ADQAPI Reference Guide

Author: Teledyne SP Devices

SPD Document Number: 14-1351

Revision: 2023.1

Release Date: January 26, 2023

Important Information

Teledyne Signal Processing Devices Sweden AB (SP Devices) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to SP Devices' general terms and conditions supplied at the time of order acknowledgment.

SP Devices warrants that each product will be free of defects in materials and workmanship, and conform to specifications set forth in published data sheets, for a period of one (1) year. The warranty commences on the date the product is shipped by SP Devices. SP Devices' sole liability and responsibility under this warranty is to repair or replace any product which is returned to it by Buyer and which SP Devices determines does not conform to the warranty. Product returned to SP Devices for warranty service will be shipped to SP Devices at Buyer's expense and will be returned to Buyer at SP Devices' expense. SP Devices will have no obligation under this warranty for any products which (i) has been improperly installed; (ii) has been used other than as recommended in SP Devices' installation or operation instructions or specifications; or (iii) has been repaired, altered or modified by entities other than SP Devices. The warranty of replacement products shall terminate with the warranty of the product. Buyer shall not return any products for any reason without the prior written authorization of SP Devices.

In no event shall SP Devices be liable for any damages arising out of or related to this document or the information contained in it.

SP DEVICES' EXPRESS WARRANTY TO BUYER CONSTITUTES SP DEVICES' SOLE LIABILITY AND THE BUYER'S SOLE REMEDY WITH RESPECT TO THE PRODUCTS AND IS IN LIEU OF ALL OTHER WARRANTIES, LIABILITIES AND REMEDIES. EXCEPT AS THUS PROVIDED, SP DEVICES DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

SP DEVICES DOES NOT INDEMNIFY, NOR HOLD THE BUYER HARMLESS, AGAINST ANY LIABILITIES, LOSSES, DAMAGES AND EXPENSES (INCLUDING ATTORNEY'S FEES) RELATING TO ANY CLAIMS WHATSOEVER. IN NO EVENT SHALL SP DEVICES BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFIT, LOST DATA AND THE LIKE, DUE TO ANY CAUSE WHATSOEVER. NO SUIT OR ACTION SHALL BE BROUGHT AGAINST SP DEVICES MORE THAN ONE YEAR AFTER THE RELATED CAUSE OF ACTION HAS ACCRUED. IN NO EVENT SHALL THE ACCRUED TOTAL LIABILITY OF SP DEVICES FROM ANY LAWSUIT, CLAIM, WARRANTY OR INDEMNITY EXCEED THE AGGREGATE SUM PAID TO SP BY BUYER UNDER THE ORDER THAT GIVES RISE TO SUCH LAWSUIT, CLAIM, WARRANTY OR INDEMNITY.

Worldwide Sales and Technical Support
www.teledyne-spdevices.com

SP Devices Corporate Headquarters
Teknikringen 6
SE-583 30 Linköping
Sweden
Phone: +46 (0)13 465 0600
Fax: +46 (0)13 991 3044
Email: info@spdevices.com

Contents

1 Functions of the ADQAPI

The functions of the ADQAPI are categorized into three main sets.

ADQAPI Specific Functions

Purely related to the API itself and not to the operation of digitizers.

ADQControlUnit Functions

Interface with the device driver for tasks such as finding and initializing digitizers

ADQ Functions

Interface directly with a specific digitizer

The functions of the ADQAPI may be called in a number of ways. Typically, they are interfaced as C-functions, but they may also be interfaced directly through a C++ class object. If running Windows, the functions may also be interfaced through a Matlab mex file or the .NET framework.

1.1 C-API

When interfacing the C-API, all functions other than the ADQControlUnit Functions and the ADQAPI Specific Functions are called by prepending the function name with 'ADQ_' and adding the previously created ADQ-ControlUnit and the unit ID as inputs. Below is a simple example of how to setup a unit and call the 'Blink' function using the C-API:

```
void* adq_cu_ptr = CreateADQControlUnit(); // Creates an ADQControlUnit called adq_cu_ptr
if(adq_cu_ptr  != NULL)
{
  int nof_devices = ADQControlUnit_FindDevices(); // Finds and starts all devices, also returns the number started
  if (nof_devices > 0)
  {
    ADQ_Blink(adq_cu_ptr, 1); // Blinks one of the LEDs for the device number 1 (the first started)
  }
}
DeleteADQControlUnit(ADQCU);
```

The C-API may be used from several programming languages (e.g. Python has excellent support for this), which makes it the most general.

1.2 Cpp-API

Once an ADQControlUnit is created and at least one unit has been found, the function ADQControlUnit_GetADQ may be used to return a pointer to a C++ class object that can be used to access all functions that operates on the unit. These functions are then called with the name and inputs listed later in the document. Below is a simple example of how to set up a unit and call the 'Blink' function using the C++-API:

```
void* adq_cu_ptr = CreateADQControlUnit(); // Creates an ADQControlUnit called adq_cu_ptr
if(adq_cu_ptr  != NULL)
{
  int nof_devices = ADQControlUnit_FindDevices(); // Finds and starts all devices, also returns the number started
  if (nof_devices > 0)
  {
    ADQInterface* ADQDevice = ADQControlUnit_GetADQ(adq_cu_ptr, 1); // Sends command to device number 1 (the first
       started)
    ADQDevice->Blink(); // Blinks one of the device LEDs
  }
}
DeleteADQControlUnit(ADQCU);
```

The ADQControlUnit Functions and the ADQAPI Specific Functions are always interfaced as C-API functions.

For the C++ API it is very important to assure that header files linked when building the application and the actually loaded DLL are of the exact same revision, otherwise corrupt behaviour may be the result. Typically incompatibility is due to installing a new API on the computer but forgetting to update to the new header files in the application development environment. This can be checked in code with the helper macro IS_VALID_DLL_REVISION:

```
int apirev = ADQAPI_GetRevision();
if (!IS_VALID_DLL_REVISION(apirev))
{
   printf("ERROR: The linked header file and the loaded DLL are of different revisions. This may cause corrupt
       behavior.\n");
   return -1;
}
```

1.3 Matlab

For Windows, there is an interface that should feel familiar for Matlab users. This is implemented in the DLL named mex_ADQ.dll, which is installed with the other DLLs. For convenient interfacing, there is also a wrapper file called interface_ADQ.m.

1.4 dot-NET

For users of the .NET framework, the ADQAPI has been wrapped using SWIG. The wrapper DLLs are installed together with the Windows installer. In the .NET framework, the C-API functions are interfaced using a .NET class object. More information is found in the ADQAPI & .NET User's Guide.

2 ADQAPI Specific Functions

Enumerations

- enum ADQAPIObjectID {
  ADQAPI_OBJECT_RESERVED = 0 ,
  ADQAPI_OBJECT_ATD_WFA_STRUCT = 1 ,
  ADQAPI_OBJECT_ADQ_RECORD_HEADER = 2 ,
  ADQAPI_OBJECT_ADQ_INFO_LIST_ENTRY = 3 ,
  ADQAPI_OBJECT_ADQ_INFO_LIST_PRE_ALLO_ARRAY = 4 ,
  ADQAPI_OBJECT_SD_CARD_CONFIGURATION = 5 ,
  ADQAPI_OBJECT_ADQ_DAISY_CHAIN_TRIGGER_INFORMATION = 7 ,
  ADQAPI_OBJECT_ADQ_DAISY_CHAIN_DEVICE_INFORMATION = 8 }

  Object IDs for ADQAPI_GetObjectSize()

Functions

- uint32_t ADQAPI_GetRevision ()

  Gets the ADQAPI revision.
- int ADQAPI_ValidateVersion (int major, int minor)

  Validate the struct definitions in the ADQAPI.
- void ∗ CreateADQControlUnit ()

  Creates an ADQControlUnit.
- void DeleteADQControlUnit (void ∗adq_cu_ptr)

  Destroys and instance of ADQControlUnit.
- unsigned int ValidateDll ()

  Checks that a C++ application is compiled with the correct ADQAPI.h.

2.1 Detailed Description

Functions used for retreiving information about the ADQAPI and creating and deleting an ADQControlUnit, which is used to set up and control the ADQs.

2.2 Enumeration Type Documentation

2.2.1 ADQAPIObjectID

`enum ADQAPIObjectID`

Object IDs for ADQAPI_GetObjectSize()

Enumerator

ADQAPI_OBJECT_RESERVED	Unused
ADQAPI_OBJECT_ATD_WFA_STRUCT	struct ATDWFABufferStruct
ADQAPI_OBJECT_ADQ_RECORD_HEADER	struct ADQRecordHeader

Enumerator

ADQAPI_OBJECT_ADQ_INFO_LIST_ENTRY	struct ADQInfoListEntry
ADQAPI_OBJECT_ADQ_INFO_LIST_PRE_ALLO_ARRAY	struct ADQInfoListPreAlloArray
ADQAPI_OBJECT_SD_CARD_CONFIGURATION	struct SDCardConfiguration
ADQAPI_OBJECT_ADQ_DAISY_CHAIN_TRIGGER_INFORMATION	struct ADQDaisyChainTriggerInformation
ADQAPI_OBJECT_ADQ_DAISY_CHAIN_DEVICE_INFORMATION	struct ADQDaisyChainDeviceInformation

2.3 Function Documentation

2.3.1 ADQAPI_GetRevision()

```
uint32_t ADQAPI_GetRevision ( )
```

Gets the ADQAPI revision.

Returns

The revision of the ADQAPI

2.3.2 ADQAPI_ValidateVersion()

```
int ADQAPI_ValidateVersion (
    int major,
    int minor )
```

Validate the struct definitions in the ADQAPI.

This function allows an application to perform run-time validation of the struct definitions provided by the ADQAPI. An application should add a call to this function as:

```
switch (ADQAPI_ValidateVersion(ADQAPI_VERSION_MAJOR,
                               ADQAPI_VERSION_MINOR))
{
case 0:
  // Compatible
  break;
case -1:
  // Incompatible: the application needs to be recompiled and relinked
  // with the installed ADQAPI. This is equivalent to an error.
  return;
case -2:
  // Backwards compatible: it's recommended to recompile and relink the
  // application with the installed ADQAPI. This is equivalent to a warning.
  break;
}
```

This function call will be added to the binary with static arguments. If an application is ever executed on a system with an incompatible ADQAPI, this function will return an error and allow you to exit with a message informing the user that the application needs to be recompiled and relinked with the new ADQAPI.

Parameters

major	The major version number. Should be set to `ADQAPI_VERSION_MAJOR`.
minor	The minor version number. Should be set to `ADQAPI_VERSION_MINOR`.

Returns

0 if compatible, -1 if backwards compatible, -2 if incompatible.

2.3.3 CreateADQControlUnit()

```
void* CreateADQControlUnit ( )
```

Creates an ADQControlUnit.

Creates an instance of ADQControlUnit, that may be used to find and setup ADQ devices

Returns

A pointer to the ADQControlUnit

Note

Only call this function once for stable behaviour

See also

DeleteADQControlUnit()

2.3.4 DeleteADQControlUnit()

```
void DeleteADQControlUnit (
    void *            adq_cu_ptr )
```

Destroys and instance of ADQControlUnit.

Parameters

adq_cu_ptr	Pointer to the control unit that is to be destroyed

See also

CreateADQControlUnit()

2.3.5 ValidateDll()

```
unsigned int ValidateDll ( )
```

Checks that a C++ application is compiled with the correct ADQAPI.h.

Only usable with the C++ API. Use one of the following macros to check the validity:

VALIDATE_DLL(ADQInterface∗ p)
Exits the application on failure

IS_VALID_DLL(ADQInterface∗ p)
Returns 1 for valid dll and 0 otherwise

Returns

0x11AABEEF if no error was found

3 ADQControlUnit Functions

Data Structures

- struct ADQInfoListEntry

  Info list structure returned by ADQControlUnit_ListDevices(). More...
- struct ADQInfoListPreAlloArray

  Structure with list of ADQInfoListEntries for use with ADQControlUnit_ListDevices(). More...

Functions

- unsigned int ADQControlUnit_ClearLastFailedDeviceError (void ∗adq_cu_ptr)

  Clear last failed device error.
- void ADQControlUnit_DeleteADQ (void ∗adq_cu_ptr, int ADQ_num)

  Deletes an ADQ object.
- unsigned int ADQControlUnit_EnableErrorTrace (void ∗adq_cu_ptr, unsigned int trace_level, const char ∗trace_file_dir)

  Enables error logging to file.
- unsigned int ADQControlUnit_EnableErrorTraceAppend (void ∗adq_cu_ptr, unsigned int trace_level, const char ∗trace_file_dir)

  Enables error logging and appends to earlier file.
- int ADQControlUnit_EnableEthernetScan (void ∗adq_cu_ptr, int eth_scn)

  Enables Ethernet communication.
- int ADQControlUnit_FindDevices (void ∗adq_cu_ptr)

  Finds and starts all devices.
- ADQInterface ∗ ADQControlUnit_GetADQ (void ∗adq_cu_ptr, int adq_num)

  Gets the pointer to a specific ADQ device.
- int ADQControlUnit_GetFailedDeviceCount (void ∗adq_cu_ptr)

  Gets the number of units that failed startup.
- unsigned int ADQControlUnit_GetLastFailedDeviceError (void ∗adq_cu_ptr)

  Get last failed device error.
- unsigned int ADQControlUnit_GetLastFailedDeviceErrorWithText (void ∗adq_cu_ptr, char ∗errstr)

  Get last failed device error (together with error code in clear textual form)

- unsigned int ADQControlUnit_ListDevices (void ∗adq_cu_ptr, struct ADQInfoListEntry ∗∗retList, unsigned int ∗retLen)

  Lists devices connected to the system.
- int ADQControlUnit_NofADQ (void ∗adq_cu_ptr)

  Gets the number of ADQs.
- int ADQControlUnit_NofADQ108 (void ∗adq_cu_ptr)

  Gets the number of ADQ108.
- int ADQControlUnit_NofADQ112 (void ∗adq_cu_ptr)

  Gets the number of ADQ112.
- int ADQControlUnit_NofADQ114 (void ∗adq_cu_ptr)

  Gets the number of ADQ114.
- int ADQControlUnit_NofADQ12 (void ∗adq_cu_ptr)

  Gets the number of ADQ12.
- int ADQControlUnit_NofADQ14 (void ∗adq_cu_ptr)

  Gets the number of ADQ14.
- int ADQControlUnit_NofADQ1600 (void ∗adq_cu_ptr)

  Gets the number of ADQ1600.
- int ADQControlUnit_NofADQ208 (void ∗adq_cu_ptr)

  Gets the number of ADQ208.
- int ADQControlUnit_NofADQ212 (void ∗adq_cu_ptr)

  Gets the number of ADQ212.
- int ADQControlUnit_NofADQ214 (void ∗adq_cu_ptr)

  Gets the number of ADQ214.
- int ADQControlUnit_NofADQ412 (void ∗adq_cu_ptr)

  Gets the number of ADQ412.
- int ADQControlUnit_NofADQ7 (void ∗adq_cu_ptr)

  Gets the number of ADQ7.
- int ADQControlUnit_NofADQ8 (void ∗adq_cu_ptr)

  Gets the number of ADQ8.
- int ADQControlUnit_NofADQDSP (void ∗adq_cu_ptr)

  Gets the number of ADQDSP.
- int ADQControlUnit_NofEV12AS350_EVM (void ∗adq_cu_ptr)

  Gets the number of EV12AS350_EVM.
- int ADQControlUnit_NofSDR14 (void ∗adq_cu_ptr)

  Gets the number of SDR14.
- int ADQControlUnit_NofSphinxAA (void ∗adq_cu_ptr)

  Gets the number of SphinxAA.
- unsigned int ADQControlUnit_OpenDeviceInterface (void ∗adq_cu_ptr, int ADQInfoListEntryNumber)

  Opens an interface to a specific device.

- unsigned int ADQControlUnit_ResetDevice (void ∗adq_cu_ptr, int ADQInfoListEntryNumber, int level)
- int ADQControlUnit_SetGeneralParameter (void ∗adq_cu_ptr, int param_index, int param_value)

  Sets general parameters.
- unsigned int ADQControlUnit_SetupDevice (void ∗adq_cu_ptr, int ADQInfoListEntryNumber)
- unsigned int ADQControlUnit_UserLogMessage (void ∗adq_cu_ptr, unsigned int trace_level, const char ∗message,...)

  Used by the user to add custom log messages when using ADQAPI.
- unsigned int ADQControlUnit_UserLogMessageAtLine (void ∗adq_cu_ptr, unsigned int trace_level, const char ∗loc_file, const char ∗loc_func, const int loc_line, const char ∗message,...)

  Used by the user to add custom log messages when using ADQAPI.

3.1 Detailed Description

These ADQControlUnit is used to find, setup, and delete devices objects.

3.2 Data Structure Documentation

3.2.1 struct ADQInfoListEntry

Info list structure returned by ADQControlUnit_ListDevices().

Data Fields

unsigned int	AddressField1	
unsigned int	AddressField2	
char	DevFile[64]	
unsigned int	DeviceInterfaceOpened	
unsigned int	DeviceSetupCompleted	
enum ADQHWIFEnum	HWIFType	
enum ADQProductID_Enum	ProductID	
unsigned int	VendorID	

3.2.2 struct ADQInfoListPreAlloArray

Structure with list of ADQInfoListEntries for use with ADQControlUnit_ListDevices().

Data Fields

struct ADQInfoListEntry	ADQlistArray[128]	

3.3 Function Documentation

3.3.1 ADQControlUnit_ClearLastFailedDeviceError()

```
unsigned int ADQControlUnit_ClearLastFailedDeviceError (
    void *          adq_cu_ptr )
```

Clear last failed device error.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

1 for successful operation and 0 for failure

See also

ADQControlUnit_EnableErrorTrace(), ADQControlUnit_GetLastFailedDeviceError()

3.3.2 ADQControlUnit_DeleteADQ()

```
void ADQControlUnit_DeleteADQ (
    void *          adq_cu_ptr,
    int             ADQ_num )
```

Deletes an ADQ object.

This function will rearrange the list of ADQ devices and a given number for an ADQ device will maybe no longer refer to the same object as before.
`1 <= ADQ_num <= NofADQ`

Parameters

adq_cu_ptr	The ADQControlUnit instance
ADQ_num	The number of the ADQ device to delete

Note

Uses 1-based index (reasons of legacy)

See also

ADQControlUnit_FindDevices()

3.3.3 ADQControlUnit_EnableErrorTrace()

```
unsigned int ADQControlUnit_EnableErrorTrace (
    void *          adq_cu_ptr,
    unsigned int    trace_level,
```

```
    const char *        trace_file_dir )
```

Enables error logging to file.

Enables log file output from the connected devices and the ADQControlUnit. Each device opens a separate log file.

Parameters

adq_cu_ptr The ADQControlUnit instance
trace_level Trace level ▪ trace_level = 0 : No error logging ▪ trace_level = 1 : Error logging ▪ trace_level = 2 : Error and warnings logging ▪ trace_level = 3 : Error, warning, info logging Additionally if bit 11 is set (e.g. by ORing 2048 with the trace_level), timestamping will be enabled in the log.
trace_file_dir Path to the directory to put the log files in. If this path points to a file, all trace will be appended to that single file instead.

Returns

1 for successful operation and 0 for failure

Note

Windows style directory separator '\' should be escaped by using '\' instead. devices to get a single, non-conflicting log file as the result.

See also

ADQControlUnit_EnableErrorTraceAppend()

3.3.4 ADQControlUnit_EnableErrorTraceAppend()

```
unsigned int ADQControlUnit_EnableErrorTraceAppend (
    void *              adq_cu_ptr,
    unsigned int        trace_level,
    const char *        trace_file_dir )
```

Enables error logging and appends to earlier file.

Enables log file output from the connected devices and the ADQControlUnit. Each device opens a separate log file. The difference between this function and ADQControlUnit_EnableErrorTrace() is that this function appends the log outputs to any previously created log files.

Parameters

adq_cu_ptr The ADQControlUnit instance	
trace_level Trace level • trace_level = 0 : No error logging • trace_level = 1 : Error logging • trace_level = 2 : Error and warnings logging • trace_level = 3 : Error, warning, info logging Additionally if bit 11 is set (e.g. by ORing 2048 with the trace_level), timestamping will be enabled in the log.	
trace_file_dir Path to the directory to put the log files in	

Returns

1 for successful operation and 0 for failure

See also

ADQControlUnit_EnableErrorTrace()

3.3.5 ADQControlUnit_EnableEthernetScan()

```
int ADQControlUnit_EnableEthernetScan (
    void *          adq_cu_ptr,
    int             eth_scn )
```

Enables Ethernet communication.

EnableEthernetScan() enables lookup of ADQ units connected over Ethernet. Set to 1 to search for ADQ14, 2 to search for ADQ7 and 3 to search for ADQ12.

EnableEthernetScan() has to be run before the device listing functions, e.g. ADQControlUnit_ListDevices(), ADQControlUnit_OpenDeviceInterface(), ADQControlUnit_SetupDevice()

Parameters

adq_cu_ptr The ADQControlUnit instance	
eth_scn Enable ethernet scan. • 0: Disable scan (default) • 1: Enable lookup of ADQ14 • 2: Enable lookup of ADQ7 • 3: Enable lookup of ADQ12	

Returns

Always returns 1

See also

ADQControlUnit_ListDevices(), ADQControlUnit_OpenDeviceInterface(), ADQControlUnit_SetupDevice()

3.3.6 ADQControlUnit_FindDevices()

```
int ADQControlUnit_FindDevices (
    void *            adq_cu_ptr )
```

Finds and starts all devices.

Finds all ADQ units connected to the computer and creates/updates a list of all ADQs within the input ADQControlUnit. The order of the devices is determined by their USB bus addresses and/or their PXIe address.

Parameters

adq_cu_ptr The control unit that will be used to control the units.

Returns

The total number of ADQs found.

Note

If it is not desired to start all units at once, ADQControlUnit_SetupDevice may be used to start a specific device.

See also

CreateADQControlUnit(), ADQControlUnit_SetupDevice()

3.3.7 ADQControlUnit_GetADQ()

```
ADQInterface* ADQControlUnit_GetADQ (
    void *            adq_cu_ptr,
    int               adq_num )
```

Gets the pointer to a specific ADQ device.

The pointer may be used to interface the device as a class object in C++.

Parameters

adq_cu_ptr The ADQControlUnit instance
adq_num The number of the ADQ device

Returns

A pointer to to the ADQInterface object for the ADQ.

Note

Uses 1-based index (reasons of legacy)

3.3.8 ADQControlUnit_GetFailedDeviceCount()

```
int ADQControlUnit_GetFailedDeviceCount (
    void *            adq_cu_ptr )
```

Gets the number of units that failed startup.

After a call to ADQControlUnit_FindDevices this function returns the number of units found, which were not possible to start correctly (error reported during start of device).

If zero is returned no devices failed to start.

Cause of failure can be one of:

- Incompatible HW device version
- Power-off during setup phase
- Malfunctioning FPGA code (if used with ADQ Development Kit)

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

The number of units that failed to start

See also

ADQControlUnit_GetLastFailedDeviceError(), ADQControlUnit_FindDevices()

3.3.9 ADQControlUnit_GetLastFailedDeviceError()

```
unsigned int ADQControlUnit_GetLastFailedDeviceError (
    void *            adq_cu_ptr )
```

Get last failed device error.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

The last error code from the last failing device

See also

ADQControlUnit_EnableErrorTrace(), ADQControlUnit_ClearLastFailedDeviceError()

3.3.10 ADQControlUnit_GetLastFailedDeviceErrorWithText()

```
unsigned int ADQControlUnit_GetLastFailedDeviceErrorWithText (
    void *            adq_cu_ptr,
    char *            errstr )
```

Get last failed device error (together with error code in clear textual form)

Parameters

adq_cu_ptr	The ADQControlUnit instance
errstr	char buffer with of at least size 256

Returns

The last error code and error text from the last failing device

See also

ADQControlUnit_EnableErrorTrace(), ADQControlUnit_ClearLastFailedDeviceError()

3.3.11 ADQControlUnit_ListDevices()

```
unsigned int ADQControlUnit_ListDevices (
    void *            adq_cu_ptr,
    struct
    ADQInfoListEntry
    **                retList,
    unsigned int *    retLen )
```

Lists devices connected to the system.

The ListDevices/OpenDeviceInterface/SetupDevice functions are intended as a more versatile replacement for FindDevices.

ListDevices creates a list of available ADQ devices without attempting to boot any firmware or set up any communication channels.

The function requires pointers to a list pointer and a length integer to be provided. The list is then returned as an array which can be indexed from retList[0] to retList[retLen-1], with each entry corresponding to an ADQ device.

The ADQInfoListEntry structure is found in the ADQAPI.h header file and contains all information which can be read non-destructively from the device:

```
struct ADQInfoListEntry
{
```

```
enum ADQHWIFEnum HWIFType;
enum ADQProductID_Enum ProductID;
unsigned int VendorID;
unsigned int AddressField1;
unsigned int AddressField2;
char DevFile[64];
unsigned int DeviceInterfaceOpened;
unsigned int DeviceSetupCompleted;
};
enum ADQProductID_Enum {
PID_ADQ214 = 0x0001,
PID_ADQ114 = 0x0003,
PID_ADQ112 = 0x0005,
PID_SphinxHS = 0x000B,
PID_SphinxLS = 0x000C,
PID_ADQ108 = 0x000E,
PID_ADQDSP = 0x000F,
PID_SphinxAA14 = 0x0011,
PID_SphinxAA16 = 0x0012,
PID_ADQ412 = 0x0014,
PID_ADQ212 = 0x0015,
PID_SphinxAA_LS2 = 0x0016,
PID_SphinxHS_LS2 = 0x0017,
PID_SDR14 = 0x001B,
PID_ADQ1600 = 0x001C,
PID_SphinxXT = 0x001D,
PID_ADQ208 = 0x001E,
PID_ADQ14 = 0x0020,
PID_EV12AS350_EVM = 0x0022,
PID_ADQ7 = 0x0023,
PID_ADQ8 = 0x0026,
PID_ADQ12 = 0x0027,
};
    enum ADQHWIFEnum {
    HWIF_USB,
    HWIF_PCIE,
    HWIF_USB3
    };
```

Parameters

adq_cu_ptr	The ADQControlUnit instance
retList	Pointer to list pointers. The list is returned as an array which can be indexed from retList[0] to retList[retLen-1], with each entry corresponding to an ADQ device.
retLen	Length of the number of list elements that may be returned maximum. This is used to ensure that the function doesn't write outside of the allocated space.

Returns

1 for successful operation and 0 for failure

See also

ADQControlUnit_OpenDeviceInterface(), ADQControlUnit_SetupDevice()

3.3.12 ADQControlUnit_NofADQ()

```
int ADQControlUnit_NofADQ (
    void *            adq_cu_ptr )
```

Gets the number of ADQs.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQs found

3.3.13 ADQControlUnit_NofADQ108()

```
int ADQControlUnit_NofADQ108 (
    void *           adq_cu_ptr )
```

Gets the number of ADQ108.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ108 found

3.3.14 ADQControlUnit_NofADQ112()

```
int ADQControlUnit_NofADQ112 (
    void *           adq_cu_ptr )
```

Gets the number of ADQ112.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ112 found

3.3.15 ADQControlUnit_NofADQ114()

```
int ADQControlUnit_NofADQ114 (
    void *           adq_cu_ptr )
```

Gets the number of ADQ114.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ114 found

3.3.16 ADQControlUnit_NofADQ12()

```
int ADQControlUnit_NofADQ12 (
    void *            adq_cu_ptr )
```

Gets the number of ADQ12.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ12 found

3.3.17 ADQControlUnit_NofADQ14()

```
int ADQControlUnit_NofADQ14 (
    void *            adq_cu_ptr )
```

Gets the number of ADQ14.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ14 found

3.3.18 ADQControlUnit_NofADQ1600()

```
int ADQControlUnit_NofADQ1600 (
    void *            adq_cu_ptr )
```

Gets the number of ADQ1600.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ1600 found

3.3.19 ADQControlUnit_NofADQ208()

```
int ADQControlUnit_NofADQ208 (
    void *          adq_cu_ptr )
```

Gets the number of ADQ208.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ208 found

3.3.20 ADQControlUnit_NofADQ212()

```
int ADQControlUnit_NofADQ212 (
    void *          adq_cu_ptr )
```

Gets the number of ADQ212.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ212 found

3.3.21 ADQControlUnit_NofADQ214()

```
int ADQControlUnit_NofADQ214 (
    void *          adq_cu_ptr )
```

Gets the number of ADQ214.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ214 found

3.3.22 ADQControlUnit_NofADQ412()

```
int ADQControlUnit_NofADQ412 (
    void *          adq_cu_ptr )
```

Gets the number of ADQ412.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ412 found

3.3.23 ADQControlUnit_NofADQ7()

```
int ADQControlUnit_NofADQ7 (
    void *          adq_cu_ptr )
```

Gets the number of ADQ7.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ7 found

3.3.24 ADQControlUnit_NofADQ8()

```
int ADQControlUnit_NofADQ8 (
    void *          adq_cu_ptr )
```

Gets the number of ADQ8.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQ8 found

3.3.25 ADQControlUnit_NofADQDSP()

```
int ADQControlUnit_NofADQDSP (
    void *              adq_cu_ptr )
```

Gets the number of ADQDSP.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of ADQDSP found

3.3.26 ADQControlUnit_NofEV12AS350_EVM()

```
int ADQControlUnit_NofEV12AS350_EVM (
    void *              adq_cu_ptr )
```

Gets the number of EV12AS350_EVM.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of EV12AS350_EVM found

3.3.27 ADQControlUnit_NofSDR14()

```
int ADQControlUnit_NofSDR14 (
    void *              adq_cu_ptr )
```

Gets the number of SDR14.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of SDR14 found

3.3.28 ADQControlUnit_NofSphinxAA()

```
int ADQControlUnit_NofSphinxAA (
    void *          adq_cu_ptr )
```

Gets the number of SphinxAA.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Returns

the number of SphinxAA found

3.3.29 ADQControlUnit_OpenDeviceInterface()

```
unsigned int ADQControlUnit_OpenDeviceInterface (
    void *          adq_cu_ptr,
    int             ADQInfoListEntryNumber )
```

Opens an interface to a specific device.

After running ListDevices and finding an entry of interest in the device list, OpenDeviceInterface is used to open a communications channel towards the device.

The ADQInfoListEntryNumber argument should be the array index of the listdevices entry you want to open, i.e. if you want to open the device corresponding to retList[0], pass 0 to this function.

Using this function will add an ADQ object to the internal lists of the ADQControlUnit. This means that the ADQ will show up when using functions such as ADQControlUnit_GetADQ or ADQControlUnit_NofADQ, etc. Simple tasks such as reading and writing registers can be done at this stage, but data collection and similar requires ADQControlUnit_SetupDevice() to be run also.

Please note that the device number when using GetADQ/NofADQ/etc will not have anything to do with the index number used in this function.

Parameters

adq_cu_ptr	The ADQControlUnit instance
ADQInfoListEntryNumber	Array index of the listdevices entry you want to open

Returns

1 for successful operation and 0 for failure

See also

ADQControlUnit_ListDevices(), ADQControlUnit_SetupDevice()

3.3.30 ADQControlUnit_ResetDevice()

```
unsigned int ADQControlUnit_ResetDevice (
    void *              adq_cu_ptr,
    int                 ADQInfoListEntryNumber,
    int                 level )
```

This function can be used to issue a reset command to the digitizer without setting up the device. The device needs to be opened using OpenDeviceInterface().

This currently only works specifically with USB3 and reset level 18.

After resetting the device the device will be closed, a new call to ListDevices may be needed after reset.

Parameters

adq_cu_ptr	The ADQControlUnit instance
ADQInfoListEntryNumber	Array index of the listdevices entry you want to open
level	Level of reset to issue

Returns

1 for successful operation and 0 for failure

Note

Zero-based index used. When using later for adq_num in C API or C++ API, 1-based index is used so you need to add 1 to the index.

See also

ADQControlUnit_ListDevices(), ADQControlUnit_OpenDeviceInterface()

3.3.31 ADQControlUnit_SetGeneralParameter()

```
int ADQControlUnit_SetGeneralParameter (
    void *        adq_cu_ptr,
    int           param_index,
    int           param_value )
```

Sets general parameters.

SetGeneralParameter sets parameters that is used when starting devices with SetupDevice. It is intended to be used only when default parameters do not work well enough for the system used. There is no check that the parameter is valid or parameter value is valid.

Parameters

adq_cu_ptr	The ADQControlUnit instance
param_index	The index of the parameter to set
param_value	The value of the parameter

Returns

Always returns 1 if there is available room for new parameter

See also

ADQControlUnit_SetupDevice()

3.3.32 ADQControlUnit_SetupDevice()

```
unsigned int ADQControlUnit_SetupDevice (
    void *        adq_cu_ptr,
    int           ADQInfoListEntryNumber )
```

After running ListDevices and having used OpenDeviceInterface to open a communication channel towards a specific device, this function is used to do everything necessary to make the device ready for use, such as initializing API variables, calibrating PLLs, calibrating ADC data interfaces, resetting internal logic, etc. After this, the digitizer is ready for use.

This function takes the same index number as was used with OpenDeviceInterface, i.e. the ListDevices array index corresponding to your device.

Please note that the device number when using GetADQ/NofADQ/etc will not have anything to do with the index number used in this function.

Parameters

adq_cu_ptr	The ADQControlUnit instance

Parameters

ADQInfoListEntryNumber Array index of the listdevices entry you want to open

Returns

1 for successful operation and 0 for failure

Note

Zero-based index used. When using later for adq_num in C API or C++ API, 1-based index is used so you need to add 1 to the index.

See also

ADQControlUnit_ListDevices(), ADQControlUnit_OpenDeviceInterface()

3.3.33 ADQControlUnit_UserLogMessage()

```
unsigned int ADQControlUnit_UserLogMessage (
    void *            adq_cu_ptr,
    unsigned int      trace_level,
    const char *      message,
     ... )
```

Used by the user to add custom log messages when using ADQAPI.

The log messages will be writen to the same file and path given when calling ADQControlUnit_EnableErrorTrace.

Parameters

adq_cu_ptr The ADQControlUnit instance.
trace_level Trace level • trace_level = 0 : Message will be printed out without a tag • trace_level = 1 : Message will be tagged as Error • trace_level = 2 : Message will be tagged as Warning • trace_level = 3 : Message will be tagged as Info
message A string containing the log message.

Returns

1 for successful operation and 0 for failure

Note

The input message should be terminated with newline (\n) in order to appear immediately in ADQUpdaterGUI. Otherwise the output message may be delayed until the next newline character.

See also

ADQControlUnit_EnableErrorTrace()

3.3.34 ADQControlUnit_UserLogMessageAtLine()

```
unsigned int ADQControlUnit_UserLogMessageAtLine (
    void *             adq_cu_ptr,
    unsigned int       trace_level,
    const char *       loc_file,
    const char *       loc_func,
    const int          loc_line,
    const char *       message,
     ... )
```

Used by the user to add custom log messages when using ADQAPI.

The log messages will be writen to the same file and path given when calling ADQControlUnit_EnableErrorTrace.

Parameters

adq_cu_ptr The ADQControlUnit instance.
trace_level Trace level • trace_level = 0 : Message will be printed out without a tag • trace_level = 1 : Message will be tagged as Error • trace_level = 2 : Message will be tagged as Warning • trace_level = 3 : Message will be tagged as Info
loc_file Should be set to **FILE**
loc_func Should be set to **FUNCTION**
loc_line Should be set to **LINE**
message A string containing the log message.

Returns

1 for successful operation and 0 for failure

Note

The input message should be terminated with newline (
) in order to appear immediately in ADQUpdaterGUI. Otherwise the output message may be delayed until the next newline character.

See also

ADQControlUnit_EnableErrorTrace()

4 ADQ Hardware and Firmware Info

Enumerations

- enum ADQHWIFEnum {
  HWIF_USB = 0 ,
  HWIF_PCIE = 1 ,
  HWIF_USB3 = 2 ,
  HWIF_PCIELITE = 3 ,
  HWIF_ETH_ADQ7 = 4 ,
  HWIF_ETH_ADQ14 = 5 ,
  HWIF_VIRTUAL = 6 ,
  HWIF_QPCIE = 7 ,
  HWIF_OTHER = 8 }

  IDs for different hardware interface types.
- enum ADQProductID_Enum {
  PID_ADQ214 = 0x0001 ,
  PID_ADQ114 = 0x0003 ,
  PID_ADQ112 = 0x0005 ,
  PID_SphinxHS = 0x000B ,
  PID_SphinxLS = 0x000C ,
  PID_ADQ108 = 0x000E ,
  PID_ADQDSP = 0x000F ,
  PID_SphinxAA14 = 0x0011 ,
  PID_SphinxAA16 = 0x0012 ,
  PID_ADQ412 = 0x0014 ,
  PID_ADQ212 = 0x0015 ,
  PID_SphinxAA_LS2 = 0x0016 ,
  PID_SphinxHS_LS2 = 0x0017 ,
  PID_SDR14 = 0x001B ,
  PID_ADQ1600 = 0x001C ,
  PID_SphinxXT = 0x001D ,
  PID_ADQ208 = 0x001E ,
  PID_DSU = 0x001F ,
  PID_ADQ14 = 0x0020 ,
  PID_SDR14RF = 0x0021 ,
  PID_EV12AS350_EVM = 0x0022 ,
  PID_ADQ7 = 0x0023 ,
  PID_ADQ8 = 0x0026 ,
  PID_ADQ12 = 0x0027 ,
  PID_ADQ7Virtual = 0x0030 ,
  PID_ADQ32 = 0x0031 ,
  PID_ADQSM = 0x0032 ,
  PID_ADQ36 = 0x0033 ,
  PID_TX320 = 0x201A ,
  PID_RX320 = 0x201C ,
  PID_S6000 = 0x2019 }

Product IDs for different device types.

Functions

- int Blink ()

  Blinks a LED to identify the unit.
- const char ∗ GetADQDSPOption ()

  Gets the motherboard option.
- int GetADQType ()

  Gets the ADQ type as an integer.
- char ∗ GetBoardProductName ()

  Gets the product name of the ADQ.
- char ∗ GetBoardSerialNumber ()

  Returns the serial number of the ADQ device.
- int GetCalibrationInformation (unsigned int index, int ∗int_info, char ∗str_info)

  Returns specified calibration information from a device.
- const char ∗ GetCardOption ()

  Gets the card option string.
- unsigned int GetHardwareAssemblyPartNumber (char ∗partnum)

  Read the hardware assembly part number of the device.
- unsigned int GetHardwareSubassemblyPartNumber (char ∗partnum)

  Read the hardware sub-assembly part number of the device.
- const char ∗ GetNGCPartNumber ()

  Gets the part number of firmware framework.
- unsigned int GetPCBAssemblyPartNumber (char ∗partnum)

  Read the PCB assembly part number of the device.
- unsigned int GetPCBPartNumber (char ∗partnum)

  Read the PCB part number of the device.
- unsigned int GetPCIeAddress ()

  Gets the bus address of the a PCIe, PXIe or MTCA unit.
- unsigned int GetProductFamily (unsigned int ∗family)

  Gets the product family of the device.
- unsigned int GetProductID ()

  Gets the product ID of the unit.
- unsigned int GetProductVariant (unsigned int ∗ProductVariant)

  Gets an integer describing the Product Variant.
- uint32_t ∗ GetRevision ()

  Returns the firmware revision numbers of the device.
- unsigned int GetUSBAddress ()

  Gets the bus address of the a USB unit.

- int HasFeature (const char *featurename)

  Check for card feature.
- int IsMTCADevice ()

  Checks whether device is connected over a MTCA interface.
- int IsPCIeDevice ()

  Checks whether device is connected over a PCIe/PXIe/MTCA interface.
- int IsPCIeLiteDevice ()

  Checks whether device is connected over a PCIe/PXIe/MTCA interface with the Lite driver stack.
- unsigned int IsStartedOK ()

  Checks whether the device has started up OK.
- int IsUSB3Device ()

  Checks whether device is connected over a USB3 interface.
- int IsUSBDevice ()

  Checks whether device is connected over a USB2 interface.

4.1 Detailed Description

These functions give information about the ADQ device. Values returned by these functions are constant for the hardware or firmware, and does thus not change during operation of the unit.

4.2 Enumeration Type Documentation

4.2.1 ADQHWIFEnum

`enum ADQHWIFEnum`

IDs for different hardware interface types.

Enumerator

HWIF_USB	USB2
HWIF_PCIE	PCIe
HWIF_USB3	USB3
HWIF_PCIELITE	PCIe lite
HWIF_ETH_ADQ7	10Gb Ethernet for ADQ7
HWIF_ETH_ADQ14	10Gb Ethernet for ADQ7
HWIF_VIRTUAL	Virtual device
HWIF_QPCIE	PCIe
HWIF_OTHER	Reserved

4.3 Function Documentation

4.3.1 Blink()

```
virtual int Blink ( )
```

Blinks a LED to identify the unit.

Makes the green status LED on the board front panel blink on and off. This can for example be used to identify a specific digitizer in a multi-digitizer system.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ112, ADQ114, ADQ1600, ADQ208, ADQ212, ADQ214, ADQ412, ADQDSP, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

4.3.2 GetADQDSPOption()

```
virtual const char* GetADQDSPOption ( )
```

Gets the motherboard option.

Returns

A null terminated string containing motherboard options

Valid for

ADQ412, ADQ1600, SDR14, ADQ108, ADQ208, ADQDSP

See also

GetCardOption()

4.3.3 GetADQType()

```
virtual int GetADQType ( )
```

Gets the ADQ type as an integer.

Returns

An integer describing the unit, for example 412 for ADQ412.

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetProductFamily()

4.3.4 GetBoardProductName()

```
virtual char* GetBoardProductName ( )
```

Gets the product name of the ADQ.

Returns

A NULL-terminated string containing the product number. The returned field is 32 positions long.

Valid for

ADQ412,ADQ1600, ADQ212, ADQ108, ADQ208,ADQ112, ADQ114, ADQ214, ADQDSP, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetBoardSerialNumber()

4.3.5 GetBoardSerialNumber()

```
virtual char* GetBoardSerialNumber ( )
```

Returns the serial number of the ADQ device.

Returns

a char* pointer to a 16 bytes long string (null-terminated).

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

4.3.6 GetCalibrationInformation()

```
virtual int GetCalibrationInformation (
    unsigned int    index,
    int *           int_info,
    char *          str_info )
```

Returns specified calibration information from a device.

Parameters

index

The index of the calibration item to read. Dependent on item, the result will be returned as an integer or a string Unavailable/unprogrammed information will return NULL or 'UNKNOWN'.

- 100: Factory calibration date string, format YYYY-MM-DD
- 101: Last recalibration date string, format YYYY-MM-DD
- 102: Last field calibration date string, format YYYY-MM-DD
- 103: Last product validation date string, format YYYY-MM-DD
- 104: Registered in service date string, format YYYY-MM-DD
- 1101: Last recalibration type unsigned 32b integer
- 1102: Last field calibration type unsigned 32b integer
- 1103: Modification counter unsigned 32b integer

Parameters

int_info	NULL or pointer to a 32b integer container
str_info	NULL or pointer to a string container (needs to be able to carry a maximum of 256 byte string)

Returns

1 for success and 0 for error

Valid for

ADQ7

4.3.7 GetCardOption()

```
virtual const char* GetCardOption ( )
```

Gets the card option string.

Example: "-3G" for ADQ412 specifies ADQ412-3G card option.

Returns

A null terminated string containing the card option

Valid for

ADQ412, ADQ1600, SDR14, ADQ208, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetADQDSPOption(), GetBoardProductName()

4.3.8 GetHardwareAssemblyPartNumber()

```
virtual unsigned int GetHardwareAssemblyPartNumber (
    char *            partnum )
```

Read the hardware assembly part number of the device.

Parameters

partnum	Pointer to a 16-byte character array for storing the part number string:

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ8

4.3.9 GetHardwareSubassemblyPartNumber()

```
virtual unsigned int GetHardwareSubassemblyPartNumber (
    char *            partnum )
```

Read the hardware sub-assembly part number of the device.

Parameters

partnum	Pointer to a 16-byte character array for storing the part number string:

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ8

4.3.10 GetNGCPartNumber()

```
virtual const char* GetNGCPartNumber ( )
```

Gets the part number of firmware framework.

This part number cannot be modified from inside an ADQ DevKit (apart from replacing the framework NGC).

Returns

A NULL-terminated string, consisting of three three-digit numbers followed by a revision letter. For example, 400-200-002-A.

Note

Older firmware revisions do not contain part number registers and will always be read out as 000-000-000-A.

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQDSP, SDR14

See also

GetUserLogicPartNumber()

4.3.11 GetPCBAssemblyPartNumber()

```
virtual unsigned int GetPCBAssemblyPartNumber (
    char *            partnum )
```

Read the PCB assembly part number of the device.

Parameters

partnum Pointer to a 16-byte character array for storing the part number string:

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ8

4.3.12 GetPCBPartNumber()

```
virtual unsigned int GetPCBPartNumber (
    char *            partnum )
```

Read the PCB part number of the device.

Parameters

partnum Pointer to a 16-byte character array for storing the part number string:

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ8

4.3.13 GetPCIeAddress()

```
virtual unsigned int GetPCIeAddress ( )
```

Gets the bus address of the a PCIe, PXIe or MTCA unit.

Returns

The address, or 0 if the ADQ was not connected PCIe, PXIe, or MTCA. The address is a 32-bit unsigned integer containing: Bits 31-16: Bus number Bits 15-0: Slot number

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetUSBAddress()

4.3.14 GetProductFamily()

```
virtual unsigned int GetProductFamily (
    unsigned int *    family )
```

Gets the product family of the device.

Parameters

family	Pointer to where the result is to be stored. This is the meaning of the valid values: ▪ 1: Reserved ▪ 5: V5 family, (ADQ214, ADQ114, ADQ212, ADQ112) ▪ 6: V6 family, (ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, SDR14TX, ADQDSP) ▪ 7: 7 family, (ADQ12, ADQ14, ADQ7, ADQ8)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, SDR14TX, ADQDSP, ADQ214, ADQ212, ADQ114, ADQ112, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetADQType()

4.3.15 GetProductID()

```
virtual unsigned int GetProductID ( )
```

Gets the product ID of the unit.

Returns

the product ID of the unit

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQDSP, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetADQType

4.3.16 GetProductVariant()

```
virtual unsigned int GetProductVariant (
    unsigned int *    ProductVariant )
```

Gets an integer describing the Product Variant.

Parameters

ProductVariant
an allocated integer which receives the product variant integer. A return value of 0 means this parameter is undefined for the device ▪ On SDR14: 1: Standard SDR14 2: SDR14RF

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14

See also

GetCardOption()

4.3.17 GetRevision()

```
virtual uint32_t* GetRevision ( )
```

Returns the firmware revision numbers of the device.

Fields 0-2 contain information for FPGA #2(Comm FPGA) and fields 3-5 contain information for FPGA #1 (Alg FPGA). The returned field (int* revision) is 6 positions long and contains:

- revision[0 and 3]: revision number
- revision[1 and 4]:
  - 0: SVN Managed
  - 1: Local Copy
- revision[2 and 5]:
  - 0: SVN Updated
  - 1: Mixed Revision

Where revision is the returned pointer.

Returns

a pointer to a memory region containing six 32-bit unsigned integers

Note

V6 products do only have one FPGA and only revision[0-2] is valid

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetBoardProductName()

4.3.18 GetUSBAddress()

```
virtual unsigned int GetUSBAddress ( )
```

Gets the bus address of the a USB unit.

Returns

The address, or 0 if the ADQ was not connected over USB

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetPCIeAddress()

4.3.19 HasFeature()

```
virtual int HasFeature (
    const char *      featurename )
```

Check for card feature.

Check whether the card has a specific feature

Parameters

featurename

String containing the name of the feature to check for

- FWDAQ (ADQ8 only)
- FWATD (Advanced time-domain, Waveform averaging)
- FWPD (Pulse detection)
- FWSDR (Software defined radio)
- FW4DDC (Software defined radio with quad DDCs)
- UserLogicFilter (Configurable linear phase FIR filter)

Returns

0 if check failed, -1 if feature is not supported, 1 if feature is supported

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

4.3.20 IsMTCADevice()

```
virtual int IsMTCADevice ( )
```

Checks whether device is connected over a MTCA interface.

Returns

1 for MTCA device and 0 for other interfaces.

Valid for

ADQ14, ADQ7, ADQ8

See also

IsUSBDevice(), IsUSB3Device()

4.3.21 IsPCIeDevice()

```
virtual int IsPCIeDevice ( )
```

Checks whether device is connected over a PCIe/PXIe/MTCA interface.

Returns

1 for PCIe/PXIe/MTCA device and 0 for other interfaces.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

IsUSBDevice(), IsUSB3Device()

4.3.22 IsPCIeLiteDevice()

```
virtual int IsPCIeLiteDevice ( )
```

Checks whether device is connected over a PCIe/PXIe/MTCA interface with the Lite driver stack.

Returns

1 for PCIe/PXIe/MTCA device with Lite driver stack and 0 for other interfaces.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ8

See also

IsUSB2Device(), IsUSB3Device(), IsPCIeDevice()

4.3.23 IsStartedOK()

`virtual unsigned int IsStartedOK ( )`

Checks whether the device has started up OK.

Returns

1 for OK status and 0 for error/failure during configuration.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

IsAlive()

4.3.24 IsUSB3Device()

`virtual int IsUSB3Device ( )`

Checks whether device is connected over a USB3 interface.

Returns

1 for USB3 device and 0 for other interfaces.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

IsUSBDevice(), IsPCIeDevice()

4.3.25 IsUSBDevice()

`virtual int IsUSBDevice ( )`

Checks whether device is connected over a USB2 interface.

Returns

1 for USB2 device and 0 for other interfaces.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

IsUSB3Device(), IsPCIeDevice()

5 ADQ Status

Data Structures

- struct ADQClockSystemStatus

Clock system status. More...

Functions

- unsigned int GetCurrentFloat (unsigned int index, float ∗current)

  Get on board current measurement.
- unsigned int GetCurrentSensorName (unsigned int index, char ∗name)

  Get descriptive name for current sensor.
- unsigned int GetDataFormat ()

  Gets the current data format of the ADQ.
- unsigned int GetErrorVector ()

  Checks whether device is still communicable (alive)
- unsigned int GetLastError ()

  Gets the last error code.
- int GetNofAdcCores (unsigned int ∗nof_adc_cores)

  Get the number of ADC cores.
- unsigned int GetNofChannels ()

  Gets the number of analog input channels.
- unsigned int GetNofCurrentSensors (void)

  Get number of current sensors.
- int GetNofProcessingChannels ()

  Gets the number of processing channels.
- unsigned int GetOutputWidth ()

  Gets the width of the output data in number of bits.
- unsigned int GetPCIeLinkRate ()

  Gets the current PCIe link generation.
- unsigned int GetPCIeLinkWidth ()

  Gets the current PCIe link width.
- unsigned int GetPCIeTLPSize ()

  Gets the current PCIe TLP size.
- int GetStatus (enum ADQStatusId id, void ∗const status)

  Reads a status variable from the digitizer instance.
- unsigned int GetTemperature (unsigned int addr)

  Gets the current on-board temperatures.
- unsigned int GetTemperatureFloat (unsigned int addr, float ∗temperature)

  Gets the current on-board temperatures.
- unsigned int IsAlive ()

  Checks whether device is still communicable (alive)

5.1 Detailed Description

These functions are used to retrieve the current status of the device. Values returned by these functions may change during operation of the units.

5.2 Data Structure Documentation

5.2.1 struct ADQClockSystemStatus

Clock system status.

Returned by ADQ_GetStatus. Invalid or unsupported fields are set to -1. The alarm values are cleared on read.

