TELEDYNE SP DEVICES
Everywhereyoulook™

ADQ7-FWATD

User Guide

Author(s): Teledyne SP Devices
Document ID: 17-1957
Classification: Public
Revision: PA11
Print date: 2019-04-04

Contents

1 Introduction

This document is the user guide of the ADQ7 digitizer running the advanced time-domain firmware (FWATD) option. For information on how to manage the firmware images of your device, please refer to the ADQUpdater user guide [1].

Release R190304

- The trigger blocking window length is now a 64-bit value.
- Introduced the API function `ResetTimestamp()` to set the digitizer's internal timestamp to zero.
- The accumulation grid can now be synchronized by an external signal. Refer to Section 4.5.4 for details.
- Select GPIO pins now have dedicated functions that can be configured via `SetFunctionGPIOPort()`. Refer to Section 4.9 for details.

1.1 Definitions and Abbreviations

Table 1 lists the definitions and abbreviations used in this document and provides an explanation for each entry.

Table 1: Definitions and abbreviations used in this document.

Term	Explanation
1CH	ADQ7 in one-channel mode (10 GSPS)
2CH	ADQ7 in two-channel mode (5 GSPS)
ADC	Analog-to-digital converter
AFE	Analog front-end
API	Application programming interface
ATD	Advanced time-domain (firmware option)
DRAM	Dynamic random access memory
FIR	Finite impulse response
FW	Firmware (digitizer feature set)
GSPS	10^9 samples per second
RAM	Random access memory
SDK	Software development kit
WFA	Waveform accumulation/accumulator

2 Features

This section presents the specification of ADQ7-FWATD along with brief descriptions of some of its core features and limitations. Detailed information may be found in Section 4.

2.1 Specification

The specification for the advanced time-domain firmware is presented in Table 2. For the general specification of the ADQ7 digitizer, please refer to the ADQ7 data sheet [2].

The digitizer can be run in two main modes: one-channel mode (at 10 GSPS) and two-channel mode (at 5 GSPS). Switching between these two modes requires changing the firmware image. The ADQUpdater user guide [1] provides details on how to perform the switch.

2.2 Overview

Fig. 1 presents a block diagram outlining the main features of the data path.

- **ADC**

  The data is sampled at 10 GSPS (one-channel mode) or at 5 GSPS (two-channel mode).

- **Configurable FIR filter**

  A configurable linear phase FIR filter of order 16 is available and may be used for noise reduction and bandwidth control. The function is described in detail in Section 4.2.

- **Sample skip**

  The full-rate data stream may be downsampled by activating the sample skip functionality. This function is usd to increase the measurement time, e.g. specifying a *sample skip factor* of two translates to a doubling of the maximum observable time (now at 400 μs). Refer to Section 4.3 for additional details.

- **Advanced threshold function**

  The data may be subjected to an advanced thresholding function, isolating weak events and effectively increasing the dynamic range of the system.

- **WFA (firmware/software)**

  The waveform accumulator resides partly in firmware and partly in software (inside the ADQAPI). This structure offers flexibility and capabilities beyond what can be achieved by using the accumulators separately.

3 Getting Started

This section provides information on how to interface with ADQ7 and the ATD firmware. Table 3 describes the peripherals and their usage. Note that only the INX input should be used when the ADQ7 is running in the one-channel mode.

Table 2: ADQ7-FWATD specification

Item	Min	Typical	Max	Unit
Data path filter				
Filter type		Linear phase FIR		
Filter order		16		
Coefficient width		16		bits
Coefficient fractional width		14		bits
Coefficient value	-2		$2 - 2^{-14}$	
Coefficient precision		2^{-14}		
Waveform accumulator				
Waveform length (1CH) [2]	32		2 097 152	samples
Waveform length (2CH) [2]	16		1 048 576	samples
Waveform length	3.2 *n*		200 μ	s
Length granularity (1CH)	32			samples
Length granularity (2CH)	16			samples
Length granularity	3.2 *n*			s
Waveform dead time			32	ns
Accumulation dead time			32	ns
Number of accumulations	1		262 144[1]	
Accumulation granularity	1			
Advanced threshold filter				
Filter type		Linear phase FIR		
Filter order		16		
Coefficient width		16		bits
Coefficient fractional width		14		bits
Coefficient value	-2		$2 - 2^{-14}$	
Coefficient precision		2^{-14}		

[1] Guaranteed safe scaling (32-bit accumulator).
[2] See Section 4.5.2

3.1 SDK Installation

The Software Development Kit (SDK) contains the ADQAPI, drivers, examples and documentation required for successfully interfacing with the digitizer. The installation procedure for Microsoft Windows

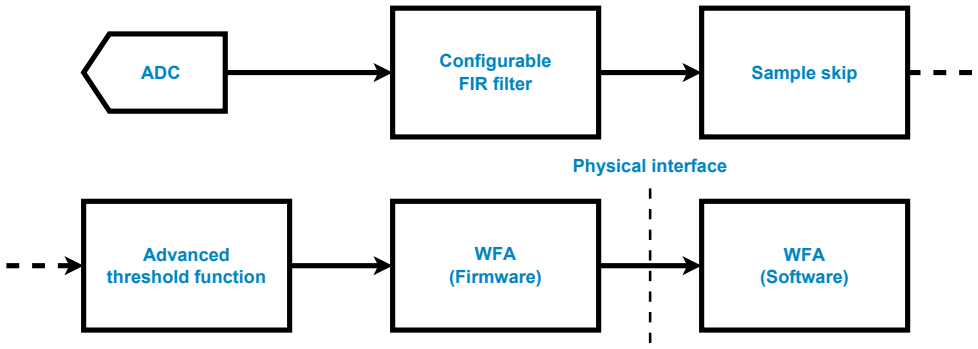

Figure 1: A block diagram presenting the main features of the ADQ7-FWATD data path.

Table 3: Peripheral connections on ADQ7.

Connector	Description
TRIG	External trigger input / output
SYNC	Sync trigger input / output
INX	Channel A input (one-channel mode)
INA	Channel A input (two-channel mode)
INB	Channel B input (two-channel mode)
CLK	External reference clock input

and Linux is described in the following sections.

3.1.1 Installing the SDK (Windows)

For Microsoft Windows the SDK is installed by running

```
ADQ-setup_rXXXXX.exe
```

where `XXXXX` is the version number. After the installation the example code is located in

```
<Path to installation directory>/C_examples/
```

and the documentation in

```
<Path to installation directory>/Documentation/
```

3.1.2 Installing the SDK (Linux)

The SDK is supported for a number of Linux distributions and versions. The required files are all included in

```
ADQ_SDK_linux_rXXXXX.tar.gz
```

where `XXXXX` is the version number. The archive contains the SDK installation files, example code and documentation. The `README` file, located in the root directory of the archive, describes the installation procedure in detail for the different distributions. The content of this file is also listed in Appendix A.

3.1.3 Unsupported Software Features

The provided SDK is used across a wide range of products. Therefore, some of the software is not supported by ADQ7-FWATD, namely

- LabView is not supported.
- ADCaptureLab (Windows only application) is not supported.
- ADQUpdaterGUI (Windows only application) is not supported.

Important

ADQ7-FWATD is not supported by LabView, ADCaptureLab or the ADQUpdaterGUI.

3.2 Collecting Data

The FWATD example in C is provided together with the SDK. For Windows it is located in

```
<Path to installation directory>/C_examples/ADQAPI_FWATD_example
```

after installing the SDK, and for Linux in the

```
examples/fwatd_example
```

directory of the SDK archive. This example illustrates the capabilities of ADQ7-FWATD, and can be used as-is. However, the example is primarily intended to be used as a guide for creating use case specific applications.

