TELEDYNE SP DEVICES
Everywhereyoulook™

ADQ7-FWPD

User Guide

Author(s): Teledyne SP Devices
Document ID: 18-2132
Classification: Public
Revision: C
Print date: 2023-11-08

Contents

1 Introduction **2**
1.1 Definitions and Abbreviations . . . 2

2 Features **3**
2.1 Specification . . . 3
2.2 Overview . . . 3

3 Getting Started **6**
3.1 SDK Installation . . . 6
3.1.1 Installing the SDK (Windows) . . . 6
3.1.2 Installing the SDK (Linux) . . . 6
3.1.3 Unsupported Software Features . . . 7
3.2 Collecting Data . . . 7

4 Detailed System Description **8**
4.1 Digital Baseline Stabilization . . . 8
4.2 Pulse Specification . . . 8
4.2.1 Edge Windows . . . 11
4.3 Trigger Features . . . 11
4.3.1 Timestamp Synchronization . . . 11
4.3.2 Trigger Blocking . . . 12
4.3.3 Synchronization Signals . . . 13
4.4 Detection Window . . . 14
4.5 Metadata channels . . . 14
4.6 Pulse Characterization . . . 14
4.6.1 Pulse Width . . . 15
4.6.2 Pulse Peak Value . . . 15
4.6.3 Pulse Peak Value Timestamp . . . 15
4.7 Padding . . . 16
4.8 User Space Record . . . 16
4.8.1 Fixed Length Record . . . 16
4.8.2 Variable Length Record . . . 16
4.9 Data Collection Modes . . . 17
4.9.1 Pulse Data . . . 17
4.9.2 Every *N*th Record . . . 17
4.9.3 Pulse Data with Padding . . . 18
4.10 Pulse Metadata . . . 18

5 User Application Example **20**
5.1 Example in C . . . 20
5.1.1 Python Scripts . . . 20

6 Troubleshooting **21**

1 Introduction

This document is the user guide for the ADQ7 digitizer running the pulse detection firmware (FWPD) option. Equipped with the pulse detection firmware, the ADQ7 digitizer provides the user with sophisticated tools to identify, collect and analyze *regions of interest* in the data stream in real-time. This relaxes the requirements on the host computer in terms of sustained bandwidth of the device-to-host interface, storage space and offline data parsing.

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
ADC	Analog-to-digital converter
API	Application programming interface
DBS	Digital baseline stabilization
FWPD	Pulse detection firmware for ADQ14 and ADQ7
GSPS	10^9 samples per second
GUI	Graphical user interface
Horizontal parameters	Used to describe parameters and settings which are specified in samples (SI unit *seconds*). For example, the length of a record of data.
LEW	Leading edge window
MSPS	10^6 samples per second
TEW	Trailing edge window
Vertical parameters	Used to describe parameters and settings which are specified in ADC codes (SI unit *Volt*). For example, the level trigger threshold.

2 Features

This section presents the specification of ADQ7-FWPD along with brief descriptions of its core features. Detailed information is found in Section 4. Additionally, information pertaining to the ADQ7 platform in general and not this firmware specifically may be found in the ADQ7 manual [1]. However, if the documents give conflicting information, this document takes precedence for devices running this particular firmware option.

2.1 Specification

The specification for the pulse detection firmware is presented in Table 2. For the general specification of the ADQ7 digitizer, please refer to the ADQ7 data sheet [2] [3].

Table 2: Specification for ADQ7-FWPD.

Item	Min	Max	Unit
General			
Pulse width	1	-	Sample
Pulse separation	1	-	Sample
Record length (one-channel mode)	32	$2^{32} - 1$	Sample
Record length (two-channel mode)	16	$2^{32} - 1$	Sample
Pulse characterization			
Pulse width [1]	1	$2^{16} - 1$	Sample
Pulse peak value [2]	-2^{15}	$2^{15} - 1$	ADC codes
Pulse peak timestamp [3]	-	$2^{32} - 1$	Sample
Average pulse rate (one-channel mode)	-	2.5	10^9 Pulses / s
Average pulse rate (two-channel mode)	-	1.25	10^9 Pulses / s

[1] See Section 4.6.1
[2] See Section 4.6.2
[3] See Section 4.6.3

2.2 Overview

Fig. 1 presents a block diagram outlining the main features of the data path. The diagram is drawn for a four-channel device but applies to one- and two-channel devices as well.

- **DC offset**

  The analog signal may be subjected to a DC offset. If only pulses with a certain polarity is expected, the user may wish to offset the analog signal to place the baseline close to the maximum or minimum ADC code. This allows pulses with amplitudes that would normally saturate the ADC to be safely collected, thus increasing the dynamic range of the digitizer.