Data Fields

int32_t	pll1_lock_detect	Indicates that PLL1 is currently locked.
int32_t	pll1_lock_lost_alarm	Indicates that PLL1 has lost its lock since the alarm was last cleared.
int32_t	pll2_lock_detect	Indicates that PLL2 is currently locked.
int32_t	pll2_lock_lost_alarm	Indicates that PLL2 has lost its lock since the alarm was last cleared.
double	reference_source_frequency_estimate	Estimated frequency of the reference source in Hz.

5.3 Function Documentation

5.3.1 GetCurrentFloat()

```
virtual unsigned int GetCurrentFloat (
    unsigned int      index,
    float *           current )
```

Get on board current measurement.

Parameters

index	The sensor index. For valid addresses, see GetNofCurrentSensors().
current	The measured current for the sensor index is written to the float pointed to. The measurement unit is Ampere. Index starts at zero. If the index is out of range a zero will be written to current and zero will be returned.

Returns

1 if successful or 0 if an error occurred

Valid for -

See also

GetCurrentSensorName, GetNofCurrentSensors

5.3.2 GetCurrentSensorName()

```
virtual unsigned int GetCurrentSensorName (
    unsigned int        index,
    char *              name )
```

Get descriptive name for current sensor.

Parameters

index	The sensor index. For valid addresses, see GetNofCurrentSensors().
name	The name of the current sensor at index is written to the char array pointed to. This function guarantees that the returned string is zero-terminated and less than or equal to 256 bytes including the zero terminator. Index starts at zero. If the index is out of range, an empty string will be written to name and zero will be returned.

Returns

1 if successful or 0 if an error occurred

Valid for -

See also

GetCurrentFloat, GetNofCurrentSensors

5.3.3 GetDataFormat()

```
virtual unsigned int GetDataFormat ( )
```

Gets the current data format of the ADQ.

Please see SetDataFormat() for information on the values.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208,ADQ212, ADQ112, ADQ114, ADQ214, SDR14

See also

SetDataFormat()

5.3.4 GetErrorVector()

```
virtual unsigned int GetErrorVector ( )
```

Checks whether device is still communicable (alive)

Returns 0 if no error has been detected, otherwise non-zero. Bold-face marked conditions are irreversible and needs a power-cycling. Others may affect functionality in different ways, but the ADQ board will continue to operate.
Bit 0: Board turned off - detected overheat condition
Bit 1: Detected broken contact bridge between FPGA #1 and #2
Bit 3: Detected fan fault
All detected error conditions will also cause the front panel STATUS LED to flash slowly.

Returns

0 for no error and non-zero for detected errors

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP

See also

GetLastError()

5.3.5 GetLastError()

```
virtual unsigned int GetLastError ( )
```

Gets the last error code.

Returns

0 if no error was detected and the error code otherwise.

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQDSP, SDR14

See also

GetErrorVector(), ADQControlUnit_EnableErrorTrace()

5.3.6 GetNofAdcCores()

```
virtual int GetNofAdcCores (
    unsigned int *    nof_adc_cores )
```

Get the number of ADC cores.

Parameters

nof_adc_cores
Reference to memory where the number of ADC cores is returned.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14

5.3.7 GetNofChannels()

```
virtual unsigned int GetNofChannels ( )
```

Gets the number of analog input channels.

Returns

The number of analog input channels

Valid for

ADQ412, ADQ212, ADQ214, ADQ114, ADQ112, ADQ108, ADQ208, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

5.3.8 GetNofCurrentSensors()

```
virtual unsigned int GetNofCurrentSensors (
    void                )
```

Get number of current sensors.

Returns

0 if the device does not have user readable current sensors, otherwise the number of readable sensors is returned. For GetCurrentFloat() and GetCurrentSensorName() indices from 0 to the number of readable sensors minus one can be used.

Valid for -

See also

GetCurrentFloat, GetCurrentSensorName

5.3.9 GetNofProcessingChannels()

```
virtual int GetNofProcessingChannels ( )
```

Gets the number of processing channels.

A processing channel is an output channel. This number may be different from the number of analog input channels reported by GetNofChannels().

Returns

The number of processing channels

Valid for

ADQ7, ADQ14, ADQ12, ADQ8

5.3.10 GetOutputWidth()

```
virtual unsigned int GetOutputWidth ( )
```

Gets the width of the output data in number of bits.

Returns

the default number of bits per sample

Note

Sample bit width returned by this function is only valid for the standard acquisition modes. It is not valid for other acqusition mode such as averaging or custom streaming solutions.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADq14, ADQ7, ADQ8

See also

GetNofBytesPerSample()

5.3.11 GetPCIeLinkRate()

```
virtual unsigned int GetPCIeLinkRate ( )
```

Gets the current PCIe link generation.

Returns

The PCIe generation used for the connection between ADQ and host. Returns 0 if the ADQ is not connected through PCIe.

Valid for

ADQ412, ADQ1600, ADQ108,ADQ208, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

5.3.12 GetPCIeLinkWidth()

```
virtual unsigned int GetPCIeLinkWidth ( )
```

Gets the current PCIe link width.

Returns

The number of lanes used for the PCIe connection between ADQ and host. Returns 0 if the ADQ is not connected through PCIe

Valid for

ADQ412, ADQ1600, ADQ108,ADQ208, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

5.3.13 GetPCIeTLPSize()

```
virtual unsigned int GetPCIeTLPSize ( )
```

Gets the current PCIe TLP size.

Returns

The TLP (Transfer Layer Packet) size that the board currently uses. Returns 0 if the ADQ is not connected through PCIe

Valid for

ADQ412, ADQ1600, ADQ108,ADQ208, SDR14, ADQDSP

5.3.14 GetStatus()

```
virtual int GetStatus (
    enum ADQStatusId id,
    void *const      status )
```

Reads a status variable from the digitizer instance.

Parameters

id	Status ID. Avilable IDs: ▪ ADQ_STATUS_ID_OVERFLOW: Overflow status ▪ ADQ_STATUS_ID_CLOCK_SYSTEM: Clock system status ▪ ADQ_STATUS_ID_DRAM: DRAM fill
status	Pointer to the status destination. Type depends on id. ▪ ADQ_STATUS_ID_OVERFLOW: status should point to an unsigned integer. $*$status $>$ 0 means overflow condition has occurred. ▪ ADQ_STATUS_ID_CLOCK_SYSTEM: status should point to a ADQClockSystemStatus struct ▪ ADQ_STATUS_ID_DRAM: status should point to a ADQDramStatus struct

Returns

1 for success and 0 for error

Valid for

ADQ14, ADQ7, ADQ8

See also

GetStreamOverflow()

5.3.15 GetTemperature()

```
virtual unsigned int GetTemperature (
    unsigned int     addr )
```

Gets the current on-board temperatures.

Parameters

addr

The temperature to read. Valid addresses are:

- ADQ214, ADQ114, ADQ212, ADQ112:
  - 1: Temperature sensor 1, alg. FPGA
  - 2: Temperature sensor 2, comm. FPGA
- ADQ108, ADQ412, ADQ208, ADQ1600, SDR14, ADQDSP:
  - 0: Sensor controller local temperature
  - 1: Temperature sensor 1 (ADC0)
  - 2: Temperature sensor 2 (ADC1)
  - 3: Temperature sensor 3 (FPGA)
  - 4: Temperature sensor 4 (PCB)
- ADQ12 / ADQ14:
  - 0: Temperature sensor 0 (PCB)
  - 1: Temperature sensor 1 (ADC1)
  - 2: Temperature sensor 2 (ADC2)
  - 3: Temperature sensor 3 (FPGA)
  - 4: Temperature sensor 4 (DCDC2A)
  - 5: Temperature sensor 5 (DCDC2B)
  - 6: Temperature sensor 6 (DCDC1)
- ADQ7:
  - 0: Temperature sensor 0 (PCB)
  - 1: Temperature sensor 1 (ADC1)
  - 2: Temperature sensor 2 (ADC2)
  - 3: Temperature sensor 3 (FPGA)
  - 4: Temperature sensor 4 (DCDC2A)
  - 5: Temperature sensor 5 (DCDC2B) Note: On ADQ7 units produced after 2021-04-01, sensor 5 (DCDC2B) is not used due to a change in DC/DC converter topology, and will report 0 degrees.
  - 6: Temperature sensor 6 (DCDC1)
  - 7: Temperature sensor 7 (RSVD)
- ADQ8:
  - 0: Temperature sensor 0 (PCB)
  - 1: Temperature sensor 1 (ADC1)
  - 2: Temperature sensor 2 (ADC2)
  - 3: Temperature sensor 3 (FPGA)
  - 4: Temperature sensor 4 (DCDC1)
  - 5: Temperature sensor 5 (DCDC2)
  - 6: Temperature sensor 6 (DCDC3)

Returns

The temperature as the actual temperature in Celsius times 256

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetTemperatureFloat

5.3.16 GetTemperatureFloat()

```
virtual unsigned int GetTemperatureFloat (
    unsigned int      addr,
    float *           temperature )
```

Gets the current on-board temperatures.

Parameters

addr	The sensor address to read from. For valid addresses, see GetTemperature().
temperature	Temperature is returned via this pointer, in degrees Celsius.

Returns

1 for successful operation and 0 for failure

Note

A returned value of 0x1000 (256 Celsius) is signaling a not valid/not available temperature.

For ADQ14 and ADQ12, a value below 0.01 degrees Celsius signals a not valid temperature measurement.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetTemperature

5.3.17 IsAlive()

```
virtual unsigned int IsAlive ( )
```

Checks whether device is still communicable (alive)

Returns

1 for alive status and 0 for not communicable.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

IsStartedOK()

6 ADQ General Setup

Functions

- int GetSampleRate (int mode, double *sampleratehz)

  Read out the sample rate of the digitizer.
- unsigned int PowerStandby ()

  Issues power standby mode for device. Will require a full new setup of device for operation.
- unsigned int ReBootADQFromFlash (unsigned int partition)

  Reboots the ADQ over PCIe.
- unsigned int ResetDevice (int resetlevel)

  Resets the device in different ways.
- int ResetOverheat ()

  Reset the device from an overheat condition.
- int SetDataFormat (unsigned int format)

  Sets the sample format.
- unsigned int SetInterleavingMode (char interleaving)

  Sets the interleaving mode.

6.1 Detailed Description

These functions are used to preform general setup of the device.

6.2 Function Documentation

6.2.1 GetSampleRate()

```
virtual int GetSampleRate (
    int          mode,
    double *     sampleratehz )
```

Read out the sample rate of the digitizer.

Parameters

mode - 0: base sample rate of the digitizer, 1: sample rate including sample skip / decimation
sampleratehz - Pointer to where the sampling frequency (in Hz) will be stored as a double

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

6.2.2 PowerStandby()

```
virtual unsigned int PowerStandby ( )
```

Issues power standby mode for device. Will require a full new setup of device for operation.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14

6.2.3 ReBootADQFromFlash()

```
virtual unsigned int ReBootADQFromFlash (
    unsigned int       partition )
```

Reboots the ADQ over PCIe.

Reads the PCIe configuration header from the ADQ, reboots it, and then writes the header back.

This effectively power cycles the FPGA of the ADQ. The ADQ must then be re-enumerated using e.g. ADQControlUnit_FindDevices().

Parameters

partition The partition to reboot the fpga into. 0 for bootloader partition. 1 for firmware partition

Returns

1 for successful operation and 0 for failure

Note

Make sure the partition you want to reboot into does exist in the flash memory.

Valid for

ADQ108, ADQ1600, ADQ208, ADQ412, ADQDSP, SDR14

See also

ResetDevice(), ADQControlUnit_FindDevices()

6.2.4 ResetDevice()

```
virtual unsigned int ResetDevice (
    int                resetlevel )
```

Resets the device in different ways.

Parameters

resetlevel

The level of the reset, according to this list:

- resetlevel = 2: Soft reset, restores to default power-on state [valid for all devices]
- resetlevel = 8: Soft reset of communication link [valid for all devices]
- resetlevel = 16: Hard reset (hardware device) [only for USB, ADQ V5 digitizers]
- resetlevel = 18: USB 3.0 hardware link reset [only for USB 3.0 digitizers]
- resetlevel = 100: Reset of the digital data path in the ADC [only for ADQ14/ADQ12 digitizers]

Returns

1 for successful operation and 0 for failure

Note

After ResetDevice with resetlevel 16 or 18 is issued, hardware must be re-enumerated through the ADQ-ControlUnit by issuing ADQControlUnit_FindDevices. This reset makes the connection between the API and the hardware invalid..

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

6.2.5 ResetOverheat()

```
virtual int ResetOverheat ( )
```

Reset the device from an overheat condition.

Device will be initiated to a default configuration.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208, ADQ412, SDR14, ADQ1600

See also

ResetDevice()

6.2.6 SetDataFormat()

```
virtual int SetDataFormat (
    unsigned int      format )
```

Sets the sample format.

This function will call SetNofBits, SetSampleWidth and WriteAlgoRegister and set all parameters needed for a sample width and/or alignment change. Use the following macros for setting a specific sample format:

ADQ214 & ADQ114:

```
ADQxxx_DATA_FORMAT_PACKED_14BIT
ADQxxx_DATA_FORMAT_UNPACKED_14BIT
ADQxxx_DATA_FORMAT_UNPACKED_16BIT
ADQxxx_DATA_FORMAT_UNPACKED_32BIT
```

ADQ112 & ADQ212:

```
ADQxxx_DATA_FORMAT_PACKED_12BIT
ADQxxx_DATA_FORMAT_UNPACKED_12BIT
ADQxxx_DATA_FORMAT_UNPACKED_16BIT
ADQxxx_DATA_FORMAT_UNPACKED_32BIT
```

ADQ108 & ADQ208:

```
0 = ADQ108_DATA_FORMAT_PACKED_8BIT
2 = ADQ108_DATA_FORMAT_UNPACKED_16BIT
3 = ADQ108_DATA_FORMAT_UNPACKED_32BIT
```

ADQ412:

```
0 = ADQ412_DATA_FORMAT_PACKED_12BIT
1 = ADQ412_DATA_FORMAT_UNPACKED_12BIT
2 = ADQ412_DATA_FORMAT_UNPACKED_16BIT
3 = ADQ412_DATA_FORMAT_UNPACKED_32BIT
```

ADQ1600 & SDR14:

```
0 = XXXX_DATA_FORMAT_PACKED_16BIT
1 = XXXX_DATA_FORMAT_UNPACKED_16BIT
3 = XXXX_DATA_FORMAT_UNPACKED_32BIT
```

ADQ8:

```
0 = XXXX_DATA_FORMAT_UNPACKED_16BIT
3 = XXXX_DATA_FORMAT_UNPACKED_32BIT
```

The packed format will configure the ADQ to store samples for minimal memory footprint, unpacking after transfer to the host PC is done automatically when using multi-record mode. Using streaming mode, unpacking will not be done and is not recommended for use.

Unpacked mode should be used for streaming, this configures the ADQ to store samples padded to 16 bits. 12 & 14 bit modes are stored with sign-extended MSBs. 16 bit mode is stored with zero-padded LSBs.

Unpacked 32 bit mode is used for decimation data, data is stored with zero-padded LSBs.

Parameters

format	Data format to select

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ412, ADQ108, ADQ208, ADQ1600, SDR14, ADQ8

See also

SetStreamStatus()

6.2.7 SetInterleavingMode()

```
virtual unsigned int SetInterleavingMode (
    char                interleaving )
```

Sets the interleaving mode.

This function is used for units where the interleaving mode is configurable by software.

Parameters

interleaving	Interleaving mode selection. This parameter has different values for different units: ▪ ADQ412: – interleaving = 0: Four channel mode (default) – interleaving = 1: Two channel mode, inputs A and C used – interleaving = 2: Two channel mode, inputs B and D used – interleaving = 3: Two channel mode, all inputs active ▪ ADQ208: – interleaving = 0: Two channel mode – interleaving = 1: One channel mode (default)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ208

7 Front-End Options

Functions

- unsigned int GetAdjustableBias (unsigned int channel, int ∗ADCcodes)

  Reads the current DC-bias level of a specified input channel.
- unsigned int GetAfeSwitch (unsigned char Channel, unsigned char ∗afemode)

  Gets the current analog front mode.
- unsigned int GetCalibratedInputImpedance (unsigned int channel, float ∗impedance)

  Gets the calibrated input impedance of the ADQ412DC frontend.
- int GetInputImpedance (unsigned int channel, unsigned int ∗mode)

  Gets the current input impedance mode for a specific analog input channel.
- unsigned int GetInputRange (unsigned int channel, float ∗InpRange)

  Reads the current input range of a specified input channel.

- unsigned int RunCalibrationADQ412DC (unsigned int calmode)

  Schedules a recalibration for the ADQ412DC frontend.
- unsigned int SetAdjustableBias (unsigned int channel, int ADCcodes)

  Sets the DC-bias level for a specified input channel.
- int SetAfeSwitch (unsigned int afe)

  Sets the analog front for DC or AC mode.
- unsigned int SetCalibrationModeADQ412DC (unsigned int calibmode)

  Sets up the continuous temperature compensation of the ADQ412DC frontend.
- int SetInputImpedance (unsigned int channel, unsigned int mode)

  Allows for switching between different input impedances for the analog inputs of the digitizer.
- unsigned int SetInputRange (unsigned int channel, float inputrangemVpp, float *result)

  Sets a desired input range.
- unsigned int SetOvervoltageProtection (unsigned int enabled)

  Allows turning the overvoltage protection on the digitizer inputs on/off.

7.1 Detailed Description

Some units have special options for their front-end. These functions are for example used for selecting analog bias level and DC/AC measurement mode. Please note that not all variants of a product can use all options. For example, ADQ412DC has adjustable bias, while the standard ADQ412 does not.

7.2 Function Documentation

7.2.1 GetAdjustableBias()

```
virtual unsigned int GetAdjustableBias (
    unsigned int      channel,
    int *             ADCcodes )
```

Reads the current DC-bias level of a specified input channel.

Hardware with support for adjustable bias is required.

Parameters

channel	Channel of interest (channels are numbered from 1 and up)
ADCcodes	The resulting bias value in ADC codes is returned via this pointer

Returns

1 for successful operation, 0 for failure

Note

The function is only valid for adjustable bias devices, and will not return the correct value for fixed bias devices with positive/negative bias circuitry.

No actual measurements of the bias seen in the ADC data are performed, the value returned is the expected value based on the current settings.

Valid for

ADQ412, ADQ1600, ADQ108, ADQ12, ADQ14, ADQ7, ADQ8

See also

HasAdjustableBias(), SetAdjustableBias()

7.2.2 GetAfeSwitch()

```
virtual unsigned int GetAfeSwitch (
    unsigned char      Channel,
    unsigned char *    afemode )
```

Gets the current analog front mode.

ADQ214 and ADQ212 has the option to select either a DC or AC coupled input for the front end. This function is used to get the current front end setup for a specified channel.

Parameters

Parameter	Description
Channel	The channel for which to read the status. 1 for channel A and 2 for channel B
afemode	Pointer to where to store the result. These results are possible: ▪ 0: Signal path in AC mode ▪ 1: Signal path in DC mode

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212

See also

SetAfeSwitch()

7.2.3 GetCalibratedInputImpedance()

```
virtual unsigned int GetCalibratedInputImpedance (
    unsigned int       channel,
    float *            impedance )
```

Gets the calibrated input impedance of the ADQ412DC frontend.

The front end setting of the ADQ412DC affects the input impedance. This function will provide the current value.

Parameters

channel	The channel to get the value for
impedance	Pointer to where the result is to be stored

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412DC

7.2.4 GetInputImpedance()

```
virtual int GetInputImpedance (
    unsigned int      channel,
    unsigned int *    mode )
```

Gets the current input impedance mode for a specific analog input channel.

Parameters

channel	The channel to set the impedance for. Indexed from 1
mode	Pointer to where the mode value will be stored. • 0: 50 Ohm • 1: 1 MOhm

Returns

int status

Valid for

ADQ8

See also

GetInputImpedance()

7.2.5 GetInputRange()

```
virtual unsigned int GetInputRange (
```

```
    unsigned int      channel,
    float *           InpRange )
```

Reads the current input range of a specified input channel.

Hardware with support for adjustable input range is required.

Parameters

channel	Channel of interest (channels are numbered from 1 and up)
InpRange	The resulting input range value in millivolts peak-to-peak is returned via this pointer

Returns

1 for successful operation and 0 for failure

Note

The function is only valid for adjustable input range devices, and will not return the correct value for devices with fixed input range.

Valid for

ADQ412DC, ADQ12(-VG), ADQ14(-VG), ADQ8(-VG)

See also

HasAdjustableInputRange(), SetInputRange()

7.2.6 RunCalibrationADQ412DC()

```
virtual unsigned int RunCalibrationADQ412DC (
    unsigned int      calmode )
```

Schedules a recalibration for the ADQ412DC frontend.

The user may specify which calibration routines that should be run an when. There are two calibration routines: **Input offset zeroing**, which removes any input offset present on the digitizer inputs. This removes any DC offsets caused by the input offset in combination with source impedance. During this calibration, the input signals are disconnected for a short amount of time (less than 900ms). The auto-zero calibration uses filtered DACs and therefore does not fully take effect until after about 4 seconds.

Bias calibration. After setting a desired bias using SetAdjustableBias(), this calibration can be performed to digitally adjust the bias even closer to the desired value. During this calibration, the input signals are disconnected for a short amount of time (less than 900ms). Make sure to wait at least 4 seconds after setting the bias before running this calibration, since the bias needs to be stable.

Parameters

calmode Option parameter that specifies which calibration routine to run and when. These are the values that may be used: ▪ 0: Run input offset zeroing calibration immediately ▪ 1: Run bias calibration immediately ▪ 2: Run auto-zero once the next data collection finishes ▪ 3: Run bias calibration once the next data collection finishes ▪ 4: Run both auto-zero and bias calibration in turn. Each one will run after a data collection finishes, and the firmware will automatically wait 4 seconds between the calibration routines

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412DC

See also

SetCalibrationModeADQ412DC()

7.2.7 SetAdjustableBias()

```
virtual unsigned int SetAdjustableBias (
    unsigned int      channel,
    int               ADCcodes )
```

Sets the DC-bias level for a specified input channel.

Hardware with support for adjustable bias is required.

Parameters

channel Channel to set bias for (channels are numbered from 1 and up)
ADCcodes The desired DC-bias, in ADC codes

Returns

1 for successful operation, 0 for failure

Note

The DC-biasing circuitry is heavily filtered, and a change in bias level will typically take around half a second to take effect.

Valid for

ADQ412, ADQ1600, ADQ108, ADQ12, ADQ14, ADQ7, ADQ8

See also

HasAdjustableBias(), GetAdjustableBias()

7.2.8 SetAfeSwitch()

```
virtual int SetAfeSwitch (
    unsigned int      afe )
```

Sets the analog front for DC or AC mode.

ADQ214 and ADQ212 has the option to select either a DC or AC coupled input for the front end. This function is used to switch between them. ADQ7 can be selected to use INX input or INA/INB inputs. To actually use the digitizer in 1ch or 2ch modes, the corresponding firmware must be loaded.

Parameters

afe	For ADQ214: Bitmask to set the analog front end. The different bits have the functions below ▪ 0: 0 for AC mode and 1 for DC mode on channel A ▪ 1: 0 for AC mode and 1 for DC mode on channel B ▪ 2: 0 to deactivate and 1 to activate LF amplification on channel A ▪ 3: 0 to deactivate and 1 to activate LF amplification on channel B For example: ▪ afe = 0x0000 gives AC coupled front end for both channels ▪ afe = 0x0005 gives DC coupled front end for channel A and AC for channel B ▪ afe = 0x000A gives AC coupled front end for channel A and DC for channel B ▪ afe = 0x000F gives DC coupled front end for both channels

For ADQ7: This command only sets the frontend relays. To actually use the digitizer in 1ch or 2ch modes, the corresponding firmware must be loaded.

- 0: Set to INA and INB active mode
- 1: Set to INX active mode

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212, ADQ7

See also

GetAfeSwitch()

7.2.9 SetCalibrationModeADQ412DC()

```
virtual unsigned int SetCalibrationModeADQ412DC (
    unsigned int      calibmode )
```

Sets up the continuous temperature compensation of the ADQ412DC frontend.

Parameters

calibmode	Selects the calibration mode ▪ 0: All temperature compensations disabled ▪ 1: All temperature compensations enabled

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412DC

See also

RunCalibrationADQ412DC()

7.2.10 SetInputImpedance()

```
virtual int SetInputImpedance (
    unsigned int    channel,
    unsigned int    mode )
```

Allows for switching between different input impedances for the analog inputs of the digitizer.

Parameters

channel	The channel to set the impedance for. Indexed from 1
mode	▪ 0: 50 Ohm ▪ 1: 1 MOhm

Returns

int status

Valid for

ADQ8

See also

GetInputImpedance()

7.2.11 SetInputRange()

```
virtual unsigned int SetInputRange (
    unsigned int    channel,
    float           inputrangemVpp,
    float *         result )
```

Sets a desired input range.

One channel is set for each call, according to the parameter listing below.

Parameters

channel	Selects which channel to operate on (channels are numbered from 1 and up)
inputrangemVpp	The desired input range in millivolts peak-to-peak
result	The software will calculate the closest approximation to the desired input range that it can achieve. The value is returned via this pointer.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412DC, ADQ12(-VG), ADQ14(-VG), ADQ8(-VG)

7.2.12 SetOvervoltageProtection()

```
virtual unsigned int SetOvervoltageProtection (
    unsigned int    enabled )
```

Allows turning the overvoltage protection on the digitizer inputs on/off.

Disabling the overvoltage protection improves linearity, but also makes the device frontend susceptible to damage if large-amplitude signals are used.

Parameters

enabled	Set to 0 to disable all overvoltage protection, 1 to enable (default after board startup).

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14

8 Peripheral Setup

Data Structures

- struct ADQDaisyChainDeviceInformation

  Daisy chain device info struct used with SDCardBackupDaisyChainGetTriggerInformation(). More...
- struct ADQDaisyChainTriggerInformation

  Daisy chain trigger info struct used with SDCardBackupDaisyChainGetTriggerInformation(). More...
- struct SDCardConfiguration

  SD Card configuration struct used with SDCardBackupGetConfiguration(). More...

Functions

- unsigned int EnableClockRefOut (unsigned int enable)

  Enables or disables the clock reference output signal.
- int EnableGPIOPort (unsigned int port, unsigned int enable)

  Enables GPIO mode for a specified I/O port.
- int EnableGPIOSupplyOutput (unsigned int enable)

  Enable GPIO supply voltage output.
- unsigned int HasAdjustableBias ()

  Checks whether the device supports adjustable DC-biasing of the ADC frontends.
- unsigned int HasAdjustableInputRange ()

  Checks whether the device supports adjustable frontend input range.
- unsigned int HasGPIOHardware ()

  Checks whether the device has GPIO hardware.
- unsigned int HasTrigHardware (unsigned int trignum)

  Checks if variable trigger threshold is available for the unit.
- unsigned int HasTrigoutHardware (unsigned int trignum)

  Checks if a specific external trigger output exists on the unit.
- unsigned int HasVariableTrigThreshold (unsigned int trignum)

  Checks if variable trigger threshold is available for the unit.
- unsigned int ReadEEPROM (unsigned int addr)

  Reads one byte from the on-board EEPROM.
- unsigned int ReadEEPROMDB (unsigned int addr)

  Reads one byte from the daughterboard EEPROM.
- unsigned int ReadGPIO ()

Gets the state of the GPIO pins.

- int ReadGPIOPort (unsigned int port, unsigned int *data)

  Reads the value seen by the GPIO pins for a specified GPIO port.

- int SDCardBackupDaisyChainGetTriggerInformation (unsigned int source, unsigned int edge, int level, unsigned int channel, unsigned int nof_records, unsigned int record_length, struct ADQDaisyChainDeviceInformation *device_info, unsigned int nof_devices, struct ADQDaisyChainTriggerInformation *trig_info)

  Get the daisy chain trigger information from SD card.

- int SDCardBackupEnable (int enable)

  Enables the backup of data to microSD card.

- int SDCardBackupGetConfiguration (struct SDCardConfiguration *sdc_config)

  Read the device configuration uses for data stored on SD card.

- int SDCardBackupGetData (void **target_buffers, void *target_headers, unsigned int target_buffer_size, unsigned char target_bytes_per_sample, unsigned int start_record_number, unsigned int number_of_records, unsigned char channel_mask, unsigned int start_sample, unsigned int nof_samples)

  Read data from SD card. See GetDataWHTS()

- int SDCardBackupGetProgress (unsigned int *percent)

  Get the progress of the backup process in percent.

- int SDCardBackupGetStatus (unsigned int *status)

  Get the status of the SD card backup procedure.

- int SDCardBackupResetWriterProcess ()

  Reset the SD card backup process.

- int SDCardBackupSetAdditionalData (unsigned int daisy_chain_position)

  Set addition information required to parse data from SD card.

- int SDCardErase (unsigned int start_block, unsigned int stop_block)

  Commands the SDCard to erase blocks from start_block to stop_block (inclusive). start_block - stop_block must be larger than one. A block has a fixed size of 512B.

- unsigned int SDCardGetNofSectors ()

  Get the number of sectors required to store data.

- int SDCardInit ()

  Attempts to bring the compatible high capacaty SD-Card through initialization. This should be done before erasing, writing or reading from/to the SD-Card.

- int SDCardIsInserted (int *is_inserted)

  Check if the micro SD card is inserted.

- int SDCardRead (unsigned int *dst, unsigned int start_block, unsigned int number_of_blocks)

  Reads the specified number_of_blocks starting from start_block params. A block has a fixed size of 512B.

- int SDCardWriterStatus (unsigned int *status)

  Retrieves the status of the microblaze mem Writer process.

- unsigned int SetConfigurationTrig (unsigned int mode, unsigned int pulselength, unsigned int invertoutput)

  Sets the configuration of the trig connector.

- unsigned int SetDACOffsetVoltage (unsigned char channel, float v)

Sets the common-mode voltage for the DAC outputs of SDR14.

- int SetDirectionGPIO (unsigned int direction, unsigned int mask)

    Sets the direction of the GPIO pins.
- int SetDirectionGPIOPort (unsigned int port, unsigned int direction, unsigned int mask)

    Sets the input/output state of the GPIO pins for a specified GPIO port.
- int SetDirectionTrig (int direction)

    Sets the direction of the trig connector.
- unsigned int SetFanControl (unsigned int fan_control)

    Sets the fan control of the device.
- int SetFunctionGPIOPort (unsigned int port, int function_id, int gpio_id)

    Control the GPIO output functions.
- unsigned int SetupTriggerOutput (int outputnum, unsigned int mode, unsigned int pulselength, unsigned int invertoutput)

    Sets the configuration of the trig connector.
- int SetupUserRangeGPIO (unsigned int channel, int threshold_high, int threshold_low)

    Configure User Range GPIO function.
- unsigned int TrigoutEnable (unsigned int bitflags)

    Selects which trigout connectors to send a trigger output to.
- unsigned int WriteEEPROM (unsigned int addr, unsigned int data, unsigned int accesscode)

    Writes one byte to the on-board EEPROM.
- unsigned int WriteEEPROMDB (unsigned int addr, unsigned int data, unsigned int accesscode)

    Writes one byte to the daughterboard EEPROM.
- int WriteGPIO (unsigned int data, unsigned int mask)

    Sets the state of the GPIO pins.
- int WriteGPIOPort (unsigned int port, unsigned int data, unsigned int mask)

    Sets the output value of the GPIO pins for a specified GPIO port.
- int WriteTrig (int data)

    Sets the output level for the trig output.

8.1 Detailed Description

These functions may be use to access and configure some of the hardware on the device.

8.2 Data Structure Documentation

8.2.1 struct ADQDaisyChainDeviceInformation

Daisy chain device info struct used with SDCardBackupDaisyChainGetTriggerInformation().

Data Fields

int64_t	Position	
int64_t	PretriggerSamples	

Data Fields

int64_t	SampleRate	
int64_t	TriggerDelaySamples	

8.2.2 struct ADQDaisyChainTriggerInformation

Daisy chain trigger info struct used with SDCardBackupDaisyChainGetTriggerInformation().

Data Fields

double *	ExtendedPrecision	
int64_t *	RecordStart	
uint64_t	Timestamp	

8.2.3 struct SDCardConfiguration

SD Card configuration struct used with SDCardBackupGetConfiguration().

Data Fields

unsigned int	CBufMemArea	
unsigned int	CBufSize	
unsigned int	ChannelMask	
unsigned int	ChunkSize	
unsigned int	CyclicBuffersEnabled	
int	DaisyChainPosition	
char	ISODate[32]	
unsigned int	Length	
unsigned int	NumberOfChannels	
unsigned int	NumberOfParallelSamples	
unsigned int	NumberOfRecords[8]	
unsigned int	PreTrigger[8]	
unsigned int	RecordLength[8]	
unsigned int	SampleSkip[8]	
char	SerialNumber[16]	
unsigned int	TriggerDelay[8]	
unsigned int	TriggerMode	

Data Fields

unsigned int	Valid	
unsigned int	Version	

8.3 Function Documentation

8.3.1 EnableClockRefOut()

```
virtual unsigned int EnableClockRefOut (
    unsigned int    enable )
```

Enables or disables the clock reference output signal.

Parameters

enable Output enable selection. Set to 1 for enabled and 0 for disabled.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetClockSource()

8.3.2 EnableGPIOPort()

```
virtual int EnableGPIOPort (
    unsigned int    port,
    unsigned int    enable )
```

Enables GPIO mode for a specified I/O port.

Parameters

port	Port number. ▪ ADQ7: – Port 0: GPIO (always enabled) – Port 1: GPDO – Port 2: External trigger – Port 3: SYNC ▪ ADQ12/ADQ14: – Port 0: GPIO (always enabled) – Port 1: GPIOCTRL (always enabled) – Port 2: External trigger – Port 3: SYNC ▪ ADQ8: – Port 0: Unused – Port 1: Unused – Port 2: External trigger – Port 3: SYNC
enable	Set to 1 to enable GPIO control of the port

Returns

1 for successful operation and 0 for failure

Note

When SYNC is enabled as GPIO, the API SetTriggerInputImpedance will not have any effect for SYNC pin.

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTriggerInputImpedance()

8.3.3 EnableGPIOSupplyOutput()

```
virtual int EnableGPIOSupplyOutput (
    unsigned int      enable )
```

Enable GPIO supply voltage output.

Parameters

enable	Set to 1 to enable the supply voltage output in the GPIO connector

Returns

1 for success, 0 otherwise

Valid for

ADQ12, ADQ14, ADQ7

8.3.4 HasAdjustableBias()

```
virtual unsigned int HasAdjustableBias ( )
```

Checks whether the device supports adjustable DC-biasing of the ADC frontends.

Returns

1 if adjustable biasing is supported, 0 otherwise

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, ADQ214, ADQ212, ADQ114, ADQ112, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetAdjustableBias(), GetAdjustableBias

8.3.5 HasAdjustableInputRange()

```
virtual unsigned int HasAdjustableInputRange ( )
```

Checks whether the device supports adjustable frontend input range.

Returns

1 if adjustable input range is supported, 0 otherwise

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, ADQ214, ADQ212, ADQ114, ADQ112, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetInputRange(), GetInputRange()

8.3.6 HasGPIOHardware()

```
virtual unsigned int HasGPIOHardware ( )
```

Checks whether the device has GPIO hardware.

Returns

1 if unit has GPIO hardware, 0 otherwise

Valid for

ADQ12, ADQ14

8.3.7 HasTrigHardware()

```
virtual unsigned int HasTrigHardware (
    unsigned int    trignum )
```

Checks if variable trigger threshold is available for the unit.

Parameters

trignum The number of the trigger to check

Returns

1 if the specified external trigger input exists in the board hardware.

Valid for

ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

HasVariableTrigThreshold()

8.3.8 HasTrigoutHardware()

```
virtual unsigned int HasTrigoutHardware (
    unsigned int    trignum )
```

Checks if a specific external trigger output exists on the unit.

Parameters

trignum The number of the trigger to check

Returns

1 if the specified external trigger output exists in the board hardware.

Valid for

ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

8.3.9 HasVariableTrigThreshold()

```
virtual unsigned int HasVariableTrigThreshold (
    unsigned int    trignum )
```

Checks if variable trigger threshold is available for the unit.

Parameters

trignum	The number of the trigger to check

Returns

1 if the specified external trigger input supports variation of the trigger threshold voltage (via the SetExtTrigThreshold() command).

Valid for

ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetExtTrigThreshold(), HasTrigHardware()

8.3.10 ReadEEPROM()

```
virtual unsigned int ReadEEPROM (
    unsigned int      addr )
```

Reads one byte from the on-board EEPROM.

Parameters

addr	The byte address to read

Returns

The byte read

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ412, ADQ108, ADQ208, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

WriteEEPROM(), ReadEEPROMDB(), WriteEEPROMDB()

8.3.11 ReadEEPROMDB()

```
virtual unsigned int ReadEEPROMDB (
    unsigned int      addr )
```

Reads one byte from the daughterboard EEPROM.

Parameters

addr
The byte address to read

Returns

The byte read

Valid for

ADQ412, ADQ108, ADQ208, ADQ1600, SDR14, ADQ12, ADQ14, ADQ8

See also

WriteEEPROMDB(), WriteEEPROM(), ReadEEPROM()

8.3.12 ReadGPIO()

```
virtual unsigned int ReadGPIO ( )
```

Gets the state of the GPIO pins.

Returns

The state as a bit field, where bit 0 corresponds to the value of GPIO pin 1, bit 1 to pin 2, and so on.

Example

If the returned value is 9, pin 1 and 4 are high, because $9 = 2^3 + 2^0$

Note

For firmware older than revision 3991, bit 2 corresponds to pin 2, bit 3 to pin 1, and bit 5 to pin 5.

Valid for

ADQ214, ADQ112, ADQ114, ADQ214, ADQ412, ADQ108, ADQ208, ADQ1600, SDR14

See also

SetDirectionGPIO(), WriteGPIO()

8.3.13 ReadGPIOPort()

```
virtual int ReadGPIOPort (
    unsigned int     port,
    unsigned int *   data )
```

Reads the value seen by the GPIO pins for a specified GPIO port.

Parameters

port Port number. ▪ ADQ7: – Port 0: GPIO (12 bits) – Port 1: GPDI (4 bits) – Port 2: External trigger (1 bit) – Port 3: SYNC (1 bit) ▪ ADQ12/ADQ14: – Port 0: GPIO (16 bits) – Port 1: GPIOCTRL (5 bits) – Port 2: External trigger (1 bit) – Port 3: SYNC (1 bit) ▪ ADQ8 – Port 0: Unused – Port 1: Unused – Port 2: External trigger (1 bit) – Port 3: SYNC (1 bit)
data A pointer to where the readout value should be stored. Each bit corresponds to an I/O pin of the port.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

ReadGPIO()

8.3.14 SDCardBackupDaisyChainGetTriggerInformation()

```
virtual int SDCardBackupDaisyChainGetTriggerInformation (
    unsigned int        source,
    unsigned int        edge,
    int                 level,
    unsigned int        channel,
    unsigned int        nof_records,
    unsigned int        record_length,
    struct
    ADQDaisyChainDeviceInformation
    *                   device_info,
    unsigned int        nof_devices,
    struct
    ADQDaisyChainTriggerInformation
    *                   trig_info )
```

Get the daisy chain trigger information from SD card.

See DaisyChainGetTriggerInformation for documentation.

Note that the input parameters must be read from the SD card using SDCardBackupGetConfiguration.

Parameters

source	See DaisyChainGetTriggerInformation()
edge	See DaisyChainGetTriggerInformation()
level	See DaisyChainGetTriggerInformation()
channel	See DaisyChainGetTriggerInformation()
nof_records	See DaisyChainGetTriggerInformation()
record_length	See DaisyChainGetTriggerInformation()
device_info	See DaisyChainGetTriggerInformation()
nof_devices	See DaisyChainGetTriggerInformation()
trig_info	See DaisyChainGetTriggerInformation()

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.15 SDCardBackupEnable()

```
virtual int SDCardBackupEnable (
    int                enable )
```

Enables the backup of data to microSD card.

This function must be called between MultiRecordSetup and ArmTrigger

Parameters

enable	0: Disables the backup 1: Enables the backup

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.16 SDCardBackupGetConfiguration()

```
virtual int SDCardBackupGetConfiguration (
    struct
    SDCardConfiguration
    *                  sdc_config )
```

Read the device configuration uses for data stored on SD card.

See ADQAPI.h for struct fields.

Parameters

sdc_config
Pointer to SDCardConfiguration struct

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.17 SDCardBackupGetData()

```
virtual int SDCardBackupGetData (
    void **          target_buffers,
    void *           target_headers,
    unsigned int     target_buffer_size,
    unsigned char    target_bytes_per_sample,
    unsigned int     start_record_number,
    unsigned int     number_of_records,
    unsigned char    channel_mask,
    unsigned int     start_sample,
    unsigned int     nof_samples )
```

Read data from SD card. See GetDataWHTS()

The length of the targer_buffers and target_headers parameters must match the available data on the SDCard. Use SDCardBackupGetConfiguration to read the configuration used when the data was stored to the SD card.

Parameters

target_buffers See GetDataWHTS()
target_headers See GetDataWHTS()
target_buffer_size See GetDataWHTS()
target_bytes_per_sample See GetDataWHTS()
start_record_number See GetDataWHTS()
number_of_records See GetDataWHTS()
channel_mask See GetDataWHTS()
start_sample See GetDataWHTS()
nof_samples See GetDataWHTS()

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.18 SDCardBackupGetProgress()

```
virtual int SDCardBackupGetProgress (
    unsigned int * percent )
```

Get the progress of the backup process in percent.

Note that the percent value is ceil, and may report 100 without being finished. Use SDCardBackupGetStatus to check if the write process is finished.

Parameters

percent The completion progress in percent.

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.19 SDCardBackupGetStatus()

```
virtual int SDCardBackupGetStatus (
    unsigned int *    status )
```

Get the status of the SD card backup procedure.

Parameters

status	- Bit 0: Backup read completed - Bit 1: Backup read in progress - Bit 2: Backup FIFO underflow (error) - Bit 3: Backup FIFO overflow (error) - Bit 4: Backup FIFO almost full - Bit 5: Backup FIFO full (error) - Bit 6: Backup FIFO empty - Bit 16: Backup write process busy (writing data) - Bit 17: Backup write process busy (writing configuration) - Bit 18: Backup write error

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.20 SDCardBackupResetWriterProcess()

```
virtual int SDCardBackupResetWriterProcess ( )
```

Reset the SD card backup process.

The reset will abort an ongoing write process to the SD card. Can be used to reset the process if an error has occurred, i.e. if bit 18 from SDCardBackupGetStatus is set.

Calling this function while the process is busy (if bit 16 or bit 17 is set) will result in loss of data.

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.21 SDCardBackupSetAdditionalData()

```
virtual int SDCardBackupSetAdditionalData (
    unsigned int      daisy_chain_position )
```

Set addition information required to parse data from SD card.

Parameters

daisy_chain_position	The position of the device in the daisy chain. Setting this parameter is only required if the daisy chain trigger mode is used.

Returns

1 for success, 0 otherwise

Valid for

ADQ8

8.3.22 SDCardErase()

```
virtual int SDCardErase (
    unsigned int      start_block,
    unsigned int      stop_block )
```

Commands the SDCard to erase blocks from start_block to stop_block (inclusive). start_block - stop_block must be larger than one. A block has a fixed size of 512B.

Parameters

start_block	Block from where to start erasing
stop_block	Block where to stop erasing

Returns

1 for successful operation and 0 for failure

Valid for

ADQ8

8.3.23 SDCardGetNofSectors()

```
virtual unsigned int SDCardGetNofSectors ( )
```

Get the number of sectors required to store data.

This function should be used to determine the number of sectors that has to be erased.

Note that this function will only return a valid result if called after MultiRecordSetup.

Returns

The number of SD card sectors which will be used.

Valid for

ADQ8

8.3.24 SDCardInit()

```
virtual int SDCardInit ( )
```

Attempts to bring the compatible high capacaty SD-Card through initialization. This should be done before erasing, writing or reading from/to the SD-Card.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ8

8.3.25 SDCardIsInserted()

```
virtual int SDCardIsInserted (
    int *              is_inserted )
```

Check if the micro SD card is inserted.

Parameters

is_inserted
Inserted status for the SD card (1 if inserted, 0 otherwise)

Returns

1 for success , 0 otherwise

Valid for

ADQ8

8.3.26 SDCardRead()

```
virtual int SDCardRead (
    unsigned int *    dst,
    unsigned int      start_block,
    unsigned int      number_of_blocks )
```

Reads the specified number_of_blocks starting from start_block params. A block has a fixed size of 512B.

Parameters

dst	destination buffer
start_block	Block to start reading from
number_of_blocks	Number of blocks to read

Returns

1 for successful operation and 0 for failure

Valid for

ADQ8

8.3.27 SDCardWriterStatus()

```
virtual int SDCardWriterStatus (
    unsigned int *    status )
```

Retrieves the status of the microblaze mem Writer process.

Parameters

status	status, 2 internal error, 1 busy, 0 idle

Returns

1 for successful operation and 0 for failure

Valid for

ADQ8

8.3.28 SetConfigurationTrig()

```
virtual unsigned int SetConfigurationTrig (
    unsigned int     mode,
    unsigned int     pulselength,
    unsigned int     invertoutput )
```

Sets the configuration of the trig connector.

This command will overwrite any previous calls to SetDirectionTrig().

Parameters

mode	For ADQ7 / ADQ12 / ADQ14 / ADQ8 digitizers, the following mode values are allowed: ▪ 0 : Trigger set to input (default) ▪ 1 : Reserved ▪ 2 : Internal trigger ▪ 3 : Record acquisition trigger event, uses pulselength argument (ADQ14/ADQ12 channel A only) ▪ 4 : Level trigger event, uses pulselength argument ▪ 5 : Internal trigger rising edge, uses pulselength argument ▪ 6 : Internal trigger falling edge, uses pulselength argument ▪ 7 : Internal trigger both edges, uses pulselength argument

For V5/V6 digitizers:

- 0x00: Trigger set to input (default)
- 0x01: WriteTrig() data
- 0x05: Trigger state (See app note)
- 0x11: Trigger event, use **pulselength** to set the length. Wired OR between units, set WriteTrig(1)
- 0x19: Level trigger, use **pulselength** to set the length. Wired OR between units, set WriteTrig(1)
- 0x41: Internal trigger 50% duty cycle
- 0x45: Internal trigger, use **pulselength** to set the length If mode is OR:ed with bit 5 (mode | 0x20) the special GPIO trigger block will be activated. Triggers are then blocked with an active high input on GPIO connector pin 5.

Some V6 digitizers support up to 2 selectable trigger out ports Trigger out 1 is configured by mode bits {20, 2, 6} Trigger out 2 is configured by mode bits {23, 22, 21}

- Bit pattern: Trigger choice
- 000 writetrig data
- 001 internal trigger signal (50% duty cycle)
- 010 use trig_selector from MultiRecord trigger
- 011 use configurable pulse from internal trigger
- 100 use ext_trig_vector_i directly
- 101 use pxie_trig_vector_i[0] (not available for all devices)

- 110 use pxie_trig_vector_i[1] (not available for all devices)
- 111 use PrecisePeriodTrig trig event

Parameters

pulselength	Sets the length of the output pulse in nanoseconds when trigger connector is used as output. Minimum is 20ns (14.4ns for ADQ112 and ADQ212) and maximum is 5100 ns (3672ns for ADQ112 and ADQ212)
invertoutput	If set to 1, the output will be inverted

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ1600, ADQ412, ADQ12, ADQ14, ADQ7, ADQ8

See also

WriteTrig(), SetDirectionTrig(), SetupTriggerOutput()

8.3.29 SetDACOffsetVoltage()

```
virtual unsigned int SetDACOffsetVoltage (
    unsigned char    channel,
    float            v )
```

Sets the common-mode voltage for the DAC outputs of SDR14.

Parameters

channel	The output channel to set the voltage for (1 or 2)
v	The common mode voltage to set, from -1.0 to 1.0

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGArm()

8.3.30 SetDirectionGPIO()

```
virtual int SetDirectionGPIO (
    unsigned int      direction,
    unsigned int      mask )
```

Sets the direction of the GPIO pins.

Parameters

direction The configuration as a bit mask: ▪ bit 0: Sets pin 1 to input if 0 and output as 1 ▪ bit 1: Sets pin 2 to input if 0 and output as 1 ▪ bit 2: Sets pin 3 to input if 0 and output as 1 ▪ bit 3: Sets pin 4 to input if 0 and output as 1 ▪ bit 4: Sets pin 5 to input if 0 and output as 1	
mask A negative bit mask. If a bit here is 1, then the same bit in **direction** will be ignored.	

Returns

The state as a bit field, where bit 0 corresponds to the value of GPIO pin 1, bit 1 to pin 2, and so on.

Example

If **direction** is 3 (2^1+2^0) and **mask** is 1, pin 1 will retain its old configuration (because of the mask), pin 2 will be set to output, and all other pins configured to be input.

Note

For firmware older than revision 3991, only pin 5 may be configured

For SDR14 bits 8-12 are used for muxing GPIO output between API control and AWG control, for example setting bit 8 will give control to the AWG data embedded for GPIO pin 1. Pins still need to be configured as output to enable this feature. Bit 16-20 will decide if GPIO is fed from channel A embedded GPIO data or from channel B embedded GPIO data. Setting a bit will enable taking data from channel B embedded GPIO data.

Valid for

ADQ214, ADQ112, ADQ114, ADQ214, ADQ412, ADQ108, ADQ208, ADQ1600, SDR14

See also

WriteGPIO(), ReadGPIO()

8.3.31 SetDirectionGPIOPort()

```
virtual int SetDirectionGPIOPort (
```

```
    unsigned int        port,
    unsigned int        direction,
    unsigned int        mask )
```

Sets the input/output state of the GPIO pins for a specified GPIO port.

Parameters

port
Port number.
- ADQ7:
  - Port 0: GPIO (6 bits. Each bit set direction for two GPIO. Bit 0 is for GPIO0 and GPIO1 and so on)
  - Port 1: Not used (GPDI and GPDO has fixed direction)
  - Port 2: External trigger (1 bit)
  - Port 3: SYNC (1 bit)
- ADQ12/ADQ14:
  - Port 0: GPIO (16 bits)
  - Port 1: GPIOCTRL (5 bits)
  - Port 2: External trigger (1 bit)
  - Port 3: SYNC (1 bit)
- ADQ8:
  - Port 0: Unused
  - Port 1: Unused
  - Port 2: External trigger (1 bit)
  - Port 3: SYNC (1 bit)

direction
Each bit corresponds to an I/O pin of the port, set to 0 for input, 1 for output.

mask
Only bits which are zero-valued in the mask will be changed

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

SetDirectionGPIO()

8.3.32 SetDirectionTrig()

```
virtual int SetDirectionTrig (
    int                 direction )
```

Sets the direction of the trig connector.

Parameters

direction

The specified direction. These values are valid:

- 0: Input
- 1: Output, data from WriteTrig calls
- 5: Output, a positive pulse for each trigger accepted (ignores WriteTrig() calls). Not available for ADQ108 and ADQ208

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, ADQ1600

See also

SetConfigurationTrig(), WriteTrig()

8.3.33 SetFanControl()

```
virtual unsigned int SetFanControl (
    unsigned int        fan_control )
```

Sets the fan control of the device.

