TELEDYNE SP DEVICES
Everywhereyoulook™

ADQ7-FW2DDC

User Guide

Author(s): Teledyne SP Devices

Document ID: 19-2335
Classification: Public
Revision: 2024.2
Date: 2024-09-26

Contents

Document History

Section	Description	
Revision 2023.1		2023-01-26
	The document version is now based on the release names, e.g. “2023.1”, rather than a letter of the alphabet.	
Revision A		2019-12-03

1 Introduction

This document presents the user guide of the ADQ7 digitizer running the dual digital down-converter (FW2DDC) firmware.

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
AFE	Analog front-end
API	Application programming interface
DRAM	Dynamic random access memory
DDC	Digital down-converter
FW	Firmware (digitizer feature set)
GSPS	10^9 samples per second
NCO	Numerically controlled oscillator
RAM	Random access memory
SDK	Software development kit
SDR	Software-defined radio

2 Features

This section provides an overview of ADQ7-FW2DDC along with brief descriptions of some of its core features and limitations. Detailed information may be found in Section 3.

2.1 Specification

The DDC specification can be found in the ADQ7-FW2DDC datasheet [1]. For the general specification of the ADQ7 digitizer, please refer to the ADQ7DC data sheet [2] or ADQ7WB data sheet [3].

2.2 System Overview

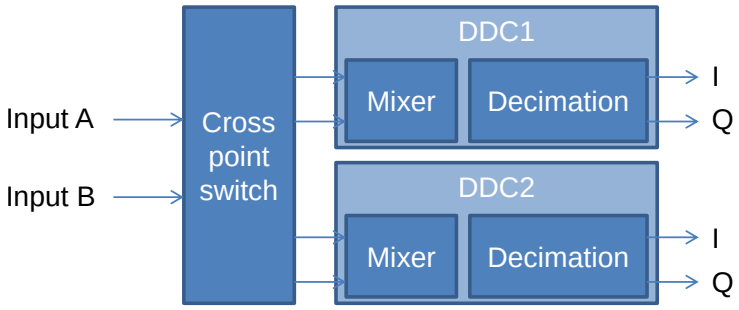

Figure 1: FW2DDC down-converter block diagram.

The FW2DDC firmware package for ADQ7 contains two digital down-conversion blocks, which are capable of frequency translation using quadrature mixers and decimation filtering for SNR improvement, data rate reduction and channel selection.

The two analog inputs can be mapped in various different ways to the two DDCs depending on the user requirements, by configuring a crosspoint switch.

2.3 Crosspoint Switch

The crosspoint switch precedes the two DDC blocks, and allows mapping of the two analog channels to the complex-valued inputs of the DDCs. Each DDC can have its own individual crosspoint configuration. The available modes are listed in Table 2.

For the differential input and the complex-valued input settings, the incoming data is scaled down by a factor of 2. This is done to avoid overrange conditions in the DDCs.

2.4 Quadrature Mixer

Each quadrature mixer consists of two parts: an NCO, and a complex-valued multiplier. The NCO generates complex-valued numbers at the same rate as the incoming sample data, corresponding to the value of a complex exponential:

$$x_{\mathrm{NCO}}[n] = e^{j2\pi\varphi[n]}$$

Table 2: FW2DDC crosspoint modes

Mode	DDC I	DDC Q	Description
0	A	0	Real-valued input from channel A
1	B	0	Real-valued input from channel B
2	(A/2)	(B/2)	Complex-valued input from both channels
3	(A-B)/2	0	Differential input from both channels

The phase value φ is set by a 32-bit accumulator, where the phase increment per sample period can be set by the user:

$$\varphi[n+1] = \varphi[n] + 2\pi\frac{k_{inc}}{2^{32}}$$

This allows synthesis of 2^{32} different frequencies spanning from $-f_s/2$ to $f_s/2$.

The input I/Q data to the DDC and the synthesized I/Q data from the NCO are multiplied together using the complex-valued multiplier. A multiplication in the time domain is equivalent to a convolution in the frequency domain. The complex exponential from the NCO is equivalent to a single dirac impulse in the frequency domain, and the result of the multiplication then becomes a pure frequency translation which shifts the entire frequency spectrum without aliasing.

