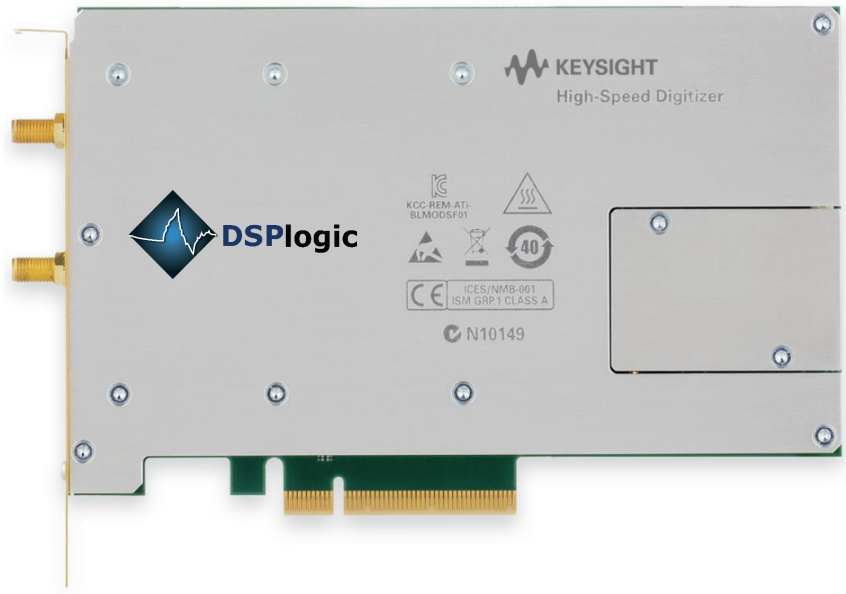
DSPlogic

# *Wideband FFT Spectrometer*

*User Guide*



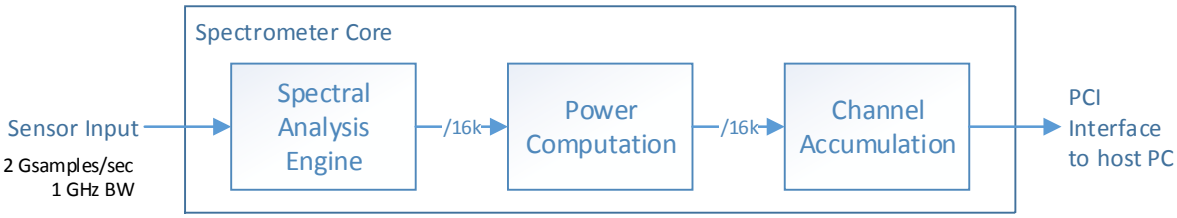

***DSPlogic Inside***

KEYSIGHT TECHNOLOGIES

*DSPlogic Innovation in a Keysight Quality Instrument*

# About Your Instrument

The Wideband FFT Spectrometer utilizes a custom "SpectroCore" hardware processor designed by DSPlogic to extract precise frequency information from the input signal in real-time.




The SpectroCore processor is hosted on the Keysight U5303a Digitizer that provides high quality analog-to-digital conversion and robust, portable drivers that allow an application to communicate with the SpectroCore processor through a high-speed PCI interface.

The U5303A host is configured with the following options:

| Option | Description |
|---|---|
| **DGT** | Digitizer firmware |
| **M02** | 256 MB (64 MSamples/ch) Acquisition memory |
| **FDK** | Custom firmware capability |
| **INT** | 2 GS/s Interleaved channel sampling functionality |
| **SR1** | 1 GS/s Sample rate |
| **F10** | Full Bandwidth |

The spectrometer operates in interleaved mode with an input frequency range of DC to 1.3 GHz (typical). The input frequency range can be limited to 600 MHz using the on-board filter.

# Getting Started

This section describes the steps required to install the host platform and enable the SpectroCore Processor.

