	- variable to receive the current high cutoff frequency

This function returns the high cutoff frequency for the input/output filter. The returned value is in Hz. The return value will be 0 if the device doesn't have an input/output filter with a high cutoff frequency.

DaioSetHighCutoff(HIF hif, INT32 chn, double frqReq, double * pfrqSet)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
frqReq	- requested value for low cutoff frequency
pfrqSet	- variable to receive the frequency obtained

This function is used to set the high cutoff frequency for the input/output filter. The requested value is specified in Hz. The high cutoff frequency will be set to the nearest supported value that doesn't exceed the requested value. The actual value set is returned in *pfrqSet*. This function is ignored if the devices doesn't support setting the high cutoff frequency on the input/output filter.

DaioGetGain(HIF hif, INT32 chn, double * pcgnCur)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
pcgnCur	- variable to receive the current channel gain setting

This function returns the current gain/attenuation of the specified channel. The value will be greater than 1.0 for gain and between 0.0 and 1.0 for attenuation. The returned value will be 1.0 for channels that don't support settable gain/attenuation.

DaioSetGain(HIF hif, INT32 chn, double cngReq, double * pcgnSet)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
cngReq	- channel gain to set
pcgnSet	- variable to receive the channel gain setting obtained

This function is used to set the gain/attenuation of the specified channel. The channel gain will be set to the nearest supported value that doesn't exceed the requested value. Values greater than 1.0 specify gain, and values between 0.0 and 1.0 specify attenuation. The actual channel gain value obtained is returned in *pcgnSet*. This function is ignored if the channel doesn't support setting the gain/attenuation.

DaioGetOffset(HIF hif, INT32 chn, double * pvltCur)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
pvltCur	- variable to receive the current channel offset voltage

This function returns the current input/output offset voltage of the specified channel. The return value will be 0 if the channel doesn't support setting the offset voltage.

DaioSetOffset(HIF hif, INT32 chn, double vltReq, double * pvltSet)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
vltReq	- channel offset voltage to set
pvltSet	- variable to receive the offset voltage obtained

This function is used to set the input/output offset voltage for the specified channel. This function will be ignored if the channel doesn't support setting the offset voltage.

DaioSetPortMode(HIF hif, DWORD smdMode)*Parameters:*

hif	- open interface handle on the device
smdMode	- sampling mode to set for the port

This function is used to set the conversion mode for the currently enabled AIO port on the specified interface handle. The conversion mode can be set to either *smdAioSingle* which indicates that all enabled channels on the port will be in single sample mode, or *smdAioContinuous*, which indicates that all enabled channels on the port will be in continuous sampling mode.

DaioGetBufferInfo(HIF hif, INT32 chn, INT32 * pcbufMax, INT32 * pcbMax, INT32 * pcbBuf)*Parameters:*

hif	- open interface handle on the device
chn	- channel number
pcbufMax	- variable to receive maximum number of buffers
pcbMax	- total buffer memory available
pcbBuf	- largest size for an individual buffer

This function returns the number of buffers and the size of each buffer available for the specified channel. The value returned in *pcbufMax* indicates the maximum number of buffers that can be set for the channel. The value returned in *pcbMax* indicates the total amount of memory available for buffers for the channel. The value returned in *pcbBuf* indicates the largest size an individual buffer can be.

The actual number of buffers to use is specified by the function *DaioSetBufferMode*. Generally, the available buffer space will be divided evenly between all buffers. If, however, this would result in buffers being larger than *pcbBuf*, the buffers will be this smaller size, and some of the available buffer memory will be unused. Additionally, many devices only support buffers that are a power-of-two number of bytes in size.

In some devices, the amount of buffer memory available to a channel depends on the number of enabled channels. In other devices, the available memory to each channel is fixed and doesn't change depending on the number of channels enabled. Refer to product documentation for the device to get information on the specific behavior for a given device.

DaioSetBufferMode(HIF hif, INT32 chn, DWORD bfmdReq, DWORD * pbfmdSet, INT32 cbufReq, INT32 * pcbufSet)

Parameters:

hif	- open interface handle on the device
chn	- channel number
bfmdReq	- requested buffer style to set for the channel
pbfmdSet	- actual buffer style set
cbufReq	- requested number of buffers to use
pcbufSet	- actual number of buffers obtained

This function is used to select the operation of buffers associated with the specified channel when the port is operating in continuous sampling mode. The buffer mode specified by *bfmdReq* specifies the buffer style to use and the wait style to use. The value to be passed in this parameter is made up of the bitwise-or of the buffer style and the wait style. The buffer style can be set to *bfmdAioBuffered* or *bfmdAioFifo* assuming that the port supports both modes. The wait style can be set to *bfmdAioWait* or *bfmdAioFail*.

If the buffer style set is *bfmdAioFifo*, then the *cbufReq* and *pcbufSet* values are ignored. In this mode, all available buffer memory is assigned to a single fifo for the channel.

