SS-OCT Solution User Manual for SA331 - 12-bit DAQ Card/Module

Up to 3.125 GS/s sampling rate

Up to 2 MHz A-scan rate

Getting Started Guide

June 2024

Table of Content

1 Introduction

This document gives an overview of the Acqiris solution to support swept source real time OCT processing (SS-OCT). It also includes helpful hints to get started.

The SS-OCT solution includes:

- A DAQ Card/Module with on-board FPGA
- A dedicated firmware with real time processing, including dynamic K-space re-mapping and fast readout of A/B/C-scans to the host computer.
- C++ and Python examples
- A graphic user interface GUI for live and offline data display to acquire and visualize raw/processed data (Acqiris AQ4 SS-OCT GUI)

The specifications in this document are preliminary values.

Main specifications

Parameter	Value
Number of OCT channels:	1 or 2
Resolution	12-bit
Max A-scan	2 MHz
Data format	- Raw data - Remapped data

Supported modes

A. 1x OCT channel @ at 3.125 GS/s & K-clock channel @ at to 3.125 GS/s*
B. 2x OCT channels @ at 1.5615 GS/s & K-clock channel @ at to 3.125 GS/s*

NOTE ***** Digital downsampling available onboard

2 Main Features

Hardware	***SA331 – 12 bits***
Configuration	*1 OCT channel @ 3.125 GS/s* *1 K-clock channel @ 3.125 GS/s*
FW Version	3.11.236 or higher
Dual OCT channel	
Binary decimation	
Fractional resampler	☑ GEN3
UP/DOWN Sweep support	Both
Acquisition/A-scan size max	64k/64k
Acquisition/A-scan size increment	64 samples
Resampling mode	☑ ADV
Channel FIR (IN1/IN3)	☑ 25/25 taps
K-clock filtering	☑ GEN2
K-clock calibration	☑ GEN2
Background subtraction	
Dispersion compensation	
FFT with zero padding and windowing	
Standard average	
Moving average	
Analog output basic	☑
Analog output advanced	☑
LUT Color Mapping	
Real-time phase stabilization	☑
C++ driver and examples	☑
Python driver and examples	☑

3 Software Installation

Refer to Step 3: Install the Software described in the SA331P Startup Guide.

4 Setup, Connections & Input Characteristics

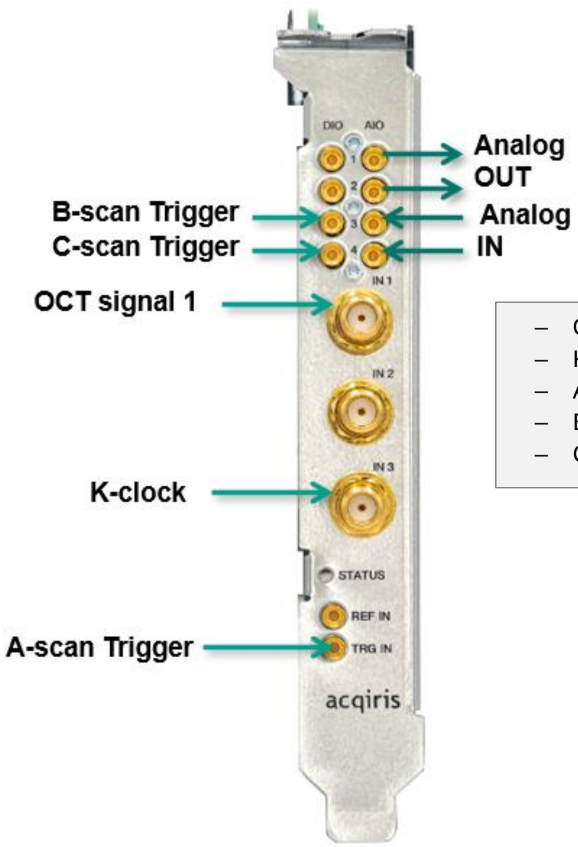
Connect the signals as presented on the figure below. The minimum requirement is to connect the SS-OCT signal, the K-clock, and the A-scan trigger.

The B-scan trigger sync is not mandatory, as the software can run in free-running mode. Connecting B-scan is required for synchronization when capturing 2-D OCT.

The C-scan trigger sync is not mandatory, as the software can run in free-running mode. Connecting C-scan is required for synchronization when capturing 3-D OCT.

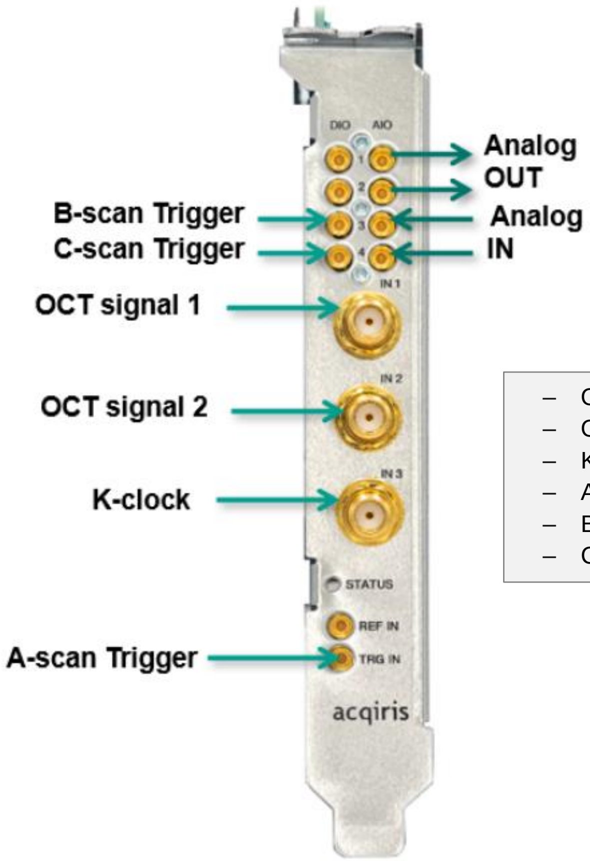
The K-clock signal must be analogue and centered around DC.
The channel vertical offset can help to center the signal.
In case only a digital K-clock is available a low pass filter can be used.

