1300 Henley Court
Pullman, WA 99163
509.334.6306
www.digilentinc.com

WaveForms™ SDK Reference Manual

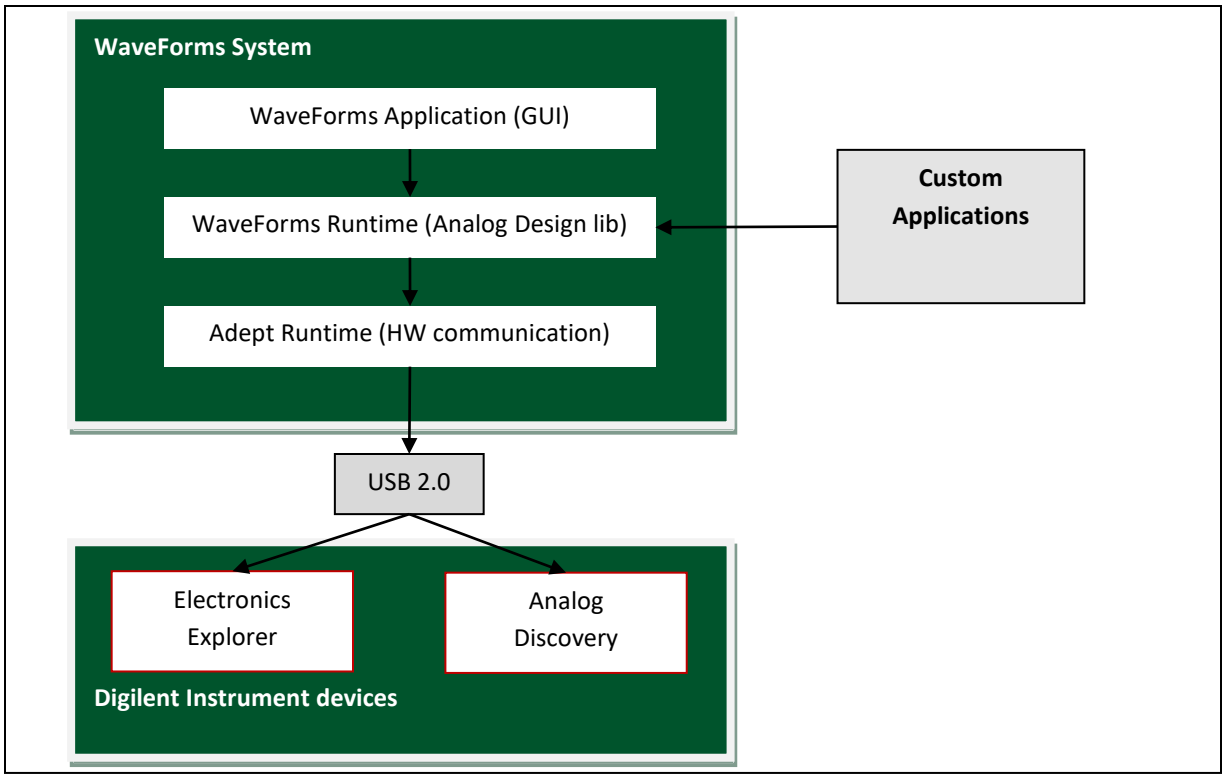
Revised April 4, 2019

Table of Contents

Overview

WaveForms™ provides an interface that allows users to interact with Digilent Analog Design hardware, such as the Analog Discovery™, Analog Discovery 2™, Digital Discovery™, and Electronics Explorer™. While the WaveForms application offers a refined graphical interface, the WaveForms SDK provides access to a public application programming interface (API) that gives users the ability to create custom PC applications.

This WaveForms SDK manual describes the main components and architecture of the WaveForms system and details each function contained in the WaveForms API. The SDK package also offers examples demonstrating how to identify, connect to, and control analog hardware devices.



1 The System

The WaveForms System is comprised of multiple components. The most visible component is the WaveForms Application; a suite of graphical instrument panels that give full access to the analog and digital instruments in the connected hardware. The WaveForms application uses WaveForms Runtime to communicate with the device. For a custom application it is sufficient to access only the WaveForms Runtime, but both of these Runtimes are required on the target machine in order to run the application. The WaveForms Runtime is comprised of the *DWF* Dynamic Library and several configuration files. This library is located in:

- Windows in System Directory: C: \Windows\System32\dwf.dll
- Linux: /usr/lib/libdwf.so.x.x.x

The static library is located in Windows through the install path:

- Windows 32-bit: C:\Program Files\Digilent\WaveFormsSDK\lib\x86
- Windows 64-bit: C:\Program Files (x86)\Digilent\WaveFormsSDK\lib\x64

The C header file is located in:

- Windows: C:\Program Files\Digilent\WaveFormsSDK\inc
- Windows 64-bit: C:\Program Files (x86)\Digilent\WaveFormsSDK\inc
- Linux: /usr/include/digilent/waveforms

Working code examples are provided with the SDK to demonstrate basic use of each API function set. You can find samples in the installation directory, which are located here:

- Windows 32-bit: C:\Program Files\Digilent\WaveFormsSDK\samples
- Windows 64-bit: C:\Program Files (x86)\Digilent\WaveFormsSDK\samples
- Linux: /usr/share/digilent/waveforms/samples
- OS X: /Applications/WaveForms.app/Contents/Resources/SDK/

The DWF Library uses Digilent Adept Runtime, which provides basic communication with the targeted hardware instruments (i.e., Analog Discovery and Electronics Explorer). Although the Adept Runtime is an integral part of the WaveForms System, knowledge of its structure *is not required* to write custom applications.

On Mac OS X the WaveFroms Runtime (including Adept Runtime) is installed to:

- /Library/Frameworks/dwf.framework

The sample path contains the following directories:

- py: several Python script examples
- C: some C code examples
- cs: C# wrapper, dwf class for the API functions
- vb: VisualBasic wrapper, dwf module for the API function
- dwfcmd: a complex C example application

1.1 The API

Everything needed to write custom applications is included in the WaveForms SDK, which provides the header/library files and documentation to access the API for the DWF Library. A custom application must properly link to these files to make the appropriate API function calls. Every function in the WaveForms public API is declared in the dwf.h header file.

Basic usage of the WaveForms API can be broken down into the following steps:

1. Call enumeration functions to discover connected hardware devices.
2. Call *FDwfDeviceOpen* function to establish a connection to specific hardware device.
3. Call function to enable instrument within hardware device.
4. Call functions to configure instrument and acquire/generate signals.
5. Call function to disable instrument.
6. Call *FDwfDeviceClose* function to disconnect from device.

There are nine main groups of API functions, each named with different prefixes:

Main Groups of API Functions	Instrument Function	Prefix
Device Enumeration	Controls the enumeration of connected and supported devices.	*DwfEnum*
Device Control	Controls opening and closing specific devices.	*DwfDevice*
AnalogIn (Oscilloscope)	Acquires samples from each enabled channel synchronously.	*DfwAnalogIn*
AnalogOut (Arbitrary Waveform Generator)	Drives signals from each channel independently.	*DfwAnalogOut*
AnalogIO	Acquires and drives various analog signals.	*DfwAnalogIO*
DigitalIn (Logic Analyzer)	Acquires samples from digital I/O pins.	*DfwDigitalIn*
DigitalOut (Pattern Generator)	Drives digital I/O signals.	*DfwDigitalOut*
DigitalIO	Acquires and drives digital I/O signals.	*DfwDigitalIO*
System	Obtain basic system information that is instrument and device independent.	*DfwGet*

Each instrument is directly controlled using three types of functions in the API:

API Functions	Instrument Function	Example
Reset function	This function resets all of the instrument parameters to default values.	*FDwfAnalogInReset*
		FDwfAnalogOutReset
		FDwfDigitalIOReset
Configure function	This function configures and/or starts the instrument.	*FDwfAnalogInConfigure*
		FDwfAnalogOutConfigure
		FDwfDigitalIOConfigure
Status function	This function polls and reads all information from the instrument.	*FDwfAnalogInStatus*
		FDwfAnalogOutStatus
		FDwfDigitalIOStatus

Note: Although there are multiple "Status" functions for each instrument, these functions are the only ones that actually read data from the device.

There are several type definitions and corresponding constants in the dwf.h include file. The majority of them are used as parameters. When a hardware device is opened, a handle is returned (HDWF), which is used to access and finally close in all instrument API functions.

The following examples are provided in Python and C++ language.

File	Description
Device_Enumeration	List the supported and connected devices.
AnalogIO_AnalogDiscovery_SystemMonitor	Reading the system monitor information
AnalogIO_AnalogDiscovery_Power	Enable power supplies.
AnalogOut_Sine	Generate sine waveform on analog out channel.
AnalogOut_Sweep	Generate frequency sweep.
AnalogOut_Custom	Arbitrary waveform generation.
AnalogOut_Sync	How to synchronize the analog output channels
AnalogOutIn	Generate analog output signal and perform analog in acquisition.
AnalogIn_Sample	Open the first device, configure analog in and read single sample.
AnalogIn_Acquisition	Perform acquisition and plot data for first channel.
AnalogIn_Trigger	Perform triggered acquisition.
AnalogIn_Record	Performs recording of large number of samples.
DigitalIO	Drive and read digital IO pins
DigitalOut_Pins	Generate pulse, random and custom signal on digital out pins.
DigitalOut_BinrayCounter	Generate binary counter
DigitalIn_Acquisition	Generate signals on digital out and perform acquisition on digital in.
DigitalIn_Record	Perform recording of large number of digital in samples.

1.2 Calling API Functions

The API functions are C style and return a Boolean value: TRUE if the call is successful, FALSE if unsuccessful. This Boolean value is an integer type definition, *not* the standard c-type bool. In general, the API functions contain variations of the following parameters:

Parameters	Parameter Function
***Info**	Returns detailed information about the parameter support for the instrument (i.e., minimum/maximum values, supported modes, etc.)
***Set**	Sets an instrument parameter. When the AutoConfigure is enabled (by default), the instrument is reconfigured and stopped.
***Get**	Gets the actual instrument parameter. Use this function to get the actual set value. For instance, an arbitrary voltage offset is set and Get returns the real DAC output value.
***Status**	Returns the parameter value from the device.

The API functions won't fail when a parameter pointer is NULL or when a setting (*Set) parameter value is out of limits. To verify the actual setting value, use *Get API to return the actual value.

The indices used in function parameters are zero based.

The supported discrete parameters are retrieved in bit field value. To decode the capabilities of the device, use the `IsBitSet` macro.

```
int fsfilter;
FDwfAnalogInChannelFilterInfo(h, &fsfilter)
if(IsBitSet(fsfilter, filterAverage)){
    FDwfAnalogInChannelFilterSet(hdwf, 0, filterAverage)
}
```

2 System

FDwfGetLastError(DWFERC *pdwferc)

Description: Retrieves the last error code in the calling process. The error code is cleared when other API functions are called and is only set when an API function fails during execution. Error codes are declared in dwf.h:

DWFERC	int	Error Code Definition
dwfercNoErc	0	No error occurred.
dwfercUnknownError	1	Call waiting on pending API time out.
dwfercApiLockTimeout	2	Call waiting on pending API time out.
dwfercAlreadyOpened	3	Device already opened.
dwfercNotSupported	4	Device not supported.
dwfercInvalidParameter0	5	Parameter 0 was invalid in last API call.
dwfercInvalidParameter1	6	Parameter 1 was invalid in last API call.
dwfercInvalidParameter2	7	Parameter 2 was invalid in last API call.
dwfercInvalidParameter3	8	Parameter 3 was invalid in last API call.

Parameters:

- pdwferc – Variable to receive error code.

FDwfGetLastErrorMsg(char szError[512])

Description: Retrieves the last error message. This may consist of a chain of messages, separated by a new line character, that describe the events leading to the failure.

Parameters:

- szError – Pointer to buffer to receive error string.

FDwfGetVersion(char szVersion[32])

Description: Retrieves the version string. The version string is composed of major, minor, and build numbers (i.e., "2.0.19").

Parameters:

- szVersion – Pointer to buffer to receive version string.

FDwfParamSet(DwfParam param, int value)

Description: Configures various parameters.

DwfParam	int	
DwfParamUsbPower	2	1 = Keep the USB power enabled even when AUX supply is connected 0 = Disable USB power when AUX is connected. Supported by Analog Discovery 2
DwfParamLedBrightness	3	0..100 % LED brightness Supported by Digital Discovery
DwfParamOnClose	4	Device close behavior. 0 = Continue, keep running 1 = Stop the device outputs but keep the device operational to prevent temperature drifts 2 = Shutdown device to minimize power consumption
DwfParamAudioOut	5	1 = Enable aution output 0 = Disable audio output Supported by Analog Discovery 1 and 2
DwfParamUsbLimit	6	USB current limitation in mA, recommended value 600-1000. Supported by Analog Discovery 1 and 2

Parameters:

- param – Pointer to buffer to receive version string.
- value – Value to set.

FDwfParamGet(DwfParam param, int *pValue)

Description: Retrieves the parameter value.

Parameters:

- param – Parameter.
- pValue – Pointer to value.

3 Device Enumeration

The *FDwfEnum* functions are used to discover all connected, compatible devices.

See examples: Device_Enumeration.py, Device_Info.py, Device_InfoEx.py.

FDwfEnum(ENUMFILTER enumfilter, int *pnDevice)

Description: Builds an internal list of detected devices filtered by the *enumfilter* parameter. It must be called before using other *FDwfEnum* functions because they obtain information about enumerated devices from this list identified by the device index.

ENUMFILTER	int	
enumfilterAll	0	Enumerate all supported devices
enumfilterEExplorer	1	Electronics Explorer
enumfilterDiscovery	2	Analog Discovery (1)
enumfilterDiscovery2	3	Analog Discovery 2
enumfilterDDiscovery	4	Digital Discovery

Parameters:

- enumfilter – Filter value to be used for device enumeration. Use the *enumfilterAll* constant to discover all compatible devices.
- pnDevice – Integer pointer to return count of found devices by reference.

FDwfEnumDeviceType(int idxDevice, DEVID *pDeviceId, DEVVER *pDeviceRevision)

Description: Returns the device ID and version ID.

DEVID	int	
devidEExplorer	1	Electronics Explorer
devidDiscovery	2	Analog Discovery (1)
devidDiscovery2	3	Analog Discovery 2
devidDDiscovery	4	Digital Discovery

Parameters:

- idxDevice – Zero based index of the enumerated device for which to return the type and revision.
- pDeviceId – Variable to return the device id.
- pDeviceRevision – Pointer to DEVVER instance to return the device revision by reference.

FDwfEnumDeviceIsOpened(int idxDevice, BOOL *pfIsUsed)

Description: Retrieves a Boolean specifying if a device is already opened by this, or any other process.
Parameters:
- idxDevice – Index of the enumerated device.
- pfIsUsed – Pointer to variable to receive Boolean indicating if the device is in use.

FDwfEnumUserName(int idxDevice, char szUserName[32])

Description: Retrieves the user name of the enumerated device.
Parameters:
- idxDevice – Index of the enumerated device.
- szUserName – Pointer to character array to return the user name string by reference.

FDwfEnumDeviceName(int idxDevice, char szDeviceName[32])

Description: Retrieves the device name of the enumerated device.

Parameters:

- idxDevice – Index of the enumerated device.
- szDeviceName – Pointer to character array to return the device name by reference.

FDwfEnumSN(int idxDevice, char szSN[32])

Description: Retrieves the 12-digit, unique serial number of the enumerated device.
Parameters:
- idxDevice – Index of the enumerated device.
- szSN – Pointer to character array to return the serial number by reference.

FDwfEnumConfig(int idxDevice, int *pcConfig)

Description: Builds an internal list of detected configurations for the selected device. The function above must be called before using other *FDwfEnumConfigInfo* function because this obtains information about configurations from this list identified by the configuration index.
Parameters:
- idxDevice – Index of the enumerated device.
- pcConfig – Integer pointer to return count of found configurations by reference.

FDwfEnumConfigInfo(int idxConfig, DwfEnumConfigInfo info, int *pValue)

Description: Returns information about the configuration. The information types are declared in dwf.h:

DwfEnumConfigInfo	int
DECIAnalogInChannelCount	1
DECIAnalogOutChannelCount	2
DECIAnalogIOChannelCount	3
DECIDigitalInChannelCount	4
DECIDigitalOutChannelCount	5
DECIDigitalIOChannelCount	6
DECIAnalogInBufferSize	7
DECIAnalogOutBufferSize	8
DECIDigitalInBufferSize	9
DECIDigitalOutBufferSize	10

Parameters:

- idxConfig – Index of the configuration for which to return the information.
- info – Information type.
- pValue – Integer pointer to return selected information type by reference.

4 Device Control

See Device_Synchronization.py example.

FDwfDeviceOpen(int idxDevice, HDWF *phdwf)

Description: Opens a device identified by the enumeration index and retrieves a handle. To automatically enumerate all connected devices and open the first discovered device, use index -1.

Parameters:

- idxDevice – Zero based index of the enumerated device.
- phdwf – Pointer to *HDWF* variable to receive opened interface handle by reference.

FDwfDeviceConfigOpen(int idxDevice, int idxCfg, HDWF *phdwf)

Description: Opens a device identified by the enumeration index with the selected configuration and retrieves a handle.

Parameters:

- idxDevice – Index of the enumerated device.
- idxCfg – Index of the device configuration.
- phdwf – Pointer to *HDWF* variable to receive opened interface handle by reference.

FDwfDeviceClose(HDWF hdwf)

Description: Closes an interface handle when access to the device is no longer needed. Once the function above has returned, the specified interface handle can no longer be used to access the device.

Parameters:

- hdwf – Interface handle to be closed.

FDwfDeviceCloseAll()

Description: Closes all opened devices by the calling process. It does not close all devices across all processes.

Parameters: None.

FDwfDeviceAutoConfigureSet(HDWF hdwf, BOOL fAutoConfigure)

Description: Enables or disables the AutoConfig setting for a specific device. When this setting is enabled, the device is automatically configured every time an instrument parameter is set. For example, when AutoConfigure is enabled, FDwfAnalogOutConfigure does *not* need to be called after FDwfAnalogOutRunSet. This adds latency to every Set function; just as much latency as calling the corresponding Configure function directly afterward. With value 3 the analog-out configuration will be applied dynamically, without stopping the instrument.

Parameters:

- hdwf – Interface handle.
- fAutoConfigure– Value for this option: 0 disable, 1 enable, 3 dynamic

FDwfDeviceAutoConfigureGet(HDWF hdwf, BOOL *pfAutoConfigure)

Description: Returns the AutoConfig setting in the device. See the function description for *FDwfDeviceAutoConfigureSet* for details on this setting.

Parameters:

- hdwf – Interface handle.
- pfAutoConfigure – Pointer to variable to receive the current value of this option.

DwfDeviceReset(HDWF hdwf)

Description: Resets and configures (by default, having auto configure enabled) all device and instrument parameters to default values.

Parameters:

- hdwf – Interface handle.

FDwfDeviceParamSet(HDWF hdwf, DwfParam param, int value)

Description: Configures various parameters.

Parameters:

- param – Pointer to buffer to receive version string.
- value – Value to set

FDwfDeviceParamGet(HDWF hdwf, DwfParam param, int *pValue)

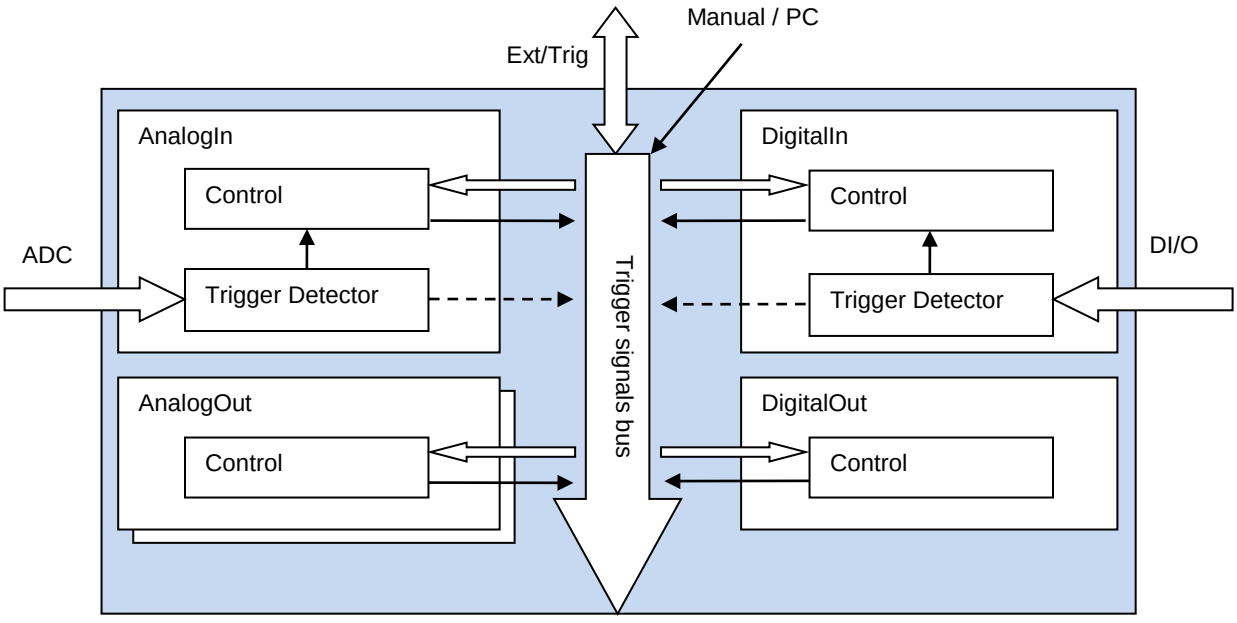
Description: Retrieves the parameter value.