Parameters

fan_control

Bit field that control the fans:

- bit 31: Set to 0 for automatic mode and 1 for manual override (not valid for ADQ12 / ADQ14)
- bit 30: Set to 0 to keep fan failure monitoring and 1 to disable (not valid for ADQ12 / ADQ14)
- bit 29: Set to 0 to restart the fan and 1 to shutdown (not valid for ADQ12 / ADQ14)
- bits 0 to 3: Set the fan speed from 0-15

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212, ADQ112, ADQ114, SDR14, ADQ412, ADQ1600, ADQ108, ADQ208, ADQ12, ADQ14

8.3.34 SetFunctionGPIOPort()

```
virtual int SetFunctionGPIOPort (
    unsigned int        port,
    int                 function_id,
```

```
int gpio_id )
```

Control the GPIO output functions.

The function is controlled for each pin. Enable the function by setting function_id to 1. The available functions for each pin are listed below.

- GPIO 0 (out):
  - Function id 0: Disabled.
  - Function id 1: Data in user range. The pin is high when the data is within a specified range. The range must be specified with SetupUserRangeGPIO().
- GPIO 1 (out):
  - Function id 0: Disabled.
  - Function id 1: Trigger blocking window. High when triggers are accepted.
- GPIO 2 (out):
  - Function id 0: Disabled.
  - Function id 1: Trigger blocking armed. High when armed, low otherwise.
- GPIO 3 (out):
  - Function id 0: Disabled.
  - Function id 1: Data path overflow.
- GPIO 4 (out):
  - Function id 0: Disabled.
  - Function id 1: Trigger out. Must be configured with SetupTriggerOutput().
- GPIO 5 (out):
  - Function id 0: Disabled.
  - Function id 1: ATD: Ongoing accumulation. DAQ: Acquiring record.
- GPIO 6 (out):
  - Function id 0: Disabled.
  - Function id 1: Frame sync signal. See SetupFrameSync()
- GPIO 7 (out):
  - Function id 0: Disabled.
  - Function id 1: 10 MHz clock reference
- GPIO 8 (out):
  - Function id 0: Disabled.
  - Function id 1: Data acquisition armed
- GPIO 9 (out):
  - Function id 0: Disabled.
  - Function id 1: N/A

- GPIO 10 (in):
  - Function id 0: Disabled.
  - Function id 1: Trigger 0 in
- GPIO 11 (in):
  - Function id 0: Disabled.
  - Function id 1: Trigger 1 in

Parameters

port	Port number. Must be 0 (single-ended GPIO)
function_id	See table above
gpio_id	GPIO pin ID

Returns

1 for success, 0 otherwise

Valid for

ADQ7-PCIe/PXIe

See also

WriteGPIO()

8.3.35 SetupTriggerOutput()

```
virtual unsigned int SetupTriggerOutput (
    int             outputnum,
    unsigned int    mode,
    unsigned int    pulselength,
    unsigned int    invertoutput )
```

Sets the configuration of the trig connector.

Parameters

outputnum Selects the trigger output port: ▪ 0 : The TRIG connector ▪ 1 : MLVDS RX17 (MTCA only) ▪ 2 : MLVDS TX17 (MTCA only) ▪ 3 : MLVDS RX18 (MTCA only) ▪ 4 : MLVDS TX18 (MTCA only) ▪ 5 : MLVDS RX19 (MTCA only) ▪ 6 : MLVDS TX19 (MTCA only) ▪ 7 : MLVDS RX20 (MTCA only) ▪ 8 : MLVDS TX20 (MTCA only) ▪ 9 : PXIe STARC (PXIe only) ▪ 10: PXI TRIG0 (PXIe only) ▪ 11: PXI TRIG1 (PXIe only) ▪ 12: The SYNC connector ▪ 13: GPIO trigout pin (ADQ7-PCIe/PXIe only). GPIO pin function must be set for output to be active, see SetFunctionGPIOPort(). Uses the same logic internally as TRIG connector (mode 0), and can not be configured independently of TRIG connector.
mode Selects a source signal for the trigger output, as follows: ▪ 0 : Trigger output disabled (default) ▪ 1 : Reserved ▪ 2 : Internal trigger ▪ 3 : Record acquisition trigger event, uses pulselength argument (ADQ14/ADQ12 channel A only) ▪ 4 : Level trigger event, uses pulselength argument ▪ 5 : Internal trigger rising edge, uses pulselength argument ▪ 6 : Internal trigger falling edge, uses pulselength argument ▪ 7 : Internal trigger both edges, uses pulselength argument ▪ 8 : External trigger (TRIG connector) ▪ 9 : Trigger blocking status event, uses pulselength argument (ADQ14 only) ▪ 10: Enable WR PPS output (SYNC connector on ADQ7 only) ▪ 11: Sync trigger input, rising edge, uses pulselength argument (ADQ7 only)
pulselength Sets the length of the output pulse in nanoseconds.
invertoutput If set to 1, the output will be inverted

Returns

1 for successful operation and 0 for failure

Note

TRIG/SYNC share the same trigger output source, pulser and inversion circuitry (cannot be independently set). All MLVDS/PXI/PXIe share the same trigger output source, pulser and inversions circuitry (cannot be independently set)

On ADQ14/ADQ12 mode 2,5,6 and 7 requires edges of the internal trigger to be set to match (set with SetTriggerEdge)

Valid for

ADQ12, ADQ14, ADQ7

8.3.36 SetupUserRangeGPIO()

```
virtual int SetupUserRangeGPIO (
    unsigned int    channel,
    int             threshold_high,
    int             threshold_low )
```

Configure User Range GPIO function.

The user range bit is high when the input signal is below the high threshold and above the low threshold (including the threshold values). Only one channel may be used.

User range GPIO function pin must also be enabled with SetFunctionGPIOPort()

Parameters

channel Channel index, starting at 1
threshold_high High threshold in ADC codes.
threshold_low Low threshold in ADC codes

Returns

1 for success, 0 otherwise

Valid for

ADQ7

See also

SetFunctionGPIOPort

8.3.37 TrigoutEnable()

```
virtual unsigned int TrigoutEnable (
    unsigned int    bitflags )
```

Selects which trigout connectors to send a trigger output to.

Parameters

bitflags A bit field where the two first bits do this: ▪ bit 0: If this bit is asserted the signal will be output on trigout1 ▪ bit 1: If this bit is asserted the signal will be output on trigout2

Returns

1 for successful operation and 0 for failure

Valid for

ADQ1600

8.3.38 WriteEEPROM()

```
virtual unsigned int WriteEEPROM (
    unsigned int      addr,
    unsigned int      data,
    unsigned int      accesscode )
```

Writes one byte to the on-board EEPROM.

Lower addresses are reserved for internal use and are protected by an access code, while higher addresses are free to use for storage of customer data.

- ADQ12/ADQ14/ADQ7/ADQ8: Addresses 0x000000 to 0x00FFFF are write protected, highest address is 0x03FFFF
- Other digitizers: Addresses 0x000000 to 0x003FFF are write protected, highest address is 0x00FFFF

Parameters

addr The byte address to write.
data The data to write (8 bits)
accesscode Internal use only, set to 0 for normal operation

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

ReadEEPROMDB(), WriteEEPROM(), ReadEEPROM()

8.3.39 WriteEEPROMDB()

```
virtual unsigned int WriteEEPROMDB (
    unsigned int    addr,
    unsigned int    data,
    unsigned int    accesscode )
```

Writes one byte to the daughterboard EEPROM.

The daughterboard EEPROM is completely reserved for internal use only.

Parameters

addr	The byte address to write.
data	The data to write (8 bits)
accesscode	Internal use only, set to 0 for normal operation

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ108, ADQ208, ADQ1600, SDR14, ADQ12, ADQ14, ADQ8

See also

ReadEEPROMDB(), WriteEEPROM(), ReadEEPROM()

8.3.40 WriteGPIO()

```
virtual int WriteGPIO (
    unsigned int    data,
    unsigned int    mask )
```

Sets the state of the GPIO pins.

Only the bits that are configured as output will be written. To select these bits, use SetDirectionGPIO()

Parameters

data	The values to write. Bit 0 sets GPIO pin 1, bit 1 pin 2 and so forth

Parameters

mask
A negative bit mask. If a bit here is 1, then the same bit in **data** will be ignored.

Returns

The state as a bit field, where bit 0 corresponds to the value of GPIO pin 1, bit 1 to pin 2, and so on.

Example

If **data** is 3 (2^1+2^0) and **mask** is 1, pin 1 will retain its old value (because of the mask), pin 2 will be set to 1, and all other pins will be 0.

Note

For firmware older than revision 3991, bit 2 corresponds to pin 2, bit 3 to pin 1, and bit 5 to pin 5.

Valid for

ADQ214, ADQ112, ADQ114, ADQ214, ADQ412, ADQ108, ADQ208, ADQ1600, SDR14

See also

SetDirectionGPIO(), ReadGPIO()

8.3.41 WriteGPIOPort()

```
virtual int WriteGPIOPort (
    unsigned int    port,
    unsigned int    data,
    unsigned int    mask )
```

Sets the output value of the GPIO pins for a specified GPIO port.

Parameters

port

Port number.

- ADQ7:
  - Port 0: GPIO (12 bits)
  - Port 1: GPDO (3 bits)
  - Port 2: External trigger (1 bit)
  - Port 3: SYNC (1 bit)
- ADQ8:
  - Port 0: Unused
  - Port 1: Unused
  - Port 2: External trigger (1 bit)
  - Port 3: SYNC (1 bit)
- ADQ12/ADQ14:
  - Port 0: GPIO (16 bits)
  - Port 1: GPIOCTRL (5 bits)
  - Port 2: External trigger (1 bit)
  - Port 3: SYNC (1 bit)

Parameters

data	Each bit corresponds to an I/O pin of the port, set desired output value.
mask	Only bits which are zero-valued in the mask will be changed

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

WriteGPIO()

8.3.42 WriteTrig()

```
virtual int WriteTrig (
    int                 data )
```

Sets the output level for the trig output.

Parameters

data	Specifies the level. Set to 0 for low output and 1 for high output.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ1600, ADQ412, ADQ108, ADQ208

See also

SetDirectionTrig()

9 Trigger Options

Functions

- int ArmTimestampSync ()

  Arm the timestamp synchronization (disarm must be done prior to arm)
- int ArmTriggerBlocking ()

  Arm the trigger blocking engine (disarm must be done prior to arm)
- int DisarmTimestampSync ()

Disarm the timestamp synchronization.

- int DisarmTriggerBlocking ()

  Disarm the trigger blocking engine.
- int EnableFrameSync (unsigned int enable)

  Enables frame sync output.
- int EnableLevelTriggerLogicOr (int channel, int enable)

  Enable the level trigger OR functionality.
- int GetExternalTimestamp (unsigned long long *value, unsigned int *valid, unsigned int *status_bits)

  Gets the saved external timestamp.
- unsigned int GetExternTrigEdge (unsigned int *edge)

  Gets the current external trigger edge.
- int GetLvlTrigChannel ()

  Gets the channels for which the level trigger is active.
- int GetLvlTrigEdge ()

  Gets the current level trigger edge.
- int GetLvlTrigLevel ()

  Gets the current level trigger level.
- int GetTimestampSyncCount (unsigned int *count)

  Gets the number of timestamp synchronizations.
- int GetTimestampSyncState (unsigned int *state)

  Gets the state of the timestamp sync.
- int GetTimestampValue (unsigned long long *value)

  Retrieve the current timestamp value.
- unsigned int GetTriggerBlockingGateCount ()

  Returns the current number of detected gates. DisarmTriggerBlocking() resets the counter.
- int GetTriggerEdge (unsigned int trigger, unsigned int *edge)

  Read the edge selection for a specified trigger source.
- int GetTriggerInputImpedance (unsigned int input_num, unsigned int *mode)

  Get the trigger connector input impedance.
- int GetTriggerMode ()

  Gets the current clock source.
- int ResetTimestamp (void)

  Reset the timestamp.
- int SetAuxTriggerMode (int trig_mode)

  Sets the trigger source for the auxiliary trigger path (only usable in the DevKit)
- unsigned int SetExternalTriggerDelay (unsigned char delaycycles)

  Sets the delay of the external trigger to match the data path.
- unsigned int SetExternTrigEdge (unsigned int edge)

  Selects which edge the external trigger input shall trigger on.

- unsigned int SetExtTrigThreshold (unsigned int trignum, double vthresh)

  Sets the threshold voltage of the specified external trigger input.
- unsigned int SetInternalTriggerFrequency (unsigned int Int_Trig_Freq)

  Sets the internal trigger frequency in Hertz.
- int SetInternalTriggerHighLow (unsigned int HighSamples, unsigned int LowSamples)

  Sets the high and low length of the internal trigger.
- unsigned int SetInternalTriggerPeriod (unsigned int TriggerPeriodClockCycles)

  Sets the period of the internal trigger.
- int SetLevelTriggerSequenceLength (unsigned int channel, unsigned int sequence_length)

  Set the level trigger sequence length.
- int SetLvlTrigChannel (int channel)

  Sets the channels to level trig from.
- int SetLvlTrigEdge (int edge)

  Sets the level trigger edge.
- int SetLvlTrigLevel (int level)

  Sets the level trigger code level.
- unsigned int SetSTARBTrigEdge (unsigned int edge)

  Selects which edge the STARB trigger input shall trigger on.
- unsigned int SetSyncTriggerDelay (unsigned char delaycycles)

  Sets the delay of the sync trigger to match the data path.
- int SetTriggerEdge (unsigned int trigger, unsigned int edge)

  Enables generation of trigger events on rising and/or falling trigger edges for a specified trigger source.
- int SetTriggerInputImpedance (unsigned int input_num, unsigned int mode)

  Sets the trigger connector input impedance.
- int SetTriggerMode (int trig_mode)

  Sets how the ADQ should be trigged.
- int SetTriggerThresholdVoltage (unsigned int trigger, double vthresh)

  Sets the threshold voltage of the specified trigger input.
- int SetTrigLevelResetValue (int OffsetValue)

  Sets the level trigger reset value.
- int SetupFrameSync (unsigned int frame_len, unsigned int frame_factor, unsigned int edge)

  Setup frame sync.
- unsigned int SetupLevelTrigger (const int ∗level, const int ∗edge, const int ∗reset_level, unsigned int channel_mask, unsigned int individual_mode)

  Sets up the level trigger.
- int SetupTimestampSync (unsigned int mode, unsigned int trig_source)

  Set up a synchronization of the timestamp counter to a trigger source.
- int SetupTriggerBlocking (unsigned int mode, unsigned int trig_source, uint64_t window_length, unsigned int tcount_limit)

Set up trigger blocking from another trigger source.

- int SWTrig ()

  Triggers the digitizer via a register write.

9.1 Detailed Description

Multiple trigger modes are available for the devices. The most frequently used ones are documented here.

9.2 Function Documentation

9.2.1 ArmTimestampSync()

```
virtual int ArmTimestampSync ( )
```

Arm the timestamp synchronization (disarm must be done prior to arm)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

DisarmTimestampSync()

9.2.2 ArmTriggerBlocking()

```
virtual int ArmTriggerBlocking ( )
```

Arm the trigger blocking engine (disarm must be done prior to arm)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

DisarmTriggerBlocking()

9.2.3 DisarmTimestampSync()

```
virtual int DisarmTimestampSync ( )
```

Disarm the timestamp synchronization.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

ArmTimestampSync()

9.2.4 DisarmTriggerBlocking()

```
virtual int DisarmTriggerBlocking ( )
```

Disarm the trigger blocking engine.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

ArmTriggerBlocking()

9.2.5 EnableFrameSync()

```
virtual int EnableFrameSync (
    unsigned int      enable )
```

Enables frame sync output.

Enables frame sync output on the SYNC connector

Parameters

enable	Set to 1 to enable frame sync output, 0 to disable.

Returns

1 for success, 0 otherwise

Valid for

ADQ12, ADQ14, ADQ7

See also

SetupFrameSync()

9.2.6 EnableLevelTriggerLogicOr()

```
virtual int EnableLevelTriggerLogicOr (
    int               channel,
    int               enable )
```

Enable the level trigger OR functionality.

This function enables or disables logic `OR` between level trigger events and events from the channel's trigger source specified through either SetTriggerMode() or SetParameters().

Parameters

channel	The target channel, indexed from 1 and upwards.
enable	Set to a nonzero value to enable, zero to disable.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ8

9.2.7 GetExternalTimestamp()

```
virtual int GetExternalTimestamp (
    unsigned long
    long *            value,
    unsigned int *    valid,
    unsigned int *    status_bits )
```

Gets the saved external timestamp.

Parameters

value	Pointer to timestamp value
valid	Status for the returned timestamp • 0: Timestamp is not valid (reset event has not occurred yet) • 1: Timestamp is valid (reset event has occurred since last ArmTimestampSync)
status_bits	Extra status bits, see White Rabbit App Note for more information

Returns

1

Valid for

ADQ7

See also

SetupTimestampSync() ArmTimestampSync()

9.2.8 GetExternTrigEdge()

```
virtual unsigned int GetExternTrigEdge (
    unsigned int *    edge )
```

Gets the current external trigger edge.

Parameters

edge Pointer to where the result is returned. One of these values are returned: ▪ 0: Falling edge ▪ 1: Rising edge	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, SDR14, ADQ1600, ADQ12, ADQ14

See also

SetExternTrigEdge()

9.2.9 GetLvlTrigChannel()

```
virtual int GetLvlTrigChannel ( )
```

Gets the channels for which the level trigger is active.

The value returned from this function is a bitmask that indicates which channels that are active.
Return value = 0: None
Return value = 1: Channel A
Return value = 2: Channel B
Return value = 4: Channel C
Return value = 8:Channel D
Return value = 10 => Both Channel B and D
Return value = 15 => All Channels

Returns

A bitmask with the active channels

Valid for

ADQ208, ADQ212, ADQ214, ADQ412, SDR14, ADQ12, ADQ14

See also

SetLvlTrigChannel()

9.2.10 GetLvlTrigEdge()

```
virtual int GetLvlTrigEdge ( )
```

Gets the current level trigger edge.

Returns

1 for rising edge and 0 for falling edge

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14

See also

SetLvlTrigEdge()

9.2.11 GetLvlTrigLevel()

```
virtual int GetLvlTrigLevel ( )
```

Gets the current level trigger level.

Please see SetlvlTrigLevel for more information.

Returns

The current set level

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14

See also

SetLvlTrigLevel()

9.2.12 GetTimestampSyncCount()

```
virtual int GetTimestampSyncCount (
    unsigned int *    count )
```

Gets the number of timestamp synchronizations.

The counter is reset by DisarmTimestampSync().

Parameters

count Pointer to an unsigned int where the result is stored.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7

9.2.13 GetTimestampSyncState()

```
virtual int GetTimestampSyncState (
    unsigned int *    state )
```

Gets the state of the timestamp sync.

Parameters

state	Pointer to unsigned int for the result (0 is not synchronized, 1 is synchronized)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ12, ADQ14, ADQ8

See also

ArmTimestampSync()

9.2.14 GetTimestampValue()

```
virtual int GetTimestampValue (
    unsigned long
    long *            value )
```

Retrieve the current timestamp value.

Capture and read out the digitizer's monotonically increasing timestamp. This function has no guarantees on timing.

Parameters

value	Pointer to timestamp value

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ14, ADQ12

9.2.15 GetTriggerBlockingGateCount()

```
virtual unsigned int GetTriggerBlockingGateCount ( )
```

Returns the current number of detected gates. DisarmTriggerBlocking() resets the counter.

Deprecated from firmware revision 43153 for ADQ7, please use GetTimestampSyncCount() instead.

Returns

The number of detected gates.

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

GetTimestampSyncCount()

9.2.16 GetTriggerEdge()

```
virtual int GetTriggerEdge (
    unsigned int      trigger,
    unsigned int *    edge )
```

Read the edge selection for a specified trigger source.

Parameters

trigger	The selected trigger source, see SetTriggerEdge for numbering
edge	Pointer to an unsigned int where the value should be stored

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTriggerEdge()

9.2.17 GetTriggerInputImpedance()

```
virtual int GetTriggerInputImpedance (
    unsigned int      input_num,
```

```
    unsigned int *    mode )
```

Get the trigger connector input impedance.

Parameters

input_num	Set to 1 for external trigger input connector and 2 for sync connector
mode	Return argument, 0 means 50 ohms input, 1 means high impedance mode

Returns

1 for successful operation and 0 for failure

Note

When SYNC is enabled as GPIO, the API SetTriggerInputImpedance will not have any effect for SYNC pin.

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

EnableGPIOPort()

9.2.18 GetTriggerMode()

```
virtual int GetTriggerMode ( )
```

Gets the current clock source.

Please see SetTriggerMode() for an explanation of the values.

Returns

The clock source

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7

See also

SetTriggerMode()

9.2.19 ResetTimestamp()

```
virtual int ResetTimestamp (
    void    )
```

Reset the timestamp.

Reset the digitizer's monotonically increasing timestamp to zero. This function has no guarantees on timing.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ14, ADQ12

9.2.20 SetAuxTriggerMode()

```
virtual int SetAuxTriggerMode (
    int                trig_mode )
```

Sets the trigger source for the auxiliary trigger path (only usable in the DevKit)

Parameters

trig_mode

The selected trigger mode:

- 1: Software trigger
- 2: External trigger
- 3: Level trigger (ADQ12/ADQ14 only)
- 4: Internal trigger
- 6: PXIe STARB trigger (ADQ12/ADQ14 only)
- 9: SYNC trigger
- 10: MLVDS trigger
- 12: External trigger, resynchronized to reference clock
- 14: PXI triggers
- 16: PXIe STARB trigger, resynchronized to reference clock (ADQ12/ADQ14 only)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7

See also

SetTriggerMode()

9.2.21 SetExternalTriggerDelay()

```
virtual unsigned int SetExternalTriggerDelay (
    unsigned char      delaycycles )
```

Sets the delay of the external trigger to match the data path.

In default configurations this is setup correctly by the API. If there is additional delay in user configured logic, this API call may be used to compensate correctly.

Parameters

delaycycles
The number of data path clock cycles to delay the external trigger. For ADQ7 and ADQ8, valid values are 1-127. For ADQ12 / ADQ14 the only valid values are 0 and 37.

Returns

1 for successful operation and 0 for failure

Note

- Data path clock cycle is 4 samples on ADQ12, ADQ14, ADQ112 and ADQ114 and 2 samples on ADQ214 and ADQ212.
- Data path clock cycle is 32 samples on ADQ7-10GSPS mode and 16 samples on ADQ7-5GSPS mode (each step is 3.2ns)
- Data path clock cycle is 4 ns on ADQ8.

Valid for

ADQ214, ADQ114, ADQ212, ADQ112, ADQ7, ADQ12, ADQ14, ADQ8

See also

SetTriggerMode()

9.2.22 SetExternTrigEdge()

```
virtual unsigned int SetExternTrigEdge (
    unsigned int        edge )
```

Selects which edge the external trigger input shall trigger on.

Parameters

edge

Specifies which edge to use:
- 0: Falling edge
- 1: Rising edge
- 2: Both edges (ADQ12 / ADQ14 only)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, SDR14, ADQ1600, ADQ12, ADQ14

See also

SetTriggerMode()

9.2.23 SetExtTrigThreshold()

```
virtual unsigned int SetExtTrigThreshold (
    unsigned int     trignum,
    double           vthresh )
```

Sets the threshold voltage of the specified external trigger input.

Parameters

trignum	The trigger number. Allowed numbers are hardware dependent (some boards only have trig1, others have 1,2,3, etc.).
vthresh	The Threshold voltage. 0.5V is default.

Returns

1 for successful operation and 0 for failure

Note

Not all hardware supports threshold adjustment, see HasVariableTrigThreshold()

Valid for

ADQ1600, ADQ12, ADQ14

See also

HasVariableTrigThreshold(), SetTriggerMode()

9.2.24 SetInternalTriggerFrequency()

```
virtual unsigned int SetInternalTriggerFrequency (
    unsigned int     Int_Trig_Freq )
```

Sets the internal trigger frequency in Hertz.

The actual frequency set by the ADQ is an **approximation** of the desired frequency. It is also dependent on the sampling frequency. Thus, if the sampling frequency is changed, please set the trigger period manually using SetInternalTriggerPeriod().

Parameters

Int_Trig_Freq	The frequency in Hertz

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ208, ADQ1600, ADQ412, SDR14

See also

SetInternalTriggerPeriod(), SetTriggerMode()

9.2.25 SetInternalTriggerHighLow()

```
virtual int SetInternalTriggerHighLow (
    unsigned int       HighSamples,
    unsigned int       LowSamples )
```

Sets the high and low length of the internal trigger.

Parameters

HighSamples	The amount of time the internal trigger should remain high, in ADC-sample units
LowSamples	The amount of time the internal trigger should remain low, in ADC-sample units

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

SetInternalTriggerPeriod()

9.2.26 SetInternalTriggerPeriod()

```
virtual unsigned int SetInternalTriggerPeriod (
    unsigned int       TriggerPeriodClockCycles )
```

Sets the period of the internal trigger.

Parameters

TriggerPeriodClockCycles The trigger period given as a number of internal clock cycles. The internal clock cycle length is related to the sample clock frequency according to the below: • ADQ112, ADQ114 & SDR14: Period = TriggerPeriodClockCycles $*$ (4 / F_s) • ADQ212 & ADQ214: Period = TriggerPeriodClockCycles $*$ (2 / F_s) • ADQ108 & ADQ208: Period = TriggerPeriodClockCycles $*$ (32 / F_s) • ADQ1600: Period = TriggerPeriodClockCycles $*$ (8 / F_s) • ADQ412: Period = TriggerPeriodClockCycles $*$ (8 / F_s), where F_s is the four-channel mode sampling frequency • ADQ12, ADQ14, ADQ7, ADQ8: Period = TriggerPeriodClockCycles $*$ (1 / F_s)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetInternalTriggerFrequency()

9.2.27 SetLevelTriggerSequenceLength()

```
virtual int SetLevelTriggerSequenceLength (
    unsigned int      channel,
    unsigned int      sequence_length )
```

Set the level trigger sequence length.

The *sequence length* is the minimum number of samples required to be above or below the level trigger threshold in order for a trigger event to be generated.

Parameters

channel The target channel, indexed from 1 and upwards.
sequence_length The length of the trigger sequence, i.e. the minimum number of samples required to have crossed the level trigger threshold. A sequence length of 1 sample is the default value. The value has to fall in the range: • ADQ7: [1, 4]

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7

See also

SetupLevelTrigger()

9.2.28 SetLvlTrigChannel()

```
virtual int SetLvlTrigChannel (
    int                channel )
```

Sets the channels to level trig from.

Parameters

channel
A mask that tells the ADQ which channels that are active
- channels = 0 => None
- channels = 1 => Channel A
- channels = 2 => Channel B
- channels = 4 => Channel C
- channels = 8 => Channel D

To trig on multiple channels add the channel code for each individual channel. Examples:
- channels = 10 => Any of Channel B and D
- channels = 15 => Any Channel Values given for any non-existing channel is simply ignored

Returns

1 for successful operation and 0 for failure

Note

When interleaving on ADQ412 or ADQ208, enable level trigger for both channels that are interleaved (that is, use ChannelCode = 0, 3, 12 or 15). This is because channel A&B and C&D are interleaved.

For V6 digitizers, the function SetupLevelTrigger may be used to set up all level trigger parameters in one call.

On ADQ7 only a single bit in the mask can be set, only one channel is allowed to at once generate the trigger. On ADQ12 and ADQ14 only masks 1,2,4,8 (single bit) and masks 3 (A+B), 12 (C+D), 15 (A+B+C+D) are allowed.

Valid for

ADQ108, ADQ1600, ADQ208, ADQ212, ADQ214, ADQ412, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetupLevelTrigger(), GetLvlTrigChannel(), SetLvlTrigEdge()

9.2.29 SetLvlTrigEdge()

```
virtual int SetLvlTrigEdge (
```

```
    int                edge )
```

Sets the level trigger edge.

Parameters

edge
Trigger edge

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetLvlTrigLevel(), SetTriggerMode()

9.2.30 SetLvlTrigLevel()

```
virtual int SetLvlTrigLevel (
    int                level )
```

Sets the level trigger code level.

Parameters

level
Trigger level in codes

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetLvlTrigEdge(), SetTriggerMode()

9.2.31 SetSTARBTrigEdge()

```
virtual unsigned int SetSTARBTrigEdge (
    unsigned int       edge )
```

Selects which edge the STARB trigger input shall trigger on.

Parameters

edge	Specifies which edge to use: ▪ 0: Falling edge ▪ 1: Rising edge ▪ 2: Both edges

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14

See also

SetTriggerMode()

9.2.32 SetSyncTriggerDelay()

```
virtual unsigned int SetSyncTriggerDelay (
    unsigned char     delaycycles )
```

Sets the delay of the sync trigger to match the data path.

In default configurations this is setup correctly by the API. If there is additional delay in user configured logic, this API call may be used to compensate correctly.

Parameters

delaycycles	The number of data path clock cycles to delay the external trigger. For ADQ7, valid values are 1-63

Returns

1 for successful operation and 0 for failure

Note

1 data path clock cycle is 32 samples on ADQ7-10GSPS mode and 16 samples on ADQ7-5GSPS mode

Valid for

ADQ7

See also

SetTriggerMode()

9.2.33 SetTriggerEdge()

```
virtual int SetTriggerEdge (
```

```
    unsigned int        trigger,
    unsigned int        edge )
```

Enables generation of trigger events on rising and/or falling trigger edges for a specified trigger source.

Parameters

trigger The selected trigger source: ▪ 1: Software trigger ▪ 2: External trigger 1 ▪ 3: Level trigger ▪ 4: Internal trigger ▪ 6: PXI triggers ▪ 9: SYNC connector ▪ 10: MTCA MLVDS triggers ▪ 25: GPIO Trig 0 (ADQ7-PCIe/PXIe only) ▪ 26: GPIO Trig 1 (ADQ7-PCIe/PXIe only)	
edge Specifies which edge should generate a trigger event: ▪ 0: Falling edge ▪ 1: Rising edge ▪ 2: Both edges	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTriggerMode()

9.2.34 SetTriggerInputImpedance()

```
virtual int SetTriggerInputImpedance (
    unsigned int        input_num,
    unsigned int        mode )
```

Sets the trigger connector input impedance.

Parameters

input_num Set to 1 for external trigger input connector and 2 for sync connector	
mode Set to 0 for 50 ohm input, set to 1 for high impedance mode	

Returns

1 for successful operation and 0 for failure

Note

When SYNC is enabled as GPIO, the API SetTriggerInputImpedance will not have any effect for SYNC pin.

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

EnableGPIOPort()

9.2.35 SetTriggerMode()

```
virtual int SetTriggerMode (
    int                trig_mode )
```

Sets how the ADQ should be trigged.

Parameters

trig_mode

The selected trigger mode:

- 1: Software trigger only
- 2: External trigger 1
- 3: Level trigger
- 4: Internal trigger
- 6: PXIe STARB trigger
- 7: External trigger 2
- 8: External trigger 3
- 9: SYNC connector
- 10: MLVDS (MTCA backplane) (ADQ12 / ADQ14 / ADQ7 / ADQ8 only)
- 11: External trigger 1, gated with sync input (ADQ7 / ADQ8 only)
- 12: External trigger 1, resynchronized to clock reference (ADQ12 / ADQ14 / ADQ8 only)
- 13: MLVDS (MTCA backplane), resynchronized to clock reference (ADQ12 / ADQ14 / ADQ7 / ADQ8 only)
- 14: PXI Trig (PXI_TRIG0, PXI_TRIG1, PXI_STARA) (ADQ12 / ADQ14 / ADQ7 / ADQ8 only)
- 16: PXIe STARB trigger, resynchronized to clock reference (ADQ12 / ADQ14)
- 19: SYNC connector, resynchronized to clock reference
- 20: Reserved for internal use (ADQ12 / ADQ14 only)
- 21: Reserved for internal use (ADQ12 / ADQ14 only)
- 23: Daisy chain trigger (ADQ8 only)
- 24: Software trigger, resynchronized to clock reference (ADQ8 only)
- 25: GPIO Trig 0 (ADQ7-PCIe/PXIe only)
- 26: GPIO Trig 1 (ADQ7-PCIe/PXIe only)
- 920: Reserved for internal use (ADQ12 / ADQ14 only)
- 921: Reserved for internal use (ADQ12 / ADQ14 only)

Returns

1 for successful operation and 0 for failure

Note

The software trigger is always enabled regardless of mode

External triggers 2 and 3 are not available on all ADQs. SYNC connector is only available on ADQ12 / ADQ14.

MLVDS triggers to listen for is configured with SetTriggerMaskMLVDS (also needs to be configured as inputs with SetDirectionMLVDS)

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SWTrig(), SetTriggerMaskMLVDS(), SetDirectionMLVDS(), SetTriggerMaskPXI()

9.2.36 SetTriggerThresholdVoltage()

```
virtual int SetTriggerThresholdVoltage (
    unsigned int        trigger,
    double              vthresh )
```

Sets the threshold voltage of the specified trigger input.

Parameters

trigger The trigger number, see SetTriggerMode() for numbering
vthresh The Threshold voltage. 0.5V is default.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ8

See also

SetTriggerMode()

9.2.37 SetTrigLevelResetValue()

```
virtual int SetTrigLevelResetValue (
    int                 OffsetValue )
```

Sets the level trigger reset value.

Parameters

OffsetValue The offset from the trigger level for which the level trigger shall arm the trigger for detecting rising or falling edges

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, SDR14, ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetLvlTrigLevel(), SetupLevelTrigger()

9.2.38 SetupFrameSync()

```
virtual int SetupFrameSync (
    unsigned int    frame_len,
    unsigned int    frame_factor,
    unsigned int    edge )
```

Setup frame sync.

Frame sync generates a 'frame_length' long pulse on every 'frame_factor' trigger. E.g. For frame_length = 128 and reduction_factor = 7, on every 7th trigger edge a 128 samples long pulse will be generated on the SYNC connector.

If the length is larger than the trigger period, a constant high signal will be outputted.

Parameters

frame_len The length of the frame, in samples. The length will be truncated to the nearest multiple of parallel samples.
frame_factor Every 'frame_factor' trigger will generate a sync pulse
edge Specify trigger edges: • 0: Falling edge • 1: Rising edge • 2: Both edges

Returns

1 for success, 0 otherwise

Note

Frame sync output also has to be enabled by calling the EnableFrameSync() function.

Valid for

ADQ12, ADQ14, ADQ7

See also

EnableFrameSync()

9.2.39 SetupLevelTrigger()

```
virtual unsigned int SetupLevelTrigger (
    const int *        level,
    const int *        edge,
    const int *        reset_level,
    unsigned int       channel_mask,
    unsigned int       individual_mode )
```

Sets up the level trigger.

This function may be used to set up all level trigger options in one call. It also has the option to set individual values for different channels. When using individual mode, care must be taken to provide the correct amount of values at the inputs.

Parameters

level A pointer to the values that the level trigger should trigger on. In individual mode, `level`[0] will be used for channel A, and `level`[1] for channel B etc. In non-individual the code value at `level`[0] will be used for all channels.
edge A pointer to the edge that the level trigger should trigger on. Set to 1 for rising edge and 0 for falling. In individual mode, edge[0] will be used for channel A, and edge[1] for channel B etc. In non-individual the value at edge[0] will be used for all channels.
reset_level A pointer to the code values should be used as reset values for the level trigger. The reset level is the offset from the trigger level which the signal must pass in order to arm the level trigger. A low level results in a more sensitive but less noise-resistant trigger. By setting the value to -1, the default value for the device will be used. In individual mode, `reset_level`[0] will be used for channel A, and `reset_level`[1] for channel B etc. In non-individual the value at `reset_level`[0] will be used for all channels.

Parameters

channel_mask A bit field that specifies which channels that are active: ▪ bit 0: Set to 1 to activate channel 1/A level trigger and 0 to deactivate ▪ bit 1: Set to 1 to activate channel 2/B level trigger and 0 to deactivate ▪ bit 2: Set to 1 to activate channel 3/C level trigger and 0 to deactivate ▪ bit 3: Set to 1 to activate channel 4/D level trigger and 0 to deactivate ▪ bit 4: Set to 1 to activate channel 5 level trigger and 0 to deactivate ▪ bit 5: Set to 1 to activate channel 6 level trigger and 0 to deactivate ▪ bit 6: Set to 1 to activate channel 7 level trigger and 0 to deactivate ▪ bit 7: Set to 1 to activate channel 8 level trigger and 0 to deactivate For example, a value of 15 ($2^3+2^2+2^1+2^0$) will activate all 4 channels. Note: On ADQ12 / ADQ14, only one single bit is supported (1,2,4,8) or the special combinations 3=A+B, 12=C+D, 15=A+B+C+D. Setting other masks will generate an error.
individual_mode Specifies whether the level trigger should be set up with different values for different channels (`individual_mode` = 1) or not (`individual_mode` = 0). If individual mode is not active, the first value of each input array is used for all available channels.

Returns

1 for successful operation and 0 for failure

Example

On ADQ412, the code below sets the level trigger to trigger on the rising edge at level 1024 for channel A and the falling edge at level 87 for channel D. The other channels are deactivated.

```
int level[4] = [1024, 0, 0, 87];
int edges[4] = [1 0 0 0];
int reset_level = [7 7 7 7];
channel_mask = 0x9; // 2^3 + 2^0
individual_mode = 1;
ADQ_SetupLevelTrigger(adq_cu_ptr, adq_num, level, edges, reset_level, channel_mask, individual_mode);
```

Note

When interleaving ADQ412 or ADQ208, all channels must still be specified if SetupLevelTrigger() is used in individual mode. For example, on ADQ412 channel A and B should have the same values, and C and D should have the same values.

Valid for

ADQ412, ADQ208, SDR14, ADQ1600, ADQ108, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetLvlTrigLevel(), SetLvlTrigEdge(), SetTrigLevelResetValue(), SetTriggerMode()

9.2.40 SetupTimestampSync()

```
virtual int SetupTimestampSync (
    unsigned int        mode,
    unsigned int        trig_source )
```

Set up a synchronization of the timestamp counter to a trigger source.

Parameters

mode	The synchronization mode, as follows: 0: Synchronize only on the first trigger event 1: Synchronize on all trigger events (until disarmed)
trig_source	See SetTriggerMode() for trigger source numbering ▪ Level trigger cannot be used for timestamp synchronization.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

9.2.41 SetupTriggerBlocking()

```
virtual int SetupTriggerBlocking (
    unsigned int    mode,
    unsigned int    trig_source,
    uint64_t        window_length,
    unsigned int    tcount_limit )
```

Set up trigger blocking from another trigger source.

After SetupTriggerBlocking() the functions DisarmTriggerBlocking() and ArmTriggerBlocking() should be called to enable the trigger blocking functionallity.

All triggers are blocked after setup. To disable the trigger blocking logic use mode DISABLE.

Parameters

mode	Set trigger blocking mode. ▪ 0 (ONCE): Block until the first trigger is detected. ▪ 1 (WINDOW): Unblock for window_length samples after the trigger. ▪ 2 (GATE): Unblock while the trigger is high. ▪ 3 (INVERSE WINDOW): Block for *window_length* samples after the trigger. ▪ 4 (DISABLE) Disable trigger blocking (ADQ7, ADQ14, ADQ12, ADQ8).

Parameters

trig_source

• Bits 0 to 7: The trigger blocking source. See SetTriggerMode() for source numbering. The accepted values change depending on the *tcount mode*.
• Bits 8 to 11: The *tcount mode*
  – 0: Disabled. Valid trigger blocking sources are
    ∗ External trigger
    ∗ SYNC connector
    ∗ External trigger, gated with sync input (ADQ7 only)
    ∗ GPIO Trig 0/1 (ADQ7-PCIe/PXIe only)
  – 1: One trigger blocking event will be passed on once *tcount_limit* triggers have been observed on the specified trigger blocking source. Any trigger source supported by SetTriggerMode() is valid as a trigger blocking source. The *mode* GATE is not supported in this mode.
  – 2: As mode 1, but the trigger blocking event is delayed until an edge on the signal input on the SYNC connector has been detected.
• Bits 12 to 31: Reserved

window_length Set the trigger blocking window length (in samples). Only applicable in mode 1 (WINDOW) and 3 (INVERSE WINDOW). ADQ7 and ADQ8 support a 64-bit window length. Other products are restricted to a 32-bit window length chosen as the low part of the input argument.
tcount_limit Specifies the number of triggers on the trigger blocking source to wait for before passing on a trigger blocking event. Only used when *tcount mode* is non-zero.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTriggerMode() SetTriggerEdge() ArmTriggerBlocking() DisarmTriggerBlocking()

9.2.42 SWTrig()

```
virtual int SWTrig ( )
```

Triggers the digitizer via a register write.

The ADQ device cannot be trigged when trigger is disarmed. When the trigger is disarmed the memory counter is reset, so next time ArmTrigger() is called and the device records a record of data, this record will overwrite the previous first record.

Returns

1 for successful operation and 0 for failure

Note

Waveform Averaging (WFA) has its own software trigger implementation, see WaveformAveragingSoftwareTrigger()

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTriggerMode()

10 Advanced Trigger Options

Functions

- unsigned int AWGSetTriggerEnable (unsigned int dacId, unsigned int bitflags)

  Selects which trigger signals that may trigger the AWG.
- unsigned int AWGTrigoutDisarm (unsigned int dacId)

  Disarms the trigger output of the specified AWG.
- unsigned int ClearInternalTriggerCounts ()

  Resets the count of internal triggers issued.
- int DaisyChainEnable (unsigned int enable)

  Enable the digitizer's local daisy chain trigger mechanism.
- int DaisyChainEnableOutput (unsigned int enable)

  Enable/disable the output buffer and select the daisy chain output as the source.
- int DaisyChainGetInputState (unsigned int ∗state)

  Get the state of the daisy chain input.
- int DaisyChainGetNofPretriggerSamples (int position, int64_t sample_rate, int ∗nof_pretrigger_samples)

  Get the number of extra pretrigger samples needed due to the daisy chain.
- int DaisyChainGetStatus (unsigned int ∗status)

  Get the digitizer's daisy chain status.
- int DaisyChainGetTriggerInformation (unsigned int source, unsigned int edge, int level, unsigned int channel, unsigned int start_record_number, unsigned int nof_records, unsigned int record_length, struct ADQDaisyChainDeviceInformation ∗device_info, unsigned int nof_devices, struct ADQDaisyChainTriggerInformation ∗trig_info)

  Get the trigger information from the daisy chain master device.
- int DaisyChainReset (void)

  Resets the digitizer's daisy chain mechanism.
- int DaisyChainSetMode (unsigned int mode)

  Set the daisy chain mode.
- int DaisyChainSetOutputState (unsigned int state)

  Set the state of the daisy chain output.
- int DaisyChainSetTriggerSource (unsigned int trig_source)

  Set the trigger source for the daisy chain master.

- int DaisyChainSetupLevelTrigger (unsigned int channel, int level, int arm_hysteresis, unsigned int edge)

  Set up the daisy chain level trigger.
- int DaisyChainSetupOutput (unsigned int sync_polarity, unsigned int sync_immediate, unsigned int sync_length)

  Set up the shape of the daisy chain output pulse.
- unsigned int DisableInternalTriggerCounts ()

  Disables the internal trigger counter.
- unsigned int EnableInternalTriggerCounts ()

  Enables the internal trigger counter.
- unsigned int EnablePXIeTriggers (unsigned int port, unsigned int bitflags)

  Selects which PXIe triggers to trigger on.
- unsigned int EnablePXIeTrigout (unsigned int port, unsigned int bitflags)

  Selects which PXIe triggers outputs to use.
- unsigned int GetPPTStatus ()

  Gets the status register of the Precise Period Trigger function.
- unsigned int InitPPT ()

  Initializes the Precise Period Trigger.
- unsigned int PXIeSoftwareTrigger ()

  Sends a trigger signal on all enabled trigger outputs.
- unsigned int SetInternalTriggerCounts (unsigned int trigger_counts)

  Sets the number of triggers that will be issued by the internal trigger.
- unsigned int SetInternalTriggerSyncMode (unsigned int mode)

  Sets up and activates the internal trigger synchronization.
- unsigned int SetPPTActive (unsigned int active)

  Activates or deactivates Precise Period Trigger.
- unsigned int SetPPTBurstMode (unsigned int burst_mode)

  Activates or deactivates the Precise Period Trigger burst mode.
- unsigned int SetPPTInitOffset (unsigned int init_offset)

  Sets the Precise Period Trigger initial offset.
- unsigned int SetPPTPeriod (unsigned int period)

  Sets the Precise Period Trigger period.
- unsigned int SetPXIeTrigDirection (unsigned int trig0output, unsigned int trig1output)

  Sets the direction of the two PXI_TRIG I/O pins.
- unsigned int SetTriggerGate (unsigned int enabled, unsigned int mode, unsigned int gate_mux)

  Sets up and activates the trigger gating functionaliy.
- int SetTriggerMaskPXI (unsigned char mask)

  Sets the mask for accepting triggers from PXI.
- unsigned int WriteSTARBDelay (unsigned int starbdelay, unsigned int writetoeeprom)

  Writes a delay value for the PXIe DSTARB trigger input.

10.1 Detailed Description

This section documents the non-standard trigger modes and options.

10.2 Function Documentation

10.2.1 AWGSetTriggerEnable()

```
virtual unsigned int AWGSetTriggerEnable (
    unsigned int    dacId,
    unsigned int    bitflags )
```

Selects which trigger signals that may trigger the AWG.

Parameters

dacId	Selects the AWG/DAC (1 or 2)
bitflags	A bit field where each bit enables a trigger if asserted, according to: • bit 0: Host trigger/software trigger from the data acquisition logic. • bit 1: External trigger • bit 2: PXIe port1 trigger • bit 3: Internal trigger • bit 4 AWG Software trigger (AWGTrig)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

SetTriggerMode()

10.2.2 AWGTrigoutDisarm()

```
virtual unsigned int AWGTrigoutDisarm (
    unsigned int    dacId )
```

Disarms the trigger output of the specified AWG.

Parameters

dacId	Selects the AWG/DAC (1 or 2)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGTrigoutArm()

10.2.3 ClearInternalTriggerCounts()

```
virtual unsigned int ClearInternalTriggerCounts ( )
```

Resets the count of internal triggers issued.

Every time the function is called the counting will restart from zero.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

SetInternalTriggerCounts()

10.2.4 DaisyChainEnable()

```
virtual int DaisyChainEnable (
    unsigned int      enable )
```

Enable the digitizer's local daisy chain trigger mechanism.

This function enables the digitizer's local daisy chain mechanism allowing triggers to propagate to the acquisition engine and trigger records. Calling this function does *not* enable the SYNC connector's output buffer. Refer to DaisyChainEnableOutput() for that functionality.

Parameters

enable	• 0 to disable • 1 to enable

Returns

1 for success, 0 otherwise

Valid for

ADQ8

See also

DaisyChainEnableOutput

10.2.5 DaisyChainEnableOutput()

```
virtual int DaisyChainEnableOutput (
    unsigned int      enable )
```

Enable/disable the output buffer and select the daisy chain output as the source.

This function activates the buffer for the connector associated with the daisy chain output signal. The connector is often shared between several digitizer functions (e.g. GPIO) but only one function can be activated at a time.

Parameters

enable
▪ 0: Disable ▪ 1: Enable

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.6 DaisyChainGetInputState()

```
virtual int DaisyChainGetInputState (
    unsigned int *    state )
```

Get the state of the daisy chain input.

Parameters

state
Reference to memory where the state should be written to. ▪ 0: The daisy chain input is logic low. ▪ 1: The daisy chain input is logic high.

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.7 DaisyChainGetNofPretriggerSamples()

```
virtual int DaisyChainGetNofPretriggerSamples (
    int          position,
    int64_t      sample_rate,
    int *        nof_pretrigger_samples )
```

Get the number of extra pretrigger samples needed due to the daisy chain.

Using the daisy chain trigger mechanism comes with a requirement that each slave device increases its pretrigger length. This is needed to capture the data at the true trigger point which occurs *earlier* in time for each slave (relative to its daisy chain trigger). The true trigger position is reconstructed from the timing information read from the master device.

Parameters

position The digitizer's position in the daisy chain. ▪ 0: The master device. ▪ 1: The first slave device. ▪ 2: The second slave device. ▪ N: The Nth slave device. An **important** note is that if a number of slave devices are *grouped* together with the `sync_immediate` setting in DaisyChainSetupOutput() they will identify as the same position.
sample_rate The sample rate of the device in hertz, including sample skip if used.
nof_pretrigger_samples Reference to memory where the number of pretrigger samples should be written to.

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.8 DaisyChainGetStatus()

```
virtual int DaisyChainGetStatus (
    unsigned int *    status )
```

Get the digitizer's daisy chain status.

Parameters

status	Reference to memory where the status should be written to. The status value is a 32-bit unsigned number interpreted as ▪ Bit 3: Output synchronizer proximity error. ▪ Bit 2: Input synchronizer proximity error. ▪ Bit 1: Input synchronizer delay error. ▪ Bit 0: State of the daisy chain input. – 1: Logic high – 0: Logic low

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.9 DaisyChainGetTriggerInformation()

```
virtual int DaisyChainGetTriggerInformation (
    unsigned int        source,
    unsigned int        edge,
    int                 level,
    unsigned int        channel,
    unsigned int        start_record_number,
    unsigned int        nof_records,
    unsigned int        record_length,
    struct
    ADQDaisyChainDeviceInformation
    *                   device_info,
    unsigned int        nof_devices,
    struct
    ADQDaisyChainTriggerInformation
    *                   trig_info )
```

Get the trigger information from the daisy chain master device.

This function reads out the trigger information from the daisy chain master device and returns the data in the `trig_info` parameter. Currently **only** supports the multirecord data collection mode.

Parameters

source	The trigger source used by the daisy chain mechanism. Has to be the same value used in the call to DaisyChainSetTriggerSource().

Parameters

edge

The edge selection. Only used when the level trigger is specified as the trigger source. This value is expected to match the value provided in the call to DaisyChainSetupLevelTrigger().

- 0: Falling edge
- 1: Rising edge

level

The level trigger threshold value. This parameter is only used when the level trigger is specified as the trigger souce and is expected to match the value provided to DaisyChainSetupLevelTrigger().

channel

The target channel. If the level trigger is used as the trigger source, then the value is expected to point to the channel sampling the trigger signal. Otherwise, any active channel index will suffice. Channels are indexed from 1 and upwards.

start_record_number

Similar to GetData(), this parameter instructs the API for which record to begin constructing the trigger information, skipping any records up until this point. The trigger information at index zero in the array `trig_info` will represent the trigger information for the record corresponding to `start_record_number`.

nof_records

The number of records to collect timing information from. This value is used to define the length of the array starting at `trig_info`.

record_length

The record length in samples. This value is expected to be the same as the value provided to the acquisition setup function, e.g. MultiRecordSetup().

device_info

A pointer to the first element in an array of ADQDaisyChainDeviceInformation structs. Each device in the daisy chain is represented by an entry in the array and the array length is provided as `nof_devices`. The entry is of type ADQDaisyChainDeviceInformation and lists the position, the number of pretrigger samples,the trigger delay in samples, and the sample rate in hertz. The sample rate should include the sample skip if used. No restrictions on the order of the elements are imposed by the API but the order will be reflected in the array pointed to by `RecordStart` in `trig_info`. The struct is defined in the ADQAPI header as

```
struct ADQDaisyChainDeviceInformation {
 int64_t SampleRate;
 int64_t Position;
 int64_t PretriggerSamples;
 int64_t TriggerDelaySamples;
};
```

nof_devices

The length of the array pointed to by `device_info`.

