SPECTRUM
INSTRUMENTATION

Perfect fit – modular designed solutions











M2p LabVIEW Driver

Driver for all M2p cards and related digitizerNETBOX, generatorNETBOX or hybridNETBOX products

Installation, Libraries, Data sorting, Examples, Standard mode, FIFO mode

Latest revision: 14. August 2025

Digitizers | Transient Recorders | Arbitrary Waveform Generators | Digital Waveform Acquisition Cards

for PCI Express, PXI Express and LXI / Ethernet

(c) SPECTRUM INSTRUMENTATION GMBH
AHRENSFELDER WEG 13-17, 22927 GROSSHANSDORF, GERMANY

SBench, digitizerNETBOX, generatorNETBOX and hybridNETBOX are registered trademarks of Spectrum Instrumentation GmbH.
Microsoft, Visual C++, Windows, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10, Windows 11 and Windows Server are trademarks/registered trademarks of Microsoft Corporation.
LabVIEW, DASYLab, Diadem and LabWindows/CVI are trademarks/registered trademarks of National Instruments Corporation.
MATLAB is a trademark/registered trademark of The Mathworks, Inc.
Delphi and C++Builder are trademarks or registered trademarks of Embarcadero Technologies, Inc.
Keysight VEE, VEE Pro and VEE OneLab are trademarks/registered trademarks of Keysight Technologies, Inc.
FlexPro is a registered trademark of Weisang GmbH & Co. KG.
PCIe, PCI Express, PCI-X and PCI-SIG are trademarks of PCI-SIG.
PICMG and CompactPCI are trademarks of the PCI Industrial Computation Manufacturers Group.
PXI is a trademark of the PXI Systems Alliance.
LXI is a registered trademark of the LXI Consortium.
IVI is a registered trademark of the IVI Foundation.
Oracle and Java are registered trademarks of Oracle and/or its affiliates.
Python is a trademark/registered trademark of Python Software Foundation.
Julia is a trademark/registered trademark of Julia Computing, Inc.
Intel and Intel Core i3, Core i5, Core i7, Core i9 and Xeon are trademarks and/or registered trademarks of Intel Corporation.
AMD, Opteron, Sempron, Phenom, FX, Ryzen and EPYC are trademarks and/or registered trademarks of Advanced Micro Devices.
Arm is a trademark or registered trademark of Arm Limited (or its subsidiaries).
NVIDIA, CUDA, GeForce, Quadro, Tesla and Jetson are trademarks and/or registered trademarks of NVIDIA Corporation.

Table of Contents

Table of Contents 3

General Information 4
- Installation 4
  - LabVIEW Driver Installation 4
  - LabVIEW Driver Update 4
- General Information 4
  - Demo mode 4
  - Driver Structure 5
  - Not supported functions 5

Libraries 6
- Library spcm_drv_interface.llb 6
  - Overview 6
  - Library Functions 6
  - Data transfer library functions 8
- Library spcm_card_common.llb 12
  - Overview 12
  - Standard library functions 12
  - Commands 14
  - Aquisition specific library functions 15
  - Replay specific library functions 17
- Library spcm_card_m2p.llb 18
  - Overview 18
  - Functions for all M2p cards 18
  - Functions for all M2p analog (input and output) cards 19
  - Functions for all M2p AI (analog input) cards 20
  - Functions for all M2p AO (analog output) cards 22

Examples 24
- General structure of the Spectrum LabVIEW examples 24
  - Initialization 24
  - GUI Event Loop 24
  - State Machine 25
- Examples for M2p cards and NETBOX products 26
  - Analog acquisition Examples 26
  - Analog generation/replay Examples 27
- Error Codes 28