Parameters:

- param – Parameter.
- pValue – Pointer to value.

The global trigger bus allows multiple instruments to trigger each other. These trigger source options are:

TRIGSRC	BYTE	Trigger Source Function
trigsrcNone	0	The trigger pin is high impedance, input. This is the default setting.
trigsrcPC	1	Trigger from PC, this can be used to synchronously start multiple instruments.
trigsrcDetectorAnalogIn	2	Trigger detector on analog in channels.
trigsrcDetectorDigitalIn	3	Trigger on digital input channels.
trigsrcAnalogIn	4	Trigger on device instruments, these output high when running.
trigsrcDigitalIn	5	Trigger on device instruments, these output high when running.
trigsrcDigitalOut	6	Trigger on device instruments, these output high when running.
trigsrcAnalogOut1	7	Trigger on device instruments, these output high when running.
trigsrcAnalogOut2	8	Trigger on device instruments, these output high when running.
trigsrcAnalogOut3	9	Trigger on device instruments, these output high when running.
trigsrcAnalogOut4	10	Trigger on device instruments, these output high when running.
trigsrcExternal1	11	External trigger signal.
trigsrcExternal2	12	External trigger signal.
trigsrcExternal3	13	External trigger signal.
trigsrcExternal4	14	External trigger signal.



FDwfDeviceTriggerInfo(HDWF hdwf, int *pfstrigsrc)

Description: Returns the supported trigger source options for the global trigger bus. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the TRIGSRC constants in dwf.h.

Parameters:

- hdwf – Interface handle.
- pfstrigsrc – Variable to receive the supported trigger sources.

FDwfDeviceTriggerSet(HDWF hdwf, int idxPin, TRIGSRC trigsrc)

Description: Configures the trigger I/O pin with a specific TRIGSRC option.

Parameters:

- hdwf – Interface handle.
- idxPin – External trigger, I/O pin index.
- trigsrc – Trigger source to set.

FDwfDeviceTriggerGet(HDWF hdwf, int idxPin, TRIGSRC *ptrigsrc)

Description: Returns the configured trigger setting for a trigger I/O pin. The trigger source can be “none”, an internal instrument, or an external trigger.

Parameters:

- hdwf – Interface handle.
- idxPin – External trigger, I/O pin index.
- ptrigsrc – Variable to receive the current trigger source.

FDwfDeviceTriggerPC(HDWF hdwf)

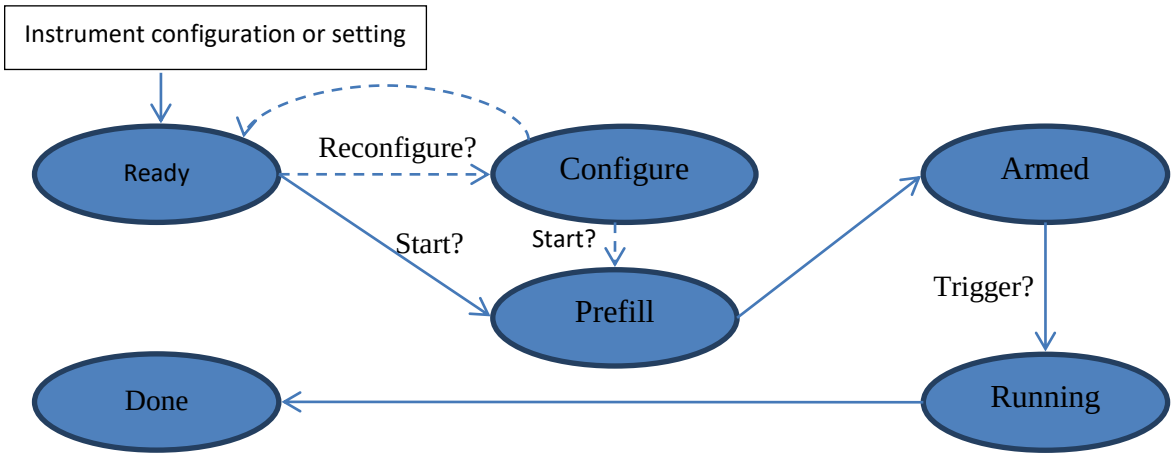
Description: Generates one pulse on the PC trigger line.

Parameters:

- hdwf – Interface handle.

5 Analog In (Oscilloscope)

The Analog In instrument states:



The states are defined in dwf.h DwfState type.

- **Ready**: Initial state. After *FDwfAnalogInConfigure* or any *FDwfAnalogIn*Set* function call goes to this state. With *FDwfAnalogInConfigure,* reconfigure goes to Configure state.
- **Configure**: The needed configurations are performed, and auto trigger is reset.
- **Prefill**: Prefills the buffer with samples needed before trigger.
- **Armed**: Waits for the trigger.
- **Running**:
    - Single acquisition mode: remains in this state to acquire samples after trigger according trigger position parameter.
    - Scan screen and shift modes: remains in this state until configure or any set function of this instrument.
    - Record mode: the time period according to the record length parameter.
- **Done**: Final state.

See the following examples: AnalogIn_Sample/Acquisition/Trigger/Record.py AnalogOutIn.py

5.1 Control

5.1 Control

FDwfAnalogInReset(HDWF hdwf)

Description: Resets all AnalogIn instrument parameters to default values. If auto configure is enabled (through **FDwfDeviceAutoConfigureSet**), the instrument is also configured.

Parameters:

- hdwf – Interface handle.

FDwfAnalogInConfigure(HDWF hdwf, BOOL fReconfigure, BOOL fStart)

Description: Configures the instrument and start or stop the acquisition. To reset the Auto trigger timeout, set fReconfigure to TRUE.

Parameters:

- hdwf – Interface handle.
- fReconfigure – Configure the device.
- fStart – Start the acquisition.

FDwfAnalogInStatus(HDWF hdwf, BOOL fReadData, DwfState *psts)

Description: Checks the state of the acquisition. To read the data from the device, set fReadData to TRUE. For single acquisition mode, the data will be read only when the acquisition is finished.

Parameters:

- hdwf – Interface handle.
- fReadData – TRUE if data should be read.
- psts – Variable to receive the acquisition state.

Note: To ensure consistency between device status and measured data, the following AnalogInStatus*functions do not communicate with the device. These functions only return information and data from the last FDwfAnalogInStatus call.

FDwfAnalogInStatusSamplesLeft(HDWF hdwf, int *pcSamplesLeft)

Description: Retrieves the number of samples left in the acquisition.

Parameters:

- hdwf – Interface handle.
- pcSamplesLeft – Variable to receive the remaining samples to acquire.

FDwfAnalogInStatusSamplesValid(HDWF hdwf, int *pcSamplesValid)

Description: Retrieves the number of valid/acquired data samples.

Parameters:

- hdwf – Interface handle.
- pcSamplesValid – Variable to receive the number of valid samples.

FDwfAnalogInStatusIndexWrite(HDWF hdwf, int *pidxWrite)

Description: Retrieves the buffer write pointer which is needed in ScanScreen acquisition mode to display the scan bar.

Parameters:

- hdwf – Interface handle.
- pidxWrite – Variable to receive the position of the acquisition.

FDwfAnalogInStatusAutoTriggered(HDWF hdwf, BOOL *pfAuto)

Description: Verifies if the acquisition is auto triggered.

Parameters:

- hdwf – Interface handle.
- pfAuto – Returns TRUE if the acquisition was auto triggered.

FDwfAnalogInStatusData(
HDWF hdwf, int idxChannel, double *rgdVoltData, int cdData)

Description: Retrieves the acquired data samples from the specified idxChannel on the AnalogIn instrument. It copies the data samples to the provided buffer.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- rgdVoltData – Pointer to allocated buffer to copy the acquisition data.
- cdData – Number of samples to copy.

FDwfAnalogInStatusData2(
HDWF hdwf, int idxChannel, double *rgdVoltData, int idxData, int cdData)

Description: Retrieves the acquired data samples from the specified idxChannel on the AnalogIn instrument. It copies the data samples to the provided buffer.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- rgdVoltData – Pointer to allocated buffer to copy the acquisition data.
- idxData – First sample index to copy.
- cdData – Number of samples to copy.

FDwfAnalogInStatusData16(
HDWF hdwf, int idxChannel, short*rgs16Data, int idxData, int cdData)

Description: Retrieves the acquired raw data samples from the specified idxChannel on the AnalogIn instrument. It copies the data samples to the provided buffer.

Example: To convert raw data voltage value:

rgs16Data[i] * FDwfAnalogInChannelRangeGet / 65536+FDwfAnalogInChannelOffsetGet

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- rgs16VoltData – Pointer to allocated buffer to copy the acquisition data.
- idxData – Source sample index to copy.
- cdData – Number of samples to copy.

FDwfAnalogInStatusNoise(
HDWF hdwf, int idxChannel, double *rgdMin, double *rgdMax, int cdData)

Description: Retrieves the acquired noise samples from the specified idxChannel on the AnalogIn instrument. It copies the data samples to the provided buffer.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- rgdMin – Pointer to allocated buffer to copy the minimum noise data.
- rgdMax – Pointer to allocated buffer to copy the maximum noise data.
- cdData – Number of min/max samples to copy.

FDwfAnalogInStatusSample(HDWF hdwf, int idxChannel, double *pdVoltSample)

Description: Gets the last ADC conversion sample from the specified idxChannel on the AnalogIn instrument.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- pdVoltSample – Variable to receive the sample value.

FDwfAnalogInStatusRecord(
HDWF hdwf, int *pcdDataAvailable, int *pcdDataLost, int *pcdDataCorrupt)

Description: Retrieves information about the recording process. The data loss occurs when the device acquisition is faster than the read process to PC. In this case, the device recording buffer is filled and data samples are overwritten. Corrupt samples indicate that the samples have been overwritten by the acquisition process during the previous read. In this case, try optimizing the loop process for faster execution or reduce the acquisition frequency or record length to be less than or equal to the device buffer size (record length <= buffer size/frequency).

Parameters:

- hdwf – Interface handle.
- pcdDataAvailable – Pointer to variable to receive the available number of samples.
- pcdDataLost – Pointer to variable to receive the lost samples after the last check.
- pcdDataCorrupt – Pointer to variable to receive the number of samples that could be corrupt.

FDwfAnalogInRecordLengthSet(HDWF hdwf, double sLegth)

Description: Sets the Record length in seconds. With length of zero, the record will run indefinitely.

Parameters:

- hdwf – Interface handle.
- sLegth – Record length to set expressed in seconds.

FDwfAnalogInRecordLengthGet(HDWF hdwf, double *psLegth)

Description: Gets the current Record length in seconds.

Parameters:

- hdwf – Interface handle.
- sLegth – Pointer to variable to receive the record length.

5.2 Configuration

FDwfAnalogInFrequencyInfo(HDWF hdwf, double *phzMin, double *phzMax)

Description: Retrieves the minimum and maximum (ADC frequency) settable sample frequency.
Parameters:
- hdwf – Interface handle.
- phzMin – Pointer to return the minimum allowed frequency.
- phzMax – Pointer to return the maximum allowed frequency.

FDwfAnalogInFrequencySet(HDWF hdwf, double hzFrequency)

Description: Sets the sample frequency for the instrument.
Parameters:
- hdwf – Interface handle.
- hzFrequency – Acquisition frequency to set.

FDwfAnalogInFrequencyGet(HDWF hdwf, double *phzFrequency)

Description: Reads the configured sample frequency. The AnalogIn ADC always runs at maximum frequency, but the method in which the samples are stored in the buffer can be individually configured for each channel with `FDwfAnalogInChannelFilterSet` function.
Parameters:
- hdwf – Interface handle.
- phzFrequency – Variable to receive the acquisition frequency.

FDwfAnalogInBitsInfo(HDWF hdwf, int *pnBits)

Description: Retrieves the number bits used by the AnalogIn ADC.
Parameters:
- hdwf – Interface handle.
- pnBits – Variable to receive the number of ADC bits.

FDwfAnalogInBufferSizeInfo(HDWF hdwf, int *pnSizeMin, int *pnSizeMax)

Description: Returns the minimum and maximum allowable buffer sizes for the instrument.

Parameters:

- hdwf – Interface handle.
- pnMin – Pointer to return the minimum buffer size.
- pnMax – Pointer to return the maximum buffer size.

FDwfAnalogInBufferSizeSet(HDWF hdwf, int nSize)

Description: Adjusts the AnalogIn instrument buffer size.

Parameters:

- hdwf – Interface handle.
- nSize – Buffer size to set.

FDwfAnalogInBufferSizeGet(HDWF hdwf, int *pnSize)

Description: Returns the used AnalogIn instrument buffer size.

Parameters:

- hdwf – Interface handle.
- pnSize – Variable to receive the current buffer size.

FDwfAnalogInNoiseSizeInfo(HDWF hdwf, int *pnSizeMax)

Description: Returns the maximum buffer size for the instrument.

Parameters:

- hdwf – Interface handle.
- pnMax – Pointer to return the maximum noise buffer size.

FDwfAnalogInNoiseSizeGet(HDWF hdwf, int *pnSize)

Description: Returns the used AnalogIn instrument noise buffer size. This is automatically adjusted according to the sample buffer size. For instance, having maximum buffer size of 8192 and noise buffer size of 512, setting the sample buffer size to 4096 the noise buffer size will be 256.

Parameters:

- hdwf – Interface handle.
- pnSize – Variable to receive the current noise buffer size.

FDwfAnalogInAcquisitionModeInfo(HDWF hdwf, int *pfsacqmode)

Description: Returns the supported AnalogIn acquisition modes. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the ACQMODE constants in dwf.h. The acquisition mode selects one of the following modes, ACQMODE:

ACQMODE	int	Constant Capabilities
acqmodeSingle	0	Perform a single buffer acquisition. This is the default setting.
acqmodeScanShift	1	Perform a continuous acquisition in FIFO style. The trigger setting is ignored. The last sample is at the end of buffer. The FDwfAnalogInStatusSamplesValid function is used to show the number of the acquired samples, which will grow until reaching the BufferSize. Then the waveform "picture" is shifted for every new sample.
acqmodeScanScreen	2	Perform continuous acquisition circularly writing samples into the buffer. The trigger setting is ignored. The IndexWrite shows the buffer write position. This is similar to a heart monitor display.
acqmodeRecord	3	Perform acquisition for length of time set by FDwfAnalogInRecordLengthSet.

Parameters:

- hdwf – Interface handle.
- pfsacqmode – Pointer to return the supported acquisition modes.

FDwfAnalogInAcquisitionModeSet(HDWF hdwf, ACQMODE acqmode)

Description: Sets the acquisition mode.

Parameters:

- hdwf – Interface handle.
- acqmode – Acquisition mode to set.

FDwfAnalogInAcquisitionModeGet(HDWF hdwf, ACQMODE *pacqmode)

Description: Retrieves the acquisition mode.
Parameters:
- hdwf – Interface handle.
- pacqmode – Variable to receive the current acquisition mode.

FDwfAnalogInSamplingSourceSet(HDWF hdwf, TRIGSRC trigsrc)

Description: Configures the AnalogIn acquisition data sampling source.
Parameters:
- hdwf – Interface handle.
- trigsrc – Trigger source to be used as data sampling clock.

FDwfAnalogInSamplingSourceGet(HDWF hdwf, TRIGSRC *ptrigsrc)

Description: Returns the configured sampling source.
Parameters:
- hdwf – Interface handle.
- ptrigsrc – Variable to receive the current sampling source.

FDwfAnalogInSamplingSlopeSet(HDWF hdwf, **Error! Reference source not found.** slope)

Description: Sets the sampling slope for the instrument.
Parameters:
- hdwf – Interface handle.
- slope – Sampling clock slope to set.

FDwfAnalogInSamplingSlopeGet(HDWF hdwf, DwfTriggerSlope *pslope)

Description: Returns the sampling for the instrument.
Parameters:
- hdwf – Interface handle.
- pslope – Variable to receive the current sampling slope.

FDwfAnalogInSamplingDelaySet(HDWF hdwf, double sec)

Description: Sets the sampling delay for the instrument.
Parameters:
- hdwf – Interface handle.
- hzFrequency – Sampling delay to set.

FDwfAnalogInSamplingDelayGet(HDWF hdwf, double *psec)

Description: Returns the configured sampling delay.
Parameters:
- hdwf – Interface handle.
- phzFrequency – Variable to receive the sampling delay.

5.3 Channels

The oscilloscope channel settings are identical across all channels.

FDwfAnalogInChannelCount(HDWF hdwf, int *pcChannel)

Description: Reads the number of AnalogIn channels of the device.
Parameters:
- hdwf – Interface handle.
- pcChannel – Variable to receive the number of channels.

FDwfAnalogInChannelEnableSet(HDWF hdwf, int idxChannel, BOOL fEnable)

Description: Enables or disables the specified AnalogIn channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Zero based index of channel to enable/disable.
- fEnable – Set TRUE to enable, FALSE to disable.

FDwfAnalogInChannelEnableGet(HDWF hdwf, int idxChannel, BOOL *pfEnable)

Description: Gets the current enable/disable status of the specified AnalogIn channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Index of channel.
- pfEnable – Variable to return enable/disable status of channel.

FILTER	int	Constant Capabilities
filterDecimate	0	Store every Nth ADC conversion, where N = ADC frequency /acquisition frequency.
filterAverage	1	Store the average of N ADC conversions.
filterMinMax	2	Store interleaved, the minimum and maximum values, of 2xN conversions.

FDwfAnalogInChannelFilterInfo(HDWF hdwf, int *pfsfilter)

Description: Returns the supported acquisition filters. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the FILTER constants in dwf.h. When the acquisition frequency (FDwfAnalogInFrequencySet) is less than the ADC frequency (maximum acquisition frequency).
Parameters:
- hdwf – Interface handle.
- pfsfilter – Pointer to return the supported acquisition modes.

FDwfAnalogInChannelFilterSet(HDWF hdwf, int idxChannel, FILTER filter)

Description: Sets the acquisition filter for each AnalogIn channel. With channel index -1, each enabled AnalogIn channel filter will be configured to use the same, new option.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- filter – Acquisition sample filter to set.

FDwfAnalogInChannelFilterGet(HDWF hdwf, int idxChannel, FILTER *pfilter)

Description: Returns the configured acquisition filter.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pfilter– Variable to receive the current sample filter.

FDwfAnalogInChannelRangeInfo(
HDWF hdwf, double *pvoltsMin, double *pvoltsMax, double *pnSteps)

Description: Returns the minimum and maximum range, peak to peak values, and the number of adjustable steps.
Parameters:
- hdwf – Interface handle.
- pvoltsMin – Variable to receive the minimum voltage range.
- pvoltsMax – Variable to receive the maximum voltage range.
- pnSteps – Variable to receive number of steps.

FDwfAnalogInChannelRangeSteps(
HDWF hdwf, double rgVoltsStep[32], int *pnSteps)

Description: Reads the range of steps supported by the device. For instance: 1, 2, 5, 10, etc.
Parameters:
- hdwf – Interface handle.
- rgVoltsStep – Pointer to buffer to receive the range steps.
- pnSteps – Variable to receive number range steps.

FDwfAnalogInChannelRangeSet(HDWF hdwf, int idxChannel, double voltsRange)

Description: Configures the range for each channel. With channel index -1, each enabled Analog In channel range will be configured to the same, new value.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- voltsRange – Voltage range to set.

FDwfAnalogInChannelRangeGet(HDWF hdwf, int idxChannel, double *pvoltsRange)

Description: Returns the real range value for the given channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pvoltsRange – Variable to receive the current voltage range.

FDwfAnalogInChannelOffsetInfo(
HDWF hdwf, double *pvoltsMin, double *pvoltsMax, double *pnSteps)

Description: Returns the minimum and maximum offset levels supported, and the number of adjustable steps.
Parameters:
- hdwf – Interface handle.
- pvoltsMin – Variable to receive the minimum offset voltage.
- pvoltsMax – Variable to receive the maximum offset voltage.
- pnSteps – Variable to receive the number offset steps.

FDwfAnalogInChannelOffsetSet(HDWF hdwf, int idxChannel, double voltOffset)

Description: Configures the offset for each channel. When channel index is specified as -1, each enabled AnalogIn channel offset will be configured to the same level.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- voltsRange – Channel offset voltage to set.

FDwfAnalogInChannelOffsetGet(HDWF hdwf, int idxChannel, double *pvoltOffset)

Description: Returns for each AnalogIn channel the real offset level.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pvoltsRange – Variable to receive the offset voltage obtained.

FDwfAnalogInChannelAttenuationSet(
HDWF hdwf, int idxChannel, double xAttenuation)

Description: Configures the attenuation for each channel. When channel index is specified as -1, each enabled AnalogIn channel attenuation will be configured to the same level. The attenuation does not change the attenuation on the device, just informs the library about the externally applied attenuation.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- voltsRange – Channel offset voltage to set.

FDwfAnalogInChannelAttenuationGet(
HDWF hdwf, int idxChannel, double *pxAttenuation)

Description: Returns for each AnalogIn channel the configured attenuation.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pxAttenuation – Variable to receive the attenuation value.