4 Detailed System Description

This section provides additional details on important system features.

4.1 Trigger Features

This section describes features relating to the trigger interface: trigger sources, blocking triggers, generating synchronization signals to external equipment and resetting the internal timestamp of the digitizer.

There are four available trigger sources: software trigger, level trigger, internal trigger and external trigger. The trigger source is selected with `SetTriggerMode()`. Please refer to the ADQAPI reference guide [3] for information on the argument list.

4.1.1 Software Trigger

The software trigger is issued by calling the API function `SWTrig()` and its main purpose is debugging. One call to the function will generate exactly one trigger event for the digitizer. Multiple calls to the function in rapid succession will *not* result in a trigger pattern with a well-defined period due to the non-real-time behavior of most modern operating systems.

4.1.2 Level Trigger

The level trigger takes a 16-bit ADC code and an edge polarity as inputs and considers the first sample at or after threshold has been crossed as the trigger point. To guard against false triggers caused by noise, a *reset level* has to be specified. This reset level is specified in absolute ADC codes and is used to arm the level trigger mechanism by requiring that the data visits ADC codes below/above this level before a new rising/falling trigger event may be generated. The level trigger achieves a resolution equal to the sample rate.

4.1.3 Internal Trigger

The internal trigger period may be configured with sample resolution using `SetInternalTriggerPeriod()`. By default, the internal trigger is free running and triggers are normally gated with the device arming mechanism. However, the function `SetInternalTriggerSyncMode()` allows synchronization of the internal trigger to some external event (detected on the TRIG connector).

4.1.4 External Trigger

The digitizer detects external trigger events on the TRIG connector with sample precision. The DC-coupled input accepts a signal in the range $[-0.5, 3.3]$ V and the detection threshold level may be configured using `SetTriggerThresholdVoltage()`.

4.1.5 Timestamp Synchronization

The timestamp synchronization feature allows the user to use an external signal to reset the time base of the digitizer. This function is used to establish a common time base across multiple digitizers or to relate the time base to some external event, useful in applications involving some repeated process.

The timestamp synchronization engine listens for external events on either the external trigger connector (TRIG), the synchronization connector (SYNC) or on one of the dedicated GPIO pins. The selected source will be referred to as the *timestamp synchronization source* throughout this section. Each trigger source has an associated edge specification to select if the rising and/or falling edges should generate trigger events. This global edge specification is used by the timestamp synchronization engine and may be set by calling `SetTriggerEdge()`.

Controlling the timestamp synchronization feature involves three functions in the ADQAPI: `SetupTimeStampSync()`, `ArmTimeStampSync()` and `DisarmTimeStampSync()`. The timestamp synchronization feature can be configured in two modes:

First event *Mode index 0*

Once the timestamp synchronization engine is armed, the first event observed on the timestamp synchronization source will reset the timestamp.

Every event *Mode index 1*

Once the timestamp synchronization engine is armed, all subsequent events observed on the timestamp synchronization source will reset the timestamp.

The timestamp synchronization feature can be queried by the user for the number of synchronization events by calling `GetTimestampSyncCount()`. Additionally, the information is included in the record header and represent the counter's value at the time the record was triggered. This allows the user to keep track of the number of times the timestamp has been reset and thus the data from two batches, separated by one or several resets of the timestamp, are able to be related to each other in time. The counter is reset when `DisarmTimestampSync()` is called.

Note

The timestamp synchronization counter counts the number of times the timestamp has been reset to zero from a synchronization event.

There is also an option to reset the timestamp to zero via the API call `ResetTimestamp()`. This function has no guarantees on timing, i.e. if `ResetTimestamp()` is called in the user application and at the same time a trigger is generated, the timestamp of that record *will* vary between program executions.

4.1.6 Trigger Blocking

The trigger blocking feature generates a blocking window that may be used together with the timestamp synchronization feature to ensure that the digitizer does not create records with unsynchronized timestamps.

The trigger blocking engine listens for external events on either the external trigger connector (TRIG), the synchronization connector (SYNC) or on one of the dedicated GPIO pins. The selected source will be referred to as the *trigger blocking source* throughout this section. The trigger blocking engine uses the global edge specification set by calling `SetTriggerEdge()`.

Controlling the trigger blocking feature involves three functions in the ADQAPI: `SetupTriggerBlocking()`, `ArmTriggerBlocking()` and `DisarmTriggerBlocking()`. The blocking function can be configured in four modes:

Once *Mode index 0*

Once the trigger blocking engine is armed, this mode inhibits the creation of records until the first event on the trigger blocking source has been observed. At this point, the blocking is disengaged and all subsequent triggers are allowed to propagate.

Window *Mode index 1*

Once the trigger blocking engine is armed, this mode inhibits the creation of records until an event on the trigger blocking source has been observed. Following an event, the blocking is disengaged for a fixed duration (the *window length*) and triggers are allowed to propagate. Any additional

events on the trigger blocking source that occurs during the window are ignored, i.e. they do not reset the window.

Gate *Mode index 2*

Once the trigger blocking engine is armed, this mode inhibits the creation of records until an event on the trigger blocking source has been observed. Following an event, the blocking is disengaged for as long as the trigger blocking source signal remains 'logic high' or 'logic low' (depending on the source's edge specification) and is reengaged once an event of the opposite type is detected.

Inverse window *Mode index 3*

Once the trigger blocking engine is armed, this mode allows triggers to propagate until an event on the trigger blocking source has been observed. Following an event, the blocking is engaged for a fixed duration (the *window length*), preventing triggers from propagating and creating records during this time.

4.1.7 Arming Order

The order in which the different trigger-related features are armed is important. Generally, the digitizer should be armed in an *outwards* direction, meaning that the trigger blocking engine should be one of the last engines to be armed. The direction is determined by how the engines are connected to each other. The trigger blocking engine (`ArmTriggerBlocking()`) modifies the stream of triggers that the acquisition engine (`StartWFA()`) can detect, thus creating a direction which must be respected. The recommended program flow is established the user application example (see Section 5).

4.1.8 Synchronization Signals

In applications where the digitizer is used to generate the trigger signal to other equipment, the method of choice is to use the internal trigger and output this periodic pulse on the external trigger connector (TRIG). The frame synchronization feature introduces an additional signal synchronized to the internal trigger which may be output on the synchronization connector (SYNC) or on its dedicated GPIO pin. This signal acts as a frame for the free-running trigger signal and has three configuration parameters: the *frame length*, the *frame factor* and the internal trigger *edge* to use as the reference point for the frame. These parameters are specified by calling the function `SetupFrameSync()`. Additionally, the output has to be activated by calling `EnableFrameSync()`.

The frame generator listens to the internal trigger signal counting the number of edges matching the edge type specified by the user. Once this number is equal to the frame factor, the output is pulled high for a duration equal to the frame's length. Fig. 2 presents an example scenario.

4.2 Configurable FIR Filter

The data path contains an order 16 linear phase FIR filter. This filter is placed before the *sample skip* function which, if combined, enables proper decimation of the data stream.