In practice, the NCO output is not an ideal complex exponential due to quantization errors. The SFDR of the NCO signal is specified in the ADQ7-FW2DDC datasheet.

Note

Users that are only interested in the decimation filtering part of FW2DDC can simply set the set the NCO frequency to 0 Hz to avoid frequency translation

2.5 Decimation Filters

The decimation filtering consists of a cascade of half-band filtering stages. Each stage filters the incoming data with a pass-band from 0 to $0.2f_s$, and a stop-band from $0.3f_s$ to $0.5f_s$, and then down-samples the data by a factor of 2. The result is a usable bandwidth of 80% of the Nyquist band in the decimated sample rate. The remaining 20% may contain aliased content, as that is where the transition band of the filter is located. Fig. 2 shows the frequency characteristic of the half-band filter used in FW2DDC.

The DDC can be configured to output the data from any half-band filter in the filtering chain, which allows the user to select any power of two decimation factor between 1 and 2^{31}.

Note

The decimation filtering replaces the sample skip feature that is present in the FWDAQ firmware. Sample skip cannot be used together with FW2DDC.

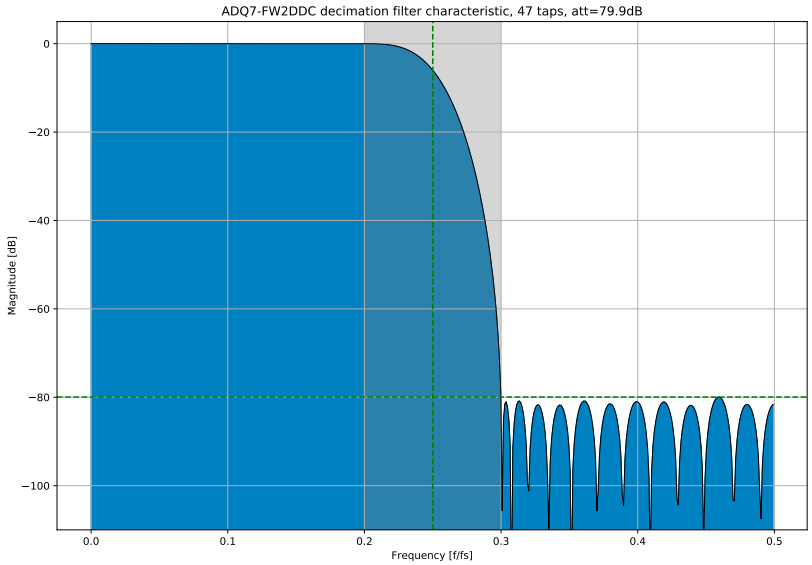

Figure 2: Decimation filter frequency characteristic (before downsampling).

2.6 Equalizer

Each DDC contains an equalizer, which consists of a general-purpose programmable FIR filter. All the filter taps are programmable by the user, and depending on the how the filter is configured it can either operate on the I/Q data as a complex-valued filter, or as two different filters on I and Q separately with real-valued coefficients.

The default state of the equalizer filters are a single one-valued tap surrounded by zero-valued taps, to avoid affecting the signal.

Note

It is only possible to route the data through the equalizer when decimation factors greater than or equal to 2^3 are used.

3 Detailed System Description

This section provides additional details on important system features, some of which have already been described briefly.

3.1 Crosspoint Switch

The crosspoint switch is configured using a single API command: `SetCrosspointSDR()`. It must be called once for each active DDC, with a mode argument corresponding to the different available settings in Table 2.

3.2 Quadrature Mixer

To set the mixer frequency, the API command `SetMixerFrequency()` must be called for each DDC. The frequency input argument is in units of Hz. It is also possible to adjust the phase of the NCO for each mixer, by using the `SetMixerPhase()` command, where the phase input argument can be set between 0 and 2π.

Note

The frequency specified via SetMixerFrequency is what the mixer NCO will be set to. If you, for example, want to receive signal band located at a center frequency of +400 MHz, the mixer frequency should be set to -400 MHz to bring the channel to baseband.

3.3 Decimation Filters

The decimation factor is set for each down-converter using the API command `SetChannelDecimation()`.

