SPIDER

VIS-SWIR Photodetector

USER TECHNICAL MANUAL

NIREOS SRL

Via Giovanni Durando, 39 - 20158 Milan (Italy)

info@nireos.com | www.nireos.com

Subject to change without notice

Document Version: 1.4

07/03/2023

Contents

1. Introduction

1.1 Features

- Si-InGaAs sandwich photodetector
- VIS+SWIR (320nm to 1700nm)
- 8 programmable gains (1.5kΩ to 4.4MΩ)
- analog bandwidth up to 56kHz
- 2 BNCs for analog output
- embedded 2ch - 24bit ADC up to 120kSPS/ch
- USB connection (transmission + settings)
- 2 buttons for gain settings
- OLED display
- 3 GPIOs (digital input-output pins)
- DLL + demo GUI available
- LabVIEW software development kit (SDK)

1.2 Brief Description

The SPIDER (Single-PIxel DEtectoR) is an amplified Si-InGaAs photodetector with programmable gain and embedded 24bit data acquisition system. The two-color detector consists in a Si photodiode mounted over an InGaAs photodiode, along the same optical axis, for a wide spectral response ranging from 320nm to 1700nm.

The Si and InGaAs photodiodes are amplified simultaneously with two independent low-noise amplifiers having 8 programmable gains each and up to 56kHz of analog bandwidth.

SPIDER has been designed to work either standalone or connected to a PC. In the first case, the amplified analog signals coming from the Si and InGaAs channels can be directly read with two BNC connectors. The gain for both channels can be easily selected by pressing a button and reading the corresponding settings on the built-in OLED display.

In the second case, when SPIDER is connected to a computer via USB, it can be fully controlled by a customizable user interface based on a user-friendly DLL. Thanks to the embedded 24bit data acquisition system, the Si and InGaAs analog outputs are simultaneously digitalized up to 120KSPS - 24bit and sent to the computer to be visualized in real time.

SPIDER also includes 3 GPIO pins (general purpose input output) which can be used to generate or read digital signals. This allows the SPIDER to control simple external device (e.g. shutter open/closed) and read a start/stop acquisition signal (e.g. external trigger).

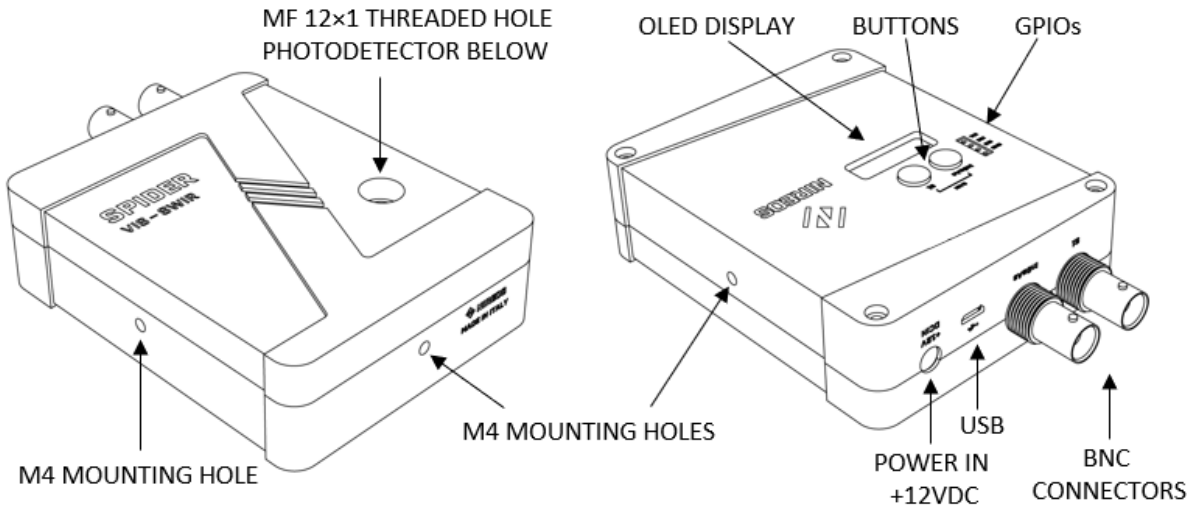


Figure 1: SPIDER controls and main parts.

1.3 Information, Warning and Safety Instructions

	ESD Warning
	The system contains electrostatic sensitive devices. To prevent equipment damage, use proper grounding techniques and observe precautions for handling.

2. Theory of operation

Install the SPIDER using a metric optical post and a post holder into one of the M4 mounting holes located on the side walls.

The SPIDER must be powered up with a stable 12VDC power supply into the power connector DCIN. Plug the power supply into a 50 to 60Hz, 100-240Vac power outlet. Power consumption is around 3.4W during normal use.

To read the Si-InGaAs analog outputs: connect a measuring device such as a DAQ system or oscilloscope by attaching a coaxial cable to one or both BNC connectors. Warning: input impedance of the measuring device must be at least 100kΩ. The SPIDER does not work with a 50Ω load to avoid too much current when output voltage reaches 10V. The bandwidth at the BNC outputs is equal to the analog bandwidth and it depends on the gain setting, see Table 2.

To read the Si-InGaAs digital outputs: connect a USB cable to the SPIDER micro-usb connector and the remaining end to a computer. Launch the GUI and open a serial communication to control settings and read the data stream. The analog outputs are digitalized at 24bit up to 120kSPS for each channel. In this case, the equivalent bandwidth is the minimum between the analog bandwidth (see Table 2) and the digital bandwidth (see Table 3). Gain settings, sampling rate (from 0.5 to 120kSPS) and GPIOs status can be programmed from the GUI by using the specific functions on the DLL. When connected to the computer, the 3 GPIOs (IO1, IO2 and IO3) can be used to read and write digital signals. This allows the SPIDER to control an external device (e.g. shutter open/closed) and read a digital trigger (e.g. start/stop acquisition).

The Si and InGaAs gains can also be changed at any time, even when SPIDER is connected to the computer, by pressing the corresponding button. There are 8 gain settings incremented in 10dB steps. The selected gain and bandwidth for each channel (Si and InGaAs) will be shown onto the OLED display. Press one of the two buttons once to turn on the display, the gain will not change, then press again the button while the display is on to change the gain. If no button is pressed, the display automatically turns off after a few seconds. Keep the Si button pressed for at least 5 seconds to flip the OLED text orientation. Gain settings are not lost when the device is turned off, instead the last gain settings are saved to be restored as soon as SPIDER is powered on again. Sampling rate and ADC filter, however, are re-initialized every time SPIDER is powered on, to their default values (80 kSps and wideband, respectively).

Install the fiber adapter to the input aperture (optional). Apply a light source and adjust the gain to the desired setting so that the measured output signal is below 10 V to avoid saturation.

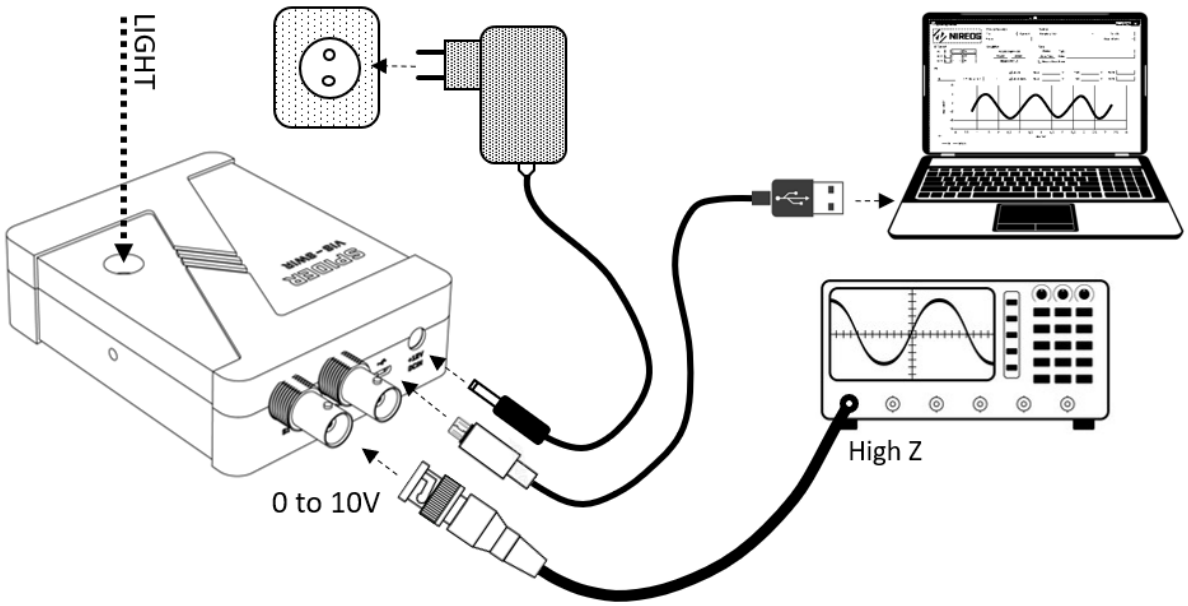


Figure 2: Connection diagram.

3. Hardware

The SPIDER is based on a Si-InGaAs photodetector amplified by two independent transimpedance amplifiers (see Figure 3: Simplified internal block diagram.) having a gain equal to Z_{Si} and Z_{InGaAs}, respectively, please refer to Table 2 for the gain/bandwidth configurations. The amplifiers output V_{Si} and V_{InGaAs} (in the range 0 to 10V) are buffered out to their respective BNC connectors (BNC Si and BNC InGaAs) to be read by an external device (e.g. external DAQ or oscilloscope) having a high input impedance (>100kΩ). The SPIDER does not work with a 50Ω load on the BNC connectors. If a computer is connected via USB to the SPIDER, the amplifier output signals can also be digitalized by two separate 24bit ADCs having a configurable sampling rate up to 120kSPS, please refer to Table 3 for the ADC characteristics. When the analog outputs are digitalized, the equivalent bandwidth is the minimum between the analog bandwidth (see Table 2) and the digital bandwidth (see Table 3). The data stream out of the ADCs are finally packed by a DSP and sent in real time to the computer with a serial communication over the USB. Here the output signals can be visualized and analyzed by the graphical user interface (GUI), see Section 5.

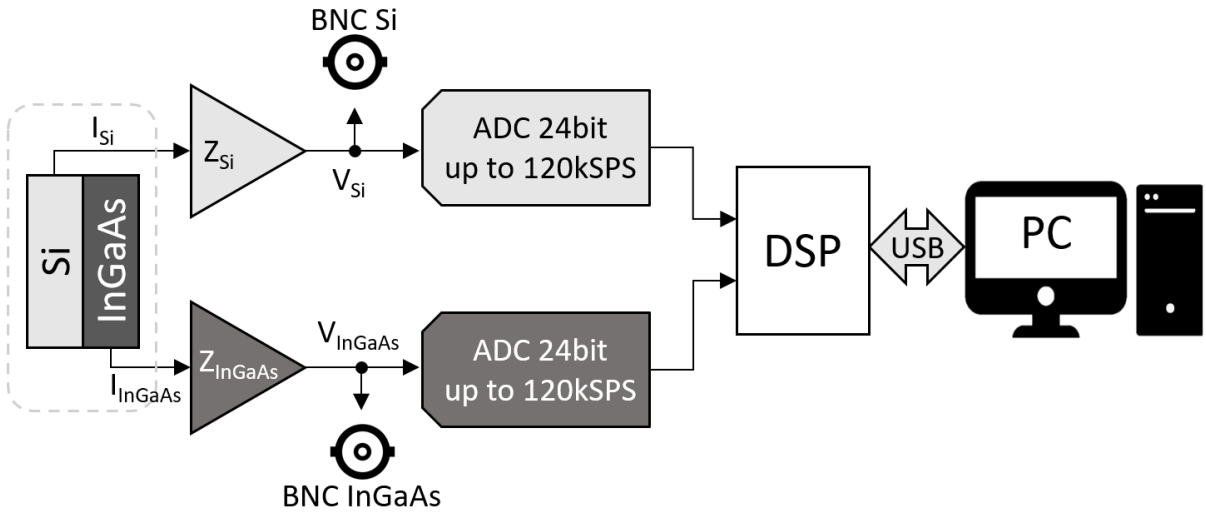


Figure 3: Simplified internal block diagram.

