ADQAPI

User Guide

Contents

	Classification	**Revision**
	Public	65395
	Document ID	**Print date**
	08-0214	2022-05-04

TELEDYNE SP DEVICES
Everywhere**you**look™

1 Introduction

This document contains instructions and guidelines of how to use the ADQAPI from different programming languages to control a TSPD digitizer.

1.1 Definitions and Abbreviations

Table 1 lists the definitions and abbreviations used in this document.

Table 1: Definitions and abbreviations used in this document.

Item	Description
ADQAPI	The Application Programming Interface for a digitizer.
DLL	Dynamic Link Library (Windows code library)
ADQAPI.dll	The Windows Dynamic Link Library containing the ADQAPI implementation.
libadq.so	In Linux, this is the library containing the ADQAPI implementation.
MEX	The interface method used for MATLAB.
Waveform Averaging	Feature for hardware supported averaging for V5/V6 products.
FWDAQ	Standard firmware for ADQ32/ADQ36/ADQ14/ADQ12/ADQ7/ADQ8 for data acquisition.
FWATD	Specific firmware forADQ14/ADQ12/ADQ7 for hardware supported averaging.
FWPD	Specific firmware for ADQ14/ADQ12/ADQ7 for data-driven Pulse Detection applications.
FWDDM	Specific firmware for ADQ7 for disk drive measurement applications
FWSDR / FW2DDC / FW4DDC	Specific firmwares for ADQ14/ADQ12/ADQ7 for Software Defined Radio.

2 Overview

The ADQAPI provides a simple and powerful programming interface to ADQ devices. The programming interface handles all communication with the connected ADQ devices with a few highly abstracted functions. The API is available for Windows and Linux and provides access through several programming languages. The user application can be implemented in for instance Python, TCL, C, C++, C#, MATLAB or LabView making use of calls into the API library.
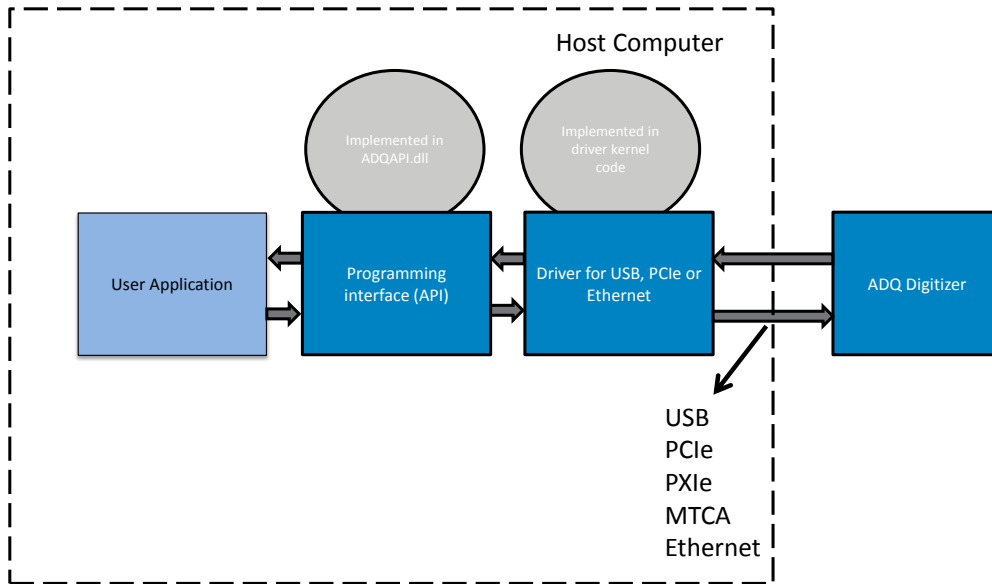
Figure 1: A block diagram showing the structure of the software.

2.1 Running on Windows

These classes are encapsulated in a DLL file and interfaced via a function set where the user specifies which ADQ device to communicate with. The interface consists of three files:

ADQAPI.lib

> This file must be linked to the code project for the program to compile successfully.

ADQAPI.dll

> This dynamic linked library must be located in the same directory as the compiled program or have a proper path for it set up. When SP Devices software development kit (SDK) is installed, this DLL is copied to the windows DLL directory and will always be accessible.

ADQAPI.h

> A header file that must be linked to the code project for declaration of the ADQAPI function set. This is used for programming in C/C++. Other languages, need a modified header file.

The SDK installation provides three different versions of these files. If the code project is compiled on a on a 32-bit system, the files in the ADQAPI folder must be used. If the code project is compiled on a 64-bit system, the files in ADQAPI_64/ must be used for 64-bit applications and the files in ADQAPI_32_64/ should be used for 32-bit applications.