General Information

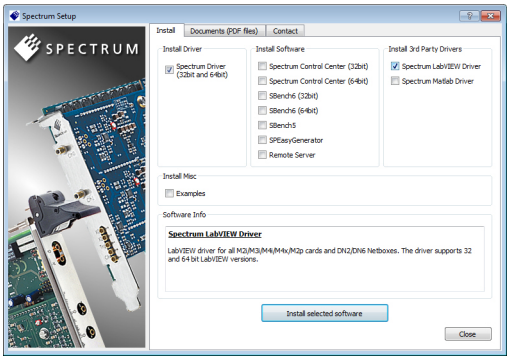
This driver is suitable for all spcm cards of the M4i, M4x, M2p and M5i series as well as the related digitizerNETBOX, generatorNETBOX and hybridNETBOX products from Spectrum. The driver supports all LabVIEW versions starting with LabVIEW 2015. The Spectrum LabVIEW driver supports Windows (32bit and 64bit) operating systems only, LabVIEW for Linux and LabVIEW RT are not supported. Please follow the instructions carefully to ensure the driver is properly installed on your system.

This manual only focusses on the LabVIEW specific parts of the driver interface. For detailed information of all the hardware features and more general information about the Spectrum Software API, please consult the hardware manual for your specific device.



Installation

LabVIEW Driver Installation



Please follow these steps when installing the LabVIEW driver:

- Install the card(s) into the system as shown in the hardware manual
- Install the standard Windows driver as shown in the hardware manual
- Install the LabVIEW driver as explained below

The LabVIEW driver is delivered as a self extracting archive. You'll find the current driver on the USB-Stick delivered with the card. Please follow the USB-Stick menu to „Software Installation" -> „Spectrum LabVIEW driver" as shown on the right side.

It is also possible to install the LabVIEW driver manually selecting the install file with the Windows explorer. Please select the path:

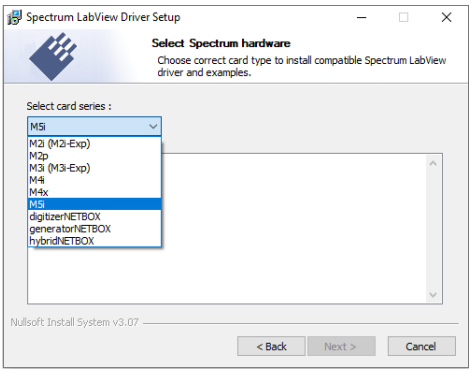
<USB-Stick>:\Install\win\spcm_drv_labview_install.exe

and execute the installer file. The installer will guide you through the installation routine step by step.

The latest driver can also be installed from the Spectrum homepage at https://spectrum-instrumentation.com/en/downloads/drivers

Please store the downloaded installer *.exe file somewhere on your system and start it from this location.

During the installation routine you will need to select which type of LabVIEW is installed on your computer (either a 32 bit or 64 bit version) and for what Spectrum products you want the examples to be installed for.



The LabVIEW driver files are installed per default in the user directory within the „my documents" folder as an extra directory:

- 32 bit LabVIEW: C:\Users\<WINDOWS_USERNAME>\Spectrum GmbH\SpcmLabVIEWDriver32
- 64 bit LabVIEW: C:\Users\<WINDOWS_USERNAME>\Spectrum GmbH\SpcmLabVIEWDriver64

When moving the files please make sure to move the complete directory with all sub-directories as the driver consists of several examples and libraries that are used together with the examples.

LabVIEW Driver Update

As the LabVIEW driver also uses the standard Windows drivers as a base, any updates on these drivers will improve the system and any changes are available under LabVIEW immediately. Updating the LabVIEW driver can simply be done by installation of the new LabVIEW driver archive.