1 OCT channel @ 3.125 GS/s & K-clock channel @ 3.125 GS/s



- OCT signal should be connected to **IN1**
- K-clock should be connected to **IN3**
- A-scan trigger should be connected to TRIG IN
- B-scan trigger (optional) should be connected to DIO 3
- C-scan trigger (optional) should be connected to DIO 4

2 OCT channel @ 1.5615 GS/s & K-clock channel @ 3.125 GS/s



- OCT signal 1 should be connected to **IN1**
- OCT signal 2 should be connected to **IN2**
- K-clock should be connected to **IN3**
- A-scan trigger should be connected to TRIG IN
- B-scan trigger (optional) should be connected to DIO 3
- C-scan trigger (optional) should be connected to DIO 4

Connector	Type	Description
DIO 1 (TRG OUT)	MMCX-V	Trigger Out signal (programmable). 2.2 V typ on 50 Ω charge
DIO 2, 3, 4	MMCX_V	User configurable digital Input / Output signal. DC coupling, LVCMOS 3.3 V. Output: 50 Ω source, Input: +5 V max.
AIO 1, 2 (AN OUT)	MMCX-V female	Application dependent analog signal from a 12-bit DAC, controlled by the internal FPGA. DC coupling, 300 Ω source, programmable output up to ± 10 V.
AIO 3, 4 (AN IN)	MMCX-V female	Application dependent analog signal from a 12-bit ADC, controlled by the internal FPGA. DC coupling, programmable input up to ± 10 V.
IN 1, 2, 3	SMA female	Analog signal inputs, DC-coupled and 50 Ω terminated. The input full scale ranges are : **IN 1, 2 Voltage** — **800 mV FSR** Recommended maximum operating voltage — ±1.2 Vpk **IN 3 Voltage** — **1.1 V FSR** Recommended maximum operating voltage — ±1.2 Vpk
REF IN	MMCX-V female	External reference clock input, AC coupled and 50 Ω terminated. It can accept a 10 MHz or a 100 MHz signal from -3 to +3 dBm.
TRG IN	MMCX-V female	External trigger input, 50 Ω DC terminated, ± 5 V range.

4.1 A-scan size

	Condition	Min	Max	Comment
Size		64	65536	Multiple of 64

4.2 A-scan trigger

	Condition	Min	Max	Comment
Frequency		-	2 MHz	
Coupling		DC		
Impedance		50 Ohms		
Amplitude		± 0.5 V	± 5 V	
Pulse width		TBC	-	Positive or negative pulse
Threshold	Programmable			
Pre-trigger delay		0	-16k Sa	Delay in number of samples, acquired before the A-scan trigger occurs
Post-trigger delay		0	16k Sa	Delay in number of samples, from the A-scan trigger and the A-scan acquisition starts

4.3 B-scan trigger

	Condition	Min	Max	Comment
Frequency		--	A-scanPeriod x A-scanNbr	The maximum B-scan rate depends on the B-scan size
Level Low		0.0 V	0.2 V	DC coupling, LVCMOS
Level High		1.0 V	3.3 V	

4.4 C-scan trigger

	Condition	Min	Max	Comment
Frequency		--	B-scanPeriod x B-scanNbr	The maximum C-scan rate depends on the C-scan size
Level Low		0.0 V	0.2 V	DC coupling, LVCMOS
Level High		1.0 V	3.3 V	

4.5 OCT-channel input

	Condition	Min	Max	Comment
Input Voltage	0.8 V Full scale range (+/- 400 mV)	--	90% of FSR	Max hardware voltage ± 1.2 Vpk
Frequency		--	1.3 GHz	OCT frequency sweep shall be kept within limits

4.6 K-clock input

	Condition	Min	Max	Comment
Amplitude	1.1 V full scale range (+/- 550 mv)	-	90% of FSR	The amplitude of the K-clock should be kept high for a better extraction of the non-linearity
Common mode		-	-	We recommend connecting the K-clock through a High Pass Filter
Frequency		80 MHz	1.3 GHz	K-clock frequency sweep shall be kept within limits
Shape		-	-	Analog K-clock

Recommendations

1. In the case where the K-clock frequency of the used swept source does not satisfy the above limits, an external MZI (Mach–Zehnder Interferometer) can be implemented to supply the K-clock to the DAQ Card / Module.
2. In the case where the K-clock is a digital type, filtering of the harmonics with a low pass filter will improve the performance.

5 Internal Calibration (or Self-Calibration)

The internal calibration (or self-calibration) measures and adjusts the internal timing, gain and offset parameters between the ADCs and against a precise reference.

The supplied software drivers include self-calibration function which can be executed upon user request.

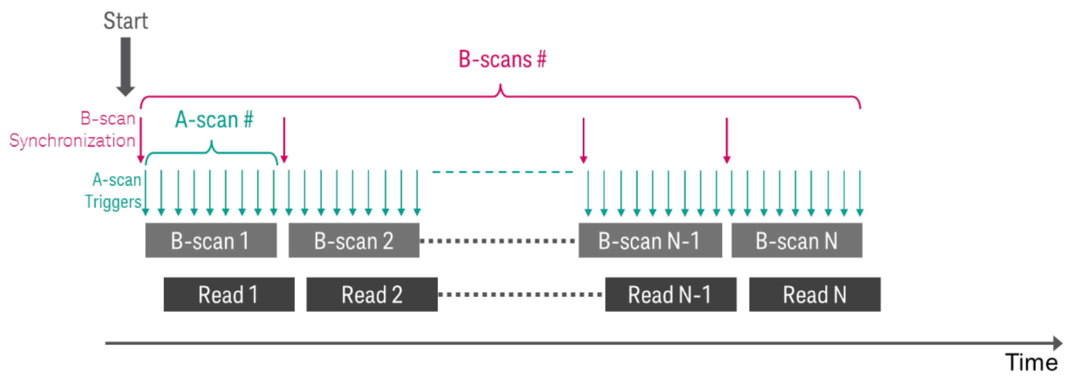
Self-calibration can usually work with signals present at the channel inputs, or trigger input. However, to ensure the best performance, or if the calibration is found to be unreliable (as shown by a calibration failure status), it is recommended to remove or lower the power of such signals.

A self-calibration is required after the initial configuration of the DAQ Card/Module. New self-calibration is only required the first time the full-scale range is changed on one of the channels.

6 Operating Mode and Synchronization

The SS-OCT-Engine is based on the concept of A-scan. Once started, the SS-OCT engine captures, processes, and saves A-scans to the A-scan read-FIFO.
Reading A-scan data can only be done by reading entire A-scans. The on-board FIFO watermark can be monitored using the AvailableAscans parameter from the fetch function.

B-scan size represents the number of A-scans acquired while the laser moves in the first direction (X), leading to capture a 2-D B-scan image.
C-scan size represents the number of B-scan acquired while the laser moves in the 2nd direction (Y), leading to capture a 3-D C-scan image.