> **❶ Important**
>
> The correct API version must be selected when building the code otherwise the API will throw an error code of [$0x00000001$ `ERROR_CODE_ADQAPI_NOT_BUILT_FOR_CORRECT_OS`] and exit execution unsuccessfully during `FindDevices()` or `ListDevices()` call. This error code can be read via the API `ADQControlUnit_GetLastFailedDeviceError()` or in the ADQControlUnit log file (called `spd_adqcontrolunit_trace.log`, if enabled by `ADQControlUnit_EnableErrorTrace()`.

2.2 Running on Linux

The Linux ADQ library providing ADQAPI will follow existing naming conventions and will be called libadq (`libadq.so`). If installed from a package, the library will be installed in `/usr/lib` and the API header (`ADQAPI.h`) will be installed in `/usr/include`. Instructions on how to install and use the ADQAPI on Linux are found in the installation package. On Linux, the ADQAPI only supports 64-bit systems.

3   API

This section describes the structure of the ADQAPI and the methods of using it from different programming languages.

3.1   ADQAPI Objects

The ADQAPI uses these types of objects (classes):

ADQControlUnit

> An object that manages the connection between the digitizers and the host computer. The ADQControlUnit creates objects of type ADQ36, ADQ32, ADQ33, ADQ8, ADQ7, ADQ14, ADQ12, ADQDSP, DSU, SDR14, SDR14TX, ADQ212, ADQ412, ADQ108, ADQ208, ADQ1600, ADQ112, ADQ114 and ADQ214 when finding devices over the available interfaces.

ADQ specific objects

> Class objects that handles the communication with each device.

3.2   ADQAPI function calls

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

ADQAPI specific functions

> Functions purely related to the API itself and not to the operation of digitizers.

ADQControlUnit functions

> Functions used to interface with the device driver for tasks such as finding and initializing digitizers.

ADQ functions

> Functions used to interface directly with a specific digitizer.

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

> **ⓘ Note**
>
> Using the API method `FindDevices()` is not intended for systems with more than one digitizer when operated from different application instances. See Section 3.8 for more information.

3.3   C API

When interfacing the C API, all functions other than the ADQControlUnit functions and the ADQAPI-specific functions are called by prefixing the function name with `ADQ_` and adding the previously created ADQControlUnit 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

// Next line configures the ADQAPI to write logfiles for any errors occurred in the current directory
ADQControlUnit_EnableErrorTrace(adq_cu_ptr, LOG_LEVEL_ERROR, ".");

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(adq_cu_ptr);
```

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

3.4   C++ 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 a C++ class object that can be used to access all functions that operates on the unit. These functions are 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

// Next line configures the ADQAPI to write logfiles for any errors occurred in the current directory
ADQControlUnit_EnableErrorTrace(adq_cu_ptr, LOG_LEVEL_ERROR, ".");

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(adq_cu_ptr);
```

The ADQControlUnit functions and the ADQAPI specific functions are always interfaced as C API functions.

> **ℹ Note**
>
> The difference between using C and C++ style API calls is only on the calling syntax, the exact same API functions and arguments are used. In C you supply the digitizer object reference as an argument, and in C++ you instead call methods for an `ADQInterface` object you have earlier fetched a reference to.

3.5   Matlab

On Windows, there is an interface that should feel familiar for Matlab users. This is implemented in the DLL called `mex_ADQ.dll`, which is installed with the other DLLs. For convenient interfacing, there is also a wrapper file called `interface_ADQ.m`. The API calls, On Linux there is no provided specific MATLAB interface support.

Classification
Public

Revision
65395

Document ID
08-0214

Print date
2022-05-04

> ❶ **Note**
>
> All existing API calls are not implemented for the MATLAB interface and will give errors if executed.

> ❶ **Important**
>
> The ADQ3 series of products, ADQ32, ADQ33 and ADQ36 do not have support for MATLAB.

3.6 .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 guide [1].

3.7 Python

Python is supported through the provided pyADQ package. Excerpt from Python example to list and setup all devices and then exit:

```python
#!/usr/bin/env python3
# Copyright 2022 Teledyne Signal Processing Devices Sweden AB
"""
    This example will enumerate and initialize all devices and then exit
"""
import pyadq

print("pyadq version:", pyadq.__version__)

acu = pyadq.ADQControlUnit()

acu.ADQControlUnit_EnableErrorTrace(pyadq.LOG_LEVEL_INFO, ".")
device_list = acu.ListDevices()

print(f"Found len(device_list) device(s)")
for index in range(len(device_list)):
    with acu.SetupDevice(index) as dev:
        print(dev)
```

3.8 Identifying and setting up a device for operation

`FindDevices()` is the easiest method to access the digitizer. In a system with one digitizer and one application controlling it, this method works fine. However, using `FindDevices()` will find all digitizer units in the system (regardless of interface types) and set them up to a new state regardless of if they have been accessed earlier and/or is under operation from another application in the system. To use different devices from different applications, all applications need to utilize the `ListDevices()` method rather than the `FindDevices()` method