3.1 Si-InGaAs photodiode

The SPIDER is based on a two-color photodetector consisting in a Si photodiode mounted over an InGaAs photodiode, along the same optical axis. This combination results in a spectral response ranging from visible up to SWIR (320nm to 1700nm). The responsivity as a function of the wavelength for each photodiode is shown in Figure 4: Si-InGaAs photodiode responsivity.

It is important to note that the photodiode active areas do not have the same dimensions: the Si photodiode is square with dimension 2.4×2.4 mm, while the InGaAs photodiode is circular, with diameter 1 mm.

Electrical and optical characteristics of the Si-InGaAs photodetector (Typ. Ta=25°C)					
Detector element (Photosensitive area)	Spectral response range [µm]	Peak sensitivity wavelength λ_p [µm]	Responsivity R $\lambda=\lambda_p$ [A/W]	Dark Current I_D	Detectivity D* $\lambda=\lambda_p$ [cm*Hz$^{1/2}$/W]
Si (2.4×2.4 mm)	0.32 to 1.08	0.94	0.45	130pA	1.4 x 10^{13}
InGaAs (Ø1 mm)	1.01 to 1.7	1.55	0.55	1nA	3.5 x 10^{12}

Table 1

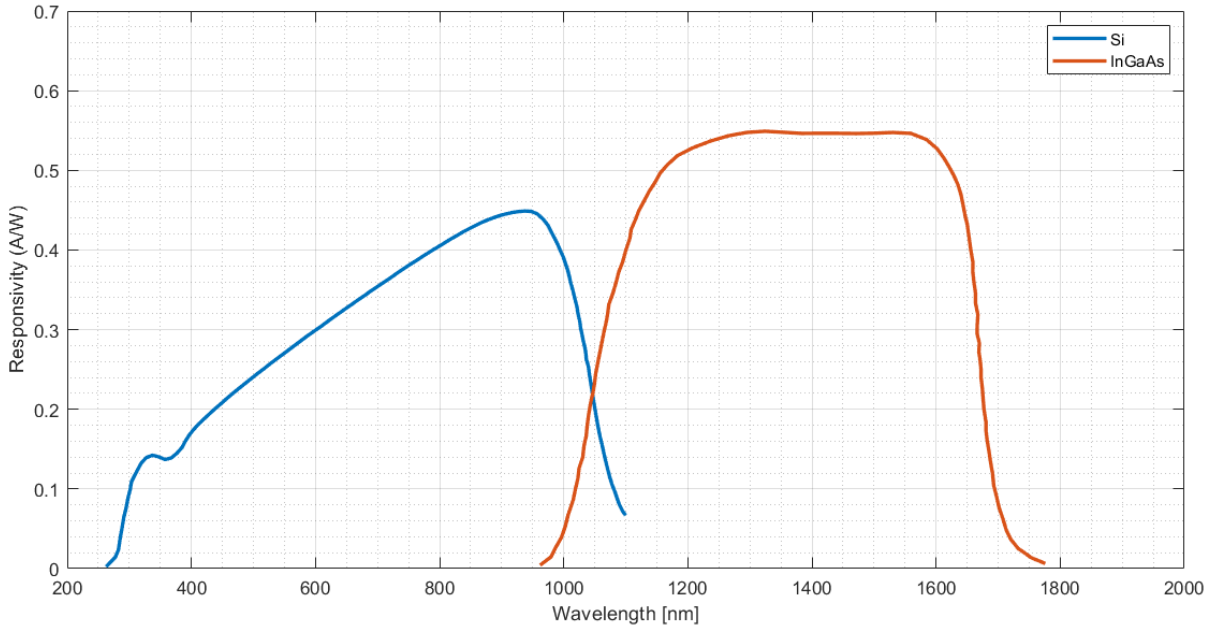


Figure 4: Si-InGaAs photodiode responsivity.

3.2 Transimpedance Amplifier

The photogenerated current signals I_{Si} and I_{InGaAs} are amplified simultaneously by two independent low-noise amplifiers having 8 programmable gains Z_{Si} and Z_{InGaAs}, respectively, ranging from 1.5kΩ up to 4.4MΩ. Because of the resistive gain, input current signals become the voltage output signals V_{Si} and V_{InGaAs} through the following relation.

$$V_{Si,InGaAs} = I_{Si,InGaAs} \cdot Z_{Si,InGaAs} \qquad \text{E3.1}$$

This is valid when the input frequency is lower than the transimpedance bandwidth. Table 2 reports the main electrical characteristics for each gain setting. BW=bandwidth.

Electrical characteristics of the transimpedance amplifier (Typ. Ta=25°C)					
Gain #	Transimpedance Gain ($Z_{Si/InGaAs}$) [Ω]		Analog BW (-3dB) [kHz]	Output Noise* [nV/√Hz]	NEP** [pW_{RMS}]
0	1580	$64dB_{\Omega}$	56	20	7509
1	4590	$73dB_{\Omega}$	55	22	2818
2	14400	$83dB_{\Omega}$	54	26	1052
3	44800	$93dB_{\Omega}$	53	36	463.7
4	142000	$103dB_{\Omega}$	52	61	245.5
5	446000	$113dB_{\Omega}$	49	121	150.5
6	1420000	$123dB_{\Omega}$	39	265	98.4
7	4440000	$133dB_{\Omega}$	29	680	65.4

Table 2

*Measured at 20kHz

**assuming Responsivity=0.5A/W

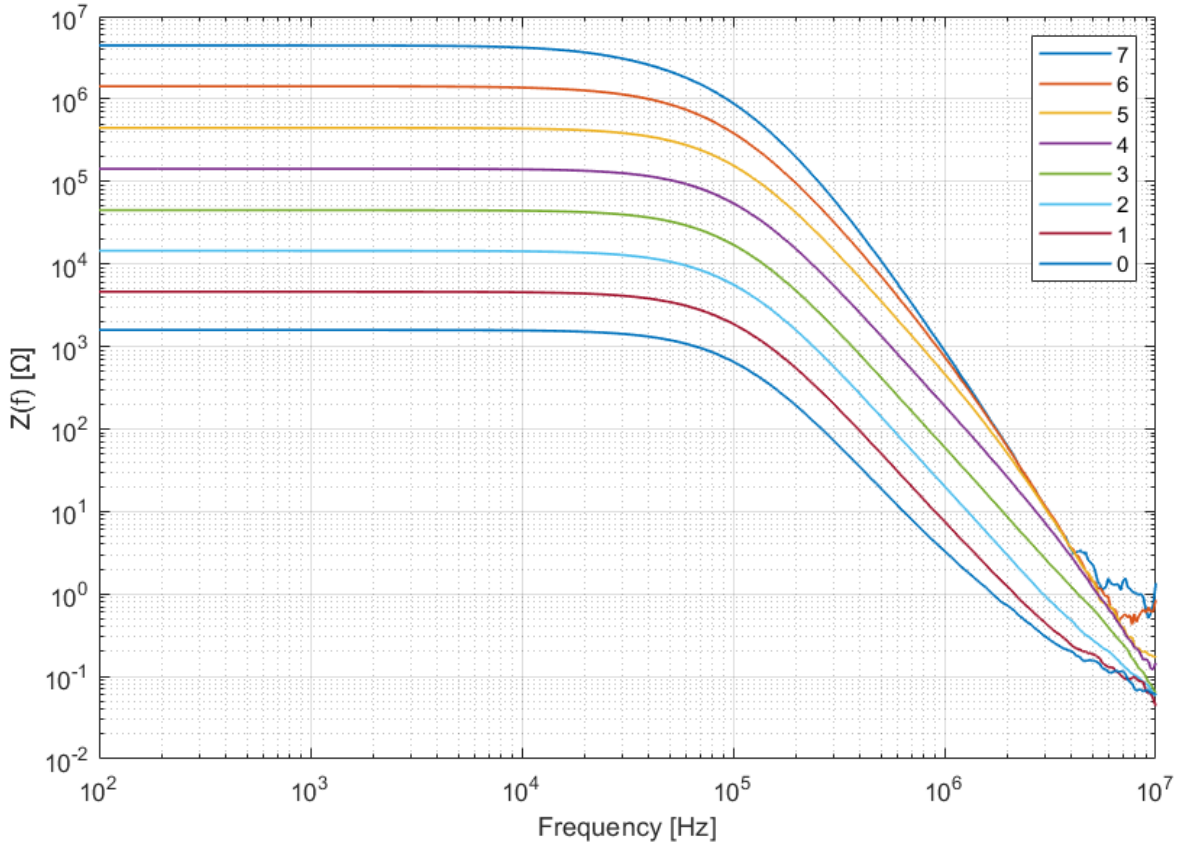


Figure 5: Transimpedance amplifier transfer functions.

The light input power P_{in} at a given wavelength λ is converted to an output voltage $V_{Si,InGaAs}$ through the following relation:

$$V_{Si,InGaAs} = P_{in}(\lambda) \cdot R(\lambda) \cdot Z_{Si,InGaAs} \qquad \text{E3.2}$$

where R(λ) is the photodetector responsivity at a given wavelength and $Z_{Si,InGaAs}$ is the selected transimpedance amplifier gain.

3.3 Analog to Digital conversion

Thanks to the embedded 24bit data acquisition system, the Si and InGaAs analog outputs are simultaneously digitalized up to 120kSPS - 24bit and sent to the computer to be visualized in real time. The analog signals V_{Si} and V_{InGaAs} are digitalized and transferred in groups of 256 samples each. Therefore, in this context, it can be defined a "packet" as follow:

$$\text{Packet=settings + 256 Si samples + 256 InGaAs samples} \qquad \text{E3.3}$$

where "settings" represents the SPIDER settings information: GPIO Status and Direction, Si and InGaAs gains. The communication between the SPIDER and the PC consists in the transmission of one or more packets.

The time interval between two consecutive samples is equal to 1/SamplingRate. In case of continuous acquisition, total acquisition time for the single packet is equal to:

$$t_{packet} = \frac{256}{SamplingRate} \qquad \text{E3.4}$$

Since the internal analog to digital converter has an input range equal to -8.192V to +8.192V, the 24bits gives a resolution of around 1µV. Because of the offset, acquired output voltage in dark conditions might result in a small negative value (Typ. -0.04V).

Table 3 reports the main electrical characteristics of the ADC as a function of the Sampling Rate (SR). BW=bandwidth.