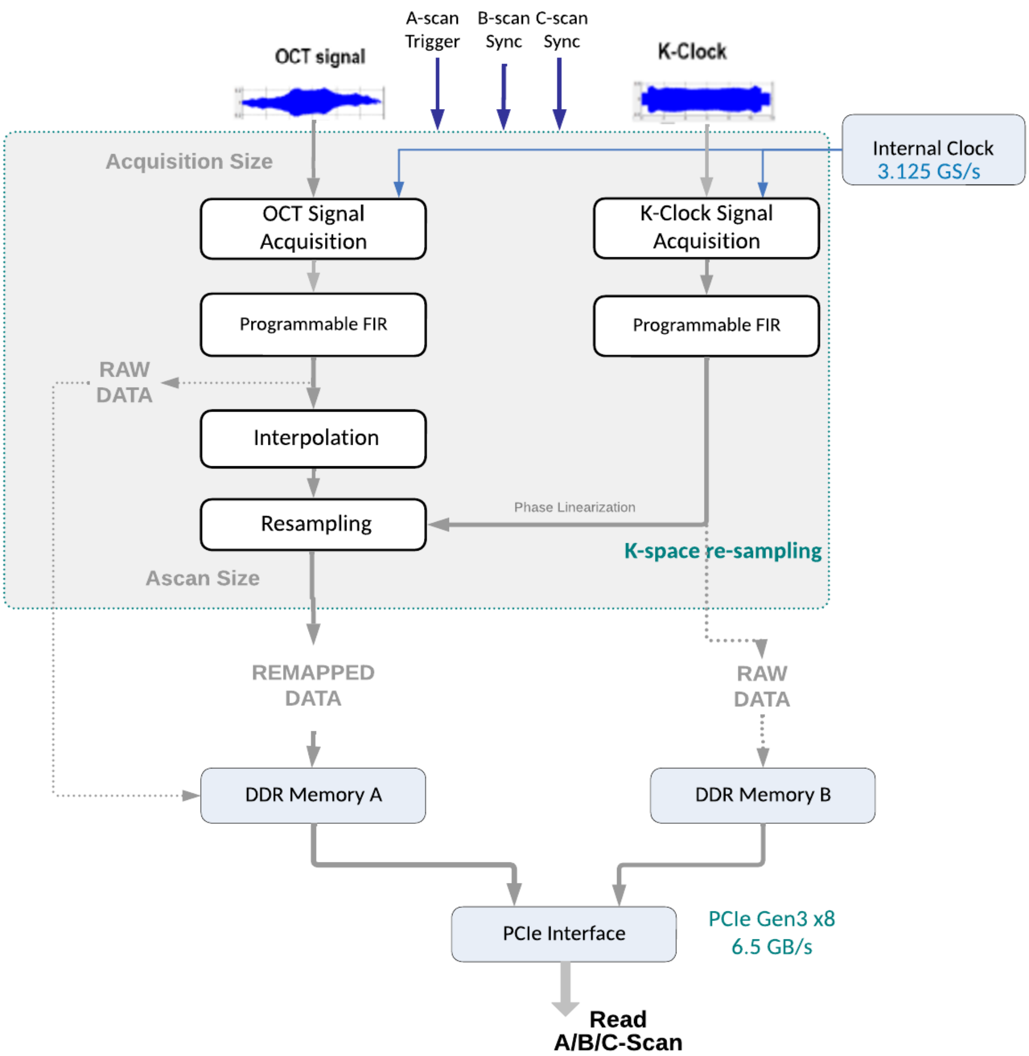


There are three different operating modes:

- **Free running** Once started, B-scans/C-scans are acquired processed and saved continuously to the onboard FIFO until the SS-OCT Engine is stopped.
- **Software trigger** Once started, upon each software trigger, a single B-scan/C-scan is acquired and available for readout.
- **PIO Sync mode:** Once started, upon each pulse trigger, a single B-scan/C-scan is acquired and available for readout.

7 Signal Processing Sequence

SS-OCT signal acquisition and processing steps implemented in real-time are detailed in the diagram below.



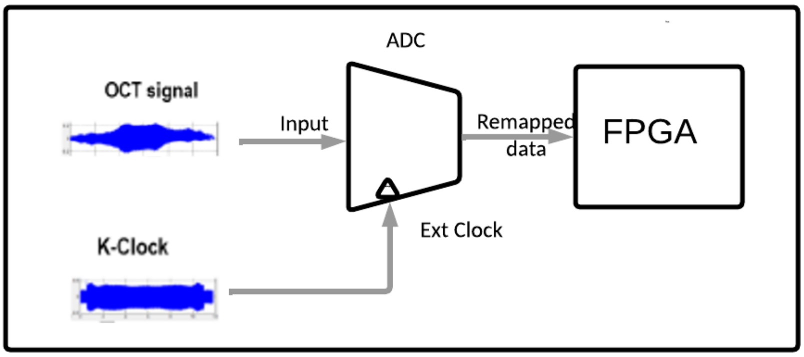
8 K-clock Direct Sampling vs Digital K-clock Resampling

In Swept-Source Optical Coherence Tomography (SS-OCT), both K-clock direct sampling and digital resampling are techniques for acquiring and processing the interference signal to move to the k-space (optical wavenumber) domain.

Acqiris implemented the Digital K-clock technique as it is the most recognized technique in the literature for a scalable, effective artifacts free resampling.

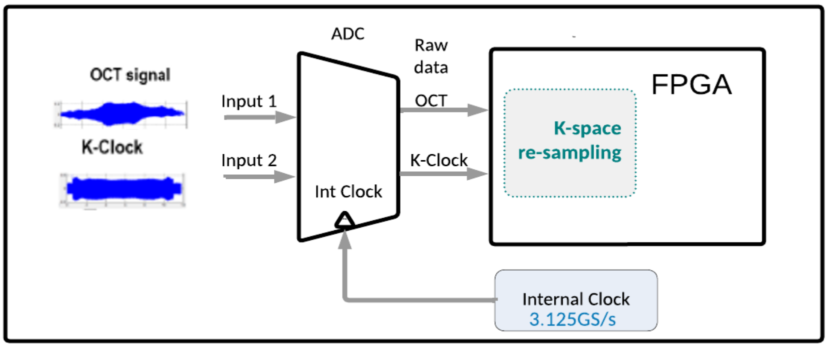
Hereafter a brief illustration of the main differences between these two techniques:

- **K-clock Direct Sampling:** Utilizes a dedicated hardware clock signal, the K-clock, generated from the swept source itself to trigger the data acquisition system (DAQ) at specific points during the source's wavelength sweep. Ideally, these trigger points correspond to equally spaced intervals in the optical wavenumber domain (k-space).



K-clock Direct Sampling

- **Digital K-clock Resampling:** Acquires data at a constant rate using a standard clock, regardless of the source's sweep pattern. The raw data is then mathematically resampled in post-processing to achieve a uniform distribution in k-space.



Acqiris Digital K-clock Resampling

9 Data Format

The SS-OCT Engine has two selectable data formats:

1. **Raw Data:**
   The raw data is acquired on all active channels.
   a. Samples of the OCT-signal waveform (IN1).
   b. Samples of the K-clock waveform (IN3).

2. **Remapped data**:
   This mode includes acquisition and K-space resampling. The data read is in the K-space domain.

10 Signal Processing

10.1 K-Space resampling

An SS-OCT system needs to resample the detected OCT fringe signal to provide remapped data evenly sampled in intervals in the optical wavenumber domain (K-space).

The resampling is performed based on the K-clock phase information used to compute the resampling step for the OCT signal.