3.8.1 Using `FindDevices()`

Example of accessing a digitizer using `FindDevices()`:

```
void* adq_cu_ptr = CreateADQControlUnit(); // Creates an ADQControlUnit called adq_cu_ptr

// Next line configures the ADQAPI to write logfiles for any errors occurred in the current directory
ADQControlUnit_EnableErrorTrace(adq_cu_ptr, LOG_LEVEL_ERROR, ".");

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(adq_cu_ptr);
```

3.8.2 Using `ListDevices()` **(advanced)**

The `ListDevices()` method is more complicated to use, but allows different digitizers to be individually accessed from different applications. As long as a device is not opened by two applications at the same time, the control will work completely independent of eachother.

Instead of finding and setting up all digitizers in one single step, this method consists of three different steps:

- `ListDevices()` – Obtains a list of all available digitizers. This will not access the digitizers themselves in any way.

- `SetupDeviceInterface()` – Opens the interface to a specific digitizer in the returned list.

- `SetupDevice()` – Sets up the digitizer for operation from this application.

> **❶ Note**
>
> When handling entries in the list from `ListDevices()`, the list is zero-based. When continuing to access the digitizer through the API, the digitizer numbers are one-based.

A list entry is defined as:

```
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;
};
```

Example of accessing a digitizer using `ListDevices()`:

```
struct ADQInfoListEntry* ADQlist;
int nof_devices;
int adq_num;
int adq_list_num;
int success;
void* adq_cu_ptr = CreateADQControlUnit(); // Creates an ADQControlUnit called adq_cu_ptr
```

```
// Next line configures the ADQAPI to write logfiles for any errors occurred in the current directory
ADQControlUnit_EnableErrorTrace(adq_cu_ptr, LOG_LEVEL_ERROR, ".");

if(adq_cu_ptr  != NULL)
{
  // Returns a list of all found devices in ADQList argument and number of devices in list in nof_devices argument
  success = ADQControlUnit_ListDevices(adq_cu_ptr, &ADQlist, &nof_devices);
  if (nof_devices > 0)
  {
    adq_list_num = 0; // There is at least one device, access first item.
    success = success && ADQControlUnit_OpenDeviceInterface(adq_cu, adq_list_num));
    success = success && ADQControlUnit_SetupDevice(adq_cu, adq_list_num);
    adq_num = adq_list_num + 1; // Set accessed digitizer number to one + list number (one-based instead of zero-based).
    if (success)
    {
      ADQ_Blink(adq_cu_ptr, adq_num); // Blinks one of the LEDs for the device number 1, the one setup
    }
    else
    {
      printf("Failed accessing digitizer");
    }
  }
}
DeleteADQControlUnit(adq_cu_ptr);
```

> ⚠ **Warning**
>
> Accessing a list item that is not inside the returned `nof_devices` range will access memory out of bounds. This may cause segmentation fault or other memory corruption problems.

4   Application Programming Flowchart

The digitizer can be operated in several different acquisition modes. Some of them are described in this section.

> **❶ Important**
>
> This section does not apply for the ADQ3 series of products, ADQ32/ADQ36. For these, please only use the ADQ3 Series Manual [**?**]

4.1   Acquisition mode: Multi-record

The multi-record mode is used for triggering the unit several times in succession and store the data in the on-board DRAM for later transfer. This means that data may be captured in real time, even if it may not be transferred as quickly. To not lose any data, the data of interest must be downloaded from the digitizer before the DRAM is full. Typical properties of a multi-record acquisition:

- Limited number of records.

- Total size contained inside DRAM size.

- Long pretrigger available (up to full record size).

- Rearm time is rather long: in the order of microseconds.

- Digitizer is capturing data, data is sent to host and then whole loop is armed again. See the flowchart in Fig. 2.
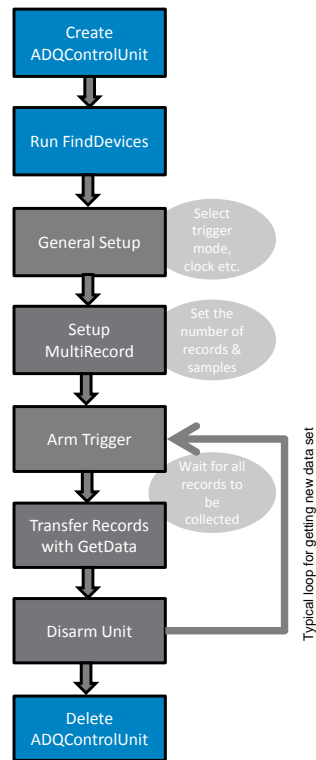
Classification
Public

Revision
65395

Document ID
08-0214

Print date
2022-05-04

Figure 2: A high level flowchart for the multi-record mode.