Parameters

trig_info

A pointer to the first element in an array of ADQDaisyChainTriggerInformation structs. This array is **allocated** by the user and **filled in** by the API. Each entry represents the trigger information for one record across all channels and devices. Hence, the expected length of the array is `nof_records`. The struct is defined in the ADQAPI header as:

```
struct ADQDaisyChainTriggerInformation {
 uint64_t Timestamp;
 int64_t *RecordStart;
 double *ExtendedPrecision;
};
```

- The `Timestamp` field indicates the time of the trigger point and is measured in 25 ps steps on ADQ8.
- The `RecordStart` field is a pointer to an array of 64-bit integers representing the `RecordStart` values of all devices in the daisy chain and thus is expected to have the same length as the `device_info` array, i.e. `nof_devices`.
  A `RecordStart` value indicates the relation between the first sample in the record and the timestamp. The timestamp of the first sample is calculated as `Timestamp` + `RecordStart`. Thus, negative values indicates that the trigger point occurs **after** the first sample in the record and vice versa for positive values.
  The elements are ordered according to the `device_info` array, i.e. the `RecordStart` value at index 3 will be the new value for the device at index 3 in the `device_info` array for that record.
- The `ExtendedPrecision` field holds a non-zero value if the selected trigger method can yield a higher resolution than what the `Timestamp` and `RecordStart` fields can represent. Currently, this field is only non-zero when the *level trigger* is used as the daisy chain source since the data is subjected to interpolation. The value will be in the range [0,1) and should be *subtracted* from `Timestamp` and *added* to `RecordStart`.

The `Timestamp` and the `RecordStart` values provided by this function is intended to **overwrite** the values reported in the headers returned by any data collection call, e.g. GetDataWHTS().

Returns

1 for success, 0 otherwise

Note

To obtain proper daisy chain timing information, both master and slaves of a chain must be set to trigger mode "Daisy chain trigger". The master is then configured with which trigger to use for the chain by using API call DaisyChainSetTriggerSource

Valid for

ADQ8

See also

DaisyChainSetTriggerSource()

10.2.10 DaisyChainReset()

```
virtual int DaisyChainReset (
    void                )
```

Resets the digitizer's daisy chain mechanism.

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.11 DaisyChainSetMode()

```
virtual int DaisyChainSetMode (
    unsigned int        mode )
```

Set the daisy chain mode.

Parameters

mode	The target mode • 0: Slave • 1: Master

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.12 DaisyChainSetOutputState()

```
virtual int DaisyChainSetOutputState (
    unsigned int        state )
```

Set the state of the daisy chain output.

This function is used to set the state of the daisy chain output for the purpose of identifying the device order. Forcing a logic high level will superseed any normal operation of the output port. Calling this function does *not* activate the output buffer. Refer to DaisyChainEnableOutput() for that functionality.

Parameters

state	• 0: Default behavior • 1: Force a logic high level

Returns

1 for success, 0 otherwise

Valid for

ADQ8

See also

DaisyChainEnableOutput

10.2.13 DaisyChainSetTriggerSource()

```
virtual int DaisyChainSetTriggerSource (
    unsigned int        trig_source )
```

Set the trigger source for the daisy chain master.

The default behavior of the daisy chain master is to use a level trigger applied to one of the data channels as the trigger source. This function is used to specify an alternate trigger source. A slave device always selects the daisy chain input as the trigger source.

Parameters

trig_source Refer to SetTriggerMode() for a source selection table.

Returns

1 for success, 0 otherwise

Valid for

ADQ8

See also

SetTriggerMode

10.2.14 DaisyChainSetupLevelTrigger()

```
virtual int DaisyChainSetupLevelTrigger (
    unsigned int        channel,
    int                 level,
    int                 arm_hysteresis,
    unsigned int        edge )
```

Set up the daisy chain level trigger.

The level trigger only needs to be configured on the *master* device.

Parameters

channel The target channel, indexed from 1 and upwards.

Parameters

level The trigger level specified in ADC codes. A trigger will be generated at the first sample crossing this threshold.
arm_hysteresis The trigger arm hysteresis protects against *false* trigger events, e.g. generated due to signal noise a the threshold crossing. The arm hysteresis is involved in computing the *trigger arm level* as ▪ arm level = trigger level + trigger arm hysteresis (for polarity 0) ▪ arm level = trigger level - trigger arm hysteresis (for polarity 1) In order for the level trigger to *be ready* to generate a trigger event, the signal has to visit ADC codes ▪ above the *trigger arm level* (for polarity 0) ▪ below the *trigger arm level* (for polarity 1)
edge ▪ 0: Falling edge ▪ 1: Rising edge

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.15 DaisyChainSetupOutput()

```
virtual int DaisyChainSetupOutput (
    unsigned int     sync_polarity,
    unsigned int     sync_immediate,
    unsigned int     sync_length )
```

Set up the shape of the daisy chain output pulse.

Parameters

sync_polarity The sync polarity determines the overall shape of the daisy chain output signal. Additionally, the setting also selects the edge of the input signal at which to detect an event. ▪ 0: Negative pulse (idle at logic high), falling edge detection ▪ 1: Positive pulse (idle at logic low), rising edge detection

Parameters

sync_immediate The immedate mode is only applicable for daisy chain slave devices. The mode accelerates the passthrough of the trigger signal by *not* resynchronizing the output pulse to the reference clock. The trigger event used to trigger the record remains unaffected and is still synchronized to the refence grid. ▪ 0: Disable ▪ 1: Enable	
sync_length The length of the output pulse, specified in samples.	

Returns

1 for success, 0 otherwise

Valid for

ADQ8

10.2.16 DisableInternalTriggerCounts()

```
virtual unsigned int DisableInternalTriggerCounts ( )
```

Disables the internal trigger counter.

This will let the internal trigger block to run freely and all the triggers generated by the internal trigger block will propagate forward as normal.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

SetInternalTriggerCounts()

10.2.17 EnableInternalTriggerCounts()

```
virtual unsigned int EnableInternalTriggerCounts ( )
```

Enables the internal trigger counter.

Allows the user to control the number of internal triggers that will be generated. A number of triggers must be specified with the function call SetInternalTriggerCounts() enabling alone will not pass through any triggers if you have not specified how many triggers that will be allowed to pass through.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

SetInternalTriggerCounts()

10.2.18 EnablePXIeTriggers()

```
virtual unsigned int EnablePXIeTriggers (
    unsigned int      port,
    unsigned int      bitflags )
```

Selects which PXIe triggers to trigger on.

All the various PXIe trigger inputs are summed together into a single PXIe trigger signal. This function allows specific inclusion/exclusion of these inputs.

Parameters

port The PXIe trigger block has two ports which can be configured independently, and which are connected to separate parts of the digitizer logic, according to: • 0: Data acquisition logic • 1: AWG	
bitflags Is a bit field where each bit enables a specific trigger input for the selected port • bit 0: DSTARA • bit 1: DSTARB • bit 2: PXI_TRIG[0] • bit 3: PXI_TRIG[1]	

Returns

1 for successful operation and 0 for failure

Note

PXI_TRIG[2 to7] are not routed on the digitizer PCB and cannot be used

Valid for

SDR14

See also

EnablePXIeTrigout(), PXIeSoftwareTrigger()

10.2.19 EnablePXIeTrigout()

```
virtual unsigned int EnablePXIeTrigout (
    unsigned int      port,
```

```
unsigned int    bitflags )
```

Selects which PXIe triggers outputs to use.

The trigger output of each port may be connected to any/all the available PXIe trigger outputs via this function.

Parameters

port Selects where from to take the trigger ▪ 0: Data acquisition logic ▪ 1: AWG ▪ 2: Software (PXIe Software trigger)	
bitflags Is a bit field where each bit enables output of the port trigout signal to a specific PXIe trigger output ▪ bit 0: DSTARC ▪ bit 1: PXI_TRIG[0] ▪ bit 2: PXI_TRIG[1]	

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

EnablePXIeTriggers(), PXIeSoftwareTrigger()

10.2.20 GetPPTStatus()

```
virtual unsigned int GetPPTStatus ( )
```

Gets the status register of the Precise Period Trigger function.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208

See also

SetPPTActive()

10.2.21 InitPPT()

```
virtual unsigned int InitPPT ( )
```

Initializes the Precise Period Trigger.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208

See also

SetPPTActive()

10.2.22 PXIeSoftwareTrigger()

```
virtual unsigned int PXIeSoftwareTrigger ( )
```

Sends a trigger signal on all enabled trigger outputs.

The trigger outputs enabled are specified using EnablePXIeTrigout()

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

EnablePXIeTriggers()

10.2.23 SetInternalTriggerCounts()

```
virtual unsigned int SetInternalTriggerCounts (
    unsigned int      trigger_counts )
```

Sets the number of triggers that will be issued by the internal trigger.

This can be used to create a finite burst of triggers.

Parameters

trigger_counts	The amount of triggers that will be issued

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

EnableInternalTriggerCounts()

10.2.24 SetInternalTriggerSyncMode()

```
virtual unsigned int SetInternalTriggerSyncMode (
    unsigned int      mode )
```

Sets up and activates the internal trigger synchronization.

Allows the user to synchronize several boards with a synchronization signal (can for instance be a PPS input). For slaves for instance, using trigger source 11 resets the internal trigger on an external trigger gated with the sync input provided by the master

Parameters

mode
• 0 : Do not use any sycnhronization scheme • 1 : Master: External trigger input as synchronization input, will provide gate on trigout • 2 : Slave: Selected trigger (selected with SetupTimeStampSync) is also used for resetting the internal trigger (ADQ7 only)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ108, SDR14, ADQ7, ADQ8

See also

SetupTimestampSync()

10.2.25 SetPPTActive()

```
virtual unsigned int SetPPTActive (
    unsigned int      active )
```

Activates or deactivates Precise Period Trigger.

The Precise Period Trigger (PPT) synchronizes the external trigger to precise period. Please refer to the PPT user guide and example for more detailed information on how to use the Precise Period Trigger.

Parameters

active
Set to 1 to activate and 0 to deactivate PPT

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208

See also

InitPPT()

10.2.26 SetPPTBurstMode()

```
virtual unsigned int SetPPTBurstMode (
    unsigned int      burst_mode )
```

Activates or deactivates the Precise Period Trigger burst mode.

In burst mode, the device will continue to trigger at each PPT period after the first external trigger event, without the need for more external triggers

Parameters

burst_mode	Set to 1 to activate and 0 to deactivate burst mode

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208

See also

SetPPTActive()

10.2.27 SetPPTInitOffset()

```
virtual unsigned int SetPPTInitOffset (
    unsigned int      init_offset )
```

Sets the Precise Period Trigger initial offset.

The offset is applied to the period on the first trig after initialization of the Precise Period Trigger

Parameters

init_offset	A number of samples from 32 to 2^{27}-1

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208

See also

SetPPTActive()

10.2.28 SetPPTPeriod()

```
virtual unsigned int SetPPTPeriod (
    unsigned int        period )
```

Sets the Precise Period Trigger period.

Parameters

period	The number of sample per period, from 32 to 2^{27}-1

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208

See also

SetPPTActive()

10.2.29 SetPXIeTrigDirection()

```
virtual unsigned int SetPXIeTrigDirection (
    unsigned int        trig0output,
    unsigned int        trig1output )
```

Sets the direction of the two PXI_TRIG I/O pins.

Parameters

trig0output

I/O selection for PXI_TRIG[0]

- 0: Input (default)
- 1: Output

Parameters

trig1output
I/O selection for PXI_TRIG[1] ▪ 0: Input (default) ▪ 1: Output

Returns

1 for successful operation and 0 for failure

Warning

Make sure that no other drivers are connected to the PXI_TRIG bus before setting the trigger pins to outputs, or you may risk damaging the digitizer.

Valid for

SDR14

See also

EnablePXIeTriggers(), EnablePXIeTrigout()

10.2.30 SetTriggerGate()

```
virtual unsigned int SetTriggerGate (
    unsigned int      enabled,
    unsigned int      mode,
    unsigned int      gate_mux )
```

Sets up and activates the trigger gating functionaliy.

Allows the user to apply trigger gating. i.e. block triggers until a condition is fulfilled. A closed trigger gate means triggers are blocked and not accepted.

Parameters

enabled If set to 1, trigger gating logic is activated (activates on arm)
mode ▪ 0 : Use level on gate control input to decide whether trigger gate is open or closed ▪ 1 : Arm closes the trigger gate initially, then first rising edge on gate_control_input open the trigger gate
gate_mux ▪ 0 : Use external trigger input as gate control input ▪ 1 : Use GPIO pin #5 as gate control input

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ108, SDR14

See also

SetConfigurationTrig()

10.2.31 SetTriggerMaskPXI()

```
virtual int SetTriggerMaskPXI (
    unsigned char     mask )
```

Sets the mask for accepting triggers from PXI.

Parameters

mask Select which PXI TRIG inputs to trigger from (inputs are ORed together if several bits are set) The bits and PXI TRIG signales are connected according to the list below: ▪ bit 0: PXI TRIG 0 ▪ bit 1: PXI TRIG 1 ▪ bit 2: PXIe STARA ▪ bit 3: PXIe STARB (ADQ7 / ADQ8 only)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7, ADQ8

10.2.32 WriteSTARBDelay()

```
virtual unsigned int WriteSTARBDelay (
    unsigned int      starbdelay,
    unsigned int      writetoeeprom )
```

Writes a delay value for the PXIe DSTARB trigger input.

The value may also be stored in the onboard EEPROM to be loaded upon each restart of the digitizer.

Parameters

starbdelay The delay value. The allowed range is 0 to 31, where each unit corresponds to a 78 ps delay.

Parameters

writetoeeprom	Specifies whether to write the value to the onboard EEPROM: • 0: Do not update the EEPROM value • 1: Update the EEPROM value so that the new value is loaded by default on power-up.

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

EnablePXIeTriggers()

11 Clock Setup

Functions

- int AdjustClockReferenceDelay (float delayadjust_ps)

  Adjust the reference clock delay relative to the factory calibrated delay.
- int GetClockSource ()

  Gets the current clock source.
- unsigned int GetExternalClockReferenceStatus (unsigned int *extrefstatus)

  Gets the status of the external clock reference.
- int GetPllFreqDivider ()

  Gets the current PLL-divider.
- int SetClockFrequencyMode (int clockmode)

  Sets the clock frequency mode.
- int SetClockInputImpedance (unsigned int input_num, unsigned int mode)

  Sets the clock connector input impedance.
- int SetClockSource (int source)

  Sets the clock source.
- int SetExternalReferenceFrequency (float ref_freq)

  Sets the external clock reference frequency.
- int SetPll (int n_divider, int r_divider, int vco_divider, int channel_divider)

  Performs setup of the clock PLL.
- int SetPllFreqDivider (int divider)

  Sets the clock PLL divider.
- int SetTargetSampleRate (int mode, double value)

  Sets the desired target sample rate.

11.1 Detailed Description

Functions used for selecting clock source and speed.

11.2 Function Documentation

11.2.1 AdjustClockReferenceDelay()

```
int AdjustClockReferenceDelay (
    float          delayadjust_ps )
```

Adjust the reference clock delay relative to the factory calibrated delay.

On boards which support programmable clock reference delay, this function can be used to adjust the clock reference delay in order to compensate for inter-digitizer skew caused by wiring or backplane routing. Make sure that the clock source is set to a mode which includes the delay circuitry in order for the function to have any effect.

Parameters

delayadjust_ps	An adjustment amount in units of picoseconds, relative to the factory calibrated delay

Returns

1 for success, 0 otherwise

Valid for

ADQ8, (ADQ7 produced post 2021-04-01)

See also

SetClockSource

11.2.2 GetClockSource()

```
virtual int GetClockSource ( )
```

Gets the current clock source.

Please see SetClockSource() for an explanation of the values.

Returns

The clock source

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14

See also

SetClockSource()

11.2.3 GetExternalClockReferenceStatus()

```
virtual unsigned int GetExternalClockReferenceStatus (
    unsigned int *    extrefstatus )
```

Gets the status of the external clock reference.

When using an external clock reference, this API returns the status of this reference.

Parameters

extrefstatus	Pointer to where to return the status. These values may be returned: ▪ 1: External reference available ▪ 0: External reference not detected

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ114, ADQ212, ADQ112

11.2.4 GetPllFreqDivider()

```
virtual int GetPllFreqDivider ( )
```

Gets the current PLL-divider.

Please see SetPllFreqDivider() for an explanation of the values.

Returns

The PLL divider

Valid for

ADQ212, ADQ112, ADQ114, ADQ214

See also

SetPllFreqDivider()

11.2.5 SetClockFrequencyMode()

```
virtual int SetClockFrequencyMode (
    int               clockmode )
```

Sets the clock frequency mode.

If internal clock and reference is used, this is handled automatically. If external clock or external reference is used, the board must be manually set to the correct mode by the user.

Parameters

clockmode Selects the clock frequency mode according to this list: • 0: Low frequency mode (external clock range 35-240MHz) • 1: High frequency mode (default, external clock range 240-550MHz)	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214

See also

SetClockSource()

11.2.6 SetClockInputImpedance()

```
virtual int SetClockInputImpedance (
    unsigned int        input_num,
    unsigned int        mode )
```

Sets the clock connector input impedance.

Parameters

input_num Set to 1 for standard clock / clock reference input connector	
mode Set to 0 for 50 ohm input, set to 1 for high impedance mode	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7

11.2.7 SetClockSource()

```
virtual int SetClockSource (
    int                 source )
```

Sets the clock source.

Parameters

source

Selects the source according to this list:

- 0: Internal clock source, internal 10 MHz reference
- 1: Internal clock source, external 10 MHz reference
- 2: External clock source
- 3: Internal clock source, external 10 MHz reference from PXIsync
- 4: Internal clock source, external TCLKA backplane reference (MTCA units only)
- 5: Internal clock source, external TCLKB backplane reference (MTCA units only)
- 6: Internal clock source, external 100 MHz reference from PXIe 100 MHz clock
- 7: Internal clock source, external 10 MHz reference plus jitter cleaning (ADQ7, ADQ8 only)
- 8: Internal clock source, external 10 MHz reference plus delay adjustment (ADQ8 only)
- 9: Internal clock source, external reference, free target sample rate (ADQ14/ADQ12 only)

Returns

1 for successful operation and 0 for failure

Note

When using external reference, make sure that an existing reference signal is connected to the correct clock reference input port. Othewise this command will fail. See data sheet for requirements on external clock reference.

For ADQ112, ADQ114, ADQ212 and ADQ214: When setting external clock source, do not follow with the command to set the pll frequency divider because it will reset the source to internal.

For ADQ412, ADQ108, ADQ208, ADQ1600, SDR14: Please note that selecting clock source also resets the timestamp counter.

Mode 9 with free target sample rate is still very limited as it has to comply with PLL configuration possibilities. Use SetTargetSampleRate for setting the target sample rate.

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetClockFrequencyMode(), SetTargetSampleRate()

11.2.8 SetExternalReferenceFrequency()

```
virtual int SetExternalReferenceFrequency (
    float            ref_freq )
```

Sets the external clock reference frequency.

By default, the digitizer is configured to use a 10 MHz external reference, when an external reference clock source is set up via SetClockSource. This function can be used to specify a reference frequency that differs from the default. The function must be called prior to SetClockSource in order to be taken into account during the clocking setup.

Parameters

ref_freq
Reference clock frequency in MHz ▪ ADQ12 / ADQ14: In the range 10 - 500 MHz (needs to fulfill PLL requirements), ▪ ADQ7: In the range of 1 - 500 MHz (needs to fulfill PLL requirements)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7

See also

SetClockSource(), SetTargetSampleRate()

11.2.9 SetPll()

```
virtual int SetPll (
    int              n_divider,
    int              r_divider,
    int              vco_divider,
    int              channel_divider )
```

Performs setup of the clock PLL.

Not all parameters can be changed on all ADQs. Please look under the specific card below to see how the sample frequency is set.

- ADQ214 & ADQ212:

  $F_s = \frac{F_{ref} \cdot n\ divider}{r\ divider \cdot vco\ divider \cdot channel\ divider}$
- ADQ114 & ADQ112:

  $F_s = \frac{2 \cdot F_{ref} \cdot n\ divider}{r\ divider \cdot vco\ divider \cdot channel\ divider}$
- ADQ108 & ADQ208:

  $F_s = \frac{4 \cdot F_{ref} \cdot n\ divider}{r\ divider}$
- ADQ1600:

  $F_s = \frac{4 \cdot F_{ref} \cdot n\ divider}{r\ divider}$
- SDR14:

  $F_s = \frac{2 \cdot F_{ref} \cdot n\ divider}{r\ divider}$
- ADQ412-1G:

  $F_s = \frac{F_{ref} \cdot n\ divider}{2 \cdot r\ divider}$
- ADQ412-3G and -4G:

  $F_s = \frac{F_{ref} \cdot n\ divider}{r\ divider}$

F_{ref} is either the internal 10 MHz reference, or an external reference.

Parameters

n_divider

- ADQ214 and ADQ114: Default is 220, 1 to 262175 are valid values
- ADQ212, and ADQ112: Default is 160, 1 to 262175 are valid values
- ADQ1600 and SDR14: Default is 160
- ADQ108: Default is 350
- ADQ208: Default is 400
- ADQ1600: Default is 160
- ADQ412-1G: Default is 400 (only multiples of 16 is allowed)
- ADQ412-3G: Default is 360 (only multiples of 8 is allowed)
- ADQ412-4G: Default is 400 (only multiples of 8 is allowed)
- SDR14: Default is 160

r_divider

- ADQ214, ADQ212, ADQ114, and ADQ112: Default is 1, 1 to 16383 are valid values
- ADQ1600, SDR14, ADQ108, ADQ208, ADQ1600 and ADQ412: Default is 2, 1 to 16383 are valid values

vco_divider

- ADQ214, ADQ212, ADQ114, and ADQ112: Default is 2, 2 to 6 are valid values
- Not used for other units

channel_divider

- ADQ214, ADQ212, ADQ114, and ADQ112: Default is 2, 1 to 32 are valid values
- Not used for other units

Returns

1 for successful operation and 0 for failure

Note

For ADQ212, ADQ112, ADQ114 and ADQ214: Frequencies lower than 100 MHz may sometimes fail to get the PLL locked as this renders a clock out of specification for the clocking circuitry in the FPGA. If you require lower sampling rates, please consider using the sample skip function.

For ADQ212, ADQ112, ADQ114 and ADQ214: This function will call SetClockFrequencyMode if the clock source is internal reference.

For ADQ108 and ADQ208: Frequencies lower than 6000 MHz may sometimes fail to get the PLL locked as this renders a clock out of specification for the clocking circuitry in the FPGA.

For ADQ412 and ADQ1600 VCO Frequencies lower than 1400 MHz may sometimes fail to get the PLL locked as this renders a clock out of specification for the clocking circuitry in the FPGA.

For SDR14 VCO Frequencies lower than 1550 MHz or higher than 1600MHz may fail due to design restrictions.

For ADQ412, ADQ108, ADQ208, ADQ1600, SDR14: Please note that selecting a new frequency also resets the timestamp counter.

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14

See also

SetClockSource()

11.2.10 SetPllFreqDivider()

```
virtual int SetPllFreqDivider (
    int                divider )
```

Sets the clock PLL divider.

After setting the divider value, the PLL is restarted. The software will also check that the PLL locks properly. The sampling rate is calculated differently for the different units:

- ADQ214: $F_s = \frac{F_{ref}*80}{divider}$
- ADQ114: $F_s = \frac{F_{ref}*160}{divider}$
- ADQ212: $F_s = \frac{F_{ref}*110}{divider}$
- ADQ112: $F_s = \frac{F_{ref}*220}{divider}$

Parameters

divider
Selects the divider. 2 to 20 are valid values (2 is default).

Returns

1 for successful operation and 0 for failure

Note

Dividers 18 to 20 may sometimes fail to get the PLL locked for ADQ114/ADQ214 devices as this renders a clock out of specification for the clocking circuitry in the FPGA. If you require lower sampling rates, please consider using the sample skip function. The function will then return failure.

Valid for

ADQ212, ADQ112, ADQ114, ADQ214

See also

SetClockSource()

11.2.11 SetTargetSampleRate()

```
int SetTargetSampleRate (
```

```
int         mode,
double      value )
```

Sets the desired target sample rate.

This function can be called prior to SetClockSource in order to specify a target sample rate that differs from the default. The allowed range of sample rates is limited by the clocking hardware of the unit.

Parameters

mode	Reserved, set to 0
value	Target sample rate in MHz / MSPS

Returns

1 for successful operation and 0 for failure

Valid for

(ADQ7 produced post 2021-04-01)

See also

SetClockSource(), SetExternalReferenceFrequency()

12 MultiRecord Mode

Functions

- int ArmTrigger ()

  Arms the trigger block.
- int DisarmTrigger ()

  Disarms the the ADQ device.
- int GetAcquired ()

  Checks if one or more record has been collected.
- unsigned int GetAcquiredAll ()

  Checks if all record have been collected.
- unsigned int GetAcquiredRecords ()

  Gets the number of records collected.
- int GetData (void **target_buffers, unsigned int target_buffer_size, unsigned char target_bytes_per_sample, unsigned int StartRecordNumber, unsigned int NumberOfRecords, unsigned char ChannelsMask, unsigned int StartSample, unsigned int nSamples, unsigned char TransferMode)

  Transfers data from ADQ to host in MultiRecord mode.
- int GetDataWH (void **target_buffers, void *target_headers, unsigned int target_buffer_size, unsigned char target_bytes_per_sample, unsigned int StartRecordNumber, unsigned int NumberOfRecords, unsigned char ChannelsMask, unsigned int StartSample, unsigned int nSamples, unsigned char TransferMode)

Transfers from ADQ to host in MultiRecord mode. Also saves headers.

- int GetDataWHTS (void **target_buffers, void *target_headers, void *target_timestamps, unsigned int target_buffer_size, unsigned char target_bytes_per_sample, unsigned int StartRecordNumber, unsigned int NumberOfRecords, unsigned char ChannelsMask, unsigned int StartSample, unsigned int nSamples, unsigned char TransferMode)

  Transfers from ADQ to host in MultiRecord mode. Also saves headers and timestamps.

- int GetTriggedCh ()

  Gets the channel that the level trigger trigged on.

- int GetWaitingForTrigger ()

  Polls the MultiRecord collection to see if it is waiting for a trigger.

- unsigned int MultiRecordClose ()

  Closes MultiRecord mode.

- unsigned int MultiRecordSetChannelMask (unsigned int channelmask)

  Enables/disables channels for a multirecord acquisition.

- unsigned int MultiRecordSetup (unsigned int NumberOfRecords, unsigned int SamplesPerRecord)
- int SetPreTrigSamples (unsigned int PreTrigSamples)

  Selects the number of pre-trigger samples for MultiRecord.

- int SetTriggerDelay (unsigned int triggerdelay_samples)

  Sets the delay between the trigger and the start of the record acquisition.

12.1 Detailed Description

MultiRecord is the standard collection mode for the ADQ digitizers. It uses the on-board memory to store multiple records of data. This section documents the functions used for setting up and transferring collections in with MultiRecord.

12.2 Function Documentation

12.2.1 ArmTrigger()

```
virtual int ArmTrigger ( )
```

Arms the trigger block.

This command must be sent before the ADQ device is allowed to be trigged. In MultiRecord mode, the ADQ will collect a record of data for each trigger that is acquired after this command is sent.

Returns

1 for successful operation and 0 for failure

Note

While the ADQ device is busy recording a record of data, the device will ignore triggers until the collection logic has finished.

To rearm the device, you must first call DisarmTrigger(), then ArmTrigger().

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

DisarmTrigger(), MultiRecordSetup(), SetTriggerMode()

12.2.2 DisarmTrigger()

```
virtual int DisarmTrigger ( )
```

Disarms the the ADQ device.

The ADQ device cannot be trigged when the trigger is disarmed. Disarming and then arming the trigger will cause any old data collected to be overwritten.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

ArmTrigger(), MultiRecordSetup(), SetTriggerMode()

12.2.3 GetAcquired()

```
virtual int GetAcquired ( )
```

Checks if one or more record has been collected.

Returns

1 if the ADQ has been trigged and at least one record has been acquired

Note

To see if all records have been acquired, use GetAcquiredAll(). To get the number of records that have been acquired, use GetAcquiredRecords()

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14

See also

GetAcquiredAll(), GetAcquiredRecords()

12.2.4 GetAcquiredAll()

```
virtual unsigned int GetAcquiredAll ( )
```

Checks if all record have been collected.

Returns

1 if all records have been collected and 0 otherwise

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetAcquired(), GetAcquiredRecords()

12.2.5 GetAcquiredRecords()

```
virtual unsigned int GetAcquiredRecords ( )
```

Gets the number of records collected.

Returns

The number of records collected

Note

For some units, the number of records collected is not updated until all records are completed.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

12.2.6 GetData()

```
virtual int GetData (
    void **          target_buffers,
    unsigned int     target_buffer_size,
    unsigned char    target_bytes_per_sample,
    unsigned int     StartRecordNumber,
    unsigned int     NumberOfRecords,
    unsigned char    ChannelsMask,
    unsigned int     StartSample,
    unsigned int     nSamples,
    unsigned char    TransferMode )
```

Transfers data from ADQ to host in MultiRecord mode.

Transfers data from the internal memory buffers in the ADQ device directly to user-assigned buffers. This function is meant to be used together with the function MultiRecordSetup().

Parameters

target_buffer_size The size of each buffer for the data that you want to transfer from each channel. This parameter should always be set to **NumberOfRecords*nSamples**.

Parameters

target_buffers Pointer to the buffers where the data should be stored, one buffer buffer for each channel of data. This parameter may therefore be an array of pointers, depending on how many channels the ADQ device has.
target_bytes_per_sample The size of each element in bytes
StartRecordNumber The record number to start collecting data from. This value can be set between 0 and up to **NumberOfRecords - 1**. For example, if you have set up the ADQ device to collect 20 records but you are only interested in collecting the last 5, **StartRecordNumber** should be set to 14 (record index starts from 0).
NumberOfRecords The number of records to collect from the unit. Please note that the sum **NumberOfRecords + StartRecordNumber** must always be smaller than the amount of records that were collected by the ADQ.
ChannelsMask A bit-mask providing a set bit for each channel to be fetched. For a unit with two channels, this parameter can be set to 0x1 to fetch data from channel A, 0x2 to fetch data from channel B, and 0x3 to fetch data from both channels. Note: For ADQ12 / ADQ14, the same channel mask should also be set using the MultiRecordSetChannelMask command
StartSample The starting sample of each record to fetch. This can be used to transfer only a part of a record. The first sample is indexed 0.
nSamples The number of samples to collect from each record. The starting point is set by **StartSample**
TransferMode The transfer mode. Please set to 0x00 for normal data fetch operations

Returns

1 for successful operation and 0 for failure

Note

The buffers pointed to by **target_buffers** must be allocated correctly by the user.

This is the recommended function for fast record data transfers.

Partial record read-out using StartSample and nSamples is not supported for ADQ14, ADQ12, ADQ7 or ADQ8. A full record - in same sequence and with no gaps as acquired - must be read on each call.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

MultiRecordSetup(), MultiRecordSetChannelMask()

12.2.7 GetDataWH()

```
virtual int GetDataWH (
    void **           target_buffers,
    void *            target_headers,
    unsigned int      target_buffer_size,
    unsigned char     target_bytes_per_sample,
    unsigned int      StartRecordNumber,
    unsigned int      NumberOfRecords,
    unsigned char     ChannelsMask,
    unsigned int      StartSample,
    unsigned int      nSamples,
    unsigned char     TransferMode )
```

Transfers from ADQ to host in MultiRecord mode. Also saves headers.

Collects data from the device with headers. See documentation for GetData(). The difference is only one added argument, the target destination for header data.

Parameters

target_buffers	See GetData()
target_headers	The memory location where headers will be written. The total amout of data that will be written is the header sizes times the number of records to fetch specified in NumberOfRecords. If set to NULL no headers will be fetched. The header size is: • 40 bytes for ADQ8, ADQ7, ADQ12 and ADQ14. • 32 bytes for all other products.
target_buffer_size	See GetData()
target_bytes_per_sample	See GetData()
StartRecordNumber	See GetData()
NumberOfRecords	See GetData()
ChannelsMask	See GetData()
StartSample	See GetData()
nSamples	See GetData()
TransferMode	See GetData()

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

MultiRecordSetup(), GetData()

12.2.8 GetDataWHTS()

```
virtual int GetDataWHTS (
    void **          target_buffers,
    void *           target_headers,
    void *           target_timestamps,
    unsigned int     target_buffer_size,
    unsigned char    target_bytes_per_sample,
    unsigned int     StartRecordNumber,
    unsigned int     NumberOfRecords,
    unsigned char    ChannelsMask,
    unsigned int     StartSample,
    unsigned int     nSamples,
    unsigned char    TransferMode )
```

Transfers from ADQ to host in MultiRecord mode. Also saves headers and timestamps.

Collects data from the device with headers and timestamps. See documentation for GetData(). The difference is only two added arguments, the target destination for header data and timestamp data.

Parameters

target_buffers See GetData()
target_headers The memory location where headers will be written. The total amout of data that will be written is the header sizes times the number of records to fetch specified in NumberOfRecords. If set to NULL no headers will be fetched. The header size is: ▪ 40 bytes for ADQ8, ADQ7, ADQ12 and ADQ14. ▪ 32 bytes for all other products.
target_timestamps The memory location where timestamps will be written. The total amount of data that will be written is 8 bytes (one int64) times the number of records to fetch specified in NumberOfRecords. If set to NULL no timestamps will be fetched.
target_buffer_size See GetData()
target_bytes_per_sample See GetData()

Parameters

StartRecordNumber See GetData()
NumberOfRecords See GetData()
ChannelsMask See GetData()
StartSample See GetData()
nSamples See GetData()
TransferMode See GetData()

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

MultiRecordSetup(), GetData()

12.2.9 GetTriggedCh()

```
virtual int GetTriggedCh ( )
```

Gets the channel that the level trigger trigged on.

- Return value = 0: None (may happen if the device was trigged in software trigger mode)
- Return value = 1: Channel A
- Return value = 2: Channel B
- Return value = 4: Channel C
- Return value = 8: Channel D

The trigged channel value is updated each time a new record is collected.

Returns

The channel that which the device was trigged by

Valid for

ADQ208, ADQ212, ADQ214, ADQ412, SDR14, ADQ8

12.2.10 GetWaitingForTrigger()

```
virtual int GetWaitingForTrigger ( )
```

Polls the MultiRecord collection to see if it is waiting for a trigger.

Returns

1 if the device is waiting for a trigger, and 0 otherwise

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14

See also

MultiRecordSetup()

12.2.11 MultiRecordClose()

```
virtual unsigned int MultiRecordClose ( )
```

Closes MultiRecord mode.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

MultiRecordSetup()

12.2.12 MultiRecordSetChannelMask()

```
virtual unsigned int MultiRecordSetChannelMask (
    unsigned int      channelmask )
```

Enables/disables channels for a multirecord acquisition.

Selects which channels should be active for a multirecord acquisition. By disabling unused channels, the remaining channels will have more memory area available for data storage, allowing larger maximum records. This command should be run prior to MultiRecordSetup

Parameters

channelmask A bit field, where bit 0 corresponds to channel A, bit 1 to channel B, etc. A bit value of 1 means the channel will be enabled.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

MultiRecordSetup()

12.2.13 MultiRecordSetup()

```
virtual unsigned int MultiRecordSetup (
    unsigned int      NumberOfRecords,
    unsigned int      SamplesPerRecord )
```

The MultiRecord mode may be used to trigger multiple records in successive order. The data is stored in the on-board DRAM and may then be read using GetData().

Parameters

NumberOfRecords	The number of records to collect
SamplesPerRecord	The number of samples per record

Returns

1 for successful operation and 0 for failure

Note

The same values are set for all channels

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

ArmTrigger(), MultiRecordSetupGP(), MultiRecordClose(), GetMaxNofSamplesFromNofRecords(), GetMaxNofRecordsFro

12.2.14 SetPreTrigSamples()

```
virtual int SetPreTrigSamples (
    unsigned int      PreTrigSamples )
```

Selects the number of pre-trigger samples for MultiRecord.

The granularity of the pre-trigger buffer depends on the product type:

- ADQ412: 8 samples in non-interleaved mode and 16 samples in interleaved mode
- ADQ1600: 8 samples
- ADQ108: 32 samples
- ADQ208: 16 samples in non-interleaved mode and 32 samples in interleaved mode

- SDR14: 4 samples
- ADQ214 & ADQ212: 2 samples
- ADQ114 & ADQ112: 4 samples
- ADQ12-4C/2C: 4 samples (value must match granularity)
- ADQ14-2X/1X: 8 samples (value must match granularity)
- ADQ14-4C/2C: 4 samples (value must match granularity)
- ADQ14-4A/2A: 2 samples (value must match granularity)
- ADQ7: 32 samples in 1ch@10GSPS mode and 16 samples in 2ch@5GSPS mode
- ADQ8-8A/8C: 4 samples Any pre-trigger size will be rounded UP by the granularity (except for ADQ12 and ADQ14).

For example on ADQ412 in non-interleaved mode; if you set pretrigsamples to 5, it will automatically be rounded up to 8 samples and if you instead set it to 9, it will be rounded up to 16 samples.

Parameters

PreTrigSamples	The number of pre-trigger samples. Must be less than the MultiRecord record size.

Returns

1 for successful operation and 0 for failure

Note

When using this function, the trigger hold-off is automatically reset to zero.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTriggerDelay(), MultiRecordSetup()

12.2.15 SetTriggerDelay()

```
virtual int SetTriggerDelay (
    unsigned int      triggerdelay_samples )
```

Sets the delay between the trigger and the start of the record acquisition.

The granularity of the delay setting depends on the product type:

- ADQ412: 8 samples in non-interleaved mode and 16 samples in interleaved mode
- ADQ1600: 8 samples
- ADQ108: 32 samples
- ADQ208: 16 samples in non-interleaved mode and 32 samples in interleaved mode

- SDR14: 4 samples
- ADQ214 & ADQ212: 2 samples
- ADQ114 & ADQ112: 4 samples
- ADQ12-4C/2C: 4 samples
- ADQ14-2X/1X: 8 samples
- ADQ14-4C/2C: 4 samples
- ADQ14-4A/2A: 2 samples
- ADQ7: 32 samples in 1ch@10GSPS mode and 16 samples in 2ch@5GSPS mode
- ADQ8-8A: 4 samples

Any delay entered will be rounded up to the closest multiple of the granularity.

Parameters

triggerdelay_samples The trigger delay in units of samples.

Returns

1 for successful operation and 0 for failure

Note

When using this function, the pre-trigger setting is automatically reset to zero.

This function supersedes the deprecated SetTriggerHoldOffSamples

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, SDR14, ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetPreTrigSamples(), SetTriggerHoldOffSamples(), MultiRecordSetup()

13 Advanced MultiRecord

Functions

- unsigned int GetAcquiredRecordsAndLoopCounter (unsigned int ∗acquired_records, unsigned int ∗loop_counter)

  Gets the number of records collected and one which loop the MultiRecord engine is acquiring.
- unsigned int GetMaxNofRecordsFromNofSamples (unsigned int NofSamples, unsigned int ∗MaxNofRecords)

  Gets the maximum number of records given a number of samples per record.
- unsigned int GetMaxNofSamplesFromNofRecords (unsigned int NofRecords, unsigned int ∗MaxNofSamples)

  Gets the maximum number of samples per record given a number of records.
- unsigned int GetNofRecords ()

  Gets the number of MultiRecord records set in the ADQ device.
- int GetOverflow ()

Checks if an overflow has occurred.

- unsigned int GetRecordSize ()

  Gets the current MultiRecord record size.

- int GetTriggerInformation ()

  Gets the enhanced trigger accuracy information.

- int GetTrigPoint ()

  Gets the position in the data where the trig occurred.

- unsigned long long GetTrigTime ()

  Gets the timestamp counter value.

- unsigned long long GetTrigTimeCycles ()

  Gets the cycle counter value of the time stamp.

- unsigned int GetTrigTimeStart ()

  Gets the start pulse value of the time stamp.

- unsigned int GetTrigTimeSyncs ()

  Gets the sync counter value of the time stamp.

- unsigned int MultiRecordSetupGP (unsigned int NumberOfRecords, unsigned int SamplesPerRecord, unsigned int *mrinfo)

  Performs an advanced setup of MultiRecord.

- int ResetTrigTimer (int TrigTimeRestart)

  Resets the trig timer.

- int SetCacheSize (unsigned int newSizeInBytes)

  Sets the cache size of transfer of data for MultiRecord mode.

- int SetTrigTimeMode (int TrigTimeMode)

  Selects the trig timer mode.

13.1 Detailed Description

These functions are implements advanced use cases for MultiRecord mode.

13.2 Function Documentation

13.2.1 GetAcquiredRecordsAndLoopCounter()

```
virtual unsigned int GetAcquiredRecordsAndLoopCounter (
    unsigned int *    acquired_records,
    unsigned int *    loop_counter )
```

Gets the number of records collected and one which loop the MultiRecord engine is acquiring.

This is only intended to be used for the acquisition mode called Continuous MultiRecord.

Parameters

acquired_records	Pointer to where the number of acquired records should be returned
loop_counter	Pointer to where the loop iteration of the MultiRecord engine should be returned

Returns

The number of records collected

Valid for

ADQ412, ADQ108, ADQ208, ADQ1600, SDR14, ADQ12, ADQ14

See also

MultiRecordSetupGP()

13.2.2 GetMaxNofRecordsFromNofSamples()

```
virtual unsigned int GetMaxNofRecordsFromNofSamples (
    unsigned int      NofSamples,
    unsigned int *    MaxNofRecords )
```

Gets the maximum number of records given a number of samples per record.

Parameters

NofSamples	The number of sample for each record
MaxNofRecords	A pointer to where the resulting maximum number of MultiRecord records will be stored

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14

See also

MultiRecordSetup(), GetMaxNofSamplesFromNofRecords()

13.2.3 GetMaxNofSamplesFromNofRecords()

```
virtual unsigned int GetMaxNofSamplesFromNofRecords (
    unsigned int      NofRecords,
    unsigned int *    MaxNofSamples )
```

Gets the maximum number of samples per record given a number of records.

Parameters

NofRecords The number of records
MaxNofSamples A pointer to where the resulting maximum number of samples per MultiRecord record

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14

See also

MultiRecordSetup(), GetMaxNofRecordsFromNofSamples()

13.2.4 GetNofRecords()

```
virtual unsigned int GetNofRecords ( )
```

Gets the number of MultiRecord records set in the ADQ device.

Returns

The number of records set

Valid for

ADQ214, ADQ114, ADQ212, ADQ112, ADQ12, ADQ14, ADQ8

13.2.5 GetOverflow()

```
virtual int GetOverflow ( )
```

Checks if an overflow has occurred.

Returns

1 if an overflow has occurred in the most recent collected record. 0 if not

Valid for

ADQ214, ADQ114, ADQ212, ADQ112

13.2.6 GetRecordSize()

```
virtual unsigned int GetRecordSize ( )
```

Gets the current MultiRecord record size.

The value is given per channel.

Returns

The record size in number of samples

Valid for

ADQ214, ADQ114, ADQ212, ADQ112

13.2.7 GetTriggerInformation()

```
virtual int GetTriggerInformation ( )
```

Gets the enhanced trigger accuracy information.

The bits in the returned value holds the information and is decoded as:

- `output[0:9]`: Reserved for future use
- `output[10:11]`: Enhanced trigger accuracy vector
- `output[12:31]`: Reserved for future use

Where `output` is the returned value.

Returns

The enhanced trigger information

Note

This information is only valid if the ADQ device is set to External trigger mode.

Valid for

ADQ214, ADQ114, ADQ212, ADQ112

13.2.8 GetTrigPoint()

```
virtual int GetTrigPoint ( )
```

Gets the position in the data where the trig occurred.

The point returned is the trigger point in the latest MultiRecord record transferred.

Returns

The position in the data array

Valid for

ADQ214, ADQ114, ADQ212, ADQ112

13.2.9 GetTrigTime()

```
virtual unsigned long long GetTrigTime ( )
```

Gets the timestamp counter value.

The result depends on the Trig Time Mode.

- SYNC_ON=cycles$*2^2$+start_val+trig_val

- SYNC_OFF=syncs$*2^{42}$+cycles$*2^{2}$+start_val+trig_val

Returns

The cycle counter value of the time stamp

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ412, ADQ1600, SDR14

See also

SetTrigTimeMode()

13.2.10 GetTrigTimeCycles()

```
virtual unsigned long long GetTrigTimeCycles ( )
```

Gets the cycle counter value of the time stamp.

Returns

The cycle counter value of the time stamp

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ412, ADQ1600, SDR14

See also

SetTrigTimeMode(), GetTrigTimeSyncs()

13.2.11 GetTrigTimeStart()

```
virtual unsigned int GetTrigTimeStart ( )
```

Gets the start pulse value of the time stamp.

Returns

A two bit value of the start pulse

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ412, ADQ1600, SDR14

See also

SetTrigTimeMode()

13.2.12 GetTrigTimeSyncs()

```
virtual unsigned int GetTrigTimeSyncs ( )
```

Gets the sync counter value of the time stamp.

Returns

The sync counter value of the time stamp

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ412, ADQ1600, SDR14

See also

SetTrigTimeMode(), GetTrigTimeCycles()

13.2.13 MultiRecordSetupGP()

```
virtual unsigned int MultiRecordSetupGP (
    unsigned int      NumberOfRecords,
    unsigned int      SamplesPerRecord,
    unsigned int *    mrinfo )
```

Performs an advanced setup of MultiRecord.

Performs setup of MultiRecord in a similar way as MultiRecordSetup, but as an extra parameter for advanced setup.

Parameters

NumberOfRecords	The number of records to collect (On ADQ12 and ADQ14 NumberOfRecords == 0 indicates infinite number of records in continuous mode)
SamplesPerRecord	The number of samples per record
mrinfo	Both an input and output. When calling this function, the contents at mrinfo used as below: ▪ mrinfo[0]: (mrinfo[0] can only be used on ADQV6 devices.) ▪ Set to 2 to enable MultiRecordModeAuto ▪ Set to 4 to enable MultiRecordModeDisablePretrigger. May be used to disable pretrigger buffering. (Note: disable pretrigger buffering cannot be used in conjunction with sample-skip or other data reduction methods) This will improve data readout performance while reading a record before all are completed. ▪ Set to 7 to enable both MultiRecordModeAuto and MultiRecordModeDisablePretrigger ▪ Set to 0 to disable both MultiRecordModeAuto and MultiRecordModeDisablePretrigger ▪ mrinfo[1] to mrinfo[9]: Reserved

When this function returns, it outputs data in these fields, according to:

- mrinfo[0]: dram_start_addr
- mrinfo[1]: dram_end_addr
- mrinfo[2]: dram_addr_per_record
- mrinfo[3]: dram_bytes_per_addr
- mrinfo[4]: setup_records
- mrinfo[5]: setup_samples

- mrinfo[6]: setup_padded_samples
- mrinfo[7]: max_number_of_records
- mrinfo[8]: shadow_size
- mrinfo[9]: reserved

These values are useful if the data is dumped using MemoryDump() for later parsing.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14

See also

MultiRecordSetup(), MemoryDump()

13.2.14 ResetTrigTimer()

```
virtual int ResetTrigTimer (
    int                 TrigTimeRestart )
```

Resets the trig timer.

Parameters

TrigTimeRestart
Restart mode. These values are valid: • 0: Timer waits for start pulse to start • 1: Timer restarts immediately

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ412, ADQ1600, SDR14

See also

SetTrigTimeMode()

13.2.15 SetCacheSize()

```
virtual int SetCacheSize (
    unsigned int        newSizeInBytes )
```

Sets the cache size of transfer of data for MultiRecord mode.

Can be used to optimize transfer performance for a specific application.

Parameters

newSizeInBytes The new cache size. Must be given in multiples of 1024 bytes.

Returns

1 for successful operation and 0 for failure

Note

When transferring small records one at the time, use a small value.

This function should rarely be used, the default setting is the best for most applications.

This function allocates memory in the kernel space.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP

See also

SetTransferBuffers()

13.2.16 SetTrigTimeMode()

```
virtual int SetTrigTimeMode (
    int                TrigTimeMode )
```

Selects the trig timer mode.

Parameters

TrigTimeMode Trig timer mode. These values are valid: • 0: Continuous count • 1: Activate sync mode, count sync pulses and reset counter

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ412, ADQ1600, SDR14

See also

ResetTrigTimer()

14 Data Streaming

Data Structures

- struct ADQRecordHeader

Header structure returned when streaming data. More...

Macros

- #define ADQ14_STREAM_CHUNK_BYTES (1024)

  Chunk size used when streaming, in bytes.
- #define ADQ14_STREAM_RECORD_MIN_BYTES (128)

  Minimum record size in raw data stream.
- #define ADQ7_STREAM_CHUNK_BYTES (1024)

  Chunk size used when streaming, in bytes.
- #define ADQ7_STREAM_RECORD_MIN_BYTES (128)

  Minimum record size in raw data stream.

Functions

- int CollectDataNextPage ()

  Transfers streamed data from the DMA buffers.
- unsigned int ContinuousStreamingSetup (unsigned char ChannelsMask)

  Sets up a continuous streaming data acquisition.
- unsigned int FlushDMA (void)

  Flushes DMA buffer.
- int FlushPacketOnRecordStop (unsigned int enable)

  Enables or disables the forced flush of a packet on record stop.
- int GetDataStreaming (void **target_buffers, void **target_headers, unsigned char channels_mask, unsigned int *samples_added, unsigned int *headers_added, unsigned int *header_status)

  Transfers data and record headers from the ADQ in streaming mode.
- int GetGPVectorMode (unsigned int channel, unsigned int *mode)

  Retrieves the configuraton of the general purpose vector fields in the triggered streaming user header.
- void * GetPtrStream ()

  Gets a pointer to the array of streamed data.
- unsigned int GetSamplesPerPage ()

  Gets the number of samples per transfer buffer for the ADQ.
- int GetStreamConfig (unsigned int option, unsigned int *value)

  Gets the streaming configuration.
- int GetStreamOverflow ()

  Returns whether there has been an overflow of the internal FPGA data buffers.
- int GetStreamStatus ()

  Gets the streaming status.
- unsigned int GetTransferBufferStatus (unsigned int *filled)

  Gets the number of DMA buffers available.
- int InitializeStreaming ()

Initialize and set up a streaming data acquisition.

- int SetChannelLevelTriggerMask (unsigned int channel, unsigned int level_trig_mask)

  Sets the level trigger mask (which level triggers are used by a channel) for a specific channel.

- int SetChannelNumberOfRecords (unsigned int channel, unsigned int nofrecords, int infinite_records)

  Sets the number of records for a specific channel.

- int SetChannelPretrigger (unsigned int channel, unsigned int pretrigger)

  Sets the amount of pre-trigger for a specific channel.

- int SetChannelRecordLength (unsigned int channel, unsigned int length, int infinite_length)

  Sets the record length for a specific channel.

- int SetChannelSampleSkip (unsigned int channel, unsigned int skipfactor)

  Sets the sample skip factor for a specific channel.

- int SetChannelTriggerDelay (unsigned int channel, unsigned int triggerdelay)

  Sets the amount of trigger delay for a specific channel.

- int SetChannelTriggerMode (unsigned int channel, int trig_mode)

  Sets the trigger mode for a specific channel.

- unsigned int SetFlushDMASize (unsigned int flush_size)

  Set DMA flush size.

- int SetGPVectorMode (unsigned int channel, unsigned int mode)

  Configures the behavior of the general purpose vector fields in the triggered streaming user header.

- int SetStreamConfig (unsigned int option, unsigned int value)

  Sets the streaming configuration.

- int SetStreamingChannelMask (unsigned int channelmask)

  Select which channels will transmit data to the host system during streaming.

- int SetStreamStatus (int status)

  Controls streaming mode.

- int SetTransferBuffers (unsigned int nOfBuffers, unsigned int bufferSize)

  Sets the number and size of the data transfer buffers.