If the buffer style set is *bfmdAioBuffered*, the *cbufReq* parameter specifies the number of buffers to assign to the channel. The *pcbufSet* value indicates the number of buffers actually assigned to the channel. This value will be less than *cbufReq* if it isn't possible to assign the requested number of buffers to the channel. When a channel is in *bfmdAioBuffered* mode, reads or writes on that channel must be done on whole-buffer numbers of bytes. (i.e. buffers are read or written as integral units).

The wait style specifies the behavior when all available buffer space is full on write, or empty on read. If the mode is set to *bfmdAioWait*, the function will not return until the data can be successfully written or read.

If the wait style is set to *bfmdAioFail*, the function call will fail on a write when there is no available buffer space or on a read when there is no data to be read. The operation will fail at the point where the buffers become full/empty, so some data may have been transferred before the buffer full/empty condition occurs and the operation fails.

Refer to product documentation for the device to determine the supported ranges of values for a specific device. This function is ignored on devices that don't support selectable buffer mode or wait style.

DaioSetChannelMode(HIF hif, INT32 chn, DWORD dprpChmd)

Parameters:

hif - open interface handle on the device
 chn - channel number
 dprpChmd - channel mode to set

This function is used to set the continuous sample mode to use for the specified channel when for a DAIO port that is in continuous sampling mode. The channel modes that can be set are: *dprpContinuous* which indicates continuous sampling; *dprpWaveOut* which indicates wave table output; or *dprpOneShot* which indicates single shot operation.

This function is ignored on devices that don't support setting the channel mode.

DaioGetRunMode(HIF hif, DWORD * prmdCur)

Parameters:

hif - open interface handle on the device
 prmdCur - pointer to variable to receive current run mode

This function is used to query the run mode currently in effect for the port.

DaioSetRunMode(HIF hif, DWORD rmdRun)

Parameters:

hif - open interface handle on the device
 rmdRun - run mode to set

This function is used to start/stop sampling on a DAIO port that is operating in a continuous sampling mode. Setting the run mode to *rmdAioRun* will cause the enabled channels on the port to start generating/consuming sample data from the buffers associated with the channel. Setting the run mode to *rmdAioRunSingle* will cause the enabled channels to perform a single capture operation and then enter the *rmdAioStop* state. Setting the run mode to *rmdAioStop* will stop sampling. Setting the run mode to *rmdAioArm* will cause the port to wait for a trigger event and then enter the *rmdAioRun* state. Setting the run mode to *rmdAioArmSingle* will cause the port to wait for a trigger event and then enter the *rmdAioRunSingle* state. (*rmdAioRunSingle*, *rmdAioArm*, *rmdAioArmSingle* are reserved for future use).

The *rmdAioRunSingle* and *rmdAioArmSingle* modes can only be used with channels that support one-shot operation (i.e. support the *dprpAioOneShot* property).

DaioGetSampleRate(HIF hif, double * pfrqCur)

Parameters:

hif - open interface handle on the device
 pfrqCur - variable to receive current sample rate of the converter

This function returns the current sample rate for the port. The returned value is in samples per second (i.e. sample rate in Hz). This sample rate is divided evenly between all enabled channels on the port.

DaioSetSampleRate(HIF hif, double frqReq, double * pfrqSet)

Parameters:

<code>hif</code>	- open interface handle on the device
<code>frqReq</code>	- requested sample rate in Hz.
<code>pfrqSet</code>	- variable to receive actual sample rate obtained

This function sets the sample rate for the port. The sample rate is set to the nearest value that doesn't exceed the requested value. The actual sample rate obtained is returned in `pfrqSet`. This sample rate is evenly divided between all enabled channels on the port. Using this function is only meaningful on DAIO ports that support the `dprpAioContinuous` property. This function should be called before the port is placed in continuous sampling mode via the `DaioSetPortMode` function.

DaioGetSampleFormat(HIF `hif`, INT32 * `pcbtSmp`, INT32 * `pcbtSig`, DWORD * `psprpFmt`)

Parameters:

<code>hif</code>	- open interface handle on the device
<code>pcbtSmp</code>	- variable to receive number of bits in the sample
<code>pcbtSig</code>	- variable to receive the number of significant bits in the sample
<code>psprpFmt</code>	- variable to receive format description bits for the sample

This function returns the information about the format of samples used by the converter associated with the DAIO port. The `pcbtSmp` value indicates the number of bits returned for each sample. This value will be a multiple of eight, as samples are always returned as whole numbers of bytes. The `pcbtSig` value indicates the number of bits within a sample that contain actual sample data. The remaining, non-significant bits in the sample will generally be 0. However, in the case of right justified, signed samples, the leading non-significant bits in a negative sample value will be 1. The `psprpFmt` value contains bit flags indicating additional information about the format of the samples. This includes information about whether the significant bits are left justified or right justified within the sample, and whether the samples are signed or unsigned values.