## Host Platform Startup

The U5303a host should be installed and verified following the manufacturer's instructions, as described in the "Startup Guide: Keysight U5303A PCIe High-Speed Digitizer," [1] which can be downloaded from

- http://literature.cdn.keysight.com/litweb/pdf/U5303-90001.pdf

These steps include:

1. Unpack and inspect the module
2. Install software, including:
    a. Keysight IO Libraries Suite
        i. www.keysight.com/find/IOsuite
    b. Keysight MD2 High-Speed Digitizer Instrument Drivers
3. Install U5303a in a PC chassis
4. Verify operation of the U5303a

## SpectroCore Installation

The SpectroCore custom real-time hardware processor core is enabled by loading the SpectroCore bit file into the Keysight host platform. The SpectroCore bit file is located in the PySpectro Demonstration Software distribution:

- /pyspectro/bit/U5303ADPULX2FDK_uhsffts_32k_float_1_0_229.bit

In order for the Keysight MD2 drivers to locate the SpectroCore, this file must be copied into one of the following folders, depending on which driver is being used:

- Windows 32-bit: C:\Program Files (x86)\IVI Foundation\IVI\Drivers\AgMD2\Firmware
- Windows 64-bit: C:\Program Files\IVI Foundation\IVI\Drivers\AgMD2\Firmware

# SpectroCore API Reference

Communication with the instrument is done using Keysight's MD2 IVI-COM and IVI-C drivers that work in the most popular development environments including Python, Visual C/C++, C#, VB.NET, MATLAB, and LabVIEW. Please refer to Keysight's "AgMD2 IVI Driver Reference" [2] (installed during Keysight software installation) for detailed information on the driver API.

The MD2 drivers provide memory-mapped access to the SpectroCore configuration/status registers and the on-board DDR memory used to store the accumulated power measurements.

## Enabling the SpectroCore

To connect to the instrument and enable the SpectroCore, use the AGMD2 `IIviDriver.Initialize()` method. When calling this method, the `DriverSetup` option string must contain the `UserDpuA` flag, e.g.:

- `DriverSetup=UserDpuA=<SpectroCore Filename>.bit`

where `<SpectroCore Filename>` is the name of the *.bit file located in the `AgMD2\Firmware` directory.

The `IIviDriver.Initialize()` method returns a driver object that can be used to interact with the instrument using the `Agilent.AgMD2.Interop.AgMD2Ex3` interface.

***Important:*** After connecting to the instrument, the `IAgMD2Acquisition.Mode` property must be se to `AgMD2AcquisitionModeUserFDK (0xF)`. As when changing any instrument settings, the `IAgMD2Acquisition.ApplySetup()` method must be called prior to initiating an acquisition.

## Communicating with the SpectroCore

The SpectroCore API provides two interfaces: Measurement Control (SMCI) and Measurement Data (SMDI).

The Measurement Control and Measurement Data interfaces are accessed through the `AgMD2Ex3` instrument interfaces, including the following:

- `IAgMD2Acquisition`
- `IAgMD2AcquisitionUserControl`
- `IAgMD2Calibration`
- `IAgMD2Channels`
- `IAgMD2LogicDevices`

Details of the Measurement Control and Measurement Data interfaces are provided in the following sections. The PySpectro software also demonstrates a complete software implementation of the SpectroCore processor interface.

## SpectroCore Measurement Control Interface

The SpectroCore Measurement Control Interface (SMCI) consists of a set of control and status registers located on the "DPUA" logic device found on the host platform. The IAgMD2LogicDevices interface must be queried to obtain the IAgMD2LogicDevice object corresponding to the DPUA device. This object is used to access the SpectroCore registers using the following methods:

- IAgMD2LogicDevice.ReadRegisterInt32()
- IAgMD2LogicDevice.WriteRegisterInt32()

In addition, the SMCI uses the IAgMD2Calibration and IAgMD2AcquisitionUserControl interfaces.

The following table identifies the registers that compose the SMCI.