- int StartStreaming ()

  Sets up the software for streaming from the ADQ.

- int StopStreaming ()

  Stops streaming in the software.

- unsigned int TriggeredStreamingSetupV5 (unsigned int SamplePerRecord, unsigned int NofPreTrigSamples, unsigned int NofTriggerDelaySamples, unsigned int TriggeredStreamingFlags)

  Sets the parameters for multi channels Triggered Streaming on V5 ADQs.

- unsigned int TriggeredStreamingShutdownV5 ()

  Issues shutdown for Triggered Streaming on V5 product.

- unsigned int WaitForTransferBuffer (unsigned int *filled, unsigned int timeout_setting)

  Wait for DMA buffers to be available (not working for USB2 interface units)

14.1 Detailed Description

These functions are used for setting up streaming mode and transferring data.

14.2 Data Structure Documentation

14.2.1 struct ADQRecordHeader

Header structure returned when streaming data.

For each record returned when streaming data a corresponding header structure is also returned.

Data Fields

uint8_t	Channel	Channel.
uint8_t	DataFormat	Data format.
uint16_t	GeneralPurpose0	General purpose 0.
uint16_t	GeneralPurpose1	General purpose 1.
uint32_t	RecordLength	Record length [samples].
uint32_t	RecordNumber	Record number.
int64_t	RecordStart	Record start.
uint8_t	RecordStatus	Status of record.
int32_t	SamplePeriod	Sample period [ps].
uint32_t	SerialNumber	Device serial.
uint64_t	Timestamp	Record timestamp.
uint8_t	UserID	ID set by user.

14.3 Macro Definition Documentation

14.3.1 ADQ14_STREAM_CHUNK_BYTES

```
#define ADQ14_STREAM_CHUNK_BYTES (1024)
```

Chunk size used when streaming, in bytes.

If a buffer of raw data parsed by GetDataStreaming() or ADQData_ParsePacketStream() the buffer size needs to be a multiple of this value.

14.3.2 ADQ14_STREAM_RECORD_MIN_BYTES

```
#define ADQ14_STREAM_RECORD_MIN_BYTES (128)
```

Minimum record size in raw data stream.

Minimum raw record size before parsing data. If a buffer of raw data of size N bytes is parsed by GetDataStreaming() or ADQData_ParsePacketStream() it can generate at most N/ADQ14_STREAM_RECORD_MIN_BYTES number of records.

14.3.3 ADQ7_STREAM_CHUNK_BYTES

```
#define ADQ7_STREAM_CHUNK_BYTES (1024)
```

Chunk size used when streaming, in bytes.

If a buffer of raw data parsed by GetDataStreaming() or ADQData_ParsePacketStream() the buffer size needs to be a multiple of this value.

14.3.4 ADQ7_STREAM_RECORD_MIN_BYTES

```
#define ADQ7_STREAM_RECORD_MIN_BYTES (128)
```

Minimum record size in raw data stream.

Minimum raw record size before parsing data. If a buffer of raw data of size N bytes is parsed by GetDataStreaming() or ADQData_ParsePacketStream() it can generate at most N/ADQ7_STREAM_RECORD_MIN_BYTES number of records.

14.4 Function Documentation

14.4.1 CollectDataNextPage()

```
virtual int CollectDataNextPage ( )
```

Transfers streamed data from the DMA buffers.

The data is stored at the location pointed to by GetPtrStream(). The number of samples retrieved is given by GetSamplesPerPage().

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetPtrStream(), GetSamplesPerPage()

14.4.2 ContinuousStreamingSetup()

```
virtual unsigned int ContinuousStreamingSetup (
    unsigned char    ChannelsMask )
```

Sets up a continuous streaming data acquisition.

At the first incoming trigger event, the digitizer will start streaming continuous data from the ADCs. Some form of data reduction will likely be required in order for the data transfer to not bottleneck the acquisition and cause data overflow. Some examples of data reduction are the sample skip feature, and using the channel mask argument to disable channels.

Parameters

ChannelsMask
A bit field specifying which channels to enable streaming for. Bit 0 enables channel A, bit 1 channel B and so forth.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7

14.4.3 FlushDMA()

```
virtual unsigned int FlushDMA (
    void            )
```

Flushes DMA buffer.

Flushes the current DMA buffer so data can be read out if the buffer is only partly filled.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7

14.4.4 FlushPacketOnRecordStop()

```
virtual int FlushPacketOnRecordStop (
    unsigned int      enable )
```

Enables or disables the forced flush of a packet on record stop.

This function may be used to control whether the capture of the last data in a record also forces a flush of the packet. This is particularly useful when configuring low latency data transfers. This is an advanced feature and should only be used in the approved contexts, i.e. another document should point you to this function.

Parameters

enable	

Returns

1 for successful operation and 0 for failure

Note

This function must be called after TriggeredStreamingSetup()

Valid for

ADQ12, ADQ14, ADQ7

14.4.5 GetDataStreaming()

```
virtual int GetDataStreaming (
    void **            target_buffers,
    void **            target_headers,
    unsigned char      channels_mask,
    unsigned int *     samples_added,
    unsigned int *     headers_added,
    unsigned int *     header_status )
```

Transfers data and record headers from the ADQ in streaming mode.

This function is not recommended for new designs, refer to document 20-2465 for information about the updated flow.

This function fetches and parses the data from the ADQ when the device is configured in streaming mode. The data and metadata is delivered to the user buffers: target_buffers and target_headers, respectively. The channels_mask is used to suppress data collection from specific channels. The parameter samples_added contains information on how many samples that were added to the user data buffers. This allows the user to increment the data buffer pointers accordingly before the next call to the GetDataStreaming() function. The parameters headers_added and header_status provide the user with information to determine if the header buffer pointers should be incremented. The parameter headers_added contains the number of initialized headers as a result of this call to GetDataStreaming(). The header_status parameter is used to denote if the last header is complete or not. E.g. a return value of headers_added = 4 and header_status = 1 signifies 4 complete headers, while headers_added = 4 and header_status = 0 would signify 3 complete headers and one incomplete header. In this case, it is important that the header buffers are incremented such that the buffer for that channel points at the incomplete header in the next call to GetDataStreaming().

Parameters

Parameter	Description
target_buffers	Pointer to the user-allocated buffers where the data is to be placed. One buffer for each channel of data.
target_headers	Pointer to the user-allocated buffers where the headers are to be placed. One buffer for each channel of data.
channels_mask	Channels mask to enable/disable data parsing from specific channels. Expecting a 4-bit bitmask.
samples_added	Contains the number of samples added to the target buffers (per channel) when the function returns successfully.
headers_added	Contains the number of initialized headers from this function call.

Parameters

header_status Contains the last header status. See description above.

Returns

1 for success, 0 otherwise

Valid for

ADQ12, ADQ14, ADQ7

14.4.6 GetGPVectorMode()

```
virtual int GetGPVectorMode (
    unsigned int      channel,
    unsigned int *    mode )
```

Retrieves the configuraton of the general purpose vector fields in the triggered streaming user header.

Parameters

channel The target channel, indexed from 1 and upwards.
mode Pointer to memory where the value is placed.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7

See also

SetGPVectorMode()

14.4.7 GetPtrStream()

```
virtual void* GetPtrStream ( )
```

Gets a pointer to the array of streamed data.

The size of the data array may be acquired using GetSamplesPerPage() after calling SetStreamStatus()

Returns

A pointer to the data array of the stream.

Valid for

ADQ212, ADQ214, ADQ112, ADQ114, ADQ412, ADQ108, ADQ208, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetStreamStatus(), GetSamplesPerPage(), CollectDataNextPage()

14.4.8 GetSamplesPerPage()

```
virtual unsigned int GetSamplesPerPage ( )
```

Gets the number of samples per transfer buffer for the ADQ.

Used with CollectDataNextPage/CollectRecord to get information on number of samples per call.

Returns

The number of samples per page

Note

per channel if applicable. This figure may change when altering the acquisition settings.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTransferBuffers()

14.4.9 GetStreamConfig()

```
virtual int GetStreamConfig (
    unsigned int      option,
    unsigned int *    value )
```

Gets the streaming configuration.

Please see SetStreamConfig() for information on the meaning of the values.

Parameters

option	Selected option
value	Retuned configuration value

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ12, ADQ14

14.4.10 GetStreamOverflow()

```
virtual int GetStreamOverflow ( )
```

Returns whether there has been an overflow of the internal FPGA data buffers.

When overflow occurs, data will be missing from the stream.

Returns

1 for overflow condition detected, 0 for no overflow condition detected and negative numbers for any errors.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetStreamStatus(), SetStreamStatus(), StartStreaming(), StopStreaming(), GetStatus()

14.4.11 GetStreamStatus()

```
virtual int GetStreamStatus ( )
```

Gets the streaming status.

Please see SetStreamStatus() for information on the meaning of the values.

Returns

The current stream status of the device

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ412, ADQ108, ADQ208, ADQ1600, SDR14

14.4.12 GetTransferBufferStatus()

```
virtual unsigned int GetTransferBufferStatus (
    unsigned int *    filled )
```

Gets the number of DMA buffers available.

This function enables the host application to balance the streaming read-out to avoid overflows. If the answer is 0, no buffer is ready to be read. If the answer is the same as the total number of buffers set by SetTransferBuffers(), all buffers are full, and an overflow will likely occur soon (if it have not already).

Parameters

filled
Pointer to where the number of filled buffers will be written

Returns

1 for successful operation and 0 for failure

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7

See also

SetTransferBuffers()

14.4.13 InitializeStreaming()

```
virtual int InitializeStreaming ( )
```

Initialize and set up a streaming data acquisition.

This function is an alternative to ContinuousStreamingSetup() and TriggeredStreamingSetup(). It allows a streaming acquisition with channel-specific settings to be set up, after first using functions such as SetChannelRecordLength() and SetChannelNumberOfRecords() to configure the channels.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7

See also

InitializeStreaming(), SetChannelRecordLength(), SetChannelNumberOfRecords(), SetChannelPretrigger(), SetChannelTriggerDelay(), SetChannelSampleSkip(), SetStreamingChannelMask()

14.4.14 SetChannelLevelTriggerMask()

```
virtual int SetChannelLevelTriggerMask (
    unsigned int      channel,
    unsigned int      level_trig_mask )
```

Sets the level trigger mask (which level triggers are used by a channel) for a specific channel.

This function allows channel-specific level trigger selection, of the level triggers setup by SetupLevelTrigger API

Parameters

channel Channel number, indexed from 1 and upwards, or set to 0xFFFFFFFF to apply to all channels.	
level_trig_mask An OR is performed for events for all 1's in this mask, to define the used trigger for the channel. Bit 0 corresponds to channel 1 and so forth. .	

Returns

1 for successful operation and 0 for failure

Note

Not all combinations of masks are allowed, see SetupLevelTrigger API documentation for limitations

This API must be called, for all channels, *after* setting up the level trigger with SetupLevelTrigger, as that call overrides the individual trigger masks

Trigger mode must be set to level trigger (at least for this channel), for these settings to have any effect

Valid for

ADQ14, ADQ7

See also

InitializeStreaming(), SetupLevelTrigger()

14.4.15 SetChannelNumberOfRecords()

```
virtual int SetChannelNumberOfRecords (
    unsigned int     channel,
    unsigned int     nofrecords,
    int              infinite_records )
```

Sets the number of records for a specific channel.

This function allows channel-specific amounts of records per data acquisition, when used prior to calling InitializeStreaming()

Parameters

channel	Channel number, indexed from 1 and upwards, or set to 0xFFFFFFFF to apply to all channels.
nofrecords	The number of records.
infinite_records	Set to 1 to enable an infinite number of records.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7

See also

InitializeStreaming()

14.4.16 SetChannelPretrigger()

```
virtual int SetChannelPretrigger (
    unsigned int    channel,
    unsigned int    pretrigger )
```

Sets the amount of pre-trigger for a specific channel.

This function allows channel-specific pretrigger settings, when used prior to calling InitializeStreaming()

The granularity of the pretrigger setting depends on the product type:

- ADQ14-2X/1X: 8 samples
- ADQ14-4C/2C: 4 samples
- ADQ14-4A/2A: 2 samples
- ADQ7: 32 samples in 1ch@10GSPS mode and 16 samples in 2ch@5GSPS mode

The requested pre-trigger value must match this granularity.

Parameters

channel	Channel number, indexed from 1 and upwards, or set to 0xFFFFFFFF to apply to all channels.
pretrigger	The amount of pretrigger (in units of samples).

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7

See also

InitializeStreaming()

14.4.17 SetChannelRecordLength()

```
virtual int SetChannelRecordLength (
    unsigned int    channel,
    unsigned int    length,
    int             infinite_length )
```

Sets the record length for a specific channel.

This function allows channel-specific record length settings, when used prior to calling InitializeStreaming()

Parameters

channel Channel number, indexed from 1 and upwards, or set to 0xFFFFFFFF to apply to all channels.
length The record length (in units of samples).
infinite_length Set to 1 to enable infinite record length (continuous streaming).

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7

See also

InitializeStreaming()

14.4.18 SetChannelSampleSkip()

```
virtual int SetChannelSampleSkip (
    unsigned int       channel,
    unsigned int       skipfactor )
```

Sets the sample skip factor for a specific channel.

This function allows channel-specific sample skip settings, when used prior to calling InitializeStreaming()

Parameters

channel Channel number, indexed from 1 and upwards, or set to 0xFFFFFFFF to apply to all channels.
skipfactor Sample skip factor (see SetSampleSkip() for valid factors per product)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7

See also

InitializeStreaming()

14.4.19 SetChannelTriggerDelay()

```
virtual int SetChannelTriggerDelay (
    unsigned int     channel,
    unsigned int     triggerdelay )
```

Sets the amount of trigger delay for a specific channel.

This function allows channel-specific trigger delay settings, when used prior to calling InitializeStreaming()

The granularity of the trigger delay setting depends on the product type:

- ADQ14-2X/1X: 8 samples
- ADQ14-4C/2C: 4 samples
- ADQ14-4A/2A: 2 samples
- ADQ7: 32 samples in 1ch@10GSPS mode and 16 samples in 2ch@5GSPS mode

The requested trigger delay value will be rounded up to the nearest multiple of the granularity.

Parameters

channel	Channel number, indexed from 1 and upwards, or set to 0xFFFFFFFF to apply to all channels.
triggerdelay	The amount of trigger delay (in units of samples).

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7

See also

InitializeStreaming()

14.4.20 SetChannelTriggerMode()

```
virtual int SetChannelTriggerMode (
    unsigned int     channel,
    int              trig_mode )
```

Sets the trigger mode for a specific channel.

This function allows channel-specific trigger selection.

Parameters

channel Channel number, indexed from 1 and upwards, or set to 0xFFFFFFFF to apply to all channels.
trig_mode See SetTriggerMode() for options.

Returns

1 for successful operation and 0 for failure

Note

This API must be called, for all channels, *after* setting up any generic trigger mode with SetTriggerMode, as that call overrides the individual trigger masks

Valid for

ADQ14, ADQ7

See also

InitializeStreaming(), SetTriggerMode()

14.4.21 SetFlushDMASize()

```
virtual unsigned int SetFlushDMASize (
    unsigned int      flush_size )
```

Set DMA flush size.

Set the flush block size for FlushDMA()

Parameters

flush_size Flush block size, must be a multiple of 16.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14

See also

FlushDMA()

14.4.22 SetGPVectorMode()

```
virtual int SetGPVectorMode (
    unsigned int      channel,
```

```
    unsigned int        mode )
```

Configures the behavior of the general purpose vector fields in the triggered streaming user header.

Parameters

channel The target channel, indexed from 1 and upwards.	
mode Valid values are ▪ 0: GPVector0: General purpose vector sampled at record start GPVector1: General purpose vector sampled at record end ▪ 1: GPVector0: General purpose vector sampled at record start GPVector1: Gate counter value ▪ 2: GPVector0: General purpose vector sampled at record end GPVector1: Gate counter value	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7

See also

GetGPVectorMode()

14.4.23 SetStreamConfig()

```
virtual int SetStreamConfig (
    unsigned int        option,
    unsigned int        value )
```

Sets the streaming configuration.

- Option = 0x1 Bypass DRAM Fifo
  - value = 0 => Using DRAM as fifo
  - value = 1 => DRAM fifo is disabled (only small fifo is used)
- Option = 0x2 Streaming mode
  - value = 0 => Packet headers are enabled
  - value = 1 => Raw without headers data order is deterministic
- Option = 0x3 Stream channel mask bit field
  - value = 0 => None
  - value = 1 => Channel A
  - value = 2 => Channel B
  - value = 4 => Channel C
  - value = 8 => Channel D

- Option = 0x5 Streaming mode (ADQ7 & ADQ14 only)
  - value = 0 => Records with headers (default)
  - value = 1 => Records without headers, data order is deterministic
- Option = 0x6 Deterministic data order (ADQ7 & ADQ14 only)
  - value = 0 => Disable deterministic data order
  - value = 1 => Enable deterministic data order

Parameters

option	Selected option
value	Configuration value

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ12, ADQ14

14.4.24 SetStreamingChannelMask()

```
virtual int SetStreamingChannelMask (
    unsigned int       channelmask )
```

Select which channels will transmit data to the host system during streaming.

This function allows enabling/disabling the transfer of data to host for specific channels, when used prior to calling InitializeStreaming()

Parameters

channelmask	Each bit corresponds to one data channel (bit 0 for channel 1, bit 1 for channel 2 and so on). Set a bit to 1 to enable data transfer for that channel

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14, ADQ7

See also

InitializeStreaming()

14.4.25 SetStreamStatus()

```
virtual int SetStreamStatus (
    int                status )
```

Controls streaming mode.

Parameters

status

The streaming mode to be selected. Use one of the following macros:

- ADQ214 & ADQ212:
  - `ADQ214_STREAM_DISABLED`
  - `ADQ214_STREAM_ENABLED_BOTH`
  - `ADQ214_STREAM_ENABLED_A`
  - `ADQ214_STREAM_ENABLED_B`
- ADQ114 & ADQ112:
  - `ADQ214_STREAM_DISABLED`
  - `ADQ214_STREAM_ENABLED`
- ADQ108, ADQ208, ADQ1600, SDR14
  - 0x0 (stream disabled)
  - 0x1 (stream enabled)
  - 0x9 (redirect data to DRAM)
- SDR14
  - 0x100 (Sort data linear)
- ADQ7/ADQ12/ADQ14: Use SetStreamConfig API instead

Returns

1 for successful operation and 0 for failure

Note

For ADQ214, ADQ114, ADQ212 and ADQ112: The start of the streaming may wait for a trigger after arming. To do this, make a bitwise 'or' between the selected macro and the macro `ADQxxx_STREAM_WAIT_FOR_TRIGGER`

When stream status is set to 0x1 and StartStreaming() is executed, data will be streamed immediately and the user application must start emptying with the API command CollectDataNextPage.

When stream status is set to 0x9, the DRAM may be emptied using the MemoryDump function after setting stream status to 0x0. Stream mode 0x9 Requires firmware revision 12920 or newer.

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7

See also

StartStreaming()

14.4.26 SetTransferBuffers()

```
virtual int SetTransferBuffers (
    unsigned int       nOfBuffers,
```

```
    unsigned int        bufferSize )
```

Sets the number and size of the data transfer buffers.

Parameters

nOfBuffers	The number of buffers
bufferSize	The transfer buffer size in bytes. The value is required to be a multiple of • 1024 on ADQ14, ADQ7 and ADQ8 • 512 on other products If MultiRecord mode is to be used, SetCacheSize() must be called to set a cache size that is less than or equal to the transfer buffer size.

Returns

1 for successful operation and 0 for failure

Note

This function could be used for MultiRecord mode, but this is rarely recommended.

This function allocates memory in the kernel space.

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

14.4.27 StartStreaming()

```
virtual int StartStreaming ( )
```

Sets up the software for streaming from the ADQ.

Calling this function will set the software to expect streamed data from the ADQ. To enable streamed data from the ADQ, SetStreamStatus() must be called.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7

See also

StopStreaming(), SetStreamStatus()

14.4.28 StopStreaming()

```
virtual int StopStreaming ( )
```

Stops streaming in the software.

To stop streaming on the ADQ, call SetStreamStatus().

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7

See also

StartStreaming(), SetStreamStatus()

14.4.29 TriggeredStreamingSetupV5()

```
virtual unsigned int TriggeredStreamingSetupV5 (
    unsigned int        SamplePerRecord,
    unsigned int        NofPreTrigSamples,
    unsigned int        NofTriggerDelaySamples,
    unsigned int        TriggeredStreamingFlags )
```

Sets the parameters for multi channels Triggered Streaming on V5 ADQs.

Please consult the Triggered Streaming example for V5 product for details on the execution flow

Parameters

SamplePerRecord The number of samples per waveform
NofPreTrigSamples The number of pretrigger samples for each waveform
NofTriggerDelaySamples The number of samples to hold off after the trigger event
TriggeredStreamingFlags A bit mask that specifies different behaviors, (similar to Waveform Averaging) ▪ 0x0001 (bit 0) Compensate data path for external trigger ▪ 0x0002 (bit 1) Compensate data path for level trigger ▪ 0x0004 (bit 2) Enable fastest readout ▪ 0x0008 (bit 3) Enable medium paced readout ▪ 0x0010 (bit 4) Enable slow readout ▪ 0x0020 (bit 5) Enable data path for using level trigger ▪ 0x0040 (bit 6) Enable the "get waveform" function. Cannot be used together with automatic readout and arm ▪ 0x0080 (bit 7) Enable automatic readout and arm (Used for streaming continuously). Cannot be used together "get waveform" function ▪ 0x0100 (bit 8) Choose to only read out data for channel A (ADQ214). ▪ 0x0200 (bit 9) Choose to only read out data for channel B (ADQ214). ▪ 0x0400 (bit 10) Immediate readout mode The different bits may be combined with the exception for bit 6 and 7.

Returns

1 for successful operation and 0 for failure

Note

If streaming over USB is used, one should preferably choose a sample size of the waveform that equals a packet size of 512 bytes. Each sample is 2 bytes but data is arriving in groups of 4 samples (2 samples for each channel), therefore sample sizes should be chosen as 128 samples increments ($128*2*2 = 512$).

Enabling the "get waveform" function will change the transfer settings of the device.

Triggered Streaming cannot be used together with Packet Streaming, Waveform Averaging or MultiRecord

Immediate readout is only available on ADQ214

On ADQ214 the maximum length waveform is 64k samples. Pretrigger, trigger delay and sample length is chosen by 2 sample increments.

Valid for

ADQ214

See also

TriggeredStreamingArmV5(), TriggeredStreamingGetWaveformV5(), TriggeredStreamingShutdownV5()

14.4.30 TriggeredStreamingShutdownV5()

```
virtual unsigned int TriggeredStreamingShutdownV5 ( )
```

Issues shutdown for Triggered Streaming on V5 product.

Used to gracefully stop the automatic readout and rearm mode. After issuing shutdown, please monitor and wait for the in_idle signal of the TriggeredStreamingGetStatusV5() command to go high before starting again.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

TriggeredStreamingDisarmV5(), TriggeredStreamingSetupV5()

14.4.31 WaitForTransferBuffer()

```
virtual unsigned int WaitForTransferBuffer (
    unsigned int *    filled,
    unsigned int      timeout_setting )
```

Wait for DMA buffers to be available (not working for USB2 interface units)

This function enables the host application to idle-wait for data to be available during streaming read-out. This call is blocking until data are available or a timeout has occurred. If filled is 0, no buffer is ready to be read. If filled is the same as the total number of buffers set by SetTransferBuffers(), all buffers are full, and an overflow will likely occur soon (if it have not already).

Parameters

filled	Pointer to where the number of filled buffers will be written
timeout_setting	Time out in ms

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7

See also

SetTransferBuffers(), GetTransferBufferStatus()

15 Offline data parsing

Functions

- int ADQData_Create (void **pref)
- int ADQData_Destroy (void *ref)
- int ADQData_EnableErrorTrace (void *ref, int trace_level, const char *filename, unsigned int append)
- int ADQData_GetDeviceStructPID (void *device_struct, const char *filename, unsigned int *pid)
- int ADQData_InitPacketStream (void *ref, void *device_struct, const char *filename)
- int ADQData_ParsePacketStream (void *ref, void *raw_data_buffer, unsigned int raw_data_size, void **target_buffers, void **target_headers, unsigned int *samples_added, unsigned int *headers_added, unsigned int *header_status, unsigned char channels_mask)
- int GetADQDataDeviceStruct (void *buffer)

  Get ADQData device structure.
- int GetADQDataDeviceStructSize (unsigned int *size)

  Get size of ADQData device structure.

15.1 Detailed Description

Functions related to parsing stored data from the digitizer offline

15.2 Function Documentation

15.2.1 ADQData_Create()

```
int ADQData_Create (
    void **           pref )
```

Create ADQData reference

This function creates a reference used in succeding ADQData calls.

Parameters

pref	Pointer to a void pointer where the reference will be returned by this call.

Note

This function uses the prefix ADQData_ instead of ADQ_ and does not have ADQControlUnit and ADQ number parameters.

Returns

1 if succesful, otherwise 0

See also

ADQData_Destroy()

Valid for

ADQ7

15.2.2 ADQData_Destroy()

```
int ADQData_Destroy (
    void *          ref )
```

Destroy ADQData reference

This function performs any cleanup related to the ADQData reference created by ADQData_Create().

Parameters

ref	ADQData reference from ADQData_Create().

Note

This function uses the prefix ADQData_ instead of ADQ_ and does not have ADQControlUnit and ADQ number parameters.

Returns

1 if succesful, otherwise 0

See also

ADQData_Create()

Valid for

ADQ7

15.2.3 ADQData_EnableErrorTrace()

```
int ADQData_EnableErrorTrace (
```

```
    void *            ref,
    int               trace_level,
    const char *      filename,
    unsigned int      append )
```

Enable tracelog from ADQData functions

This function enables logging of errors and warnings to file from ADQData functions..

Parameters

ref	ADQData reference from ADQData_Create().
trace_level	Trace level • trace_level = 0 : No error logging • trace_level = 1 : Error logging • trace_level = 2 : Error and warnings logging • trace_level = 3 : Error, warning, info logging Additionally if bit 11 is set (e.g. by ORing 2048 with the trace_level), timestamping will be enabled in the log.
filename	Path and filename of log file.
append	If set zero the log file will be truncated before new messages are added.

Note

This function uses the prefix ADQData_ instead of ADQ_ and does not have ADQControlUnit and ADQ number parameters.

Returns

1 if succesful, otherwise 0

See also

ADQData_Create()

Valid for

ADQ7

15.2.4 ADQData_GetDeviceStructPID()

```
int ADQData_GetDeviceStructPID (
    void *            device_struct,
    const char *      filename,
    unsigned int *    pid )
```

Get ADQData device structure PID

Parameters

device_struct ADQ device structure from GetADQDataDeviceStruct().	
filename Filename for stored ADQ device structure. If set to NULL device_struct parameter is used instead, if set to a filename the device_struct parameter is ignored.	
pid Return argument for PID from given device struct.	

Note

This function uses the prefix ADQData_ instead of ADQ_ and does not have ADQControlUnit and ADQ number parameters.

Returns

1 if succesful, otherwise 0

See also

GetADQDataDeviceStruct()

Valid for

ADQ7

15.2.5 ADQData_InitPacketStream()

```
int ADQData_InitPacketStream (
    void *          ref,
    void *          device_struct,
    const char *    filename )
```

Initialize ADQData packet stream

This function needs to be called before calling ADQData_ParsePacketStream()

Parameters

ref ADQData reference from ADQData_Create().	
device_struct ADQ device structure from GetADQDataDeviceStruct().	
filename Filename for stored ADQ device structure. If set to NULL device_struct parameter is used instead, if set to a filename the device_struct parameter is ignored.	

Note

This function uses the prefix ADQData_ instead of ADQ_ and does not have ADQControlUnit and ADQ number parameters.

Returns

1 if succesful, otherwise 0

See also

GetADQDataDeviceStruct()

Valid for

ADQ7

15.2.6 ADQData_ParsePacketStream()

```
int ADQData_ParsePacketStream (
    void *              ref,
    void *              raw_data_buffer,
    unsigned int        raw_data_size,
    void **             target_buffers,
    void **             target_headers,
    unsigned int *      samples_added,
    unsigned int *      headers_added,
    unsigned int *      header_status,
    unsigned char       channels_mask )
```

Parse out data and headers from raw packet stream

Parse raw packet stream from an earlier data collection.

Parameters

ref ADQData reference from ADQData_Create().
raw_data_buffer Pointer to raw packet stream from earlier collection.
raw_data_size Size of raw_data_buffer.
target_buffers Pointer to the user-allocated buffers where the data is to be placed. One buffer for each channel of data.
target_headers Pointer to the user-allocated buffers where the headers are to be placed. One buffer for each channel of data.
samples_added Number of samples parsed
channels_mask Channels mask to enable/disable data parsing from specific channels. Expecting a 4-bit bitmask.

Parameters

headers_added	Contains the number of initialized headers from this function call.
header_status	Contains the last header status. See description above.

Note

This function uses the prefix ADQData_ instead of ADQ_ and does not have ADQControlUnit and ADQ number parameters.

Returns

1 if succesful, otherwise 0

See also

GetADQDataDeviceStruct()

Valid for

ADQ7

15.2.7 GetADQDataDeviceStruct()

```
int GetADQDataDeviceStruct (
    void *              buffer )
```

Get ADQData device structure.

Returns the ADQData device structure. When storing data for later parsing this structure needs to be stored as well. This structure needs to be passed to ADQData_InitPacketStream() before stored data can be parsed using ADQData_ParsePacketStream(). All setup related to the transfer needs to be performed before calling this function and getting the relevant ADQData device structure.

Parameters

buffer	Pointer to allocated buffer where the device structure will be stored

Returns

1 for success, 0 otherwise

Valid for

ADQ7

See also

GetADQDataDeviceStruct

15.2.8 GetADQDataDeviceStructSize()

```
int GetADQDataDeviceStructSize (
    unsigned int * size )
```

Get size of ADQData device structure.

Returns the size of the ADQData device structure so that memory can be allocated beforehand.

Parameters

size	Pointer to integer where size will be stored

Returns

1 for success, 0 otherwise

Valid for

ADQ7

See also

GetADQDataDeviceStruct

16 Data Decimation and Sample Skip

Functions

- int ForceResynchronizationSampleSkip ()

  Force synchronization of sample skip.
- int GetChannelDecimation (unsigned int channel, unsigned int *decfactor)

  Get the current decimation factor for a specific channel.
- unsigned int GetSampleDecimation ()

  Gets the current decimation factor.
- unsigned int GetSampleSkip ()

  Gets the current sample skip factor.
- int SetChannelDecimation (unsigned int channel, unsigned int decfactor)

  Enables decimation with filtering and downsampling for a specific channel.
- unsigned int SetSampleDecimation (unsigned int SampleDecimation)

  Enables decimation with filtering and downsampling.
- unsigned int SetSampleSkip (unsigned int DecimationFactor)

  Enable sample skip of input data.
- int SetSampleSkipSyncMode (int sync_mode)

Set the synchronization mode used for sample skip.

16.1 Detailed Description

Especially for streaming applications it may be desired to reduce the rate of data. For this, data decimation or sample skip may be used depending on the unit type.

16.2 Function Documentation

16.2.1 ForceResynchronizationSampleSkip()

```
virtual int ForceResynchronizationSampleSkip ( )
```

Force synchronization of sample skip.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14

See also

SetSampleSkipSyncMode()

16.2.2 GetChannelDecimation()

```
virtual int GetChannelDecimation (
    unsigned int      channel,
    unsigned int *    decfactor )
```

Get the current decimation factor for a specific channel.

Parameters

channel	The channel for which the decimation factor will be read
decfactor	A pointer to an unsigned int where the read value will be stored. A value of N here indicates a decimation factor of 2^N.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

See also

SetChannelDecimation

16.2.3 GetSampleDecimation()

```
virtual unsigned int GetSampleDecimation ( )
```

Gets the current decimation factor.

Returns

The current value of the sample decimation unit. SetSampleDecimation() an explanation of the values

Note

For ADQ7, ADQ12 and ADQ14, decimation is only supported by firmware option FWSDR or FW4DDC

Valid for

ADQ214, ADQ212, SDR14, ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

See also

SetSampleDecimation()

16.2.4 GetSampleSkip()

```
virtual unsigned int GetSampleSkip ( )
```

Gets the current sample skip factor.

Returns

the current value of the sample-skip unit. See SetSampleSkip() for explanations of the values.

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, SDR14, ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetSampleSkip(), SetStreamStatus()

16.2.5 SetChannelDecimation()

```
virtual int SetChannelDecimation (
    unsigned int    channel,
    unsigned int    decfactor )
```

Enables decimation with filtering and downsampling for a specific channel.

Parameters

channel The channel for which the decimation factor will be set

Parameters

decfactor	A value of N here results in a decimation factor of 2^N.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

See also

GetChannelDecimation

16.2.6 SetSampleDecimation()

```
virtual unsigned int SetSampleDecimation (
    unsigned int       SampleDecimation )
```

Enables decimation with filtering and downsampling.

Parameters

SampleDecimation

The decimation factor. Examples:

- ADQ214 / ADQ14-FWSDR:
- SampleDecimation = 0 => No decimation
- SampleDecimation = 1 => Decimation by 2^1=2
- SampleDecimation = 2 => Decimation by 2^2=4
- etc...
- SampleDecimation = 34 => Decimation by 2^{34}
- SDR14:
  - SampleDecimation = 1 => No decimation
  - SampleDecimation = 2 => Decimation by 2
  - SampleDecimation = 4 => Decimation by 4
  - SampleDecimation = 8 => Decimation by 8
  - SampleDecimation = 16 => Decimation by 16
  - SampleDecimation = 32 => Decimation by 32
- ADQ7-FW2DDC:
  - SampleDecimation = 0 => No decimation
  - SampleDecimation = 1 => Decimation by 2^1=2
  - SampleDecimation = 2 => Decimation by 2^2=4
  - etc...
  - SampleDecimation = 34 => Decimation by 2^{34}

Returns

1 for successful operation and 0 for failure

Note

For ADQ7, ADQ12 and ADQ14, decimation is only supported by firmware option FWSDR or FW4DDC

Valid for

ADQ214, ADQ212, SDR14, ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

See also

GetSampleDecimation(), SetSampleSkip()

16.2.7 SetSampleSkip()

```
virtual unsigned int SetSampleSkip (
    unsigned int      DecimationFactor )
```

Enable sample skip of input data.

The sample skip factors available differs for different units:

ADQ214, ADQ212: 2, 4, 6, 8, ... , 131072
ADQ112, ADQ114: 2, 4, 8, 12, ... , 262140
ADQ1600, ADQ412, SDR14, ADQ208, ADQ108: 2, 4, 8, 16, 32, 64, 128
ADQ12-2X/-1X: 2, 4, 8, 9, 10, ..., 65536
ADQ12-4C/-2C: 2, 4, 5, 6, 7, ..., 65536
ADQ12-4A/-2A: 2, 3, 4, 5, 6, ..., 65536
ADQ14-2X/-1X: 2, 4, 8, 9, 10, ..., 65536
ADQ14-4C/-2C: 2, 4, 5, 6, 7, ..., 65536
ADQ14-4A/-2A: 2, 3, 4, 5, 6, ..., 65536
ADQ7-1CH: 2, 4, 8, 16, 32, 33, 34 ..., 65536
ADQ7-2CH: 2, 4, 8, 16, 17, 18 ..., 65536
ADQ8-8C: 2, 4, 5, 6, 7, ..., 65536

Parameters

DecimationFactor

The factor with which to skip samples

- skipsamples = 1: No samples skipped
- skipsamples = N: Every N:th sample kept

Returns

1 for successful operation and 0 for failure

Note

For FWSDR/FW4DDC firmware options, Sample Skip is not available, instead use SetChannelDecimation

Valid for

ADQ212, ADQ112, ADQ114, ADQ214, ADQ108, ADQ208, ADQ412, SDR14, ADQ1600, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetSampleSkip(), SetChannelDecimation()

16.2.8 SetSampleSkipSyncMode()

```
virtual int SetSampleSkipSyncMode (
    int                 sync_mode )
```

Set the synchronization mode used for sample skip.

Parameters

SyncMode	• 0: Internal only synchronization (default) • 1: Synchronized to next trigger for channel • 2: Synchronized to next edge of the reference clock

Returns

1 for successful operation and 0 for failure

Note

The synchronization is done when setting sample skip, but may also be forced later with ForceSynchronizationSampleSkip command

Valid for

ADQ12, ADQ14

See also

ForceResynchronizationSampleSkip()

17 ADX Interleaving IP

Functions

- unsigned int GetInterleavingIPBypassMode (unsigned char IPInstanceAddr, unsigned int *bypassflag)

  Gets the current ADX bypass selection.
- unsigned int GetInterleavingIPCalibration (unsigned char IPInstanceAddr, unsigned int *calibration)

  Gets the current calibration state of the ADX interleaving IP.
- unsigned int GetInterleavingIPEstimationMode (unsigned char IPInstanceAddr, unsigned int *updatetype)

  Gets the ADX parameter update mode.
- int InterleavingIPTemperatureAutoUpdate (unsigned int enable)

  Enabled update of ADC temperature to the ADX IP.

- unsigned int ResetInterleavingIP (unsigned char IPInstanceAddr)

  Resets the ADX interleaving IP.
- unsigned int SendIPCommand (unsigned int IPInstanceAddr, unsigned int cmd, unsigned int arg1, unsigned int arg2, unsigned int *answer)

  Gets the user direct access to the ADX/DBS command interface.
- unsigned int SetInterleavingIPBypassMode (unsigned char IPInstanceAddr, unsigned int bypassflag)

  Selects whether to bypass the ADX interleaving IP or not.
- unsigned int SetInterleavingIPCalibration (unsigned char IPInstanceAddr, unsigned int *calibration)

  Loads a calibration state into the ADX interleaving IP.
- unsigned int SetInterleavingIPEstimationMode (unsigned char IPInstanceAddr, unsigned int updatetype)

  Selects the ADX parameter update mode.

17.1 Detailed Description

The ADX Interleaving IP significantly increases performance when interleaving multiple channels. In most cases, no special configuration needs to be made.

17.2 Function Documentation

17.2.1 GetInterleavingIPBypassMode()

```
virtual unsigned int GetInterleavingIPBypassMode (
    unsigned char    IPInstanceAddr,
    unsigned int *   bypassflag )
```

Gets the current ADX bypass selection.

Parameters

IPInstanceAddr The ADX instance to get the current value from. ADQ412 and SDR14 contain two ADX IP instances (addressed by 0 and 1) and the other ADQs contain one (addressed by 0)	
bypassflag Pointer to where to return the value. Please see SetInterleavingIPBypassMode() for the meaning of the different values.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

SetInterleavingIPBypassMode()

17.2.2 GetInterleavingIPCalibration()

```
virtual unsigned int GetInterleavingIPCalibration (
    unsigned char     IPInstanceAddr,
    unsigned int *    calibration )
```

Gets the current calibration state of the ADX interleaving IP.

Parameters

IPInstanceAddr The ADX instance to get the current value from. ADQ412 and SDR14 contain two ADX IP instances (addressed by 0 and 1) and the other ADQs contain one (addressed by 0)	
calibration Pointer to where to store the output. This area must be allocated by the user, and at least 8kB (2048 32-bit integers) large.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

SetInterleavingIPCalibration()

17.2.3 GetInterleavingIPEstimationMode()

```
virtual unsigned int GetInterleavingIPEstimationMode (
    unsigned char     IPInstanceAddr,
    unsigned int *    updatetype )
```

Gets the ADX parameter update mode.

The ADX interleaving IP may be configured to not update its internal parameters. This function gets the current value.

Parameters

IPInstanceAddr The ADX instance to get the current value from. ADQ412 and SDR14 contain two ADX IP instances (addressed by 0 and 1) and the other ADQs contain one (addressed by 0).	
updatetype A pointer to where to return the value. Please see SetInterleavingIPEstimationMode() for the meaning of the different values.	

Returns

1 for successful operation and 0 for failure

Note

This function is for advanced use cases. Please contact SP Devices support for more information on the different modes.

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

SetInterleavingIPEstimationMode()

17.2.4 InterleavingIPTemperatureAutoUpdate()

```
virtual int InterleavingIPTemperatureAutoUpdate (
    unsigned int      enable )
```

Enabled update of ADC temperature to the ADX IP.

Parameters

enable	Set to 1 to enable, 0 to disable.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14

17.2.5 ResetInterleavingIP()

```
virtual unsigned int ResetInterleavingIP (
    unsigned char      IPInstanceAddr )
```

Resets the ADX interleaving IP.

Parameters

IPInstanceAddr	The ADX instance to reset.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14, ADQ14, ADQ7

See also

SetInterleavingIPBypassMode()

17.2.6 SendIPCommand()

```
virtual unsigned int SendIPCommand (
    unsigned int      IPInstanceAddr,
    unsigned int      cmd,
    unsigned int      arg1,
    unsigned int      arg2,
    unsigned int *    answer )
```

Gets the user direct access to the ADX/DBS command interface.

This is an advanced user function, which should rarely be used.

Parameters

IPInstanceAddr	The ADX instance to get the current value from ADQ412 and SDR14 contain two ADX IP instances (addressed by 0 and 1) and the other ADQs contain one (addressed by 0) To instead communicate with DBS IPs, set bit 16-23 to 1.
cmd	Selects the command
arg1	First argument
arg2	Second argument
answer	Pointer to the answer returned from the function

Returns

1 for successful operation and 0 for failure

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7

See also

SetInterleavingIPBypassMode(), ResetInterleavingIP(), SetInterleavingIPEstimationMode()

17.2.7 SetInterleavingIPBypassMode()

```
virtual unsigned int SetInterleavingIPBypassMode (
```

```
    unsigned char     IPInstanceAddr,
    unsigned int      bypassflag )
```

Selects whether to bypass the ADX interleaving IP or not.

While the ADX IP is bypassed, it will still update its current parameters.

Parameters

IPInstanceAddr The ADX instance to update setting for. ADQ412 and SDR14 contain two ADX IP instances (addressed by 0 and 1) and the other ADQs contain one (addressed by 0)
bypassflag The bypass selection: • 0: use correction • 1: bypass correction (default)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

GetInterleavingIPBypassMode(), SetInterleavingIPEstimationMode()

17.2.8 SetInterleavingIPCalibration()

```
virtual unsigned int SetInterleavingIPCalibration (
    unsigned char     IPInstanceAddr,
    unsigned int *    calibration )
```

Loads a calibration state into the ADX interleaving IP.

Parameters

IPInstanceAddr The ADX instance to update setting for. ADQ412 and SDR14 contain two ADX IP instances (addressed by 0 and 1) and the other ADQs contain one (addressed by 0)
calibration Pointer to where to read the calibration state from. The memory contents must be fetched by GetInterleavingIPCalibration().

Returns

1 for successful operation and 0 for failure

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

GetInterleavingIPCalibration()

17.2.9 SetInterleavingIPEstimationMode()

```
virtual unsigned int SetInterleavingIPEstimationMode (
    unsigned char    IPInstanceAddr,
    unsigned int     updatetype )
```

Selects the ADX parameter update mode.

The ADX interleaving IP may be configured to not update its internal parameters. It is typically not desired to change the default setting.

Parameters

IPInstanceAddr	The ADX instance to update setting for.
updatetype	The setting. These options are available: • 0: No updates allowed • 1: Normal mode, with continuous updates (default) • 2: Time-domain mode

Returns

1 for successful operation and 0 for failure

Note

This function is for advanced use cases. Please contact SP Devices support for more information on the different modes.

Valid for

ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

GetInterleavingIPEstimationMode(), SetInterleavingIPBypassMode()

18 Digital Baseline Stabilization

Functions

- unsigned int GetNofDBSInstances (unsigned int *nof_dbs_instances)

  Gets the number of verified DBS instances.
- unsigned int SetupDBS (unsigned char DBS_instance, unsigned int bypass, int dc_target, int lower_saturation_level, int upper_saturation_level)

Performs setup of Digital Baseline Stabilization (DBS)

18.1 Detailed Description

Digital Baseline Stabilization (DBS) is used to provide a stable baseline by active digital compensation.

18.2 Function Documentation

18.2.1 GetNofDBSInstances()

```
virtual unsigned int GetNofDBSInstances (
    unsigned int *    nof_dbs_instances )
```

Gets the number of verified DBS instances.

Parameters

nof_dbs_instances
Pointer to where the number of verified DBS cores is to be returned

Returns

1 for successful operation and 0 for failure

Valid for

ADQ1600, ADQ412, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetupDBS()

18.2.2 SetupDBS()

```
virtual unsigned int SetupDBS (
    unsigned char    DBS_instance,
    unsigned int     bypass,
    int              dc_target,
    int              lower_saturation_level,
    int              upper_saturation_level )
```

Performs setup of Digital Baseline Stabilization (DBS)

Initializes the DBS algorithm, sets a number of user-specified settings, and enables or bypasses the DBS block.

Parameters

DBS_instance The instance of DBS to setup. Only 0 is valid for ADQ1600. 0 or 1 is valid for ADQ412. 0 to 3 are valid for ADQ7/ADQ12/ADQ14. Addressing with 0xFF sets up all 4 cores (if 4 is available) to same settings Addressing with 0xFE sets up core 0 and 1 to same settings Addressing with 0xFD sets up core 2 and 3 to same settings	
bypass Selects whether the DBS instance should be active (bypass = 0) or bypassed (bypass = 1)	
dc_target Sets the DC target in ADC codes. The data format is 16-bit MSB aligned for ADQ1600.	
lower_saturation_level Advanced paramameter that selects how many codes below the baseline the signal may be before it is ignored in the DC estimation. The value is given as a negative number. Set this parameter to 0 to use the default level.	
upper_saturation_level advanced paramameter that selects how many codes above the baseline the signal may be before it is ignored in the DC estimation. Set this parameter to 0 to use the default level.	

Returns

1 for successful operation and 0 for failure

Note

The analog baseline must be stable when starting DBS. If the adjustable bias level is changed (on a unit were this is possible), a pause of around two seconds should be made before starting DBS.

Valid for

ADQ1600, ADQ412, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetNofDBSInstances()

19 Waveform Averaging

Functions

- unsigned int WaveformAveragingArm ()

  Arms the Waveform Averaging.
- unsigned int WaveformAveragingDisarm ()

  Disarms the Waveform Averaging and puts it in bypass mode.
- unsigned int WaveformAveragingGetStatus (unsigned char *ready, unsigned int *nofrecordscompleted, unsigned char *in_idle)

  Gets the status of the averaging block.
- unsigned int WaveformAveragingGetWaveform (int *waveform_data)

  Collects a waveform from the Waveform Averaging block.

- unsigned int WaveformAveragingParseDataStream (unsigned int samples_per_record, int ∗data_stream, int ∗∗data_target)

  Parses a buffer filled with a single record of WFA data.
- unsigned int WaveformAveragingSetup (unsigned int NofWaveforms, unsigned int NofSamples, unsigned int NofPreTrigSamples, unsigned int NofTriggerDelaySamples, unsigned int WaveformAveragingFlags)

  Sets up the Waveform Averaging block.
- unsigned int WaveformAveragingShutdown ()

  Issues shutdown for Waveform Averaging.
- unsigned int WaveformAveragingSoftwareTrigger ()

  Issues a software trigger to the WFA module.
- unsigned int WfaArm ()

  Arms the Waveform Averaging on SDR14.
- unsigned int WfaDisarm ()

  Disarms the Waveform Averaging and puts it in bypass mode.
- unsigned int WfaGetStatus (unsigned int ∗data_available, unsigned int ∗in_idle, unsigned int ∗overflow, unsigned int ∗transfer_in_progress, unsigned int ∗channel_sync_error, unsigned int ∗waveforms_accumulated)

  Gets the status of the averaging block.
- unsigned int WfaGetWaveform ()

  Send a readout signal to the Waveform Averaging block when using manual re-arm mode.
- unsigned int WfaSetup (unsigned int NofWaveforms, unsigned int NofSamples, unsigned int NofPreTriggerSamples, unsigned int NofTriggerDelaySamples, unsigned int NofReadoutWaitCycles, unsigned int trigger_mode, unsigned int trigger_edge, unsigned int triggers_limit, unsigned int ArmMode, unsigned int ReadoutMode, unsigned int AccMode)

  Sets up the Waveform Averaging block on SDR14 from firmare version 25118.
- unsigned int WfaShutdown ()

  Issues shutdown for Waveform Averaging.

19.1 Detailed Description

Waveform Averaging (WFA) is used to average multiple collections together on the devices. These are then typically transferred to host using the streaming mode.

19.2 Function Documentation

19.2.1 WaveformAveragingArm()

```
virtual unsigned int WaveformAveragingArm ( )
```

Arms the Waveform Averaging.

Triggers will be accepted by the Waveform Averaging block only after this call. If automatic readout and arm is active, readout will occur once average is done and a new average will restart when readout is completed.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212, ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

WaveformAveragingDisarm(), WaveformAveragingSetup()

19.2.2 WaveformAveragingDisarm()

```
virtual unsigned int WaveformAveragingDisarm ( )
```

Disarms the Waveform Averaging and puts it in bypass mode.

After this call, the ADQ may again be used in e.g. MultiRecord mode

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212, ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

WaveformAveragingArm(), WaveformAveragingShutdown(), WaveformAveragingSetup()

19.2.3 WaveformAveragingGetStatus()

```
virtual unsigned int WaveformAveragingGetStatus (
    unsigned char *   ready,
    unsigned int *    nofrecordscompleted,
    unsigned char *   in_idle )
```

Gets the status of the averaging block.

Parameters

ready Pointer to where to store whether data is available for readout. If this pointer is NULL, the value will not be written. If value stored at pointer is 3 this means that all data is available for readout. If this value is non-zero but lower than 3, then it means that some data is available but not all. User should wait untill all data is ready before readout.
nofrecordscompleted Pointer to where the to store the number of acquired records. If this pointer is NULL, the value will not be written
in_idle Pointer to where to store whether the collection logic is idle. If this pointer is NULL, the value will not be written

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212, ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

WaveformAveragingGetWaveform(), WaveformAveragingSetup()

19.2.4 WaveformAveragingGetWaveform()

```
virtual unsigned int WaveformAveragingGetWaveform (
    int *              waveform_data )
```

Collects a waveform from the Waveform Averaging block.

Readout will only succeed if data has been collected by the device. In manual rearm mode, WaveformAveragingGetStatus() can be called to verify that data is available.

Parameters

waveform_data Pointer to where the output is to be stored. The data is stored as signed 32-bit integers, and the user is responsible that enough memory is allocated at this pointer.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212, ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

WaveformAveragingArm(), WaveformAveragingGetStatus(), WaveformAveragingSetup()

19.2.5 WaveformAveragingParseDataStream()

```
virtual unsigned int WaveformAveragingParseDataStream (
    unsigned int       samples_per_record,
    int *              data_stream,
    int **             data_target )
```

Parses a buffer filled with a single record of WFA data.

The parsed data is stored in buffers provided by the user.

Parameters

samples_per_record The number of samples per channel in the buffer data_stream	

Parameters

data_stream	Pointer to the buffer containing the data that will be parsed
data_target	Pointer table to buffers where the outputs will be stored, one buffer per channel. For example, data_target[0] is a pointer to the channel A output buffer, and data_target[1] is a pointer to the channel B output buffer.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, SDR14, ADQ214, ADQ212, ADQ114, ADQ112

See also

WaveformAveragingGetWaveform()

19.2.6 WaveformAveragingSetup()

```
virtual unsigned int WaveformAveragingSetup (
    unsigned int     NofWaveforms,
    unsigned int     NofSamples,
    unsigned int     NofPreTrigSamples,
    unsigned int     NofTriggerDelaySamples,
    unsigned int     WaveformAveragingFlags )
```

Sets up the Waveform Averaging block.