4.2 Acquisition mode: Triggered streaming

The triggered streaming mode is used for triggering devices and produce records, sent to the host computer. There is an option to run triggered streaming in an infinite mode, meaning that it will continue to produce records indefinitely.

> **❶ Note**
>
> Triggered streaming is only available on the products ADQ7, ADQ14, ADQ12 and ADQ8[1]. There is also some limited legacy support for ADQ412 and ADQ214 (see ADQAPI Reference Guide [2] for more information.

> **❶ Important**
>
> The recommended flow for triggered streaming on ADQ7, ADQ14 and ADQ8 is documented in a separate document and is *not* the same as the one presented in this section which is a legacy implementation flow (still fully supported but not recommended). Please refer to the user guide for streaming on Gen3 digitizers [3] for more details.

- Limited or unlimited number of records.

---

[1]This is described in the user guide for streaming on Gen3 digitizers. [3]

- DRAM used as FIFO to secure no loss of data.

- Only short pretrigger length available (in the order of kiloSamples).

- Rearm time is short: in the order of tens of nanoseconds.

- Digitizer is capturing data, data is sent to host and then can go on until the user stops it. See the flowchart in Fig. 3
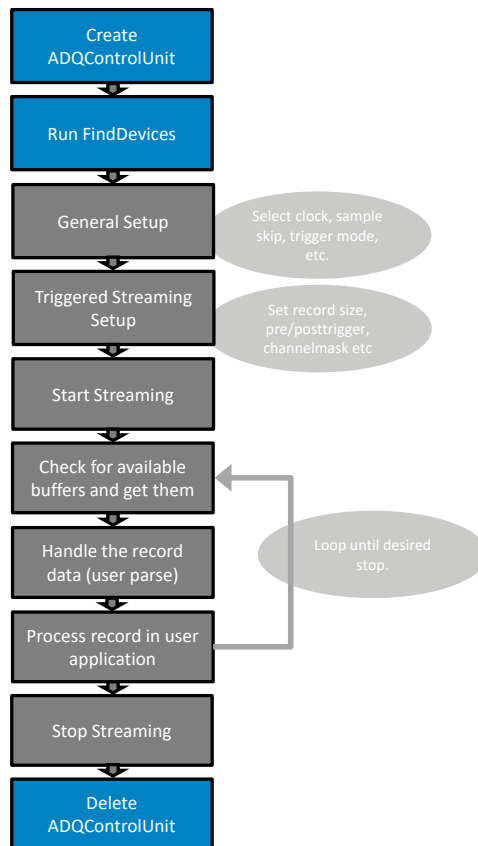

Figure 3: A high level flowchart for the triggered streaming mode.

		Classification	Revision
		Public	65395
		Document ID	**Print date**
		08-0214	2022-05-04

TELEDYNE SP DEVICES
Everywhere**you**look™

4.3   Acquisition mode: Streaming

Streaming is used to continuously transfer data from the digitizer to the host computer. Because the bandwidth of the interface is, in most cases, not sufficient to transfer data in real time, there are several options to provide data reduction, for instance using sample skip or sample decimation. The streaming interface can also be used to obtain data from custom firmware designs, in a raw fashion.

> **❶ Note**
>
> Sample decimation is only available for some product versions.

- No records, just raw data.

- DRAM used as FIFO to secure no loss of data.

- No concept of pre or posttrigger or trigger timing.

- Digitizer is capturing data, data is sent to host and then can go on until the user stops it.
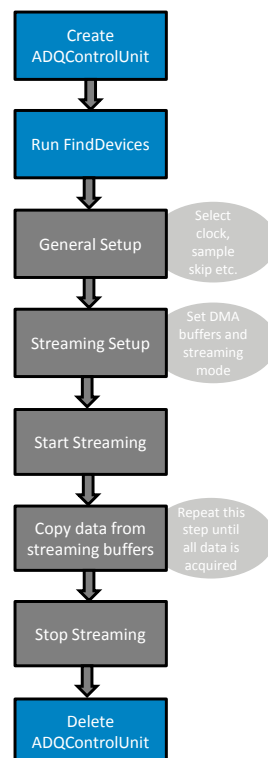

Figure 4: A high level flowchart for the streaming mode.

5 Multithreading

The ADQAPI is in general not thread safe. The rule is simple, it is not allowed to call any API function that communicates with the hardware from two different threads on the PC at the same time. It is strongly recommend to keep all digitizer access control in one single thread and use other threads for handling the data acquired, storing data acquired and other tasks.

Still, in some cases, it may be beneficial to use threading to communicate with devices and in those cases we ask the user to contact technical support at Teledyne SP Devices to verify that commands used in different threads do not suffer from collision risks (communicating with hardware).