**Table 1 SpectroCore Measurement Control Interface – Memory map**

| Register Name | Register ID | Type | Offset |
|---|---|---|---|
| Main control | main_control | R/W | 0x3300 |
| Main status | main_status | R | 0x3304 |
| Measurement window | num_average | R/W | 0x3308 |
| Measurement count | msrmnt_count | R | 0x330C |

### Instrument Calibration

Prior to acquiring any measurements, the instrument must be calibrated in accordance with manufacturer specifications using the SelfCalibrate() method of the AgMD2Calibration interface.

### Measurement Acquisition

A measurement acquisition is initiated using the StartProcessing() method of the IAgMD2AcquisitionUserControl interface. The ProcessingType flag must be set to 0x1. Other processing types are ignored by the SpectroCore processor.

Two types of acquisitions are supported: one-shot measurement and continuous measurement. The type of measurement that is initiated depends on the continuous_mode field in the Main Control Register.

Acquisition status is provided by the Measurement Count register. The measurement count equals zero after the beginning of each acquisition, and is incremented at the completion of each measurement.

The `IAgMD2AcquisitionUserControl.StopProcessing()` method must be called to stop an acquisition. It must be called with the same `ProcessingType` flag (0x1) as the start command. In single-shot mode, the `StopProcessing()` method must be called after the single measurement is complete prior to starting a new acquisition. In continuous mode, the `StopProcessing()` method can be called at any time to stop the SpectroCore from writing new measurements to acquisition memory.

## Main Control Register

The main control register is composed of the fields shown in Table 2. This register should not be modified during an acquisition.

**Table 2 SMCI: Main control register fields**

| Field | Data type | # Bits | Bit range | Description |
|---|---|---|---|---|
| continuous_mode | Bool | 1 | [0:0] | 0 = Single-shot mode<br>1 = Continuous mode |
| disable_polyphase | Bool | 1 | [1:1] | 0 = Polyphase filter algorithm enabled<br>1 = Polyphase filter algorithm disabled |
| reset | Bool | 1 | [31:31] | 1 = Reset SpectroCore<br>0 = Remove SpectroCore from reset |

## Main Status Register

The Main status register is composed of the fields shown in Table 3. The occurrence of a memory indicates that both software and the PySpectro core tried to access acquisition memory at the same time. (See Section “Memory Sharing” for additional information.)

**Table 3 SMCI: Main status register fields**

| Field | Data type | # Bits | Bit range | Description |
|---|---|---|---|---|
| memory_error | Unsigned | 8 | [7:0] | 0 = No measurement memory error<br>1 = Measurement memory error occurred |

## Measurement Window Register

The measurement window register is composed of the fields shown in Table 4. This register determines the number of 32-kFFT results to accumulate in the SpectroCore.

The length of the measurement window, and the resulting measurement update period is

*Tmsr (seconds) = (num_averages_m1 + 1) * 32768 / 2.0e9.*

**Table 4 SMCI: Measurement window register fields**

| Field | Data type | # Bits | Bit range | Description |
|---|---|---|---|---|
| num_averages_m1 | Unsigned | 32 | [31:0] | Number of FFTs to accumulate minus one. |

## Measurement Count Register

The measurement count register indicates the number of measurements that have occurred since the last `StartProcessing` command. This register can be read at any time during a continuous-mode acquisition.

**Table 5 SMCI: Measurement Count Registers**

| Field | Data type | # Bits | Bit range | Description |
|---|---|---|---|---|
| msrmnt_count | Unsigned | 32 | [31:0] | Number of measurements that have occurred since the last StartProcessing command. |

# SpectroCore Measurement Data Interface

The SpectroCore Measurement Data Interface (SMDI) provides access to measurement data stored in memory onboard the host U5303a. This section describes how read measurement data from the SpectroCore processor. The PySpectro demonstration software also demonstrates usage of this interface.

## Memory Sharing

On the U5303a, access to acquisition memory is time-shared between the SpectroCore processor and software access through PCI, as shown in Figure 1. The SpectroCore requires exclusive access to memory for approximately one FFT window (16.4 us) during each measurement accumulation window. In continuous mode, the SpectroCore processor performs this write access at regular intervals of N*16.4 microseconds, where N is the number of FFTs accumulated, resulting in a small duty cycle for any significant number of accumulations. The SpectroCore is capable of internally accumulating millions of FFTs, minimizing interface throughput requirements.