Please consult the Waveform Averaging example for details on the execution flow

Parameters

NofWaveforms	The number of waveforms to average
NofSamples	The number of samples per waveform
NofPreTrigSamples	The number of pretrigger samples for each waveform
NofTriggerDelaySamples	The number of samples to hold off after the trigger event

Parameters

WaveformAveragingFlags	A bit mask that specifies different behaviors ▪ 0x0001 (bit 0) Compensate data path for external trigger ▪ 0x0002 (bit 1) Compensate data path for level trigger ▪ 0x0004 (bit 2) Enable fastest readout ▪ 0x0008 (bit 3) Enable medium paced readout ▪ 0x0010 (bit 4) Enable slow readout ▪ 0x0020 (bit 5) Enable data path for using level trigger ▪ 0x0040 (bit 6) Enable the waveform get function ▪ 0x0080 (bit 7) Enable automatic readout and arm (Used for streaming continuously) ▪ 0x0400 (bit 10) Immediate readout mode ▪ 0x1000 (bit 12) Choose channel A as input when running WFA in one channel mode (ADQ214) ▪ 0x2000 (bit 13) Choose channel B as input when running WFA in one channel mode (ADQ214) The different bits may be combined

Returns

1 for successful operation and 0 for failure

Note

It is important to use the correct size for the waveform depeding on which interface (USB or P∗Ie) and which product is being used. Waveform Averaging is running via streaming mode and the minimum packet size for this mode is different for different interface. On USB the streaming packet size is 512 bytes and on P∗Ie it is 128 bytes. Different ADQ products can have different number of channels and also different number of samples per clock cycle. It is therefore required by the user to set the appropriate waveform size to match the product in use. The following formula can be used to calculate the smalest waveform size for different product that respects the streaming packet size on different interface. Any other waveform size that the user wishes to use should be in multiples of this smallest size:

```
min_waveform_size = interface_packet_size/(number_of_channels*4)
```

Since different waveform size for individual channel is not supported, this number is the smalest waveform size possible, no matter how many channels the product has.

Enabling the waveform get function will change the transfer settings of the device.

Waveform Averaging cannot be used together with Packet Streaming, Triggered Streaming or MultiRecord

Immediate readout is only available on ADQ214, ADQ212, ADQ114, ADQ112 and SDR14.

When running in one channel input mode (to gain longer record length) only one channel can be chosen. Special custom firmware is required to use this mode. If both channels have been set OR no channel has been set when using such a firmware, default channel will be A. On standard firmware without support for this one channel input mode, these two flags will have no meaning.

On ADQ114 and ADQ112 the maximum length waveform is 32k samples and maximum waveform count is 64k. Pretrigger, trigger delay and sample length is chosen by 4 sample increments.

Valid for

ADQ214, ADQ212, ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

WaveformAveragingArm(), WaveformAveragingGetWaveform(), WaveformAveragingShutdown()

19.2.7 WaveformAveragingShutdown()

```
virtual unsigned int WaveformAveragingShutdown ( )
```

Issues shutdown for Waveform Averaging.

Used to gracefully stop the automatic readout and rearm mode. After issuing shutdown, please monitor and wait for the in_idle signal of the WaveformAveragingGetStatus() command to go high before starting again.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214, ADQ212, ADQ112, ADQ114, ADQ412, ADQ1600, SDR14

See also

WaveformAveragingDisarm(), WaveformAveragingSetup()

19.2.8 WaveformAveragingSoftwareTrigger()

```
virtual unsigned int WaveformAveragingSoftwareTrigger ( )
```

Issues a software trigger to the WFA module.

Only valid for V6 digitizers. For V5 digitizers (ADQ214, ADQ114, ADQ212 & ADQ112), SWTrig() should be used instead.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, SDR14

See also

SWTrig()

19.2.9 WfaArm()

```
virtual unsigned int WfaArm ( )
```

Arms the Waveform Averaging on SDR14.

Triggers will be accepted by the Waveform Averaging block only after this call. If auto-rearm and readout is active, readout will occur once average is done and a new average will restart when readout is completed.

Returns

1 for successful operation and 0 for failure 1 for success, 0 otherwise. Check the errorlog or ADQ Monitor in ADQUpdater for further details on failed return.

Valid for

SDR14 firmware revision 25118 or greater.

See also

WfaDisarm(), WfaSetup()

19.2.10 WfaDisarm()

```
virtual unsigned int WfaDisarm ( )
```

Disarms the Waveform Averaging and puts it in bypass mode.

After this call, the ADQ may again be used in e.g. MultiRecord mode

Returns

1 for successful operation and 0 for failure

Note

This function will call ResetDevice(3) if streaming overflow is detected to restore the streaming interface and make it possible to switch back to multi-record mode.

Valid for

SDR14 firmware revision 25118 or greater.

See also

WfaArm(), WfaShutdown(), WfaSetup()

19.2.11 WfaGetStatus()

```
virtual unsigned int WfaGetStatus (
    unsigned int * data_available,
    unsigned int * in_idle,
    unsigned int * overflow,
    unsigned int * transfer_in_progress,
    unsigned int * channel_sync_error,
    unsigned int * waveforms_accumulated )
```

Gets the status of the averaging block.

Parameters

data_available Pointer to store the data available flag when using manual re-arm mode.
in_idle Pointer to store the idle flag which indicates whether WFA has entered the idle state. Used to in manual re-arm mode.

Parameters

overflow Pointer to store the overflow flag. There are different kinds of overflow. The bit field below defines the overflow type: Overflow Bit position meaning: • 0 = Global Device Streaming Overflow Flag • 1 = Waveform Average Output Fifo Overflow Channel A • 2 = Waveform Average Output Fifo Overflow Channel B • 3 = Waveform Average Accumulation Overflow Channel A (Accumulation sum exceeds 32 bits) • 4 = Waveform Average Accumulation Overflow Channel A (Accumulation sum exceeds 32 bits)
transfer_in_progress Pointer to store the transfer in progress flag. Useful when collecting waveforms with long time spann (via sample skip or decimation)
channel_sync_error Pointer to store the sync error signal between the channels. Useful to detect if the outputs from the different channels are in sync or not. Sync error should normally not happen unless there is something wrong with the firmware or when the sepup for both channels differs in some way.
waveforms_accumulated Pointer to store the number of waveforms that has been accumulated so far. Useful when averaging many waveforms.

Returns

1 for success, 0 otherwise. Check the errorlog or ADQ Monitor in ADQUpdater for further details on failed return.

Note

The flags are returned as one bit per channel. I.E if data_available = 3 (bin 11), this means that both channels have data avaiable.

Valid for

SDR14 firmware revision 25118 or greater.

See also

WfaGetWaveform(), WfaSetup()

19.2.12 WfaGetWaveform()

```
virtual unsigned int WfaGetWaveform ( )
```

Send a readout signal to the Waveform Averaging block when using manual re-arm mode.

Readout will only succeed if data has been collected by the device. It is the users responsibility to check for the status (WfaGetStatus()) and determines when it is appropriate to readout the accumulated resulting waveform. These two status flags are useful to check befor reading out:

- data_available
- waveforms_accumulated

Returns

1 for success, 0 otherwise. Note that success here only indicates that the command was send successfully and not the success of readout.

Valid for

SDR14 firmware revision 25118 or greater.

See also

WfaArm(), WfaGetStatus(), WfaSetup()

19.2.13 WfaSetup()

```
virtual unsigned int WfaSetup (
    unsigned int        NofWaveforms,
    unsigned int        NofSamples,
    unsigned int        NofPreTriggerSamples,
    unsigned int        NofTriggerDelaySamples,
    unsigned int        NofReadoutWaitCycles,
    unsigned int        trigger_mode,
    unsigned int        trigger_edge,
    unsigned int        triggers_limit,
    unsigned int        ArmMode,
    unsigned int        ReadoutMode,
    unsigned int        AccMode )
```

Sets up the Waveform Averaging block on SDR14 from firmare version 25118.

Please consult the SDR14 Waveform Averaging example for details on the execution flow.

Parameters

NofWaveforms The number of waveforms to average. (to add together sample by sample)
NofSamples The number of samples per waveform
NofPreTriggerSamples The number of pretrigger samples for each waveform
NofTriggerDelaySamples The number of samples to hold off after the trigger event
NofReadoutWaitCycles The number of clock cycles to wait between each readout pulse. Used to adjust readout speed. Only valid for readout mode 0.

Parameters

trigger_mode Trigger mode to collect the individual waveforms. ▪ 1: Software Trigger. ▪ 2: External Trigger. ▪ 3: Not supported yet (level trigger) ▪ 4: Internal trigger.
trigger_edge Set the trigger edge to initiate waveform colelction for trigger mode that has this property (I. E external trigger). ▪ 0: Falling Edge. ▪ 1: Rising Edge
triggers_limit Should be set to zero by default. Used to set the total number of triggers that will be accepted by the accumulator. Can be used for diagnostic and debugging purpose.
ArmMode Set the arm mode after each accumulation is done. ▪ 0: Manual Re-arm. User must manually readout the averaged waveform and re-arm the accumulator from the PC once an accumulation is done. ▪ 1: = Auto Re-arm. The accumulator will automatically push out the resulting waveform and re-arm itself once an accumlation is done. This will require that the software on the PC side must be fast enough to receive the data. Otherwise overflow might occur.
ReadoutMode Set the readout mode once an accumulation is done. ▪ 0: Normal Readout. The resulting waveform is first stored in memory and can be readout according the readout speed set by NofReadoutWaitCycles. ▪ 1: Immediate Readout. The final resulting waveform will not be stored in memory. Readout will automatically follow once NofWaveforms has been reached. This Readout mode is only avaiable when using Auto Re-arm mode. Used in high performance application.
AccMode Set the accumulation mode for each individual waveform that will make up the final averaged result. ▪ 0: No dead time between each waveform. Only one trigger pulse is needed to produce the final average result. The first sample of one waveform will be right after the last sample of the previous waveform. ▪ 1: One trigger per waveform. Every individual waveform must be triggered by a trigger.

Returns

1 for success, 0 otherwise. Check the errorlog or ADQ Monitor in ADQUpdater for further details on failed return.

Note

It is important to use the correct size for the waveform depeding on which interface (USB or P∗Ie) and which product is being used. Waveform Averaging is running via streaming mode and the minimum packet size for this mode is different for different interface. On USB the streaming packet size is 512 bytes and on P∗Ie it is 128 bytes. Different ADQ products can have different number of channels and also different number of samples per clock cycle. It is therefore required by the user to set the appropriate waveform size to match the product in use. The following formula can be used to calculate the smalest waveform size for different product that respects the streaming packet size on different interface. Any other waveform size that the user wishes to use should be in multiples of this smalest size:

```
min_waveform_size = interface_packet_size/(number_of_channels*4)
```

Waveform Averaging cannot be used together with Packet Streaming, Triggered Streaming or Multi-Record.

Waveform Averaging in firmware only performs addition on the waveforms samples. No division is performed due to the hardware cost. Pretrigger, trigger delay and sample length is chosen by 4 sample increments.

Valid for

SDR14 firmware revision 25118 or greater.

See also

WfaArm(), WfaGetWaveform(), WfaShutdown()

19.2.14 WfaShutdown()

```
virtual unsigned int WfaShutdown ( )
```

Issues shutdown for Waveform Averaging.

Used to gracefully stop the automatic readout and rearm mode. After issuing shutdown, please monitor and wait for the in_idle signal of the WfaGetStatus() command to go high before starting again.

Returns

1 for success, 0 otherwise. Check the errorlog or ADQ Monitor in ADQUpdater for further details on failed return.

Valid for

SDR14 firmware revision 25118 or greater.

See also

WfaDisarm(), WfaSetup()

20 Advanced Time Domain

Data Structures

- struct ATDWFABufferStruct

  WFA buffer struct. More...

Enumerations

- enum ATDWFABufferFormat {
  ATD_WFA_BUFFER_FORMAT_INT32 = 0 ,

ATD_WFA_BUFFER_FORMAT_STRUCT = 1 }

ATD buffer format enum.

Functions

- int ATDEnableAccumulationGridSync (unsigned int enable)

  Enable the reset of the accumulation grid.
- int ATDFlushWFA (void)

  Flush ATD WFA streaming buffer.
- int ATDGetAdjustedRecordLength (unsigned int record_length, int search_direction)

  Receive help with selecting an appropriate record length.
- unsigned int ATDGetDeviceNofAccumulations (unsigned int nof_accumulations)

  Get the number of accumulations performed by the ADQ.
- int ATDGetWFAPartitionBoundaries (unsigned int *partition_lower_bound, unsigned int *partition_upper_bound)

  Get the upper and lower boundaries for the WFA workload partitioning.
- int ATDGetWFAStatus (unsigned int *wfa_progress_percent, unsigned int *records_collected, unsigned int *stream_status, unsigned int *wfa_status)

  Read out status of ATD Wave Form Averaging module.
- int ATDRegisterWFABuffer (unsigned int channel, void *buffer)

  Register ATD WFA buffers.
- int ATDSetThresholdFilter (unsigned int channel, unsigned int *coefficients)

  Setup ATD Threshold module filter.
- int ATDSetupThreshold (unsigned int channel, int threshold, int baseline, unsigned int polarity, unsigned int bypass)

  Setup ATD Threshold module.
- int ATDSetupWFA (unsigned int record_length, unsigned int nof_pretrig_samples, unsigned int nof_triggerdelay_samples, unsigned int nof_accumulations, unsigned int nof_repeats)

  Setup ATD Wave Form Averaging module.
- int ATDSetupWFAAdvanced (unsigned int segment_length, unsigned int segments_per_record, unsigned int accumulations_per_batch, unsigned int record_length, unsigned int nof_accumulations, unsigned int nof_pretrig_samples, unsigned int nof_triggerdelay_samples, unsigned int bypass)

  Setup ATD Wave Form Averaging module using detailed parameters.
- int ATDSetWFABufferFormat (enum ATDWFABufferFormat format)

  Set the format of the ATD data buffer.
- int ATDSetWFAInternalTimeout (unsigned int timeout_ms)

  Set the default internal timeout setting in milliseconds.
- int ATDSetWFAPartitionBoundaries (unsigned int partition_lower_bound, unsigned int partition_upper_bound)

  Set the upper and lower boundaries for the WFA workload partitioning.
- int ATDSetWFAPartitionBoundariesDefault ()

  Set the default upper and lower boundaries for the WFA workload partitioning.
- int ATDStartWFA (void **target_buffers, unsigned char channels_mask, unsigned int blocking)

Start ATD WFA.

- int ATDStopWFA (void)

  Stop ATD WFA.

- int ATDUpdateNofAccumulations (unsigned int nof_accumulations)

  Set the number of accumulations.

- int ATDWaitForWFABuffer (unsigned int channel, void **buffer, int timeout)

  Wait for ATD WFA buffer.

- int ATDWaitForWFACompletion (void)

  Wait for ATD WFA completion.

20.1 Detailed Description

These functions are used to configure devices with the -FWATD firmware option.

20.2 Data Structure Documentation

20.2.1 struct ATDWFABufferStruct

WFA buffer struct.

Data structure used in FWATD. Valid fields:

- ADQ14: Timestamp, Data and Channel
- ADQ7: All

Data Fields

uint8_t	Channel	Indexed from 1
int32_t *	Data	Pointer to data array. Memory is manage by user application
uint32_t	RecordNumber	Starts at 1
uint32_t	RecordsAccumulated	Number of accumulated records in current batch
uint32_t	Status	
uint64_t	Timestamp	Timestamp of first trigger in accumulation

20.3 Enumeration Type Documentation

20.3.1 ATDWFABufferFormat

`enum ATDWFABufferFormat`

ATD buffer format enum.

Used in ATDSetWFABufferFormat() to switch the WFA buffer format.

Enumerator

ATD_WFA_BUFFER_FORMAT_INT32	int_32t (default)
ATD_WFA_BUFFER_FORMAT_STRUCT	struct ATDWFABufferStruct

20.4 Function Documentation

20.4.1 ATDEnableAccumulationGridSync()

```
virtual int ATDEnableAccumulationGridSync (
    unsigned int      enable )
```

Enable the reset of the accumulation grid.

When enabled the accumulation grid will reset on each trigger blocking event. Unfinished accumulations may be discarded.

Parameters

enable
1 to enable, 0 to disable

Returns

1 for success, 0 otherwise

Valid for

ADQ7-FWATD

20.4.2 ATDFlushWFA()

```
virtual int ATDFlushWFA (
    void              )
```

Flush ATD WFA streaming buffer.

Flushes streaming buffer during ATD WFA transfer.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

20.4.3 ATDGetAdjustedRecordLength()

```
virtual int ATDGetAdjustedRecordLength (
```

```
    unsigned int        record_length,
    int                 search_direction )
```

Receive help with selecting an appropriate record length.

This function will help the user select a valid record length according to the rule set imposed by FWATD. The maximum record length returned by this function is 1M samples.

Parameters

record_length
search_direction '-1' to perform a descending search, returning a record length equal to or less than record_length. '1' performs an ascending search, returning a record length equal to or greater than record_length.

Returns

The adjusted record length. Negative numbers are error codes.

Valid for

ADQ12-FWATD, ADQ14-FWATD

20.4.4 ATDGetDeviceNofAccumulations()

```
virtual unsigned int ATDGetDeviceNofAccumulations (
    unsigned int        nof_accumulations )
```

Get the number of accumulations performed by the ADQ.

Perform the same WFA workload partitioning that will be performed by ATDSetupWFA().

Parameters

nof_accumulations The total number of accumulations

Returns

The number of accumulations performed by the device. nof_accumulations divided by the return value is the number of accumulations performed by the API.

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

See also

ATDSetWFAPartitionBoundaries() ATDSetWFAPartitionBoundariesDefault() ATDGetWFAPartitionBoundaries()

20.4.5 ATDGetWFAPartitionBoundaries()

```
virtual int ATDGetWFAPartitionBoundaries (
    unsigned int *   partition_lower_bound,
    unsigned int *   partition_upper_bound )
```

Get the upper and lower boundaries for the WFA workload partitioning.

Retrieve the values of the WFA partition boundaries.

Parameters

partition_lower_bound	The lower boundary for the WFA partitioning.
partition_upper_bound	The upper boundary for the WFA partitioning.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD

See also

ATDSetWFAPartitionBoundaries() ATDSetWFAPartitionBoundariesDefault() ATDGetDeviceNofAccumulations()

20.4.6 ATDGetWFAStatus()

```
virtual int ATDGetWFAStatus (
    unsigned int *   wfa_progress_percent,
    unsigned int *   records_collected,
    unsigned int *   stream_status,
    unsigned int *   wfa_status )
```

Read out status of ATD Wave Form Averaging module.

Parameters

wfa_progress_percent	WFA progress in percent, for all accumulations (including repeats).
records_collected	Number of records collected by the WFA.
stream_status	Status of transfer to host, zero if no errors have occurred.

Parameters

wfa_status Status of accumulation, zero if no errors have occurred. Set bits correspond to the following errors on ADQ12-FWATD and ADQ14-FWATD: • 0x001 Accumulator overflow. • 0x002 Input FIFO overflow (fast). • 0x004 DRAM FIFO overflow (fast). • 0x008 Input FIFO overflow (slow). • 0x010 DRAM FIFO overflow (slow). • 0x020 RAM collision. • 0x040 Accumulation segment store FIFO overflow (fast). • 0x080 Accumulation segment fetch FIFO overflow (fast). • 0x100 WFA segment FIFO overflow (fast). • 0x200 Raw segment FIFO overflow (slow). • 0x400 RAM collision during readout. • 0x800 Host accumulation overflow. Set bits correspond to the following errors on ADQ7-FWATD: • Bit 31 WFA software overflow (queue starvation). • Bit 30 Records discarded • Bit 29 Collection thread is not running • Bits 27-24, 11-8, 3-0 WFA hardware overflow.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

20.4.7 ATDRegisterWFABuffer()

```
virtual int ATDRegisterWFABuffer (
    unsigned int        channel,
    void *              buffer )
```

Register ATD WFA buffers.

Adds a buffer where ATD WFA can write data. Size of buffers must be the length previously set using ATDSetupWFA() or ATDSetupWFAAdvanced().

Parameters

channel Channel select, 1 to 4
buffer Pointer to buffer

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

20.4.8 ATDSetThresholdFilter()

```
virtual int ATDSetThresholdFilter (
    unsigned int      channel,
    unsigned int *    coefficients )
```

Setup ATD Threshold module filter.

Program the filter coefficients for the linear-phase FIR filter (Type-I, order 16) in the threshold module. The coefficient vector consists of the impulse response values in the following order: h(0), h(1), h(2), h(3), h(4), h(5), h(6), h(7), h(8), where h(8) is the point of symmetry. The coefficients are represented using a 16-bit 2's complement representation with 14 fractional bits, yielding a coefficient range of $[-2, 1.999938965]$. The coefficient values fed into this function are the codes $[-2^{15}, 2^{15}-1]$. For example, a filter coefficient value of 1.5 would be input as the code 24576 (0x6000).

Parameters

channel	Target channel, 1 to 4 (ADQ12/ADQ14), 1 to 2 (ADQ7)
coefficients	Array of filter coefficients. The first 9 values are expected to be the coefficients represented using a 16-bit fix-point representation with 14 fractional bits.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

20.4.9 ATDSetupThreshold()

```
virtual int ATDSetupThreshold (
    unsigned int      channel,
    int               threshold,
    int               baseline,
    unsigned int      polarity,
    unsigned int      bypass )
```

Setup ATD Threshold module.

Parameters

channel	Target channel, 1 to 4 (ADQ12/ADQ14), 1 to 2 (ADQ7)
threshold	Threshold level in ADC codes. [-32768, 32767]
baseline	Baseline level in ADC codes. [-32768, 32767]
polarity	Specify the threshold polarity ▪ 0: Positive (samples below the threshold are masked with the baseline) ▪ 1: Negative (samples above the threshold are masked with the baseline)
bypass	Bypass the threshold module.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

20.4.10 ATDSetupWFA()

```
virtual int ATDSetupWFA (
    unsigned int      record_length,
    unsigned int      nof_pretrig_samples,
    unsigned int      nof_triggerdelay_samples,
    unsigned int      nof_accumulations,
    unsigned int      nof_repeats )
```

Setup ATD Wave Form Averaging module.

Parameters

record_length	Length of record in samples.
nof_pretrig_samples	The number of samples to collect from before the trigger point
nof_triggerdelay_samples	The number of samples to wait before collecting data, after the trigger arrives
nof_accumulations	Number of accumulations to perform.

Parameters

nof_repeats Number of times to repeat the accumulation, creating a new record. A value of 4294967295 (0xFFFFFFFF) indicates infinite repeats.

Returns

1 for success, 0 for failure, -1 if the settings exceed the available DRAM on the device

Note

The record_length, nof_pretrig_samples and nof_triggerdelay_samples parameters must be a multiple of the parallel samples used:

- ADQ7: In 1ch@10GSPS mode, parallel samples are 32. In 2ch@5GSPS mode, parallel samples are 16.
- ADQ12-1X, ADQ12-2X: Parallel samples are 8
- ADQ12-2C, ADQ12-4C: Parallel samples are 4
- ADQ12-2A, ADQ12-4A: Parallel samples are 2
- ADQ14-1X, ADQ14-2X: Parallel samples are 8
- ADQ14-2C, ADQ14-4C: Parallel samples are 4
- ADQ14-2A, ADQ14-4A: Parallel samples are 2

At the successful return of ATDSetupWFA() any buffers registered with ATDRegisterWFABuffer() will be automatically unregistered.

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

20.4.11 ATDSetupWFAAdvanced()

```
virtual int ATDSetupWFAAdvanced (
    unsigned int    segment_length,
    unsigned int    segments_per_record,
    unsigned int    accumulations_per_batch,
    unsigned int    record_length,
    unsigned int    nof_accumulations,
    unsigned int    nof_pretrig_samples,
    unsigned int    nof_triggerdelay_samples,
    unsigned int    bypass )
```

Setup ATD Wave Form Averaging module using detailed parameters.

Parameters

segment_length Length in samples of the WFA segment, the minimum quanta of data used by the WFA module. Must be specified in multiples of 32.

Parameters

segments_per_record	Number of segments used to make up a record. The record length will be segment_length x segments_per_records long.
accumulations_per_batch	Number of records that will be stored in each DRAM bank.
record_length	Record length, must be set to segment_length x segments_per_records.
nof_accumulations	Number of accumulations to perform before emitting the result. Must be a multiple of 2.
nof_pretrig_samples	The number of samples to collect from before the trigger point
nof_triggerdelay_samples	The number of samples to wait before collecting data, after the trigger arrives
bypass	Bypass the WFA module. ▪ 0: Bypass disabled ▪ 1: Bypass enabled

Returns

1 for success, 0 otherwise

Note

At the successful return of ATDSetupWFAAdvanced() any buffers registered with ATDRegisterWFABuffer() will be automatically unregistered.

Valid for

ADQ12-FWATD, ADQ14-FWATD

20.4.12 ATDSetWFABufferFormat()

```
int ATDSetWFABufferFormat (
    enum
    ATDWFABufferFormatformat )
```

Set the format of the ATD data buffer.

Set the type of the returned buffers from ATDWaitForWFABuffer. The default is ATD_WFA_BUFFER_FORMAT_INT32 (int32_t).

Parameters

format	Buffer data format. See ATDWFABufferFormat for supported formats.

Returns

1 for success, 0 otherwise

Valid for

ADQ14-FWATD

20.4.13 ATDSetWFAInternalTimeout()

```
virtual int ATDSetWFAInternalTimeout (
    unsigned int    timeout_ms )
```

Set the default internal timeout setting in milliseconds.

Set the internal timeout (default is 1000ms). The internal timeout is used when the internal data collection thread is coming to an end. This is a one-time timeout occurring when either a fixed length acquisition has collected all the raw data needed or the user has requested a stop by calling ATDStopWFA().

Parameters

timeout_ms The number of ms to use for timeout.

Returns

1 for success, 0 otherwise

Valid for

ADQ7-FWATD

20.4.14 ATDSetWFAPartitionBoundaries()

```
virtual int ATDSetWFAPartitionBoundaries (
    unsigned int    partition_lower_bound,
    unsigned int    partition_upper_bound )
```

Set the upper and lower boundaries for the WFA workload partitioning.

This function specifies the boundaries used by the partitioning algorithm in ATDSetupWFA(). The lower bound must never be set lower than 50. The limit on the upper bound is determined by the maximum record length that the user wants to support. Constrained by the interval, the partitioning algorithm will choose the highest integer which yields the best internal goal function score. Given a maximum supported record length, the upper boundary is computed as partition_upper_bound = floor(1024^3 / (8/9 ∗ nof_channels ∗ max_record_length ∗ 2)) The values set by this function is applied in all later calls to e.g. ATDGetDeviceNofAccumulations() and ATDSetupWFA().

Parameters

partition_lower_bound The lower boundary for the WFA partitioning.

Parameters

partition_upper_bound
The upper boundary for the WFA partitioning.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD

See also

ATDGetWFAPartitionBoundaries() ATDSetWFAPartitionBoundariesDefault() ATDGetDeviceNofAccumulations()

20.4.15 ATDSetWFAPartitionBoundariesDefault()

```
virtual int ATDSetWFAPartitionBoundariesDefault ( )
```

Set the default upper and lower boundaries for the WFA workload partitioning.

Reset the WFA partition boundaries to their default values (supporting 1M samples on ADQ12-FWATD/ADQ14-FWATD).

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

20.4.16 ATDStartWFA()

```
virtual int ATDStartWFA (
    void **          target_buffers,
    unsigned char    channels_mask,
    unsigned int     blocking )
```

Start ATD WFA.

Parameters

target_buffers
Array of buffers where data will be stored or NULL if ATDRegisterWFABuffer() is used to register target buffers.
channels_mask
A bit field specifying which channels to enable streaming for. Bit 0 enables channel A, bit 1 channel B and so forth.

Parameters

blocking
Selects blocking or non-blocking mode. If set to 1 the function will not return until all of the requested data has been collected, accumulated and transferred to target_buffers. If set to 0 the function will return as soon as data collection has been started. After calling ATDStartWFA in non-blocking mode no other ADQAPI functions other than ATDRegisterWFABuffer(), ATDWaitForWFABuffer(), ATDGetWFAStatus(), ATDFlushWFA() and ADQ_ATDStopWFA() may be called before ATDWaitForWFACompletion() has been called.

Returns

1 for success, 0 otherwise

Note

When ATDStartWFA() is called with NULL as target_buffers argument, transfer buffers need to be registered in advance using ATDRegisterWFABuffer().

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

See also

ATDGetWFAStatus(), ATDWaitForWFACompletion(), ATDStopWFA()

20.4.17 ATDStopWFA()

```
virtual int ATDStopWFA (
    void                )
```

Stop ATD WFA.

Stops a data collection started by ATDStartWFA().

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

See also

ATDStartWFA()

20.4.18 ATDUpdateNofAccumulations()

```
virtual int ATDUpdateNofAccumulations (
    unsigned int        nof_accumulations )
```

Set the number of accumulations.

Update the number of accumulations. This function may only be called when the WFA is running. Refer to the ADQ7-FWATD User Guide (17-1957) for details on when the settings are updated. When an update has been requested, another change cannot occur until the incoming data reflects the requested changes.

Parameters

nof_accumulations The total number of accumulations

Returns

1 for success, 0 for error, -1 indicates an update is currently in progress

Valid for

ADQ7-FWATD

See also

ATDSetWFAPartitionBoundaries() ATDSetWFAPartitionBoundariesDefault() ATDGetWFAPartitionBoundaries()

20.4.19 ATDWaitForWFABuffer()

```
virtual int ATDWaitForWFABuffer (
    unsigned int      channel,
    void **           buffer,
    int               timeout )
```

Wait for ATD WFA buffer.

Waits for filled ATD WFA buffer previously registered using ATDRegisterWFABuffer().

Parameters

channel Channel select, 1 to 4
buffer Pointer to a pointer where the returned buffer pointer will be stored. Returns -1 if a streaming overflow has occurred, -2 if WFA streaming is not running and NULL NULL on timeout. The format of the buffer is controlled with ATDSetWFABufferFormat().
timeout Timeout in milliseconds, use 0 for infinite timeout, use -1 to return immediately if no buffer is available.

Returns

Status code:

- 1 if a filled buffer was successfully returned.
- 0 if streaming overflow has occurred or buffer wait timed out.

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

See also

ATDRegisterWFABuffer()

20.4.20 ATDWaitForWFACompletion()

```
virtual int ATDWaitForWFACompletion (
    void                )
```

Wait for ATD WFA completion.

Blocks execution until ATD WFA collection is completed.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWATD, ADQ14-FWATD, ADQ7-FWATD

See also

ATDStartWFA()

21 Pulse Detection

Functions

- int PDAutoTrig (unsigned int channel, int ∗detected_trigger_level, unsigned int ∗detected_arm_hystersis)

    Let the ADQ attempt to acquire settings for the level trigger.
- int PDClearHistogram (unsigned int histogram_type, unsigned int channel)

    Clear the histogram bins.
- int PDEnableLevelTrig (unsigned int enable)

    Enable the -FWPD level trigger.
- int PDEnableTriggerCoincidence (unsigned int enable)

    Enable the -FWPD trigger coincidence.
- int PDGetCharacterizationStatus (unsigned int channel, unsigned int ∗status)

    Get pulse characterization status.
- int PDGetGeneration (unsigned int ∗generation)

    Get the FWPD generation.
- int PDGetHistogramStatus (unsigned int ∗overflow_bin, unsigned int ∗underflow_bin, unsigned int ∗histogram_count, unsigned int ∗histogram_status, unsigned int histogram_type, unsigned int channel)

    Get histogram status.
- int PDGetLevelTrigStatus (unsigned int ∗status)

    Readout of the -FWPD level trigger status.
- int PDReadHistogram (unsigned int ∗data, unsigned int histogram_type, unsigned int channel)

Read histogram data.

- int PDResetTriggerCoincidence (void)

  Resets the -FWPD trigger coincidence module.
- int PDSetDataMux (unsigned int input_channel, unsigned int output_channel)

  Control the FWPD data multiplexer.
- int PDSetMinimumFrameLength (unsigned int channel, unsigned int minimum_frame_length)

  Set the minimum frame length.
- int PDSetupCharacterization (unsigned int channel, unsigned int collection_mode, unsigned int reduction_factor, unsigned int detection_window_length, unsigned int record_length, unsigned int padding_offset, unsigned int minimum_frame_length, unsigned int trigger_polarity, unsigned int trigger_mode, unsigned int padding_trigger_mode)

  Configure pulse characterization.
- int PDSetupHistogram (unsigned int offset, unsigned int scale, unsigned int histogram_type, unsigned int channel)

  Setup FWPD Histograms.
- int PDSetupLevelTrig (unsigned int channel, int trigger_level, int reset_hysteresis, int trigger_arm_hysteresis, int reset_arm_hysteresis, unsigned int trigger_polarity, unsigned int reset_polarity)

  Configure the -FWPD level trigger.
- int PDSetupMovingAverageBypass (unsigned int bypass, int reference_level)

  Enable bypass for the moving average filter.
- int PDSetupStreaming (unsigned char channels_mask)

  Configure the streaming mode for the -FWPD option.
- int PDSetupTiming (unsigned int channel, unsigned int nof_pretrigger_samples, unsigned int nof_moving_average_sample
  unsigned int moving_average_delay, unsigned int trailing_edge_window, unsigned int number_of_records, unsigned int record_variable_length)

  Configure the -FWPD timing.
- int PDSetupTriggerCoincidence (unsigned int channel, unsigned int window_length, unsigned int mask)
- int PDSetupTriggerCoincidence2 (unsigned int channel, unsigned int core_id, unsigned int enable)

  Assign and enable -FWPD coincidence core to channel.
- int PDSetupTriggerCoincidenceCore (unsigned int core_id, unsigned int window_length, unsigned char ∗expr_array, unsigned int mask)

  Configure the -FWPD trigger coincidence cores.

21.1 Detailed Description

These functions are used to configure devices with the -FWPD firmware option.

21.2 Function Documentation

21.2.1 PDAutoTrig()

```
virtual int PDAutoTrig (
    unsigned int      channel,
    int *             detected_trigger_level,
    unsigned int *    detected_arm_hysteresis )
```

Let the ADQ attempt to acquire settings for the level trigger.

This function returns the trigger level for which events are detected on channel A.

Parameters

channel The target channel. [1, 4]	
detected_trigger_level Suggested level trigger value. If 0, no signal could be acquired.	
detected_arm_hysteresis Suggested trigger- and reset arm hysteresis levels. If 0, no signal could be acquired.	

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.2 PDClearHistogram()

```
virtual int PDClearHistogram (
    unsigned int      histogram_type,
    unsigned int      channel )
```

Clear the histogram bins.

Set histogram bins to zero. The histogram overflow status bit will also be reset.

Parameters

histogram_type Specifies the histogram type: ▪ 0: Pulse width histogram ▪ 1: Peak value histogram	
channel The channel from which the status should be read. The channel index start at 1 (channel A).	

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.3 PDEnableLevelTrig()

```
virtual int PDEnableLevelTrig (
    unsigned int      enable )
```

Enable the -FWPD level trigger.

Enable the level trigger. Requires a valid -FWPD license key.

Parameters

enable	Value 1 to enable, 0 to disable.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.4 PDEnableTriggerCoincidence()

```
virtual int PDEnableTriggerCoincidence (
    unsigned int      enable )
```

Enable the -FWPD trigger coincidence.

This function enables the -FWPD trigger coincidence masking if enable is set to 1. By default the trigger coincidence masking is bypassed.

Parameters

enable	

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.5 PDGetCharacterizationStatus()

```
virtual int PDGetCharacterizationStatus (
    unsigned int      channel,
    unsigned int *    status )
```

Get pulse characterization status.

Read the status of the pulse characterization module for the target channel.

Parameters

channel Channel ID, starting at 1.
status For ADQ14: ▪ Bit 0: Metadata FIFO overflow ▪ Bit 1: Pulse width histogram FIFO overflow ▪ Bit 2: Pulse peak histogram FIFO overflow For ADQ7: ▪ Bit 0: Metadata buffer FIFO overflow ▪ Bit 1: Metadata packer overflow ▪ Bit 2: Metadata window discarded ▪ Bit 3: Time over threshold (arithmetic) overflow ▪ Bit 4: Metadata alignement error (internal error)

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.6 PDGetGeneration()

```
virtual int PDGetGeneration (
    unsigned int *    generation )
```

Get the FWPD generation.

Parameters

generation Output parameter with the firmware PD generation. Only valid if the return status is 1

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.7 PDGetHistogramStatus()

```
virtual int PDGetHistogramStatus (
    unsigned int *    overflow_bin,
    unsigned int *    underflow_bin,
    unsigned int *    histogram_count,
    unsigned int *    histogram_status,
    unsigned int      histogram_type,
    unsigned int      channel )
```

Get histogram status.

Reads the histogram status, histogram count, overflow and underflow bin. The output parameters are ignored if the pointer is NULL (0).

Parameters

overflow_bin	Read result of the underflow bin.
underflow_bin	Read result of the overflow bin.
histogram_count	Read result of the histogram count.
histogram_status	Read result of the histogram status. • Bit 0: Histogram FIFO empty • Bit 1: Histogram FIFO overflow
histogram_type	Specifies the histogram type: • 0: Pulse width histogram • 1: Peak value histogram
channel	The channel from which the status should be read. The channel index start at 1 (channel A).

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.8 PDGetLevelTrigStatus()

```
virtual int PDGetLevelTrigStatus (
    unsigned int *    status )
```

Readout of the -FWPD level trigger status.

Parameters

status	

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.9 PDReadHistogram()

```
virtual int PDReadHistogram (
    unsigned int *    data,
    unsigned int      histogram_type,
    unsigned int      channel )
```

Read histogram data.

Read histogram data bins. The data array has to be allocated by the user, and should be sufficiently large to fit all bins. Refer to the FWPD User Guide for the number of bins per histogram. The histogram overflow and underflow bins are read with PDGetHistogramStatus.

Parameters

data Pointer to the return data array. This should be large enough to hold all histogram bins.
histogram_type Specifies the histogram type: • 0: Pulse width histogram • 1: Peak value histogram
channel Channel ID. The index start at 1 (channel A)

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.10 PDResetTriggerCoincidence()

```
virtual int PDResetTriggerCoincidence (
    void              )
```

Resets the -FWPD trigger coincidence module.

Returns

1 for success, 0 otherwise

Valid for

ADQ14-FWPD

21.2.11 PDSetDataMux()

```
virtual int PDSetDataMux (
    unsigned int       input_channel,
    unsigned int       output_channel )
```

Control the FWPD data multiplexer.

Forward the data on the channel specified by input_channel to the channel specified by output_channel. The channel indices start at 1.

Parameters

input_channel	Input channel ID starting at 1 (channel A).
output_channel	Output channel ID starting at 1 (channel A).

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.12 PDSetMinimumFrameLength()

```
virtual int PDSetMinimumFrameLength (
    unsigned int       channel,
    unsigned int       minimum_frame_length )
```

Set the minimum frame length.

Set the minimum frame length used by the padding module. The minimum frame length may be changed while the acqusision is active.

Parameters

channel	The target channel. Index start at 1.
minimum_frame_length	The minimum frame length in samples. This has to be a multiple of the number of parallel samples used in the hardware. Note that the number of parallel samples may differ between the data and metadata channels.

Returns

1 for success, 0 otherwise

Valid for

ADQ7-FWPD

21.2.13 PDSetupCharacterization()

```
virtual int PDSetupCharacterization (
    unsigned int    channel,
    unsigned int    collection_mode,
    unsigned int    reduction_factor,
    unsigned int    detection_window_length,
    unsigned int    record_length,
    unsigned int    padding_offset,
    unsigned int    minimum_frame_length,
    unsigned int    trigger_polarity,
    unsigned int    trigger_mode,
    unsigned int    padding_trigger_mode )
```

Configure pulse characterization.

Configure the Pulse Characterization module according to specified settings. This function has to be called *after* PDSetupTiming on ADQ14-FWPD.

Parameters

channel	Channel ID, starting at 1.
collection_mode	Pulse characterization collection mode
reduction_factor	Every Nth record reduction factor

Parameters

detection_window_length Detection window length in number of samples.	
record_length Record length in number of samples.	
padding_offset Padding offset in number of samples.	
minimum_frame_length Minimum frame length in number of samples.	
trigger_polarity Trigger polarity, should match the pulse polarity. ▪ 0: Negative pulses ▪ 1: Positive pulses	
trigger_mode Trigger mode used for record acquisition	
padding_trigger_mode Trigger mode used for the padding grid.	

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.14 PDSetupHistogram()

```
virtual int PDSetupHistogram (
    unsigned int    offset,
    unsigned int    scale,
    unsigned int    histogram_type,
    unsigned int    channel )
```

Setup FWPD Histograms.

Configure the FWPD histogram.

Parameters

offset Offset used for mapping values to bins. See FWPD User Guide.	
scale Scale factor used for mapping values to bins. See FWPD User Guide.	

Parameters

histogram_type Specifies the histogram which the settings should be applied to ▪ 0: Pulse width histogram ▪ 1: Peak value histogram	
channel Channel ID. The index start at 1 (channel A)	

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.15 PDSetupLevelTrig()

```
virtual int PDSetupLevelTrig (
    unsigned int      channel,
    int               trigger_level,
    int               reset_hysteresis,
    int               trigger_arm_hysteresis,
    int               reset_arm_hysteresis,
    unsigned int      trigger_polarity,
    unsigned int      reset_polarity )
```

Configure the -FWPD level trigger.

Configure the data driven level trigger for a channel. The reset_polarity must have the opposite edge of the trigger_polarity.

Parameters

channel Channel identifier. Indexed from 1 and upwards.	
trigger_level Trigger level as a value in the range [-32768, 32767]	
reset_hysteresis Reset hysteresis (trigger polarity 1: negative offset from the trigger level, trigger polarity 0: positive offset from the trigger level)	
trigger_arm_hysteresis Trigger event rearm level (trigger polarity 1: negative offset from the trigger level, trigger polarity 0: positive offset from the trigger level)	

Parameters

reset_arm_hysteresis	Reset event rearm level (reset polarity 1: negative offset from the reset level, reset polarity 0: positive offset from the reset level)
trigger_polarity	Logic value, 1: Rising edge, 0: Falling edge.
reset_polarity	Logic value, 1: Rising edge, 0: Falling edge.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.16 PDSetupMovingAverageBypass()

```
virtual int PDSetupMovingAverageBypass (
    unsigned int        bypass,
    int                 reference_level )
```

Enable bypass for the moving average filter.

The purpose of this function is to allow the user to configure the moving average filter without using the filter output as a baseline for the level trigger. Instead, the value of reference_level is used as a baseline. This function does not need to be called for normal operation, i.e. using the MA filter output as the level trigger baseline.

Parameters

bypass	Set to 1 to activate bypass and trigger relative to the fixed value of reference_level, 0 to allow a moving baseline from the MA filter output.
reference_level	Specifies the constant level to use as a reference to the level trigger instead of the output from the MA filter.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.17 PDSetupStreaming()

```
virtual int PDSetupStreaming (
    unsigned char      channels_mask )
```

Configure the streaming mode for the -FWPD option.

Parameters

channels_mask	Channels mask, four bits.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.18 PDSetupTiming()

```
virtual int PDSetupTiming (
    unsigned int      channel,
    unsigned int      nof_pretrigger_samples,
    unsigned int      nof_moving_average_samples,
    unsigned int      moving_average_delay,
    unsigned int      trailing_edge_window,
    unsigned int      number_of_records,
    unsigned int      record_variable_length )
```

Configure the -FWPD timing.

Parameters

channel	Channel identifier. Indexed from 1 and upwards.
nof_pretrigger_samples	Set the number of pretrigger samples.
nof_moving_average_samples	Set the number of samples to be used for the moving average. Values [0, 100]
moving_average_delay	Specify the moving average delay. Range [0, 100 - nof_moving_average_samples]

Parameters

trailing_edge_window	Synonymous to the record length when record_variable_length is set to 0.
number_of_records	Number of records to collect.
record_variable_length	Activate variable length mode. Logic value.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD, ADQ7-FWPD

21.2.19 PDSetupTriggerCoincidence()

```
virtual int PDSetupTriggerCoincidence (
    unsigned int      channel,
    unsigned int      window_length,
    unsigned int      mask )
```

Parameters

channel	Channel identifier. Indexed from 1 and upwards.
window_length	Coincidence window length (in samples). Must be a multiple of 4 on -C devices and 8 on -X devices.
mask	The coincidence bitmask. It represents a logic OR expression which combines the channel triggers. E.g. a mask of 0b1010 (0xA) configured for channel 1 signifies that triggers are accepted on channel 1 if any of the channels 2 or 4 have triggered and the event on channel A occurs within the configured coincidence window.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.20 PDSetupTriggerCoincidence2()

```
virtual int PDSetupTriggerCoincidence2 (
```

```
    unsigned int      channel,
    unsigned int      core_id,
    unsigned int      enable )
```

Assign and enable -FWPD coincidence core to channel.

Parameters

channel	Channel identifier. Indexed from 1 and upwards.
core_id	Core identifier. Indexed from 0 and upwards.
enable	Enables use or bypass of coincidence core

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

21.2.21 PDSetupTriggerCoincidenceCore()

```
virtual int PDSetupTriggerCoincidenceCore (
    unsigned int      core_id,
    unsigned int      window_length,
    unsigned char *   expr_array,
    unsigned int      mask )
```

Configure the -FWPD trigger coincidence cores.

Parameters

core_id	Core identifier. Indexed from 0 and upwards.
window_length	Coincidence window length (in samples). Must be a multiple of 4 on -C devices and 8 on -X devices.
expr_array	Pointer to array to use when populating expression-memory.
mask	The coincidence start expression bit mask. Each bit in the mask corresponds to a channel where channel A is bit 0 and so on. The start expression is logic OR between all channels where the corresponding bit is set, i.e. '1'.

Returns

1 for success, 0 otherwise

Valid for

ADQ12-FWPD, ADQ14-FWPD

22 Software-defined radio

Functions

- int ForceResynchronizationSDR ()

  Synchronizes the phase of the mixer and decimation stages between all DDCs, as an alternative to synchronizing with respect to an external signal via SetupTimestampSync.
- int SetCrosspointSDR (unsigned int iqchannel, unsigned int mode)

  Selects the input data for the SDR down-converters.
- int SetEqualizerSDR (unsigned int iqchannel, float *coeffs1, float *coeffs2, unsigned int mode)

  Sets up the SDR channel equalizer filter, in either real-valued or complex-valued mode.
- int SetMixerFrequency (unsigned int iqchannel, double freq_hz)

  Sets the local oscillator frequency for the quadrature mixer of a firmware I/Q channel.
- int SetMixerPhase (unsigned int iqchannel, double radians)

  Sets the local oscillator phase for the quadrature mixer of a firmware I/Q channel.

22.1 Detailed Description

These functions are used to configure devices with the -FWSDR or -FW4DDC firmware options.

22.2 Function Documentation

22.2.1 ForceResynchronizationSDR()

```
virtual int ForceResynchronizationSDR ( )
```

Synchronizes the phase of the mixer and decimation stages between all DDCs, as an alternative to synchronizing with respect to an external signal via SetupTimestampSync.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

22.2.2 SetCrosspointSDR()

```
virtual int SetCrosspointSDR (
    unsigned int     iqchannel,
    unsigned int     mode )
```

Selects the input data for the SDR down-converters.

Parameters

iqchannel	The desired I/Q channel number, between 1 and the total number of I/Q channels
mode	- 0: Real valued data from channel A -> A + j*0 1: Real valued data from channel B -> B + j*0 2: Channel A and B as I/Q -> A/2 + j*B/2 3: Channel A and B as differential real-valued data: (A-B)/2 + j*0

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7-FW2DDC

22.2.3 SetEqualizerSDR()

```
virtual int SetEqualizerSDR (
    unsigned int    iqchannel,
    float *         coeffs1,
    float *         coeffs2,
    unsigned int    mode )
```

Sets up the SDR channel equalizer filter, in either real-valued or complex-valued mode.

Parameters

iqchannel	The desired I/Q channel number, between 1 and the total number of I/Q channels
coeffs1	Coefficient array 1 ▪ ADQ14: The array is expected to contain 15 coefficients. ▪ ADQ7: The array is expected to contain 5 coefficients.
coeffs2	Coefficient array 2 ▪ ADQ14: The array is expected to contain 15 coefficients. ▪ ADQ7: The array is expected to contain 5 coefficients.
mode	- 0: Bypassed (default) 1: Real-valued equalizer, coeffs1 is applied to I data, coeffs2 is applied to Q data, with no interaction 2: Complex-valued equalizer, the filter is (coeffs1 + i * coeffs2) and is applied to both I and Q data in a complex multiplication

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

22.2.4 SetMixerFrequency()

```
virtual int SetMixerFrequency (
    unsigned int    iqchannel,
    double          freq_hz )
```

Sets the local oscillator frequency for the quadrature mixer of a firmware I/Q channel.

Parameters

iqchannel	The desired I/Q channel number, between 1 and the total number of I/Q channels
freq_hz	The desired LO frequency in Hz

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

22.2.5 SetMixerPhase()

```
virtual int SetMixerPhase (
    unsigned int    iqchannel,
    double          radians )
```

Sets the local oscillator phase for the quadrature mixer of a firmware I/Q channel.

Parameters

iqchannel	The desired I/Q channel number, between 1 and the total number of I/Q channels
radians	The desired phase offset in radians

Returns

1 for successful operation and 0 for failure

Valid for

ADQ14-FWSDR, ADQ14-FW4DDC, ADQ7-FW2DDC

23 FPGA DNA Readout

Functions

- int GetDNA (unsigned int *dna)

  Returns FPGA DNA value.
- int ResetDNA (unsigned int assert)

  Reset FPGA DNA extraction.

23.1 Detailed Description

These functions are used to read the FPGA DNA, a value used to differentiate devices from each other.

23.2 Function Documentation

23.2.1 GetDNA()

```
virtual int GetDNA (
    unsigned int *    dna )
```

Returns FPGA DNA value.

Parameters

dna	A pointer to an array of four 32-bit unsigned integers to store the DNA / board unique ID.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

23.2.2 ResetDNA()

```
virtual int ResetDNA (
    unsigned int      assert )
```

Reset FPGA DNA extraction.

Parameters

assert	Set to 1 to reset the DNA extraction.

Returns

1 for success, 0 otherwise

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

24 Triggered Streaming

Functions

- unsigned int GetTriggeredStreamingHeaderSizeBytes ()

  Gets the number of bytes for the record header.
- unsigned int GetTriggeredStreamingRecords (unsigned int NofRecordsToRead, void ∗∗data_buf, void ∗header_buf, unsigned int ∗NofRecordsRead)

  Collects a number of Triggered Streaming-record.
- unsigned int GetTriggeredStreamingRecordSizeBytes ()

  Gets the number of bytes for the samples of each record.
- unsigned int HasTriggeredStreamingFunctionality ()

  Polls the ADQ to see if it has the Triggered Streaming functionality.
- unsigned int ParseTriggeredStreamingHeader (void ∗HeaderPtr, unsigned long long ∗Timestamp, unsigned int ∗Channel, unsigned int ∗ExtraAccuracy, int ∗RegisterValue, unsigned int ∗SerialNumber, unsigned int ∗RecordCounter)

  Reads a Triggered Streaming header and returns the values.
- unsigned int SetTriggeredStreamingHeaderRegister (char RegValue)

  Puts a user-defined 8-bit value in the Triggered Streaming record header.
- unsigned int SetTriggeredStreamingHeaderSerial (unsigned int SerialNumber)

  Overwrites the SerialNumber field in the Triggered Streaming header.
- unsigned int SetTriggeredStreamingTotalNofRecords (unsigned int MaxNofRecordsTotal)

  Sets an optional total number of Triggered Streaming records for all channels combined.
- unsigned int TriggeredStreamingArm ()

  Arms Triggered Streaming.
- unsigned int TriggeredStreamingArmV5 ()

  Arms Triggered Streaming for V5 ADQs.
- unsigned int TriggeredStreamingDisarm ()

  Disarms triggered streaming.
- unsigned int TriggeredStreamingDisarmV5 ()

  Disarms Triggered Streaming for V5 ADQs.
- unsigned int TriggeredStreamingGetNofRecordsCompleted (unsigned int ChannelsMask, unsigned int ∗NofRecordsCompleted)

  Gets the number of records that have been collected.
- unsigned int TriggeredStreamingGetStatus (unsigned int ∗InIdle, unsigned int ∗TriggerSkipped, unsigned int ∗Overflow)

Gets the status of the Triggered Streaming block.