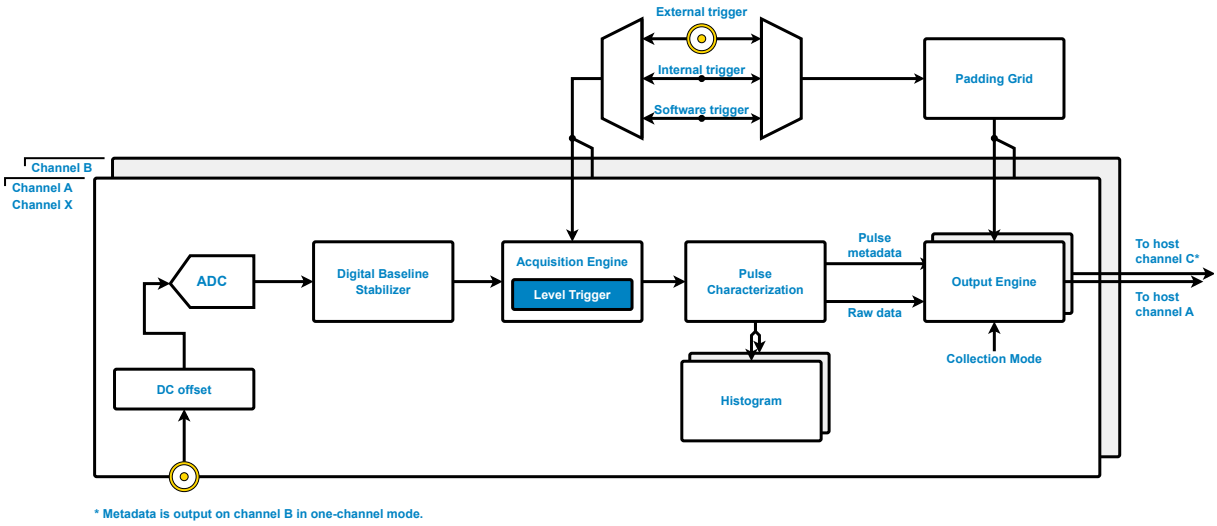


Figure 1: A block diagram presenting the main features of the ADQ7-FWPD data path.

- **ADC**

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

- **Digital baseline stabilizer**

  The digital baseline stabilizer (DBS) provides a highly accurate and stable baseline, specifically intended for pulse applications. Additional information on the DBS is provided in Section 4.1.

- **Acquisition engine**

  The acquisition engine is tasked with creating records of data. Each channel is configured independently and requires a trigger condition and a specification of the horizontal parameters.

  - **Trigger**

    The trigger condition may be selected from one of the three global trigger sources: external trigger, internal trigger or software trigger. The latter is mainly used for debugging purposes and has limited practical application. The selected source is shared among all channels using the global trigger source. Another option is to use the level trigger, in which case each channel operates independently since the trigger condition is detected by analyzing the local data stream. The different ways of triggering a record are discussed throughout Section 4, particularly in Sections 4.3, 4.4 and 4.8.

  In addition to the trigger condition, the definition of a region of interest may be refined further by using a *detection window* (Section 4.4). This window defines a period in time during which pulses are accepted for analysis. Any pulses outside of the window are rejected and are prevented from propagating to the user.

- **Pulse characterization**

  The pulse characterization function offers real-time analysis of the pulse data where three key metrics are extracted: the *peak value*, the *width* (time-over-threshold) and the *time of the peak value* relative to the trigger timestamp. The pulse metadata format is described in detail in Section 4.10.

Additionally, the metadata collection mode (discussed in Section 4.9) allows the user to reduce the output data rate further by choosing to only output the pulse metadata.

- **Padding grid**

  The padding grid defines a window in time, which acts as a guide when the digitizer inserts padding data. This padding data is required in applications which require a certain minimum activity regardless of the event frequency, while still maintaining a high throughput capacity of the device-to-host interface.

  Inserting padding data into the data stream introduces overhead and thus lowers the effective bandwidth of the device-to-host interface (the bandwidth used to transfer useful data). This is an inevitable consequence since the concept of padding targets the trade-off between throughput and latency. The goal is to provide the user with the tools needed to find a balanced compromise between the requirements of the application. The padding mechanism is described in detail in Section 4.7.

- **Output engine**

  The output engine offers flexible control over the flow of data. There are four different data collection modes described in detail in Section 4.9.

3 Getting Started

This section provides information on how to interface with ADQ7 and the pulse detection firmware. Table 3 describes the peripherals and their usage.

Table 3: Peripheral connections on ADQ7.

Connector	Description
TRIG	External trigger input/output
SYNC	Synchronization input/output
A	Channel A input (two-channel devices)
B	Channel B input (two-channel devices)
X	Channel X input (one-channel devices)
CLK	External reference clock input

3.1 SDK Installation

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

3.1.3 Unsupported Software Features

Listed below are the features supported by ADQ7 which are **not** supported by ADQ7-FWPD

- ADCaptureLab
- LabView API
- MATLAB API

3.2 Collecting Data

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

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

after installing the SDK, and for Linux in the

```
examples/fwpd_example
```

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

4 Detailed System Description

This section provides additional details on important system features.

4.1 Digital Baseline Stabilization

For optimal performance in pulse applications, it is advantageous if the baseline of the signal is locked to a specific value. This is the purpose of the Digital Baseline Stabilizer (DBS). The signal processing block is placed directly after the ADC in the data path.

The DBS uses a blind identification process operating in the background. When activated by the user, DBS will constantly monitor and follow variations in the baseline and correct for time-varying behavior such as temperature variations. The baseline is adjusted to the target value with 22-bit precision.