Electrical characteristics of the Analog to Digital converter (Typ. Ta=25°C)							
SR index	Sampling Rate (SR) [kSPS]	Digital BW [kHz]		SNR* [dB]	RMS Noise* [V]	BITs	ENOB*
		Wideband 0.433xSR	*Sinc Filter 0.2617xSR*				
10	120	51.96	31.4	113,0	1.4E-05	24	19.2
9	100	43.3	26.17	113,8	1.3E-05	24	19.3
8	80	34.64	20.94	114,8	1.2E-05	24	19.4
7	64	27.712	16.75	115,7	1.0E-05	24	19.6
6	32	13.856	8.37	118,8	7.2E-06	24	20.1
5	16	6.928	4.19	121,8	5.3E-06	24	20.6
4	10	4.330	2.617	123,8	4.2E-06	24	20,9
3	5	2.165	1.308	126,8	3.2E-06	24	21.3
2	2,5	1.083	0.654	129,8	2.5E-06	24	21.6
1	1	0.433	0.2617	133,8	2.0E-06	24	22.0
0	0,5	0.217	0.131	136,8	1.8E-06	24	22,2

*Table 3. *gain # = 0 and wideband filter selected*

Sampling Rate (SR) [kSPS]	RMS Noise [V] (Wideband Filter)			
	Si		InGaAs	
	1.5k	4.7M	1.5k	4.7M
120	9.7E-06	3.6E-4	9.8E-06	9.7e-4
80	8.8E-06	3.3E-4	8.9E-06	9.6e-4
32	5.6E-06	2.7E-4	5.7E-06	9.1e-4
10	3.4E-06	1.2E-4	3.4E-06	6.5e-4
2,5	2.1E-06	2.6E-5	2.1E-06	9.8E-5
0,5	1.6E-06	4.6E-6	1.6E-06	8.9E-6

Table 4

3.4 ADC Digital Filter

Like most ADCs, the data acquisition system includes a low-pass filter to prevent aliasing artifacts from signals with frequency components higher than the Nyquist frequency (i.e. 0.5×SR, where SR is sampling rate). In SPIDER, there are two types of digital filter, selectable in software: Wideband and Sinc. The Wideband filter offers a larger bandwidth, a low ripple pass band, narrow transition band, and high stop band rejection. The filter response is similar to an ideal brick wall filter, making it ideal for frequency domain measurements. The Sinc filter has a response close to a “sinc” function and it is characterized by a low latency signal path that makes it ideal for time domain analysis and DC measurements.

In other words, the wideband filter can be used to measure higher frequency signals, but has a longer settling time than the sinc filter, which can respond more cleanly to sudden changes. For example: when measuring a square pulse with the wideband filter, oscillatory artifacts can be found near the edges of the function, and the sinc filter would be the correct choice. On the other hand, when measuring a sinusoidal signal with frequency approaching 0.5×SR, the measured signal will be better suited (i.e. less attenuated) by the wideband filter.

Filter type	-3 dB point	Settling time
Wideband	0.433 × SR	3.5/SR (i.e. 3.5 samples)
Sinc	0.217 × SR	79.6/SR (i.e. 79.6 samples)

Table 5

3.5 GPIO

The SPIDER also includes 3 GPIO pins (general purpose input output): IO1, IO2 and IO3 which can be used to generate and/or read digital signals. This allows the SPIDER to control simple external device (e.g. shutter open/closed) and read a start/stop acquisition signal (e.g. external trigger). The direction (input or output) of each GPIO can be configured individually via software. If a GPIO is configured as an output, then it can be defined the logic status: 1=High (+3.3V) or 0=Low (0V).

GPIO Input/output Voltage characteristics (Typ. Ta=25°C)					
Parameter	**Min**	**Typ**	**Max**	**Unit**	**Note**
Input Voltage	0		10	V	input voltage range
VIL	0		1	V	input voltage for level “LOW”
VIH	2.31		10	V	input voltage for level “HIGH”
Output Voltage	0		3.3	V	output voltage range
VOL	0		0.4	V	output voltage for level “LOW”
VOH	2.9		3.3	V	output voltage for level “HIGH”
t_{rise} , t_{fall}		<500		ns	rise/fall time, 10–90%, 1 pF load
Trigger input frequency			170	Hz	When used to request single shots at 120 kSps sample rate

Table 6

4. General Characteristics

General Characteristics (Typ. Ta=25°C)					
Parameter	Min	Typ	Max	Unit	Note
Power Consumption		2.1		W	during normal use
Power supply voltage	10	12	14	V	input power supply voltage range
Output series resistance		1.5		kΩ	on the bnc
Output analog voltage	0		10	V	on the bnc
Weight		218		g	
Package size	101.5x81x28.5			mm	external size
Operating Temp	10		40	°C	
Storage Temp	-20		70	°C	

Table 7

5. Software

SPIDER does not require any driver installation. In this way, truly portable applications can be developed. Communication between a PC (running Microsoft Windows) and SPIDER takes place over USB using a .NET assembly: *SpiderLibrary.dll.*

Using *SpiderLibrary.dll*, applications can be developed for SPIDER in any of the many programming languages that can interface with .NET assemblies, such as C#, C++, Python, MATLAB, LabVIEW, and many others.

NIREOS has developed example applications for SPIDER in both C# and LabVIEW. We include the source code for both of these applications for users to aid in developing their own applications.

We have also included the compiled .exe for each of these applications for users to begin using SPIDER immediately. The C# version (*SpiderGUI.exe*) is windows-native and can be used directly. The LabVIEW version features some additional features, but needs to be installed (*SPIDER_installer.exe*) as it requires the LabVIEW runtime (no license necessary).

Finally, a software-development kit (SDK) is provided for use in LabVIEW, which makes developing applications even simpler than using the DLL.

In the section below, we first introduce the core concepts of using *SpiderLibrary.dll*, and then discuss their implementation in the C# and LabVIEW examples.

5.1 SpiderLibrary.dll

SpiderLibrary.dll is a .NET assembly that has been designed to simplify, as much as possible, the serial communication between the SPIDER and the computer for the final user.

The list of functions provided by *SpiderLibrary.dll* are described below. Each function is used to send a specific command to the SPIDER (e.g. set GPIO, set gain, start/stop acquisition, and so on).

When the computer receives data from SPIDER, one or more events can be raised. The list of possible events and their associated parameters are listed below.

The GUI must manage the different events and defines a specific action for each of them.

Events and Event Parameters

- SerialStatusChanged //Invoked when the connection status is changed (Spider connected/disconnected).
  - IsActive=1 (connected) or 0 (disconnected)
- SerialError // Invoked when a transmission error is detected.
  - e.Error=0 (invalid data), 1 (time out) or 2 (transmission error)
- SerialPacketReceived // Invoked when a serial packet is received.
  - e.Voltage_Si_array = array of double containing the Si output in V
  - e.Voltage_InGaAs_array = array of double containing the InGaAs output in V
  - e.status_GPIO1 = 0 (IO1=low) or 1 (IO1=high)
  - e.status_GPIO2 = 0 (IO2=low) or 1 (IO2=high)
  - e.status_GPIO3 = 0 (IO3=low) or 1 (IO3=high)
  - e.dir_GPIO1 = 0 (IO1=input) or 1 (IO1=output)
  - e.dir_GPIO2 = 0 (IO2=input) or 1 (IO2=output)
  - e.dir_GPIO3 = 0 (IO3=input) or 1 (IO3=output)
  - e.gain_index_Si = gain # for Si amplifier
  - e.gain_index_InGaAs = gain # for InGaAs amplifier
- GPIOStatusChanged // Invoked when there is a GPIO status changed.
  - e.status_GPIO1 = 0 (IO1=low) or 1 (IO1=high)
  - e.status_GPIO2 = 0 (IO2=low) or 1 (IO2=high)
  - e.status_GPIO3 = 0 (IO3=low) or 1 (IO3=high)
  - e.dir_GPIO1 = 0 (IO1=input) or 1 (IO1=output)
  - e.dir_GPIO2 = 0 (IO2=input) or 1 (IO2=output)
  - e.dir_GPIO3 = 0 (IO3=input) or 1 (IO3=output)
- GainSiChanged // Invoked when the Si gain is changed by pressing the external button.
  - e.ReceivedData = gain # for Si amplifier
- GainInGaAsChanged // Invoked when the InGaAs gain is changed by pressing the external button.
  - e.ReceivedData = gain # for InGaAs amplifier

Functions

- ***public bool OneShot()***

  Starts the acquisition of a single packet (settings + 256 samples for Si + 256 samples for InGaAs). Returns “true” if the command is correctly executed.

  Once the SPIDER has acquired the Si and InGaAs samples and correctly sent the packet to the computer, the events SerialPacketReceived, GainInGaAsChanged, GainSiChanged and GPIOStatusChanged might be invoked.

- ***public bool StartContinuous()***

  Starts a continuous acquisition of multiple packets. Returns “true” if the command is correctly executed.

  The SPIDER starts the acquisition of Si and InGaAs channels and sends to the computer one packet (settings + 256 samples for Si + 256 samples for InGaAs) every t_{packet} (see E3.4). The events SerialPacketReceived, GainInGaAsChanged, GainSiChanged and GPIOStatusChanged might be invoked at each packet received.

- ***public bool StopContinuous()***

  Stops the continuous acquisition. Returns “true” if the command is correctly executed.

  The SPIDER stops acquiring the samples and stops the packets transmission. The events GainInGaAsChanged, GainSiChanged and GPIOStatusChanged may still be invoked if an external gain button is pressed or if there is a change in at least one GPIO status.

- ***public bool ReadStatus()***

  Reads the SPIDER Status: GPIO Status, GPIO Directions, Si and InGaAs gains. Returns “true” if the command is correctly executed.

  If the Spider is not acquiring, will return one packet containing only the settings information.

- *public bool SetGainInGaAs(int index)*
  Sets the gain of the InGaAs transimpedance amplifier, index=integer value between 0 and 7 representing the Gain # as in Table 2. Returns “true” if the command is correctly executed.

- *public bool SetGainSi(int index)*
  Sets the gain of the Si transimpedance amplifier, index=integer value between 0 and 7 representing the Gain # as in Table 2. Returns “true” if the command is correctly executed.

- *public bool SetDirectionGPIO(byte gpio, byte direction)*
  Sets the direction of a particular GPIO, gpio=1 if IO1, 2 if IO2 and 3 if IO3. Direction=0 if Input, 1 if Output. Returns “true” if the command is correctly executed.

- *public bool SetStatusGPIO(byte gpio, byte status)*
  Sets the status of a particular GPIO, gpio=1 if IO1, 2 if IO2 and 3 if IO3. Status=0 if Low (0V), 1 if High (3.3V). Returns “true” if the command is correctly executed.

- public double ComputeEqBW(int gain_index, int SR_index, int filter_index)
  Computes the equivalent bandwidth as the minimum between the analog and digital bandwidth. gain_index=integer between 0 and 7 representing the Gain # as in Table 2. SR_index=integer between 0 and 10 representing the sampling rate index as in Table 3. filter_index=0 for Wideband of 1 for Sinc filter. Returns the equivalent bandwidth as a double.

- public bool SetAdcDigFilter(int data)
  Set the ADC digital filter type. data=0 for wideband filter and data=1 for sinc filter.

- public bool SetSamplingRate(int sampling_rate_index)
  Set the ADC sampling rate to the SR index = sampling_rate_index (from 0 to 10). See Table 3.

- public bool TriggerOutMode(int sampling_rate_index)
  Configures GPIO 1 as a trigger output (can not be applied to GPIO 2 or 3). This can enable synchronisation of external instruments to SPIDER. Note that synchronisation is not per sample, but per shot/packet (256 samples).

mode = sampling_rate_index (0, 1, or 2), where:

- 0: disabled [no output]
- 1: Trig Out (÷ by 2) [output changes state once per packet. This could, for example, modulate a laser or shutter, enabling differential detection. See calcDifferentialSignal.vi]
- 2: Trig Out [one output pulse per packet]

- **public bool SetTriggerFunction(byte enable, byte rise_Nfall, byte gpio, byte function, int data)**

  Set the trigger function for a particular GPIO. The Spider can be programmed to automatically execute a specific function (see Table 8) when a trigger event occurs.
  Enable=0 or 1 to disable or enable, respectively, this feature.
  rise_Nfall=0 for a falling edge trigger and rise_Nfall=1 for a rising edge trigger.
  gpio=1 if IO1, 2 if IO2 and 3 if IO3.
  function=value corresponding to the desired function to be executed with the trigger.
  data=value from 0 to 65535 required by some functions.