- unsigned int TriggeredStreamingGetStatusV5 (unsigned char *ready, unsigned int *nofrecordscompleted, unsigned char *in_idle)

  Gets the status of Triggered Streaming for V5 ADQs.

- unsigned int TriggeredStreamingGetWaveformV5 (short *waveform_data_short)

  Collects one record for Triggered Streaming for V5 ADQs.

- unsigned int TriggeredStreamingOneChannelSetup (unsigned int SamplePerRecord, unsigned int NofPreTrigSamples, unsigned int NofTriggerDelaySamples, unsigned int ArmMode, unsigned int ReadOutSpeed, unsigned int Channel)

  Sets the parameters for one channel Triggered Streaming on V5 ADQs. Special firmware is needed. Please contact SP Devices for more information.

- unsigned int TriggeredStreamingParseDataStream (unsigned int samples_per_record, int *data_stream, int **data_target)

  Parses a buffer filled with a single record of Triggeres Streaming data.

- unsigned int TriggeredStreamingSetup (unsigned int NofRecords, unsigned int NofSamples, unsigned int NofPreTrigSamples, unsigned int NofTriggerDelaySamples, unsigned char ChannelsMask)

  Sets up a triggered streaming data acquisition.

24.1 Detailed Description

Triggered Streaming is a collection block for small collections with short rearm-time. The records are typically transferred to host using streaming mode.

24.2 Function Documentation

24.2.1 GetTriggeredStreamingHeaderSizeBytes()

```
virtual unsigned int GetTriggeredStreamingHeaderSizeBytes ( )
```

Gets the number of bytes for the record header.

This function may be convenient when allocating memory for storing the output.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

GetTriggeredStreamingRecordSizeBytes()

24.2.2 GetTriggeredStreamingRecords()

```
virtual unsigned int GetTriggeredStreamingRecords (
    unsigned int      NofRecordsToRead,
    void **           data_buf,
    void *            header_buf,
    unsigned int *    NofRecordsRead )
```

Collects a number of Triggered Streaming-record.

The data is retrieved from the ADQ and stored in user-allocated memory space. One record from one channel is transferred at a time, in the order of collection.

Parameters

NofRecordsToRead	Specifies the number of records to read from the ADQ. The records arrive in the order of collection.
data_buf	Pointer to different buffers, one for each channel of the device, where the actual data is output (without headers). If multiple records are collected from a channel, these are simply stored after each other in the buffer. The user is responsible for allocating these buffers.
header_buf	pointer to a buffer where the headers are stored in order. In level trigger mode this information is needed to determine from which buffer in **data_buf** to read the data, as the channels collect data individually. For other modes, the channel order is always A,B,C,D for ADQ412 if all channels are enabled. The user is responsible for allocating this buffer.
NofRecordsRead	Pointer to an integer where the function returns the number of records that were collected

Returns

1 for successful operation and 0 for failure

Note

When streaming data to host, this function assumes that the buffer size of the transfer buffers have been set using SetTransferBuffers(). The buffers sizes must follow two rules: The first is that they must be a multiple of the size of one record (including header). The second is that the total amount of data that is to be collected must be a multiple of the transfer buffer size.

If data is directed to the DRAM instead of the streaming interface, this function may be used after calling MemoryDump() and MemoryShadow().

Valid for

ADQ412

See also

TriggeredStreamingSetup(), GetTriggeredStreamingRecordSizeBytes(), GetTriggeredStreamingHeaderSizeBytes()

24.2.3 GetTriggeredStreamingRecordSizeBytes()

```
virtual unsigned int GetTriggeredStreamingRecordSizeBytes ( )
```

Gets the number of bytes for the samples of each record.

The size returned will not include the header. This function may be convenient when allocating memory for storing the output.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

GetTriggeredStreamingHeaderSizeBytes()

24.2.4 HasTriggeredStreamingFunctionality()

```
virtual unsigned int HasTriggeredStreamingFunctionality ( )
```

Polls the ADQ to see if it has the Triggered Streaming functionality.

Returns

1 if Triggered Streaming is available and 0 otherwise

Valid for

ADQ412, ADQ12, ADQ14, ADQ7

See also

TriggeredStreamingSetup()

24.2.5 ParseTriggeredStreamingHeader()

```
virtual unsigned int ParseTriggeredStreamingHeader (
    void *                  HeaderPtr,
    unsigned long long *    Timestamp,
    unsigned int *          Channel,
    unsigned int *          ExtraAccuracy,
    int *                   RegisterValue,
    unsigned int *          SerialNumber,
    unsigned int *          RecordCounter )
```

Reads a Triggered Streaming header and returns the values.

Parameters

HeaderPtr Pointer to the first byte of the header to parse	
Timestamp Pointer to where to return the value of the internal time counter stored in the header. Useful for knowing when a record was triggered. In interleaved mode for ADQ412, an increment of the counter means that one more sample passed. In non-interleaved mode, every two increments indicates that a sample passed.	
Channel Pointer to where to return the channel that was read. Channel 1 = A, 2 = B, 4 = C, and 8 = D.	
ExtraAccuracy Not used	
RegisterValue Pointer to where to return the register value that was stored in the header. The value may be specified using SetTriggeredStreamingHeaderRegister().	
SerialNumber Pointer to where to return the serial number of the board. The value may be overridden using SetTriggeredStreamingHeaderSerial().	
RecordCounter Pointer to where to return the record number stored in the header. This value starts at 0 and is then incremented for each record. If infinite streaming is used, this value will wrap back to 0 after 131072 records have been collected for the specific channel.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

GetTriggeredStreamingRecords()

24.2.6 SetTriggeredStreamingHeaderRegister()

```
virtual unsigned int SetTriggeredStreamingHeaderRegister (
    char              RegValue )
```

Puts a user-defined 8-bit value in the Triggered Streaming record header.

Useful for keeping track of different measurements and for debugging purposes.

Parameters

RegValue The value to set in the register

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

SetTriggeredStreamingHeaderSerial()

24.2.7 SetTriggeredStreamingHeaderSerial()

```
virtual unsigned int SetTriggeredStreamingHeaderSerial (
    unsigned int      SerialNumber )
```

Overwrites the SerialNumber field in the Triggered Streaming header.

Parameters

SerialNumber
The value to set in the register

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

SetTriggeredStreamingHeaderRegister()

24.2.8 SetTriggeredStreamingTotalNofRecords()

```
virtual unsigned int SetTriggeredStreamingTotalNofRecords (
    unsigned int      MaxNofRecordsTotal )
```

Sets an optional total number of Triggered Streaming records for all channels combined.

Parameters

MaxNofRecordsTotal
The maximum number of records to collect in total, regardless of from which channel the record came. If set to 0, there will be no limit. For technical reasons, one record per channel extra *may* be collected than the specified amount.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

TriggeredStreamingSetup()

24.2.9 TriggeredStreamingArm()

```
virtual unsigned int TriggeredStreamingArm ( )
```

Arms Triggered Streaming.

Must be called after TriggeredStreamingSetup() to start collection

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

TriggeredStreamingDisarm(), TriggeredStreamingSetup()

24.2.10 TriggeredStreamingArmV5()

```
virtual unsigned int TriggeredStreamingArmV5 ( )
```

Arms Triggered Streaming for V5 ADQs.

After this command is issued, triggers will be accepted. If automatic rearm and readout is turned on, readout will occur once a record is collected and a new record will be collected when readout is done.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

TriggeredStreamingDisarmV5(), TriggeredStreamingSetupV5()

24.2.11 TriggeredStreamingDisarm()

```
virtual unsigned int TriggeredStreamingDisarm ( )
```

Disarms triggered streaming.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

TriggeredStreamingArm()

24.2.12 TriggeredStreamingDisarmV5()

```
virtual unsigned int TriggeredStreamingDisarmV5 ( )
```

Disarms Triggered Streaming for V5 ADQs.

This function will disarm Triggered Streaming, and put the block in bypass mode so that other collection methods may be used.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

TriggeredStreamingArmV5(), TriggeredStreamingSetupV5()

24.2.13 TriggeredStreamingGetNofRecordsCompleted()

```
virtual unsigned int TriggeredStreamingGetNofRecordsCompleted (
    unsigned int      ChannelsMask,
    unsigned int *    NofRecordsCompleted )
```

Gets the number of records that have been collected.

The user may specify which channel(s) to ask.

Parameters

ChannelsMask Bit field to select which channel(s) to read from. Bit 0 selects channel A, bit 1 selects channel B, and so forth.
NofRecordsCompleted Pointer to where to store the result. All selected channels are added together.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

TriggeredStreamingGetStatus()

24.2.14 TriggeredStreamingGetStatus()

```
virtual unsigned int TriggeredStreamingGetStatus (
    unsigned int *    InIdle,
    unsigned int *    TriggerSkipped,
    unsigned int *    Overflow )
```

Gets the status of the Triggered Streaming block.

Parameters

InIdle Pointer to where to tell if the block is idle (1) or not (0)	
TriggerSkipped Pointer to a bit field, one per channel, that tells if a trigger was skipped by the particular channel due to the module not being able to buffer an extra record.	
Overflow Pointer to where to tell if an data overflow have occurred (1) or not (0). An overflow may cause data loss.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412

See also

TriggeredStreamingGetNofRecordsCompleted()

24.2.15 TriggeredStreamingGetStatusV5()

```
virtual unsigned int TriggeredStreamingGetStatusV5 (
    unsigned char *    ready,
    unsigned int *     nofrecordscompleted,
    unsigned char *    in_idle )
```

Gets the status of Triggered Streaming for V5 ADQs.

Parameters

ready Pointer to where to store whether data is available for readout. If this pointer is NULL, the value will not be written	

Parameters

nofrecordscompleted Pointer to where the to store the number of acquired records. If this pointer is NULL, the value will not be written	
in_idle Pointer to where to store whether the collection logic is idle. If this pointer is NULL, the value will not be written	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

TriggeredStreamingGetWaveformV5(), TriggeredStreamingSetupV5()

24.2.16 TriggeredStreamingGetWaveformV5()

```
virtual unsigned int TriggeredStreamingGetWaveformV5 (
    short *            waveform_data_short )
```

Collects one record for Triggered Streaming for V5 ADQs.

Readout will only succeed if data has been collected by the device. In manual rearm mode, TriggeredStreamingGetStatusV5() can be called to verify that data is available.

Parameters

waveform_data_short Pointer to where the output is to be stored. The data is stored as signed 16-bit integers, and the user is responsible that enough memory is allocated at this pointer.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

TriggeredStreamingGetStatusV5(), TriggeredStreamingSetupV5()

24.2.17 TriggeredStreamingOneChannelSetup()

```
virtual unsigned int TriggeredStreamingOneChannelSetup (
```

```
unsigned int        SamplePerRecord,
unsigned int        NofPreTrigSamples,
unsigned int        NofTriggerDelaySamples,
unsigned int        ArmMode,
unsigned int        ReadOutSpeed,
unsigned int        Channel )
```

Sets the parameters for one channel Triggered Streaming on V5 ADQs. Special firmware is needed. Please contact SP Devices for more information.

Triggered Streaming will stream a record of fixed size when it is triggered.

Parameters

SamplePerRecord

The number of samples to collect per record. If streaming over USB, one should choose a number of samples per record that gives a record size that is a multiple of 512 bytes. Because each sample is 2 bytes, this means that the record sizes should be chosen with 256 sample increments.

NofPreTrigSamples

The number of samples to collect from before the trigger point

NofTriggerDelaySamples

The number of samples to wait before collecting data, after the trigger arrives

ArmMode

Selects the rearm strategy:

- 0: Manual rearm and readout
  – Manual mode will collect a record, signal to the user that it has been collected, and then wait for the user to read it. Readout is done using TriggeredStreamingGetWaveformV5(). After the waveform has been read, the user must rearm the Triggered Streaming block by calling the function TriggeredStreamingArmV5() before collecting a new record.
- 1: Auto rearm and readout
  – Auto mode will directly push an acquired record through the streaming interface and rearm itself to collect the next record. The user must read the data fast enough, or the streaming interface will overflow. TriggeredStreamingArmV5() must be called once, before the collection starts.

ReadOutSpeed

Is the readout speed from the Triggered Streaming block:

- 0: Slow readout speed (use for USB)
- 1: Medium readout speed
- 2: Fast readout speed (requires four lanes PXIe or very short records)

Channel

Specifies from which channel the data will be streamed:

- 0: Not supported. For multi channels Triggered Streaming please use TriggeredStreamingSetupV5 instead
- 1: Channel A.
- 2: Channel B.

Returns

1 for successful operation and 0 for failure

Note

Manual rearm and readout will change the transfer settings of the device

Using Triggered Streaming excludes use of Packet Streaming, Waveform Averaging and MultiRecord

Valid for

ADQ214

See also

TriggeredStreamingArmV5(), TriggeredStreamingGetWaveformV5(), TriggeredStreamingDisarmV5()

24.2.18 TriggeredStreamingParseDataStream()

```
virtual unsigned int TriggeredStreamingParseDataStream (
    unsigned int     samples_per_record,
    int *            data_stream,
    int **           data_target )
```

Parses a buffer filled with a single record of Triggeres Streaming data.

The parsed data is stored in buffers provided by the user.

Parameters

samples_per_record	The number of samples per channel in the buffer data_stream
data_stream	Pointer to the buffer containing the data that will be parsed
data_target	Pointer table to buffers where the outputs will be stored, one buffer per channel. For example, data_target[0] is a pointer to the channel A output buffer, and data_target[1] is a pointer to the channel B output buffer.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

TriggeredStreamingGetWaveformV5()

24.2.19 TriggeredStreamingSetup()

```
virtual unsigned int TriggeredStreamingSetup (
```

```
unsigned int      NofRecords,
unsigned int      NofSamples,
unsigned int      NofPreTrigSamples,
unsigned int      NofTriggerDelaySamples,
unsigned char     ChannelsMask )
```

Sets up a triggered streaming data acquisition.

Triggering may be done either individually for each channel (with level trigger), or for all channels at the same time (other trigger modes). Data is output one channel at a time. Readout is easiest to do with the function GetTriggeredStreamingRecords().

Parameters

NofRecords The number of times to trigger a data collection for each active channel. For ADQ7/ADQ14/ADQ12: From 1 to 2^{31}-1 records. Setting the most significant bit (any value over 0x7FFFFFFF) will enable infinite number of records
NofSamples The number of samples to collect for each record. This value may be adjusted internally if necessary to align with the number of parallel samples per clock cycle for the product in use. This adjusted number of samples per record will also be what the record header will show. For ADQ412, which is a legacy product, triggered streaming only supports NofSamples of maximum 16384 samples in non-interleaved mode, and 32768 samples in interleaved mode. Furthermore, this record length must also be set in multiples of 32 samples, even though the number of parallel samples per clock cycle for ADQ412 is 8 in non-interleaved mode and 16 in interleaved mode.
NofPreTrigSamples The number of samples to collect from before the trigger arrives. When using pre-trigger, NofTriggerDelaySamples should be set to 0. The pre-trigger value is internally rounded downwards to a multiple of a constant factor. This factor is also often called "number of parallel samples". See note below.
NofTriggerDelaySamples NofTriggerDelaySamples is the number of samples to ignore after the trigger arrives. When this is used, NofPreTrigSamples should be 0. Trigger delay affects the rearm time in a negative way. For fast triggering, NofTriggerDelaySamples should be set to 0. The trigger delay value is internally rounded downwards to a multiple of a constant factor. This factor is also often called "number of parallel samples". See note below.
ChannelsMask Is used to specify from which channels to collect data. Bit 0 enables channel A, bit 1 channel B and so forth. For example on ADQ412, ChannelsMask = 0xF enables all channels while ChannelsMask = 0x3 enables only channel A and B.

Returns

1 for successful operation and 0 for failure

Note

Triggered streaming is typically with the streaming interface. This is done by calling SetStreamStatus(0x7) after this function. It is also possible to redirect the data to DRAM, by instead calling SetStreamStatus(0x9). The data can then later be dumped using MemoryDump() and then read using MemoryShadow() and GetTriggeredStreamingRecords()

Number of parallel samples on supported products:

- ADQ412: In non-interleaved mode number of parallel samples is 8. In interleaved mode, number of parallel samples 16.
- ADQ7: In 1ch@10GSPS mode, number of parallel samples are 32. In 2ch@5GSPS mode, parallel samples are 16.
- ADQ12-1X, ADQ12-2X: Number of parallel samples are 8
- ADQ12-2C, ADQ12-4C: Number of parallel samples are 4
- ADQ12-2A, ADQ12-4A: Number of parallel samples are 2
- ADQ14-1X, ADQ14-2X: Number of parallel samples are 8
- ADQ14-2C, ADQ14-4C: Number of parallel samples are 4
- ADQ14-2A, ADQ14-4A: Number of parallel samples are 2

Valid for

ADQ412, ADQ12, ADQ14, ADQ7

See also

TriggeredStreamingArm(), GetTriggeredStreamingRecords(), SetStreamStatus()

25 Packet Streaming

Functions

- unsigned int PacketStreamingArm ()

  Arms Packet Streaming.
- unsigned int PacketStreamingDisarm ()

  Disarms Packet Streaming.
- unsigned int PacketStreamingSetup (unsigned int PacketSizeSamples, unsigned int NofPreTrigSamples, unsigned int NofTriggerDelaySamples)

  Sets up the Packet Streaming block on the ADQ.

25.1 Detailed Description

Packet Streaming is a special streaming mode for ADQ214.

25.2 Function Documentation

25.2.1 PacketStreamingArm()

```
virtual unsigned int PacketStreamingArm ( )
```

Arms Packet Streaming.

Packets will be pushed on the data interface for each trigger

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

PacketStreamingDisarm(), PacketStreamingSetup()

25.2.2 PacketStreamingDisarm()

```
virtual unsigned int PacketStreamingDisarm ( )
```

Disarms Packet Streaming.

After disarming, the Packet Streaming block is inactive and bypassed.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ214

See also

PacketStreamingArm()

25.2.3 PacketStreamingSetup()

```
virtual unsigned int PacketStreamingSetup (
    unsigned int      PacketSizeSamples,
    unsigned int      NofPreTrigSamples,
    unsigned int      NofTriggerDelaySamples )
```

Sets up the Packet Streaming block on the ADQ.

Parameters

PacketSizeSamples The number of samples in each package
NofPreTrigSamples The number of samples to keep from before the trigger event
NofTriggerDelaySamples The number of samples to hold off after the trigger event

Returns

1 for successful operation and 0 for failure

Note

Packet size, pretrig and trigger delay may be chosen by 2 sample increments.

If streaming over USB is used, one should preferably choose a sample size of the waveform that equals a packet size of 512 bytes. Each averaged sample is 4 bytes, therefore sample sizes should be chosen as 128 sample increments. Also, best practice is to use SetTransferBuffers to complete each packet independently, i.e. set the transfer buffer size to the expected number of bytes of each packet.

The packet streaming block and waveform averaging block cannot be used at the same time.

Valid for

ADQ214

See also

PacketStreamingArm()

26 Disk streaming

Functions

- int DSUUpdateDoorbellAddress (unsigned int endpoint, unsigned int STE, uint64_t doorbell_address)
- int GetDSUParameters (uint64_t *BAR_addr, unsigned int *BAR_size_MiB, unsigned int *read_size_min, unsigned int *read_size_max, unsigned int *nof_endpoints_max, unsigned int *nof_dsu_ch)

26.1 Detailed Description

These functions are used to configure devices with the -FWDSU firmware option.

26.2 Function Documentation

26.2.1 DSUUpdateDoorbellAddress()

```
virtual int DSUUpdateDoorbellAddress (
    unsigned int     endpoint,
    unsigned int     STE,
    uint64_t         doorbell_address )
```

Update doorbell address for a DSU endpoint.

Parameters

endpoint	Endpoint index
STE	Sequence position to update, 0 or 1
doorbell_address	Address to doorbell register

Returns

1 for success, 0 otherwise

Valid for

ADQ7

26.2.2 GetDSUParameters()

```
virtual int GetDSUParameters (
    uint64_t *        BAR_addr,
    unsigned int *    BAR_size_MiB,
    unsigned int *    read_size_min,
    unsigned int *    read_size_max,
    unsigned int *    nof_endpoints_max,
    unsigned int *    nof_dsu_ch )
```

Get DSU parameters

Parameters

BAR_addr	DSU data BAR address
BAR_size_MiB	Effective BAR size
read_size_min	Minimum supported operation size
read_size_max	Maximum supported operation size
nof_endpoints_max	Maximum number of supported endpoints
nof_dsu_ch	Total number of data and metadata channels supported

Returns

1 for success, 0 otherwise

Valid for

ADQ7

27 ADQDSP

Functions

- int GetDSPData ()

  Starts and waits for transfer of data from the device to host.

- int GetDSPDataNowait ()

    Starts transfer of data from the device to host.
- unsigned int GetSendLength ()

    Gets the current transfer length.
- int InitTransfer ()

    Initiates and flush the data path.
- int SetSendLength (unsigned int length)

    Sets the size of data transfers.
- int TrigOutEn (unsigned int en)

    Enable or disable Trigger In to Trigger Out propagation.
- int WaitForPCIeDMAFinish (unsigned int length)

    Waits for transfer from the device to complete.
- unsigned int WriteToDataEP (unsigned int *pData, unsigned int length)

    Writes data to the device.

27.1 Detailed Description

Because ADQDSP does not collect data, it has several specific functions. These are documented here.

27.2 Function Documentation

27.2.1 GetDSPData()

```
virtual int GetDSPData ( )
```

Starts and waits for transfer of data from the device to host.

Data will be transferred from the internal memory buffers of the device to the host computer. This function will return when the transfer is completed.

Returns

1 for successful operation and 0 for failure

Valid for

ADQDSP

See also

GetDSPDataNowait(), SetSendLength(), InitTransfer()

27.2.2 GetDSPDataNowait()

```
virtual int GetDSPDataNowait ( )
```

Starts transfer of data from the device to host.

Data will be transferred from the internal memory buffers of the device to the host computer. This function will return before the transfer is completed. Use WaitForPCIeDMAFinish() before reading data to ensure that the transfer is complete

Returns

1 for successful operation and 0 for failure

Valid for

ADQDSP

See also

WaitForPCIeDMAFinish(), GetDSPData(), SetSendLength(), InitTransfer()

27.2.3 GetSendLength()

```
virtual unsigned int GetSendLength ( )
```

Gets the current transfer length.

Returns

The current send length. This is the value that was previously set by SetSendLength()

Valid for

ADQDSP

See also

SetSendLength()

27.2.4 InitTransfer()

```
virtual int InitTransfer ( )
```

Initiates and flush the data path.

Must be issued before any transfer of data to or from the unit is made.

Returns

1 for successful operation and 0 for failure

Valid for

ADQDSP

See also

GetDSPData(), GetDSPDataNowait()

27.2.5 SetSendLength()

```
virtual int SetSendLength (
    unsigned int      length )
```

Sets the size of data transfers.

The length is used by GetDSPData() and GetDSPDataNowait()

Parameters

length
The transfer length, given as the number of 32-bit words to transfer

Returns

1 for successful operation and 0 for failure

Valid for

ADQDSP

See also

GetDSPData(), GetDSPDataNowait()

27.2.6 TrigOutEn()

```
virtual int TrigOutEn (
    unsigned int      en )
```

Enable or disable Trigger In to Trigger Out propagation.

Parameters

en

Specifies whether the connection is enabled or not:

- 0: Disabled
- 1: Enabled

Returns

1 for successful operation and 0 for failure

Valid for

ADQDSP

27.2.7 WaitForPCIeDMAFinish()

```
virtual int WaitForPCIeDMAFinish (
    unsigned int      length )
```

Waits for transfer from the device to complete.

Parameters

length
Should be set to the same length as in SetSendLength()

Returns

1 for successful operation and 0 for failure

Valid for

ADQDSP

See also

GetDSPDataNowait(), SetSendLength()

27.2.8 WriteToDataEP()

```
virtual unsigned int WriteToDataEP (
    unsigned int *    pData,
    unsigned int      length )
```

Writes data to the device.

Parameters

pData	A Pointer to where the data is stored
length	the number of 32 bit words stored at pData. This length is not affected by SetSendLength()

Returns

1 for successful operation and 0 for failure

Valid for

ADQDSP, SDR14

28 Micro-TCA Specific

Functions

- unsigned int SetDirectionMLVDS (unsigned char direction)

  Sets the direction of the MTCA backplane LVDS pairs.
- unsigned int SetEthernetPll (unsigned short refdiv, unsigned char useref2, unsigned char a, unsigned short b, unsigned char p, unsigned char vcooutdiv, unsigned char eth10_outdiv, unsigned char eth1_outdiv)

  Sets the 10G and 1G Ethernet GTX clocks.
- unsigned int SetEthernetPllFreq (unsigned char eth10_freq, unsigned char eth1_freq)

  Sets the 10G and 1G Ethernet GTX clocks to predefined values.
- unsigned int SetPointToPointPll (unsigned short refdiv, unsigned char useref2, unsigned char a, unsigned short b, unsigned char p, unsigned char vcooutdiv, unsigned char pp_outdiv, unsigned char ppsync_outdiv)

  Sets the point-to-point interface GTX clock.
- unsigned int SetPointToPointPllFreq (unsigned char pp_freq)

Sets the point-to-point interface GTX clock to predefined values.

- int SetTriggerMaskMLVDS (unsigned char mask)

  Sets the mask for accepting triggers from MLVDS in MTCA.

28.1 Detailed Description

These functions are for units based on MTCA

28.2 Function Documentation

28.2.1 SetDirectionMLVDS()

```
virtual unsigned int SetDirectionMLVDS (
    unsigned char     direction )
```

Sets the direction of the MTCA backplane LVDS pairs.

The MTCA backplane contains eight LVDS pairs. This function may be used to set the direction of these.

Parameters

direction
A bit field where each bit that is 0 configures its pair as input and each bit that is 1 configures its pair as output. The bits and LVDS pairs are connected according to the list below: ▪ bit 0: R17 ▪ bit 1: T17 ▪ bit 2: R18 ▪ bit 3: T18 ▪ bit 4: R19 ▪ bit 5: T19 ▪ bit 6: R20 ▪ bit 7: T20

Returns

1 for successful operation and 0 for failure

Note

Only available for MTCA units

Valid for

ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQ14, ADQ7, ADQ8

28.2.2 SetEthernetPll()

```
virtual unsigned int SetEthernetPll (
```

```
unsigned short   refdiv,
unsigned char    useref2,
unsigned char    a,
unsigned short   b,
unsigned char    p,
unsigned char    vcooutdiv,
unsigned char    eth10_outdiv,
unsigned char    eth1_outdiv )
```

Sets the 10G and 1G Ethernet GTX clocks.

This function provides an advanced way to set the frequencies. The function SetEthernetPllFreq() may be used to set predefined values in a more simple manner. Please refer to the AD9517-1 PLL datasheet for more info on parameters and allowed values.

Parameters

refdiv	Reference divider, 0 to 16383 are valid values
useref2	Reference selector ▪ 0: 10MHz TCX0 ▪ 1: Output from the clock reference mux
a	VCO feedback parameter A, 0 to 31 are valid values
b	VCO feedback parameter B, 0 to 4095 are valid values
p	VCO feedback parameter P, 2,4,8,16 and 32 are valid values
vcooutdiv	VCO divider, 1 to 6 are valid values
eth10_outdiv	10G clock output divider, 0 to 32 are valid values
eth1_outdiv	1G clock output divider, 0 to 32 are valid values

Returns

1 for successful operation and 0 for failure

Note

Only available for MTCA units

Valid for

ADQ108, ADQ208, ADQ412, ADQ1600, SDR14

28.2.3 SetEthernetPllFreq()

```
virtual unsigned int SetEthernetPllFreq (
    unsigned char    eth10_freq,
    unsigned char    eth1_freq )
```

Sets the 10G and 1G Ethernet GTX clocks to predefined values.

This function provides a simple way to set the frequencies. For more direct control of the PLL, please refer to SetEthernetPll().

Parameters

eth10_freq Frequency value for the the 10G clock. Allowed values are: ETH10_FREQ_156_25MHZ (156.25 MHz) ETH10_FREQ_125MHZ (125 MHz)
eth1_freq Frequency value for the the 1G clock. Allowed values are: ETH1_FREQ_156_25MHZ (156.25 MHz) ETH1_FREQ_125MHZ (125 MHz)

Returns

1 for successful operation and 0 for failure

Note

Only available for MTCA units

Valid for

ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQ7

See also

SetEthernetPll()

28.2.4 SetPointToPointPll()

```
virtual unsigned int SetPointToPointPll (
    unsigned short   refdiv,
    unsigned char    useref2,
    unsigned char    a,
    unsigned short   b,
    unsigned char    p,
    unsigned char    vcooutdiv,
    unsigned char    pp_outdiv,
    unsigned char    ppsync_outdiv )
```

Sets the point-to-point interface GTX clock.

This function provides an advanced way to set the frequency. The function SetPointToPointPllFreq() may be used to set predefined values in a more simple manner. Please refer to the AD9517-1 PLL datasheet for more info on parameters and allowed values.

Parameters

refdiv	Reference divider, 0 to 16383 are valid values
useref2	Reference selector ▪ 0: 10MHz TCX0 ▪ 1: Output from the clock reference mux
a	VCO feedback parameter A, 0 to 31 are valid values
b	VCO feedback parameter B, 0 to 4095 are valid values
p	VCO feedback parameter P, 2,4,8,16 and 32 are valid values
vcooutdiv	VCO divider, 1 to 6 are valid values
pp_outdiv	Point-to-point output divider, 0 to 32 are valid values
ppsync_outdiv	Point-to-point synched clock for 1G Ethernet output divider, 0 to 32 are valid values

Returns

1 for successful operation and 0 for failure

Note

Only available for MTCA units

Valid for

ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQ14-MTCA

See also

SetPointToPointPllFreq()

28.2.5 SetPointToPointPllFreq()

```
virtual unsigned int SetPointToPointPllFreq (
    unsigned char      pp_freq )
```

Sets the point-to-point interface GTX clock to predefined values.

This function provides a simple way to set the frequency. For more direct control of the PLL, please refer to SetPointToPointPll().

Parameters

pp_freq
Frequency value for the point-to-point interface GTX clock. Allowed values are: PP_FREQ_330MHZ (330 MHZ) PP_FREQ_250MHZ (250 MHZ) PP_FREQ_156_25MHZ (156.25 MHZ) PP_FREQ_125MHZ (125 MHZ)

Returns

1 for successful operation and 0 for failure

Note

Only available for MTCA units

Valid for

ADQ108, ADQ208, ADQ412, ADQ1600, SDR14, ADQ14-MTCA, ADQ7-MTCA, ADQ8-MTCA

See also

SetPointToPointPll()

28.2.6 SetTriggerMaskMLVDS()

```
virtual int SetTriggerMaskMLVDS (
    unsigned char    mask )
```

Sets the mask for accepting triggers from MLVDS in MTCA.

Parameters

mask
Select which MLVDS inputs to trigger from (inputs are ORed together if several bits are set) The bits and LVDS pairs are connected according to the list below:

- bit 0: R17
- bit 1: T17
- bit 2: R18
- bit 3: T18
- bit 4: R19
- bit 5: T19
- bit 6: R20
- bit 7: T20

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ14, ADQ8

See also

SetDirectionMLVDS()

29 Peer-to-Peer Streaming

Functions

- unsigned int GetP2pSize (unsigned int channel)

  Gets the current peer-to-peer package size.
- unsigned int GetP2pStatus (unsigned int *pending, unsigned int channel)

  Gets the number of pending DMA transfers.
- unsigned long GetPhysicalAddress ()

  Get the physical DMA address of the unit.
- unsigned int SendDataDev2Dev (unsigned long PhysicalAddress, unsigned int channel, unsigned int options)

  Configures the device for transaction of one DMA package from one device to another.
- unsigned int SetP2pSize (unsigned int bytes, unsigned int channel)

  Sets the peer-to-peer package size.
- unsigned int SetupDMADev2GPUDDMA (unsigned int num_buffers, unsigned long long *physical_address_list, unsigned int *size_list)

  Configures the device for DMA transfer to GPU using nvidia GPUDirect.
- unsigned int SetupDMADev2GPUDGMA (unsigned int num_buffers, unsigned long long *physical_address_list, unsigned int *size_list)

  Configures the device for DMA transfer to GPU using DirectGMA.
- int SetUserTransferBuffers (uint32_t nof_buffers, size_t buffer_size, const uint64_t *const physical_address_list)

  Sets DMA to use user supplied transfer buffers.
- unsigned int WaitforGPUMarker (unsigned int *marker_list, unsigned int list_size, unsigned int marker, unsigned int timeout_ms)

  ADQ14: Wait for device to write the specified marker (greater or equal) when transfer data with peer-to-peer to GPU using nvidia GPUDirect.

29.1 Detailed Description

Peer-to-peer streaming may be used to stream data directly between units without intervention from the host.

29.2 Function Documentation

29.2.1 GetP2pSize()

```
virtual unsigned int GetP2pSize (
    unsigned int      channel )
```

Gets the current peer-to-peer package size.

Parameters

channel The DMA channel to get the value for

Returns

The current peer-to-peer package size, which was set by SetP2pSize()

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, ADQDSP,

See also

SetP2pSize()

29.2.2 GetP2pStatus()

```
virtual unsigned int GetP2pStatus (
    unsigned int *    pending,
    unsigned int      channel )
```

Gets the number of pending DMA transfers.

Parameters

pending Pointer to where the number of pending transfers is to be stored
channel The DMA channel to get the value from

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, ADQDSP,

See also

SendDataDev2Dev()

29.2.3 GetPhysicalAddress()

```
virtual unsigned long GetPhysicalAddress ( )
```

Get the physical DMA address of the unit.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, SDR14, ADQDSP

See also

GetDSPDataNowait()

29.2.4 SendDataDev2Dev()

```
virtual unsigned int SendDataDev2Dev (
    unsigned long    PhysicalAddress,
    unsigned int     channel,
    unsigned int     options )
```

Configures the device for transaction of one DMA package from one device to another.

This function may be called multipe times to enqueue DMA transfers before data is available in order to prepare the device.

Parameters

PhysicalAddress	Receivers address returned by GetPhysicalAddress()
channel	Senders DMA channel to be used
options	Set to 1 for constant address (recomended). Set to 0 for incrementing address.

Returns

1 for valid parameters. Retuning 1 does not garatee that the transacation is registerd since this command does not know if the DMA queue is full. The number of pending DMA transfers must be checked with GetP2pStatus().

Note

This command will not work on USB devices

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, ADQDSP

29.2.5 SetP2pSize()

```
virtual unsigned int SetP2pSize (
    unsigned int     bytes,
    unsigned int     channel )
```

Sets the peer-to-peer package size.

Parameters

bytes	The package size to set in bytes
channel	The DMA channel to to set the package size for

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, SDR14, ADQDSP,

See also

GetP2pSize()

29.2.6 SetupDMADev2GPUDDMA()

```
virtual unsigned int SetupDMADev2GPUDDMA (
    unsigned int        num_buffers,
    unsigned long
    long *              physical_address_list,
    unsigned int *      size_list )
```

Configures the device for DMA transfer to GPU using nvidia GPUDirect.

Parameters

num_buffers	Number of receiving buffers (including markers) setup on GPU.
physical_address_list	Array of physical addresses of receiving buffers, including marker buffers.
size_list	Array of receiving buffer sizes in bytes, including marker buffers that need to be set to 32 bytes.

Returns

1 on success or 0 on failure

Note

This command will only work with PCIe device under Linux.

Valid for

ADQ12, ADQ14-FWDT

29.2.7 SetupDMADev2GPUDGMA()

```
virtual unsigned int SetupDMADev2GPUDGMA (
    unsigned int            num_buffers,
    unsigned long long *    physical_address_list,
    unsigned int *          size_list )
```

Configures the device for DMA transfer to GPU using DirectGMA.

Parameters

num_buffers Number of receiving buffers (including markers) setup on GPU.	
physical_address_list Array of physical addresses of receiving buffers, including marker buffers.	
size_list Array of receiving buffer sizes in bytes, including marker buffers that need to be set to 32 bytes.	

Returns

1 on success or 0 on failure

Note

This command will not work on USB devices

Valid for

ADQ12, ADQ14 with PCIe gen3 option

29.2.8 SetUserTransferBuffers()

```
virtual int SetUserTransferBuffers (
    uint32_t                nof_buffers,
    size_t                  buffer_size,
    const uint64_t *const   physical_address_list )
```

Sets DMA to use user supplied transfer buffers.

Parameters

nof_buffers The number of buffers in physical_address_list	
buffer_size The buffer size in bytes, must be a multiple of 1024	

Parameters

physical_address_list A list of physical addresses to transfer buffers

Returns

1 on success, 0 on failure

Valid for

ADQ14, ADQ7

29.2.9 WaitforGPUMarker()

```
virtual unsigned int WaitforGPUMarker (
    unsigned int *    marker_list,
    unsigned int      list_size,
    unsigned int      marker,
    unsigned int      timeout_ms )
```

ADQ14: Wait for device to write the specified marker (greater or equal) when transfer data with peer-to-peer to GPU using nvidia GPUDirect.

Parameters

list_size ADQ14: Number of marker buffers setup on GPU
marker_list ADQ14: Array of marker values
marker marker value to wait for.
timeout_ms Time out time in ms to for wait.

Returns

1 on success, 0 on failure, ADQ14: 2 on timeout

Note

This command will only work with PCIe device under Linux.

Valid for

ADQ12, ADQ14-FWDT

30 Arbitrary Waveform Generator

Functions

- unsigned int AWGArm (unsigned int dacId)

  Arms the AWG.

- unsigned int AWGAutoRearm (unsigned int dacId, unsigned int enable)

  Turns auto-rearm of the AWG on or off.

- unsigned int AWGContinuous (unsigned int dacId, unsigned int enable)

  Turns continuous mode of the AWG on or off.

- unsigned int AWGDisarm (unsigned int dacId)

  Disarms the AWG.

- unsigned int AWGEnableSegments (unsigned int dacId, unsigned int enableSeg)
- unsigned int AWGPlaylistMode (unsigned int dacId, unsigned int mode)

  Sets the playlist mode.

- unsigned int AWGReset (unsigned int dacId)

  Resets the AWG controller.

- unsigned int AWGSegmentMalloc (unsigned int dacId, unsigned int segId, unsigned int length, unsigned char reallocate)

  Allocates segment space for an AWG segment.

- unsigned int AWGSetupTrigout (unsigned int dacId, unsigned int trigoutmode, unsigned int pulselength, unsigned int enableflags, unsigned int autorearm)

  Configures the way the AWG uses the PXIe trigger outputs.

- unsigned int AWGTrig (unsigned int dacId)

  Triggers the AWG.

- unsigned int AWGTrigMode (unsigned int dacId, unsigned int trigmode)

  Sets special trigger modes for the AWG.

- unsigned int AWGTrigoutArm (unsigned int dacId)

  Arms the trigger output of the specified AWG.

- unsigned int AWGWritePlaylist (unsigned int dacId, unsigned int NofPlaylistElements, unsigned int ∗index, unsigned int ∗write_mask, unsigned int ∗segId, unsigned int ∗NofLaps, unsigned int ∗nextIndex, unsigned int ∗triggerType, unsigned int ∗triggerLength, unsigned int ∗triggerPolarity, unsigned int ∗triggerSample, unsigned int ∗triggerULSignals)

  Writes one or more playlist items.

- unsigned int AWGWritePlaylistItem (unsigned int dacId, unsigned int index, unsigned int write_mask, unsigned int segId, unsigned int NofLaps, unsigned int nextIndex, unsigned int triggerType, unsigned int triggerLength, unsigned int triggerPolarity, unsigned int triggerSample, unsigned int triggerULSignals)

  Writes one playlist item.

- unsigned int AWGWriteSegment (unsigned int dacId, unsigned int segId, unsigned int enable, unsigned int NofLaps, unsigned int length, int *data)

    Writes a segment to the AWG memory.
- unsigned int AWGWriteSegments (unsigned int dacId, unsigned int NofSegs, unsigned int *segId, unsigned int *NofLaps, unsigned int *length, short int **data)

    Writes segments to the AWG memory.
- int SetDACNyquistBand (unsigned int dacId, unsigned int nyquistband)

    Allows optimization of DAC characteristics for operation in a specific nyquist band.

30.1 Detailed Description

The Arbitrary Waveform Generator (AWG) of the SDR14 is controlled using these functions.

30.2 Function Documentation

30.2.1 AWGArm()

```
virtual unsigned int AWGArm (
    unsigned int      dacId )
```

Arms the AWG.

This preloads the first set of data from the DRAM so that the AWG is ready to output data as soon as it is triggered.

Parameters

dacId	Selects the AWG/DAC (1 or 2)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGDisarm()

30.2.2 AWGAutoRearm()

```
virtual unsigned int AWGAutoRearm (
    unsigned int      dacId,
    unsigned int      enable )
```

Turns auto-rearm of the AWG on or off.

Auto-rearm mode will rearm the AWG immediately upon a finished readout cycle, to make it ready for a new trigger event. Allows the user to select which segments that are output before restarting.

Parameters

dacId	Selects the AWG/DAC (1 or 2)
enable	Selects whether the auto-rearm is enabled (enable = 1) or disabled (enable = 0)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGArm()

30.2.3 AWGContinuous()

```
virtual unsigned int AWGContinuous (
    unsigned int      dacId,
    unsigned int      enable )
```

Turns continuous mode of the AWG on or off.

If this mode is turned on, the AWG will start outputting data as soon as it is armed and triggered to start.

Parameters

dacId	Selects the AWG/DAC (1 or 2)
enable	Selects whether the continuous mode is enabled (enable = 1) or disabled (enable = 0)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGArm()

30.2.4 AWGDisarm()

```
virtual unsigned int AWGDisarm (
    unsigned int     dacId )
```

Disarms the AWG.

Turns of the AWG. A trigger event will not cause the AWG to output data once it is disarmed.

Parameters

dacId Selects the AWG/DAC (1 or 2)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGArm()

30.2.5 AWGEnableSegments()

```
virtual unsigned int AWGEnableSegments (
    unsigned int     dacId,
    unsigned int     enableSeg )
```

Allows the user to select which segments that are output before restarting.

Parameters

dacId Selects the AWG/DAC (1 or 2)
enableSeg The highest segment number to output. All segments up to and including this number are output.

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGWriteSegments()

30.2.6 AWGPlaylistMode()

```
virtual unsigned int AWGPlaylistMode (
    unsigned int    dacId,
    unsigned int    mode )
```

Sets the playlist mode.

Requires firmware support for the AWG Playlist mode.

Parameters

dacId Selects the AWG/DAC (1 or 2)
mode - 0 for no playlist and 1 for playlist activation.

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGWritePlaylist()

30.2.7 AWGReset()

```
virtual unsigned int AWGReset (
    unsigned int    dacId )
```

Resets the AWG controller.

The function resets the AWG controller to a known state

Parameters

dacId Sets the AWG to reset (for DAC 1 or 2)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGWriteSegment()

30.2.8 AWGSegmentMalloc()

```
virtual unsigned int AWGSegmentMalloc (
    unsigned int     dacId,
    unsigned int     segId,
    unsigned int     length,
    unsigned char    reallocate )
```

Allocates segment space for an AWG segment.

The function uses the end address of the preceding segment internally during allocation, so an allocation loop using this function should always go from segment 1 and upwards sequentially, never the other way around.

Parameters

dacId	Sets the AWG/DAC to allocate for (1 or 2)
segId	Selects the segment number to allocate for
length	The number of samples to be allocated to the segment selected by segId. Must be a multiple of 16.
reallocate	Parameter that can be used to reallocate the memory mapping of all the segments following the one that is being modified. This is useful if only a few segments are to be reallocated and the user desires the update of the remaining segments to be done automatically. If, however, every segment in the entire AWG is to be reallocated within a loop by the user, the reallocate parameter should be set to 0 in order to avoid wasting computations.

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGWriteSegment()

30.2.9 AWGSetupTrigout()

```
virtual unsigned int AWGSetupTrigout (
```

```
    unsigned int      dacId,
    unsigned int      trigoutmode,
    unsigned int      pulselength,
    unsigned int      enableflags,
    unsigned int      autorearm )
```

Configures the way the AWG uses the PXIe trigger outputs.

The AWG may be used to output a trigger signal to the PXIe backplane, trigger output connector or similar.

Parameters

dacId	Selects the AWG/DAC (1 or 2)
trigoutmode	Selects the mode: • 0: Off • 1: Pulse at the start of each segment • 2: Pulse at the end of each segment • 3 Use trigger in data (ignores pulselength and autorearm arguments) • 3 Use playlist trigger data (ignores pulselength and autorearm arguments)
pulselength	The trigout pulse length, in 5ns increments (200 MHz clock period)
enableflags	A bitfield, where each bit enables output to the following: • bit 0: Trigout connector • bit 1: PXIe port1 trigger output
autorearm	Sets auto rearm on or off • 0: autorearm off (requires manual rearm after every triggered trigout pulse) • 1: autorearm on

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

EnablePXIeTrigout()

30.2.10 AWGTrig()

```
virtual unsigned int AWGTrig (
    unsigned int      dacId )
```

Triggers the AWG.

This triggers the waveform sequencing for the specified channel.

Parameters

dacId	Selects the AWG/DAC (1 or 2) [dacId == 3 will send a trigger to both channels]

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGArm(), AWGDisarm()

30.2.11 AWGTrigMode()

```
virtual unsigned int AWGTrigMode (
    unsigned int    dacId,
    unsigned int    trigmode )
```

Sets special trigger modes for the AWG.

Parameters

dacId	Selects the AWG/DAC (1 or 2)
trigmode	Trigger mode selection. These inputs are available: • 0: Normal single-shot triggering. • 1: Requires trigger event before starting each segment lap. • 2:Seamless segment switching mode. This mode should be used in conjunction with infinite-laps programmed segments (see AWGWriteSegment description). Upon being triggered, the AWG will wait until the end of the current segment lap before seamlessly switching to the next segment. This allows the user to loop segments indefinitely, with the trigger acting as break for the loop, and without any junk data being output when the segment switch occurs.

Returns

1 for successful operation and 0 for failure

Note

If seamless mode is enabled during the very first trigger that starts the AWG, the AWG will immediately seamlessly skip to segment 2. For this reason, always trigger the AWG without seamless mode initially, and then enable it for subsequent triggers.

Valid for

SDR14

See also

AWGSetTriggerEnable()

30.2.12 AWGTrigoutArm()

```
virtual unsigned int AWGTrigoutArm (
    unsigned int      dacId )
```

Arms the trigger output of the specified AWG.

If the AWG is to be rearmed after having triggered, an AWGTrigoutDisarm() command must first be issued.

Parameters

dacId	Selects the AWG/DAC (1 or 2)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGTrigoutDisarm()

30.2.13 AWGWritePlaylist()

```
virtual unsigned int AWGWritePlaylist (
    unsigned int      dacId,
    unsigned int      NofPlaylistElements,
    unsigned int *    index,
    unsigned int *    write_mask,
    unsigned int *    segId,
    unsigned int *    NofLaps,
    unsigned int *    nextIndex,
    unsigned int *    triggerType,
    unsigned int *    triggerLength,
    unsigned int *    triggerPolarity,
    unsigned int *    triggerSample,
    unsigned int *    triggerULSignals )
```

Writes one or more playlist items.

Requires firmware support for the AWG Playlist mode.

Parameters

dacId Selects the AWG/DAC (1 or 2)	
NofPlaylistElements The number of playlist elements to transfer and describes the size of the pointer arguments. Failure to comply with pointer table lengths may cause read failures in the system.	
index A pointer table to 32-bit unsigned integers that specifies playlist indices for each table entry	
write_mask A pointer table to 32-bit unsigned integers that masks what parameters are written for this specific playlist element (15 writes all). Bit 0 = 1 to write segId information Bit 1 = 1 to write nextIndex information Bit 2 = 1 to write NofLaps information Bit 3 = 1 to write trigger information	
segId A pointer table to 32-bit unsigned integers that specifies the segment ID to play	
NofLaps A pointer table to 32-bit unsigned integers that specifies the number of laps for this playlist element By setting bit 31 high in the NofLaps values, the *infinite-laps mode* is enabled. This will repeat the corresponding segment until the AWG is disarmed or a special trigger mode forces a segment switch (see AWGTrigMode()).	
nextIndex A pointer table to 32-bit unsigned integers that specifies which playlist index to play next	
triggerType A pointer table to 32-bit unsigned integers that specifies trigger type (1 = no trigger, 2 = trigger once, 3 = trigger every lap)	
triggerLength A pointer table to 32-bit unsigned integers that specifies trigger duration in samples	
triggerPolarity A pointer table to 32-bit unsigned integers that specifies the polarity of the trigger	
triggerSample A pointer table to 32-bit unsigned integers that specifies at which sample the trigger should start	
triggerULSignals A pointer table to 32-bit unsigned integers that specifies if signals should be sent to user logic or not	

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGPlaylistMode()

30.2.14 AWGWritePlaylistItem()

```
virtual unsigned int AWGWritePlaylistItem (
    unsigned int      dacId,
    unsigned int      index,
    unsigned int      write_mask,
    unsigned int      segId,
    unsigned int      NofLaps,
    unsigned int      nextIndex,
    unsigned int      triggerType,
    unsigned int      triggerLength,
    unsigned int      triggerPolarity,
    unsigned int      triggerSample,
    unsigned int      triggerULSignals )
```

Writes one playlist item.

Requires firmware support for the AWG Playlist mode.

Parameters

dacId	Selects the AWG/DAC (1 or 2)
index	A 32-bit unsigned integers that specifies playlist index
write_mask	A 32-bit unsigned integers that masks what parameters are written for this specific playlist element (15 writes all). Bit 0 = 1 to write segId information Bit 1 = 1 to write nextIndex information Bit 2 = 1 to write NofLaps information Bit 3 = 1 to write trigger information
segId	A 32-bit unsigned integers that specifies the segment ID to play
NofLaps	A 32-bit unsigned integers that specifies the number of laps for this playlist element By setting bit 31 high in the NofLaps values, the *infinite-laps mode* is enabled. This will repeat the corresponding segment until the AWG is disarmed or a special trigger mode forces a segment switch (see AWGTrigMode()).
nextIndex	A 32-bit unsigned integers that specifies which playlist index to play next
triggerType	A 32-bit unsigned integers that specifies trigger type (1 = no trigger, 2 = trigger once, 3 = trigger every lap)
triggerLength	A 32-bit unsigned integers that specifies trigger duration in nanoseconds (5ns steps are available)

Parameters

triggerPolarity	A 32-bit unsigned integers that specifies the polarity of the trigger
triggerSample	A 32-bit unsigned integers that specifies at which sample the trigger should start
triggerULSignals	A 32-bit unsigned integers that specifies if signals should be sent to user logic or not

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGPlaylistMode()

30.2.15 AWGWriteSegment()

```
virtual unsigned int AWGWriteSegment (
    unsigned int     dacId,
    unsigned int     segId,
    unsigned int     enable,
    unsigned int     NofLaps,
    unsigned int     length,
    int *            data )
```

Writes a segment to the AWG memory.