5.4 Trigger

The trigger is used for Single and Record acquisitions. For ScanScreen and ScanShift, the trigger is ignored.
To achieve the classical trigger types:
- **None**: Set *FDwfAnalogInTriggerSourceSet* to *trigsrcNone.*
- **Auto***:* Set *FDwfAnalogInTriggerSourceSet* to something other than *trigsrcNone,* such as trigsrcDetectorAnalogIn and *FDwfAnalogInTriggerAutoTimeoutSet* to other than zero.
- **Normal**: Set *FDwfAnalogInTriggerSourceSet* to something other than *trigsrcNone,* such as *trigsrcDetectorAnalogIn* or *FDwfAnalogInTriggerAutoTimeoutSet* to zero.

See the description of **FDwfDeviceTriggerInfo**.

FDwfAnalogInTriggerSourceSet(HDWF hdwf, TRIGSRC trigsrc)

Description: Configures the AnalogIn acquisition trigger source.
Parameters:
- hdwf – Interface handle.
- trigsrc – Trigger source to set.

FDwfAnalogInTriggerSourceGet(HDWF hdwf, TRIGSRC *ptrigsrc)

Description: Returns the configured trigger source. The trigger source can be “none” or an internal instrument or external trigger. To use the trigger on AnalogIn channels (edge, pulse, etc.), use trigsrcDetectorAnalogIn.
Parameters:
- hdwf – Interface handle.
- ptrigsrc – Variable to receive the current trigger source.

FDwfAnalogInTriggerPositionInfo(
HDWF hdwf, double *psecMin, double *psecMax, double *pnSteps)

Description: Returns the minimum and maximum values of the trigger position in seconds.
For Single/Repeated acquisition mode the horizontal trigger position is used is relative to the buffer middle point.
For Record mode the position is relative to the start of the capture.

Parameters:

- hdwf – Interface handle.
- psecMin – Variable to receive the minimum trigger position.
- psecMax – Variable to receive the maximum trigger position.
- pnSteps – Variable to return the number of steps.

FDwfAnalogInTriggerPositionSet(HDWF hdwf, double secPosition)

Description: Configures the horizontal trigger position in seconds.

Parameters:

- hdwf – Interface handle.
- secPosition – Trigger position to set.

FDwfAnalogInTriggerPositionGet(HDWF hdwf, double *psecPosition)

Description: Returns the configured trigger position in seconds.

Parameters:

- hdwf – Interface handle.
- psecPosition – Variable to receive the current trigger position.

FDwfAnalogInTriggerAutoTimeoutInfo(
HDWF hdwf, double *psecMin, double *psecMax, int *pnSteps)

Description: Returns the minimum and maximum auto trigger timeout values, and the number of adjustable steps.
The acquisition is auto triggered when the specified time elapses. With zero value the timeout is disabled, performing “Normal” acquisitions.

Parameters:

- hdwf – Interface handle.
- psecMin – Variable to receive the minimum timeout.
- psecMax – Variable to receive the maximum timeout.
- pnSteps – Variable to return the number of steps.

FDwfAnalogInTriggerAutoTimeoutSet(HDWF hdwf, double secTimeout)

Description: Configures the auto trigger timeout value in seconds.

Parameters:

- hdwf – Interface handle.
- secTimeout – Timeout to set.

FDwfAnalogInTriggerAutoTimeoutGet(HDWF hdwf, double *psecTimeout)

Description: Returns the configured auto trigger timeout value in seconds.

Parameters:

- hdwf – Interface handle.
- psecTimeout – Variable to receive the current timeout.

FDwfAnalogInTriggerHoldOffInfo(
HDWF hdwf, double *psecMin, double *psecMax, double *pnStep)

Description: Returns the supported range of the trigger Hold-Off time in Seconds. The trigger hold-off is an adjustable period of time during which the acquisition will not trigger. This feature is used when you are triggering on burst waveform shapes, so the oscilloscope triggers only on the first eligible trigger point.

Parameters:

- hdwf – Interface handle.
- psecMin – Variable to receive the minimum hold off value.
- psecMax – Variable to receive the maximum hold off value.

FDwfAnalogInTriggerHoldOffSet(HDWF hdwf, double secHoldOff)

Description: Sets the trigger hold-off for the AnalongIn instrument in Seconds.

Parameters:

- hdwf – Interface handle.
- secHoldOff – Holdoff to set.

FDwfAnalogInTriggerHoldOffGet(HDWF hdwf, double *psecHoldOff)

Description: Gets the current trigger hold-off for the AnalongIn instrument in Seconds.

Parameters:

- hdwf – Interface handle.
- psecHoldOff – Variable to receive the current holdoff value.

5.5 Trigger Detector

The following functions configure the trigger detector on analog in channels. To use this, set trigger source with FDwfAnalogInTriggerSourceSet to trigsrcDetectorAnalogIn.
See the AnalogIn_Trigger.py example.

FDwfAnalogInTriggerTypeInfo(HDWF hdwf, int *pfstrigtype)

Description: Returns the supported trigger type options for the instrument. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the TRIGTYPE constants in dwf.h. These trigger type options are:

- trigtypeEdge: trigger on rising or falling edge. This is the default setting.
- trigtypePulse: trigger on positive or negative; less, timeout, or more pulse lengths.
- trigtypeTransition: trigger on rising or falling; less, timeout, or more transition times.

Parameters:

- hdwf – Interface handle.
- pfstrigtype – Variable to receive the supported trigger types.

FDwfAnalogInTriggerTypeSet(HDWF hdwf, TRIGTYPE trigtype)

Description: Sets the trigger type for the instrument.

Parameters:

- hdwf – Interface handle.
- trigtype – Trigger type to set.

FDwfAnalogInTriggerTypeGet(HDWF hdwf, TRIGTYPE *ptrigtype)

Description: Gets the current trigger type for the instrument.

Parameters:

- hdwf – Interface handle.
- ptrigtype – Variable to receive the current trigger type.

FDwfAnalogInTriggerChannelInfo(HDWF hdwf, int *pidxMin, int *pidxMax)

Description: Returns the range of channels that can be triggered on.

Parameters:

- hdwf – Interface handle.
- pidxMin – Variable to receive the minimum channel index.
- pidxMax – Variable to receive the maximum channel index.

FDwfAnalogInTriggerChannelSet(HDWF hdwf, int idxChannel)

Description: Sets the trigger channel.

Parameters:

- hdwf – Interface handle.
- idxChannel – Trigger channel index to set.

FDwfAnalogInTriggerChannelGet(HDWF hdwf, int *pidxChannel)

Description: Retrieves the current trigger channel index.

Parameters:

- hdwf – Interface handle.
- pidxChannel – Variable to receive the current trigger channel index.

FDwfAnalogInTriggerFilterInfo(HDWF hdwf, int *pfsfilter)

Description: Returns the supported trigger filters. They are returned (by reference) as a bit field which can be parsed using the IsBitSet Macro. Individual bits are defined using the FILTER constants in DWF.h. Select trigger detector sample source, FILTER:

- filterDecimate: Looks for trigger in each ADC conversion, can detect glitches.
- filterAverage: Looks for trigger only in average of N samples, given by FDwfAnalogInFrequencySet.

Parameters:

- hdwf – Interface handle.
- pfsFilter – Variable to receive the supported trigger filters.

FDwfAnalogInTriggerFilterSet(HDWF hdwf, FILTER filter)

Description: Sets the trigger filter.

Parameters:

- hdwf – Interface handle.
- filter – Trigger filter to set.

FDwfAnalogInTriggerFilterGet(HDWF hdwf, FILTER *pfilter)

Description: Gets the trigger filter.

Parameters:

- hdwf – Interface handle.
- pfilter – Variable to receive the current trigger filter.

FDwfAnalogInTriggerConditionInfo(HDWF hdwf, int *pfstrigcond)

Description: Returns the supported trigger type options for the instrument. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the DwfTriggerSlope constants in dwf.h. These trigger condition options are:

- DwfTriggerSlopeRise (This is the default setting):
  - For edge and transition trigger on rising edge.
  - For pulse trigger on positive pulse.
- DwfTriggerSlopeFall:
  - For edge and transition trigger on falling edge.
  - For pulse trigger on negative pulse.
- DwfTriggerSlopeEither:
  - For edge and transition trigger on either edge.
  - For pulse trigger on either positive or negative pulse.

Parameters:

- hdwf – Interface handle.
- pfstrigcond – Variable to receive the supported trigger conditions.
  -

FDwfAnalogInTriggerConditionSet(HDWF hdwf, **Error! Reference source not found.** trigcond)

Description: Sets the trigger condition for the instrument.

Parameters:

- hdwf – Interface handle.
- trigcond – Trigger condition to set.

FDwfAnalogInTriggerConditionGet(HDWF hdwf, **Error! Reference source not found.** * ptrigcond)

Description: Sets the trigger condition for the instrument.
Parameters:
- hdwf – Interface handle.
- ptrigcond – Variable to receive the current trigger condition.

FDwfAnalogInTriggerLevelInfo(
HDWF hdwf, double *pvoltsMin, double *pvoltsMax, int *pnSteps)

Description: Retrieves the range of valid trigger voltage levels for the AnalogIn instrument in Volts.
Parameters:
- hdwf – Interface handle.
- pvoltsMin – Variable to receive the minimum voltage level.
- pvoltsMax – Variable to receive the maximum voltage level.
- pnSteps – Variable to receive the number of voltage level steps.

FDwfAnalogInTriggerLevelSet(HDWF hdwf, double voltsLevel)

Description: Sets the trigger voltage level in Volts.
Parameters:
- hdwf – Interface handle.
- voltsLevel – Trigger voltage level to set.

FDwfAnalogInTriggerLevelGet(HDWF hdwf, double *pvoltsLevel)

Description: Gets the current trigger voltage level in Volts.
Parameters:
- hdwf – Interface handle.
- pvoltsLevel – Variable to receive the current trigger voltage level.

FDwfAnalogInTriggerHysteresisInfo(
HDWF hdwf, double *pvoltsMin, double *pvoltsMax, int *pnSteps)

Description: Retrieves the range of valid trigger hysteresis voltage levels for the AnalogIn instrument in Volts. The trigger detector uses two levels: low level (TriggerLevel - Hysteresis) and high level (TriggerLevel + Hysteresis). Trigger hysteresis can be used to filter noise for Edge or Pulse trigger. The low and high levels are used in transition time triggering.

Parameters:

- hdwf – Interface handle.
- pvoltsMin – Variable to receive the minimum hysteresis level.
- pvoltsMax – Variable to receive the maximum hysteresis level.
- pnSteps – Variable to receive the number of hysteresis level steps.

FDwfAnalogInTriggerHysteresisSet(HDWF hdwf, double voltsLevel)

Description: Sets the trigger hysteresis level in Volts.

Parameters:

- hdwf – Interface handle.
- voltsLevel – Trigger hysteresis level to set.

FDwfAnalogInTriggerHysteresisGet(HDWF hdwf, double *pvoltsHysteresis)

Description: Gets the current trigger hysteresis level in Volts.

Parameters:

- hdwf – Interface handle.
- pvoltsLevel – Variable to receive the current trigger hysteresis level.

FDwfAnalogInTriggerLengthConditionInfo(HDWF hdwf, int *pfstriglen)

Description: Returns the supported trigger length condition options for the AnalogIn instrument. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the TRIGLEN constants in DWF.h. These trigger length condition options are:

- triglenLess: Trigger immediately when a shorter pulse or transition time is detected.
- triglenTimeout: Trigger immediately as the pulse length or transition time is reached.
- triglenMore: Trigger when the length/time is reached, and pulse or transition has ended.

Parameters:

- hdwf – Interface handle.
- pfsstriglen – Variable to receive the supported trigger length conditions.

FDwfAnalogInTriggerLengthConditionSet(HDWF hdwf, TRIGLEN triglen)

Description: Sets the trigger length condition for the AnalongIn instrument.
Parameters:
- hdwf – Interface handle.
- triglen – Trigger length condition to set.

FDwfAnalogInTriggerLengthConditionGet(HDWF hdwf, TRIGLEN *ptriglen)

Description: Gets the current trigger length condition for the AnalongIn instrument.
Parameters:
- hdwf – Interface handle.
- ptriglen – Variable to receive the current trigger length condition.

FDwfAnalogInTriggerLengthInfo(
HDWF hdwf, double *psecMin, double *psecMax, double *pnStep)

Description: Returns the supported range of trigger length for the instrument in Seconds. The trigger length specifies the minimal or maximal pulse length or transition time.
Parameters:
- hdwf – Interface handle.
- psecMin – Variable to receive the minimum trigger length.
- psecMax – Variable to receive the maximum trigger length.

FDwfAnalogInTriggerLengthSet(HDWF hdwf, double secLength)

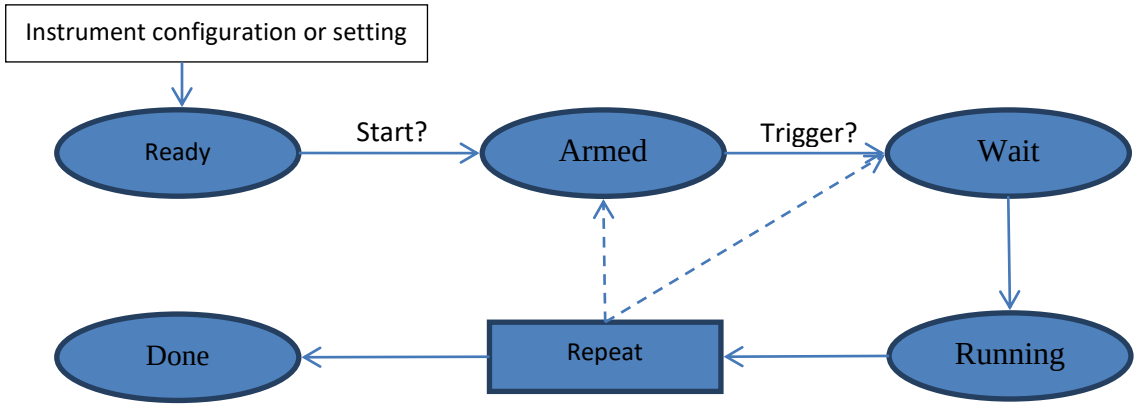
Description: Sets the trigger length in Seconds.
Parameters:
- hdwf – Interface handle.
- secLength – Trigger length to set.

FDwfAnalogInTriggerLengthGet(HDWF hdwf, double *psecLength)

Description: Gets the current trigger length in Seconds.
Parameters:
- hdwf – Interface handle.
- secLength – Variable to receive the current trigger length.

6 Analog Out (Arbitrary Waveform Generator)

The Analog Out instrument states:

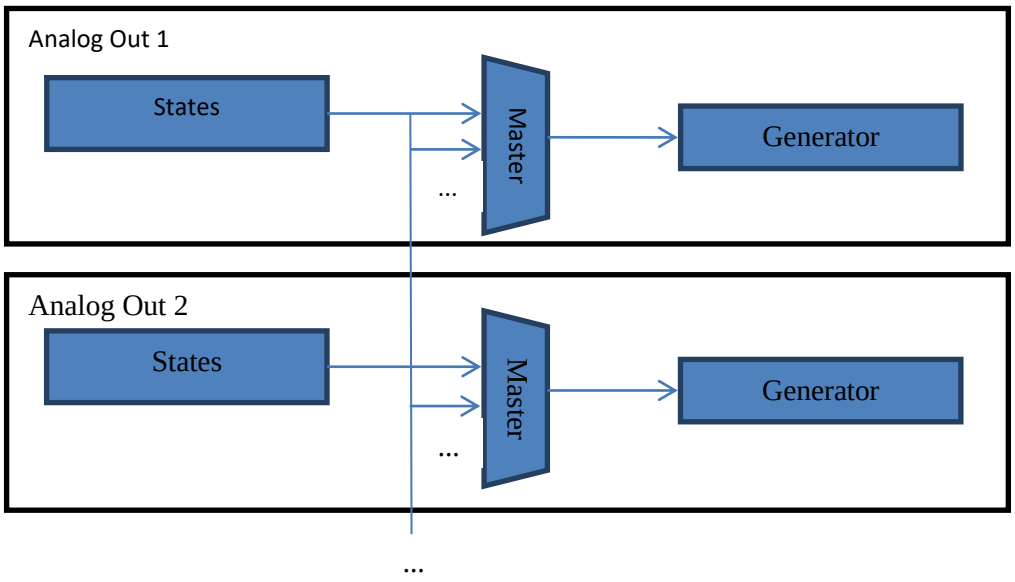


The states are defined in dwf.h DwfState type.

- **Ready**: Initial state. After *FDwfAnalogOutConfigure* or any *FDwfAnalogOut*Set* function call goes to this state. With digital out, configure start command goes to Armed state.
- **Armed**: It waits for trigger.
- **Wait**: Remains in this state for the time period specified by *FDwfAnalogOutWaitSet* function.
- **Running**: Remains in this state for the time period specified by *FDwfAnalogOutRunSet* function.
- **Repeat**: Goes to Armed or Wait state according to the FDwfAnalogOutRepeatTriggerSet setting for the number of times specified by FDwfAnalogOutRepeatSet.
- **Done**: Final state.

The analog out channels can run independently or synchronized using the master parameter. The states are defined by trigger, wait, run, and repeat options. It is enough to start with FDwfAnalogOutConfigure (the master channel) the slave channels will also start.

See the following examples: AnalogOut_Custom/Pattern/Play/Sine/Sweep/Sync.py AnalogOutIn.py



6.1 Control

FDwfAnalogOutReset(HDWF hdwf, int idxChannel)

Description: Resets and configures (by default, having auto configure enabled) all AnalogOut instrument parameters to default values for the specified channel. To reset instrument parameters across all channels, set idxChannel to -1.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.

FDwfAnalogOutConfigure(HDWF hdwf, int idxChannel, BOOL fStart)

Description: Starts or stops the instrument. Value 3 will apply the configuration dynamically without changing the state of the instrument. With channel index -1, each enabled Analog Out channel will be configured.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- fStart – Start the instrument: 0 stop, 1 start, 3 apply.

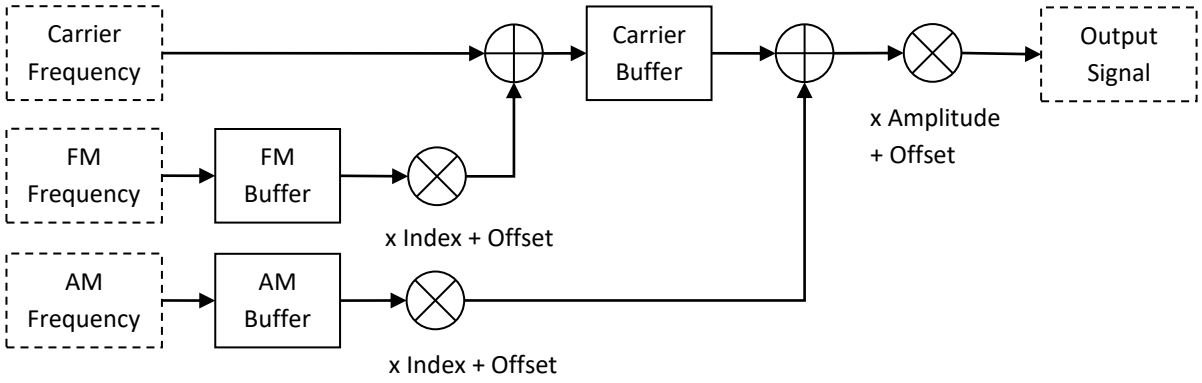
FDwfAnalogOutStatus(HDWF hdwf, int idxChannel, DwfState *psts)

Description: Checks the state of the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- psts – Pointer to variable to return the state.

The device output signal is output from carrier buffer but can also be frequency and/or amplitude modulated.



The Carrier/FM/AM nodes can be configured by selecting with the AnalogOutNode constants.

AnalogOutNode	int	
AnalogOutNodeCarrier	0	Carrier signal
AnalogOutNodeFM	1	Frequency modulation
AnalogOutNodeAM	2	Amplitude modulation

FDwfAnalogOutNodePlayStatus(HDWF hdwf, int idxChannel, AnalogOutNode node, int *cdDataFree, int *cdDataLost, int *cdDataCorrupted)

Description: Retrieves information about the play process. The data lost occurs when the device generator is faster than the sample send process from the PC. In this case, the device buffer gets emptied and generated samples are repeated. Corrupt samples are a warning that the buffer might have been emptied while samples were sent to the device. In this case, try optimizing the loop for faster execution; or reduce the frequency or run time to be less or equal to the device buffer size (run time <= buffer size/frequency).

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- cdDataFree – Pointer to variable to return the available free buffer space, the number of new samples that can be sent.
- cdDataLost – Pointer to variable to return the number of lost samples.
- cdDataCorrupted – Pointer to variable to return the number of samples that could be corrupted.

FDwfAnalogOutNodePlayData(
HDWF hdwf, int idxChannel, AnalogOutNode node, double *rgdData, int cdData)

Description: Sends new data samples for play mode. Before starting the Analog Out instrument, prefill the device buffer with the first set of samples using the AnalogOutNodeDataSet function. In the loop of sending the following samples, first call AnalogOutStatus to read the information from the device, then AnalogOutPlayStatus to find out how many new samples can be sent, then send the samples with AnalogOutPlayData.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- rgdData – Pointer to samples array to be sent to the device.
- cdData – Number of samples to send.

6.2 Configuration

FDwfAnalogOutCount(HDWF hdwf, int *pcChannel)

Description: Returns the number of Analog Out channels by the device specified by hdwf.
Parameters:
- hdwf – Open interface handle on a device.
- pcChannel – Pointer to variable to receive the number of channels in the instrument.