4.2 Pulse Specification

In order for the system to sift through the constant stream of data, separating regions of interest from uninteresting regions, the user has to tune the system to look for pulses fulfilling a certain criteria—the *pulse specification*.

A pulse is defined by a *trigger level* and three *hysteresis* (offset) values: the *reset hysteresis*, the *trigger arm hysteresis* and the *reset arm hysteresis*. These four values are specified by the user together with a polarity setting, indicating if the target pulse is

- *positive*, i.e. the pulse grows toward *higher* ADC codes from the baseline or
- *negative*, i.e. the pulse grows toward *lower* ADC codes from the baseline.

The arithmetic involved in computing the absolute detection levels of the pulse differs depending on the polarity. Equations (1) and (2) present the relations for positive and negative pulses, respectively.

Positive pulses

$$\begin{aligned} \text{Reset level} &= \text{Trigger level} - \text{Reset hysteresis} \\ \text{Trigger arm level} &= \text{Trigger level} - \text{Trigger arm hysteresis} \\ \text{Reset arm level} &= \text{Reset level} + \text{Reset arm hysteresis} \end{aligned} \tag{1}$$

Negative pulses

$$\begin{aligned} \text{Reset level} &= \text{Trigger level} + \text{Reset hysteresis} \\ \text{Trigger arm level} &= \text{Trigger level} + \text{Trigger arm hysteresis} \\ \text{Reset arm level} &= \text{Reset level} - \text{Reset arm hysteresis} \end{aligned} \tag{2}$$

Moreover, the conditions to generate a trigger or reset event are stated in (3) and (4) for positive and negative pulses, respectively. The data at time instance n is denoted $x[n]$.

Positive pulses

$$\begin{aligned} &\text{Trigger event:} && x[n] \geq \text{Trigger level} \\ &\text{Reset event:} && x[n] \leq \text{Reset level} \end{aligned} \tag{3}$$

Negative pulses

$$\begin{aligned} &\text{Trigger event:} && x[n] \leq \text{Trigger level} \\ &\text{Reset event:} && x[n] \geq \text{Reset level} \end{aligned} \tag{4}$$

Fig. 2 places the levels discussed above into the context of a positive pulse. The figure consists of five subfigures:

- **Fig. 2a—the reset level and events**

  The reset level is computed by subtracting the reset hysteresis from the trigger level. When the *rising edge* of the pulse crosses the trigger level a *trigger event* is generated. Conversely, when the *falling edge* of the pulse crosses the reset level a *reset event* is generated. An event always coincides with a sample point and is chosen as the first sample on or beyond the threshold. These two events are the heart of the pulse detection firmware as they frame the region of interest, aiding subsequent functions such as pulse characterization.

- **Fig. 2b—the trigger arm level**

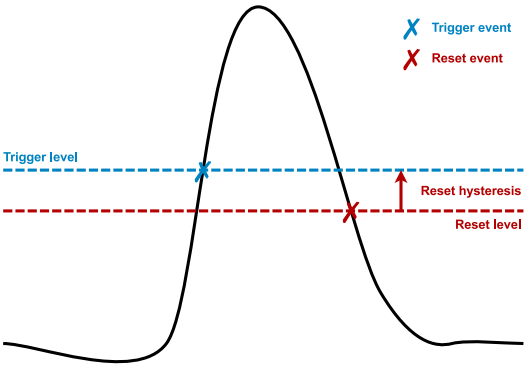
  The trigger arm level is computed by subtracting the trigger arm hysteresis from the trigger level. In order for the rising edge to generate a trigger event, the system has to be *armed*. For a positive pulse, this means that the signal has to visit ADC codes equal to or less than the trigger arm level. Following a trigger event, the system is automatically disarmed until the arming criteria is met once again.

- **Fig. 2c—the reset arm level**

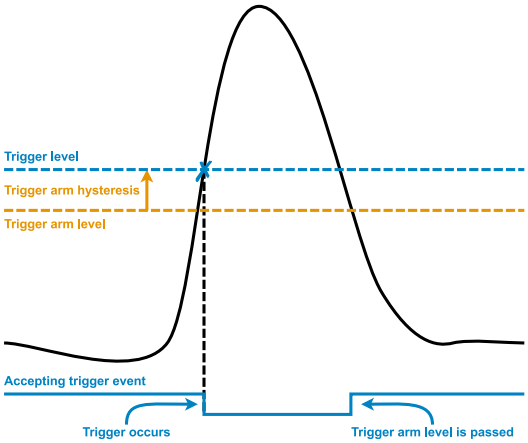
  The arming mechanism for the reset event is identical to that of the trigger event apart from one detail: they are mirror versions of each other. In order for the falling edge of the pulse to generate a reset event, the signal has to visit ADC codes equal to or greater than the reset arm level. Following a reset event, the system is automatically disarmed until the arming criteria is met once again.