The filter coefficients use a 16-bit 2's complement representation with 14 fractional bits. Thus, the representable range is

$$[-2,\ 2 - 2^{-14}].$$

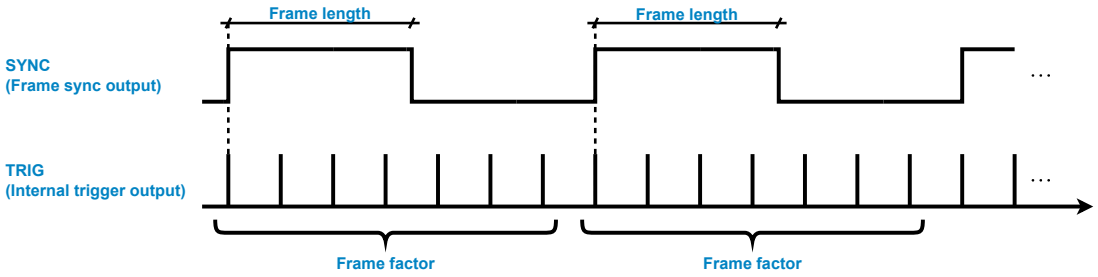

Figure 2: An illustration of the frame synchronization feature. In this scenario, the frame factor is set to 7.

The properties of the filter yields a symmetric impulse response with the point of symmetry at coefficient $h(8)$. The API function `SetUserLogicFilter()` is used to update the filter coefficients and expects exactly 9 coefficients—up to and including the point of symmetry.

It is possible to specify the coefficients using a floating point representation. However, this is a feature provided for convenience and values will still be converted to fixed-point numbers before being transferred to the device. The result of the rounding is visible in the device's trace log file. Please refer to the function's entry in the ADQAPI reference guide [3] for more information.

Important

For convenience, the filter coefficients may be specified using a floating-point representation. However, these value are subjected to rounding as dictated by the argument `rounding_method` and converted to the filter's fixed-point representation.

The filter output is subjected to symmetric rounding to 16 bits with rounding *away from zero* as the tie-breaking rule, i.e. $0.5 \approx 1$ and $-0.5 \approx -1$. Additionally, the output is saturated if the value is not representable by the 16-bit 2's complement representation used by the data.

4.3 Sample Skip

The sample skip function allows downsampling of the full-rate data stream. The set of allowed sample skip factors depends on the firmware channel configuration. The one-channel mode supports skip factors in the set

$$\{1, 2, 4, 8, 16, 32, 33, \ldots, 65536\}$$

and the two-channel mode supports skip factors in the set

$$\{1, 2, 4, 8, 16, 17, \ldots, 65536\}.$$

A skip factor of '1' implies no downsampling and is the default value.

4.4 Advanced Threshold

The advanced threshold function consists of a programmable FIR filter and decision logic aimed to single out weak pulses from the surrounding noise. A block diagram of the function is presented in Fig. 3. Refer to the firmware data sheet [4] for more information about the feature.

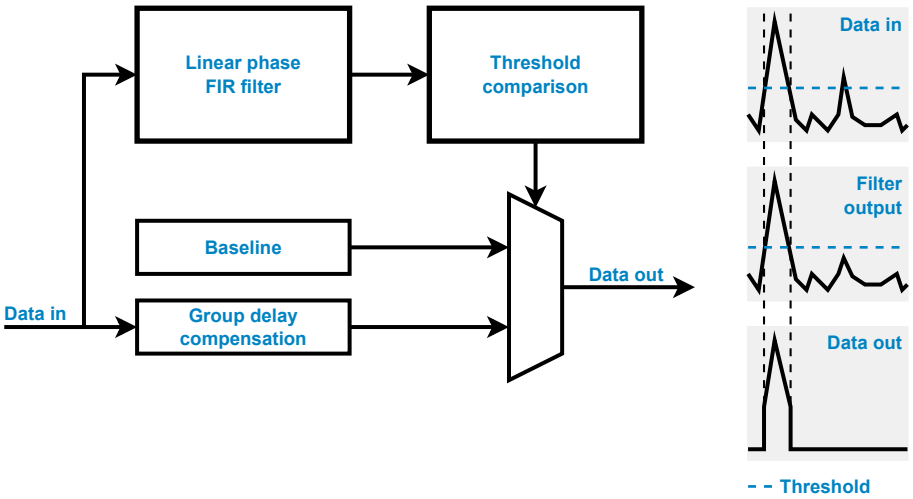

Figure 3: Block diagram of the advanced threshold function.

4.4.1 Threshold Logic

The threshold function consists of a multiplexer which selects between the *baseline* and the original data on a sample-by-sample basis. The decision logic compares the user-defined *threshold* value to the filter output and depending on the *polarity*, determines if the sample should be replaced by the baseline or remain unchanged.

The data is substituted by the baseline value if the relation in (1) corresponding to the current polarity is met.

$$\begin{aligned} y(n) < T \quad &\text{for } \textit{positive} \text{ polarity} \\ y(n) > T \quad &\text{for } \textit{negative} \text{ polarity} \end{aligned} \tag{1}$$

In (1), $y(n)$ is the filter output and T is the user-defined threshold value. The parameters of the advanced threshold feature are specified using the function `ATDSetupThreshold()`.

4.4.2 Filter

The threshold filter is a linear phase FIR filter of order 16. The filter coefficients use a 16-bit 2's complement representation with 14 fractional bits. Thus, the representable range is

$$[-2,\ 2 - 2^{-14}].$$

The coefficients are specified using the function `ATDSetThresholdFilter()`.

The filter output is subjected to symmetric rounding to 16 bits with rounding *away from zero* as the tie-breaking rule, i.e. $0.5 \approx 1$ and $-0.5 \approx -1$. Additionally, the output is saturated if the value is not representable by the 16-bit 2's complement representation used by the data.

Note

The filter output is subjected to symmetric rounding with rounding away from zero as the tie-breaking rule and saturation if the value lies outside the representable range.

Because of the properties of the filter, the impulse response is symmetric around coefficient $h(8)$ and only coefficients up until that point are needed. Thus, the coefficient array argument in `ATDSetThreshold-Filter()` will attempt to access 9 elements. If the allocated memory is less than expected, the call will result in an memory access violation.

Important

The coefficient array argument in the call to `ATDSetThresholdFilter()` is expected to contain at least 9 elements, otherwise a memory access violation will occur.

4.5 Waveform Accumulator

The WFA on ADQ7-FWATD is *partitioned*, meaning that the work load is split between the digitizer and the host computer. The WFA is composed of a 32-bit accumulator running in the digitizer's FPGA working in tandem with a second 32-bit accumulator, implemented in software and running in a separate thread from the user application. Data is delivered to user space through a queue interface where the user is responsible for allocating memory for the accumulated records and keeping the queue of available memory from running out. Fig. 4 presents a block diagram of the ADQ7-FWATD structure, starting from the digitizer and ending in user space.

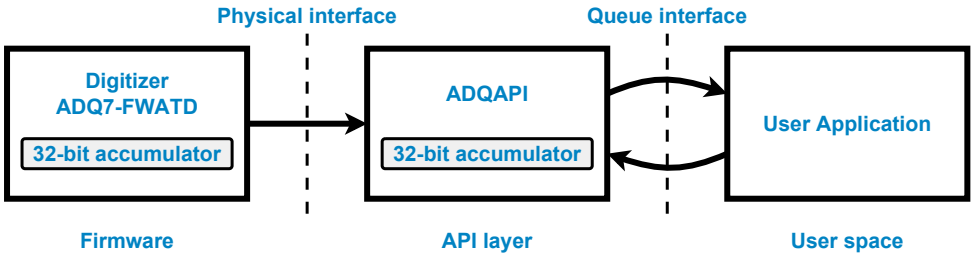

Figure 4: A simple block diagram of the ADQ7-FWATD structure.

The ADC data is 14-bit, yielding codes in the range

$$[-8192,\ 8191].$$

Thus, the 32-bit accumulator guarantees that the ADC code with the largest absolute value, i.e. 8192, can safely be accumulated 262 144 times without an arithmetic overflow occurring.