FDwfAnalogOutNodeInfo(HDWF hdwf, int idxChannel, int *pfsnode)

Description: Returns the supported AnalogOut nodes of the AnalogOut channel. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pfsnode – Variable to receive the supported nodes.

FDwfAnalogOutNodeEnableSet(
HDWF hdwf, int idxChannel, AnalogOutNode node, BOOL fEnable)

Description: Enables or disables the channel node specified by idxChannel and node. The Carrier node enables or disables the channel and AM/FM the modulation. With channel index -1, each Analog Out channel enable will be configured to use the same, new option.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- fEnable – TRUE to enable, FALSE to disable.

FDwfAnalogOutNodeEnableGet(
HDWF hdwf, int idxChannel, AnalogOutNode node, BOOL *pfEnable)

Description: Verifies if a specific channel and node is enabled or disabled.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pfEnable – Pointer to variable to receive enabled state.

FDwfAnalogOutNodeFunctionInfo(
HDWF hdwf, int idxChannel, AnalogOutNode node, int *pfsfunc)

Description: Returns the supported generator function options. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the FUNC constants in dwf.h. These are:

FUNC Constants	FUNC Constant Capabilities
funcDC	Generate DC value set as offset.
funcSine	Generate sine waveform.
funcSquare	Generate square waveform.
funcTriangle	Generate triangle waveform.
funcRampUp	Generate a waveform with a ramp-up voltage at the beginning.
funcRampDown	Generate a waveform with a ramp-down voltage at the end.
funcNoise	Generate noise waveform from random samples.
funcCustom	Generate waveform from custom repeated data.
funcPlay	Generate waveform from custom data in stream play style.

Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pfsfunc – Variable to receive the supported generator function options.

FDwfAnalogOutNodeFunctionSet(
HDWF hdwf, int idxChannel, AnalogOutNode node, FUNC func)

Description: Sets the generator output function for the specified instrument channel. With channel index -1, each enabled Analog Out channel function will be configured to use the same, new option.

Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- func – Generator function option to set.

FDwfAnalogOutNodeFunctionGet(
HDWF hdwf, int idxChannel, AnalogOutNode node, FUNC *pfunc)

Description: Retrieves the current generator function option for the specified instrument channel.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- ptrigsrc – Pointer to variable to receive the generator function option.

FDwfAnalogOutNodeFrequencyInfo(HDWF hdwf, int idxChannel, AnalogOutNode node, double *phzMin, double *phzMax)

Description: Returns the supported frequency range for the instrument. The maximum value shows the DAC frequency. The frequency of the generated waveform: repetition frequency for standard types and custom data; DAC update for noise type; sample rate for play type.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Zero based channel index.
- node – Zero based node index.
- phzMin – Variable to receive the supported minimum frequency.
- phzMax – Variable to receive the supported maximum frequency.

FDwfAnalogOutNodeFrequencySet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double hzFrequency)

Description: Sets the frequency. With channel index -1, each enabled Analog Out channel frequency will be configured to use the same, new option.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- hzFrequency – Frequency value to set expressed in Hz.

FDwfAnalogOutNodeFrequencyGet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double *phzFrequency)

Description: Gets the currently set frequency for the specified channel-node on the instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- hzFrequency – Pointer to variable to receive frequency value in Hz.

FDwfAnalogOutNodeAmplitudeInfo(
HDWF hdwf, int idxChannel, AnalogOutNode node, double *pvMin, double *pvMax)

Description: Retrieves the amplitude range for the specified channel-node on the instrument. The amplitude is expressed in Volt units for carrier and in percentage units (modulation index) for AM/FM.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pvMin – Minimum amplitude level or modulation index.
- pvMax – Maximal amplitude level or modulation index.

FDwfAnalogOutNodeAmplitudeSet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double vAmplitude)

Description: Sets the amplitude or modulation index for the specified channel-node on the instrument. With channel index -1, each enabled Analog Out channel amplitude (or modulation index) will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- vAmplitude – Amplitude of channel in Volts or modulation index in percentage.

FDwfAnalogOutNodeAmplitudeGet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double *pvAmplitude)

Description: Gets the currently set amplitude or modulation index for the specified channel-node on the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pvAmplitude – Pointer to variable to receive amplitude value in Volts or modulation index in percentage.

FDwfAnalogOutNodeOffsetInfo(
HDWF hdwf, int idxChannel, AnalogOutNode node, double *pvMin, double *pvMax)

Description: Retrieves available the offset range. For carrier node in units of volts, and in percentage units for AM/FM nodes.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pvMin – Minimum offset voltage or modulation offset percentage.
- pvMax – Maximum offset voltage or modulation offset percentage.

FDwfAnalogOutNodeOffsetSet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double vOffset)

Description: Sets the offset value for the specified channel-node on the instrument. With channel index -1, each enabled Analog Out channel offset will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- vOffset – Value to set voltage offset in Volts or modulation offset percentage.

FDwfAnalogOutNodeOffsetGet(HDWF hdwf, int idxChannel, AnalogOutNode node, double *pvOffset)

Description: Gets the current offset value for the specified channel-node on the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pvOffset – Pointer to variable to receive offset value in Volts or modulation offset percentage.

FDwfAnalogOutNodeSymmetryInfo(HDWF hdwf, int idxChannel, AnalogOutNode node, double *ppercentageMin, double *ppercentageMax)

Description: Obtains the symmetry (or duty cycle) range (0 … 100). This symmetry is supported for standard signal types. It the pulse duration divided by the pulse period.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- ppercentageMin – Minimum value of Symmetry percentage.
- ppercentageMax – Maximum value of Symmetry percentage.

FDwfAnalogOutNodeSymmetrySet(HDWF hdwf, int idxChannel, AnalogOutNode node, double percentageSymmetry)

Description: Sets the symmetry (or duty cycle) for the specified channel-node on the instrument. With channel index -1, each enabled Analog Out channel symmetry will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- percentageSymmetry –Value of percentage of Symmetry (duty cycle).

FDwfAnalogOutNodeSymmetryGet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double *ppercentageSymmetry)

Description: Gets the currently set symmetry (or duty cycle) for the specified channel-node of the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- ppercentageSymmetry – Pointer to variable to receive value of Symmetry (duty cycle).

FDwfAnalogOutNodePhaseInfo(HDWF hdwf, int idxChannel, AnalogOutNode node, double *pdegreeMin, double *pdegreeMax)

Description: Retrieves the phase range (in degrees 0 ... 360) for the specified channel-node of the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pdegreeMin – Minimum value of Phase (in degrees).
- pdegreeMax – Maximum value of Phase (in degrees).

FDwfAnalogOutNodePhaseSet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double degreePhase)

Description: Sets the phase for the specified channel-node on the instrument. With channel index -1, each enabled Analog Out channel phase will be configured to use the same, new option.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- degreePhase – Value of Phase in degrees.

FDwfAnalogOutNodePhaseGet(HDWF hdwf, int idxChannel, AnalogOutNode node, double *pdegreePhase)

Description: Gets the current phase for the specified channel-node on the instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pdegreePhase – Pointer to variable to receive Phase value (in degrees).

FDwfAnalogOutNodeDataInfo(HDWF hdwf, int idxChannel, AnalogOutNode node, int *pnSamplesMin, double *pnSamplesMax)

Description: Retrieves the minimum and maximum number of samples allowed for custom data generation.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- pnSamplesMin – Minimum number of samples available for custom data.
- pnSamplesMax – Maximum number of samples available for custom data.

FDwfAnalogOutNodeDataSet(
HDWF hdwf, int idxChannel, AnalogOutNode node, double *rgdData, int cdData)

Description: Set the custom data or to prefill the buffer with play samples. The samples are double precision floating point values (rgdData) normalized to ±1.
With the custom function option, the data samples (cdData) will be interpolated to the device buffer size. The output value will be Offset + Sample*Amplitude, for instance:

- 0 value sample will output: Offset.
- +1 value sample will output: Offset + Amplitude.
- -1 value sample will output: Offset – Amplitude.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- node – Node index.
- rgbData – Buffer of samples to set.
- cData – Number of samples to set in rgbData.

-

FDwfAnalogOutLimitationInfo(
HDWF hdwf, int idxChannel, double *pvMin, double *pvMax)

Description: Retrieves the limitation range supported by the channel. This option is supported on Electronics Explorer Analog Out Channel 3 and 4, Positive and Negative Power supplies, to set current or voltage limitation.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pvMin – Minimum limitation value.
- pvMax – Maximum offset voltage or modulation offset percentage.

FDwfAnalogOutLimitationSet(HDWF hdwf, int idxChannel, double limit)

Description: Sets the limitation value for the specified channel on the instrument. With channel index -1, each enabled Analog Out channel limitation will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- limit – Value to set voltage offset in Volts or modulation offset percentage.

FDwfAnalogOutLimitationGet(HDWF hdwf, int idxChannel, double *plimit)

Description: Gets the current limitation value for the specified channel on the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- limit – Pointer to variable to receive offset value in Volts or modulation offset percentage.

FDwfAnalogOutModeSet(HDWF hdwf, int idxChannel, DwfAnalogOutMode mode)

Description: Set the generator output mode for the specified instrument channel. With channel index -1, each enabled Analog Out channel mode will be configured to use the same, new option. This option is supported on Electronics Explorer Analog Out Channel 3 and 4, Positive and Negative Power supplies, to set current or voltage waveform generator mode.

The DwfAnalogOutMode options are:

- DwfAnalogOutModeVoltage
- DwfAnalogOutModeCurren

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- mode – Generator mode option to set.

FDwfAnalogOutModeGet(HDWF hdwf, int idxChannel, DwfAnalogOutMode *pmode)

Description: Retrieves the current generator mode option for the specified instrument channel.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- mode – Pointer to variable to receive the generator mode option.

DwfAnalogOutIdle	Idle Constant Capabilities
DwfAnalogOutIdleDisable	Supported on Electronics Explorer: Channel 1&2 0V ouput Channel 3&4. having 1 kΩ resistor to GND and diode
DwfAnalogOutIdleOffset	The idle output is the configured Offset level.
DwfAnalogOutIdleInitial	The idle output voltage level has the initial waveform value of the current configuration. This depends on the selected signal type, Offset, Amplitude and Amplitude Modulator configuration.

FDwfAnalogOutIdleInfo(HDWF hdwf, int idxChannel, int *pfsidle)

Description: Returns the supported generator idle output options. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the DwfAnalogOutIdle constants in dwf.h.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pfsidlec – Variable to receive the supported generator idle options.

FDwfAnalogOutIdleSet(HDWF hdwf, int idxChannel, AnalogOutNode idle)

Description: Sets the generator idle output for the specified instrument channel. The idle output selects the output while not running, Ready, Stopped, Done, or Wait states.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- idle – Generator function option to set.

FDwfAnalogOutIdleGet(HDWF hdwf, int idxChannel, AnalogOutNode *pidle)

Description: Retrieves the generator idle output option for the specified instrument channel.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pidle – Pointer to variable to receive the generator function option.

6.3 States

See the description of **FDwfDeviceTriggerInfo**.

FDwfAnalogOutTriggerSourceSet(HDWF hdwf, int idxChannel, TRIGSRC trigsrc)

Description: Sets the trigger source for the channel on instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- trigsrc – Trigger source to set.

FDwfAnalogOutTriggerSourceGet(HDWF hdwf, int idxChannel, TRIGSRC *ptrigsrc)

Description: Gets the current trigger source setting for the channel on instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- ptrigsrc – Pointer to variable to receive the trigger source.

FDwfAnalogOutTriggerSlopeSet(
HDWF hdwf, int idxChannel, **Error! Reference source not found.** slope)

Description: Sets the trigger slope for the channel on instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- slope – Trigger slope to set.

FDwfAnalogOutTriggerSlopeGet(
HDWF hdwf, int idxChannel, **Error! Reference source not found.** *pslope)

Description: Gets the current trigger slope setting for the channel on instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pslope – Pointer to variable to receive the trigger slope.

FDwfAnalogOutRunInfo(
HDWF hdwf, int idxChannel, double *psecMin, double *psecMax)

Description: Returns the supported run length range for the instrument in Seconds. Zero values represent an infinite (or continuous) run. Default value is zero.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- psecMin – Variable to receive the supported minimum run length.
- psecMax – Variable to receive the supported maximum run length.

FDwfAnalogOutRunSet(HDWF hdwf, int idxChannel, double secRun)

Description: Sets the run length for the instrument in Seconds.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- secRun – Run length to set expressed in seconds.

FDwfAnalogOutRunGet(HDWF hdwf, int idxChannel, double *psecRun)

Description: Reads the configured run length for the instrument in Seconds.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- psecRun – Pointer to variable to receive the run length.

FDwfAnalogOutRunStatus(HDWF hdwf, int idxChannel, double *psecRun)

Description: Reads the remaining run length. It returns data from the last FDwfAnalogOutStatus call.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- psecRun – Pointer to variable to receive the remaining run length.

FDwfAnalogOutWaitInfo(
HDWF hdwf, int idxChannel, double *psecMin, double *psecMax)

Description: Returns the supported wait length range in Seconds. The wait length is how long the instrument waits after being triggered to generate the signal. Default value is zero.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- psecMin/Max – Variable to receive the supported minimum/maximum wait length.

FDwfAnalogOutWaitSet(HDWF hdwf, int idxChannel, double secWait)

Description: Sets the wait length for the channel on instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- secWait – Wait length to set expressed in seconds.

FDwfAnalogOutWaitGet(HDWF hdwf, int idxChannel, double *psecWait)

Description: Gets the current wait length for the channel on instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- psecWait – Pointer to variable to receive the wait length.

FDwfAnalogOutRepeatInfo(HDWF hdwf, int idxChannel, int *pnMin, int *pnMax)

Description: Returns the supported repeat count range. This is how many times the generated signal will be repeated upon. Zero value represents infinite repeat. Default value is one.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pnMin – Variable to receive the supported minimum repeat count.
- pnMax – Variable to receive the supported maximum repeat count.

The function above is used to :

FDwfAnalogOutRepeatSet(HDWF hdwf, int idxChannel, int cRepeat)

Description: Sets the repeat count.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- cRepeat – Repeat count to set.

FDwfAnalogOutRepeatGet(HDWF hdwf, int idxChannel, int *pcRepeat)

Description: Reads the current repeat count.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pcRepeat – Pointer to variable to receive the repeat count.

FDwfAnalogOutRepeatStatus(HDWF hdwf, int idxChannel, int *pcRepeat)

Description: Reads the remaining repeat counts. It only returns information from the last FDwfAnalogOutStatus function call, it does not read from the device.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pcRepeat – Pointer to variable to receive the remaining repeat counts.

FDwfAnalogOutRepeatTriggerSet(HDWF hdwf, int idxChannel, BOOL fRepeatTrigger)

Description: Sets the repeat trigger option. To include the trigger in wait-run repeat cycles, set fRepeatTrigger to TRUE. It is disabled by default.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- fRepeatTrigger – Boolean used to specify if the trigger should be included in a repeat cycle.

FDwfAnalogOutRepeatTriggerGet(
HDWF hdwf, int idxChannel, BOOL *pfRepeatTrigger)

Description: Verifies if the trigger has been included in wait-run repeat cycles.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pfRepeatTrigger – Pointer to variable to receive the repeat trigger option.

FDwfAnalogOutMasterSet(HDWF hdwf, int idxChannel, int idxMaster)

Description: Sets the state machine master of the channel generator. With channel index -1, each enabled Analog Out channel will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- idxMaster – Node index.

FDwfAnalogOutMasterGet(HDWF hdwf, int idxChannel, int *pidxMaster)

Description: Verifies the parameter set by FDwfAnalogOutMasterSet.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pidxMaster – Pointer to variable to receive parameter.

7 Analog I/O

The AnalogIO functions are used to control the power supplies, reference voltage supplies, voltmeters, ammeters, thermometers, and any other sensors on the device. These are organized into channels which contain a number of nodes. For instance, a power supply channel might have three nodes: an enable setting, a voltage level setting/reading, and current limitation setting/reading.

See the AnalogIO.py example.

FDwfAnalogIOReset(HDWF hdwf)

Description: Resets and configures (by default, having auto configure enabled) all AnalogIO instrument parameters to default values.

Parameters:

- hdwf – Open interface handle on a device.

FDwfAnalogIOConfigure(HDWF hdwf)

Description: Configures the instrument.

Parameters:

- hdwf – Open interface handle on a device.

FDwfAnalogIOStatus(HDWF hdwf)

Description: Reads the status of the device and stores it internally. The following status functions will return the information that was read from the device when the function above was called.

Parameters:

- hdwf – Open interface handle on a device.

FDwfAnalogIOEnableInfo(HDWF hdwf, BOOL *pfSet, BOOL *pfStatus)

Description: Verifies if Master Enable Setting and/or Master Enable Status are supported for the AnalogIO instrument. The Master Enable setting is essentially a software switch that “enables” or “turns on” the AnalogIO channels. If supported, the status of this Master Enable switch (Enabled/Disabled) can be queried by calling FDwfAnalogIOEnableStatus.

Parameters:

- hdwf – Open interface handle on a device.
- pfSet – Returns true when the master enable setting is supported.
- pfStatus – Return true when the status of the master enable can be read.

FDwfAnalogIOEnableSet(HDWF hdwf, BOOL fMasterEnable)

Description: Sets the master enable switch.
Parameters:
- hdwf – Open interface handle on a device.
- fMasterEnable – Set TRUE to enable the master switch; FALSE to disable the master switch.

FDwfAnalogIOEnableGet(HDWF hdwf, BOOL *pfMasterEnable)

Description: Returns the current state of the master enable switch. This is not obtained from the device.
Parameters:
- hdwf – Open interface handle on a device.
- pfMasterEnable – Pointer to variable to return the enabled configuration.

FDwfAnalogIOEnableStatus(HDWF hdwf, BOOL *pfMasterEnable)

Description: Returns the master enable status (if the device supports it). This can be a switch on the board or an overcurrent protection circuit state.
Parameters:
- hdwf – Open interface handle on a device.
- pfMasterEnabled – Pointer to variable to return the active status.

FDwfAnalogIOChannelCount(HDWF hdwf, int *pnChannel)

Description: Returns the number of AnalogIO channels available on the device.
Parameters:
- hdwf – Open interface handle on a device.
- pnChannel – Pointer to variable to return the number of channels.

FDwfAnalogIOChannelName(
HDWF hdwf, int idxChannel, char szName[32], char szLabel[16])

Description: Returns the name (long text) and label (short text, printed on the device) for a channel.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- szName – Pointer to character array to return the user name.
- szLabel – Pointer to character array to return the label.

FDwfAnalogIOChannelInfo(HDWF hdwf, int idxChannel, int *pnNodes)

Description: Returns the number of nodes associated with the specified channel.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pnNodes – Pointer to variable to return the number of node .

FDwfAnalogIOChannelNodeName(
HDWF hdwf, int idxChannel, int idxNode,
char szNodeName[32], char szUnits[16])

Description: Returns the node name (“Voltage”, “Current”…) and units (“V”, “A”) for an Analog I/O node **.**

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- idxNode – Node index.
- szNodeName – Pointer to character array to return the node name.
- szUnits – Pointer to character array to return the value units.

FDwfAnalogIOChannelNodeInfo(
HDWF hdwf, int idxChannel, int idxNode, ANALOGIO *panalogio)

Description: Returns the supported channel nodes. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the *ANALOGIO* constants in dwf.h. The acquisition mode selects one of the following modes, *ANALOGIO*:

ANALOGIO Modes	*ANALOGIO* Mode Functions
analogioEnable	Enable I/O node; used to enable a power supply, reference voltage, etc.
analogioVoltage	Voltage I/O node; used to input/output voltage levels.
analogioCurrent	Current I/O node; used to input/output current levels.
analogioTemperature	Temperature I/O node; used to retrieve read values from a temperature sensor.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- idxNode – Node index.
- panalogio – Pointer to variable to return the node type.

FDwfAnalogIOChannelNodeSetInfo(HDWF hdwf, int idxChannel, int idxNode, double *pmin, double *pmax, int *pnSteps)

Description: Returns node value limits. Since a Node can represent many things (Power supply, Temperature sensor, etc.), the *Minimum*, *Maximum*, and *Steps* parameters also represent different types of values. In broad terms, the (Maximum – Minimum)/Steps = the number of specific input/output values. **FDwfAnalogIOChannelNodeInfo** returns the type of values to expect and **FDwfAnalogIOChannelNodeName** returns the units of these values.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Analog I/O channel index of the device.
- idxNode – Node index.
- pmin – Minimum settable value.
- pmax – Maximum settable value.
- pnSteps – Number of steps between minimum and maximum values.

FDwfAnalogIOChannelNodeSet(
HDWF hdwf, int idxChannel, int idxNode, double value)

Description: Sets the node value for the specified node on the specified channel.

Parameters:

- hdwf – Open interface handle on a device.
- idxNode – Node index.
- idxChannel – Analog I/O channel index of the device.
- value – Value to set.

FDwfAnalogIOChannelNodeGet(
HDWF hdwf, int idxChannel, int idxNode, double *pvalue)

Description: Returns the currently set value of the node on the specified channel.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Analog I/O channel index of the device.
- idxNode – Node index.
- pvalue – Pointer to variable to return the configured value.

FDwfAnalogIOChannelNodeStatusInfo(
HDWF hdwf, int idxChannel, int idxNode, double *pmin, double *pmax, int *pnSteps)

Description: Returns node the range of reading values available for the specified node on the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- idxNode – Node index.
- pmin – Minimum reading value.
- pmax – Maximum reading value.
- pnSteps – Number of steps between minimum and maximum values.