- **Figs. 2d and 2e—false events**

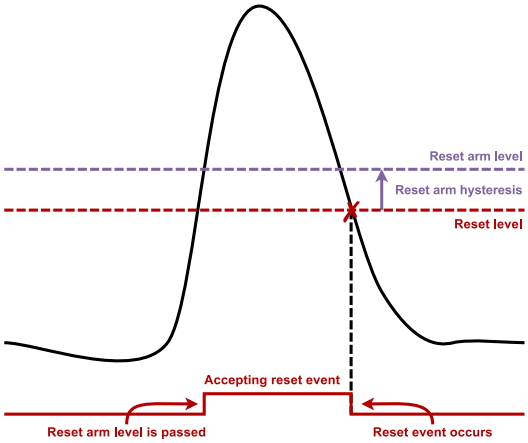
  The arming mechanisms are what enables the protection of the data stream against false events. However, this protection hinges on the hysteresis values input by the user to work correctly. As shown in Fig. 2d, badly configured arm levels may have the same effect as not having them at all. The arming levels should be set *at least* in the same range as the noise to be effective, as shown in Fig. 2e.



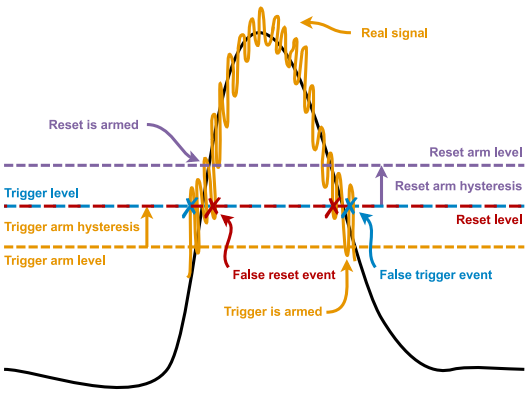
(a) The reset level is determined from the *trigger level* and the *reset hysteresis*.



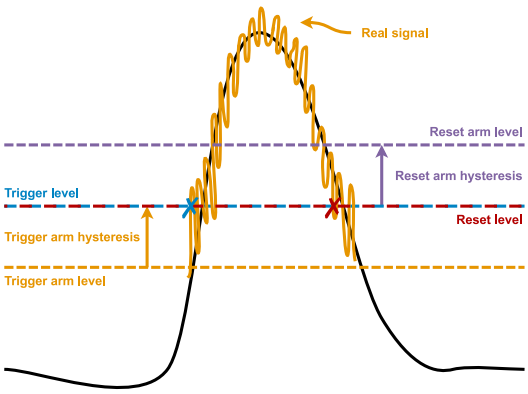
(b) The trigger arm level is determined from the *trigger level* and the *trigger arm hysteresis*. The level is used to arm the mechanism generating trigger events.



(c) The reset arm level is determined from the *reset level* and the *reset arm hysteresis*. The level is used to arm the mechanism generating reset events.



(d) Badly configured arm levels may yield false events from the incorrectly detecting pulses within the signal noise.



(e) Setting the arm levels greater than the noise variation protects against false events.

Figure 2: Figs. 2a–2c demonstrate the pulse definition in the context of a positive pulse. Figs. 2d and 2e highlight the purpose of the arm levels.

4.2.1 Edge Windows

The values discussed in the previous section target the vertical characteristics of a pulse. How the horizontal characteristics of a pulse are treated by the pulse detection firmware are discussed in more detail in Sections 4.8 and 4.9. However, this section will introduce the *leading edge window* (LEW) and the *trailing edge window* (TEW). The names hint at their purpose; the former defines a window in time *before* the trigger event while the latter defines a window *after* the reset event, as shown in Fig. 3. These windows extend the region of interest outward from the pulse.

Note

The leading edge window and trailing edge window are two fixed-length regions that surround the pulse and allow the user to extend the region of interest.

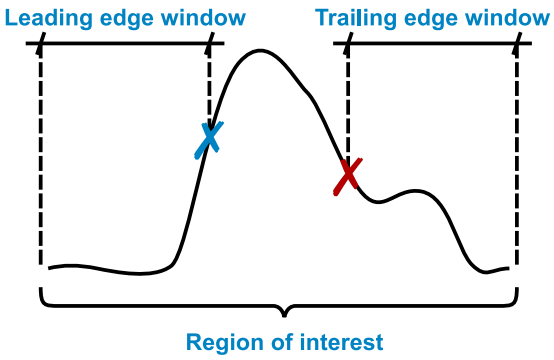


Figure 3: The leading edge window and the trailing edge windows are used to extend the region of interest relative to the trigger event and reset event, respectively.

4.3 Trigger Features

This section describes features relating to the external trigger interface: blocking triggers, generating synchronization signals to external equipment and resetting the internal timestamp of the digitizer. Information regarding how a trigger event is used to acquire data is discussed in other sections, namely Sections 4.4, 4.8 and 4.9.

4.3.1 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.

4.3.2 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.

4.3.3 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. 4 presents an example scenario.

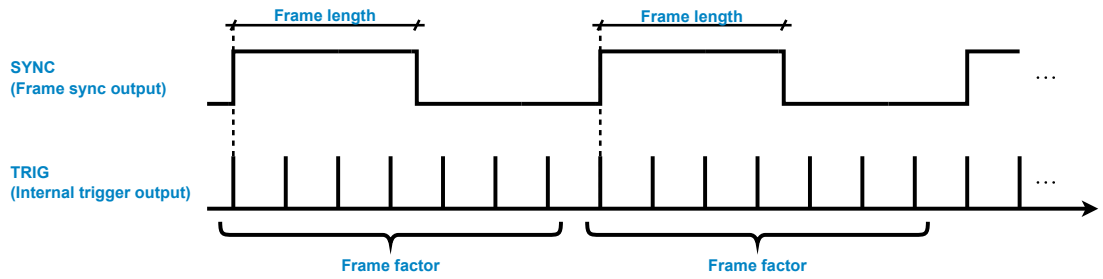


Figure 4: An illustration of the frame synchronization feature. In this scenario, the frame factor is set to 7.