4.5.1 Partitioning

The user provides a desired number of accumulations, i.e. a number of records following each other in time which should be accumulated and presented as one single record in user space. This number is passed through a partitioning algorithm to determine how to divide the work load between the digitizer and the host computer. The algorithm attempts to divide the work load so as to perform as many accumulations as possible in hardware. However, this complex decision making process depends on many factors, several of which are beyond the scope of this document. For example, the user defined record length directly affects the partitioning. Since the on-board DRAM is finite in size, shorter records will enable a higher number of accumulations to be performed in hardware while longer records shifts the boundary in the other direction.

4.5.2 Record Length

The WFA engine requires that the record length divided by 32 has at least one prime factor in the range $[2, 512]$. That is the intersection

$$P_{N/32} \cap [2, 512] \tag{2}$$

where P_X is the prime factors of X and N is the record length in samples, should not be empty. This means that some of the record lengths will not be supported. If such a record length is specified `ATDSetupWFA` will return 0 and an error message will be printed to the log.

Example

- **Record length 471 904**: $471904/32$ has the factor 14747. This record length **is not** supported since no factors are less than 512.
- **Record length 471 936**: $471936/32$ has the unique factors $2, 3, 1229$. This record length **is** supported since at least one factor is smaller than 512.

4.5.3 Run-time Reconfiguration

Changing the number of accumulations while a measurement is ongoing is possible by calling the ADQ-API function `ATDUpdateNofAccumulations()`. When the function is called, the device will update the accumulation grid at the next grid point in the current grid.

Note

The *grid* refers to the accumulation grid which is equivalent to the trigger grid downsampled with a factor equal to the number of accumulations. For example, the accumulation grid period is

$$\frac{N_A}{f_{\text{trig}}}$$

in a system using a constant trigger frequency f_{trig} and N_A number of accumulations.

Timing is not guaranteed for this function, i.e. when the function is called, an unknown number of records with the previous number of accumulations will propagate to the user space before the changes are reflected in the received data.

There are two conditions which need to be met every time the user wishes to alter the accumulation grid:

1. The WFA is running, i.e. a call to `ATDUpdateNofAccumulations()` may only occur after a call to `ATDStartWFA()` and before a call to `ATDStopWFA()`.
2. Any previous changes must have taken effect.

The meaning of *taken effect* is that the changes have been applied to the WFA engine and at least one record reflecting the new settings has propagated through the system and been observed by the ADQAPI.

If any condition is not met, the function will return with a negative return value indicating what went wrong. Refer to the corresponding function description in the ADQAPI reference guide [3] for information on how to interpret the return value.

4.5.4 Synchronizing the Accumulation Grid

In some applications the digitizer may be set up for N_A accumulations but the source cannot reliably guarantee that the grid can be maintained by only counting triggers. For example, the source may occasionally generate more than N_A triggers in a burst and other times fewer. To handle these scenarios, the accumulation grid (explained in Section 4.5.3) can be resynchronized by an external signal. This feature is tied to the trigger blocking engine (see Section 4.1.6) and will use events on the *trigger blocking source* to synchronize the grid.

By default, the grid synchronization is disabled and activation requires the user to call `ATDEnableAccumulationGridSync()`. Once the feature is activated the only additional requirements are that the trigger blocking engine is configured and armed.

Note

The accumulation grid synchronization feature shares its event source with the trigger blocking engine (see Section 4.1.6).

When the *window* mode is used by the trigger blocking engine, the window length *cannot* exceed the expected period of the grid synchronization events since events occurring inside a window are ignored.

Important

The trigger blocking window length cannot exceed the expected period of the grid synchronization events.

When the grid is synchronized, the next raw record that follows will mark the start of the new grid. This record will thus be the first raw record in the resulting user space record and its timestamp will propagate to the data struct (see Section 4.6).

It is safe to synchronize the grid at any point during an acquisition but each time the action will cause a transient behavior in user space. This behavior is well-defined and regulated by the robustness mechanism (see Section 4.7), more specifically—the way the system handles an overflow. Please refer to Section 4.7.1 to understand the possible outcomes.

Important

This feature *cannot* be used to define some maximum number of accumulations N_A and expect any number of triggers $< N_A$ to generate user space records reflecting this lower number of accumulations.

Example

Fig. 5 presents a timing diagram demonstrating the accumulation grid synchronization in three different cases:

- the incoming triggers exactly match the number of accumulations (region 0),
- the incoming triggers fail to reach the number of accumulations (region 1) and

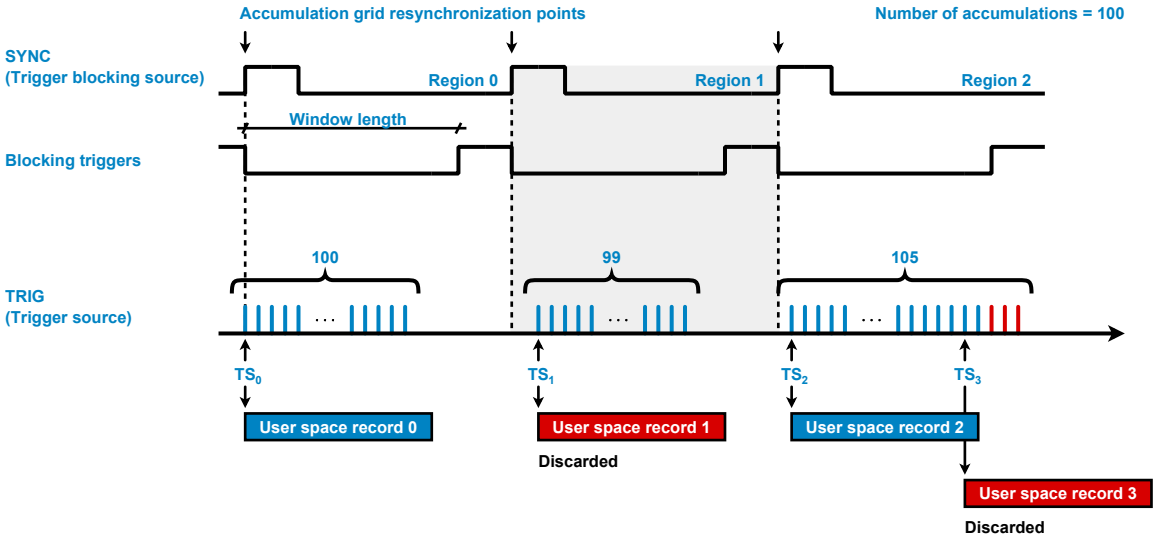

Figure 5: A timing diagram demonstrating the accumulation grid synchronization feature. There are three regions, initiated by an event on the trigger blocking source, where the incoming triggers exactly match, fail to reach or exceed the number of accumulations $N_A = 100$, respectively.

- the incoming triggers exceed the number of accumulations (region 2).

The WFA is configured to carry out 100 accumulations ($N_A = 100$) and the trigger blocking engine is set up in the *window* mode with a window length set to a value that covers the worst-case time range range for the source to output a trigger frame. The SYNC connector is chosen as the trigger blocking source with its edge polarity set to *rising* and the accumulation grid synchronization feature is activated.

Region 0

The first region starts at the first rising edge of the SYNC signal. At this point, the accumulation grid is resynchronized and the first trigger that follows marks the start of user space record 0. The timestamp of this record, TS_0, will propagate to the user via the data struct. The region contains exactly 100 triggers which all fall inside of the trigger blocking window.