10.2 Channel Programmable FIR filters

The programmable FIR on both channels contributes to reduce the in-band noise and improve the signa/noise ratio (SNR).
Two independent FIR filters are available on OCT (IN1) and K-clock (IN3) channels. They can be configured independently avoiding unwanted frequencies folding in the band of interest and smoothing a noisy or digital K-clock. There are 25 taps (coefficients) for each filter. See section 13.7. Custom FIR channel filter for more information.

10.3 Interpolation

Interpolation plays a crucial role in the SS-OCT resampling process by providing a higher resolution representation of the OCT signal in the K-domain. This denser grid is essential for accurate resampling, ensuring optimal image quality and reducing artifacts in the final OCT image.

10.4 Phase linearization

The k-clock signal itself is not linear, but it provides the necessary information to achieve uniform k-sampling in SS-OCT. By linearizing the k-clock, we can ensure accurate image reconstruction.

10.5 Resampling

The interpolated OCT signal is resampled at the positions determined by the calculated phase steps. This effectively creates a new OCT signal with a uniform sampling rate in the K-domain, which is essential for subsequent processing steps like Fourier transformation to generate the depth profile image. See section 13.6 Resampling for more information.

11 Managing Delays

The **A-scan trigger delay** is the delay between the A-scan trigger time and the beginning of the recording of the OCT signal.
It can be configured using the parameter `AQ4SSOCT_ATTR_ASCAN_TRIGGER_DELAY_PRIMARY_SWEEP` (in C++) / `instr.AScan.TriggerDelayPrimarySweep` (in Python).

The **K-clock delay** is the delay between the trigger and the K-clock signal.
It can be configured using the parameter `AQ4SSOCT_ATTR_KCLOCK_DELAY_PRIMARY_SWEEP` (in C++) / `instr.KClock.DelayPrimarySweep` (in Python).

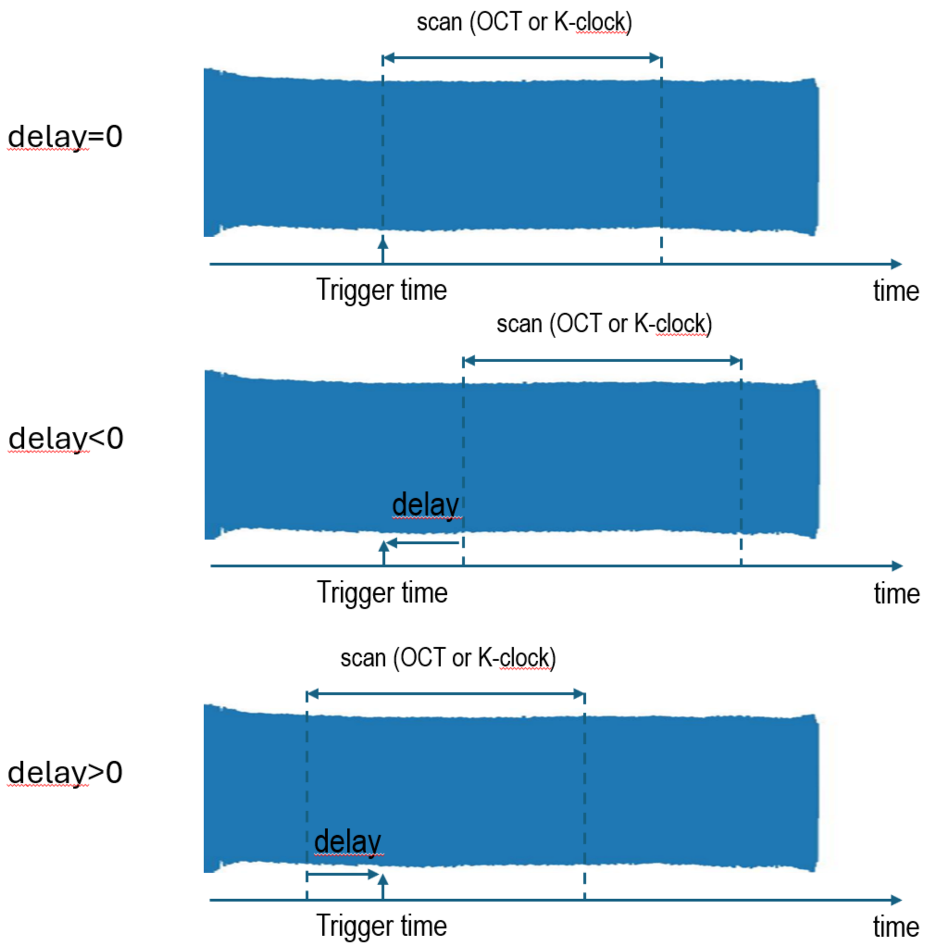
User can program independently the A-scan trigger delay and the K-clock delay.
The accepted range for the delay parameters is [X, 16383+X) samples where X is within [-16383, 0] samples.

For example:

- if all the delays are ≥ 0, the accepted range is [0; 16383] for all the delay parameters. It is defined in samples at 3.125 GS/s, therefore this corresponds to a delay of up to ~ +5.2us.

- if all the delays are ≤ 0, the accepted range is [-16383; 0] for all the delay parameters. This corresponds to a delay of up to ~ -5.2us.

The following diagram illustrates the timing sequence for different delay configurations.



12 Digital Input / Output signals

12.1 Signals available

DIO1	DIO2	DIO3
Disabled Out-LowLevel Out-HighLevel Out-AScanSync Out-AScanEna Out-AScanUp Out-CScanSync	Disabled Out-LowLevel Out-HighLevel Out-BScanSync Out-AScanEna Out-AScanUp In-BScanSync In-BScanSyncNeg	Disabled Out-LowLevel Out-HighLevel Out-CScanSync Out-AScanEna Out-AScanUp In-CScanSync In-CScanSyncNeg

12.2 Signals description

Disabled:	The IO is disabled.
Out-LowLevel:	The IO is configured as output. The voltage is set to logic 0.
Out-HighLevel:	The IO is configured as output. The voltage is set to logic 1.
Out-AScanSync:	The IO is configured as output. The output is a replica with a delay of the A-scan trigger.
Out-BScanSync:	The IO is configured as output. The output is a replica with a delay of the B-scan trigger.
Out-CScanSync:	The IO is configured as output. The output is a replica with a delay of the C-scan trigger.
Out-AScanEna:	The IO is configured as output. It indicates the period when the A-scan trigger is accepted.
Out-AScanUp:	The IO is configured as output. If high, the currently processed A-scan is an up-sweep. If low, it is a down-sweep.
In-BScanSync:	The IO is configured as input. It is ready to accept a B-scan trigger (positive edge).
In-BScanSyncNeg:	The IO is configured as input. It is ready to accept a B-scan trigger (negative edge).
In-CScanSync:	The IO is configured as input. It is ready to accept a C-scan trigger (positive edge).
In-CScanSyncNeg:	The IO is configured as input. It is ready to accept a C-scan trigger (negative edge).