4.4 Detection Window

A detection window defines a period in time in which pulses are accepted and forwarded to the pulse characterization engine. The detection window has a fixed length and an associated trigger event which may be the output of the global trigger multiplexer or the per-channel level trigger. Fig. 5 demonstrates detection window where two corner cases are highlighted: any pulse not *completely* inside the detection window is rejected. From a technical point of view, *completely* is defined as both the trigger event and the reset event (Section 4.2) occurring inside the window. How the detection window is set up depends on the data collection mode. Refer to Section 4.9 for details.

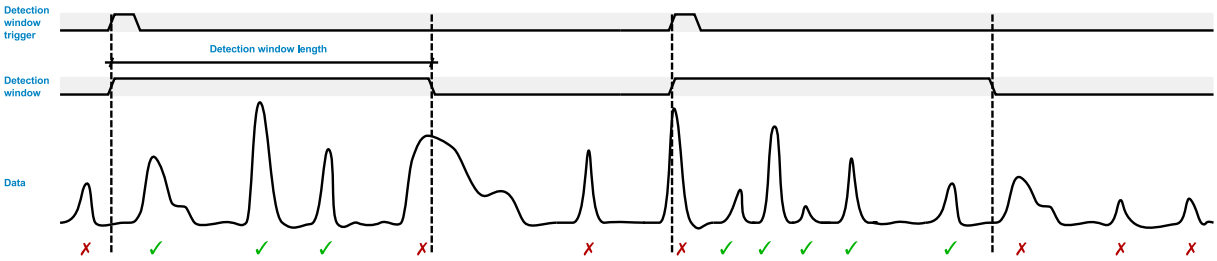


Figure 5: An example of how a detection window is used to mask undesired pulses. Pulses marked with a green check mark are accepted. Pulses marked with a red *x*-mark are rejected.

4.5 Metadata channels

The pulse characterization data (metadata) is received on separate channels from the ADC data. The number of metadata channels is equal to the number of data channels. The metadata channels use the same settings as the corresponding data channel, with the exception of the minimum record length. The metadata and data channels are activated by setting the corresponding bit in the channel mask equal to one. The metadata channels are indexed after the data channels, see Table 4.

Example

- To collect metadata on channel B (two-channel firmware), the channel mask is set to `0x8`.
- To collect metadata on channel B and raw data on channel A (two-channel firmware), the channel mask is set to `0x8` + `0x1`.
- To collect metadata on channel X (one-channel firmware), the channel mask is set to `0x2`.

4.6 Pulse Characterization

The pulse characterization feature is used to compute key pulse metrics in hardware. The metrics are:

- the pulse width (time-over-threshold),
- the pulse peak timestamp and

Table 4: Channel numbers for metadata and data channels.

Firmware	Channel no.	Description
One-channel	1	Pulse data for input X
	2	Pulse metadata for input X
	3	Not used
	4	Not used
Two-channel	1	Pulse data for input A
	2	Pulse data for input B
	3	Pulse metadata for input A
	4	Pulse metadata for input B

- the pulse peak value (extreme value).

These metrics form a pulse metadata packet described in Section 4.10. Pulses are characterized during the detection window, as illustrated in Fig. 5 where pulses starting before the window trigger or ending after the window ends are ignored.

4.6.1 Pulse Width

The pulse width is computed as the number of samples between the pulse trigger and reset level. This value will wrap if the pulse width is larger than $2^{16} - 1$ samples.

Note

The pulse width metadata value will wrap if the pulse length is larger than $2^{16} - 1$ samples.

4.6.2 Pulse Peak Value

The maximum value or the minimum value of the pulse. For positive pulses the peak is defined as the maximum value, and for negative pulses the peak is defined as the minimum value. The peak is a 16-bit signed value.

Note

Since the peak value is a 16-bit signed value, the absolute difference between the baseline and the peak cannot exceed $2^{15} - 1$.

4.6.3 Pulse Peak Value Timestamp

The peak value timestamp is the sample index at which the extreme value occurred, relative to the record header timestamp. That is, the time in samples between the detection window trigger and the extreme value.

The timestamp is a 32-bit unsigned value and will wrap if the distance between the detection window trigger and the pulse is larger than $2^{32} - 1$ samples. If the peak value is reached at multiple samples, the timestamp value will report the last one.

Note

The timestamp value will wrap after $2^{32} - 1$ samples.

4.7 Padding

The data padding functionality is used to keep a constant data rate from the digitizer. Padding is always active for the metadata channels but only active for the data channels when the data collection mode is set to *pulse data with padding* (Section 4.9.3). The padding engine will append zeros to the data to ensure that the total amount of data measured over one padding frame is at least equal to the *minimum frame length*. The padding data is always inserted as a new record filled with zeros with the header field *User ID* set to 1. Setting the minimum frame length to zero disables the padding.