Region 1

The second region starts at the second rising edge of the SYNC signal. Once again, the accumulation grid is resynchronized and the first trigger that follows marks the start of user space record 1. The timestamp is TS_1 and taken exactly at the point where the first raw record in the region is triggered. However, this time there are only 99 triggers which means that the requirement of $N_A = 100$ is not satisfied. This record will not propagate to user space but the actual decision is deferred to the start of region 2, where any ongoing (incomplete) accumulation is discarded.

Region 2

The third region starts at the third rising edge of the SYNC signal. The accumulation grid is resynchronized causing the ongoing accumulation of user space record 1 to be discarded. The first trigger that follows marks the start of user space record 2 (timestamp TS_2). The region contains

105 triggers in total which means that the first 100 will be used to complete user space record 2 and the one following will mark the start of user space record 3 (timestamp TS_3). A total of two raw records fall inside the trigger blocking window before it is closed causing the remaining three triggers to be ignored by the digitizer. Similar to region 1, user space record 3 will be discarded the next time the grid is synchronized.

4.6 Data Struct

The ADQ7-FWATD data struct is used to represent a record in user space and is used by all FWATD-specific functions associated with handling data. The struct offers an intuitive abstraction of all parts which together constitute a record: the metadata fields as well as the actual waveform data.

The struct is 32 B (after padding) and consists of the fields presented in Table 4 where the first table entry starts at byte zero.

Table 4: Contents of the ADQ7-FWATD data struct.

Field name	Type	Description
Timestamp	`uint64_t`	The timestamp of the first raw record in the batch.
Data	`int32_t *`	Pointer to memory where the record data is located.
RecordNumber	`uint32_t`	The record number.
Status	`uint32_t`	System status at the point where the record was collected. See Section 4.7.2 for details.
RecordsAccumulated	`uint32_t`	Number of accumulated records reflected in the data.
Channel	`uint8_t`	The originating channel.

4.7 Robustness

ADQ7-FWATD features a data discarding mechanism to deal with the many different types of overflow that may occur. The core concept is to discard data in a well-defined manner such that the overall accumulation grid is preserved and corrupted records are prevented from propagating to the user space.

The digitizer firmware is engineered as a well-balanced system, meaning no overflows can occur due to internal effects. The first and most critical point in the system where overflows can occur is the device-to-host interface, which connects the digitizer to the host computer. This connection is beyond the complete control of the digitizer and determines the effective data transfer rate which directly translates to the limits of the WFA settings. The digitizer on its own is a real-time system, but once the host computer is included in the system view, this is no longer true as it is at the discretion of the operating system to deliver the data to the user application.

Note

The most critical point is the device-to-host interface. The maximum data transfer rate translates into the limits of the WFA settings.

The second critical point in the system is the queue interface which provides the user with accumulated records as long as there is available memory. If there is no free memory to place the newly transferred data, the data flow will stall and eventually cause data to be discarded.

Note

Another critical point is the queue interface which takes user provided references to memory and returns the accumulation results.

Please be aware that the number of buffers needed to ensure stable operation is highly dependent on the use case since the buffers circulate in the queue interface at a rate determined by the user application. For example, 20 buffers may be enough to allow stable operations when writing the incoming records to disk using a binary format while 50 may be required if instead, the data is formatted as ASCII.

There are strategies available to design the user application in such a way that 20 queued buffers will be sufficient in both cases.

Note

The number of queued buffers needed ensure stable operation of the system is highly dependent on buffer handling in the user application.

4.7.1 Overflow Behavior

An overflow is caused by a stall of the data transfer interface for an extended period of time. This may in turn be caused by an imbalance between the transfer bandwidth of the device-to-host interface and the output data rate of the digitizer.

During an overflow, data collected up until that point (waiting to be transferred) remains intact and incoming data is discarded in a well-defined manner. What *well-defined* means in detail is beyond the scope of this document. However, the resulting user space behavior is worth commenting on. An overflow will manifest itself in two possible ways, depending on the WFA settings and current work load partitioning. Both of these events are completely discernible to the user by reading the record header information.

1. A user space record will contain fewer number of accumulated waveforms than the defined number of accumulations.
2. One or several user space records will be missing completely.

What is guaranteed not to happen is the corruption of data, e.g. that some sections of a user space record are the result of X accumulated waveforms while Y is the number of accumulated waveforms for other sections.

A different type of overflow occurs if the trigger rate is not well-matched to the record length. For example, if the trigger period is 8 μs and the record length is 10 μs, the digitizer will still be recording data for the previous record when the new trigger arrives. In this case the trigger is simply ignored, causing the effective trigger period to be 16 μs.

Note

Triggers occurring while the previous record is not yet completed are ignored by the digitizer.

4.7.2 Status Codes

The status value reported in the data struct (Table 4) provides information on the overall health of the WFA. The exact meaning of the value can be interpreted using Table 5.

Table 5: Status codes for the WFA.

Bit	Description
3	Arithmetic overflow. One or several samples inside this record have been saturated as result of an arithmetic operation yielding a value which could not be represented.
2	Data has been lost within the record. The header field 'RecordsAccumulated' is expected to report a lower value than the configured number of accumulations.
1	Buffer queue starving. When the ADQAPI requested a new buffer to place this record, none were available, causing a stall of the device-to-host interface.
0	Records have been lost. The header field 'RecordNumber' is expected to reflect that one or several records have been discarded.

For example, a status value of `0x3` indicates that one or several preceding records have been discarded by the robustness mechanism. Additionally, the first attempt by the ADQAPI to retrieve a reference to available memory failed, causing a stall. The stall occurs in the queue interface in Fig. 4 due to the user not having registered enough buffers.

4.8 Collection Modes

ADQ7-FWATD features two collection modes: single-shot and streaming. The modes are geared towards different use cases, however the streaming mode is more general and is the recommended mode of operation due to its relaxed memory requirements.

Note

The streaming mode is the recommended collection mode due to its relaxed memory requirements.

Common for both modes is that the user is responsible for allocating all the required memory at the start of the application and later returning the memory once the application is about to close.

Important

The user is responsible for managing the memory used for data handling. At no point during collection does the ADQAPI assume ownership of memory.

Refer to the ADQAPI reference guide [3] and the function `ATDStartWFA()` for details on how the different modes are activated.

4.8.1 Single-shot Mode

The single-shot mode is intended for quick set up and capture of a sequence of events with well-defined defined start and stop points and, more importantly, where it is not too costly to hold *all* the measurement data in RAM.

The mode is not suitable for use cases where the user application uses the data to react in some way, e.g. by automatically changing the experiment parameters, since the data is returned to the user once the measurement has completed.

Since the single-shot mode is quite straight-forward to grasp and use, most of the discussion in this document is centered around the streaming mode. Any mention of *queues* should indicate to the reader that the streaming mode is being discussed.

4.8.2 Streaming Mode

The streaming mode is centered around the use of a queue interface to transfer data between the device and the user, see Fig. 4. The user is responsible for keeping the queue filled with references to available memory. The reuse of memory locations enables an unlimited amount of data using a limited amount of memory.

Additionally, due to the continuous nature of the streaming mode, it is well suited for applications which require a feedback mechanism to control devices placed earlier in the signal chain. However, the loop delay cannot be given a constant value and will in practice depend on several factors, many of which are beyond the control of the digitizer since the host computer is rarely using a real-time OS.

Data Collection Flowchart

Fig. 6 presents a flowchart of the data collection loop in the streaming mode. Entering the loop, the digitizer is assumed to be configured according to the user's specification in regards to the trigger configuration, clock source, filter coefficients etc. In the text below *buffer* is synonymous to a *user space record* but specifically means the host memory allocated to hold the data.