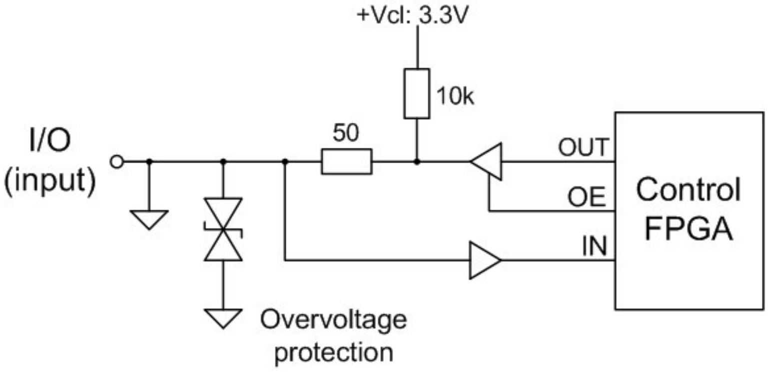
12.3 Signal Logic Levels

The Digital Input / Output IO signals are 3.3 V CMOS compatible (5V Tolerant buffer). The levels shown in the table below should be observed.

Direction	Low level	High level
Input	< 0.8 V	> 2.0 to 3.45 V
Output	In the range 0 to 0.8 V	In the range 1.6 to 3.3 V

As an Input

The input is high-impedance and will be pulled high if unconnected via an internal weak pull-up (10 k pull-up resistor).

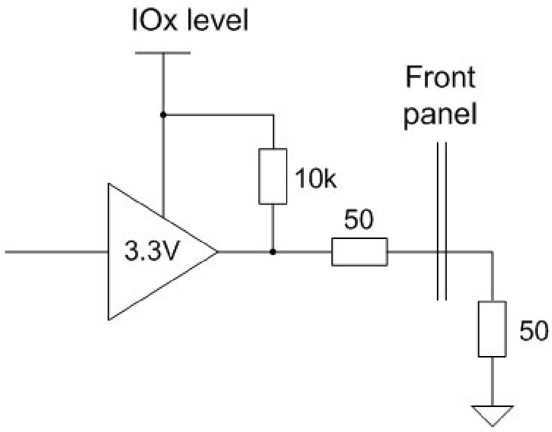


Programmable IO schematic

As an Output

The high level output will typically give 1.6 V into 50 Ω. As can be seen in the diagram below, the 3.3 V output buffer has a 50 Ω resistor in series. Therefore the available output high level voltage will depend on the load applied. In the example below a 50 Ω termination will result in a nominal high level of 1.6 V.

($V_o = (R_{load}/(50 + R_{load})) * 3.3$).



Output equivalent circuit.

13 SS-OCT Programming in C++ / Python

13.1 Introduction

The C++ and Python get started examples provided with the SS-OCT development package help to discover the SS-OCT features.

13.2 Install the examples

IVI-C and Python program examples can be found in the **`Aq4Ssoct-DevPackage`** in the folder **`Examples`**. You may download the Aq4Ssoct-DevPackage from **`Your Dedicated Content`** on the Acqiris Extranet. Please contact support@acqiris.com for further information.

Connections required to run the example programs:

- Connect **`OCT signal to IN1`**
- Connect **`A-scan trigger signal to TRG IN`**
- Connect **`K-clock signal to IN3`**

13.3 Attributes default value and range

You may find detailed documentation of the IVI-C driver API functions, as well as information to help you get started with using the IVI drivers in your application development environment in the IVI Driver reference documentation. The CHM file can be accessed on **`C:\Program Files\IVI Foundation\IVI\Drivers\Aq4Ssoct\Aq4Ssoct.chm`** or from **`Startup Menu > Acqiris > Documentation > Aq4Ssoct IVI Driver<version#> Documentation`**.

NOTE — If you migrate your application from the AqSsoct IVI-C driver, you may refer to the to **`AqSsoct to Aq4Ssoct Software Migration Note`** provided in the **`Aq4Ssoct-DevPackage`** in the folder **`Docs`**.

C++: AQ4SSOCT_ATTR_VERTICAL_RANGE **Python:** instr.Channels['*ChannelName*'].Range	
Attribute type	ViReal64
RepCapIdentifier	Channel1, Channel3
Attribute default value	Channel1: 0.8V Channel3: 1.1V
Range	Channel1: 0.8V Channel3: 1.1V

C++: AQ4SSOCT_ATTR_VERTICAL_OFFSET **Python:** instr.Channels['*ChannelName*'].Offset	
Attribute type	ViReal64
RepCapIdentifier	Channel1, Channel3
Attribute default value	0
Range	±0.6 FSR Channel1: -0.48V ≤ value ≤ 0.48V Channel3: -0.66V ≤ value ≤ 0.66V

C++: AQ4SSOCT_ATTR_ASCAN_ACQUISITION_SIZE **Python:** instr.AScan.AcquisitionSize	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	2048
Range	64 ≤ value ≤ 65536

C++: AQ4SSOCT_ATTR_ASCAN_PRIMARY_SWEEP_SYNC Python: instr.AScan.PrimarySweepSync	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	C++: AQ4SSOCT_VAL_ASCAN_PRIMARY_SWEEP_SYNC_EDGE_NEGATIVE Python: AScanPrimarySweepSync.EdgeNegative
Range	C++: AQ4SSOCT_VAL_ASCAN_PRIMARY_SWEEP_SYNC_EDGE_POSITIVE AQ4SSOCT_VAL_ASCAN_PRIMARY_SWEEP_SYNC_EDGE_NEGATIVE Python: AScanPrimarySweepSync.EdgePositive AScanPrimarySweepSync.EdgeNegative

C++: AQSSOCT_ATTR_ASCAN_TRIGGER_LEVEL Python: instr.AScan.TriggerLevel	
Attribute type	ViReal64
RepCapIdentifier	VI_NULL
Attribute default value	0
Range	-5V ≤ value ≤ 5V

C++: AQ4SSOCT_ATTR_ASCAN_SECONDARY_SWEEP_ENABLED Python: instr.AScan.SecondarySweepEnabled	
Attribute type	ViBoolean
RepCapIdentifier	VI_NULL
Attribute default value	VI_FALSE

C++: AQ4SSOCT_ATTR_ASCAN_TRIGGER_DELAY_PRIMARY_SWEEP Python: instr.AScan.TriggerDelayPrimarySweep	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	0
Range	X ≤ value < 16383+X where X is within [-16383, 0] samples See section 11. Managing Delays

C++: AQ4SSOCT_ATTR_ASCAN_TRIGGER_DELAY_SECONDARY_SWEEP Python: instr.AScan.TriggerDelaySecondarySweep	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	0
Range	X ≤ value < 16383+X where X is within [-16383, 0] samples See section 11. Managing Delays

C++: AQ4SSOCT_ATTR_CONTROL_IO_SIGNAL Python: instr.ControlIOs['*ControlIOName*'].Signal	
Attribute type	ViString
RepCapIdentifier	ControlIO1, ControlIO2, ControlIO3
Attribute default value	Disabled
Range	For ControlIO1: Disabled Out-LowLevel Out-HighLevel Out-AScanSync Out-AScanEna Out-AScanUp Out-CScanSync For ControlIO2: Disabled Out-LowLevel Out-HighLevel Out-BScanSync