The memory must first be allocated using AWGSegmentMalloc()

Parameters

dacId	Selects the AWG/DAC (1 or 2)
segId	Selects the segment number
enable	Deprecated. Use AWGEnableSegments() instead.
NofLaps	Sets the number of laps which the segment should be looped before the AWG continues to the next segment. By setting bit 31 high in this value, the *infinite-laps mode* is enabled. This will repeat the segment until the AWG is disarmed or a special trigger mode forces a segment switch (see AWGTrigMode()).

Parameters

length	The number of samples to write. Must be a multiple of 16.
data	Pointer to the data to write. If this is a NULL pointer, all other settings will be set but no new data will be written. Data is required to be 16-bit short integers (14-bit two's complement integers with range -8192 to 8191, where the two 2 MSB bits (bits 15-14) are used for special feature encoding otherwise zeroed). Min DAC code is represented by hexadecimal 0x2000 (-8192) and max DAC code represented by hexadecimal code 0x1FFF (8191)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGSegmentMalloc(), AWGWriteSegments(), AWGTrigMode()

30.2.16 AWGWriteSegments()

```
virtual unsigned int AWGWriteSegments (
    unsigned int      dacId,
    unsigned int      NofSegs,
    unsigned int *    segId,
    unsigned int *    NofLaps,
    unsigned int *    length,
    short int **      data )
```

Writes segments to the AWG memory.

The memory must first be allocated using AWGSegmentMalloc(). This function is faster than AWGWriteSegment(), especially when writing many small segments.

Parameters

dacId	Selects the AWG/DAC (1 or 2)
NofSegs	The number of segments to transfer and describes the size of the pointer arguments. Failure to comply with pointer table lengths may cause read failures in the system.
segId	A pointer table to 32-bit unsigned integers that specifies segment IDs for each table entry

Parameters

NofLaps	A pointer table to 32-bit unsigned integers that sets the number of laps which the segment should be looped before the AWG continues to the next segment. By setting bit 31 high in the NofLaps values, the *infinite-laps mode* is enabled. This will repeat the corresponding segment until the AWG is disarmed or a special trigger mode forces a segment switch (see AWGTrigMode()).
length	A pointer table to 32-bit unsigned integers that specifies segment length for each table entry. The data lengths must be multiples of 16 samples.
data	A pointer table of memory sections of 16-bit short integers (14-bit two's complement integers with range -8192 to 8191, where the two 2 MSB bits (bits 15-14) are used for special feature encoding otherwise zeroed). Min DAC code is represented by hexadecimal 0x2000 (-8192) and max DAC code represented by hexadecimal code 0x1FFF (8191)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14

See also

AWGSegmentMalloc(), AWGWriteSegment(), AWGTrigMode()

30.2.17 SetDACNyquistBand()

```
virtual int SetDACNyquistBand (
    unsigned int      dacId,
    unsigned int      nyquistband )
```

Allows optimization of DAC characteristics for operation in a specific nyquist band.

Parameters

dacId	Select the DAC channel to change nyquist band for (1 or 2)
nyquistband	Select nyquist band to optimize for (1 or 2)

Returns

1 for successful operation and 0 for failure

Valid for

SDR14TX

31 Digital Gain and Offset

Functions

- unsigned int GetGainAndOffset (unsigned char Channel, int ∗Gain, int ∗Offset)

  Gets the current digital gain and offset for an individual ADC core.
- unsigned int SetGainAndOffset (unsigned char Channel, int Gain, int Offset)

  Sets the digital gain and offset for an individual ADC core.

31.1 Detailed Description

These functions may be used to fine-tune the digital gain and offset settings on some units

31.2 Function Documentation

31.2.1 GetGainAndOffset()

```
virtual unsigned int GetGainAndOffset (
    unsigned char    Channel,
    int *            Gain,
    int *            Offset )
```

Gets the current digital gain and offset for an individual ADC core.

Parameters

Channel	The ADC core for which to get the Gain and Offset parameters. This parameter is not always equivalent to an analog input channel number, see note belowl
Gain	Pointer to where to store the gain value output
Offset	Pointer to where to store the offset value output

Returns

1 for successful operation and 0 for failure

Note

On some digitizer models, a single analog input is implemented by interleaving multiple ADC cores. Changing gain and offset for a single core, without also updating the values for other cores belonging to the same input, can therefore introduce gain/offset interleaving mismatch into the captured data.

The settings are relative to the factory calibration. To override this relativeness, set bit 7 of the channel argument to 1.

Valid for

ADQ114, ADQ112, ADQ214, ADQ212, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetGainAndOffset()

31.2.2 SetGainAndOffset()

```
virtual unsigned int SetGainAndOffset (
    unsigned char    Channel,
    int              Gain,
    int              Offset )
```

Sets the digital gain and offset for an individual ADC core.

The gain and offset block is located directly after the sampling circuit. This function can be used if it is desired to change factory calibrated values for this block.

Parameters

Channel The ADC core for which to set the Gain and Offset parameters. This parameter is not always equivalent to an analog input channel number, see note below.
Gain The gain value to set, normalized to 10 bits. A value of 1024 corresponds to unity gain. The allowed range is -32768 to 32767.
Offset The offset value to set. An offset value of 8 will changed the offset by 8 codes (multiplied by the gain setting). The allowed range is -32768 to 32767.

Returns

1 for successful operation and 0 for failure

Note

On some digitizer models, a single analog input is implemented by interleaving multiple ADC cores. Changing gain and offset for a single core, without also updating the values for other cores belonging to the same input, can therefore introduce gain/offset interleaving mismatch into the captured data.

The settings are relative to the factory calibration. To override this relativeness, set bit 7 of the channel argument to 1.

Valid for

ADQ114, ADQ112, ADQ214, ADQ212, ADQ1600, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

GetGainAndOffset()

32 Miscellaneous Functions

Functions

- int EnableTestPatternPulseGenerator (unsigned int channel, unsigned int enable)

  Enables the test pattern pulse generator.
- int EnableTestPatternPulseGeneratorOutput (unsigned int enable_bitmask)

  Enables output of the test pattern pulse generator on TRIG and/or SYNC connector.
- unsigned int GetNofBytesPerSample (unsigned int *bytes_per_sample)

  Gets the number of bytes needed to store each sample.
- void * GetPtrData (unsigned int channel)

  Gives a pointer for data access.
- unsigned int GetTransferTimeout (unsigned int *timeout)

  Reads out the current timeout setting for data transfers.
- unsigned int MemoryDump (unsigned int StartAddress, unsigned int EndAddress, unsigned char *buffer, unsigned int *bufctr, unsigned int transfersize)

  Transfers data from the ADQ:s DRAM without parsing.
- unsigned int MemoryDumpRecords (unsigned int StartRecord, unsigned int NofRecords, unsigned char *buffer, unsigned int *bufctr, unsigned int transfersize)

  Transfers data from the ADQ:s DRAM without parsing, but with respect to record alignments rather than raw addresses.
- unsigned int MemoryShadow (void *MemoryArea, unsigned int ByteSize)

  Sets the API to use a DRAM shadow for parsing data.
- unsigned int SetTestPatternConstant (int value)

  Sets a constant value for some of the test pattern modes.
- unsigned int SetTestPatternMode (int mode)

  Sets a specified test pattern mode.
- unsigned int SetTransferTimeout (unsigned int value)

  Sets the timeout for data transfers.
- int SetupTestPatternPulseGenerator (unsigned int channel, int baseline, int amplitude, unsigned int pulse_period, unsigned int pulse_width, unsigned int nof_pulses_in_burst, unsigned int nof_bursts, unsigned int burst_period, unsigned int mode)

  Sets up the test pattern pulse generator.
- int SetupTestPatternPulseGeneratorPRBS (unsigned int channel, unsigned int prbs_id, unsigned int seed, int offset, unsigned int scale_bits)

  Sets up the PRBS module for the test pattern pulse generator.

32.1 Detailed Description

These functions contains functions that are not part of a specific function block.

32.2 Function Documentation

32.2.1 EnableTestPatternPulseGenerator()

```
virtual int EnableTestPatternPulseGenerator (
    unsigned int      channel,
    unsigned int      enable )
```

Enables the test pattern pulse generator.

Parameters

channel	The target channel, indexed 1-4.
enable	1 to enable, 0 to disable.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

32.2.2 EnableTestPatternPulseGeneratorOutput()

```
virtual int EnableTestPatternPulseGeneratorOutput (
    unsigned int      enable_bitmask )
```

Enables output of the test pattern pulse generator on TRIG and/or SYNC connector.

Parameters

enable_bitmask

A bitmask specifying if the output is active or not.

- Bit 0: TRIG
- Bit 1: SYNC

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14

32.2.3 GetNofBytesPerSample()

```
virtual unsigned int GetNofBytesPerSample (
    unsigned int *    bytes_per_sample )
```

Gets the number of bytes needed to store each sample.

The sample size is affected by SetDataFormat().

Parameters

bytes_per_sample
Pointer to where the number bytes per sample is returned

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208,ADQ112, ADQ114, ADQ214, ADQDSP, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetDataFormat()

32.2.4 GetPtrData()

```
virtual void* GetPtrData (
    unsigned int      channel )
```

Gives a pointer for data access.

ADQ14, ADQ7: Gives a pointer to transfer buffer with the specified index. Others: Gives a pointer to the data array of a channel.

Parameters

channel
ADQ14, ADQ7: Buffer index to get pointer to. Others: Channel to get pointer to.

Returns

Pointer to the specified object.

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, ADQ112, ADQ114, ADQDSP, SDR14, ADQDSP, ADQ14, ADQ7

See also

CollectDataNextPage()

32.2.5 GetTransferTimeout()

```
virtual unsigned int GetTransferTimeout (
    unsigned int *    timeout )
```

Reads out the current timeout setting for data transfers.

Read out the current setting for timeout

Parameters

timeout	The timeout value, specified in milliseconds. 0 is infinite timeout.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTransferBuffers(),SetTransferTimeout()

32.2.6 MemoryDump()

```
virtual unsigned int MemoryDump (
    unsigned int      StartAddress,
    unsigned int      EndAddress,
    unsigned char *   buffer,
    unsigned int *    bufctr,
    unsigned int      transfersize )
```

Transfers data from the ADQ:s DRAM without parsing.

This will simply store the raw data in the PC RAM for later parsing. May be used to improve transfer rate. For MultiRecord and Triggered Streaming, MemoryShadow() can be used to make the parsing functions use the dumped data. when data can be parsed at a later time.

Parameters

StartAddress

The first address to read. The number of bits per address is:

- ADQ214, ADQ114, ADQ212, ADQ112: 128 bits
- SDR14: 256 bits
- Other: 512 bits Must be a multiple of 32

Parameters

EndAddress	The last address to read. The number of bits per address is: • ADQ214, ADQ114, ADQ212, ADQ112: 128 bits • SDR14: 256 bits • Other: 512 bits Must be a multiple of 32, minus one (for example, 63 is valid)
buffer	A pointer to where the output will be stored. The user is responsible for allocating this memory.
bufctr	A pointer to where the number of bytes collected will be stored
transfersize	Is the transfer size to use. Set to NULL to use the default value.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ112, ADQ114, ADQ214, SDR14, ADQ1600, ADQ208, ADQ12, ADQ14

See also

MemoryShadow(), MultiRecordSetupGP()

32.2.7 MemoryDumpRecords()

```
virtual unsigned int MemoryDumpRecords (
    unsigned int       StartRecord,
    unsigned int       NofRecords,
    unsigned char *    buffer,
    unsigned int *     bufctr,
    unsigned int       transfersize )
```

Transfers data from the ADQ:s DRAM without parsing, but with respect to record alignments rather than raw addresses.

This will simply store the raw data in the PC RAM for later parsing. May be used to improve transfer rate. For MultiRecord and Triggered Streaming, MemoryShadow() can be used to make the parsing functions use the dumped data. when data can be parsed at a later time. Required size of buffer (shadow RAM) is obtained by calling MultiRecordSetupGP() This function will return an error if MultiRecordSetupGP() has not been called in advance.

Parameters

StartRecord	The first record to include in raw read.

Parameters

NofRecords	The number of records to read
buffer	A pointer to where the output will be stored. The user is responsible for allocating this memory.
bufctr	A pointer to where the number of bytes collected will be stored
transfersize	Is the transfer size to use. Set to NULL to use the default value.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ112, ADQ114, ADQ214, SDR14, ADQ1600, ADQ208

See also

MemoryShadow(), MultiRecordSetupGP()

32.2.8 MemoryShadow()

```
virtual unsigned int MemoryShadow (
    void *          MemoryArea,
    unsigned int    ByteSize )
```

Sets the API to use a DRAM shadow for parsing data.

The shadow is a copy of the ADQ:s DRAM contents in the PC RAM. This is used together with MemoryDump() to separate the tasks of transfer and parsing for higher transfer rates, where parsing is possible to perform offline.

Parameters

MemoryArea	MemoryArea pointer to memory area with **ByteSize** allocated bytes. The user is responsible for correct allocation/deallocation of this area. If MemoryArea is NULL, the shadow function is deactivated.
ByteSize	The number of bytes that should be read from the memory area

Returns

1 for successful operation and 0 for failure

Note

To retrieve product/settings dependent sizes to know which DRAM addresses to read, you may use the MultiRecordSetupGP function. To parse the data at a later stage use this MemoryShadow() of the API, together with GetData()

This function is typically used for MultiRecord, but may also be used when if data from Triggered Streaming has been redirected to DRAM and dumped with MemoryDump.

Valid for

ADQ412, ADQ212, ADQ108, ADQ112, ADQ114, ADQ214, SDR14, ADQ1600, ADQ208

See also

MemoryDump(), GetData(), GetTriggeredStreamingRecords()

32.2.9 SetTestPatternConstant()

```
virtual unsigned int SetTestPatternConstant (
    int                 value )
```

Sets a constant value for some of the test pattern modes.

Parameters

value	The constant value to set

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ112, ADQ114, ADQ1600, ADQ208, ADQ212, ADQ214, ADQ412, SDR14

See also

SetTestPatternMode()

32.2.10 SetTestPatternMode()

```
virtual unsigned int SetTestPatternMode (
    int                 mode )
```

Sets a specified test pattern mode.

Parameters

mode	The selected mode. One of these values are available: ▪ 0: Normal operation mode (direct data) ▪ 1: Test mode with user constant output (ADQ214,ADQ114 only) ▪ 2: Count upwards ▪ 3: Count downwards ▪ 4: Count alternating upwards and downwards ▪ 5 to 6: Reserved ▪ 7: Mode for merging GPIO with data (unpacked 16-bit modes for ADQ214, ADQ114 only)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ108, ADQ112, ADQ114, ADQ1600, ADQ208, ADQ212, ADQ214, ADQ412, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTestPatternConstant()

32.2.11 SetTransferTimeout()

```
virtual unsigned int SetTransferTimeout (
    unsigned int      value )
```

Sets the timeout for data transfers.

This is used in situations where certain data amounts are expected over the streaming interface at certain update rates. This value should always be significantly higher than the expected data rate, to avoid problems with the communication link.

Parameters

value	The timeout value, specified in milliseconds. Set to 0 for infinite timeout. Default is: ▪ ADQ14: 60000 ms ▪ ADQ12: 60000 ms ▪ ADQ7: Infinite ▪ Others: 1000 ms

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQ1600, SDR14, ADQDSP, ADQ12, ADQ14, ADQ7, ADQ8

See also

SetTransferBuffers(),GetTransferTimeout()

32.2.12 SetupTestPatternPulseGenerator()

```
virtual int SetupTestPatternPulseGenerator (
    unsigned int    channel,
    int             baseline,
    int             amplitude,
    unsigned int    pulse_period,
    unsigned int    pulse_width,
    unsigned int    nof_pulses_in_burst,
    unsigned int    nof_bursts,
    unsigned int    burst_period,
    unsigned int    mode )
```

Sets up the test pattern pulse generator.

Parameters

channel	The target channel, indexed 1-4.
baseline	The pulse resting value.
amplitude	The pulse 'high' value, offset from the baseline.
pulse_period	The pulse period in samples.
pulse_width	The pulse width in samples (i.e. the area where the value is baseline+amplitude).
nof_pulses_in_burst	The number of pulses in a burst, if left zero, the burst mode is disabled.
nof_bursts	The number of bursts, if left zero, the burst pattern wil keep repeating.
burst_period	The burst period in number of samples, must be greater than the pulse period. If left zero, the burst mode is disabled.

Parameters

mode The pulse generator mode. One of these values are available: ▪ 0: Bypassed ▪ 1: Regular pulse output. ▪ 2: PRBS width mode. ▪ 3: PRBS amplitude mode. ▪ 4: PRBS width & amplitude mode. ▪ Bit 3: Activate trigger sensing mode. ▪ Bit 4: Activate PRBS signal noise.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

32.2.13 SetupTestPatternPulseGeneratorPRBS()

```
virtual int SetupTestPatternPulseGeneratorPRBS (
    unsigned int    channel,
    unsigned int    prbs_id,
    unsigned int    seed,
    int             offset,
    unsigned int    scale_bits )
```

Sets up the PRBS module for the test pattern pulse generator.

Parameters

channel The target channel, indexing starts at 1.
prbs_id The target PRBS within the channel. Valid ID are: ▪ 0: PRBS applied as the pulse width. ▪ 1: PRBS applied as the pulse amplitude. ▪ 2: PRBS applied as noise to the signal.
seed The seed value for the PRBS. Valid range is [0,16383].
offset The offset value, applied after the scaling. Valid range is [-32768, 32767].
scale_bits Number of right shifts to perform. Valid range is [0, 15].

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

33 DevKit Functions

Functions

- unsigned int BypassUserLogic (unsigned int ul_target, unsigned int bypass)

  Bypasses curren user logic design.
- unsigned int EnableUseOfUserHeaders (unsigned int mode, unsigned int api_value)

  Enables connection from DevKit or API for the User ID header field in user logic.
- int EnableUserLogicFilter (unsigned int channel, unsigned int enable)

  Enable filter located in User Logic 1.
- const char * GetUserLogicPartNumber ()

  Gets the part number of the user logic block on the firmware.
- int ReadBlockUserRegister (int ul_target, uint32_t start_addr, uint32_t *data, uint32_t num_bytes, uint32_t options)

  Read a set of 32-bit registers from the ADQ DevKit user logic.
- int ReadUserRegister (int ul_target, uint32_t regnum, uint32_t *retval)

  Reads a 32-bit register in the ADQ DevKit.
- int ResetUserLogicFilter (unsigned int channel)

  Reset filter located in User Logic 1.
- int SetUserLogicFilter (unsigned int channel, void *coefficients, unsigned int length, unsigned int format, unsigned int rounding_method)

  Set filter coefficients for the FIR filter located in User Logic 1.
- int WriteBlockUserRegister (int ul_target, uint32_t start_addr, uint32_t *data, uint32_t num_bytes, uint32_t options)

  Write a set of 32-bit registers in the ADQ DevKit user logic.
- int WriteUserRegister (int ul_target, uint32_t regnum, uint32_t mask, uint32_t data, uint32_t *retval)

  Write a 32-bit register in the ADQ DevKit.

33.1 Detailed Description

These functions are used to interface the user logic block for customers of the ADQ DevKit.

33.2 Function Documentation

33.2.1 BypassUserLogic()

```
virtual unsigned int BypassUserLogic (
    unsigned int      ul_target,
    unsigned int      bypass )
```

Bypasses curren user logic design.

Parameters

ul_target
Select between different user logic blocks when there exists several on the digitizer. Different settings may be used on different units:
▪ V6 family (ADQ412/ADQ108/ADQ208/ADQ1600/ADQDSP/SDR14)
  – 0: Main FPGA user logic
▪ ADQ12 / ADQ14 / ADQ7 / ADQ8
  – 1: User logic 1
  – 2: User logic 2	

bypass Bypass setting ▪ 0: Do not bypass ▪ 1: Bypassed	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ108, ADQ208, ADQDSP, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

33.2.2 EnableUseOfUserHeaders()

```
virtual unsigned int EnableUseOfUserHeaders (
    unsigned int      mode,
    unsigned int      api_value )
```

Enables connection from DevKit or API for the User ID header field in user logic.

Parameters

mode User ID mode ▪ 0: Take User ID from User Logic 2 ▪ 1: Take User ID from inserted api_value
api_value 8-bit User ID value (ignored when mode == 0)

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14

33.2.3 EnableUserLogicFilter()

```
virtual int EnableUserLogicFilter (
    unsigned int      channel,
    unsigned int      enable )
```

Enable filter located in User Logic 1.

Parameters

channel Channel ID, indexed from 1 and upwards.
enable Set to 1 to enable, 0 to bypass.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ8, ADQ14

33.2.4 GetUserLogicPartNumber()

```
virtual const char* GetUserLogicPartNumber ( )
```

Gets the part number of the user logic block on the firmware.

This part number may be modified from inside the DevKit, using either the set_userlogicpartnumber command while building the DevKit, or by modifying the assignment statements to the registers in the user logic module. This allows the DevKit user to keep track of different firmware types and revisions.

Returns

A NULL-terminated string, consisting of three three-digit numbers followed by a revision letter. For example, 400-013-011-A.

Note

Older firmware revisions do not contain part number registers and will always be read out as 000-000-000-A.

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQDSP, SDR14

See also

GetNGCPartNumber()

33.2.5 ReadBlockUserRegister()

```
virtual int ReadBlockUserRegister (
    int          ul_target,
    uint32_t     start_addr,
    uint32_t *   data,
    uint32_t     num_bytes,
    uint32_t     options )
```

Read a set of 32-bit registers from the ADQ DevKit user logic.

Performs a read of several values at once from the 32-bit user logic input registers.

Parameters

ul_target
Select between different user logic blocks when there exists several on the digitizer. Different settings may be used on different units:
• ADQ12 / ADQ14 / ADQ7
  – 1: User logic 1
  – 2: User logic 2	

start_addr Specifies which register to start reading at (from 0 and up)	
data Pointer for storing the read data.	
num_bytes Data size in bytes (4 x number of registers).	
options • 0: Do not increment address. All values are read from start_addr. This is useful for FIFO interfaces. • 1: Increment address. Used for registers and RAM intefaces.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

WriteBlockUserRegister()

33.2.6 ReadUserRegister()

```
virtual int ReadUserRegister (
    int          ul_target,
    uint32_t     regnum,
    uint32_t *   retval )
```

Reads a 32-bit register in the ADQ DevKit.

Parameters

ul_target
Select between different user logic blocks when there exists several on the digitizer. Different settings may be used on different units:
▪ V6 family (ADQ412/ADQ108/ADQ208/ADQ1600/ADQDSP/SDR14)
  – 0: Main FPGA user logic
▪ V5 family (ADQ214/ADQ114/ADQ212/ADQ112)
  – 0: Alg FPGA user logic
  – 1: Comm FPGA user logic
▪ ADQ12 / ADQ14 / ADQ7 / ADQ8
  – 1: User logic 1
  – 2: User logic 2	

regnum Specifies which register to read (from 0 and up)	
retval Pointer to where the function will store the read result	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQDSP, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

WriteUserRegister()

33.2.7 ResetUserLogicFilter()

```
virtual int ResetUserLogicFilter (
    unsigned int   channel )
```

Reset filter located in User Logic 1.

Resetting the filter will reload the default coefficient set.

Parameters

channel Channel ID, indexed from 1 and upwards.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ8, ADQ14

33.2.8 SetUserLogicFilter()

```
virtual int SetUserLogicFilter (
    unsigned int     channel,
    void *           coefficients,
    unsigned int     length,
    unsigned int     format,
    unsigned int     rounding_method )
```

Set filter coefficients for the FIR filter located in User Logic 1.

Parameters

channel Channel ID, indexed from 1 and upwards.
coefficients Array of filter coefficients. The array is interpreted as either an array of unsigned integers or as an array of floating point numbers, according to the value of `format`. In the case of floating point numbers, the coefficients are subject to rounding, specified by `rounding_method`.
length The length of the filter coefficient array. The value is checked against the number of coefficients
format The format of the coefficent array. Valid values are: ▪ 0: 32-bit unsigned integers. By default, the values are interpreted as fixed point numbers using a 16-bit 2's complement representation with 14 fractional bits. ▪ 1: Single-precision floating point numbers. The values are subject to rounding.
rounding_method The rounding method used when the format is set to single-precision floating point numbers. Valid values are: ▪ 0: Round to nearest, tie away from zero. ▪ 1: Round to nearest, tie towards zero. ▪ 2: Round to nearest, tie to even.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7, ADQ8, ADQ14

33.2.9 WriteBlockUserRegister()

```
virtual int WriteBlockUserRegister (
    int             ul_target,
    uint32_t        start_addr,
    uint32_t *      data,
    uint32_t        num_bytes,
    uint32_t        options )
```

Write a set of 32-bit registers in the ADQ DevKit user logic.

Performs a write of several values at once to the 32-bit user logic input registers.

Parameters

ul_target
Select between different user logic blocks when there exists several on the digitizer. Different settings may be used on different units:
• ADQ12 / ADQ14 / ADQ7
  – 1: User logic 1
  – 2: User logic 2	

start_addr Specifies which register to start writing at (from 0 and up)	
data Pointer to write data.	
num_bytes Data size in bytes (4 x number of registers).	
options • 0: Do not increment address. All values are written to start_addr. This is useful for FIFO interfaces. • 1: Increment address. Used for registers and RAM interfaces.	

Returns

1 for successful operation and 0 for failure

Valid for

ADQ12, ADQ14, ADQ7, ADQ8

See also

ReadBlockUserRegister()

33.2.10 WriteUserRegister()

```
virtual int WriteUserRegister (
    int            ul_target,
    uint32_t       regnum,
    uint32_t       mask,
    uint32_t       data,
    uint32_t *     retval )
```

Write a 32-bit register in the ADQ DevKit.

Performs a masked write of a value to one of the 32-bit user logic input registers. Also re-reads the data and returns it for user validation.

Parameters

ul_target	Select between different user logic blocks when there exists several on the digitizer. Different settings may be used on different units:
	▪ V6 family (ADQ412/ADQ108/ADQ208/ADQ1600/ADQDSP/SDR14)
	  – 0: Main FPGA user logic
	▪ V5 family (ADQ214/ADQ114/ADQ212/ADQ112)
	  – 0: Alg FPGA user logic
	  – 1: Comm FPGA user logic
	▪ ADQ12 / ADQ14 / ADQ7 / ADQ8
	  – 1: User logic 1
	  – 2: User logic 2

regnum	Specifies which register to write (from 0 and up)
mask	Performs a negative mask, i.e. only the bits that are zero in the mask will be written. You can use a mask of 0xFFFFFFFF to simply check the current value of an input register without overwriting it.
data	The data to write. The data will be first be masked by the mask parameter.
retval	Pointer to where the function will store its re-read result from of the register

Returns

1 for successful operation and 0 for failure

Valid for

ADQ412, ADQ1600, ADQ212, ADQ108, ADQ208, ADQ112, ADQ114, ADQ214, ADQDSP, SDR14, ADQ12, ADQ14, ADQ7, ADQ8

See also

ReadUserRegister()

34 Configuration

Data Structures

- struct ADQClockSystemParameters

  Clock system parameters. More...

Functions

- int GetParameters (enum ADQParameterId id, void *const parameters)
- int InitializeParameters (enum ADQParameterId id, void *const parameters)
- int SetParameters (void *const parameters)
- int ValidateParameters (const void *const parameters)

34.1 Detailed Description

These functions are used to configure the digitizer's parameters before proceeding with the data acquisition step.

34.2 Data Structure Documentation

34.2.1 struct ADQClockSystemParameters

Clock system parameters.

See also

SetParameters, GetParameters

Used to configure the clock system through the parameter interface.

Data Fields

enum ADQClockGenerator	clock_generator	Selects internal or external clock generation.
double	delay_adjustment	Delay adjustment, in seconds.
int32_t	delay_adjustment_enabled	Enables delay adjustment.
enum ADQParameterId	id	Must be set to ADQ_PARAMETER_ID_CLOCK_SYSTEM.
int32_t	low_jitter_mode_enabled	Enables low-jitter mode.
uint64_t	magic	Must be set to ADQ_PARAMETERS_MAGIC.
double	reference_frequency	Clock reference frequency, in Hz.
enum ADQReferenceClockSource	reference_source	Selects clock reference source.
int32_t	reserved	Reserved.

Data Fields

double	sampling_frequency	Desired sampling frequency, in Hz.

34.3 Function Documentation

34.3.1 GetParameters()

```
virtual int GetParameters (
    enum
    ADQParameterId    id,
    void *const       parameters )
```

See document 20-2465

Parameters

id
parameters

Returns

If the operation is successful, the return value is set to the size of the retrieved parameter set. Negative values are error codes.

Note

Specifically for ADQ7, this function can also be used to retrieve the ADQClockSystemParameters struct.

Valid for

ADQ14, ADQ7, ADQ8

34.3.2 InitializeParameters()

```
virtual int InitializeParameters (
    enum
    ADQParameterId    id,
    void *const       parameters )
```

See document 20-2465

Parameters

id
parameters

Returns

If the operation is successful, the return value is set to the size of the initialized parameter set. Negative values are error codes.

Valid for

ADQ14, ADQ7, ADQ8

34.3.3 SetParameters()

```
virtual int SetParameters (
    void *const        parameters )
```

See document 20-2465

Parameters

parameters

Returns

If the operation is successful, the return value is set to the size of the written parameter set. Negative values are error codes.

Note

Specifically for ADQ7, this function can also be used to set up the clock system of the digitizer via the parameter interface, as an alternative to using SetClockSource, SetTargetSampleRate, and so on. This is done via the ADQClockSystemParameters structure, and allows greater control over the clock system features.

Valid for

ADQ14, ADQ7, ADQ8

34.3.4 ValidateParameters()

```
virtual int ValidateParameters (
    const void
    *const             parameters )
```

See document 20-2465

Parameters

parameters

Returns

If the operation is successful, the return value is set to the size of the validated parameter set. Negative values are error codes.

Valid for

ADQ14, ADQ7, ADQ8

35 Data Acquisition

Data Structures

- struct ADQRecord

  Structure that represents a record buffer for Gen3 digitizers. See document 20-2465. More...
- struct ADQRecordArray

  Structure that represents an array of record buffers for Gen3 digitizers. See document 20-2465. More...

Functions

- int ReturnRecordBuffer (int channel, void *buffer)
- int StartDataAcquisition (void)
- int StopDataAcquisition (void)
- int64_t WaitForRecordBuffer (int *channel, void **buffer, int timeout, struct ADQDataReadoutStatus *status)

35.1 Detailed Description

These functions are used to initiate, terminate and control the data acquisition process.

35.2 Data Structure Documentation

35.2.1 struct ADQRecord

Structure that represents a record buffer for Gen3 digitizers. See document 20-2465.

See also

WaitForRecordBuffer, ReturnRecordBuffer

Valid for

ADQ8, ADQ14, ADQ7

Data Fields

void *	data	
struct ADQRecordHeader *	header	
uint64_t	size	

35.2.2 struct ADQRecordArray

Structure that represents an array of record buffers for Gen3 digitizers. See document 20-2465.

See also

WaitForRecordBuffer, ReturnRecordBuffer

Valid for

ADQ8, ADQ14, ADQ7

Data Fields

int32_t	nof_records	
struct ADQRecord **	record	

35.3 Function Documentation

35.3.1 ReturnRecordBuffer()

```
virtual int ReturnRecordBuffer (
    int            channel,
    void *         buffer )
```

See document 20-2465

Parameters

channel	
buffer	

Returns

ADQ_EOK if successful, negative values are error codes.

Valid for

ADQ8

35.3.2 StartDataAcquisition()

```
virtual int StartDataAcquisition (
    void            )
```

See document 20-2465

Returns

ADQ_EOK if successful, negative values are error codes.

Valid for

ADQ8

35.3.3 StopDataAcquisition()

```
virtual int StopDataAcquisition (
    void                )
```

See document 20-2465

Returns

`ADQ_EOK` if successful, `ADQ_EINTERRUPTED` may be an expected return value if an unbounded acquisition is interrupted. Otherwise, negative values are error codes.

Valid for

ADQ8

35.3.4 WaitForRecordBuffer()

```
virtual int64_t WaitForRecordBuffer (
    int *                   channel,
    void **                 buffer,
    int                     timeout,
    struct
    ADQDataReadoutStatus
    *                       status )
```

See document 20-2465

Parameters

channel	
buffer	
timeout	
status	

Returns

If the operation is successful, the return value is the size of the record buffer's data payload in bytes. The value zero indicates a successful operation, but that only the `status` parameter can be read. Negative values are error codes.

Valid for

ADQ8

36 Internal Use Only

Functions

- int **ADCCalibrate** ()
- int **ADCReg** (unsigned char addr, unsigned char adc, unsigned int val)
- int **ArmInternal001** (void)
- int **ATDEnableTestPattern** (unsigned int enable)
- int **ATDSetupTestPattern** (unsigned int record_length, unsigned int number_of_records)
- unsigned int AWGSetInterpolationFilter (unsigned int dacId, unsigned char interpolation_filter)

  Chooses the interpolation filter to use.
- unsigned int **BootAdqFromFlash** (unsigned int addr)
- unsigned int **BreakRecorderCommand** (unsigned int inst)
- int **CollectDataNextPageWithPrefetch** (unsigned int prefetch)
- int **ConManSPI** (unsigned char cmd, void ∗wr_buf, unsigned int wr_buf_len, void ∗rd_buf, unsigned int rd_buf_len)
- unsigned int **DACSpiRead** (unsigned char channel, unsigned char address, unsigned char ∗data)
- unsigned int **DACSpiWrite** (unsigned char channel, const unsigned char address, const unsigned char data)
- int **DebugCmd** (unsigned int cmd, unsigned int arg1, unsigned int arg2, unsigned int arg3, float arg4, unsigned int ∗ptr1, unsigned int ∗ptr2, unsigned int ∗ptr3)
- int **DebugParsePacketDataStreaming** (void ∗raw_data_buffer, unsigned int raw_data_size, void ∗∗target_buffers, void ∗∗target_headers, unsigned int ∗bytes_added, unsigned int ∗headers_added, unsigned int ∗header_status, unsigned char channels_mask)
- int **DisarmInternal002** (void)
- unsigned int **DisconnectInputs** (unsigned int channelmask)
- int **EnableFixedShift** (unsigned int channel, unsigned int enable)
- int **EnableRecordSegmenter** (unsigned int channel, unsigned int enable)
- unsigned int **FX2ReadRequest** (unsigned int requestcode, unsigned int value, unsigned int index, long len, char ∗buf)
- unsigned int **FX2SetRetryLimit** (unsigned int retry_limit)
- unsigned int **FX2WriteRequest** (unsigned int requestcode, unsigned int value, unsigned int index, long len, char ∗data)
- unsigned int **GetBcdDevice** ()
- unsigned int **GetComFlashEnableBit** ()
- int **GetConManSPIVersion** (unsigned int ∗major, unsigned int ∗minor)
- int **GetDeviceSNConManSPI** (char ∗device_sn)
- unsigned int **GetDeviceStatus** (unsigned int ∗status)
- unsigned int **GetDRAMPhysEndAddr** (unsigned int ∗DRAM_MAX_END_ADDRESS)
- unsigned int **GetFPGApart** (unsigned int fpganum, char ∗partstr)
- unsigned int **GetFPGASpeedGrade** (unsigned int fpganum, unsigned int ∗sgrade)

- unsigned int **GetFPGATempGrade** (unsigned int fpganum, char ∗tgrade)
- unsigned int **GetNextDSURecordingAddress** (unsigned int inst, unsigned int ∗next_address)
- unsigned int **GetNofFPGAs** ()
- unsigned int **GetNofHwChannels** ()
- unsigned int **GetNofRecorderIP** (unsigned int ∗answer)
- unsigned int **GetPageCount** ()
- unsigned int **GetRecorderBytesPerAddress** ()
- unsigned int **GetRecorderDiskStatus** (unsigned int inst, unsigned int diskno, unsigned int ∗status)
- unsigned int **GetRecorderStatus** (unsigned int inst, unsigned int ∗status)
- int **GetStreamErrors** (unsigned int channel, unsigned int ∗error)
- int **GetSystemManagerType** ()
- unsigned int **GetTrigType** ()
- unsigned int **GetUSB3Config** (unsigned int option, unsigned int ∗value)
- int **GetUSBFWVersion** (unsigned int ∗major, unsigned int ∗minor)
- int **GetWriteCount** (unsigned int ∗write_count)
- int **GetWriteCountMax** (unsigned int ∗write_count)
- int **HasConManSPIFeature** (const char ∗const feature_name)
- unsigned int **InvalidateCache** ()
- unsigned int **IsBootloader** ()
- int **IsVirtualDevice** ()
- unsigned int **MeasureInputOffset** (unsigned int channel, int ∗value)
- int **MeasureSupplyVoltage** (unsigned int sensor_num, float ∗value)
- unsigned int **OffsetDACSpiWrite** (unsigned char channel, unsigned int data)
- int **ParseEEPROMBlock** (char ∗blockname, char ∗map_version, unsigned int buffer_len, unsigned char ∗buffer, unsigned int i2c_addr)
- unsigned int **PlotAssist** (const char ∗MemoryName, void ∗MemoryPointer, unsigned int MemoryMaxBytesCount, unsigned int PlotSamplesCount, const char ∗Format)
- unsigned int **ProcessorFlashControl** (unsigned char cmd, unsigned int data)
- unsigned int **ProcessorFlashControlData** (unsigned int ∗data, unsigned int len)
- unsigned int **ra** (const char ∗regname)
- unsigned int **ReadADCCalibration** (unsigned char ADCNo, unsigned short ∗calibration)
- unsigned int **ReadAlgoRegister** (unsigned int addr)
- unsigned int **ReadDataFromDSU** (unsigned int inst, unsigned int start_address, unsigned int nofbytes, unsigned char ∗data)
- unsigned int **ReadDBI2C** (unsigned int addr, unsigned int nbytes)
- unsigned int **ReadEEPROM** (unsigned int addr, unsigned int i2c_addr)
- unsigned int **ReadI2C** (unsigned int addr, unsigned int nbytes)
- int **ReadInternal000** (unsigned int ∗arg0)
- unsigned int **ReadRegister** (unsigned int addr)
- unsigned int **RebootALGFPGAFromPrimaryImage** ()
- unsigned int **RebootCOMFPGAFromSecondaryImage** (unsigned int PCIeAddress, unsigned int PromAddress)

- unsigned int **RegisterNameLookup** (const char ∗regname, unsigned int ∗address, unsigned int allow_assertion)
- unsigned int **ReloadPCIeConfig** (unsigned int ∗pci_space)
- unsigned int **ResetCalibrationStateADQ412DC** ()
- unsigned int **ResetFIFOPaths** (unsigned int inst)
- unsigned int **ResetRecorder** (unsigned int inst)
- int **ResetWriteCountMax** ()
- unsigned int **RunRecorderSelfTest** (unsigned int inst, unsigned int ∗inout_vector)
- unsigned int **RxSetDcOffsetDac** (unsigned char channel, unsigned short dacValue)
- unsigned int **RxSetIfGainDac** (unsigned char channel, unsigned char dacValue)
- unsigned int **RxSetLinearityDac** (unsigned short dacValue)
- unsigned int **RxSetLoFilter** (unsigned char filter)
- unsigned int **RxSetLoOut** (unsigned char mode)
- unsigned int **RxSetRfPath** (unsigned char mode)
- unsigned int **RxSetVcomDac** (unsigned char channel, unsigned short dacValue)
- unsigned int **SendLongProcessorCommand** (unsigned int command, unsigned int addr, unsigned int mask, unsigned int data)
- unsigned int **SendProcessorCommand** (int command, int argument)
- unsigned int **SendProcessorCommand** (unsigned int command, unsigned int addr, unsigned int mask, unsigned int data)
- unsigned int **SendRecorderCommand** (unsigned int inst, unsigned char cmd, unsigned int arg1, unsigned int arg2, unsigned int ∗answer)
- unsigned int **SetADCClockDelay** (unsigned int adcnum, float delayval)
- unsigned int **SetAttenuators** (unsigned int channel, unsigned int attmask)
- unsigned int **SetBiasDACPercentage** (unsigned int channel, float percent)
- int **SetClockReferenceDelayDAC** (unsigned int dacvalue)
- unsigned int **SetDACPercentage** (unsigned int spi_addr, unsigned int output_num, float percent)
- int **SetDelayLineValues** (int samplerate, unsigned int linear_interpolation)
- int **SetDelayLineValuesDirect** (unsigned int delay1, unsigned int delay2)
- unsigned int **SetDMATest** (unsigned int option, unsigned int value)
- int **SetFixedShiftValue** (unsigned int channel, unsigned int shift)
- unsigned int **SetOffsetCompensationDAC** (unsigned int channel, unsigned int daccode)
- int **SetPreTrigWords** (unsigned int PreTrigWords)
- int **SetSWTrigValue** (float samples)
- unsigned int SetTimeoutFlush (unsigned int stream_timeout, unsigned int packet_timeout)

  Activate automatic flush after timeout.
- int **SetTrigCompareMask1** (unsigned int TrigCompareMask)
- int **SetTrigCompareMask2** (unsigned int TrigCompareMask)
- int **SetTrigLevel1** (int TrigLevel)
- int **SetTrigLevel2** (int TrigLevel)
- int **SetTrigMask1** (unsigned int TrigMask)
- int **SetTrigMask2** (unsigned int TrigMask)

- int **SetTrigPreLevel1** (int TrigLevel)
- int **SetTrigPreLevel2** (int TrigLevel)
- int **SetupBlockAvg** (unsigned int channel, unsigned int size, unsigned int scale)
- unsigned int **SetupDSUAcquisition** (unsigned int inst, unsigned int start_address, unsigned int end_address)
- int **SetupInternal000** (unsigned int arg0, int arg1, int arg2, unsigned int arg3, unsigned int arg4, unsigned int arg5, unsigned int arg6)
- int **SetupInternal004** (unsigned int arg0, unsigned int arg1, unsigned int arg2, int arg3)
- int **SetupInternal005** (unsigned int arg0)
- int **SetupInternal006** (unsigned int arg0)
- int **SetupInternal007** (unsigned int ∗arg0)
- int **SetupInternal008** (unsigned int arg0)
- int **SetupInternal009** (unsigned int arg0)
- int **SetupInternal010** (unsigned int arg0)
- unsigned int **SetupInternal011** (unsigned int arg0, unsigned int arg1, unsigned int arg2, unsigned int arg3)
- int **SetupRecordSegmenter** (unsigned int channel, unsigned int ∗seg_length, unsigned int ∗gap_length, unsigned int nof_segments)
- unsigned int **SetUSB3Config** (unsigned int option, unsigned int value)
- int **SetWordsAfterTrig** (unsigned int WordsAfterTrig)
- int **SetWordsPerPage** (unsigned int WordsPerPage)
- int **SmTransaction** (uint16_t cmd, const void ∗const wr_buf, size_t wr_buf_len, void ∗const rd_buf, size_t rd_buf_len)
- int **SpiSend** (unsigned char addr, const char ∗data, unsigned char length, unsigned int negedge, unsigned int ∗ret)
- unsigned int **StartDSUAcquisition** (unsigned int inst)
- unsigned int **StorePCIeConfig** (unsigned int ∗pci_space)
- unsigned int **SynthDisableAutoLevel** (unsigned char channel, unsigned char mode)
- unsigned int **SynthGetAlcDac** (unsigned char channel, unsigned int ∗dacValue)
- unsigned int **SynthSetAlcDac** (unsigned char channel, unsigned int dacValue)
- unsigned int **SynthSetAlcMode** (unsigned char channel, unsigned char mode)
- unsigned int **SynthSetDeviceStandby** (unsigned char channel, unsigned char standbyStatus)
- unsigned int **SynthSetReferenceDac** (unsigned int dacValue)
- unsigned int **TriggeredStreamingSetupGatedAcq** (unsigned int NofRecords, unsigned int NofPreTrigSamples, unsigned int NofTriggerDelaySamples, unsigned int NofPostTrigSamples, unsigned char ChannelsMask)
- unsigned int **TxSetDcOffsetDac** (unsigned char channel, unsigned short dacValue)
- unsigned int **TxSetFrequency** (unsigned long long int frequency)
- unsigned int **TxSetLinearityDac** (unsigned char channel, unsigned short dacValue)
- unsigned int **TxSetLoFilter** (unsigned char filter)
- unsigned int **TxSetLoOut** (unsigned char mode)
- unsigned int **TxSetRfPath** (unsigned char mode)
- int **USBLinkupTest** (unsigned int retries)
- unsigned int **USBReConnect** ()

- unsigned int **WaveformAveragingStartReadout** ()
- unsigned int **WriteADCCalibration** (unsigned char ADCNo, unsigned short ∗calibration)
- unsigned int **WriteADQATTStateManual** (unsigned int channel, unsigned int relay16, unsigned int relay8, unsigned int ptap8, unsigned int ptap4, unsigned int ptap2, unsigned int ptap1, unsigned int ptap05, unsigned int ptap025)
- unsigned int **WriteAlgoRegister** (unsigned int addr, unsigned int mask, unsigned int data)
- unsigned int **WriteDataToDSU** (unsigned int inst, unsigned int start_address, unsigned int nofbytes, unsigned char ∗data)
- unsigned int **WriteDBI2C** (unsigned int addr, unsigned int nbytes, unsigned int data)
- unsigned int **WriteEEPROM** (unsigned int addr, unsigned int data, unsigned int accesscode, unsigned int i2c_addr)
- unsigned int **WriteI2C** (unsigned int addr, unsigned int nbytes, unsigned int data)
- int **WriteInternal003** (void ∗arg0, unsigned int arg1)
- unsigned int **WriteReadI2C** (unsigned int addr, unsigned int rbytes, unsigned int wbytes, unsigned int wrdata)
- unsigned int **WriteRegister** (unsigned int addr, unsigned int mask, unsigned int data)

36.1 Detailed Description

These functions are for internal use by SP Devices, and are listed here for completeness.

36.2 Function Documentation

36.2.1 AWGSetInterpolationFilter()

```
virtual unsigned int AWGSetInterpolationFilter (
    unsigned int      dacId,
    unsigned char     interpolation_filter )
```

Chooses the interpolation filter to use.

If the AWG is to be rearmed after having triggered, an AWGTrigoutDisarm() command must first be issued.

Parameters

dacId Selects the AWG/DAC (1 or 2)	
interpolation_filter 0 - Default (no filter). DRAM Data rate will be 1600 MHz. 4 - 4x interpolation stage activated. DRAM Data rate will be 400MHz.	

Returns

unsuccessful if interpolation filter is not available

Valid for

SDR14

36.2.2 SetTimeoutFlush()

```
virtual unsigned int SetTimeoutFlush (
    unsigned int      stream_timeout,
    unsigned int      packet_timeout )
```

Activate automatic flush after timeout.

Set timeout for automatic data flushing

Parameters

stream_timeout	Set timeout in us for flushing stream data fifo.
packet_timeout	Set timeout in us for packet data fifo.

Returns

1 for successful operation and 0 for failure

Valid for

ADQ7

37 Deprecated Functions

Functions

- void ADQControlUnit_DeleteADQ108 (void ∗adq_cu_ptr, int adq108_num)

  Use ADQControlUnit_DeleteADQ() instead.
- void **ADQControlUnit_DeleteADQ112** (void ∗adq_cu_ptr, int adq112_num)
- void ADQControlUnit_DeleteADQ114 (void ∗adq_cu_ptr, int adq114_num)

  Use ADQControlUnit_DeleteADQ() instead.
- void ADQControlUnit_DeleteADQ1600 (void ∗adq_cu_ptr, int adq1600_num)

  Use ADQControlUnit_DeleteADQ() instead.
- void ADQControlUnit_DeleteADQ208 (void ∗adq_cu_ptr, int adq208_num)

  Use ADQControlUnit_DeleteADQ() instead.
- void ADQControlUnit_DeleteADQ212 (void ∗adq_cu_ptr, int adq212_num)

  Use ADQControlUnit_DeleteADQ() instead.
- void ADQControlUnit_DeleteADQ214 (void ∗adq_cu_ptr, int adq214_num)

  Use ADQControlUnit_DeleteADQ() instead.
- void ADQControlUnit_DeleteADQ412 (void ∗adq_cu_ptr, int adq412_num)

Use ADQControlUnit_DeleteADQ() instead.

- void ADQControlUnit_DeleteADQDSP (void *adq_cu_ptr, int ADQDSP_num)

  Use ADQControlUnit_DeleteADQ() instead.

- void ADQControlUnit_DeleteDSU (void *adq_cu_ptr, int ADQDSP_num)

  Use ADQControlUnit_DeleteADQ() instead.

- void ADQControlUnit_DeleteSDR14 (void *adq_cu_ptr, int sdr14_num)

  Use ADQControlUnit_DeleteADQ() instead.

- unsigned int AWGmalloc (unsigned int dacId, unsigned int LengthSeg1, unsigned int LengthSeg2, unsigned int LengthSeg3, unsigned int LengthSeg4)

  Use AWGSegmentMalloc() instead.

- unsigned int CollectRecord (int RecordNumber)

  Use GetData() instead.

- unsigned int DisableWFATriggerCounter ()

  Do not use.

- unsigned int EnableWFATriggerCounter ()

  Do not use.

- unsigned int GetBufferSize ()

  No usage necessary.

- unsigned int GetBufferSizePages ()

  No usage necessary.

- unsigned int GetDataMultiRecordSetup (unsigned int NumberOfRecords, unsigned int SamplesPerRecord)

  Use MultiRecordSetup() instead.

- unsigned int GetInterleavingIPFrequencyCalibrationMode (unsigned char IPInstanceAddr, unsigned int *freqcalflag)

  No usage necessary.

- int GetLvlTrigFlank ()

  Use GetLvlTrigEdge instead.

- unsigned long long GetMaxBufferSize ()

  No usage necessary.

- unsigned int GetMaxBufferSizePages ()

  No usage necessary.

- unsigned int * GetMultiRecordHeader ()

  Use GetDataWH instead.

- int * GetPtrDataChA ()

  Use GetPtrData() instead.

- int * GetPtrDataChB ()

  Use GetPtrData() instead.

- unsigned int GetRxFifoOverflow ()

  No usage necessary.

- int GetTrigged ()

  Use GetAcquired instead.
- unsigned int GetTriggedAll ()

  Use GetAcquiredAll() instead.
- unsigned int MultiRecordGetRecord (int RecordNumber)

  Use GetData() instead.
- int SetAlgoNyquistBand (unsigned int band)

  No usage necessary.
- int SetAlgoStatus (int status)

  Use SetInterleavingIPBypassMode() instead.
- int SetBufferSize (unsigned int samples)

  Use MultiRecordSetup() instead.
- int SetBufferSizePages (unsigned int pages)

  No usage necessary.
- int SetBufferSizeWords (unsigned int words)

  No usage necessary.
- unsigned int SetInterleavingIPFrequencyCalibrationMode (unsigned char IPInstanceAddr, unsigned int freqcalmode)

  No usage necessary.
- int SetNofBits (unsigned int NofBits)

  Use SetDataFormat() instead.
- int SetSampleWidth (unsigned int SampleWidth)

  Use SetDataFormat() instead.
- int SetTriggerHoldOffSamples (unsigned int TriggerHoldOffSamples)

  Use SetTriggerDelay() instead.
- unsigned int SetWFANumberOfTriggers (unsigned int number_of_triggers)

  Do not use.
- unsigned int StartWFATriggerCounter ()

  Do not use.

37.1 Detailed Description

These functions are deprecated