3.4 DDC Synchronization

After setting decimation factors and mixer frequencies for the different DDC blocks, it is necessary to perform a synchronization step to start the channels at the same time for a known phase relationship between the different mixers and decimation filters.

There are two alternatives for DDC synchronization. The easiest way is to use the `ForceResynchronizationSDR()` API command. This simply sends a synchronization pulse from the API via a register, which starts the mixer and decimation filters synchronously across the DDCs.

If instead synchronization against an external event is needed (for example when operating two ADQ7 units simultaneously with requirements on synchronous NCOs and decimation filters across the two units), the alternative is to use the timestamp synchronization framework of the ADQ7. Performing a timestamp synchronization also synchronizes the DDCs. This is set up via the API command `SetupTimestampSync()` and can be configured to act on an external synchronization pulse via the TRIG or SYNC connectors.

3.5 Output Data Format

Every decimation filtering stage removes some of the broadband noise from the input signal. For narrowband acquisitions with large decimation factors, the ENOB of the resulting data can exceed 16 bits. Therefore, FW2DDC uses a 32-bit data path internally in the digitizer. The user can configure the digitizer to either quantize the result to 16-bit data before transferring it to the host system (which is useful for reducing the data rate), or to acquire the full 32-bit data to preserve the SNR. The data format is selected via the acquisition parameter `bytes_per_sample`. Refer to the data transfer [4] for more information.

Note

The 32-bit data format may only be enabled together with the decimation filter.

4 Software Examples

This section aims to explain how to write the core code in the user application interfacing to ADQ7-FW2DDC. The ADQAPI reference guide [5] 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. For general information on the architecture of the ADQ7 digitizer and how data collection is structured, please refer to the ADQ7 manual [6].

Note

Since ADQ7-FW2DDC extends the functionality of the base ADQ7 product, most of the functions listed as valid for ADQ7 in the ADQAPI reference guide [5] are valid for ADQ7-FW2DDC as well.

4.1 Typical Setup Sequence

The following is a rough outline for the setup sequence of a FW2DDC application:

Table 3: FW2DDC setup sequence.

	Setup stage	Relevant API commands
1	Initialize the digitizer.	`CreateADQControlUnit()`, `FindDevices()`
2	Configure the data acquisition, data transfer and data readout processes. Set the data format via the acquisition parameter `bytes_per_sample`	Refer to the data transfer and data readout user guide [4].
3	Configure crosspoint switch for each DDC.	`SetCrosspointSDR()`
4	Set the mixer frequency for each DDC.	`SetMixerFrequency()`
5	Set the decimation factor for each DDC.	`SetChannelDecimation()`
6	If the equalizer is to be used, setup equalizer for each DDC.	`SetEqualizerSDR()`
7	Synchronize DDCs, either via a software command, or via an external pulse.	`ForceResynchronizationSDR()`, `SetupTimestampSync()`
8	Configure trigger source and set up streaming.	`SetTriggerMode()`, `TriggeredStreamingSetup()`, `ContinuousStreamingSetup()`
9	Start collecting data.	`StartDataAcquisition()`, `WaitForRecordBuffer()`

4.2 Example Source Code

4.2.1 C/C++

The main FW2DDC-example is written in C, and can be found in the release archive. The example supports both Windows and Linux.

4.2.2 Python

A Python example can be found in the PyADQ package as `adq7_fw2ddc_streaming_example.py`.

4.3 ADCaptureLab

The FW2DDC package for ADQ7 is not supported by ADCaptureLab.

5 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 DDC 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.

5.1 Firmware License

ADQ7-FW2DDC requires a valid firmware license to be able to operate properly. An unsuccessful license validation will result a trace log error message and data collection will be disallowed. The digitizer licenses are managed with the ADQLicenseUtil tool, included in the SDK installation directory.

If the digitizer unit was ordered with together with FW2DDC, it will come preprogrammed with a FW2DDC license. If FW2DDC is ordered for a unit already in the field, the license will need to be uploaded manually to the unit.

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.

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, *19-2325 ADQ7-FW2DDC Datasheet*. Technical Specification.

[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, *20-2465 ADQ14 ADQ7 ADQ8 Data Transfer and Data Readout User Guide*. Technical Manual.

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

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

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