FDwfAnalogIOChannelNodeStatus(
HDWF hdwf, int idxChannel, int idxNode, double *pvalue)

Description: Returns the value reading of the node.
Parameters:
- hdwf –Interface handle.
- idxChannel – Channel index.
- idxNode – Node index.
- pvalue – Pointer to variable to return the value reading.

8 Digital I/O

See the DigitalIO.py example.

FDwfDigitalIOReset(HDWF hdwf)

Description: Resets and configures (by default, having auto configure enabled) all DigitalIO instrument parameters to default values. It sets the output enables to zero (tri-state), output value to zero, and configures the DigitalIO instrument.

Parameters:

- hdwf – Open interface handle on a device.

FDwfDigitalIOConfigure(HDWF hdwf)

Description: Configures the DigitalIO instrument. This doesn't have to be used if AutoConfiguration is enabled.

Parameters:

- hdwf – Open interface handle on a device.

FDwfDigitalIOStatus(HDWF hdwf)

Description: Reads the status and input values, of the device DigitalIO to the PC. The status and values are accessed from the `FDwfDigitalIOInputStatus` function.

Parameters:

- hdwf – Open interface handle on a device.

FDwfDigitalIOOutputEnableInfo(HDWF hdwf, unsigned int *pfsOutputEnableMask)

Description: Returns the output enable mask (bit set) that can be used on this device. These are the pins that can be used as outputs on the device.

Parameters:

- hdwf – Open interface handle on a device.
- pfsOutputEnableMask – Variable to return the OE mask bit field.

FDwfDigitalIOOutputEnableSet(HDWF hdwf, unsigned int fsOutputEnable)

Description: Enables specific pins for output. This is done by setting bits in the fsOutEnable bit field (1 for enabled, 0 for disabled).

Parameters:

- hdwf – Open interface handle on a device.
- fsOutputEnable – Output enable bit set.

FDwfDigitalIOOutputEnableGet(HDWF hdwf, unsigned int *pfsOutputEnable)

Description: Returns a bit field that specifies which output pins have been enabled.

Parameters:

- hdwf – Open interface handle on a device.
- pfsOutputEnable – Pointer to variable to returns output enable bit set.

FDwfDigitalIOOutputInfo(HDWF hdwf, unsigned int *pfsOutputMask)

Description: Returns the settable output value mask (bit set) that can be used on this device.

Parameters:

- hdwf – Open interface handle on a device.
- pfsOutputMask – Variable to return the output value mask.

FDwfDigitalIOOutputSet(HDWF hdwf, unsigned int fsOutput)

Description: Sets the output logic value on all output pins.

Parameters:

- hdwf – Open interface handle on a device.
- fsOutput – Output bit set.

FDwfDigitalIOOutputGet(HDWF hdwf, unsigned int *pfsOutput)

Description: Returns the currently set output values across all output pins.

Parameters:

- hdwf – Open interface handle on a device.
- pfsOutput – Pointer to variable to returns output bit set.

FDwfDigitalIOInputInfo(HDWF hdwf, unsigned int *pfsInputMask)

Description: returns the readable input value mask (bit set) that can be used on the device.

Parameters:

- hdwf – Open interface handle on a device.
- pfsInputMask – Variable to return the input value mask.

FDwfDigitalIOInputStatus(HDWF hdwf, unsigned int *pfsInput)

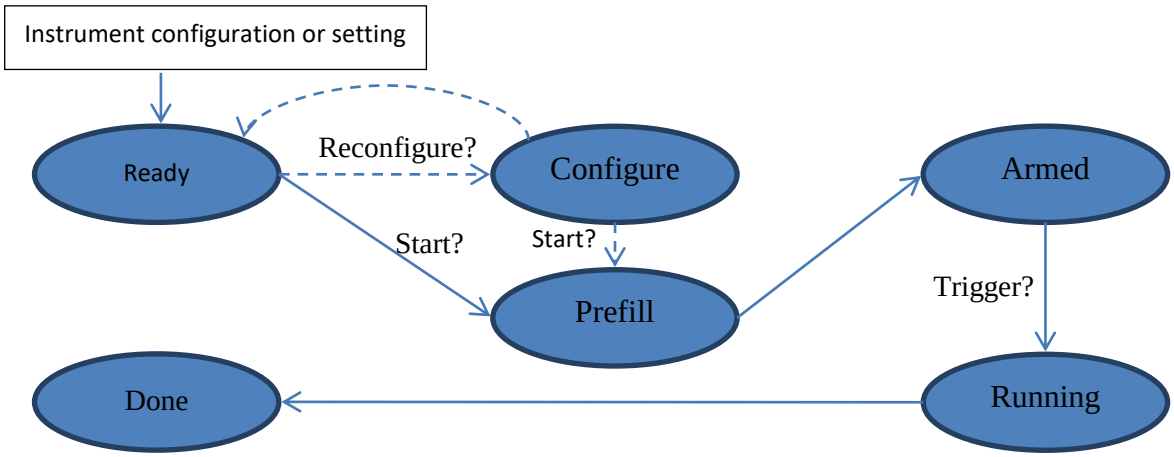
Description: Returns the input states of all I/O pins. Before calling the function above, call the FDwfDigitalIOStatus function to read the Digital I/O states from the device.

Parameters:

- hdwf – Open interface handle on a device.
- pfsInput – Variable to return the input value.

9 Digital In (Logic Analyzer)

The Digital In instrument states:



The states are defined in dwf.h DwfState type.

- **Ready**: Initial state. After *FDwfDigitalInConfigure* or any *FDwfDigitalIn*Set* function call goes to this state. With *FDwfDigitalInConfigure,* reconfigure goes to Configure state.
- **Configure**: The digital in auto trigger is reset.
- **Prefill**: Prefills the buffer with samples need before trigger.
- **Armed**: It waits for trigger.
- **Running**: For single acquisition mode remains in this state to acquire samples after trigger set by *FDwfDigitalInTriggerPositionSet*. Scan screen and shift modes remain until configure or any set function of this instrument.
- **Done**: Final state.

See the following examples: DigitalIn_Acquisition/Record.py.

9.1 Control

FDwfDigitalInReset(HDWF hdwf)

Description: Resets and configures (by default, having auto configure enabled) all DigitalIn instrument parameters to default values.

Parameters:

- hdwf – Interface handle.

FDwfDigitalInConfigure(HDWF hdwf, BOOL fReconfigure, BOOL fStart)

Description: Configures the instrument and start or stop the acquisition. To reset the Auto trigger timeout, set fReconfigure to TRUE.

Parameters:

- hdwf – Interface handle.
- fReconfigure – Configure the device.
- fStart – Start the acquisition.

FDwfDigitalInStatus(HDWF hdwf, BOOL fReadData, DwfState *psts)

Description: Checks the state of the instrument. To read the data from the device, set fReadData to TRUE. For single acquisition mode, the data will be read only when the acquisition is finished.

Parameters:

- hdwf – interface handle.
- fReadData – TRUE if data should be read.
- psts – Variable to receive the acquisition state.

FDwfDigitalInStatusSamplesLeft(HDWF hdwf, int *pcSamplesLeft)

Description: Retrieves the number of samples left in the acquisition.

Parameters:

- hdwf – Interface handle.
- pcSamplesLeft – Variable to receive the remaining samples to acquire.

FDwfDigitalInStatusSamplesValid(HDWF hdwf, int *pcSamplesValid)

Description: Retrieves the number of valid/acquired data samples.

Parameters:

- hdwf – Interface handle.
- pcSamplesValid – Variable to receive the number of valid samples.

FDwfDigitalInStatusIndexWrite(HDWF hdwf, int *pidxWrite)

Description: Retrieves the buffer write pointer. This is needed in ScanScreen acquisition mode to display the scan bar.
Parameters:
- hdwf – Interface handle.
- pidxWrite – Variable to receive the position of the acquisition.

FDwfDigitalInStatusAutoTriggered(HDWF hdwf, BOOL *pfAuto)

Description: Verifies if the acquisition is auto triggered.
Parameters:
- hdwf – Interface handle.
- pfAuto – Returns TRUE if the acquisition was auto triggered.

FDwfDigitalInStatusData(HDWF hdwf, void *rgData, int countOfDataBytes)

Description: Retrieves the acquired data samples from the instrument. It copies the data samples to the provided buffer. The sample format is specified by **FDwfDigitalInSampleFormatSet** function.
Parameters:
- hdwf – Interface handle.
- rgData – Pointer to allocated buffer to copy the acquisition data.
- countOfDataBytes – Number of bytes to copy.

FDwfDigitalInStatusData2(HDWF hdwf,
void *rgData, int idxSample, int countOfDataBytes)

Description: Retrieves the acquired data samples from the instrument. It copies the data samples to the provided buffer. The sample format is specified by **FDwfDigitalInSampleFormatSet** function.
Parameters:
- hdwf – Interface handle.
- rgData – Pointer to allocated buffer to copy the acquisition data.
- idxSample – First sample index to copy.
- countOfDataBytes – Number of bytes to copy.

FDwfDigitalInStatusRecord(
HDWF hdwf, int *pcdDataAvailable, int *pcdDataLost, int *pcdDataCorrupt)

Description: Retrieves information about the recording process. The data loss occurs when the device acquisition is faster than the read process to PC. In this case, the device recording buffer is filled and data samples are overwritten. Corrupt samples indicate that the samples have been overwritten by the acquisition process during the previous read. In this case, try optimizing the loop process for faster execution or reduce the acquisition frequency or record length to be less than or equal to the device buffer size (record length <= buffer size/frequency).

Parameters:

- hdwf – Interface handle.
- pcdDataAvailable – Pointer to variable to receive the available number of samples.
- pcdDataLost – Pointer to variable to receive the lost samples after the last check.
- pcdDataCorrupt – Pointer to variable to receive the number of samples that could be corrupt.

9.2 Configuration

FDwfDigitalInInternalClockInfo(HDWF hdwf, double *phzFreq)

Description: Retrieves the internal clock frequency.
Parameters:
- hdwf – Interface handle.
- phzFreq – Pointer to return the internal clock frequency.

FDwfDigitalInClockSourceInfo(HDWF hdwf, int *pfsDwfDigitalInClockSource)

Description: Returns the supported clock sources for Digital In instrument. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the *DwfDigitalInClockSource* constants in dwf.h:
- DwfDigitalInClockSourceInternal: Internal clock.
- DwfDigitalInClockSourceExternal: External clock source.

Parameters:
- hdwf – Open interface handle on a device.
- pfsDwfDigitalInClockSource – Pointer to variable to return the available clock source options.

FDwfDigitalInClockSourceSet(HDWF hdwf, DwfDigitalInClockSource v)

Description: Sets the clock source of instrument.
Parameters:
- hdwf – Open interface handle on a device.
- v – Clock source.

FDwfDigitalInClockSourceGet(HDWF hdwf, DwfDigitalInClockSource *pv)

Description: Gets the clock source of instrument.
Parameters:
- hdwf – Open interface handle on a device.
- pv – Pointer to variable to return the configured value.

FDwfDigitalInDividerInfo(HDWF hdwf, unsigned int *pdivMax)

Description: Returns the maximum supported clock divider value. This specifies the sample rate.
Parameters:
- hdwf –Interface handle.
- pdivMax – Pointer to variable to return the available maximum divider value.

FDwfDigitalInDividerSet(HDWF hdwf, unsigned int div)

Description: Sets the clock divider value.
Parameters:
- hdwf – Interface handle.
- div – Divider value.

FDwfDigitalInDividerGet(HDWF hdwf, unsigned int *pdiv)

Description: Gets the configured clock divider value.
Parameters:
- hdwf – Interface handle.
- pdiv – Pointer to return configured value.

FDwfDigitalInBitsInfo(HDWF hdwf, int *pnBits)

Description: Returns the number of Digital In bits.
Parameters:
- hdwf – Interface handle.
- pnBits – Pointer to variable to return the number of bits.

FDwfDigitalInInputOrderSet(HDWF hdwf, bool fDioFirst)

Description: Configures the order of values stored in the sampling array. If fDIOFirst = true DIO24..39 are placed at the beginning of the array followed by DIN0..23. With fDIOFirst = false DIN0..23 are placed at the beginning followed by DIO24..31. Valid only for Digital Discovery device.
Parameters:
- hdwf – Interface handle.
- fDioFirst – DIO or DIN lines start from index zero.

FDwfDigitalInSampleFormatSet(HDWF hdwf, int nBits)

Description: Sets the sample format, the number of bits starting from least significant bit. Valid options are 8, 16, and 32.

Parameters:

- hdwf – Interface handle.
- nBits – Sample format.

FDwfDigitalInSampleFormatGet(HDWF hdwf, int *pnBits)

Description: Returns the configured sample format.

Parameters:

- hdwf – Interface handle.
- pnBits – Pointer to return configured value.

FDwfDigitalInBufferSizeInfo(HDWF hdwf, int *pnSizeMax)

Description: Returns the Digital In maximum buffer size.

Parameters:

- hdwf – Interface handle.
- pnSizeMax – Pointer to variable to return maximum buffer size.

FDwfDigitalInBufferSizeSet(HDWF hdwf, int nSize)

Description: Set the buffer size.

Parameters:

- hdwf – Interface handle.
- nSize – Buffer size.

FDwfDigitalInBufferSizeGet(HDWF hdwf, int *pnSize)

Description: Returns the configured buffer size.

Parameters:

- hdwf – Interface handle.
- nSize – Pointer to return configured value.

FDwfDigitalInSampleModeInfo(HDWF hdwf, int *pfsDwfDigitalInSampleMode)

Description: Returns the supported sample modes. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the DwfDigitalInSampleMode constants in dwf.h:

- DwfDigitalInSampleModeSimple: Stores one sample on every divider clock pulse.
- DwfDigitalInSampleModeNoise: Stores alternating noise and sample values, where noise is more than one transition between two samples. This could indicate glitches or ringing. It is available when sample rate is less than maximum clock frequency, divider is greater than one.

Parameters:

- hdwf – Interface handle.
- pfsDwfDigitalInSampleMode – Pointer to return the supported sample modes.

The function above

FDwfDigitalInSampleModeSet(HDWF hdwf, DwfDigitalInSampleMode v)

Description: Set the sample mode.

Parameters:

- hdwf – Open interface handle on a device.
- v – Sample mode.

FDwfDigitalInSampleModeGet(HDWF hdwf, DwfDigitalInSampleMode *pv)

Description: Return the configured sample mode.

Parameters:

- hdwf – Open interface handle on a device.
- pv – Pointer to return configured value.

FDwfDigitalInAcquisitionModeInfo(HDWF hdwf, int *pfsacqmode)

Description: Returns the supported DigitalIn acquisition modes. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the ACQMODE constants in DWF.h. The acquisition mode selects one of the following modes, ACQMODE:

- **acqmodeSingle**: Perform a single buffer acquisition. This is the default setting.
- **acqmodeScanShift**: Perform a continuous acquisition in FIFO style. The trigger setting is ignored. The last sample is at the end of buffer. The FDwfDigitalInStatusSamplesValid function is used to show the number of the acquired samples, which will grow until reaching the BufferSize. Then the waveform "picture" is shifted for every new sample.
- **acqmodeScanScreen**: Perform continuous acquisition circularly writing samples into the buffer. The trigger setting is ignored. The IndexWrite shows the buffer write position. This is similar to a heart monitor display.

Parameters:

- hdwf – Interface handle.
- pfsacqmode – Pointer to return the supported acquisition modes.

FDwfDigitalInAcquisitionModeSet(HDWF hdwf, ACQMODE acqmode)

Description: Sets the acquisition mode.

Parameters:

- hdwf – Interface handle.
- acqmode – Acquisition mode to set.

FDwfDigitalInAcquisitionModeGet(HDWF hdwf, ACQMODE *pacqmode)

Description: Retrieves the acquisition mode.

Parameters:

- hdwf – Interface handle.
- pacqmode – Variable to receive the current acquisition mode.

FDwfDigitalInSampleSensibleSet(HDWF hdwf, unsigned int fs)

Description: Selects the signals to be used for data compression in record acquisition mode.

Parameters:

- hdwf – Interface handle.
- fs – bit field set of signals to look for compression.

FDwfDigitalInSampleSensibleGet(HDWF hdwf, unsigned int *pfs)

Description: Retrieves the signals being used for data compression in record acquisition mode.
Parameters:
- hdwf – Interface handle.
- pfs – Pointer to variable to receive configured value

9.3 Trigger

See the description of **FDwfDeviceTriggerInfo**.

FDwfDigitalInTriggerSourceSet(HDWF hdwf, TRIGSRC trigsrc)

Description: Sets the trigger source for the instrument.
Parameters:
- hdwf – Interface handle.
- trigsrc – Trigger source to set.

FDwfDigitalInTriggerSourceGet(HDWF hdwf, TRIGSRC *ptrigsrc)

Description: Gets the current trigger source setting for the instrument.
Parameters:
- hdwf – Interface handle.
- ptrigsrc – Pointer to variable to receive the trigger source.

FDwfDigitalInTriggerSlopeSet(HDWF hdwf, **Error! Reference source not found.** slope)

Description: Sets the trigger slope for the instrument.
Parameters:
- hdwf – Interface handle.
- slope – Trigger source to set.

FDwfDigitalInTriggerSlopeGet(HDWF hdwf, **Error! Reference source not found.** *pslope)

Description: Gets the current trigger source setting for the instrument.
Parameters:
- hdwf – Interface handle.
- pslope – Pointer to variable to receive the trigger slope.

FDwfDigitalInTriggerPositionInfo(
HDWF hdwf, unsigned int *pnSamplesAfterTriggerMax)

Description: Returns maximum values of the trigger position in samples. This can be greater than the specified buffer size.
Parameters:
- hdwf – Interface handle.
- pnSamplesAfterTriggerMax – Variable to receive the maximum trigger position.

FDwfDigitalInTriggerPositionSet(HDWF hdwf, unsigned int cSamplesAfterTrigger)

Description: Sets the number of samples to acquire after trigger.
Parameters:
- hdwf – Interface handle.
- cSamplesAfterTrigger – Samples after trigger.

FDwfDigitalInTriggerPositionGet(
HDWF hdwf, unsigned int *pcSamplesAfterTrigger)

Description: Gets the configured trigger position.
Parameters:
- hdwf – Interface handle.
- pcSamplesAfterTrigger – Pointer to variable to receive configured value

FDwfDigitalInTriggerPrefillSet(HDWF hdwf, unsigned int cSamplesBeforeTrigger)

Description: Sets the number of samples to acquire before arming in Record acquisition mode. The prefill is used for record with trigger to make sure at last the required number of samples are collected before arming, before looking for trigger event.
With prefill 0 the recording process will stream data only after trigger event.
With prefill more than zero the recording will stream until trigger occurs plus the samples specified by trigger position.
Parameters:
- hdwf – Interface handle.
- cSamplesBeforeTrigger – Samples before trigger.

FDwfDigitalInTriggerPrefillGet(
HDWF hdwf, unsigned int *pcSamplesBeforeTrigger)

Description: Gets the configured trigger prefill.
Parameters:
- hdwf – Interface handle.
- pcSamplesBeforeTrigger – Pointer to variable to receive configured value

FDwfDigitalInTriggerAutoTimeoutInfo(
HDWF hdwf, double *psecMin, double *psecMax, double *pnSteps)

Description: Returns the minimum and maximum auto trigger timeout values, and the number of adjustable steps.
Parameters:
- hdwf – Interface handle.
- psecMin – Variable to receive the minimum timeout.
- psecMax – Variable to receive the maximum timeout.
- pnSteps – Variable to return the number of steps.

FDwfDigitalInTriggerAutoTimeoutSet(HDWF hdwf, double secTimeout)

Description: Configures the auto trigger timeout value in seconds.
Parameters:
- hdwf – Interface handle.
- secTimeout – Timeout to set.

FDwfDigitalInTriggerAutoTimeoutGet(HDWF hdwf, double *psecTimeout)

Description: Returns the configured auto trigger timeout value in seconds. The acquisition is auto triggered when the specified time elapses. With zero value the timeout is disabled, performing “Normal” acquisitions.
Parameters:
- hdwf – Interface handle.
- psecTimeout – Variable to receive the current timeout.

9.4 Trigger Detector

In order to use trigger on digital in pins, set trigger source with FDwfDigitalInTriggerSourceSet to trigsrcDetectorDigitalIn.

```
FDwfDigitalInTriggerInfo(HDWF hdwf,
unsigned int *pfsLevelLow, unsigned int *pfsLevelHigh,
unsigned int *pfsEdgeRise, unsigned int *pfsEdgeFall)
```

Description: Returns the supported digital in triggers. The bits of the arguments represent pins.
The logic for the trigger bits is: *Low and High and (Rise or Fall)*. Setting a bit in both rise and fall will trigger on any edge, any transition. For instance FDwfDigitalInTriggerInfo(hdwf, 1, 2, 4, 8) will generate trigger when DIO-0 is low and DIO-1 is high and DIO-2 is rising or DIO-3 is falling.

Parameters:

- hdwf – Interface handle.
- pfsLevelLow – Variable to receive the supported low state triggers.
- pfsLevelHigh – Variable to receive the supported low state triggers.
- pfsEdgeRise – Variable to receive the supported rising edge triggers.
- pfsEdgeFall – Variable to receive the supported falling edge triggers.

```
FDwfDigitalInTriggerSet(HDWF hdwf,
unsigned int fsLevelLow, unsigned int fsLevelHigh,
unsigned int fsEdgeRise, unsigned int fsEdgeFall)
```

Description: Configures the digital in trigger detector.