1. The first step in the outer loop is to set up the WFA engine by calling `ATDSetupWFA()`, specifying the horizontal settings of the acquisition, i.e. the record length and any pretrigger or trigger delay values. Additionally, the number of accumulations per user space record as well as the number of times to repeat the accumulation process, i.e. the number of user space records, is supplied as arguments to this function. Furthermore, the memory allocated for the target buffers are registered in the queue by calling `ATDRegisterWFABuffer()`.

2. At this point, the WFA engine is primed and ready to collect data. The acquisition process is started by calling `ATDStartWFA()`, arming the device. This function call launches a thread in the ADQAPI which assumes control of the digitizer. Therefore, from this point on, it is crucial that the user only calls supported *thread-safe* functions. Please refer to Section 4.12 for further details on threading.

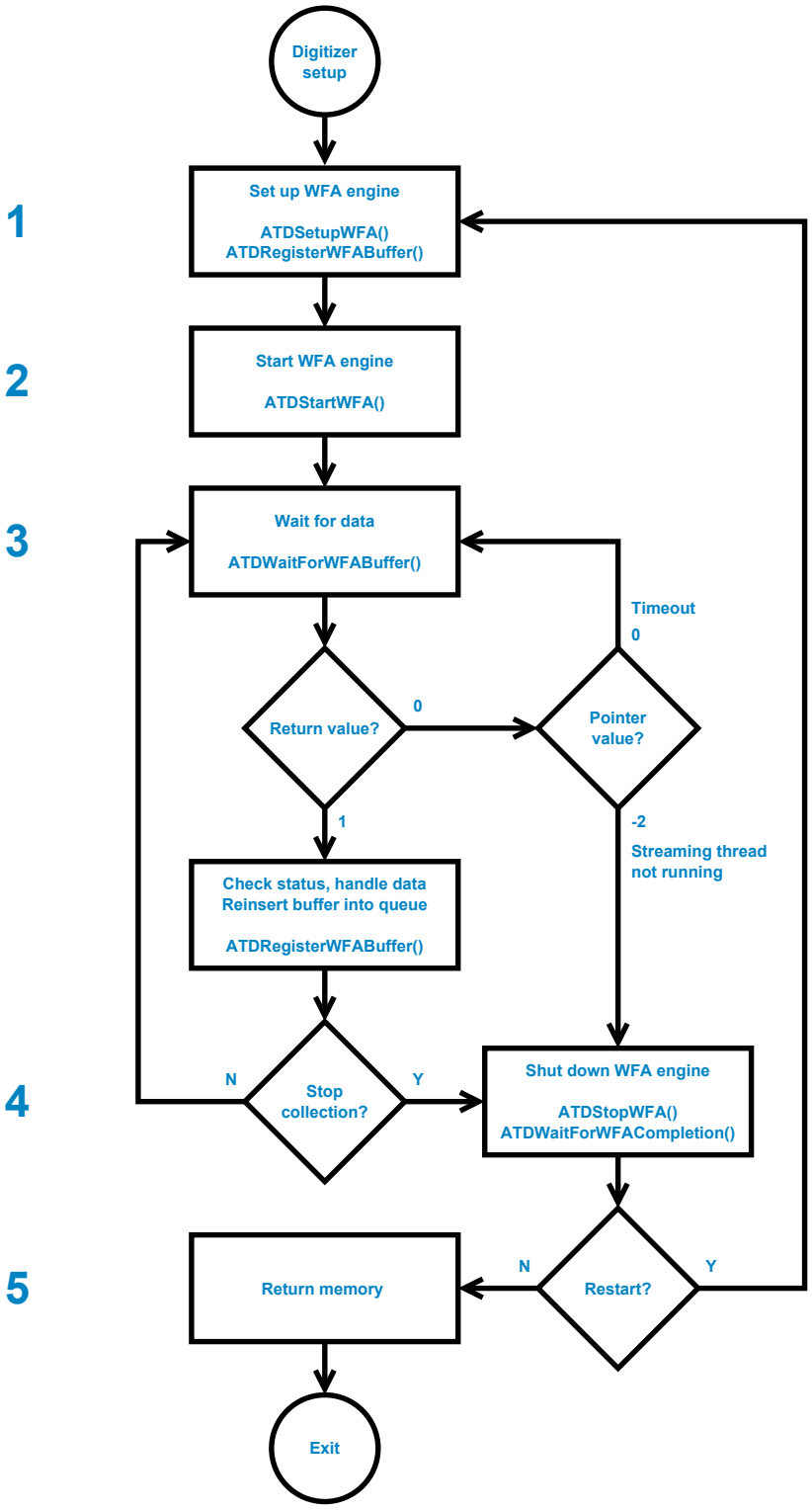

Figure 6: Data collection flowchart for the streaming mode.

3. Once the WFA engine is running, the application enters the inner loop, repeating until the expected amount of data has been received and processed or some unexpected event occurs.

   When data is ready to be transferred to user space, the function `ATDWaitForWFABuffer()` will return a reference to one of the registered buffers, informing the user which buffer has updated contents. At this point, the memory reference is forgotten by the ADQAPI. The user is expected to manipulate the data in whatever way mandated by the use case and then register the memory reference as available memory once again. Thus, the memory is reused and in theory enables observation of an unlimited amount of data using a limited amount of memory.

4. Halting the WFA engine is carried out by calling `ATDStopWFA()` and then waiting for the internal thread to close. The wait state is implemented by the function `ATDWaitForWFACompletion()` which blocks until the thread has closed properly and returns its return value. At this point, the digitizer is once again in the user's control.

5. If the user wishes to restart the acquisition, another iteration of the outer loop is initiated. Otherwise, the dynamically allocated memory is returned to the host and the application exits.

4.9 GPIO

The ATD firmware supports outputting a range of different signals on the *single-ended* GPIO ports. The ports and the corresponding functions are listed in Table 6. The relative timing between the different signals is not guaranteed.

Table 6: Dedicated GPIO pin functions. The GPIO column refers to the ID of the single-ended GPIO pin (see the ADQ7 Manual [5])

GPIO	Direction	Function
0	Out	User range
1	Out	Trigger blocking window
2	Out	Trigger blocking armed
3	Out	Overflow
4	Out	Trigger out
5	Out	Accumulation active
6	Out	Frame sync
7	Out	Clock reference
8	Out	Acquisition armed
9	-	N/A
10	In	Trigger in 0
11	In	Trigger in 1

4.9.1 Enable GPIO Functions

The GPIO functions are enabled per GPIO pin. The following actions must be taken to enable the dedicated function of a pin:

- Set the direction of the GPIO pin with `SetDirectionGPIOPort()`. The direction is controlled in groups of two pins.
- Enable the pin function with `SetFunctionGPIOPort()`.
- Some GPIO pins require additional configuration. For example, `SetupTriggerOutput()` must be called to set up the `GPIO4` (trigger out) pin.

As an example, the following code enables the *accumulation active* signal on `GPIO5`.

```
/* Enable accumulation active signal to be output on GPIO5 */
ADQ_SetFunctionGPIOPort(adq_cu, adq_num, 0, 1, 5);
/* Set GPIO4 and GPIO5 as outputs */
ADQ_SetDirectionGPIOPort(adq_cu, adq_num, 0, (1 << 2), ~(1 << 2));
```

Note

The direction of the GPIO pins is controlled in groups of two, i.e. `GPIO0` and `GPIO1` must have the same direction, `GPIO2` and `GPIO3` must have the same direction etc.