	Out-AScanEna Out-AScanUp In-BScanSync In-BScanSyncNeg For ControlIO3: Disabled Out-LowLevel Out-HighLevel Out-CScanSync Out-AScanEna Out-AScanUp In-CScanSync In-CScanSyncNeg

C++: **Python:**	**AQ4SSOCT_ATTR_IMAGE_DESCRIPTOR_ENABLED** **instr.Image.DescriptorEnabled**
Attribute type	ViBoolean
RepCapIdentifier	VI_NULL
Attribute default value	VI_TRUE

C++: **Python:**	**AQ4SSOCT_ATTR_IMAGE_ASCAN_SIZE** **instr.Image.AScanSize**
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	2048
Range	64 ≤ value ≤ 65536, multiple of 32

C++: **Python:**	**AQ4SSOCT_ATTR_IMAGE_BSCAN_SIZE** **instr.Image.BScanSize**
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	2048
Range	1 ≤ value ≤ 65536

C++: **Python:**	**AQ4SSOCT_ATTR_IMAGE_BSCAN_TRIGGER_MODE** **instr.Image.BScanTriggerMode**
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	**C++: AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_FREE_RUNNING** **Python: ScanTriggerMode.FreeRunning**
Range	**C++: AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_SOFTWARE** **AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_PIO_TRIGGER** **AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_FREE_RUNNING** **Python: ScanTriggerMode.FreeRunning** **ScanTriggerMode.Software** **ScanTriggerMode.PioTrigger**

C++: **Python:**	**AQ4SSOCT_ATTR_IMAGE_CSCAN_SIZE** **instr.Image.CScanSize**
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	1
Range	1 ≤ value ≤ 65536

C++: AQ4SSOCT_ATTR_IMAGE_CSCAN_TRIGGER_MODE Python: instr.Image.CScanTriggerMode	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	C++: AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_FREE_RUNNING Python: ScanTriggerMode.FreeRunning
Range	C++: AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_SOFTWARE AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_PIO_TRIGGER AQ4SSOCT_VAL_SCAN_TRIGGER_MODE_FREE_RUNNING Python: ScanTriggerMode.FreeRunning ScanTriggerMode.Software ScanTriggerMode.PioTrigger

C++: AQ4SSOCT_ATTR_IMAGE_DATA_FORMAT Python: instr.Image.DataFormat	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	C++: AQSSOCT_VAL_IMAGE_DATA_FORMAT_RAW Python: ImageDataFormat.Raw
Range	C++: AQSSOCT_VAL_IMAGE_DATA_FORMAT_RAW AQSSOCT_VAL_IMAGE_DATA_FORMAT_REMAPPED Python: ImageDataFormat.Raw ImageDataFormat.Remapped

C++: AQ4SSOCT_ATTR_KCLOCK_DELAY_PRIMARY_SWEEP Python: instr.KClock.DelayPrimarySweep	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	0
Range	X ≤ value < 16383+X where X is within [-16383, 0] samples See section 11. Managing Delays

C++: AQSSOCT_ATTR_KCLOCK_DELAY_SECONDARY_SWEEP Python: instr.KClock.DelaySecondarySweep	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	0
Range	X ≤ value < 16383+X where X is within [-16383, 0] samples See section 11. Managing Delays

C++: AQSSOCT_ATTR_KCLOCK_FILTER_MODE Python: instr.KClock.FilterMode	
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	C++ : AQ4SSOCT_VAL_KCLOCK_FILTER_MODE_DISABLED Python: KClockFilterMode.Disabled
Range	C++: AQ4SSOCT_VAL_KCLOCK_FILTER_MODE_DISABLED AQ4SSOCT_VAL_KCLOCK_FILTER_MODE_DEFAULT Python: KClockFilterMode.Disabled KClockFilterMode.Default

C++: Python:	AQSSOCT_ATTR_KCLOCK_HILBERT_GAIN instr.KClock.HilbertGain
Attribute type	ViReal64
RepCapIdentifier	VI_NULL
Attribute default value	1
Range	0 < value ≤ 3.99

C++: Python:	AQSSOCT_ATTR_KCLOCK_HILBERT_OFFSET instr.KClock.HilbertOffset
Attribute type	ViInt32
RepCapIdentifier	VI_NULL
Attribute default value	0
Range	-128 ≤ value ≤ 127

13.4 Primary and secondary sweep

Attributes or functions that contains `AQ4SSOCT_xxxx_PRIMARY_xxxx` (in C++) or `instr.xxxx.xxxxPrimarySweep` (in Python) correspond to the upsweep and attributes or functions that contains `AQ4SSOCT_xxxx_SECONDARY_xxxx` (in C++) / `instr.xxxxSecondarySweepxxxx` (in Python) correspond to the downsweep.

13.5 Acquisition size / Scan size

The attribute `AQ4SSOCT_ATTR_ASCAN_ACQUISITION_SIZE` (in C++) / `instr.AScan.AcquisitionSize` (in Python) specifies the number of samples acquired per A-scan, before resampling.

The attribute `AQ4SSOCT_ATTR_IMAGE_ASCAN_SIZE` (in C++) / `instr.AScan.AscanSize` (in Python) specifies the scan size in number of points/samples per A-scan, after remapping of the acquisition time window.

Acquisition size should be always ≥ to A-scan size.

When Acquisition size > A-scan size, a downsampling of Acquisition Size / Ascan Size ratio is performed by using the digital resampler available in the FPGA.

Examples:

- If Acquisition size = 2048 and A-scan size = 2048
  Acquisition size and A-scan size are equal, there is no downsampling.

- If Acquisition size = 16384 and A-scan size = 4096
  A downsampling of 4 (16384/4096) is done by the digital resampler in the FPGA.

- If A-scan size = 4096 and you want to perform a downsampling of 8, you can set
  Acquisition size = 8 * A-scan size = 32768

13.6 Resampling

A calibration step is required to calculate the resampling step that will be applied to all acquired A-scans.

1. Configure the channels, A-scan, K-clock (delays, Hilbert gain), and Image parameters with the appropriate settings.
2. Ensure B-scan >= 128
3. Perform the self-calibration (if not done or if the range has changed))
4. Load any custom filter if needed
5. Set the Data Format to Remapped
6. Ensure the Descriptor is enabled
7. Ensure the acquisition is non running (Abort)
8. Call the function "Acquire K-clock" that automatically acquire the K-clock and computes the resampling step
   C++: `Aq4Ssoct_ProcessingAcquireKClock(session, 1000);`
   Python: `instr.Calibration.AcquireKClock(1000)`
9. Run acquisitions

The examples codes show how to operate this mode.

13.7 Custom FIR channel filter

It is possible to use custom FIR channels filters by uploading beforehand the corresponding coefficients on each channel (OCT and/or K-clock).