  Note that attempting to trigger events (such as "one shot") faster than the time required to collect one packet (256/SR) or greater than the max trigger input frequency (see Table 6) may generate unexpected results, including lost or corrupt packets.

Function	Data	Description
16	0 to 10	Set the ADC sampling rate to SR index = data (as in Table 3)
17	0 to 7	Set gain of Si with the Gain # = data (as in Table 2)
18	0 to 7	Set gain of InGaAs with the Gain # = data (as in Table 2)
32	x	Start one shot
33	x	Start continuous acquisition
34	x	Stop continuous acquisition
48	x	Increment Si gain by one step
49	x	Increment InGaAs gain by one step
51	x	Decrement Si gain by one step
52	x	Decrement InGaAs gain by one step

Table 8

5.2 GUI example & source code in C#

Figure 6: Demo GUI for SPIDER designed in C# in Visual Studio

Find and launch the example GUI application from the following location on the supplied USB stick:

`…\SPIDER_C#\GUI\SpiderGUI.exe`

Note that SpiderLibrary.dll must be present in the same folder. The source code can be opened in Visual Studio, and can be found in `…\SPIDER_C#\Spider\` .

The interface controls are as follows:

- Port: select the serial COM port of SPIDER, go to Device Manager in Windows to get the correct COM port number
- Connect: open the serial connection passing the COM port defined in the comboBox
- Status: is “connected” or “not connected” depending on the serial connection status
- Sampling Rate: set the ADC sampling rate and recalculates the plot time array
- ADC Dig. Filter: Set the adc digital filter
- Gain Si/InGaAs: set the Si/InGaAs amplifier gain
- BW: show the equivalent bandwidth that is equal to the minimum between the analog and digital bandwidth
- IO1,2,3: read the IO status, set the direction and logic status

- START ONESHOT: Start the acquisition of a single packet, this is plotted once correctly received
- START/STOP: Start and Stop continuous acquisition
- READ STATUS: read Si/InGaAs selected Gains and GPIO status/direction
- Folder: open a window to define the saving folder
- Save Data: save plot data as .csv file, it is enabled once the Saving folder is defined
- Record data stream: if checked, the data stream is saved when pressing START/STOP buttons
- Y scaling mode: switch the plot Y axis scaling mode between automatic/manual
- plot Si/InGaAs: if checked, plot of Si/InGaAs is enabled
- Mean/RMS: mean and rms value of the plotted channel

5.3 LabVIEW SDK

SPIDER comes with a library of LabVIEW virtual instruments (VIs) implemented using the .dll described in section 5.1, that serves as a software development kit (SDK) that users can use to build their own applications with SPIDER. The library (*SPIDERlib.lvlib*), with associated VIs and dependencies, can be found on the USB drive delivered with SPIDER in the following folders:

```
\Software\SPIDER_LV\SPIDER_SDK_LabVIEW_2013\
\Software\SPIDER_LV\SPIDER_SDK_LabVIEW_2019\
```

These files are provided unlocked and with an open license, so that they can be used as-is, or adapted for use as required. The only difference between the 2013 and 2019 versions are aesthetic (2019 looks nicer, so use that if possible!).

In addition, several example applications are included, and a complete GUI (*Spider_GUI.vi*) has been developed that mirrors the features of the C# GUI described in section 5.2. The examples are intended as learning tools and templates for user applications. The GUI naturally is more complex, and serves as a means for demonstrating the full feature set of SPIDER and how they can be implemented in LabVIEW.

Installation

The SPIDER files are delivered as a LabVIEW library (*SPIDERlib.lvlib*), which requires an installation of LabVIEW 2013 or later. The library is compatible with both 32- and 64-bit versions of LabVIEW.

To begin, copy the entire folder to a convenient location on your computer, and open the library. It is important that the accompanying files and folders (such as *SpiderLibrary.dll*) are included with the library.

Each of the VIs in the SDK has accompanying help information, describing their functionality. While included in this manual, the relevant entries can also be viewed within LabVIEW by activating the Context Help by clicking on the tool bar *Help → Context Help*, or pressing CTRL+H.

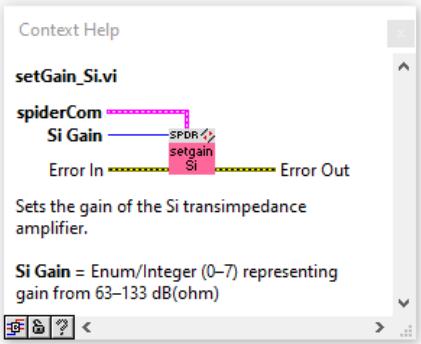


Figure 7: A typcal context help entry

An example is shown in Figure 7. Note that essential inputs are shown in bold (**Si gain**), while optional inputs are shown in regular type.

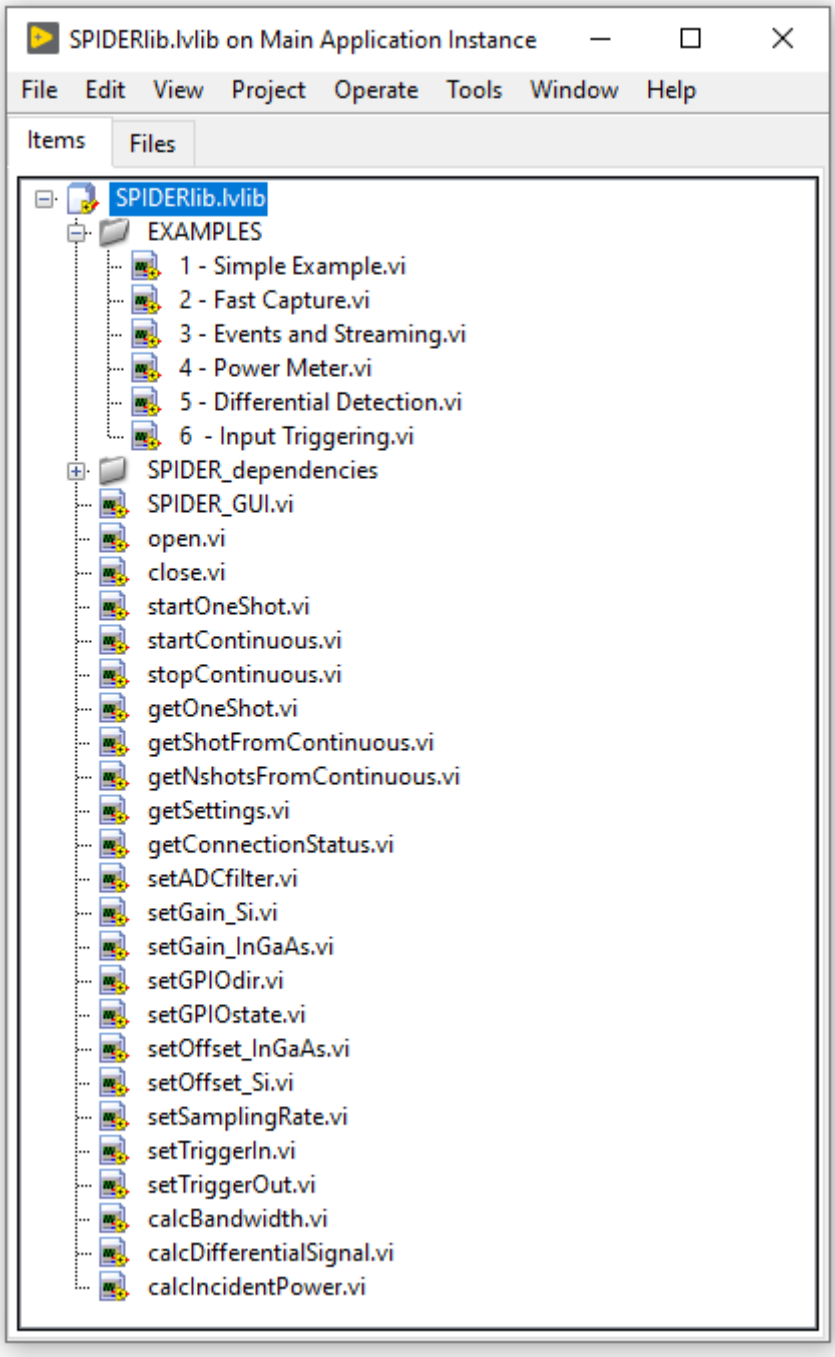


Figure 8: SPIDERlib.lvlib, the SPIDER SDK for LabVIEW

Example 1: Basic operation

Every application calling SPIDER must begin (and end) by opening (and closing) a connection. This is done, appropriately, by *open.vi* (and *close.vi*). *Open.vi* generates a cluster called *SpiderCOM*, which contains the various references required to manage communication between LabVIEW and the .DLL described in section 5.1.

This is demonstrated in …\E*XAMPLES\1 – Simple Example.vi,* which simply opens a connection to SPIDER, takes a single shot using *getOneShot.vi*, and returns the resulting data packet[1] before closing the connection. Note that the *SpiderCOM* cluster must be connected to *getOneShot.vi* for it to communicate with SPIDER.

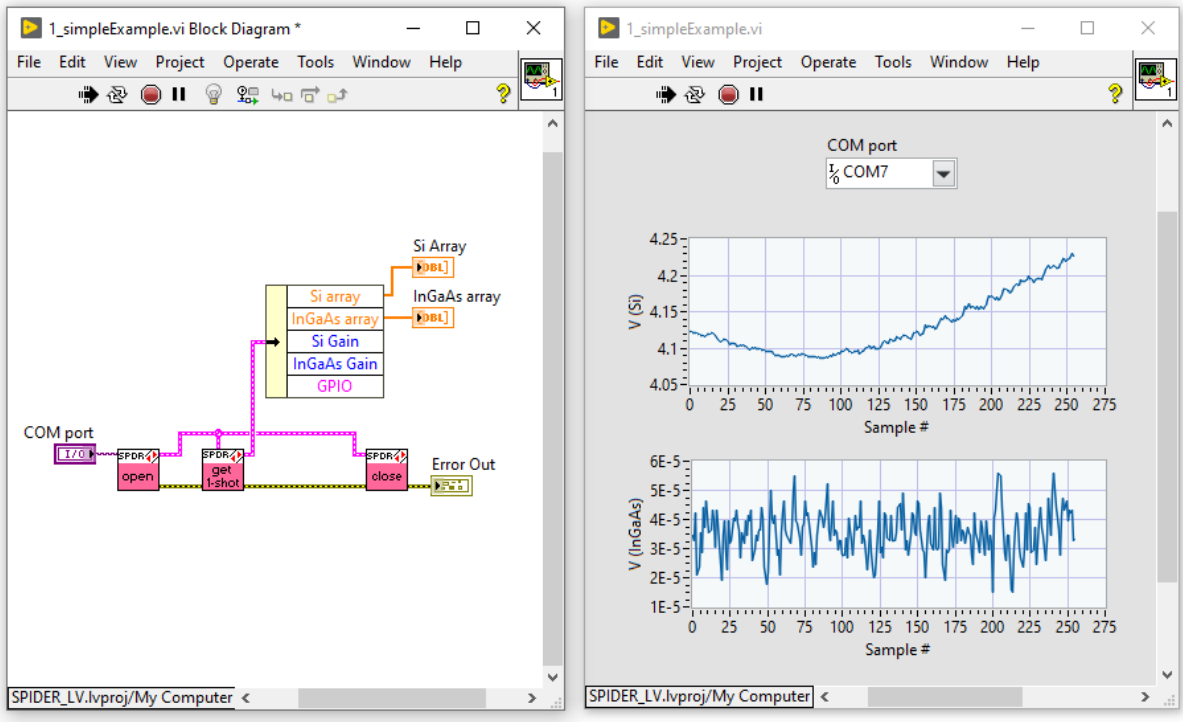


Figure 9: Example 1 – Simple Example.vi, perhaps the simplest possible SPIDER application

Here we see the fundamental unit of data transmitted by SPIDER: a packet, which in the LabVIEW SDK is presented as a cluster. This packet contains:

[1] As described in section 3.3, the minimum measurement (or 'shot') that can be made by SPIDER is comprised of 256 samples for each of the Si and InGaAs detectors. These two arrays are then sent to the computer as a single 'packet'. Each packet also contains the gain settings for each channel, and state of each of the three GPIOs during that packet acquisition.
Note that while the maximum sample rate for the detectors is 120 kS/s, as the GPIOs are only read once per packet, their effective maximum sample rate is 120k/256 = 468.75 S/s.