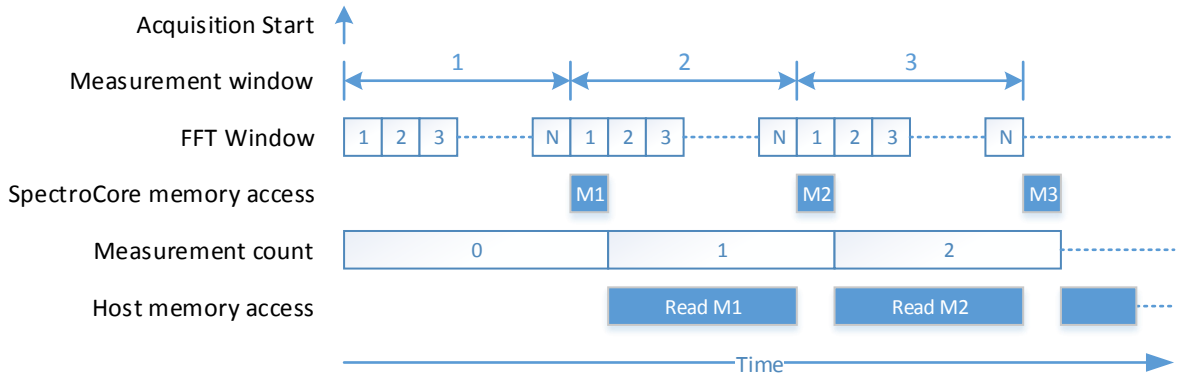



**Figure 1: Acquisition memory - shared access timing.**

In continuous mode, time sharing can performed by software polling of the Measurement Count register. Whenever a new measurement is detected, software should take control of acquisition memory, read the measurement results, then return control to the SpectroCore processor prior to the next SpectroCore write interval. If the SpectroCore tries to access memory and is blocked by software, a `memory_error` will be raised in the main status register.

In continuous mode, the minimum number of accumulations achievable without memory access conflict depends on the capabilities of the user software, the OS under which it runs, and the performance of the PCI interface.

There are two memory banks connected to the DPUA `IAgMD2LogicDevice`, identified as "DDR3A" and "DDR3B". To access DDR memory controls, the use DPUA's `IAgMD2LogicDeviceMemoryBanks` interface to obtain a `IAgMD2LogicDeviceMemoryBank` reference for each of the "DDR3A" and "DDR3B" memory banks.

To arbitrate memory access, software must set the `AccessMode` property of the `IAgMD2LogicDeviceMemoryBank` interface in accordance with the following table. DDR3A and DDR3B are arbitrated independently.

**Table 6 SMDI: DDR Memory IAgMD2LogicDeviceMemoryBank.AccessMode property**

| Field | Data type | # Bits | Bit range | Description |
|---|---|---|---|---|
| AccessMode | Unsigned | 32 | [31:0] | 0x00000000: SpectroCore has access to memory. Software should not access memory over PCI.<br>0x00000001: Software has access to memory over PCI. SpectroCore cannot write measurements to memory. |

## Reading Measurement Data

When the measurement count indicates that a new measurement is available, and software has taken control of acquisition memory (DDR3A or DDR3B), the measurement can be read using the `ReadIndirectInt32()` method of the `IAgMD2LogicDevice` interface for DPUA.

In order to perform spectral measurements efficiently, the SpectroCore processor writes the measurement results to both acquisition memory banks (DDR3A and DDR3B) in the order in which they are computed. Therefore, after reading from DDR3A and DDR3B, the data must be rearranged to put the spectral measurements in normal frequency order.