Parameters:

- hdwf – Interface handle.
- fsLevelLow – Set low state condition.
- fsLevelHigh – Set high state condition.
- fsEdgeRise – Set rising edge condition.
- fsEdgeFall – Set falling edge condition.

```
FDwfDigitalInTriggerGet(HDWF hdwf,
unsigned int *pfsLevelLow, unsigned int *pfsLevelHigh,
unsigned int *pfsEdgeRise, unsigned int *pfsEdgeFall)
```

Description: Returns the configured digital in trigger detector option.

Parameters:

- hdwf – Interface handle.
- pfsLevelLow – Variable to receive the configured value.
- pfsLevelHigh – Variable to receive the configured value.
- pfsEdgeRise – Variable to receive the configured value.
- pfsEdgeFall – Variable to receive the configured value.

```
FDwfDigitalInTriggerResetSet(HDWF hdwf,
unsigned int fsLevelLow, unsigned int fsLevelHigh,
unsigned int fsEdgeRise, unsigned int fsEdgeFall)
```

Description: Configures the digital in trigger reset condition.

Parameters:

- hdwf – Interface handle.
- fsLevelLow – Set low state condition.
- fsLevelHigh – Set high state condition.
- fsEdgeRise – Set rising edge condition.
- fsEdgeFall – Set falling edge condition.

```
FDwfDigitalInTriggerCountSet(HDWF hdwf, int cCount, int fRestart)
```

Description: Configures the trigger counter.

Parameters:

- hdwf – Interface handle.
- cCount – Set event count.
- fRestart – Set to restart counter after expires or not and wait for next reset condition.

```
FDwfDigitalInTriggerLengthSet(HDWF hdwf,
double secMin, double secMax, int idxSync)
```

Description: Configures the trigger timing. The synchronization modes are the following:

0 – Normal

1 – Timing: use for UART, CAN. The min length specifies bit length and max the timeout length.

2 – PWM: use for 1-Wire. The min length specifies sampling time and max the timeout length.

Parameters:

- hdwf – Interface handle.
- secMin – Set minimum length in seconds, up to 20sec.
- secMax – Set maximum length in seconds, up to 20sec.
- idxSync – Set synchronization mode.

FDwfDigitalInTriggerMatchSet(HDWF hdwf,
int iPin, unsigned int fsMask, unsigned int fsValue, int cBitStuffing)

Description: Configure the deserializer. The bits are left shifted. The mask and value should be specified according to this, in MSBit first order. Like to trigger on first to fourth bits received from a sequence of 8, b1010XXXX, set mask to 0x000000F0 and value to 0x000000A0.

Pulse: To trigger on a pulse configure the following:
- For positive pulse specify rising and for negative falling edge reset.
FDwfDigitalInTriggerResetSet(hdwf, 0, 0, positive?1<<dio:0, negative?1<<dio:0);
- For positive pulse specify high and for negative low level trigger.
FDwfDigitalInTriggerSet(hdwf, negative?1<<dio:0, positive?1<<dio:0, 0, 0);

Glitch: To trigger on glitch, a pulse length at most the specified value:
FDwfDigitalInTriggerLengthSet(hdwf, 0, max, 0); // maximum pulse length in seconds

Timeout: To trigger on pulse timeout, on a pulse after the specified minimum time expires:
FDwfDigitalInTriggerLengthSet(hdwf, min, 0, 0); // minimum pulse length in seconds
More: To trigger on pulse ending slope which is longer than the specified minimum time:
FDwfDigitalInTriggerLengthSet(hdwf, min, -1, 0); // minimum pulse length in seconds

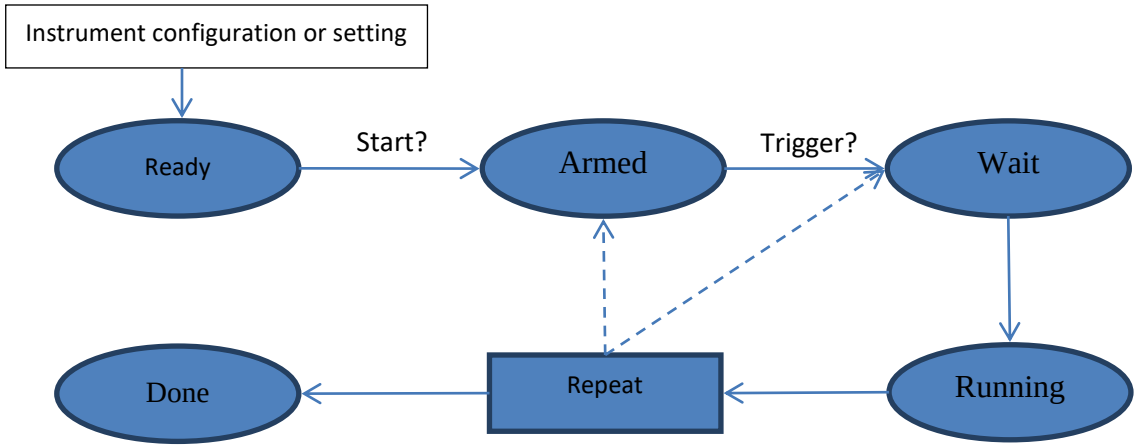
Length: To trigger on a pulse length with the specified minimum and maximum lengths use:
FDwfDigitalInTriggerLengthSet(hdwf, min, max, 0); // min/max pulse length in seconds

Parameters:

- hdwf – Interface handle.
- iPin – Set pin to deserialize.
- fsMask – Set bit mask pattern.
- fsValue – Set bit match pattern.
- cBitStuffing – Set bit stuffing count.

10 Digital Out (Pattern Generator)

The DigitalOut instrument states:



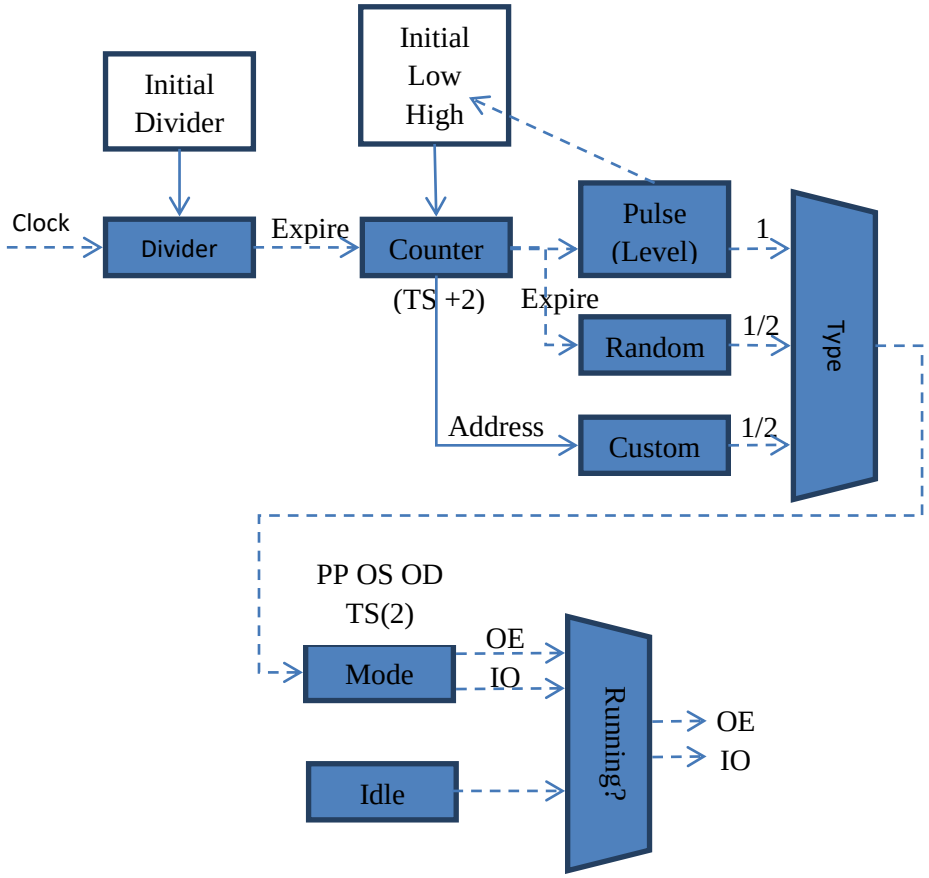
The states are described defined in dwf.h DwfState type.

- **Ready**: Initial state. After *FDwfDigitalOutConfigure* or any *FDwfDigitalOut*Set* function call goes to this state. With digital out, configure start command goes to Armed state
- **Armed**: It waits for trigger.
- **Wait**: Remains in this state for the time period specified by *FDwfDigitalOutWaitSet* function.
- **Running**: Remains in this state for the time period specified by *FDwfDigitalOutRunSet* function.
- ***Repeat***: Goes to Armed or Wait state according to the FDwfDigitalOutRepeatTriggerSet setting for the number of times specified by FDwfDigitalOutRepeatSet.
- **Done**: Final state.

This states machine controls all digital out channels.

See the following examples: DigitalOut_BinaryCounter/Pins.py

Channel configuration:

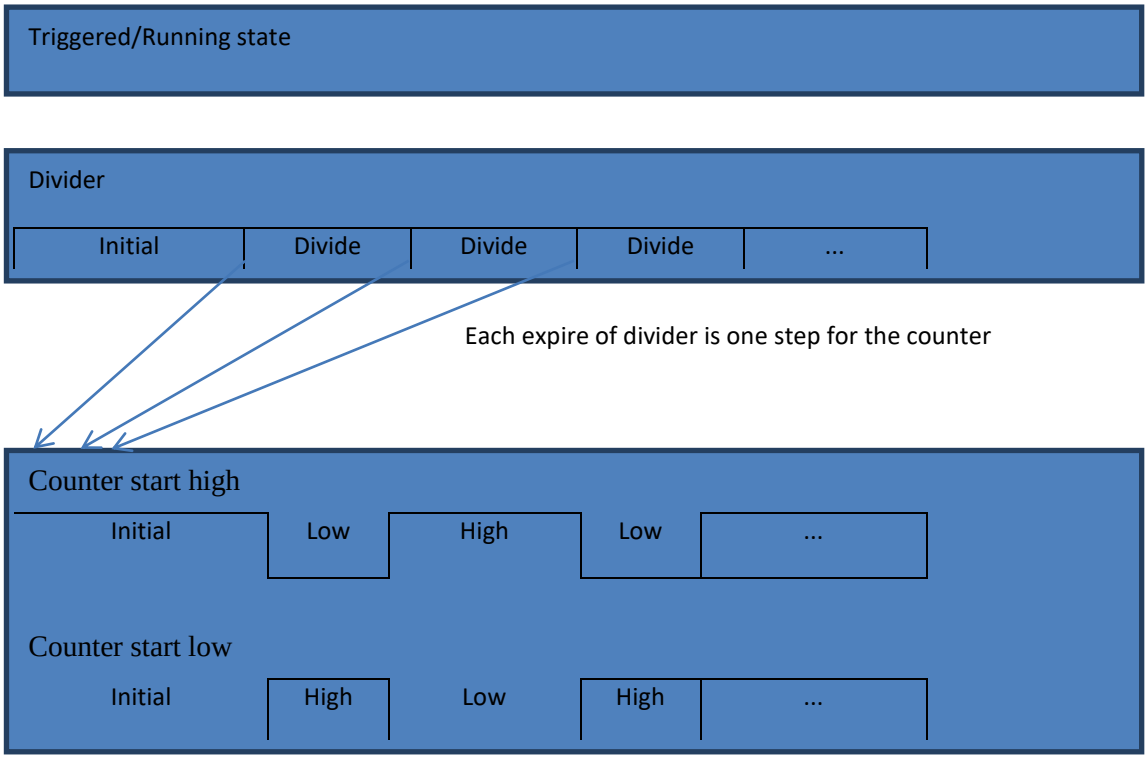


The initial values, for divider and counter, specify the initially loaded values, initial delay, when entering in Running state. The Divider specifies the clock division. This rate will be the custom sample frequency and step for the counter. When entering Running state, the initial value specified with *FDwfDigitalOutDividerInitSet* is loaded. When this expires, the value specified by *FDwfDigitalOutDividerSet* will be loaded upon each expiration.
The Counter initial value is set by *FDwfDigitalOutCounterInitSet* function. This function also sets the initial level. When this expires the level values specified by *FDwfDigitalOutCounterSet* are loaded upon further expiration. On counter expiration the level is toggled, and this directs the low or high value loading. In case one of these is zero, the level is not toggled.

The Counter is used for:
- Pulse to generate the low and high state lengths.
- Random to set update rate.
- Custom to address buffer. The samples are configured by *FDwfDigitalOutDataSet* function. This also configures the counter low/high according *countOfBits* parameter. In TS mode the counter step is double, providing two bits of samples for output: value and enable.

The output Mode (*FDwfDigitalOutModeSet*) selects between: PP, OS, OD and TS.
The Idle output (*FDwfDigitalOutIdleSet*) selects the output while not in *Running* state.

Pulse signal:



For pulse signal the initial level and initial value are specified with *FDwfDigitalOutCounterInitSet* function. These are loaded when entering Running state.

10.1 Control

FDwfDigitalOutReset(HDWF hdwf)

Description: Resets and configures (by default, having auto configure enabled) all the instrument parameters to default values.
Parameters:
- hdwf – Interface handle.

FDwfDigitalOutConfigure(HDWF hdwf, BOOL fStart)

Description: Starts or stops the instrument.
Parameters:
- hdwf – Interface handle.
- fStart – Start the instrument. To stop, set to FALSE.

FDwfDigitalOutStatus(HDWF hdwf, DwfState *psts)

Description: Checks the state of the instrument.
Parameters:
- hdwf – Interface handle.
- psts – Pointer to variable to return the state.

10.2 Configuration

FDwfDigitalOutInternalClockInfo(HDWF hdwf, double *phzFreq)

Description: Retrieves the internal clock frequency.
Parameters:
- hdwf – Interface handle.
- phzFreq – Pointer to return the internal clock frequency.

See the description of **FDwfDeviceTriggerInfo**.

FDwfDigitalOutTriggerSourceSet(HDWF hdwf, TRIGSRC trigsrc)

Description: Sets the trigger source for the instrument. Default setting is trigsrcNone.
Parameters:
- hdwf – Interface handle.
- trigsrc – Trigger source to set.

FDwfDigitalOutTriggerSourceGet(HDWF hdwf, TRIGSRC *ptrigsrc)

Description: Gets the current trigger source setting for the instrument.
Parameters:
- hdwf – Interface handle.
- ptrigsrc – Pointer to variable to receive the trigger source.

FDwfDigitalOutTriggerSlopeSet(HDWF hdwf, **Error! Reference source not found.** slope)

Description: Sets the trigger slope for the instrument.
Parameters:
- hdwf – Interface handle.
- slope – Trigger source to set.

FDwfDigitalOutTriggerSlopeGet(HDWF hdwf, **Error! Reference source not found.** *pslope)

Description: Gets the current trigger source setting for the instrument.
Parameters:
- hdwf – Interface handle.
- pslope – Pointer to variable to receive the trigger slope.

FDwfDigitalOutRunInfo(HDWF hdwf, double *psecMin, double *psecMax)

Description: Returns the supported run length range for the instrument in seconds. Zero value (default) represent an infinite (or continuous) run.
Parameters:
- hdwf – Interface handle.
- psecMin – Variable to receive the supported minimum run length.
- psecMax – Variable to receive the supported maximum run length.

FDwfDigitalOutRunSet(HDWF hdwf, double secRun)

Description: Sets the run length for the instrument in Seconds.
Parameters:
- hdwf – Interface handle.
- secRun – Run length to set expressed in seconds.

FDwfDigitalOutRunGet(HDWF hdwf, double *psecRun)

Description: Reads the configured run length for the instrument in seconds.
Parameters:
- hdwf – Interface handle.
- psecRun – Pointer to variable to receive the run length.

FDwfDigitalOutRunStatus(HDWF hdwf, double *psecRun)

Description: Reads the remaining run length. It returns data from the last FDwfDigitalOutStatus call.
Parameters:
- hdwf – Interface handle.
- psecRun – Pointer to variable to receive the remaining run length.

FDwfDigitalOutWaitInfo(HDWF hdwf, double *psecMin, double *psecMax)

Description: Returns the supported wait length range in seconds. The wait length is how long the instrument waits after being triggered to generate the signal. Default value is zero.
Parameters:
- hdwf – Interface handle.
- psecMin – Variable to receive the supported minimum wait length.
- psecMax – Variable to receive the supported maximum wait length.

FDwfDigitalOutWaitSet(HDWF hdwf, double secWait)

Description: Sets the wait length.
Parameters:
- hdwf – Interface handle.
- secWait – Wait length to set expressed in seconds.

FDwfDigitalOutWaitGet(HDWF hdwf, double *psecWait)

Description: Gets the current wait length.
Parameters:
- hdwf – Interface handle.
- psecWait – Pointer to variable to receive the wait length.

FDwfDigitalOutRepeatInfo(HDWF hdwf, unsigned int *pnMin, unsigned int *pnMax)

Description: Returns the supported repeat count range. This is how many times the generated signal will be repeated. Zero value represents infinite repeats. Default value is one.
Parameters:
- hdwf – Interface handle.
- pnMin – Variable to receive the supported minimum repeat count.
- pnMax – Variable to receive the supported maximum repeat count.

FDwfDigitalOutRepeatSet(HDWF hdwf, unsigned int cRepeat)

Description: Sets the repeat count.
Parameters:
- hdwf – Interface handle.
- cRepeat – Repeat count to set.

FDwfDigitalOutRepeatGet(HDWF hdwf, unsigned int *pcRepeat)

Description: Reads the current repeat count.
Parameters:
- hdwf – Interface handle.
- pcRepeat – Pointer to variable to receive the repeat count.

FDwfDigitalOutRepeatStatus(HDWF hdwf, unsigned int *pcRepeat)

Description: Reads the remaining repeat counts. It only returns information from the last FDwfDigitalOutStatus function call, it does not read from the device.
Parameters:
- hdwf – Interface handle.
- pcRepeat – Pointer to variable to receive the remaining repeat counts.

FDwfDigitalOutRepeatTriggerSet(HDWF hdwf, BOOL fRepeatTrigger)

Description: Sets the repeat trigger option. To include the trigger in wait-run repeat cycles, set fRepeatTrigger to TRUE. It is disabled by default.
Parameters:
- hdwf – Interface handle.
- fRepeatTrigger – Boolean used to specify if the trigger should be included in a repeat cycle.

FDwfDigitalOutRepeatTriggerGet(HDWF hdwf, BOOL *pfRepeatTrigger)

Description: Verifies if the trigger has been included in wait-run repeat cycles.
Parameters:
- hdwf – Open interface handle on a device.
- pfRepeatTrigger – Pointer to variable to receive the repeat trigger option.

FDwfDigitalOutCount(HDWF hdwf, int *pcChannel)

Description: Returns the number of Digital Out channels by the device specified by hdwf.
Parameters:
- hdwf – Interface handle.
- pcChannel – Pointer to variable to receive the number of channels in the instrument.

FDwfDigitalOutEnableSet(HDWF hdwf, int idxChannel, BOOL fEnable)

Description: Enables or disables the channel specified by idxChannel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- fEnable – TRUE to enable, FALSE to disable.

FDwfDigitalOutEnableGet(HDWF hdwf, int idxChannel, BOOL *pfEnable)

Description: Verifies if a specific channel is enabled or disabled.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pfEnable – Pointer to variable to receive enabled state.

FDwfDigitalOutOutputInfo(
HDWF hdwf, int idxChannel, int *pfsDwfDigitalOutOutput)

Description: Returns the supported output modes of the channel. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the DwfDigitalOutOutput constants in DWF.h:

- **DwfDigitalOutOutputPushPull**: Default setting.
- **DwfDigitalOutOutputOpenDrain**: External pull needed.
- **DwfDigitalOutOutputOpenSource**: External pull needed.
- **DwfDigitalOutOutputThreeState**: Available with custom and random types.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- pfsDwfDigitalOutOutput – Pointer to variable to receive the supported output modes.

FDwfDigitalOutOutputSet(HDWF hdwf, int idxChannel, DwfDigitalOutOutput v)

Description: Specifies output mode of the channel.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- v – Output mode.

FDwfDigitalOutOutputGet(HDWF hdwf, int idxChannel, DwfDigitalOutOutput *pv)

Description: Verifies if a specific channel output mode.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- pfEnable – Pointer to variable to receive enabled state.

FDwfDigitalOutTypeInfo(HDWF hdwf, int idxChannel, int *pfsDwfDigitalOutType)

Description: Returns the supported types of the channel. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the DwfDigitalOutType constants in dwf.h:

- **DwfDigitalOutTypePulse**: Frequency = internal frequency/divider/(low + high counter).
- **DwfDigitalOutTypeCustom**: Sample rate = internal frequency / divider.
- **DwfDigitalOutTypeRandom**: Random update rate = internal frequency/divider/counter alternating between low and high values.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- pfsDwfDigitalOutType – Pointer to variable to receive the supported output types.

FDwfDigitalOutTypeSet(HDWF hdwf, int idxChannel, DwfDigitalOutType v)

Description: Sets the output type of the specified channel.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- v – Output mode.