Threading errors/problems are usually hard to detect and the error messages usually do not show that the problem relates to threading, but instead can point in many confusing directions. However, some typical errors when having violated the threading rule is that transfers suddenly do not complete correctly or some reading of a setting shows an error or not allowed value.

5.1 Recommendation

> **❶ Important**
>
> It is strongly recommend to keep all digitizer access control in one single thread.

6 Troubleshooting

This section aims to provide some guidance when troubleshooting unexpected behavior. It is recommended that the user application is written in a robust manner, able to capture and report error codes from failed ADQAPI function calls. In the event of a function call failure, reading the ADQAPI trace log for additional information is a useful first step. Trace logging must be activated by calling `ADQControlUnit_EnableErrorTrace()` with the `trace_level` argument set to 3. See details in [2].

The ADQAssist (GUI) application can also be used to enable monitoring of the digitizer interfacing and capture logs. For more information on the ADQMonitor function, consult the ADQAssist User Guide [4].

If the error message is difficult to interpret, the Teledyne SP Devices support can be reached via e-mail at spd_support@teledyne.com. Please include information about your use case such as the trigger settings as well as the specification of the signals connected, if any. Please also include a description of the behavior and how it differs from the behavior expected. Make sure to include a trace log file from a run where the error appears. Also always include your digitizer's serial number when reporting a new ticket.

> **ⓘ Note**
>
> Activating trace logs will consume host system resources for writing the log to disk during execution. To achieve maximum performance in an application, trace logging should be turned off or set to only produce error messages, not informational and warning types of messages.

7 Code Examples

7.1 Overview

Every example does not support all devices and programming languages. The main example code for ADQ14/ADQ12/ADQ7/ADQ8/SDR14TX/ADQ214/ADQ1600 (note, not for the ADQ3 series) is called ADQAPI_example and is written in C. It features examples of most operational modes but thus also contains a lot of code that is not relevant for the approach the user has chosen. A sutiable basic starting point for those products can instead be the ADQAPI_simple_example where a very simple route to acquire a triggered record is implemented. The ADQ3 series comes with its own set of example code, separate from other digitizers, and the data_readout example is a good starting point (for more information, consult the ADQ3 Series Manual [**?**]).

> **❶ Important**
>
> All references to the term *waveform averaging*, hardware supported averaging, are related to the V5/V6 family of products (ADQ412, SDR14, ADQ1600, ADQ114, ADQ214). In the products ADQ14, ADQ12, ADQ7 the hardware supported averaging is handled by the firmware option FWATD (*advanced time-domain*). The examples are not interchangeable.

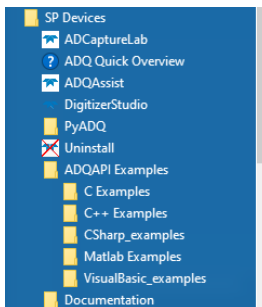Figure 5: The code examples can be found in the start menu (on Windows).

> **❶ Important**
>
> When building the code examples, make sure you are building for the correct target platform - otherwise the API will throw an error and not find any devices. If you intend to run the application on a 64-bit OS you must select x64 as target and if you intend to run it on a 32-bit OS you must select Win32 as target.

7.2   Definitions

Table 2 lists the definitions used throughout Section 7.

Table 2: A list of definitions for code example reference section.

Item	Description
Interface language	The language used in the example (for instance C/C++/Python/MATLAB/etc.).
Complexity level	Indicates the level of complexity of the example: basic, medium or high. The high level requires more competence, programming experience and attention to details.
Build environment	Provided environments for building. Windows typically uses Microsoft Visual Studio projects and Linux typically uses automake files.
Valid for product(s)	The digitizer products the example works for (with no modifications).
Suitable as example for	The digitizer products for which the example is suitable to use as base for a user application.
Demonstrates features	A list of the API and/or product features demonstrated in the example.
Overview	An abstract of what the example demonstrates.
Helper files	Pointer to any helper files, e.g for plotting, handling data etc.

7.3   C/C++ Language examples

7.3.1   data_readout.zip

Interface language

   C

Complexity level

   Basic

Build environment

   Windows, Linux

Valid for product(s)

   ADQ32, ADQ33, ADQ36

Suitable as example for

   ADQ32, ADQ33, ADQ36

Demonstrates features

   Finding devices, setting up, acquisition

Overview

> Sets up the digitizer in a state controlled by compiler definitions, acquires a record and outputs the record samples to a file. For more information, consult the ADQ3 Series Manual [**?**] and the README supplied in the example archive.