A total of 16,384 channels (FFT bins) are available for each measurement. Half are stored in DDR3A and other half in DDR3B. To read all channels, two calls to the `ReadIndirectInt32()` method are required, with arguments set as follows:

- `Id (Integer) = BUFFER_ID (from Table 7)`
- `StartAddress (Integer) = 0`
- `NumElements (Long) = 8192`

**Table 7 SMDI: BUFFER_ID for DDR Access using ReadIndirectInt32()**

| DDR Bank | BUFFER_ID |
|---|---|
| DDRA | 28 |
| DDRB | 30 |

The measurements are stored in acquisition memory using single-precision floating point format. Although the result of the ReadIndirectInt32() method vector of 32-bit integers, the contents must be re-interpreted (without changing the stored value) as 32-bit floating-point.

The mapping to rearrange memory contents to FFT bins is fixed and should be mapped to a table to optimize performance. Please refer to the PySpectro software documentation for details on the memory to FFT bin mapping.

# PySpectro Demonstration Software

The PySpectro Demonstration Software illustrates the use of the SpectroCore processor as hosted on the Keysight U5303a digitizer. PySpectro consists of the following three components:

- Spectrometer Driver
- PySpectro Core Application
- PySpectro GUI

The Spectrometer Driver is a low-level implementation of the SMCI and SMDI interfaces described in this document. The PySpectro Core is a high performance, multi-threaded spectral measurement acquisition and logging application. It can be used programmatically, or can be controlled through the PySpectro Graphical User Interface.

PySpectro is written in Python with well documented source-code. It has been tested on Windows 32 and 64 bit platforms, but should also work on other platforms supported by the Anaconda Python distribution.

## PySpectro Core Application

The PySpectro Core application is a high performance, multi-threaded spectral measurement acquisition and logging application that interfaces with the Wideband FFT Spectrometer.

The PySpectro Core (`pyspectro.applib.core.PySpectroCore`) class provides non-blocking interfaces to manage instrument connections, control acquisition and log data. An event interface can be used to monitor data transfers and changes in instrument state. User callbacks are also supported for data processing, state change handling and 1 PPS heartbeat processing. Supporting classes for the PySpectro Core include:

- Instrument connection management
  - `pyspectro.applib.connection.ConnectionManager`
- Acquisition control
  - `pyspectro.applib.acq_control.AcquisitionControlInterface`
- Data logging
  - `pyspectro.applib.datalogger.SpectrumDataLogger`

The PySpectro Core can be accessed directly from Python code or may be controlled by the GUI.

## PySpectro GUI

The PySpectro GUI provides a convenient way to get started collecting measurements from the Wideband Spectrometer. It uses the PySpectro Core to interact with the instrument and provides the following features:

- Connection management
- Spectral Display
  - Power vs. Frequency display
  - Zoom, pan, and select data points during continuous acquisition
  - Highlight frequency bin using mouse to display bin power
  - Auto-scale
  - Units in dBm, dB-FS, or FS (raw)
- Acquisition Control
  - Start/Stop Acquisition
  - Single-shot or continuous mode
  - Acquisition window duration setting (in seconds or number of FFT averages)
  - Algorithm selection (FFT or Polyphase Filter Bank)
- Input Settings
  - Full scale input range (FSR) adjustment
  - Input voltage offset adjustment
  - Enable bandwidth limiting filter (600 MHz)
- HDF5 Measurement Data Logging
  - Stores measurement data in HDF5 format
  - Supports both one-shot and continuous acquisitions.
  - Continuously log measurements during continuous acquisitions (Limited to first N measurements of each acquisition).
- Calibration
  - Initiate instrument calibration
  - Calibration required notification
- Acquisition monitoring
  - Board and ADC temperatures
  - Number of acquisitions (total for instrument session)
  - Number of measurements (current acquisition)
  - Dropped measurement detection

An example screenshot of the PySpectro GUI is shown in Figure 2. The spectral display shows a measurement of a 20 MHz sine wave input using the Polyphase Filter Bank (PFB) algorithm. Figure 3 shows the measurement using the FFT algorithm, demonstrating the superiority of the PFB.