1. Please note that you must load the coefficients after each new self-calibration.
2. It is possible to apply FIR channels filters to one or both channels (OCT / K-clock).
3. The coefficients for the FIR channels filters are real values in the range [-1, to +1].
4. The number of coefficients should be equal to 25.

Typical sequence:

- Configure the Channels, A-scan, K-clock, Image, …parameters as needed
- Perform the self-calibration. You must load the coefficients after each new self-calibration.

  in C++: `checkApiCall(AqSsoct_SelfCalibrate(session));`
  In Python: `instr.Calibration.SelfCalibrate()`

- Load the custom FIR channels filters on OCT and/or K-clock

  In C++:

```
static const ViInt32 nCoeff = 25;
// For OCT
ViReal64 coeffsOCT[nCoeff] = { -0.00223957, -0.00390468, … ,0 };
checkApiCall(Aq4Ssoct_SetChannelFilter (session, "Channel1", nCoeff, coeffsOCT));
// For K-clock
ViReal64 coeffsKclock[nCoeff] = { -0.00223957, -0.00390468, … ,0 };
checkApiCall(Aq4Ssoct_SetChannelFilter (session, "Channel3", nCoeff, coeffsKclock));
```

  In Python:

```
Coeff = 25
# For OCT
coeffsChannel1 = np.array([ -0.00223957, -0.00390468, … ,0])
```

```
instr.Channels[octChannelName].SetFilter(coeffsChannel1)
# For K-clock
coeffsChannel3 = np.array([ -0.00223957, -0.00390468, … ,0])
instr.Channels[kClkChannelName].SetFilter(coeffsChannel3)
```

To disable the custom channel filters and get back to the default behavior, you can call **`Aq4Ssoct_SelfCalibrate`** (in C++) / **`instr.Calibration.SelfCalibrate()`**(in Python).
It is also possible to load the unitary filter where all values are set to 0 except the 13th which is set to 1.0.

0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
1.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0

The python script below shows how to generate the FIR coefficients.

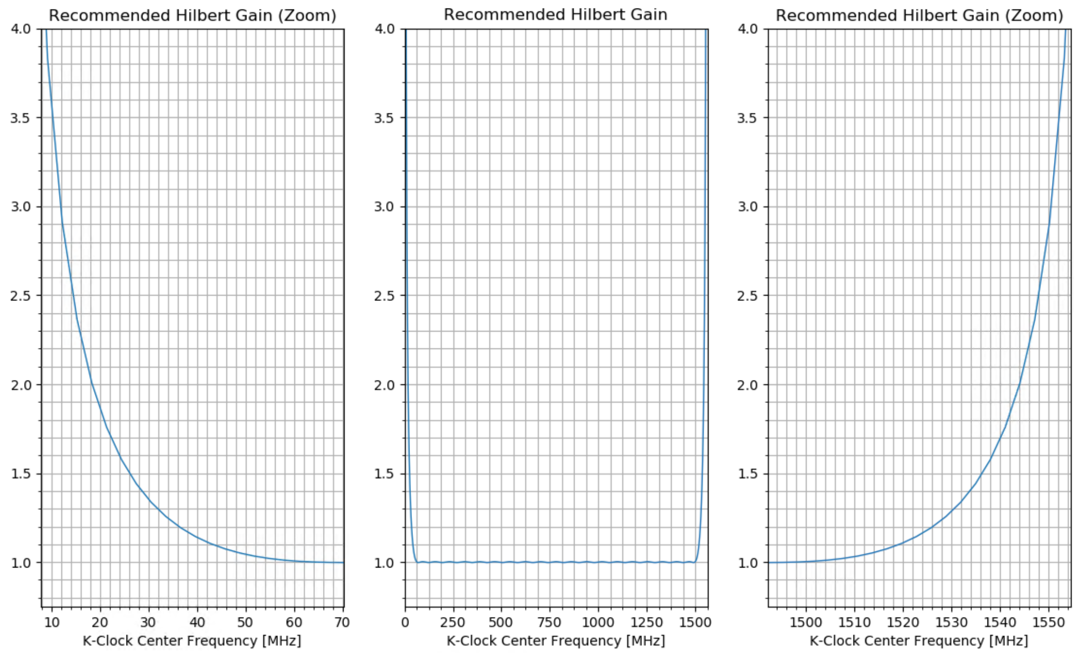
```
# requires Python 3.6 or higher
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

fs = 3.125e9                   # max sampling frequency.
filt_num_taps = 25   # number of taps. it must be odd (required by the signal.firls function)
is_bandpass = False

if is_bandpass: # to generate a 40 MHz to 200 MHz band-pass FIR filter
  filt_sb1 =  10e6 / fs # stop-band 1
  filt_pb1 =  40e6 / fs # pass-band 1
  filt_pb2 = 200e6 / fs # pass-band 2
  filt_sb2 = 300e6 / fs # stop-band 2
  filt_b = signal.firls(filt_num_taps, [0, 2*filt_sb1, 2*filt_pb1, 2*filt_pb2, 2*filt_sb2, 1],
[0, 0, 1, 1, 0, 0], weight = [10, 1, 10])

else:      # to generate a 200 MHz low-pass FIR filter
  filt_pb  = 200e6 / fs # pass-band
  filt_sb  = 300e6 / fs # stop-band
  filt_b = signal.firls(filt_num_taps, [0, 2*filt_pb, 2*filt_sb, 1], [1, 1, 0, 0], weight = [1, 10])
  filt_b /= np.sum(filt_b)

# to plot the FIR filter
w, h = signal.freqz(filt_b)
w = w * fs / (2 * np.pi)

plt.figure()
plt.subplot(1, 2, 1)
plt.stem(filt_b)
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(w/1e6, 20 * np.log10(abs(h)))
plt.grid(True)
plt.xlabel("[MHz]")
plt.ylabel("[dB]")
plt.autoscale(enable = True, axis = 'x', tight = True)

plt.show()

# the resulting filt_b array contains the 25 coefficients of the FIR channel filter
print(filt_b)
```

13.8 Hilbert gain

The purpose of the Hilbert gain is to compensate the response of the Hilbert filter which is not perfect at low and high K-clock center frequencies (< 40 MHz and > 960 MHz) .

It can be configured using the `AQ4SSOCT_ATTR_KCLOCK_HILBERT_GAIN` (in C++) / `instr.KClock.HilbertGain` (in Python) attribute. This can be defined in the range 0 < value ≤ 3.99.

The graphs below provided the recommended Hilbert gain value as a function of the K-clock center frequency in MHz.



For example, if your K-clock is centered at 30 MHz you may set the Hilbert gain to 1.35.

13.9 Data organization

The SS-OCT engine uses the streaming mode. It allows to read multiple A-scans from the host computer while the capture, process, and saving of the next A-scans to A-scan read-FIFO continues. Each A-scan requires its own trigger.