7.3.2 data_transfer_gpu_amd.zip

Interface language

> C

Complexity level

> Advanced

Build environment

> Windows

Valid for product(s)

> ADQ32, ADQ33, ADQ36

Suitable as example for

> ADQ32, ADQ33, ADQ36

Demonstrates features

> Transfer of data to GPU

Overview

> This example demonstrates streaming to an AMD GPU using DirectGM. DirectGMA is only supported by professional AMD GPUs, specified as DirectGMA in datasheet. For more information, consult the ADQ3 Series Manual [**?**] and the README supplied in the example archive.

7.3.3 data_transfer_gpu_nvidia_through_host.zip

Interface language

> C

Complexity level

> Advanced

Build environment

> Windows, Linux

Valid for product(s)

> ADQ32, ADQ33, ADQ36

Suitable as example for

> ADQ32, ADQ33, ADQ36

Demonstrates features

Transfer of data to GPU

Overview

This example demonstrates streaming to an Nvidia GPU through host buffers. The data is first transferred to host RAM buffers and then copied to GPU buffers. The host RAM buffers are registered by Cuda (cudaHostRegister) for accelerated memcpy to GPU. Supported by all Cuda capable devices. For more information, consult the ADQ3 Series Manual [**?**] and the README supplied in the example archive.

7.3.4 ADQAPI_simple_example

> **ⓘ Note**
>
> For ADQ14 and ADQ7 this example will only run with the standard FWDAQ firmware on the device.

Interface language

C

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

ADQ14, ADQ7, ADQ12, ADQ214, ADQ412, SDR14

Suitable as example for

ADQ14, ADQ7, ADQ12, ADQ214, ADQ412, SDR14

Demonstrates features

Finding devices, setting trigger modes, multi-record acquisition

Overview

Sets up the digitizer in a state controlled by compiler definitions, acquires a record and outputs the record samples to a file.

Helper files

└ `plot_simple_example.m`　　　　　　　　　　a plot helper for MATLAB, plotting the data from the saved file

7.3.5 ADQAPI_example

> **ⓘ Note**
>
> For ADQ14, ADQ7 and ADQ8 this example will only run with the standard FWDAQ firmware on the device.

Interface language

C

Complexity level

Medium

Build environment

Windows, Linux

Valid for product(s)

ADQ14, ADQ7, ADQ8, ADQ12, ADQ214, ADQ412, SDR14

Suitable as example for

ADQ14, ADQ7, ADQ8, ADQ12, ADQ214, ADQ412, SDR14

Demonstrates features

Lisiting devices, setting trigger modes, multi-record acquisition, triggered streaming, raw streaming, continuous streaming, configuring DBS, reading temperatures

Overview

Can be used for all products and demonstrates several modes and features in different combinations.

Detailed description

If used as a base for a user application, code that is not used should be removed to simplify the process.

Helper files

`plot.py`	a plot helper for Python
`plot_data_file.m`	a plot helper for MATLAB
`ADQ214_GetData_Plot.m`	a plot helper for MATLAB for ADQ214
`ADQ412_GetData_Plot.m`	a plot helper for MATLAB for ADQ412
`ADQ412_TriggeredStreaming_Plot.m`	a plot helper for MATLAB for ADQ412

7.3.6 ADQAPI_transfer_test_example

> **ⓘ Note**
>
> This example will only run with the standard FWDAQ firmware on the device.

Interface language

> C

Complexity level

> Medium

Build environment

> Windows, Linux

Valid for product(s)

> ADQ14, ADQ12. ADQ7

Suitable as example for

> ADQ14, ADQ12, ADQ7

Demonstrates features

> Listing devices, internal trigger, triggered streaming

Overview

> Tests the system transfer performance by running triggered streaming acquisitions with an internal trigger generator creating the desired amount of data.

Detailed description

> The application example is described in more detail in the "Application Note: System Transfer Test" [5]. The example measures the transfer performance and can be used to test and debug any transfer performance issues in a system.

7.3.7 ADQAPI_FWATD_example

> **ⓘ Note**
>
> This example will only run with a licensed FWATD firmware on the device.

Interface language

> C

Complexity level

> Medium

Build environment

Windows, Linux

Valid for product(s)

ADQ14-FWATD, ADQ7-FWATD, ADQ12-FWATD

Suitable as example for

ADQ14-FWATD, ADQ7-FWATD, ADQ12-FWATD

Demonstrates features

Finding devices, configure DBS, acquire accumulation results, FWATD, saving data to disk.

Overview

Sets up the digitizer to generate accumulated records with the hardware accelerated FWATD solution.

Detailed description

By default, the example uses a test pattern to verify digital data integrity. To enable the analog input, please adjust code line implementing the call to the `SetTestPattern()` API. The example also contains code for configuring a pre-processing FIR filter.

7.3.8   ADQAPI_FWPD_example

> **ℹ Note**
>
> This example will only run with a licensed FWPD firmware on the device.