FDwfDigitalOutTypeGet(HDWF hdwf, int idxChannel, DwfDigitalOutType *pv)

Description: Verifies the type of a specific channel.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- pfEnable – Pointer to variable to receive enabled state.

FDwfDigitalOutIdleInfo(HDWF hdwf, int idxChannel, int *pfsDwfDigitalOutIdle)

Description: Returns the supported idle output types of the channel. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the DwfDigitalOutIdle constants in dwf.h. Output while not running:

- **DwfDigitalOutIdleInit**: Output initial value.
- **DwfDigitalOutIdleLow**: Low level.
- **DwfDigitalOutIdleHigh**: High level.
- **DwfDigitalOutIdleZet**: Three state.

Parameters:

- hdwf – Interface handle.
- idxChannel – Channel index.
- pfsDwfDigitalOutIdle – Pointer to variable to receive the supported idle output types.

FDwfDigitalOutIdleSet(HDWF hdwf, int idxChannel, DwfDigitalOutIdle v)

Description: Sets the idle output of the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- v – Value to set idle output.

FDwfDigitalOutIdleGet(HDWF hdwf, int idxChannel, DwfDigitalOutIdle *pv)

Description: Verifies the idle output of a specific channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pv – Pointer to variable to receive configured value.

FDwfDigitalOutDividerInfo(
HDWF hdwf, int idxChannel, unsigned int *vMin, unsigned int *vMax)

Description: Returns the supported clock divider value range.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pnMin – Variable to receive the supported minimum divider value.
- pnMax – Variable to receive the supported maximum divider value.

FDwfDigitalOutDividerInitSet(HDWF hdwf, int idxChannel, unsigned int v)

Description: Sets the initial divider value of the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- v – Divider initial value.

FDwfDigitalOutDividerInitGet(HDWF hdwf, int idxChannel, unsigned int *pv)

Description: Verifies the initial divider value of the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pv – Pointer to variable to receive configured value.

FDwfDigitalOutDividerSet(HDWF hdwf, int idxChannel, unsigned int v)

Description: Sets the divider value of the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- v – Divider value.

FDwfDigitalOutDividerGet(HDWF hdwf, int idxChannel, unsigned int *pv)

Description: Verifies the divider value of the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pv – Pointer to variable to receive configured value.

FDwfDigitalOutCounterInfo(
HDWF hdwf, int idxChannel, unsigned int *vMin, unsigned int *vMax)

Description: Returns the supported counter value range.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pnMin – Variable to receive the supported minimum counter value.
- pnMax – Variable to receive the supported maximum counter value.

FDwfDigitalOutCounterInitSet(
HDWF hdwf, int idxChannel, BOOL fHigh, unsigned int v)

Description: Sets the initial state and counter value of the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- fHigh – Start high.
- v – Divider initial value.

FDwfDigitalOutCounterInitGet(
HDWF hdwf, int idxChannel, int *pfHigh, unsigned int *pv)

Description: Retrieves the initial state and counter value for the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pfHigh – Pointer to variable to receive configured value.
- pv – Pointer to variable to receive configured value.

FDwfDigitalOutCounterSet(
HDWF hdwf, int idxChannel, unsigned int vLow, unsigned int vHigh)

Description: Sets the counter low and high values for the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- vLow – Counter low value.
- vHigh – Counter high value.

FDwfDigitalOutCounterGet(
HDWF hdwf, int idxChannel, unsigned int *pvLow, unsigned int *pvHigh)

Description: Verifies the low and high counter value of the specified channel.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pvLow – Pointer to variable to receive configured value.
- pvHigh – Pointer to variable to receive configured value.

FDwfDigitalOutDataInfo(
HDWF hdwf, int idxChannel, unsigned int *pcountOfBitsMax)

Description: Returns the maximum buffers size, the number of custom data bits.
Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- pcountOfBitsMax – Variable to receive the maximum number of bits.

FDwfDigitalOutDataSet(
HDWF hdwf, int idxChannel, void *rgBits, unsigned int countOfBits)

Description: Sets the custom data bits. The function also sets the counter initial, low and high value, according the number of bits. The data bits are sent out in LSB first order. For TS output, the count of bits is the total number of output value (I/O) and output enable (OE) bits, which should be an even number.

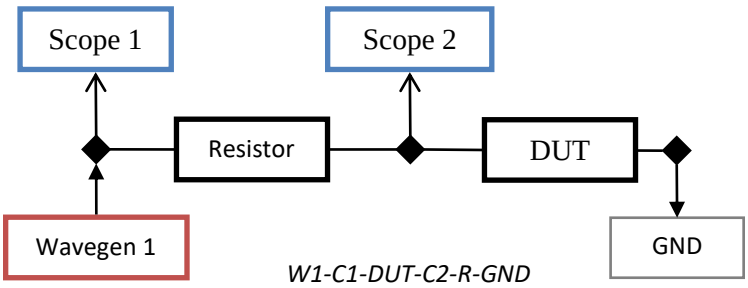
Custom Data Bits									
BYTE	0						1		
Bits	0	1	2	3	...	7	0	1	...
Output	I/O(0)	OE(0)	I/O(1)	OE(1)	...	OE(3)	I/O(4)	OE(4)	...

Parameters:
- hdwf – Interface handle.
- idxChannel – Channel index.
- rgBits – Custom data array.
- countOfBits –Number of bits.

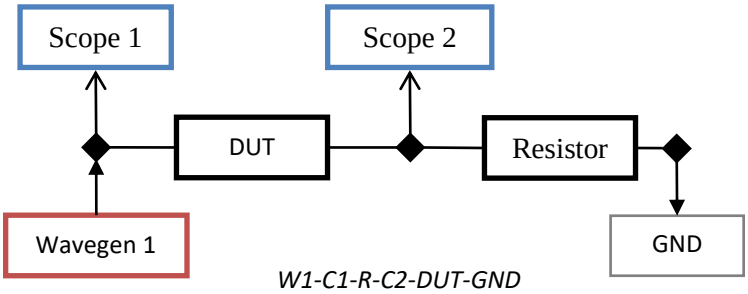
11 Analog Impedance

The Analog Impendace functions use the Analog Input and Output channel 1.
When used with Imepdance Analyzer module the Analog IO (Power Supplies) and Digital IO will be used too.

These functions can be used with Impedance Analyzer for Analog Discovery (specify this with value 8 in FDwfAnalogImpedanceModeSet function) or the analyzer circuit can be constructed in the following ways:



W1-C1-DUT-C2-R-GND



W1-C1-R-C2-DUT-GND

The DUT (Device Under Test) stands for the inductive or capacitive load to be analyzed and Resistor is the reference resistor. The resistor value depends on the load value and frequency.

See AnalogImpedance_Meter.py, AnalogImpedance_ Compensation.py examples.

FDwfAnalogImpedanceReset(HDWF hdwf)

Description: Resets the AI configuration to default value.
Parameters:
- hdwf – Interface handle.

FDwfAnalogImpedanceModeSet(HDWF hdwf, int mode)

Description: Specifies the circuit to be used.
Parameters:
- hdwf – Interface handle.
- mode – circuit model to be used
  - 0: W1-C1-DUT-C2-R-GND
  - 1: W1-C1-R-C2-DUT-GND
  - 8: Impedance Analyzer module for Analog Discovery

FDwfAnalogImpedanceModeGet(HDWF hdwf, int *pmode)

Description: Returns the selected circuit model.
Parameters:
- hdwf – Interface handle.
- pmode – Pointer to variable to receive configured value.

FDwfAnalogImpedanceReferenceSet(HDWF hdwf, double ohms)

Description: Specifies the reference resistor to be used.
For AD IA module the resistor is selected by relays controlled by power supplies and digital IOs.
Parameters:
- hdwf – Interface handle.
- ohms – Reference resistor value.

FDwfAnalogImpedanceReferenceGet(HDWF hdwf, double *pohms)

Description: Returns the reference resistor value.
Parameters:
- hdwf – Interface handle.
- pohms – Pointer to variable to receive configured value.

FDwfAnalogImpedanceFrequencySet(HDWF hdwf, double hz)

Description: Configures the stimulus frequency.
Parameters:
- hdwf – Interface handle.
- hz – Frequency value.

FDwfAnalogImpedanceFrequencyGet(HDWF hdwf, double *phz)

Description: Returns the frequency value.
Parameters:
- hdwf – Interface handle.
- phz – Pointer to variable to receive configured value.

FDwfAnalogImpedanceAmplitudeSet(HDWF hdwf, double volts)

Description: Configures the stimulus signal amplitude, half of the peak to peak value.
Parameters:
- hdwf – Interface handle.
- volts – Amplitude value.

FDwfAnalogImpedanceAmplitudeGet(HDWF hdwf, double *pvolts)

Description: Returns the amplitude value.
Parameters:
- hdwf – Interface handle.
- pvolts – Pointer to variable to receive configured value.

FDwfAnalogImpedanceOffsetSet(HDWF hdwf, double volts)

Description: Configures the stimulus signal offset.
Parameters:
- hdwf – Interface handle.
- volts – Offset value.

FDwfAnalogImpedanceOffsetGet(HDWF hdwf, double *pvolts)

Description: Returns the offset value.
Parameters:
- hdwf – Interface handle.
- pvolts – Pointer to variable to receive configured value.

FDwfAnalogImpedanceProbeSet(HDWF hdwf, double ohmRes, double faradCap)

Description: Specifies the probe impedance that will be taken in consideration for measurements.
The default values are set specific for device when calling the FDwfAnalogImpedanceReset function.
Parameters:
- hdwf – Interface handle.
- ohmRes – Probe resistance.
- faradCap – Probe capacitance.

FDwfAnalogImpedanceProbeGet(HDWF hdwf, double *pohmRes, double *pfaradCap)

Description: Returns the probe impedance.
Parameters:
- hdwf – Interface handle.
- pohmRes – Pointer to variable to receive configured value.
- pfaradCap – Pointer to variable to receive configured value.

FDwfAnalogImpedancePeriodSet(HDWF hdwf, double cMinPeriods)

Description: Specifies the minimum number of periods to be captured.
Parameters:
- hdwf – Interface handle.
- cMinPeriods – Number of minimum periods, default 16.

FDwfAnalogImpedancePeriodGet(HDWF hdwf, double *pcMinPeriods)

Description: Returns the periods value.
Parameters:
- hdwf – Interface handle.
- pcMinPeriods – Pointer to variable to receive configured value.

FDwfAnalogImpedanceCompReset(HDWF hdwf)

Description: Resets the currently configured compensation parameters.
Parameters:
- hdwf – Interface handle.

FDwfAnalogImpedanceCompSet(HDWF hdwf,
double ohmOpenResistance, double ohmOpenReactance,
double ohmShortResistance, double ohmShortReactance)

Description: Specifies the open and short compensation parameters. These values are specific for the circuit/adapter.

Parameters:

- hdwf – Interface handle.
- ohmOpenResistance – Open resistance value.
- ohmOpenReactance – Open reactance value.
- ohmShortResistance – Short resistance value.
- ohmShortReactance – Short reactance value.

FDwfAnalogImpedanceCompGet(HDWF hdwf,
double *pohmOpenResistance, double *pohmOpenReactance,
double *pohmShortResistance, double *pohmShortReactance)

Description: Returns the compensation parameters.

Parameters:

- hdwf – Interface handle.
- pohmOpenResistance – Pointer to variable to receive configured value.
- pohmOpenReactance – Pointer to variable to receive configured value.
- pohmShortResistance – Pointer to variable to receive configured value.
- pohmShortReactance – Pointer to variable to receive configured value.

FDwfAnalogImpedaceConfigure(HDWF hdwf, int fStart)

Description: Configures the instrument and start or stop the analysis.

Parameters:

- hdwf – Interface handle.
- fStart – Start the analysis.

FDwfAnalogImpedanceStatus(HDWF hdwf, DwfState *psts)

Description: Checks the state of the acquisition.

Parameters:

- hdwf – Interface handle.
- psts – Variable to receive the acquisition state.

FDwfAnalogImpedanceStatusInput(HDWF hdwf, int idxChannel, double *pgain, double *pradian)

Description: Read the raw input, for network analysis purpose. This returns the raw values without taking in consideration the probe characteristics or compensation parameters.
For scope channel 1 (idxChannel = 0) the gain is relative to Wavegen amplitude (Ampltiude/Channel1) and the phase is zero. For further channels the gain and phase is relative to channel 1, gain = C#/C1

DwfAnalogImpedance	int	Impedance measurement
DwfAnalogImpedanceImpedance	0	|Z| in Ohms = Sqrt(Rs^2 + Xs^2)
DwfAnalogImpedanceImpedancePhase	1	Ø in Radians = Tan2(Xs,Rs)
DwfAnalogImpedanceResistance	2	Rs in Ohms
DwfAnalogImpedanceReactance	3	Xs in Ohms = -1/(ω*Cs) = ω*Ls
DwfAnalogImpedanceAdmittance	4	|Y| in Siemen = Sqrt(Gp^2 + Bp^2)
DwfAnalogImpedanceAdmittancePhase	5	Ø in Radians = Tan2(Bp,Gp)
DwfAnalogImpedanceConductance	6	Gp in Siemen
DwfAnalogImpedanceSusceptance	7	Bp in Siemen = ω*Cp = -1/(ω*Lp)
DwfAnalogImpedanceSeriesCapactance	8	Cs in Farad = -1/(ω*Xs)
DwfAnalogImpedanceParallelCapacitance	9	Cp in Farad = Bp/ ω
DwfAnalogImpedanceSeriesInductance	10	Ls in Henry = Xs/ ω
DwfAnalogImpedanceParallelInductance	11	Lp in Henry = = -1/(ω*Bp)
DwfAnalogImpedanceDissipation	12	Dissipation factor = Rs/Xs = Gp /Bp
DwfAnalogImpedanceQuality	13	Quality factor = Xs/Rs = Bp/Gp
		ω = 2*PI*Hz

Parameters:
- hdwf – Interface handle.
- idxChannel – Scope channel index.
- pgain – Pointer to variable to receive the gain.
- pradian – Pointer to variable to receive the phase.

FDwfAnalogImpedanceStatusMeasure(HDWF hdwf, DwfAnalogImpedance measure, double *pvalue)

Description: Read the DUT measurements. These take in account the scope probe characteristics and compensation parameters.

Parameters:
- hdwf – Interface handle.
- measure – Select measurement to return.
- pvalue – Pointer to variable to receive the measurement.

12 Digital Protocols

The protocols use the Digital-In/Out resources to create various communication protocols. Only one of the protocols can be used at a time and no digital-io or out other functions.
Nore: The DIO channe indexing for Digital Discovery starts from 0, 0 is DIO-24, 1 is DIO-25…

12.1 UART

See Digital_Uart.py example.

FDwfDigitalUartReset(HDWF hdwf)

Description: Resets the UART configuration to default value. Use FDwfDigitalOutReset to reset the output.
Parameters:
- hdwf – Interface handle.

FDwfDigitalUartRateSet(HDWF hdwf, double hz)

Description: Sets the data rate.
Parameters:
- hdwf – Interface handle.
- hz – data rate to set

FDwfDigitalUartBitsSet(HDWF hdwf, double cBits)

Description: Sets the character length, typically 8, 7, 6 or 5.
Parameters:
- hdwf – Interface handle.
- cBits – character length

FDwfDigitalUartParitySet(HDWF hdwf, int parity)

Description: Sets the parity bit: 0 for no parity, 1 for odd and 2 for even parity.
Parameters:
- hdwf – Interface handle.
- parity – parity mode to set

FDwfDigitalUartStopSet(HDWF hdwf, double cBits)

Description: Sets the stop length as number of bits.
Parameters:
- hdwf – Interface handle.
- cBits – stop length

FDwfDigitalUartTxSet(HDWF hdwf, int idxChannel)

Description: Specifies the DIO channel to use for transmission.
Parameters:
- hdwf – Interface handle.
- idxChannel – DIO channel to use for TX

FDwfDigitalUartRxSet(HDWF hdwf, int idxChannel)

Description: Specifies the DIO channel to use for reception.
Parameters:
- hdwf – Interface handle.
- idxChannel – DIO channel to use for RX

FDwfDigitalUartTx(HDWF hdwf, char *szTx, int cTx)

Description: Transmits the specified characters.
Parameters:
- hdwf – Interface handle.
- szTX – array of characters to send
- cTX – number of characters to send

FDwfDigitalUartRx(HDWF hdwf, char *szRx, int cRxMax, int *pcRx, int *pParity)

Description: Initializes the reception with cRxMax zero. Otherwise returns the received characters since the last call.
Parameters:
- hdwf – Interface handle.
- szRX – buffer to receive characters
- cRxMax – the maximum number of characters to receive, the buffer size
- pcRX – pointer to return the number of characters received
- pParity – pointer to return:
  - negative value for buffer overflow, in case the buffer got full since the previos call of this function
  - the first parity check failure as one based index
  - zero for no error

12.2 SPI

See examples: Digital_Spi.py, Digital_Spi_Dual.py, Digital_Spi_Quad.py, Digital_Spi_Siso.py

FDwfDigitalSpiReset(HDWF hdwf)

Description: Resets the SPI configuration to default value. Use FDwfDigitalOutReset to reset the outputs.
Parameters:
- hdwf – Interface handle.

FDwfDigitalSpiFrequencySet(HDWF hdwf, double hz)

Description: Sets the SPI frequency.
Parameters:
- hdwf – Interface handle.
- hz – bit rate to set (default 1kHz)

FDwfDigitalSpiClockSet(HDWF hdwf, int idxChannel)

Description: Specifies the DIO channel to use for SPI clock.
Parameters:
- hdwf – Interface handle.
- idxChannel – DIO channel to use for SPI clock (default DIO1)

FDwfDigitalSpiDataSet(HDWF hdwf, int idxDQ, int idxChannel)

Description: Specifies the DIO channels to use for SPI data.
Parameters:
- hdwf – Interface handle.
- idxDQ – specify data index to set, 0 = DQ0_MOSI_SISO, 1 = DQ1_MISO, 2 = DQ2, 3 = DQ3
- idxChannel – DIO channel to use for SPI data

FDwfDigitalSpiModeSet(HDWF hdwf, int iMode)

Description: Sets the SPI mode.

iMode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Parameters:
- hdwf – Interface handle.
- iMode – specify SPI mode, bit1 CPOL and bit0 CPHA

FDwfDigitalSpiOrderSet(HDWF hdwf, int fMSBLSB)

Description: Sets the bit order for SPI data.
Parameters:
- hdwf – Interface handle.
- fMSB – Specify bit order, 1 MSB first (default), 0 LSB first

FDwfDigitalSpiSelect(HDWF hdwf, int idxChannel, int iLevel)

Description: Controls the SPI CS signal(s).
Parameters:
- hdwf – Interface handle.
- idxChannel – Specify DIO channel for which to set the level
- iLevel – Set the channel level: 0 low, 1 high, -1 release (Z, high impedance)

FDwfDigitalSpiWriteRead(HDWF hdwf, int cDQ, int cBitPerWord, unsigned char *rgTX, int cTX, unsigned char *rgRX, int cRX)

Description: Performs SPI transfer of up to 8bit words. This function is intended for standard MOSI/MISO (cDQ 1) operations, but it can be used for other modes as long only write (rgTX/cTX) or read (rgRX/cRX) is specified.
The number of clock signals generated is the maximum of cTX and cRX.
Parameters:
- hdwf – Interface handle.
- cDQ – specify the DQ lines to use to transfer data:
  0 – SISO, use only DQ0 for read and write.
  1 – MISI/MISO, use DQ0 for write and DQ1 to read data.
  2 – dual, use DQ0 for even and DQ1 for odd bits.
  4 – quad, use DQ0 for 0,4,8…, DQ1 for 1,5,9…, DQ2 for 2,6,10…, DQ3 for 3,7,11… data bits.
- cBitPerWord – specify the number of bits to transfer for each word.
- rgTX – array of 8bit values (byte words) to write.
- cTX – number of words to write.
- rgRx – buffer for read words.
- cRx – number of words to read.

FDwfDigitalSpiWriteOne(HDWF hdwf, int cDQ, int cBits, unsigned int vTX)

Description: Performs SPI transmit of up to 32 bits. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – Specify the DQ lines to use to transfer data.
- cBits – Specify the number of bits to transmit.
- vTX – Specify the data to transmit.

FDwfDigitalSpiReadOne(HDWF hdwf, int cDQ, int cBits, unsigned int *pRX)

Description: Performs SPI reception of up to 32 bits. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – Specify the DQ lines to use to transfer data.
- cBits – Specify the number of bits to receive.
- pRX – Pointer to variable to return the received bits.

FDwfDigitalSpiWrite(HDWF hdwf, int cDQ, int cBitPerWord, unsigned char *rgTX, int cTX)

Description: Performs SPI transmission of up to 8-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – Specify the DQ lines to use to transfer data.
- cBitPerWord – Specify the number of bits to transfer for each word.
- rgTX – Array of 8-bit values (words) to transmit.
- cTX – Number of words to transmit.

FDwfDigitalSpiRead(HDWF hdwf, int cDQ, int cBitPerWord, unsigned char *rgRX, int cRX)

Description: Performs SPI reception of up to 8-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – Specify the DQ lines to use to transfer data.
- cBitPerWord – Specify the number of bits to transfer for each word.
- rgRx – Buffer for receive words.
- cRx – Number of words to received.