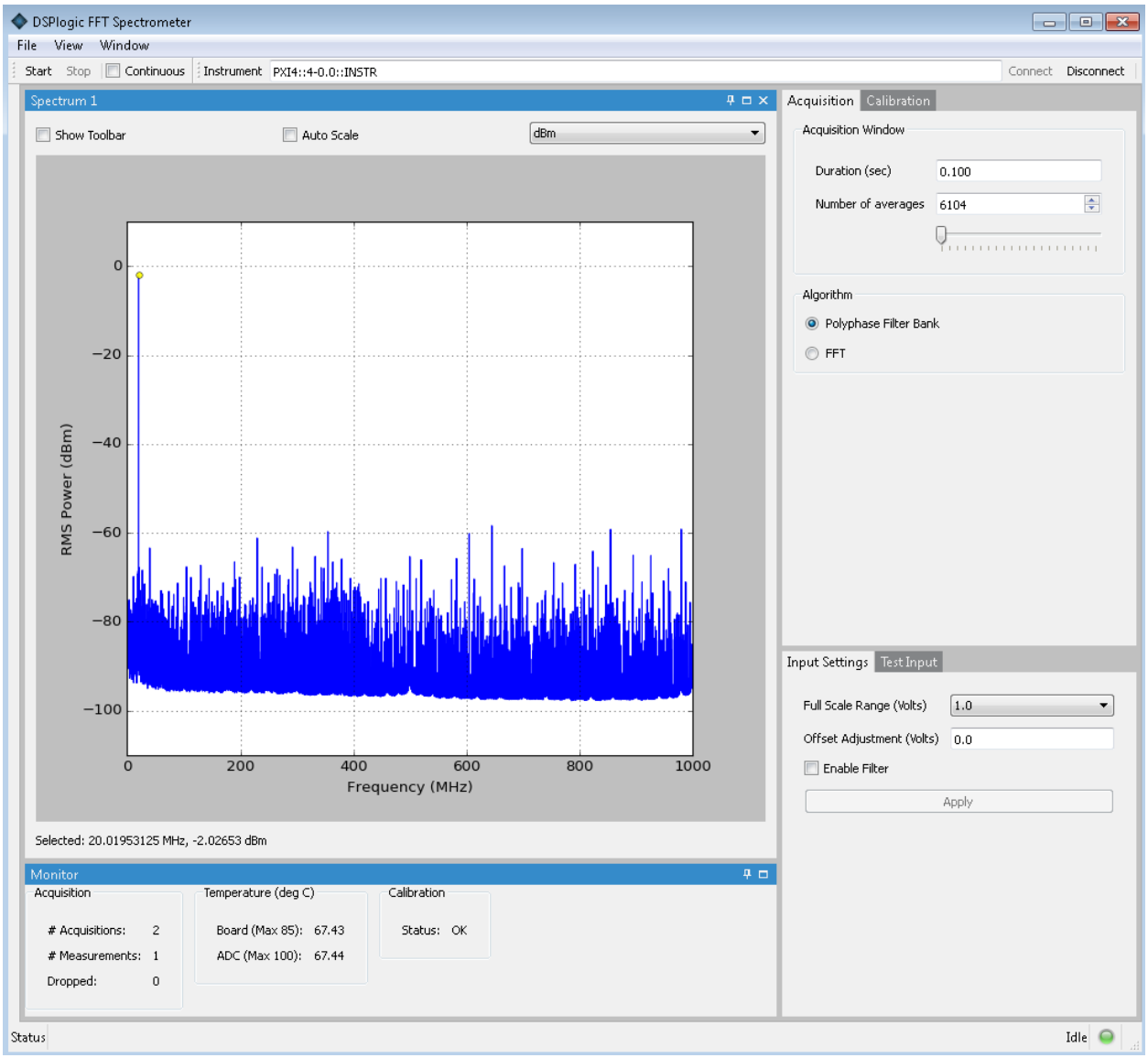



**Figure 2 PySpectro GUI**

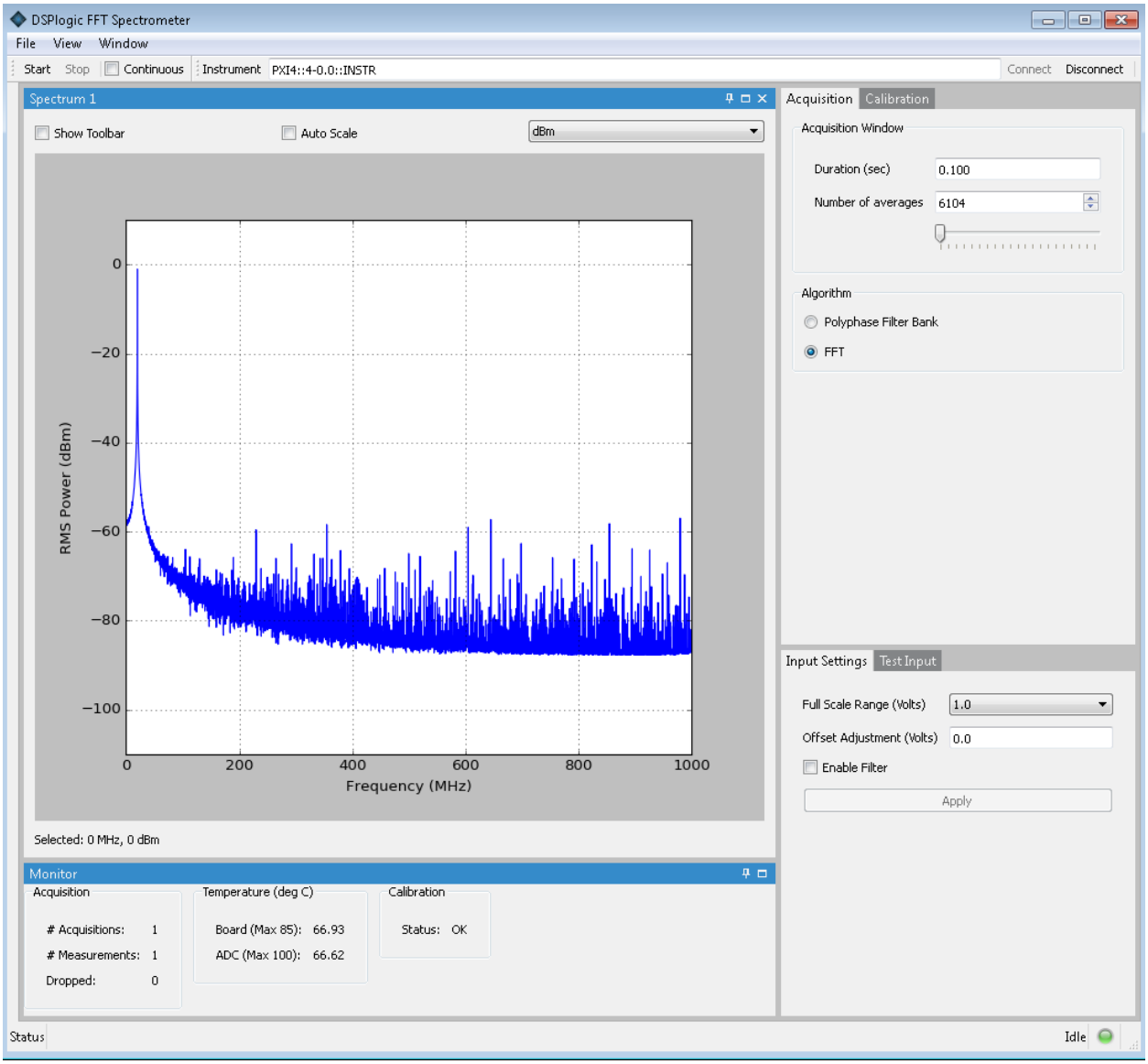



**Figure 3 Pyspectro GUI (FFT Algorithm)**

## PySpectro Installation

PySpectro requires a Python 2.7 interpreter to be installed on the host computer. In addition to the interpreter, the Python packages listed in Table 8 are required.