The output data consists of up to 3 streams of data:

- 1 data stream containing the OCT scans (`StreamCh1`)
- 1 data stream containing the K-clock scans (in raw data mode) (`StreamCh3`)
- 1 data stream containing the A-scan descriptors (if enabled) (`StreamDescr`)

Each stream can be read independently and in a time multiplexed manner allowing a fine tuning of the system and application performance.
All the streams are channelled into a single "pipe" which corresponds to the PCIe interface. The size of the “pipe” represents the volume of data flow that can be extracted from the DAQ Card/Module. When the stream data rate is larger than the available PCIe data bandwidth, an overflow occurs and the acquisition stops.

The overflow status can be checked using the `AQ4SSOCT_ATTR_IMAGE_IS_STREAM_OVERFLOW` (in C++) or `IsStreamOverflow` (in Python) parameter.

Each scan (OCT or K-clock) is composed of the data in the selected format. The output data are returned in 16-bit format and contain no headers.

DATA 1	DATA 2	....	DATA N

13.10 Descriptors

If the descriptors are enabled (`AQ4SSOCT_ATTR_IMAGE_DESCRIPTOR_ENABLED` (in C++) / `instr.Image.DescriptorEnabled` (in Python)), descriptors are generated for each A-scan.

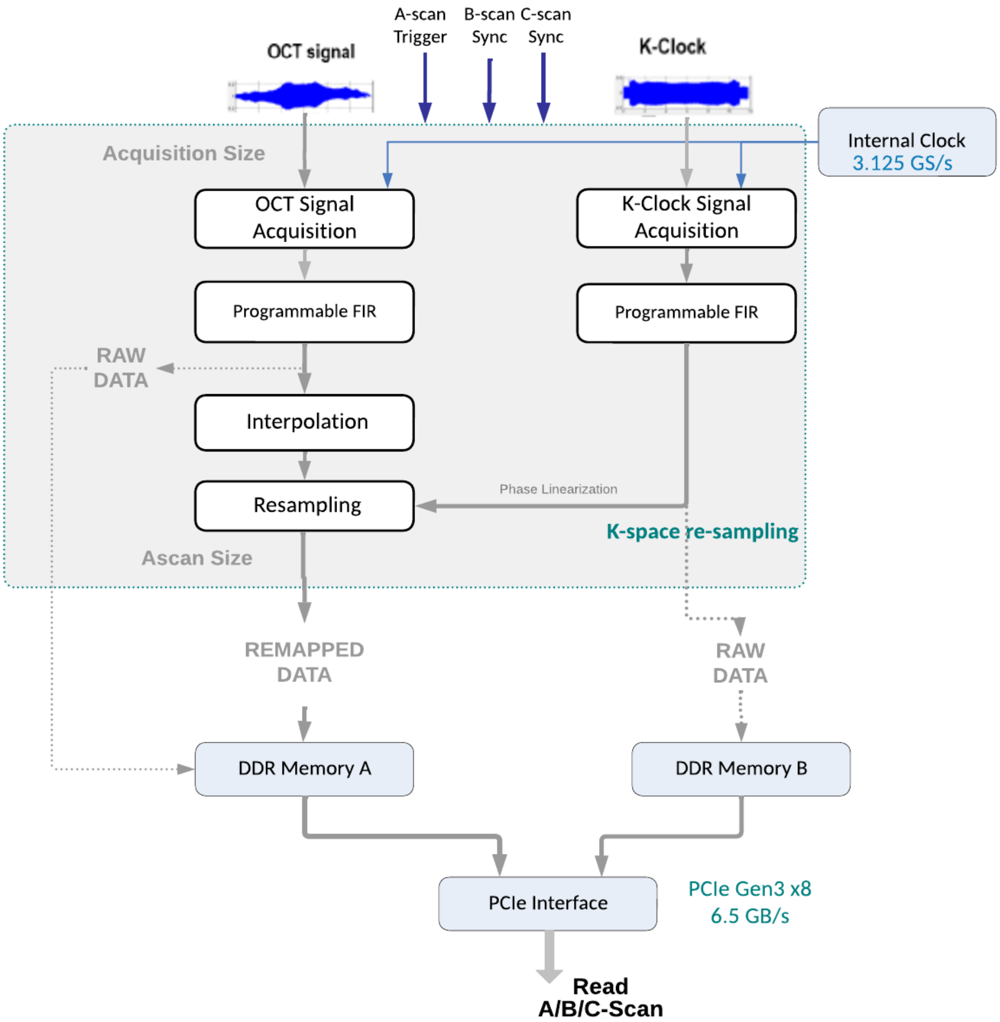
The same descriptor format is used for all data formats. The size of the descriptors is 32 Bytes per A-scan.

Descriptor 1	Descriptor 2	....	Descriptor N

Bit	Content	Size	Comment
7:0	Type	8 bits	0x01 = A-scan descriptor 0x0f = alignment descriptor
12	Forward Sweep	1 bit	If yes forward sweep, if no backward sweep
13	K-clock Over-Range Status	1 bit	If yes, at least 1 A-scan sample is in ADC over-range
14	OCT Over-Range Status	1 bit	If yes, at least 1 A-scan sample is in ADC over-range
31:16	A-scan count	16 bits	Number of A-scans
47:32	B-scan count	16 bits	Number of B-scans
63:48	C-scan count	16 bits	Number of C-scans
127 … 64	Time Stamp (with 1.25 ps resolution)	64 bits	Trigger position. The timestamp wraps-around to 0 after 16.6 days (1.25 ps x 2^60)
159 … 128	Phase Initial	32 bits	Unwrapped phase of the first K-clock sample. To convert in radian, divide by XXX
191 … 160	Phase Max	32 bits	Unwrapped phase of the last K-clock sample minus unwrapped phase of first K-clock sample. To convert in radian, divide by XXX

13.11 Data format

The data can be read in multiple formats, at different stages of the processing chain.



The desired data format can be selected by setting the `AQ4SSOCT_ATTR_IMAGE_DATA_FORMAT` attribute or by using the `Aq4Ssoct_ConfigureImage` function (in C++) or `instr.Image.DataFormat` (in Python).

NOTE Use the `DataSize` parameter returned from the `Aq4Ssoct_FetchImageInt16` (in C++) / `FetchImageInt16` (in Python) function to determine the data size currently used by your application settings.

Raw data format (`AQ4SSOCT_VAL_IMAGE_DATA_FORMAT_RAW` (in C++) / `ImageDataFormat.Raw` (in Python))

The raw data are the data as acquired by the DAQ Card / Module with no processing added to them.

Re-mapped data format (`AQ4SSOCT_VAL_IMAGE_DATA_FORMAT_REMAPPED` (in C++) / `ImageDataFormat.Remapped` (in Python))

The remapped data are the raw data remapped to the K space.

13.12 Simulation mode

A virtual instrument can be simulated by supplying the following option string: "**`Simulate=false, DriverSetup= Model=SA331P`**". All attributes can be set and read. Data can be read on all supported data format.

SS-OCT Solution User Manual for SA331 - 12-bit DAQ Card/Module/Module