Packet

Si Array [*256×1 array of doubles*], the 256 voltage samples from the Si transimpedance amplifier

InGaAs Array [*256×1 array of doubles*], the 256 voltage samples from the InGaAs transimpedance amplifier

Si Gain [*integer*], from 0–7 representing gain from 64–133 dB_Ω (see Table 2)

InGaAs Gain [*integer*], from 0–7 representing gain from 64–133 dB_Ω (see Table 2)

GPIO

1

Direction [*integer*], specifying GPIO direction. 0 = input; 1 = output.

Status [*integer*], specifying GPIO state. 0 = low (0 V); 1 = high (3.3 V).

2

Direction [*integer*], specifying GPIO direction. 0 = input; 1 = output.

Status [*integer*], specifying GPIO state. 0 = low (0 V); 1 = high (3.3 V).

3

Direction [*integer*], specifying GPIO direction. 0 = input; 1 = output.

Status [*integer*], specifying GPIO state. 0 = low (0 V); 1 = high (3.3 V).

Example 2: Continuous acquisition for fast operation

While requesting a single shot is simple, there is some built-in delay (typically some tens of milliseconds) while SPIDER receives the command, collects the samples, and transmits the data.

For applications where the user wishes to minimise lost time between measurements (which in our experience is **every** application), it is more efficient to have SPIDER continuously acquire and transmit packets of data, and for the user to retrieve packets from this stream as desired.

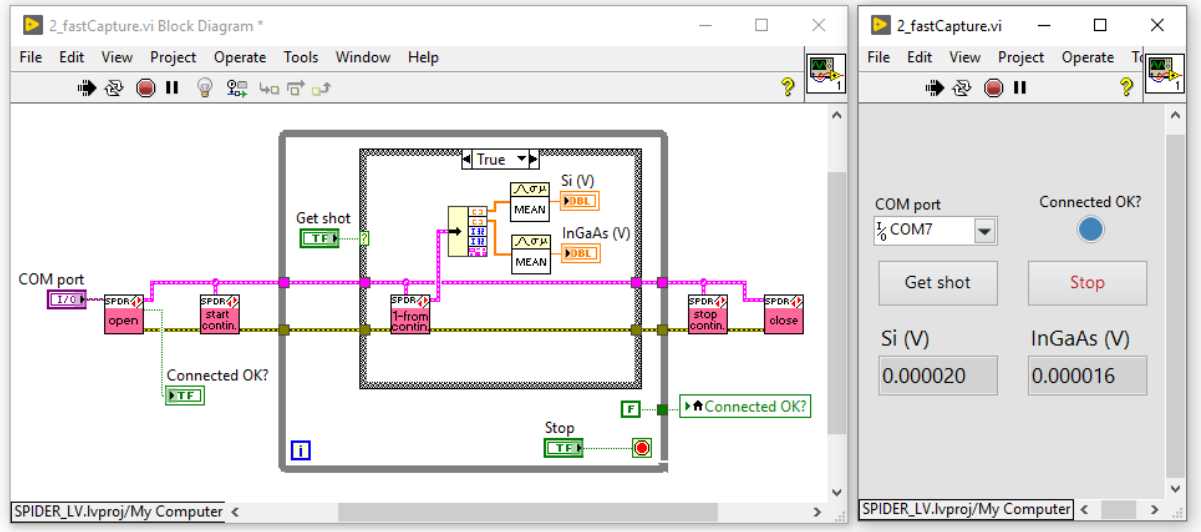


Figure 10: Example 2 – Fast Capture.vi

As shown in example *2 – Fast Capture.vi* (and in Figure 10) This is managed using several VIs. First, *startContinuous.vi* is called, and SPIDER begins continuously acquiring and transmitting data packets (which are ignored, unless a structure is configured to receive the .NET events used by SPIDER to transmit data). When the user wishes to collect a packet, they can call either *getShotFromContinuous.vi* or *getNshotsFromContinuous.vi*. These VIs first clear the queue (to not return data recorded prior to the VI call), then return the next shot (or N shots) transmitted by SPIDER. Finally, *stopContinuous.vi* or *close.vi* are called to stop continuous acquisition.

This is the fastest way to collect a shot or cluster of shots on demand. However, it does not allow for continuous streaming of data without missing shots – for that, we will need to use event-driven operation, as shown in the next section.

Example 3: Event-driven operation for maximum data throughput

At the maximum sample rate of 120 kS/s per channel (24-bit), SPIDER generates 469 packets – about 1 MB of data – per second. At this rate, truly continuous data acquisition (i.e., without dropped shots) requires careful structuring of our LabVIEW project.

First a quick overview: LabVIEW communicates with SPIDER via the SpiderLibrary.dll, which is a .NET assembly and uses an event-driven programming paradigm. As described in Chapter 5, SPIDER generates .NET events to both announce and transmit each packet (as well as other events, like changing channel gain, or USB disconnection). These .NET events are registered and converted to LabVIEW events by *open.vi*, which outputs an array of references in the **spiderCOM** cluster called **LVevents**.

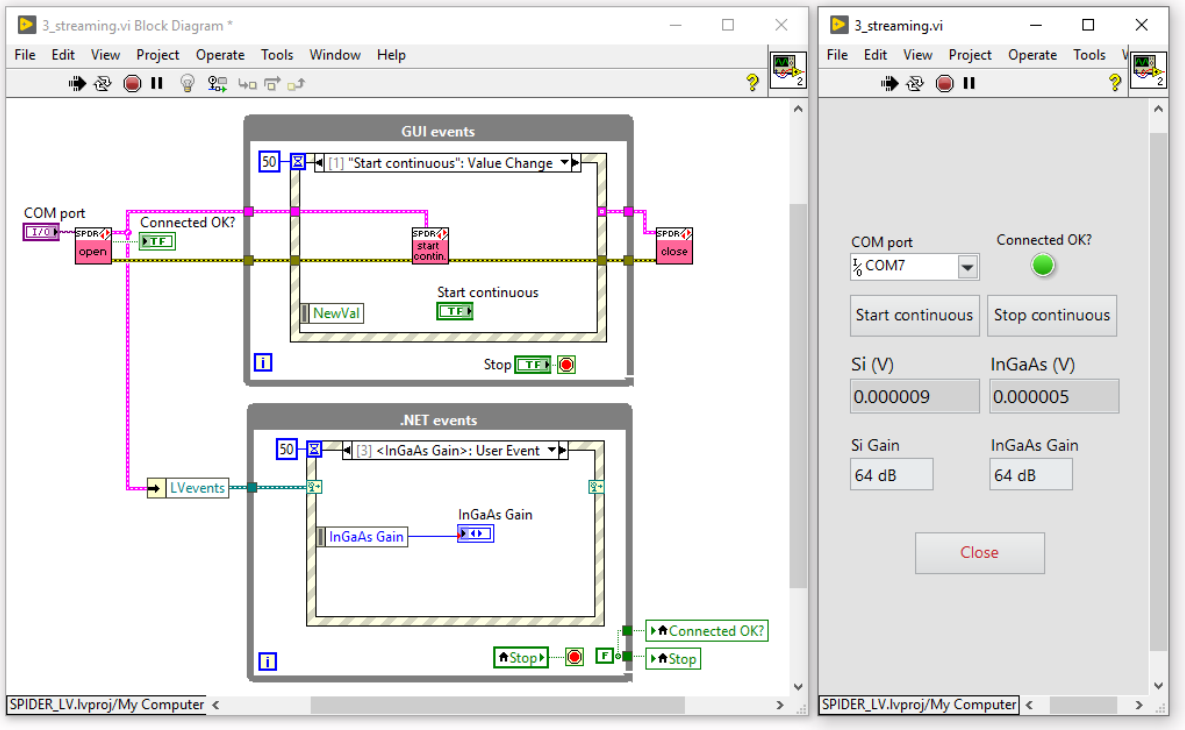


Figure 11: Example 3 – Events and Streaming.vi

Example *3 – Events and Streaming.vi* (see also Figure 11) gives an example of how to use the **LVevents** array in a producer–consumer architecture to continuously stream data from SPIDER. This is also the general structure used in *SPIDER_GUI.vi*.

By wiring **LVevents** to the Dynamic Event Terminal of a LabVIEW event structure[2], we reveal new options under "Add Event Case…" that correspond to the SPIDER events described on Page 16. In this way, we enable LabVIEW to respond to those events (such as transmission of a data packet, or change in GPIO state). Additionally, the data transmitted by those events is available in the Event Data Node for that event.

This structure enables efficient and continuous data transfer from SPIDER to LabVIEW, and unlike in Example 2, does not cause any elements to "hang" while waiting for data, as the events are only triggered when the packets are ready.

Note that placing any resource demanding components (such as graphs) in the data collecting loop may slow down performance and result in data loss. Additionally, writing data to disk is not trivial at the fastest sample rates, and use of a buffer (as seen in *SPIDER_GUI.vi*) or similar solution may be required.

[2] if not visible, right click on the event structure frame → "Show Dynamic Event Terminals"

Example 4: Power meter

SPIDER exhibits many of the characteristics required of a high-quality power meter: excellent sensitivity and linearity, broadband response, and programmable gain. Furthermore, the relatively low bandwidth (~50 kHz) enables SPIDER to integrate the signals from short pulses without saturating the transimpedance amplifiers.

However, the small active area of the photodetectors[3] – and the fact that they are not equivalent – makes SPIDER somewhat impractical as a true power meter. For this reason, SPIDER units are not individually calibrated. In other words, while SPIDER is not an ideal *absolute* power meter, it is a fantastic *relative* power meter.