**Table 8 – PySpectro software requirements**

| Package | Version |
|---|---|
| **python** | 2.7.11 |
| **numpy** | 1.11.0 |
| **matplotlib** | 1.5.1 |
| **comtypes** | 1.1.2 |
| **enaml** | 0.9.8 |
| **h5py** | 2.6.0 |

The PySpectro source code is packaged and can be installed using standard Python distribution utilities. However, to simplify the process of obtaining the prerequisite packages, we highly recommend using the Anaconda distribution of Python, which can be downloaded and installed from:

- http://repo.continuum.io/archive/Anaconda2-4.0.0-Windows-x86.exe

### *Create a Conda Environment*

After installing the Anaconda distribution, create a "Conda Environment" in which to execute the PySpectro software. Each conda environment allows you to have a different version of Python and any packages installed in them. This allows PySpectro to be installed and used with the prerequisite packages without affecting any other python installations on the host computer.

First, create a conda environment called "pyspectro", which will be used to run the PySpectro software. From the command line, enter:

- `conda create -n pyspectro python=2.7.11 matplotlib numpy comtypes enaml h5py`

This creates a standalone interpreter instance with all of the required packages. To enable the conda environment, it must be "activated". From the command line, enter:

- `activate pyspectro`

To verify that the required software is installed, use the "conda list" command to display a list of the python packages installed in the environment. The output should be similar to the results below:

```
[pyspectro] C:\>conda  list
```

```
# packages in environment at C:\Anaconda\envs\pyspectro_env:
#
atom                      0.3.10                   py27_0
comtypes                  1.1.2                    py27_0
cycler                    0.10.0                   py27_0
enaml                     0.9.8                    py27_1
h5py                      2.6.0               np111py27_1
hdf5                      1.8.16                    vc9_0
jpeg                      8d                        vc9_0
kiwisolver                0.1.3                    py27_0
libpng                    1.6.17                    vc9_1
libtiff                   4.0.6                     vc9_1
matplotlib                1.5.1               np111py27_0
mkl                       11.3.3                       1
numpy                     1.11.0                   py27_1
openssl                   1.0.2h                    vc9_0
pip                       8.1.1                    py27_1
ply                       3.8                      py27_0
pyparsing                 2.1.1                    py27_0
pyqt                      4.11.4                   py27_5
python                    2.7.11                        4
python-dateutil           2.5.3                    py27_0
pytz                      2016.4                   py27_0
qt                        4.8.7                     vc9_7
setuptools                20.7.0                   py27_0
sip                       4.16.9                   py27_2
six                       1.10.0                   py27_0
tk                        8.5.18                    vc9_0
vs2008_runtime            9.00.30729.1                 0
wheel                     0.29.0                   py27_0
zlib                      1.2.8                     vc9_2
```

### *Install PySpectro Software (optionally in 'develop' mode)*

Next, PySpectro should be installed into the `pyspectro` environment. While the environment is activated, change to the folder containing the PySpectro source (including the `setup.py` file). To install, enter the command:

```
[pyspectro] C:\<download>\pyspectro> python setup.py install
```

where <*download*> is the path to the PySpectro source tree.

Alternately, PySpectro can be installed in development mode, allowing you to modify and debug the source code within a development workspace.

```
[pyspectro] C:\<workspace>\pyspectro> python setup.py develop
```

where <*workspace*> is the path where PySpectro has been downloaded or checked out from GitHub.

### *Running the PySpectro software*

To execute the PySpectro software, activate the `pyspectro` environment, then run the `pyspectro` command:

```
C:\ > activate pyspectro
[pyspectro] C:\ > pyspectro
```

A windows batch file/shortcut is included that can be used to run PySpectro in a single step.

# References

1. Keysight Technologies, Inc., “Startup Guide: Keysight U5303A PCIe High-Speed Digitizer,” http://literature.cdn.keysight.com/litweb/pdf/U5303-90001.pdf

2. Keysight Technologies, Inc., “AgMD2 IVI Driver Reference,” Compiled help file, installed with AgMD2 drivers.