4.9.2 GPIO Pin Functions

Each single-ended GPIO pin may be configured for a specific function. The functions are described below.

User range `GPIO0`

The user range pin is asserted when the analog input is within a specified range. The range is configured with the API call `SetupUserRangeGPIO()`. The pin is asserted when the data is below the high threshold and above the low threshold. The thresholds are inclusive. The user range is only available for one channel at a time.

Trigger blocking window `GPIO1`

The trigger blocking window pin is asserted when trigger blocking is enabled and triggers are accepted.

Trigger blocking armed `GPIO2`

The trigger blocking armed pin is asserted when the trigger blocking engine is armed.

Overflow `GPIO3`

The overflow pin is asserted when the WFA overflows and discards incoming data (see Section 4.7). This pin requires no additional configuration.

Trigger out `GPIO4`

The GPIO trigger out pin is configured with `SetupTriggerOutput()`. The GPIO trigger out and the TRIG connector output cannot be configured independently. That is, if a certain trigger is output on GPIO, another trigger may not be output on the TRIG connector. However, the TRIG connector may still be used as an input independently of the GPIO trigger out pin.

Accumulation active `GPIO5`

The accumulation active pin is asserted on the first raw record of the user space record and is deasserted after the last raw record of the user space record has been acquired. Fig. 7 presents a timing digram. This pin requires no additional configuration.

Frame synchronization `GPIO6`

The frame synchronization feature is configured with `SetupFrameSync()`, and must be enabled with `EnableFrameSync()`.

Clock reference `GPIO7`

The clock reference pin outputs the 10 MHz clock reference.

Acquisition armed `GPIO8`

The acquisition armed pin is asserted when the data acquisition is armed.

Trigger in 0 `GPIO10`

External trigger input. This input can be used as a trigger source, a source for the trigger blocking engine or to synchronize the timestamp. `GPIO10` can be configured independently of `GPIO11`.

Trigger in 1 `GPIO11`

External trigger input. This input can be used as a trigger source, a source for the trigger blocking engine or to synchronize the timestamp. `GPIO11` can be configured independently of `GPIO10`.

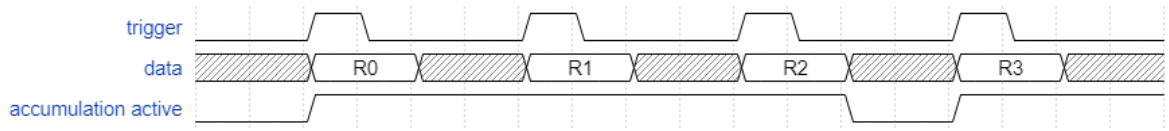

Figure 7: Timing diagram for `GPIO5` (accumulation active). In this case, the number of accumulations is 3. Note that the width of the trigger signal is not to scale.

4.10 Converting from ADC Codes to Volts

The unit of the received data is ADC codes. The full scale input range is 1 V peak-to-peak. This is mapped to the range $[-2^{14-1}, 2^{14-1}-1]$, meaning one LSB is

$$\mathrm{LSB}_{\mathrm{Volt}} = \frac{1}{2^{14}}\ \mathrm{V}. \tag{3}$$

The ADC codes may then be converted to Volt by using (4).

$$X_{\text{Volt}} = \frac{1}{2^{14}} \cdot X_{\text{ADC}} \tag{4}$$

4.11 ADC Code Ranges

Vertical settings in the ADQAPI are defined using 16-bit ADC codes, i.e. in the range

$$[-32768,\ 32767]$$

regardless of the digitizer resolution. In the case of ADQ7-FWATD, which delivers 14-bit data aligned with the LSB at bit index zero. For example, this means that there will be difference of a factor of four between the configured level trigger threshold value and the corresponding sample value in the user space record.

Note

Vertical settings in the ADQAPI are defined using 16-bit ADC codes regardless of the digitizer resolution while ADQ7-FWATD returns data which should be interpreted in the 14-bit range. For example, setting the level trigger to ADC code 1024 corresponds to a sample value of 256 ADC codes in the user space record.

4.12 Threading

The ADQAPI is not thread-safe. This means that if the user wishes to communicate with the digitizer from multiple threads in the application, a locking mechanism (implemented by the user) is required in order to avoid race conditions and undefined behavior. Section 4.5 explained that the WFA is partitioned and that the final accumulation step is performed in software by the ADQAPI. This accumulator is running in its own thread and as such, while the data is being collected (i.e. after calling `ATDStartWFA()`), there are only a few functions that are safe to call, namely

- `ATDGetWFAStatus()`
- `ATDRegisterWFABuffer()`
- `ATDStopWFA()`
- `ATDUpdateNofAccumulations()`
- `ATDWaitForWFABuffer()`
- `ATDWaitForWFACompletion()`
- `ArmTriggerBlocking()`
- `WriteGPIOPort()`

Warning

The ADQAPI is not thread-safe. Calling API functions from multiple threads may yield undefined behavior.

5 User Application Example

This section aims to explain how to write the core code in the user application interfacing to ADQ7-FWATD by using the provided C code as an example.

The ADQAPI reference guide [3] lists all the functions available in the ADQAPI and is a good supplementary source of information. Please note that not all functions are valid for ADQ7.

5.1 Overview

The code implementing the digitizer interface in the user application centers around memory management, configuration, running the measurement and reacting to any unexpected events in a way that makes sense in the context of the user application.

Since ADQ7-FWATD extends the functionality of the base product, ADQ7-FWDAQ, almost all of the functions listed as valid for ADQ7 in the ADQAPI reference guide [3] are valid for ADQ7-FWATD as well. This includes functions handling trigger configuration, DC offset in the analog front-end, digital baseline stabilization, clocking etc. The exceptions are the data management functions where the FWATD-specific functions should be used instead of the general streaming functions.

The example is divided into five parts and the principal actions should be imitated in any code interfacing to ADQ7-FWATD.

Parameter definition

The digitizer settings are defined in this step together with the allocation of static memory.

Memory allocation

The application dynamically allocates system memory to be used as target buffers, used to shuttle data between the API and user space.

Digitizer setup

Based on the defined parameters, relevant API functions are called to carry out the digitizer configuration. The order of operations does not matter in this step as long as no data collection functions are called and the digitizer is disarmed (default state).

Data collection

The heart of the application is the data collection loop. There are two versions, controlled by the define `SINGLE_SHOT_COLLECTION` (disabled by default). The define will cause either the streaming- or the single-shot version of the data collection loop to execute. The implementation of the single-shot mode is straight-forward and while its simplicity may seem attractive, it is not without drawbacks (see Section 4.8.1). The more powerful streaming mode is described further in Section 4.8.2.

Returning memory

Finally, any dynamically allocated memory is returned to the host computer.

5.2 Modifying Settings

The digitizer configuration parameters reside in the file `example_adq7.c` immediately following the declaration of the function `adq7()`. The example code is filled with comments intended to guide the user in choosing the parameters.

5.3 Compiling

Compiling the example application is accomplished by generating object files from the three source files and linking to the ADQAPI. A simple make file for Linux systems is provided as well as a Microsoft Visual Studio solution.

5.4 Default Behavior

By default, the test pattern generator and level trigger are activated. This means that the example can be run out of the box and data will be received even without connecting any input signals or triggers. The test pattern can be deactivated by changing the test pattern mode from '4' to '0' on the following line:

```
const int test_pattern_mode = 4;
```

The test pattern is 16 bits whereas the accumulator is 14 bits. Therefore, the test pattern will update every fourth samples. For example, the raw data stream will follow the pattern

```
N, N, N, N, N+1, N+1, N+1, N+1, N+2, ...
```

for the positive slope, and equivalently

```
N, N, N, N, N-1, N-1, N-1, N-1, N-2, ...
```

for the negative slope.