FDwfDigitalSpiWriteRead16(HDWF hdwf, int cDQ, int cBitPerWord, unsigned short *rgTX, int cTX, unsigned short *rgRX, int cRX)

Description: Performs SPI transfer of up to 16-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – Specify the DQ lines to use to transfer data.
- cBitPerWord – Specify the number of bits to transfer for each word.
- rgTX – Array of 16bit values (words) to transmit.
- cTX – Number of words to transmit.
- rgRx – Buffer for read words.
- cRx – Number of words to read.

FDwfDigitalSpiWriteRead32(HDWF hdwf, int cDQ, int cBitPerWord, unsigned int *rgTX, int cTX, unsigned int *rgRX, int cRX)

Description: Performs SPI transfer of up to 32-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – Specify the DQ lines to use to transfer data.
- cBitPerWord – Specify the number of bits to transfer for each word.
- rgTX – Array of 32bit values (words) to write.
- cTX – Number of words to write.
- rgRx – Buffer for read words.
- cRx – Number of words to read.

FDwfDigitalSpiRead16(HDWF hdwf, int cDQ, int cBitPerWord, unsigned short *rgRX, int cRX)

Description: Performs SPI read of up to 16-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – specify the DQ lines to use to transfer data.
- cBitPerWord – specify the number of bits to transfer for each word.
- rgRx – buffer for read words.
- cRx – number of words to read.

FDwfDigitalSpiRead32(HDWF hdwf, int cDQ, int cBitPerWord, unsigned int *rgRX, int cRX)

Description: Performs SPI read of up to 32-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – specify the DQ lines to use to transfer data.
- cBitPerWord – specify the number of bits to transfer for each word.
- rgRx – buffer for read words.
- cRx – number of words to read.

FDwfDigitalSpiWrite16(HDWF hdwf, int cDQ, int cBitPerWord, unsigned short *rgTX, int cTX)

Description: Performs SPI read of up to 16-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – specify the DQ lines to use to transfer data.
- cBitPerWord – specify the number of bits to transfer for each word.
- rgTX – array of 16-bit values (words) to write.
- cTX – number of words to write.

FDwfDigitalSpiWrite32(HDWF hdwf, int cDQ, int cBitPerWord, unsigned int *rgTX, int cTX)

Description: Performs SPI read of up to 32-bit words. See FDwfDigitalSpiWriteRead for more information.
Parameters:
- hdwf – Interface handle.
- cDQ – specify the DQ lines to use to transfer data.
- cBitPerWord – specify the number of bits to transfer for each word.
- rgTX – array of 32-bit values (int) to write.
- cTX – number of bytes to write.

12.3 I2C

See Digital_I2c.py exampels.

FDwfDigitalI2cReset(HDWF hdwf)

Description: Resets the I2C configuration to default value.
Parameters:
- hdwf – Interface handle.

FDwfDigitalI2cClear(HDWF hdwf, int *pfFree)

Description: Verifies and tries to solve eventual bus lockup. The argument returns true, non-zero value if the bus is free.
Parameters:
- hdwf – Interface handle.
- pfFree – pointer to return the

FDwfDigitalI2cRateSet(HDWF hdwf, double hz)

Description: Sets the data rate.
Parameters:
- hdwf – Interface handle.
- hz – bit rate to set (default 100kHz).

FDwfDigitalI2cReadNakSet(HDWF hdwf, int fNakLastReadByte)

Description: Specifies if the last read byte should be acknowledged or not. The I2C specifications require NAK, this parameter set to true.
Parameters:
- hdwf – Interface handle.
- fNakLastReadByte – value to set (default 1, true).

FDwfDigitalI2cStretchSet(HDWF hdwf, int fEnable)

Description: Enables or disables clock stretching.
Parameters:
- hdwf – Interface handle.
- fEnable – Enable (default) or disable the clock stretching.

FDwfDigitalI2cSclSet(HDWF hdwf, int idxChannel)

Description: Specifies the DIO channel to use for I2C clock.
Parameters:
- hdwf – Interface handle.
- idxChannel – DIO channel to use for SCL.

FDwfDigitalI2cSdaSet(HDWF hdwf, int idxChannel)

Description: Specifies the DIO channel to use for I2C data.
Parameters:
- hdwf – Interface handle.
- idxChannel – DIO channel to use for SDA.

FDwfDigitalI2cWriteRead(HDWF hdwf, unsigned char adr8bits, unsigned char *rgbTx, int cTx, unsigned char *rgbRx, int cRx, int *pNak)

Description: Performs I2C write, repeated start and read. In case zero bytes are specified for read (cRx) only write and for zero write (cTx) only read is peformed. The read/write bit in the address is controlled by the function. The returned NAK index returns one based index of the first negative ackedged transfer byte, zero when all the bytes where acknowledged. When the first address is acknowledged it returns 1. Returns negative value for other communication falures like timeout.
Parameters:
- hdwf – Interface handle.
- adr8bits – specify the address in 8-bit format, bAAAAAAAX.
- rgbTX – array of bytes to write.
- cTX – number of bytes to write.
- rgbRx – buffer for read bytes.
- cRx – number of bytes to read.
- pNak – pointer to variable to return eventual NAK index.

FDwfDigitalI2cRead(HDWF hdwf, unsigned char adr8bits, unsigned char *rgbRx, int cRx, int *pNak)

Description: Performs I2C read. See DwfDigitalI2cWriteRead function for more information.
Parameters:
- hdwf – Interface handle.
- adr8bits – specify the address.
- rgbRx – buffer for read bytes.
- cRx – number of bytes to read.
- pNak – pointer to variable to return eventual NAK index.

FDwfDigitalI2cWrite(HDWF hdwf, unsigned char adr8bits, unsigned char *rgbTx, int cTx, int *pNak)

Description: Performs I2C write. See DwfDigitalI2cWriteRead function for more information.
Parameters:
- hdwf – Interface handle.
- adr8bits – specify the address.
- rgbTX – array of bytes to write.
- cTX – number of bytes to write.
- pNak – pointer to variable to return eventual NAK index.

FDwfDigitalI2cWriteOne(HDWF hdwf, unsigned char adr8bits, unsigned char bTx, int *pNak)

Description: Performs I2C write of one byte. See DwfDigitalI2cWriteRead function for more information.
Parameters:
- hdwf – Interface handle.
- adr8bits – specify the address.
- bTX – bytes to write.
- pNak – pointer to variable to return eventual NAK index.

12.4 CAN

See Digital_Can.py example.

FDwfDigitalCanReset(HDWF hdwf)

Description: Resets the CAN configuration to default value. Use FDwfDigitalOutReset to reset the output.
Parameters:
- hdwf – Interface handle.

FDwfDigitalCanRateSet(HDWF hdwf, double hz)

Description: Sets the data rate.
Parameters:
- hdwf – Interface handle.
- hz – data rate to set.

FDwfDigitalCanPolaritySet(HDWF hdwf, double fInvert)

Description: Sets the signal polarity.
Parameters:
- hdwf – Interface handle.
- fInvert – 0 normal, 1 invert.

FDwfDigitalCanTxSet(HDWF hdwf, int idxChannel)

Description: Specifies the DIO channel to use for transmission.
Parameters:
- hdwf – Interface handle.
- idxChannel – DIO channel to use for TX.

FDwfDigitalCanRxSet(HDWF hdwf, int idxChannel)

Description: Specifies the DIO channel to use for reception.
Parameters:
- hdwf – Interface handle.
- idxChannel – DIO channel to use for RX.

FDwfDigitalCanTx(HDWF hdwf, int vID, int fExtended, int fRemote, int cDLC, unsigned char *rgTX)

Description: Performs a CAN transmission. Specifying -1 for vID it initializes the TX channel.
Parameters:
- hdwf – Interface handle.
- vID – identifier.
- fExtended – 0 base frame, 1 extended frame; 11- or 29-bit identifier.
- fRemote – 0 data frame, 1 remote request.
- rgTX - array of data bytes to send.

FDwfDigitalCanRx(HDWF hdwf, int *pvID, int *pfExtended, int *pfRemote, int *pcDLC, unsigned char *rgRX, int cRX, int *pvStatus)

Description: Returns the received frames since the last call. With cRX zero initializes the reception.
Parameters:
- hdwf – Interface handle.
- pvID – pointer to return the identifier.
- pfExtended - pointer to return the extended bit value.
- pfRemote – pointer to return the remote bit value.
- pcDLC – pointer to return the data length code.
- rgRX – poiter to array to return the data bytes.
- cRX - the maximum number of data bytes to receive, rgRX buffer size
- pvStatus – pointer to return the function status:
  0 - nothing received.
  1 – frame received without error.
  2 – frame received with bit stuffing error.
  3 – frame received with CRC error.

13 Devices

13.1 Electronics Explorer

The Electronics Explorer has nine AnalogIO channels. The master enable validates the power supplies and reference voltages.

Channel	Node	Name	Description
0		Vcc	Fixed supply
	0	Enable	Enables or disables the supply channel, 1/0
	1	Voltage	Select voltage 3.3V or 5V Status returns the voltage reading
	2	Current	Status returns the current reading
1		VP+	Positive supply
	0	Enable	Enables or disables the supply channel, 1/0
	1	Voltage	Specify voltage level, 0 to 9V Status returns the voltage reading
	2	Current	Specify the current limitation between 0 and 1.5A Status returns the current reading
2		VP-	Negative supply
	0	Enable	Specify voltage level, 0 to -9V Status returns the voltage reading
	1	Voltage	Specify voltage level, -0.5 to -5V
	2	Current	Specify the current limitation between 0 and -1.5A Status returns the current reading
3		Ref1	Voltage Reference 1
	0	Enable	Enables or disables the supply channel, 1/0
	1	Voltage	Specify voltage level between -10V and -10V
4		Ref2	Voltage Reference 2
	0	Enable	Enables or disables the supply channel, 1/0
	1	Voltage	Specify voltage level between -10V and -10V
5		Vmtr1	Voltmeter 1
	0	Voltage	Status returns the voltage reading, -15V to 15V
6		Vmtr2	Voltmeter 2
	0	Voltage	Status returns the voltage reading, -15V to 15V
7		Vmtr3	Voltmeter 3
	0	Voltage	Status returns the voltage reading, -15V to 15V
8		Vmtr4	Voltmeter 4
	0	Voltage	Status returns the voltage reading, -15V to 15V

13.2 Analog Discovery

The Analog Discovery has four AnalogIO channels. The master enable validates the power supplies.

Channel	Node	Name	Description
0		V+	Positive supply
	0	Enable	Enables or disables the supply channel, 1/0
1		V-	Negative supply
	0	Enable	Enables or disables the supply channel, 1/0
2		SYS	USB power monitor
	0	Voltage	Status returns the line voltage
	1	Current	Status returns the line current
	2	Temp	Status returns the device temperature
3		V+-	Power supply monitor
	0	Voltage	Status returns the voltage input for the supply regulators
	1	Current	Status returns the current taken by the supply regulators

13.3 Analog Discovery 2

The Analog Discovery 2 has four AnalogIO channels. The master enable validates the power supplies.

Channel	Node	Name	Description
0		V+	Positive supply
	0	Enable	Enables or disables the supply channel, 1/0
	1	Voltage	Specify voltage level, 0.5 to 5V
1		V-	Negative supply
	0	Enable	Enables or disables the supply channel, 1/0
	1	Voltage	Specify voltage level, -0.5 to -5V
2		USB	USB power monitor
	0	Voltage	Status returns the line voltage
	1	Current	Status returns the line current
	2	Temp	Status returns the device temperature
3		AUX	AUX power monitor
	0	Voltage	Status returns the line voltage
	1	Current	Status returns the line current

13.4 Digital Discovery

For the DigitalOut and IO functions, and AnalogIO DIOPP/PE the indexing 15:0 refers to DIO39:24.
The Digital Discovery has tree AnalogIO channels. The master enable activates the VIO output.

Channel	Node	Name	Description
0	0	Voltage	Configures the digital voltage between 1.2V and 3.3V
	1	DINPP	Configures the week pull for DIN lines with values: 0 down, 0.5 middle, 0.75 up, 1 up See Digital Discovery RM Input Dividers
	2	DIOPE	Configure pull enable (1) for DIO 39-24 as bit field set, like: 0x00F1 enables for DIO 31,30,29,28 and 24 See Digital Discovery RM I/O Level Translators
	3	DIOPP	Configure pull up/down (1/0) for DIO 39-24 as bit field set
	4	Drive	Configure drive strength for DIO lines: 0 (auto), 2mA, 4mA, 6mA, 8mA, 12mA, 16mA
	5	Slew	Configure slew rate for DIO lines: quietio, slow, fast See Xilinx DS162
1		VIO	VIO power monitor See Digital Discovery RM Power Supplies
	0	Voltage	Status returns the VIO voltage reading
	1	Current	Status returns the VIO current reading
2		USB	USB power monitor
	0	Voltage	Status returns the USB voltage reading
	1	Current	Status returns the USB current reading

14 Deprecated functions

The following functions are replaced by FDwfAnalogOutNode *providing access to the Amplitude and Frequency Modulation of Analog Out channels.

FDwfAnalogOutPlayStatus(HDWF hdwf, int idxChannel, int *cdDataFree, int *cdDataLost, int *cdDataCorrupted)

Description: Retrieves information about the play process. The data lost occurs when the device generator is faster than the sample send process from the PC. In this case, the device buffer gets emptied and generated samples are repeated. Corrupt samples are a warning that the buffer might have been emptied while samples were sent to the device. In this case, try optimizing the loop for faster execution; or reduce the frequency or run time to be less or equal to the device buffer size (run time <= buffer size/frequency).

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- cdDataFree – Pointer to variable to return the available free buffer space, the number of new samples that can be sent.
- cdDataLost – Pointer to variable to return the number of lost samples.
- cdDataCorrupted – Pointer to variable to return the number of samples that could be corrupted.

FDwfAnalogOutPlayData(HDWF hdwf, int idxChannel, double *rgdData, int cdData)

Description: Sends new data samples for play mode. Before starting the Analog Out instrument, prefill the device buffer with the first set of samples using the AnalogOutDataSet function. In the loop of sending the following samples, first call AnalogOutStatus to read the information from the device, then AnalogOutPlayStatus to find out how many new samples can be sent, then send the samples with AnalogOutPlayData.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- rgdData – Pointer to samples array to be sent to the device.
- cdData – Number of samples to send.

FDwfAnalogOutEnableSet(HDWF hdwf, int idxChannel, BOOL fEnable)

Description: Enables or disables the channel specified by idxChannel. With channel index -1, each Analog Out channel enable will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- fEnable – TRUE to enable, FALSE to disable.

FDwfAnalogOutEnableGet(HDWF hdwf, int idxChannel, BOOL *pfEnable)

Description: Verifies if a specific channel is enabled or disabled.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pfEnable – Pointer to variable to receive enabled state.

FDwfAnalogOutFunctionInfo(HDWF hdwf, int idxChannel, int *pfsfunc)

Description: Returns the supported generator function options. They are returned (by reference) as a bit field. This bit field can be parsed using the IsBitSet Macro. Individual bits are defined using the FUNC constants in dwf.h. These are:

FUNC Constants	FUNC Constant Capabilities
funcDC	Generate DC value set as offset.
funcSine	Generate sine waveform.
funcSquare	Generate square waveform.
funcTriangle	Generate triangle waveform.
funcRampUp	Generate a waveform with a ramp-up voltage at the beginning.
funcRampDown	Generate a waveform with a ramp-down voltage at the end.
funcNoise	Generate noise waveform from random samples.
funcCustom	Generate waveform from custom repeated data.
funcPlay	Generate waveform from custom data in stream play style.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pfsfunc – Variable to receive the supported generator function options.

FDwfAnalogOutFunctionSet(HDWF hdwf, int idxChannel, FUNC func)

Description: Sets the generator output function for the specified instrument channel. With channel index -1, each enabled Analog Out channel function will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- func – Generator function option to set.

FDwfAnalogOutFunctionGet(HDWF hdwf, int idxChannel, FUNC *pfunc)

Description: Retrieves the current generator function option for the specified instrument channel.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- ptrigsrc – Pointer to variable to receive the generator function option.

FDwfAnalogOutFrequencyInfo(
HDWF hdwf, int idxChannel, double *phzMin, double *phzMax)

Description: Returns the supported frequency range for the instrument. The maximum value shows the DAC frequency. The frequency of the generated waveform: repetition frequency for standard types and custom data; DAC update for noise type; sample rate for play type.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- phzMin – Variable to receive the supported minimum frequency.
- phzMax – Variable to receive the supported maximum frequency.

FDwfAnalogOutFrequencySet(HDWF hdwf, int idxChannel, double hzFrequency)

Description: Sets the frequency. With channel index -1, each enabled Analog Out channel frequency will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- hzFrequency – Frequency value to set expressed in Hz.

FDwfAnalogOutFrequencyGet(HDWF hdwf, int idxChannel, double *phzFrequency)

Description: Gets the currently set frequency for the specified channel on the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- hzFrequency – Pointer to variable to receive frequency value in Hz.

FDwfAnalogOutAmplitudeInfo(
HDWF hdwf, int idxChannel, double *pvMin, double *pvMax)

Description: Retrieves the amplitude range for the specified channel on the instrument. The amplitude is expressed in Volts units for carrier and in percentage units (modulation index) for AM/FM.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pvMin – Minimum amplitude level or modulation index.
- pvMax – Maximal amplitude level or modulation index.

FDwfAnalogOutAmplitudeSet(HDWF hdwf, int idxChannel, double vAmplitude)

Description: Sets the amplitude or modulation index for the specified channel on the instrument. With channel index -1, each enabled Analog Out channel amplitude will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- vAmplitude – Amplitude of channel in Volts or modulation index in percentage.

FDwfAnalogOutAmplitudeGet(HDWF hdwf, int idxChannel, double *pvAmplitude)

Description: Gets the currently set amplitude or modulation index for the specified channel on the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pvAmplitude – Pointer to variable to receive amplitude value in Volts or modulation index in percentage.

FDwfAnalogOutOffsetInfo(HDWF hdwf, int idxChannel, double *pvMin, double *pvMax)

Description: Retrieves available the offset range in units of volts.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pvMin – Minimum offset voltage or modulation offset percentage.
- pvMax – Maximum offset voltage or modulation offset percentage.

FDwfAnalogOutOffsetSet(HDWF hdwf, int idxChannel, double vOffset)

Description: Sets the offset value for the specified channel on the instrument. With channel index -1, each enabled Analog Out channel offset will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- vOffset – Value to set voltage offset in Volts or modulation offset percentage.

FDwfAnalogOutOffsetGet(HDWF hdwf, int idxChannel, double *pvOffset)

Description: Gets the current offset value for the specified channel on the instrument.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pvOffset – Pointer to variable to receive offset value in Volts or modulation offset percentage.

The function above is used to:

FDwfAnalogOutSymmetryInfo(
HDWF hdwf, int idxChannel, double *ppercentageMin, double *ppercentageMax)

Description: Obtains the symmetry (or duty cycle) range (0..100). This symmetry is supported for standard signal types. It the pulse duration divided by the pulse period.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- ppercentageMin – Minimum value of Symmetry percentage.
- ppercentageMax – Maximum value of Symmetry percentage.

FDwfAnalogOutSymmetrySet(
HDWF hdwf, int idxChannel, double percentageSymmetry)

Description: Sets the symmetry (or duty cycle) for the specified channel on the instrument. With channel index -1, each enabled Analog Out channel symmetry will be configured to use the same, new option.

Parameters:

- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- percentageSymmetry –Value of percentage of Symmetry (duty cycle).

FDwfAnalogOutSymmetryGet(
HDWF hdwf, int idxChannel, double *ppercentageSymmetry)

Description: Gets the currently set symmetry (or duty cycle) for the specified channel of the instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- ppercentageSymmetry — Pointer to variable to receive value of symmetry (duty cycle).

FDwfAnalogOutPhaseInfo(
HDWF hdwf, int idxChannel, double *pdegreeMin, double *pdegreeMax)

Description: Retrieves the phase range (in degrees 0...360) for the specified channel of the instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pdegreeMin – Minimum value of Phase (in degrees).
- pdegreeMax – Maximum value of Phase (in degrees).

FDwfAnalogOutPhaseSet(
HDWF hdwf, int idxChannel, double degreePhase)

Description: Sets the phase for the specified channel on the instrument. With channel index -1, each enabled Analog Out channel phase will be configured to use the same, new option.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- degreePhase – Value of Phase in degrees.

FDwfAnalogOutPhaseGet(HDWF hdwf, int idxChannel, double *pdegreePhase)

Description: Gets the current phase for the specified channel on the instrument.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pdegreePhase – Pointer to variable to receive Phase value (in degrees).

FDwfAnalogOutDataInfo(
HDWF hdwf, int idxChannel, int *pnSamplesMin, double *pnSamplesMax)

Description: Retrieves the minimum and maximum number of samples allowed for custom data generation.
Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- pnSamplesMin - Minimum number of samples available for custom data.
- pnSamplesMax – Maximum number of samples available for custom data.

FDwfAnalogOutDataSet(HDWF hdwf, int idxChannel, double *rgdData, int cdData)

Description: Sets the custom data or to prefill the buffer with play samples. The samples are double precision floating point values (rgdData) normalized to ±1.
With the custom function option, the data samples (cdData) will be interpolated to the device buffer size. The output value will be Offset + Sample*Amplitude, for instance:
- 0 value sample will output: Offset.
- +1 value sample will output: Offset + Amplitude.
- -1 value sample will output: Offset – Amplitude.

Parameters:
- hdwf – Open interface handle on a device.
- idxChannel – Channel index.
- rgbData – Buffer of samples to set.
- cData – Number of samples to set in rgbData.