By considering the amplifier gain ($Z_{Si,InGaAs}$), and typical (wavelength-dependent) responsivity of the detectors ($R(\lambda)$), we can calculate the approximate power incident on each detector. As discussed in Section 3.2, the wavelength-dependent relation between incident optical power ($P_{in}(\lambda)$) and output voltage of the transimpedance amplifier ($V_{Si,InGaAs}$) is given by Equation 3.1:

$$V_{Si,InGaAs} = P_{in}(\lambda) \cdot R(\lambda) \cdot Z_{Si,InGaAs}$$

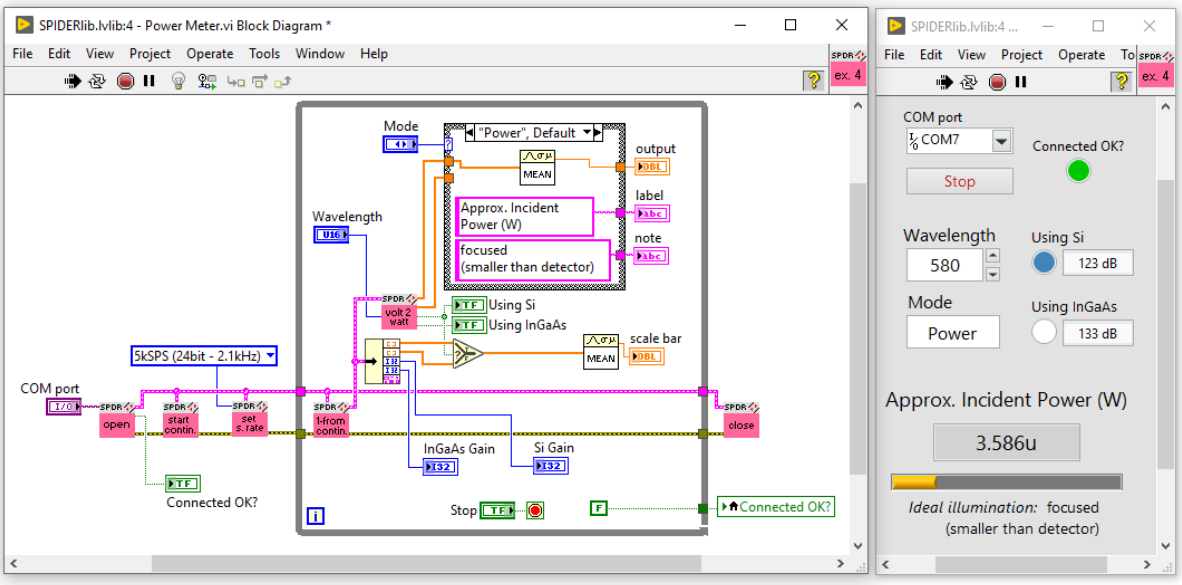


Figure 12: Example 4 – Power Meter.vi

Example 4 (shown in Figure 12) shows a simple implementation of SPIDER as a power meter. The general structure is the same as introduced in Example 2. However, here the output packet

[3] 2.4×2.4 mm square for Si, and 1 mm diameter circular for InGaAs

from *getShotFromContinuous.vi* is passed through a new VI, *calcIncidentPower.vi*, which (unsurprisingly) converts the measured voltages to optical power (in Watts). This is done using an additional input to specify wavelength, which determines whether to use the voltage from the Si or InGaAs detector, and applies the correct responsivity correction.

For applications such as this, where the absolute values of the two channels are considered, it is important to remember that the Si and InGaAs elements have different dimensions (see Chapter 3.1). Our power meter example therefore features two modes. Power is for measuring (in W) focused signals that are smaller than 1 mm diameter, and therefore completely collected by the photodiodes. Intensity is for measuring homogeneous unfocused signals, and includes a correction for the active area of each detector to give intensity in W/m^2.

Example 5: Differential detection

Many measurement systems take advantage of modulation to recover signals from noisy backgrounds. The integrated GPIO and high bit-rate of SPIDER mean that we too can partake of these delightful techniques.

In Example *5 – Differential Detection.vi* (Figure 13), we see a simple implementation of a differential detection setup.

Imagine, for example, that we desire to measure a weak photoluminescence (PL) emission from a sample in a well illuminated room. If the excitation source is a laser that can be modulated, we can use differential detection to recover the PL from the bright background.

By calling *setTriggerOut.vi* with mode set to "Trig Out (÷ by 2)", GPIO #1 will be set as an output, and during continuous acquisition will change state each shot (i.e., every 256 samples). If the output from GPIO #1 is now used to modulate the laser output, the packets measured by SPIDER will be alternatively measured during periods where the laser (and therefore the PL signal) is on and off.

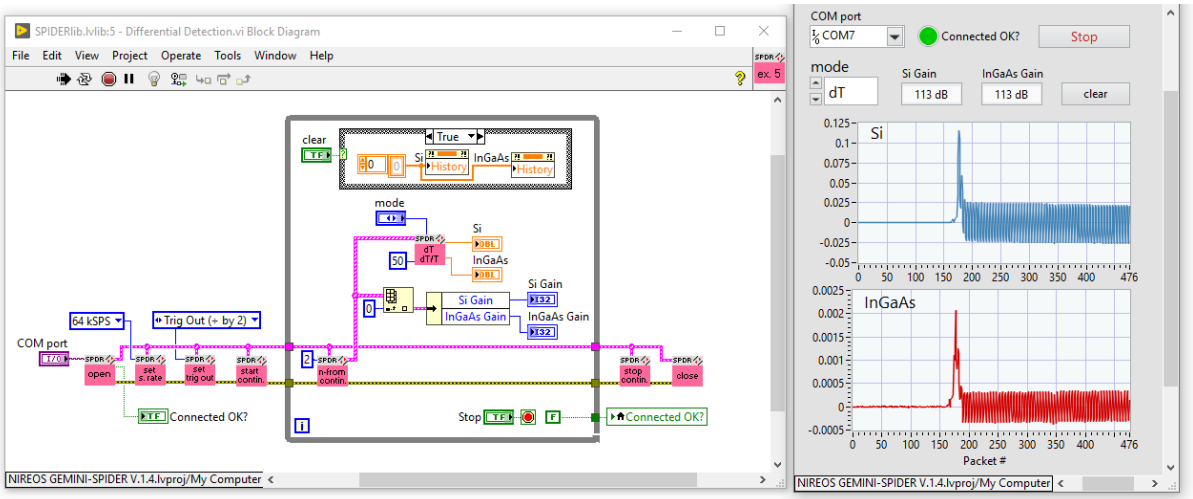


Figure 13: Example 5 – Differential Detection.vi

In our example, we are using *getNshotsFromContinuous.vi* to collect pairs of packets. These pairs (which are output as **Packets**, as an array of clusters) are then sent to *calcDifferentialSignal.vi*. As **Packets** contains the GPIO state for each shot, *calcDifferentialSignal.vi* is able to determine which packet was measured while the laser was active (referred to as T(1)), and which was with laser off (T(0)). Using the **mode** selector, we can now choose which signal to visualise:

- T(0): direct signal without laser
- T(1): direct signal with laser
- dT = T(1) – T(0): differential signal
- dT/T = (T(1) – T(0))/T(0): normalised differential signal

Example 6: External triggering

Any of the three GPIOs of SPIDER can be configured to trigger data acquisition (or several other actions, such as changing gain) from an external voltage source. This is configured using *setTriggerIn.vi*. An example of how this can be implemented is shown in Example *6 – Input Triggering.vi* (Figure 14).

In this example, we are receiving packets from SPIDER using an event structure such as in Example 3. *setTriggerIn.vi* is called, with arguments in this case specifying that SPIDER should take one shot whenever a rising voltage (i.e. from 0 V to 3.3 V) is detected on GPIO #2.

The three GPIOs can be programmed separately by calling *setTriggerIn.vi* more than once. For example applying a rising voltage to GPIO 2 and 3 could be set to start and stop acquisition, respectively.

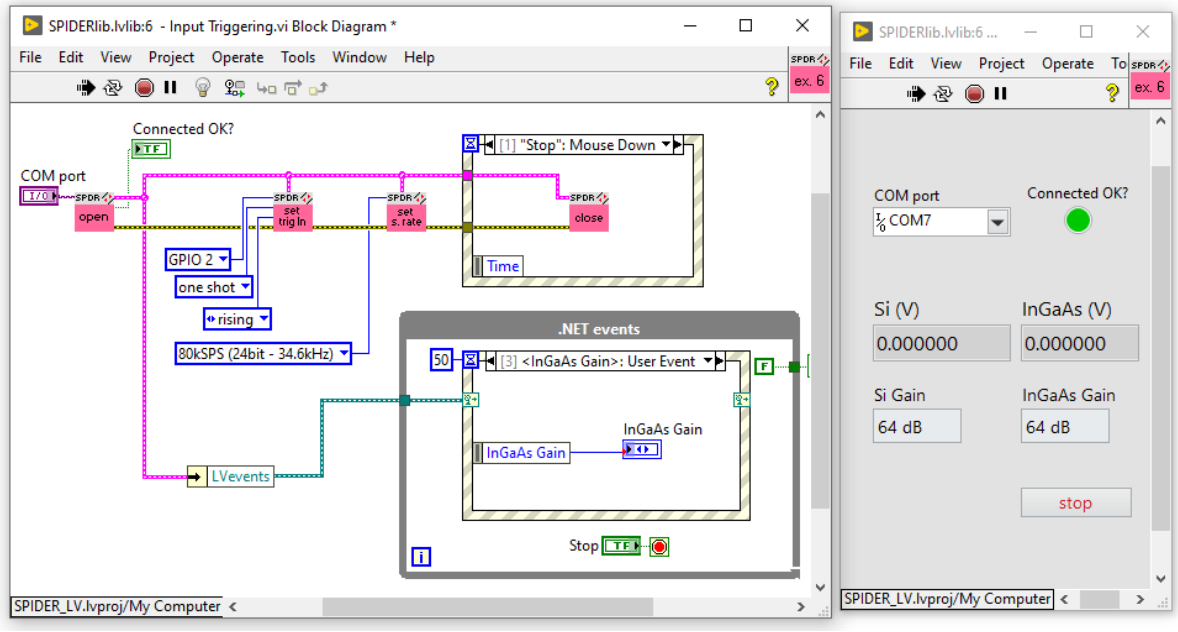


Figure 14: Example 6 – Input Triggering.vi

The list of actions able to be triggered in this way is set via the **function** input of *setTriggerIn.vi*, with options as follows:

0: disabled [no action]

1: one shot

2: start continuous

3: stop continuous

4: increase gain (Si)

5: decrease gain (Si)

6: increase gain (InGaAs)

7: decrease gain (InGaAs)

Note that attempting to trigger events (such as "one shot") faster than the time required to collect one packet (256/SR) or greater than the max trigger input frequency (see Table 6) may generate unexpected results, including lost or corrupt packets.

GUI example & in LabVIEW

Section 5.2 presents a graphical user interface (GUI) for SPIDER built in C# using the SpiderLibrary.dll. *SPIDER_GUI.vi* (Figure 15) is a recreation of the same GUI constructed using the LabVIEW SDK.

SPIDER_GUI.vi is useful for testing and demonstrating SPIDER's capabilities. Furthermore, as the VI is not locked, all implementations in the back panel can be freely investigated. In fact, we encourage users to copy and adapt any of these components in their own applications. In this way, the GUI itself serves to act as a library of methods and ideas for how to implement each of SPIDER's functions.