Interface language

C

Complexity level

Medium

Build environment

Windows, Linux

Valid for product(s)

ADQ14-FWPD, ADQ7-FWPD, ADQ12-FWPD

Suitable as example for

ADQ14-FWPD, ADQ7-FWPD, ADQ12-FWPD
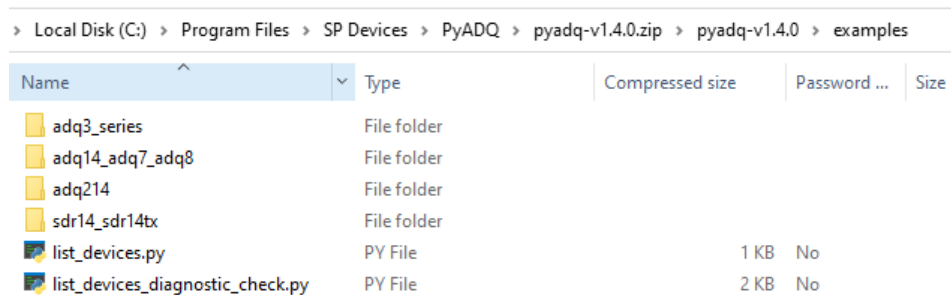
Demonstrates features

Finding devices, configure DBS, acquire pulse results, FWPD, saving results to disk

Overview

Sets up the digitizer to generate data-driven pulse data with the hardware accelerated FWPD solution.

7.4 Python Language examples

Python is supported through a package called pyADQ, which is included in the installer. The examples in the pyADQ package are sorted into directories dependent on the type of digitizer used.

> Local Disk (C:) > Program Files > SP Devices > PyADQ > pyadq-v1.4.0.zip > pyadq-v1.4.0 > examples

Name	Type	Compressed size	Password ...	Size
adq3_series	File folder			
adq14_adq7_adq8	File folder			
adq214	File folder			
sdr14_sdr14tx	File folder			
list_devices.py	PY File	1 KB	No	
list_devices_diagnostic_check.py	PY File	2 KB	No	

Figure 6: The Python code examples structure.

In Linux packages, the pyadq package resides in the `/packages/many` directory.

7.4.1 adq3_series_example.py

Interface language

Python3

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

ADQ32, ADQ33, ADQ36

Suitable as example for

ADQ32, ADQ33, ADQ36

Demonstrates features

Finding devices, setting clocking, acquisition and readout parameters, streaming

Overview

Sets up the digitizer in a state controlled by code definitions, acquires records in streaming mode.

7.4.2 adq14_adq7_adq8_streaming_example.py

Interface language

Python3

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

ADQ14, ADQ12, ADQ7, ADQ8

Suitable as example for

ADQ14, ADQ7, ADQ8, ADQ12

Demonstrates features

Finding devices, setting trigger modes, triggered streaming

Overview

Sets up the digitizer in a state controlled by code definitions, acquires records in triggered streaming mode and outputs the header info to the console and makes a simple plot of the data.

7.4.3 adq14_adq7_adq8_multirecord_example.py

Interface language

Python3

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

ADQ14, ADQ12, ADQ7, ADQ8

Suitable as example for

ADQ14, ADQ7, ADQ8, ADQ12

Demonstrates features

Finding devices, setting trigger modes, multi-record

Overview

Sets up the digitizer in a state controlled by code definitions, acquires records in multi-record mode

and outputs the header info to the console and makes a simple plot of the data.

7.4.4 adq14_adq7_adq8_streaming_oscilloscope_view.py

Interface language

Python3

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

ADQ14, ADQ12, ADQ7, ADQ8

Suitable as example for

ADQ14, ADQ7, ADQ8, ADQ12

Demonstrates features

Finding devices, setting trigger modes, triggered streaming

Overview

Sets up the digitizer in a state controlled by code definitions, acquires records in triggered streaming mode and provides an oscilloscope-like view of data.

7.4.5 ADQ214_example.py

Interface language

Python3

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

ADQ214

Suitable as example for

ADQ214

Demonstrates features

Finding devices, setting trigger modes, multi-record acquisition

Overview

Sets up the digitizer in a state controlled by code definitions, acquires a record and plots some data.

7.4.6   sdr14tx_fileoutput.py

Interface language

Python3

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

SDR14TX

Suitable as example for

SDR14TX, SDR14

Demonstrates features

Finding devices, configuring DAC data.

Overview

Sets up the generator, loads samples from a file, uploads the samples to the generator and plays the data.

7.4.7   sdr14tx_toneoutput.py

Interface language

Python3

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

SDR14TX

Suitable as example for

SDR14TX, SDR14

Demonstrates features

Finding devices, configuring DAC data.

Overview

Sets up the generator, synthesizes a sine wave through Python code, uploads the samples to the generator and plays the data.