Note

The samples will update in groups of four if the test pattern is enabled.

The example in C has no built-in plotting. Instead, the data is written to file. Channel A is written to `dataA.out` and channel B is written to `dataB.out` for the two-channel firmware. For the one-channel (10 GSPS) firmware, the data is written to `dataA.out`.

Note

If file writing is activated, the data is written to `dataA.out` and `dataB.out`.

The data can either be stored in ASCII or binary format. The format is specified with

```
const unsigned int write_mode = 2;
```

and the valid writing modes are listed in Table 7.

5.5 Interpreting the Data

The data stored as the raw accumulated waveforms, one after another with one file per channel. A sample is represented as a 64-bit signed integer and the file format can be binary or ASCII. This makes it

Table 7: The different file writing modes in the FWATD example in C.

Mode	Description
0	File writing disabled
1	ASCII
2	Binary

impractical to have the data storage active for measurements which are expected to yield large amounts of DC offset data, resulting in a large file (several gigabytes). Such a file is cumbersome to handle for any operating system. If the example application has to be used for such a use case, an alternative approach is to split the data file once a certain maximum file size has been reached.

Interpreting the data involves dividing the raw accumulated data with the number of accumulations to convert the values back into ADC codes and subsequently Volts (see Section 4.10).

5.6 MATLAB

On Windows systems, the ADQAPI has support for MATLAB through the use of a MEX file. This intermediate layer handles, among other things, the memory allocation and data transfers between the user's MATLAB code and the ADQAPI. The example scripts are located in

```
<Path to installation directory>/Matlab_examples/MATLAB_ExampleScripts/
```

Currently, only the single-shot mode (Section 4.8.1) is supported and is demonstrated in the script

```
ADQ7_FWATD_example_script.m
```

This script should run right away provided a correctly configured ADQ7 digitizer is connected to the host computer. The default behavior is to use the internal trigger to collect a single record (no accumulations) on each channel.

The data grabbing function `ATDStartWFA()` returns a struct containing data and headers for all records separated into channels. The struct fields are named after the channel label, i.e. 'ChannelX' in one-channel mode and 'ChannelA' and 'ChannelB' in two-channel mode.

In order for MATLAB to correctly locate the MEX file, it is important to either extract the helper script

```
get_adq_installer_path.m
```

and place it in the working directory next to the example, or to manually add the path

```
<Path to installation directory>/Matlab
```

to MATLAB's internal path variable.

Important

MATLAB must be given knowledge of where the MEX file is located. This can be accomplished by manually adding the path to the Teledyne SP Devices MATLAB directory or by using the helper script `get_adq_installer_path.m`

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.

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 WFA settings as well as the specification for both the trigger and data signals. Make sure to include a trace log file from a run where the error appears.

6.1 Managing License Files

ADQ7-FWATD requires a valid license to be able to operate properly. An unsuccessful license validation will result a trace log error message and failure of the ADQAPI functions `ATDStartWFA()` and `ATDSetupWFA()`. The digitizer licenses are managed with the ADQLicenseUtil tool, included in the SDK. Please direct all license related questions at spd_support@teledyne.com.

6.1.1 Reading the DNA

Licenses are locked to each individual digitizer by means of the *DNA*, a unique 128-bit number. Performing field updates to the digitizer licenses may require the DNA and digitizer serial number to be read out and forwarded to Teledyne SP Devices. This task can be accomplished by opening a command prompt or terminal window with access to the ADQLicenseUtil application and running

```
$ adqlicenseutil d
```

Make a note of the digitizer serial number and DNA in the resulting output text.

6.1.2 Updating the Digitizer License

Provided a valid license file, `<license>.lic`, the command

```
$ adqlicenseutil w <license>.lic
```

transfers the license to the digitizer.

References

[1] Teledyne Signal Processing Devices Sweden AB, *18-2059 ADQUpdater User Guide*. Technical Manual.

[2] Teledyne Signal Processing Devices Sweden AB, *16-1692 ADQ7 datasheet*. Technical Specification.

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

[4] Teledyne Signal Processing Devices Sweden AB, *16-1794 ADQ7-FWATD Datasheet*. Technical Specification.

[5] Teledyne Signal Processing Devices Sweden AB, *16-1796 ADQ7 manual*. Technical Manual.

A Linux SDK README

```
=== INSTALLATION ===
To install the SDK packages, enter the "packages" directory and find the linux
distribution and processor architecture you are using.  Instructions for each
distribution follows below.

 - Ubuntu and Debian
      Install delivered versions of our packages using 'dpkg -i packagename'.
      Use the following order:
         spd-adq-pci-dkms
         libadq0
         adqtools

 - OpenSUSE and SUSE Linux Enterprise
      Install these packages using 'zypper install packagename':
         make
         kernel-devel
         gcc
      The version of kernel-devel must match your current kernel.
      Install delivered versions of our packages using 'rpm -U packagename'.
      Use the following order:
         dkms
         spd-adq-pci-dkms
         libadq0
         adqtools

 - Fedora 19, 20 and 21
      Install these packages using 'yum install packagename':
         dkms
      Install delivered versions of our packages using 'rpm -U packagename'.
      Use the following order:
         spd-adq-pci-dkms
         libadq0
         adqtools

 - Fedora 22 and higher
      Install these packages using 'dnf install packagename':
         dkms
      Install delivered versions of our packages using 'dnf install packagename'.
      Use the following order:
         spd-adq-pci-dkms
         libadq0
         adqtools
```

```
- CentOS / Red Hat Enterprise Linux / Scientific Linux
      Note: For RHEL6, use CentOS6 packages.
      Install these packages using 'yum install packagename':
         make
         kernel-devel
         gcc
      The version of kernel-devel must match your current kernel.
      Install delivered versions of our packages using 'rpm -U packagename'.
      Use the following order:
         dkms
         spd-adq-pci-dkms
         libadq0
         adqtools

After installing all necessary packages, reboot the system so that udev reads
the updated configuration and the driver loads.

In earlier versions there was a package called 'adqupdater' which is now
replaced with the 'adqtools' package.

=== DEVICE ACCESS RIGHTS ===
ADQ devices show up as /dev/adq_pcie_ and /dev/adq_usb_, the default udev
setting is to add read/write access to the user group "adq".  The libadq0
package will create a user group called "adq" if that group doesn't already
exist in the system.  To be able to access the devices, in order to add your
user to the "adq" group use the command:

    usermod -a -G adq username

The user will have to logout and login again for the changes to take effect.

=== COMPATIBILITY ===
The PCIe kernel module supports kernel versions from 2.6.32 and forward,
however kernel version 3.8.0 or newer is recommended.

The PCIe kernel module is not signed and thus will not load if your kernel uses
"secure boot".  If "secure boot" is enabled, it will need to be disabled before
the kernel module can be loaded.  Refer to your distribution documentation on
how to do this.

=== API USER GUIDE ===
The API user guide is included as a .pdf document in the "doc" directory. All
API functions are described there.
```

```
=== EXAMPLE CODE ===
Some example code can be found in the "examples" directory.
```

TELEDYNE SP DEVICES
Everywhereyoulook™

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

Teledyne SP Devices Corporate Headquarters
Teknikringen 6
SE-583 30 Linköping
Sweden
Phone: +46 (0)13 645 0600
Fax: +46 (0)13 991 3044
Email: spd_info@teledyne.com