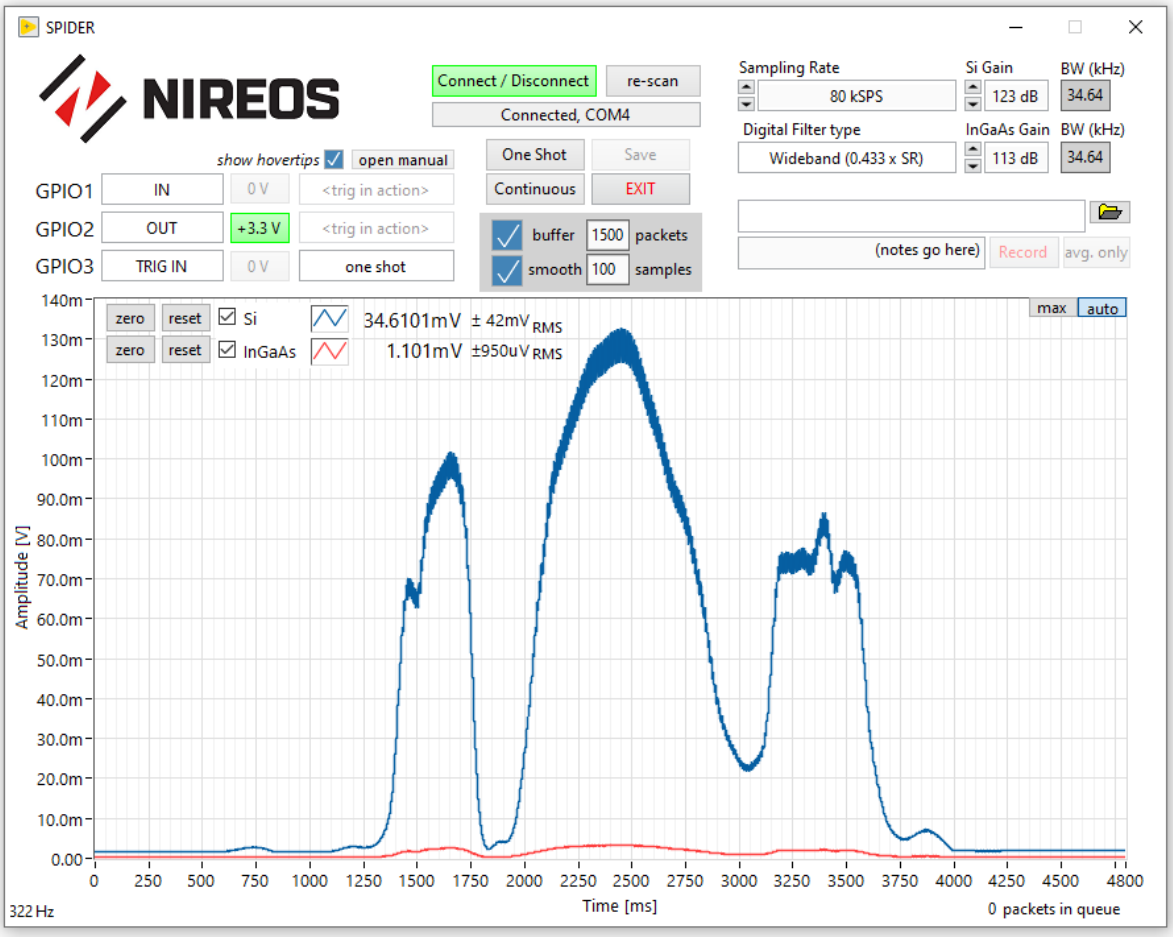


Figure 15: SPIDER_GUI.vi: A SPIDER GUI implemented in LabVIEW

To get more information about any of the front panel controls in *SPIDER_GUI.vi*, ensure the "show hovertips" box is selected, and hover over the element with the mouse pointer to show a short description.

Several features have been added to this GUI that are not present in the C# version. Selecting buffer retains the data from previous shots, so that trends over time can be tracked. Smooth applies a moving average filter to the displayed data. These two features combined enable detection of very weak signals, as low as several picowatts. On a powerful computer, they can be increased to buffer and smooth 1000's of packets. This can be very CPU-intensive. To monitor this, a counter in the bottom-right corner of the window indicates how many packets are in the queue passing data to the front panel graph. This can be an indicator of CPU load – if the queue reaches 500, it will begin to loose packets from the stream.

To export data, first specify a target folder in the path field on the right of the interface. Following this, the options Save, Record, and Avg. only will become available. When exporting data, filenames will be automatically generated using the current time and date.

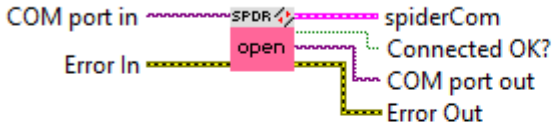
Save exports a file with the current data shown in the front panel figure, including any smoothing and accumulation.

Record, if enabled during continuous acquisition, will stream data continuously to disk, without smoothing. If Avg. only is selected, only the average voltage from each packet of 256 samples will be recorded, drastically reducing file size. If not continuously acquiring, using One shot (or externally triggering a single shot) with record enabled will save single shots, as if pressing save.

Individual VIs

Below are replicated each of the individual context help descriptions for the VIs included in the SPIDER LabVIEW SDK:

open.vi



Creates a connection to SPIDER via SpiderLibrary.dll and registers the callback functions neccessary for LabVIEW to respond to the .NET events that SPIDER uses to transmit data and status information.

COM port in: COM port to attempt to connect to SPIDER.
If **COM port in** is empty or not connected, or if initial connection is not successful, the .vi will attempt to autodetect and connect to the correct COM port. If multiple possible SPIDERs are detected, the user will be prompted to select the correct port.

Users can find the correct COM port in Windows Device Manager, where SPIDER presents as "USB Serial Device".

spiderCOM: a cluster containing the references required by all other VIs in this library to communicate with SPIDER.

Connected OK?: returns TRUE or FALSE indicating whether connection was successful.

COM port out: indicates the final COM port that was used in case of successful connection.

---

close.vi

Stops acquisition (if any is occuring), closes a connection to SPIDER, and frees resources by de-registering all references and callbacks.

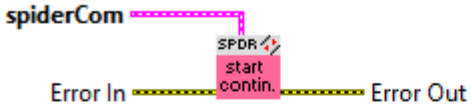
---

startOneShot.vi

Calling **startOneShot.vi** will command SPIDER to collect and transmit one packet (256 samples per channel).

IMPORTANT: This VI does not return any data itself, and should be used in combination with an event structure to receive the packet once collection is complete. See .../EXAMPLES/**3_events_and_streaming.vi** for how to use event-driven communication with SPIDER.

startContinuous.vi



Commands SPIDER to begin continuous acquisition, which enables faster data collection than by using single shots.

IMPORTANT: This VI does not return any data itself.

After starting continuous acquisition, use either **getShotFromContinuous.vi** or **getNshotsFromContinuous** to receive either a single or group of packets.

For truly continuous data collection, use an event structure, as shown in .../EXAMPLES/**3_events_and_streaming.vi**.

Acquisition can be stopped using either **stopContinuous.vi** or **close.vi**.

stopContinuous.vi

Commands SPIDER to stop continuous acquisition.

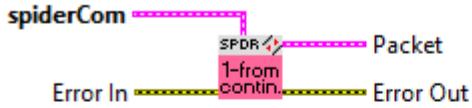
getOneShot.vi

Commands SPIDER to take a single shot, waits for acquisition to complete, and returns the resulting packet (256 samples for each channel).

Unlike **startOneShot.vi**, this VI does not require the use of an external event structure.

Packet: Cluster containing the arrays of voltage values for both Si and InGaAs detectors, as well as the gain values (index value from 0-7) and GPIO direction and state during the shot.

getShotFromContinuous.vi

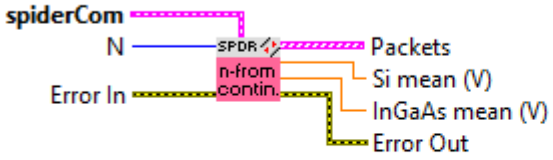


Will return the next packet (256 samples per channel) collected entirely after the VI call (i.e. will not return packets measured partially or fully prior to calling **getShotFromContinuous.vi**).

SPIDER must be in continuous acquisition mode (see **startContinuous.vi**) prior to calling this VI.

Packet: Cluster containing the arrays of voltage values for both Si and InGaAs detectors, as well as the gain values (index value from 0-7) and GPIO direction and state during the shot.

getNshotsFromContinuous.vi



Will return an array of the next N packets collected entirely after the VI call (i.e. will not return packets measured partially or fully prior to calling **getShotFromContinuous.vi**).

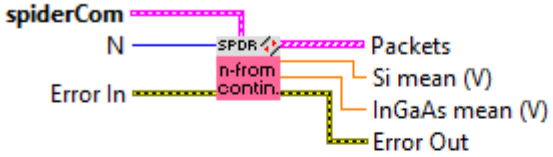
SPIDER must be in continuous acquisition mode (see **startContinuous.vi**) prior to calling this VI.

Packets: Array of clusters , each containing the arrays of voltage values for Si and InGaAs detectors, and the gain values (index value from 0-7) and GPIO direction and state during each shot.

Si mean (V): Double, the mean value (in volts) of all samples from all shots for the Si channel.

InGaAs mean (V): Double, the mean value (in volts) of all samples from all shots for the InGaAs channel.

getNshotsFromContinuous.vi

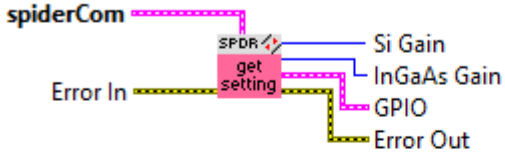


Will return an array of the next N packets collected entirely after the VI call (i.e. will not return packets measured partially or fully prior to calling **getShotFromContinuous.vi**).

SPIDER must be in continuous acquisition mode (see **startContinuous.vi**) prior to calling this VI.

Packets: Array of clusters , each containing the arrays of voltage values for Si and InGaAs detectors, and the gain values (index value from 0-7) and GPIO direction and state during each shot.

getSettings.vi



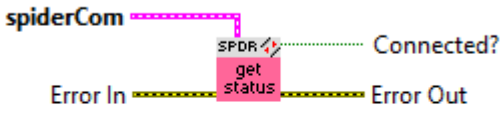
Queries SPIDER and returns the current parameters:

Si Gain: Integer (0–7) representing the current gain (from 63–133 dB(ohm)) of the Si transimpedance amplifier

InGaAs Gain: Integer (0–7) representing the current gain (from 63–133 dB(ohm)) of the InGaAs transimpedance amplifier

GPIO: Cluster containing the direction (0 = input; 1 = output) and state (0 = low, 0 V; 1 = high, 3.3 V) of each of the three GPIOs.

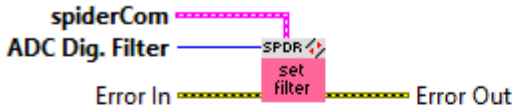
getConnectionStatus.vi



Queries SPIDER to check connection status.

Connected?: Boolean = TRUE (connected) or FALSE (disconnected).

setADCfilter.vi

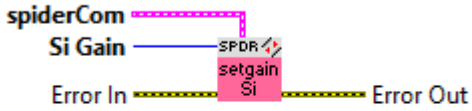


Set the analog-to-digital converter (ADC) digital filter type.

The Wideband filter offers a larger bandwidth, a low ripple pass band, narrow transition band, and high stop band rejection. The filter response is similar to an ideal brick wall filter, making it ideal for frequency domain measurements. The Sinc filter has a response close to a "sinc" function and it is characterized by a low latency signal path that makes it ideal for time domain analysis and DC measurements.

ADC Dig. FIlter: Integer = 0 (wideband filter) or = 1 (sinc filter).

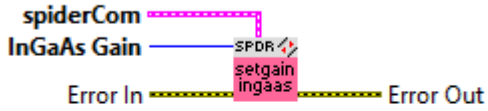
setGain_Si.vi



Sets the gain of the Si transimpedance amplifier.

Si Gain = integer from 0–7 representing gain from 63–133 dB(ohm)

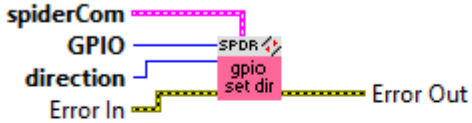
setGain_InGaAs.vi



Sets the gain of the InGaAs transimpedance amplifier.

InGaAs Gain = integer from 0–7 representing gain from 63–133 dB(ohm)

setGPIOdir.vi

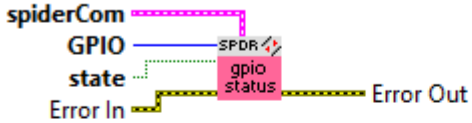


Set the direction (i.e., input or output) of one of the three general-purpose input/outputs (GPIOs).

GPIO: Integer (1,2 or 3) indicating which of the three GPIOs to address.

direction: Integer specifying direction. 0 = input; 1 = output.

setGPIOstate.vi



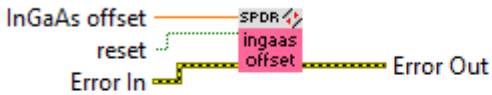
Set the state (i.e., low (0 V) or high (3.3 V)) of one of the three general-purpose input/outputs (GPIOs).

This VI has no effect if the target GPIO direction is set to "input".