Note

The padding functionality is disabled by setting the minimum record length to zero.

The padding is applied to each *padding frame* after a certain offset. The padding offset is equal to the detection window length in pulse metadata mode. The padding is configured using the API call `PDSetupCharacterization()`. How the padding frame is set up depends on the data collection mode, see Table 5.

4.8 User Space Record

User space records are the records of data received by the user. A record consists of a data part and a header part. The header includes information about e.g. the length and the trigger timestamp. The contents of the user space records will depend on the data collection mode. In all modes but the metadata mode, the record will consist of the data sampled by the ADC. In the metadata mode, the record will consist of the pulse metadata packets. The raw data records can either have a *fixed length* or a *variable length*. Fig. 6 illustrates the differences between the two record types. The length of a metadata record will depend of the number of pulses detected.

4.8.1 Fixed Length Record

A fixed length record is defined by the trigger event and the record length. The length of a fixed length record is specified by the user, and is independent of the data collected. A fixed length record can be triggered by any of the trigger sources.

4.8.2 Variable Length Record

The length of a variable length record will depend on the data. A variable length record is defined as the data between the trigger event and reset event. If trailing or leading edge windows are used, these will also be included in the record.

If two pulses have overlapping trailing and leading edge windows, they will be concatenated into one single record. This is illustrated in Fig. 6. The variable length mode can only be used with the level trigger.

4.9 Data Collection Modes

This section describes the different data collection modes. The collection mode only applies to the data channels, i.e. channels A, B and X, and is configured using the API function `PDSetupCharacterization()`. The available data collection modes are:

Mode	Description
0	Pulse data
1	*Reserved*
2	Every *N*th record
3	Pulse data with padding (with detection window)
4	Pulse data with padding (without detection window) Mode 4 is **not** supported for the metadata channels.

Note

The metadata channels are also configured by calling `PDSetupCharacterization()`. However, not all arguments are used. Refer to the ADQAPI reference guide [4] for additional details.

The collection mode determines how the trigger arguments of `PDSetupCharacterization()` are used, see Table 5.

Table 5: This table lists how the arguments of `PDSetupCharacterization()` are used to control the pulse detection features.

Mode	Padding frame trigger	Record trigger	Padding offset
0	`trigger_mode`	`trigger_mode`	`record_length`
1	N/A	N/A	N/A
2	`trigger_mode`	`trigger_mode`	`record_length`
3	`padding_trigger_mode`[1]	`trigger_mode`	`detection_window_length`
4	`padding_trigger_mode`	`trigger_mode`	`padding_offset`

[1] The padding trigger is also used as the detection window trigger in these modes

4.9.1 Pulse Data

This is the default collection mode. In pulse data mode the data received is the data sampled by the ADCs, without any processing. The pulse data mode can be used with any trigger if fixed length records are used, and with the level trigger if variable length records are used. No padding data is inserted in this mode. Fig. 6 illustrates the pulse data mode with both fixed length and variable length records using the level trigger.

4.9.2 Every *N*th Record

The every *N*th record mode behaves in the same way as the raw pulse data mode with the exception that only every *N*th record is transferred to the host. The reduction factor *N* is specified in the API call

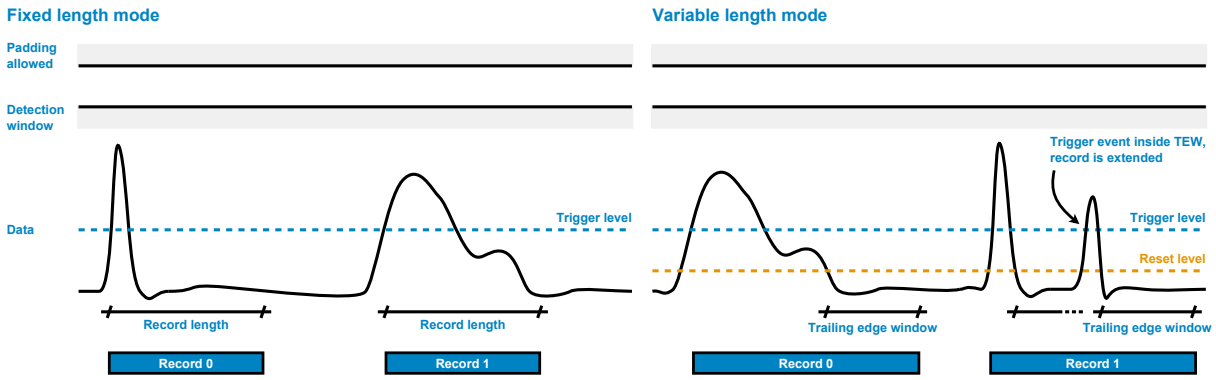


Figure 6: Data collection mode 0 (pulse data).

`PDSetupCharacterization()` and is *only* used in this mode.

Example

If the reduction factor is set to 5, every 5th record will be transferred to the host.

4.9.3 Pulse Data with Padding

The pulse data with padding mode will output padding records if the amount of data collected is less than the minimum frame length. The padding records will have all its data equal set to zero and the LSB of the *User ID* header field set to 1. Pulse data with padding can either be configured to discard (mode 3) or to keep (mode 4) data outside of the detection window.