7.4.8 sdr14tx_Playlist_example.py

Interface language

Python3

Complexity level

Medium

Build environment

Windows, Linux

Valid for product(s)

SDR14

Suitable as example for

SDR14TX, SDR14

Demonstrates features

Finding devices, configuring DAC data, playlist mode.

Overview

Sets up the generator, synthesizes data in code, uploads the samples to the generator in playlist mode and plays the data. Uses the analog input of the SDR14 (not available on SDR14TX) to record data and plots the data.

Detailed description

This script showcases in Python:

- How to connect to ADQ devices in Python

- Upload of waveforms to the SDR14TX

- Using a playlist on the SDR14TX

- How to setup an acquisition of data

- How to read data by `GetData()` API in Python

- How to plot data in Python

7.5   MATLAB Language examples

> **❶ Important**
>
> The ADQ3 series of products, ADQ32, ADQ33 and ADQ36 do not have support for MATLAB.

7.5.1   ADQ7_ts_example_script.m

Interface language

　　MATLAB

Complexity level

　　Basic

Build environment

　　Windows

Valid for product(s)

　　ADQ7

Suitable as example for

　　ADQ7

Demonstrates features

　　Finding device, setting trigger modes, triggered streaming acquisition.

Overview

　　Sets up the digitizer in a state controlled by code definitions and acquires records continuosly.

7.5.2   ADQ14_ts_example_script.m

Interface language

　　MATLAB

Complexity level

　　Basic

Build environment

　　Windows

Valid for product(s)

　　ADQ14

Suitable as example for

　　ADQ14

Demonstrates features

Finding device, setting trigger modes, triggered streaming acquisition.

Overview

Sets up the digitizer in a state controlled by code definitions and acquires records continuosly.

7.5.3   ADQ14_example_script.m

Interface language

MATLAB

Complexity level

Basic

Build environment

Windows

Valid for product(s)

ADQ14

Suitable as example for

ADQ14

Demonstrates features

Finding device, setting trigger modes, multi-record acquisition.

Overview

Sets up the digitizer in a state controlled by code definitions and acquires records as configured.

7.5.4   SDR14_AWG_Playlist_example.m

Interface language

MATLAB

Complexity level

Basic

Build environment

Windows

Valid for product(s)

SDR14TX

Suitable as example for

SDR14TX

Demonstrates features

> Finding device, defining waveforms and playlist, uploading and play.

Overview

> Sets up the waveform generator with waveforms and a playlist of sequencing and starts the sequence.

7.6 C# Language examples

7.6.1 ADQAPI_CSharp_example

Interface language

> C# (.NET)

Complexity level

> Basic

Build environment

> Windows, Linux

Valid for product(s)

> ADQ14, ADQ7, ADQ12, ADQ214, ADQ412, SDR14

Suitable as example for

> ADQ14, ADQ7, ADQ12, ADQ214, ADQ412, SDR14

Demonstrates features

> Finding devices, setting trigger modes, multi-record acquisition.

Overview

> Sets up the digitizer in a state controlled by compiler definitions, acquires a record and outputs the record samples to a file.

7.7 VisualBasic Language examples

> **⚠ Warning**
>
> For applications where performance is important (in terms of transfer and processing speed of acquired data) it is strongly recommend to avoid Visual Basic entirely, due to the limitations it entails.

7.7.1 ADQAPI_VisualBasic_Example

Interface language

> VB (.NET)

Complexity level

Basic

Build environment

Windows, Linux

Valid for product(s)

ADQ14, ADQ7, ADQ12, ADQ214, ADQ412, SDR14

Suitable as example for

ADQ14, ADQ7, ADQ12, ADQ214, ADQ412, SDR14

Demonstrates features

Finding devices, setting trigger modes, multi-record acquisition.

Overview

Sets up the digitizer in a state controlled by code definitions, acquires a record and outputs the record samples to a file.

References

[1] Teledyne Signal Processing Devices Sweden AB, *14-1367 ADQAPI and dotNet User Guide*. Technical Manual.

[2] Teledyne Signal Processing Devices Sweden AB, *14-1351 ADQAPI Reference Guide*. Technical Manual.

[3] Teledyne Signal Processing Devices Sweden AB, *20-2465 ADQGen3 Streaming User Guide*. Technical Manual.

[4] Teledyne Signal Processing Devices Sweden AB, *20-2521 ADQAssist User Guide*. Technical Manual.

[5] Teledyne Signal Processing Devices Sweden AB, *19-2245 Application Note: System transfer test*. Technical Manual.

Worldwide Sales and Technical Support

spdevices.com

Teledyne SP Devices Corporate Headquarters

Teknikringen 8D

SE-583 30 Linköping

Sweden

Phone: +46 (0)13 645 0600

Fax:　　+46 (0)13 991 3044

Email:　spd_info@teledyne.com