GPIO: Integer (1,2 or 3) indicating which of the three GPIOs to address.

state: Integer specifying state. 0 = low (0 V); 1 = high (3.3 V).

setOffset_InGaAs.vi



Sets the voltage offset applied to all values coming from the InGaAs ADC. Can be used (for example) to correct for stray or background light.

Note that this will also offset the apparent saturation voltage of the transimpedance amplifier, which is typically ~ 8.2 V.

Offsets are stored in the global **offsets.vi**, and applied in the callback function **cb_PacketReceived.vi**. Values are not retained after closing/ restarting VIs.

InGaAs offset: Double, new offset value (V).

reset: Boolean, if TRUE will ignore **InGaAs offset** and restore offset to factory values.

setOffset_Si.vi

Sets the voltage offset applied to all values coming from the Si ADC. Can be used (for example) to correct for stray or background light.

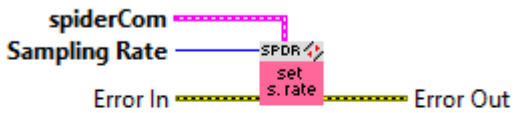
Note that this will also offset the apparent saturation voltage of the transimpedance amplifier, which is typically ~ 8.2 V.

Offsets are stored in the global **offsets.vi**, and applied in the callback function **cb_PacketReceived.vi**. Values are not retained after closing/ restarting VIs.

Si offset: Double, new offset value (V).

reset: Boolean, if TRUE will ignore **Si offset** and restore offset to factory values.

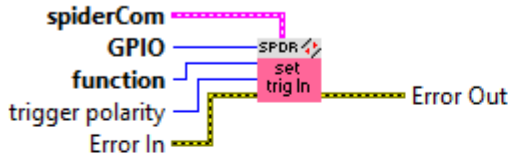
setSamplingRate.vi



Sets the sampling rate of SPIDER.

Sampling Rate: Integer (0–10), corresponding to sampling rate from 0.5–120 kiloSamples per second (kSPS) per channel.

setTriggerIn.vi



Set the input trigger function for a particular GPIO. SPIDER can be programmed to automatically execute a specific function when a trigger event (either rising or falling edge) occurs. The three GPIOs can be programmed separately, for example applying a rising voltage to GPIO 2 and 3 could be set to start and stop acquisition, respectively.

This VI will automatically set the target GPIO direction to "input". Note that attempting to trigger events (such as "one shot") faster than the time required to collect one packet (256/SR) or greater than the max trigger input frequency (see Table 6) may generate unexpected results, including lost or corrupt packets.

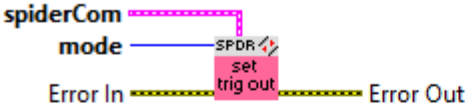
GPIO: Integer (1,2 or 3) indicating which of the three GPIOs to address.

function: Enum/Integer (0–7) indicating what action to take on trigger event. Options are:

0: disabled [no action]
1: one shot
2: start continuous
3: stop continuous
4: increase gain (Si)
5: decrease gain (Si)
6: increase gain (InGaAs)
7: decrease gain (InGaAs)

trigger polarity [optional]: Enum/Integer = 0 (trigger on falling edge) or = 1 (trigger on rising edge)[default].

setTriggerOut.vi



Configures the output trigger mode of GPIO 1 (can not be applied to GPIO 2 or 3). This can enable synchronisation of external instruments with SPIDER.

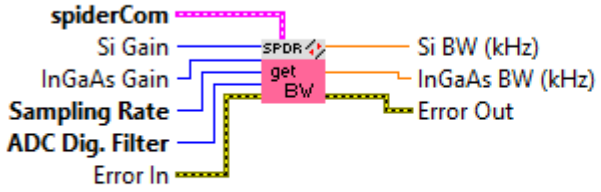
As the GPIO state can only be set once per packet, synchronisation is not per sample, but per shot/packet (256 samples).

This VI will automatically set GPIO 1 direction to "output" if mode >0.

mode: Integer (0–2) indicating output trigger mode. Options are:

0: disabled [no output]
1: Trig Out [one output pulse per packet]
2: Trig Out (÷ by 2) [output changes state once per packet. This could, for example, modulate a laser or shutter, enabling differential detection. See **calcDifferentialSignal.vi**]

calcBandwidth.vi



Calculates the effective bandwidth of the combined amplifier-ADC system. Bandwidth is determined by sampling rate, gain, and choice of digital filter. Depending on the combination of settings, the bandwidth may be limited by either the transimpedance amplifier or the ADC digital filter.

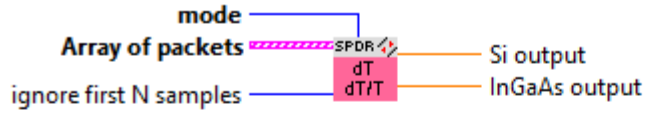
Si Gain: Integer (0–7) representing the gain (from 63–133 dB(ohm)) of the Si transimpedance amplifier.

InGaAs Gain: Integer (0–7) representing the gain (from 63–133 dB(ohm)) of the InGaAs transimpedance amplifier.

Sampling Rate: Integer (0–10) corresponding to sampling rate from 0.5–120 kiloSamples per second (kSPS) per channel.

ADC Dig. FIlter: Integer = 0 (wideband filter) or = 1 (sinc filter).

calcDifferentialSignal.vi



Calculates the differential signal (dT) or normalised differential signal (dT/T) from two shots. Shots are labelled as either T(0) or T(1) based on whether the state of GPIO1 was low or high, respectively.

This mode is intended to be used with modulated experiments, similarly to how one would use a lock-in amplifier. By using **setTriggerOut.vi** to set SPIDER to "÷ by 2" mode, the output of GPIO1 can be used to modulate external instruments such as lasers or shutters.

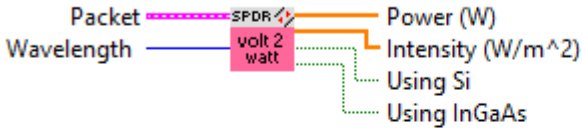
As the GPIO state change is not instantaneous, the first N samples of each packet can be ignored using the integer input **ignore first N samples**.

see also .../EXAMPLES/**5 - Differential Detection.vi**

Mode: Integer (0–2), which determines the mode of operation. Options are:
0, "T" = T(0)
1, "dT" = T(1)-T(0)
2, "dT/T" = (T(1)-T(0))/T(0)

Packets: Array of two clusters , each containing the arrays of voltage values for Si and InGaAs detectors, and the gain values (index value from 0-7) and GPIO direction and state during each shot.

calcIncidentPower.vi



Calculates approximate incident power or intensity of a monochromatic light source.

NOTE: this is only an approximate value, as SPIDER units are not individually calibrated.

Additionally, remember that the Si and InGaAs photodiodes have different active area dimensions (2.4 × 2.4 mm square for Si, and 1 mm diameter circular for InGaAs).

For measurements where the illumination is focused and smaller than both detectors, the **Power (W)** output can be used. For measurements of unfocused signals, the **Intensity (W/m^2)** output includes a correction for the active area of the two detectors.

see also .../EXAMPLES/**4 - Power Meter.vi**

Packet: Cluster containing the arrays of voltage values for both Si and InGaAs detectors, as well as the gain values (index value from 0-7) and GPIO direction and state during the shot.

Wavelength (nm): Integer, center wavelength of the monochromatic light source (used to select appropriate photodiode channel and find responsivity).

6. Troubleshooting

Problem	Solution
The SPIDER does not turn on	➢ Check power supply.
No analog signal out of one or both BNC connectors (or signal very weak/attenuated);	➢ Check if SPIDER is correctly powered. ➢ Check BNC cable, try with a different one if available. ➢ Check if the BNC connector are connected to an external device having a high impedance (>100kΩ). The SPIDER does not work with a 50Ω load.
The PC cannot connect to SPIDER Or PC appears to be connected to SPIDER, but cannot collect measurements	➢ Turn SPIDER power off and back on ➢ Check power supply and USB connections ➢ Ensure the COM port is correct. Check the COM port by looking in Windows → Device Manager → Ports (COM & LPT), and checking which device disappears and appears when SPIDER power is cycled.
Oscillatory artifacts (ripples) appear near sudden changes in signal	➢ Consider using the sinc filter. See Section 3.4: ADC Digital Filter

7. Mechanical Drawing

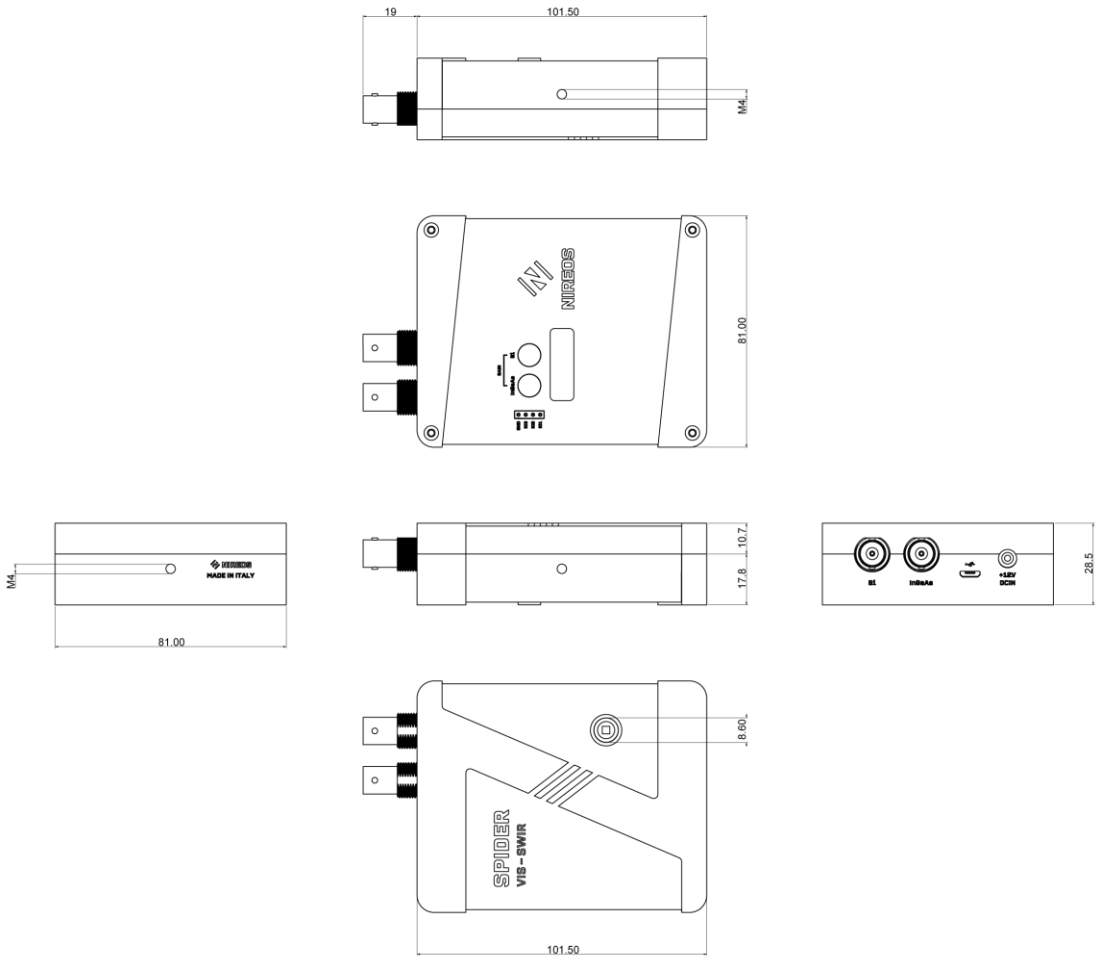


Figure 16: Spider mechanical drawing, all dimensions are expressed in mm


NIREOS