Important

Mode 4 is not supported for the metadata channels

Fig. 7 illustrates the two different modes. The padding record timestamp and record number is undefined and should not be used.

Example

If the minimum frame length is set to 1024 samples and only 800 samples of data are captured, an additional record with 224 samples will be transmitted to the host computer.

Warning

The timestamp and record number for the padding record is undefined.

4.10 Pulse Metadata

The individual metadata fields are described in Section 4.6 and Fig. 8 presents how the mode acts on a stream of pulse data with a detection window. The detection window trigger will generate a record which is populated with metadata packets. If no pulses are detected during a detection window, empty metadata data packets will be transferred to the host.

Table 6 describes the layout of the metadata packets. The byte column refers to the position in the returned array. One record can consist of one or several metadata packets. The size of each metadata packet is 8 bytes. Packets used for padding have all fields equal to zero and should be discarded.

Table 7 describes the format of the returned data for each record in pulse characterization mode.

Note

Invalid packets are identified by all fields being equal to zero.

Table 6: The metadata packet structure.

Field	Offset (bytes)	Size (bytes)	Format
Pulse peak timestamp	0	4	32-bits unsigned integer
Pulse peak value	4	2	16-bits signed integer
Pulse width	6	2	16-bits unsigned integer

Table 7: This table describes how the contents of a pulse metadata record is structured.

Byte	Description	Comment
7:0	Metadata packet 0	Analysis result of the first pulse in the detection window
15:8	Metadata packet 1	Analysis result of the second pulse in the detection window
23:16	Metadata packet 2	Analysis result of the third pulse in the detection window
…	…	…

5 User Application Example

This section aims to explain the example code provided for ADQ7-FWPD. The ADQAPI reference guide [4] 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 Example in C

An example in C is included in the release archive and located in the `examples/` directory. The example application consists of four files listed in the hierarchy structure below. The example application consists of the source files listed in the hierarchy structure below.

`examples/c/fwpd/`

`ADQAPI_FWPD_example.c`	Main example file, general setup and digitizer identification.
`example_adq14.c`	ADQ14 configuration and data collection.
`example_adq7.c`	ADQ7 configuration and data collection.
`formatter.h`	Definition of console output formatting functions.
`formatter.c`	Implementation of console output formatting functions.
`streaming_header.h`	Definition of the ADQ7 record header for triggered streaming.

The example is verbose by design and is intended as reference for deigning custom applications. The example will configure the device, acquire all the data, save the data to file if instructed to do so and finally return dynamically allocated memory and delete the ADQ control unit object. If the `save_data_to_file` variable is set, the data and header information will be saved to files in the current working directory.

5.1.1 Python Scripts

Two Python scripts are included: `plot.py` and `parse_metadata.py`. The scripts require Python 3, Numpy and Matplotlib. The `plot.py` script can be used for plotting pulse data and the `parse_metadata.py` script can be used to parse and print metadata records.

6 Troubleshooting

This section aims to provide 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 the use case such as the pulse detection 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.

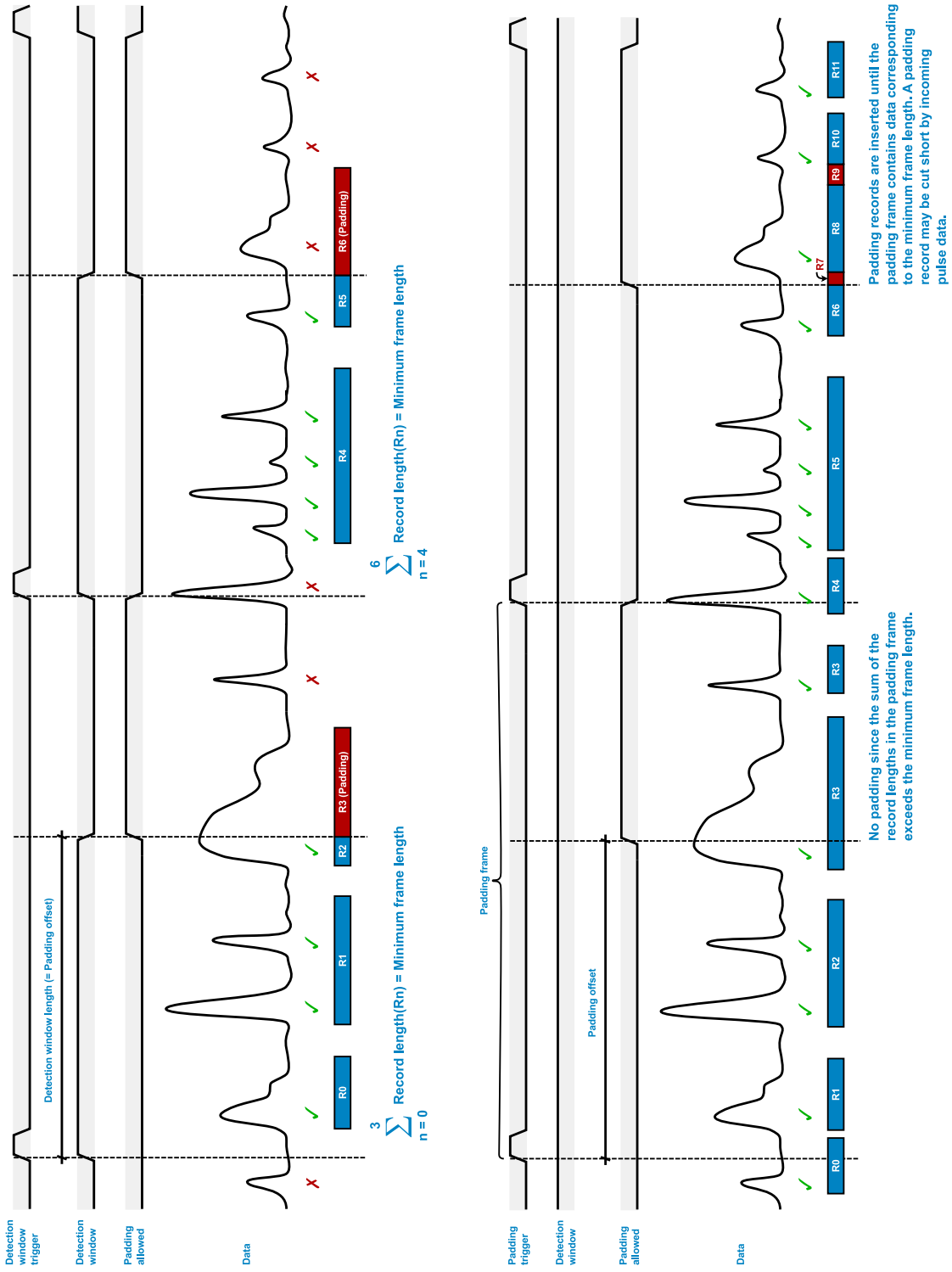
References

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

[2] Teledyne Signal Processing Devices Sweden AB, *17-2017 ADQ7DC datasheet*. Technical Specification.

[3] Teledyne Signal Processing Devices Sweden AB, *19-2226 ADQ7WB datasheet*. Technical Specification.

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



(a) Data collection mode 3 (with detection window).

(b) Data collection mode 4 (without detection window).

Figure 7: Two timing diagrams demonstrating the insertion of padding data into the data stream in collection modes 3 and 4.

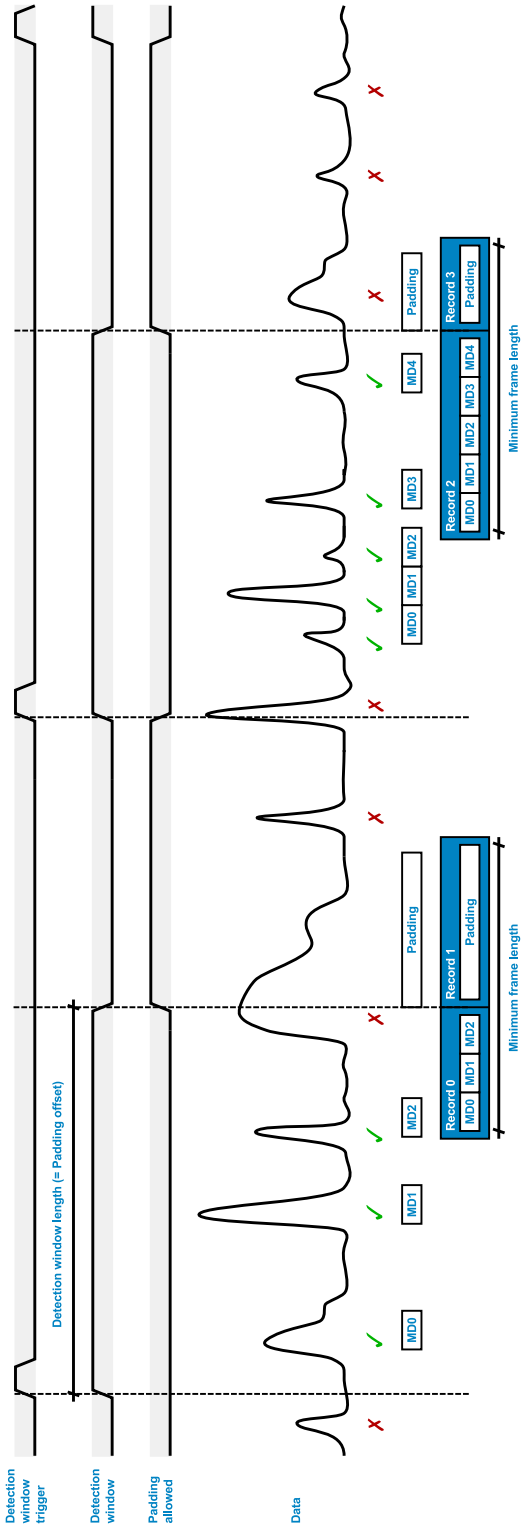


Figure 8: Behavior of the pulse metadata channel. In this case, the source channel may for example use collection mode 3 with both trigger arguments in `PDSetupCharacterization()` set to use the external trigger.

TELEDYNE SP DEVICES
Everywhereyoulook™

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

Copyright © 2023 Teledyne Signal Processing Devices Sweden AB

All rights reserved, including those to reproduce this publication or parts thereof in any form without permission in writing from Teledyne SP Devices.