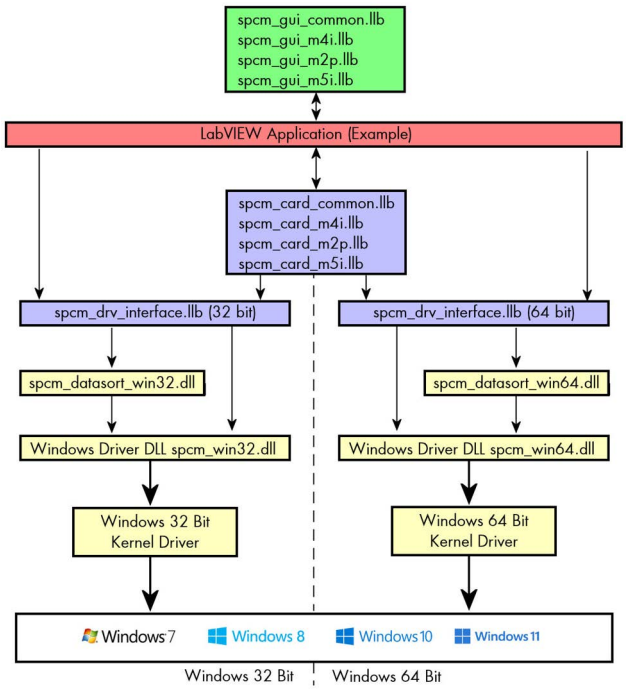
General Information

Demo mode

For demo purposes, the LabView driver can be tested with demo Spectrum cards installed on your system. More information on setting up a demo card can be found in the Hardware Manual. Please follow the steps in the hardware manual to see how you insert a simulated demo card into your system. Please keep in mind that the generated data is only simulated. The simulation and calculation of demo data takes more time than just transferring data from hardware to the PC. Therefore the performance of the system is worse when using demo cards.

Driver Structure

The driver itself consists of an driver interface LabVIEW library in either 32 bit or 64 bit version as well as some card related setup libraries „spcm_card...“ one with common functions and one for each supported card family like e.g. M2p (shown in blue). Additionally one special data handling DLLs is provided (spcm_datasort_win32.dll or spcm_datasort_win64.dll) to keep compute intensive tasks outside the LabVIEW environment (shown in yellow). All hardware accesses are routed through the standard Windows drivers and using the standard Windows kernel driver.



In addition to these libraries handling the access fro/to the hardware, the examples also contain a set of purely GUI related libraries, again seperated into a common part and card specific one (shown in green).

Access of the cards can also be done by just using the direct driver interface spcm_drv_interface.llb but using the more comfortable spcm_card_xxx.llb as shown in the examples is the more convenient way.

The components of the Spectrum LabVIEW driver are:

spcm_win32.dll / spcm_win64.dll

This is the standard Windows driver as it is installed along with the kernel driver when the new hardware is detected in the system for the first time. The Windows driver can be updated from the Spectrum website at any time under www.spectrum-instrumentation.com. This driver is used by all software that will access the cards.
The driver library is available as 32 bit version (spcm_win32.dll) and 64 bit version (spcm_win64.dll).

spcm_datasort_win32.dll / spcm_datasort_win64.dll

This is a special helper DLL that is used by several Spectrum drivers for third-party products like LabVIEW or MATLAB. It handles the data access and offers some additional functions to sort data and allows also to re-calculate RAW data samples to true voltage values. This library also handles the FIFO mode and holds the application data buffer when FIFO mode is used. This DLL is also updated with the regular Windows driver updates.

spcm_drv_interface.llb

This LabVIEW library implements the complete driver interface between LabVIEW and the DLL. It mainly handles the driver handle and the error code and calls the different driver function inside the DLLs. The installer will automatically select the matching version for either 32 bit or 64 bit systems.

spcm_card_common.llb

This is an additional LabVIEW library that uses the functions of the driver interface spcm_drv_interface.llb and groups functions that contain together. The herein included VIs are more complex and offer an easy way to get started. All the spcm_card_common.llb VIs are explained in greater detail later on. This library contains common VIs which will work with all of the supported Spectrum devices independent of the card family.

spcm_card_m4i.llb / spcm_card_m2p.llb / spcm_card_m5i.llb

In addition to the „spcm_card_common.llb“ there is also one card specific a set of VIs provided, that contains family (M4i, M2p, M5i) specific device functions. All of included card specific VIs are also explained in greater detail later on.

spcm_tools.llb

This library offers some simple helper functions to convert hardware details to readable strings like version or data conversion. Feel free to use these tools or to implement your own ones.

Not supported functions

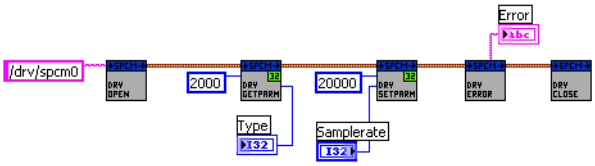
The spcm_card_common.llb and the card spcm_card_specific.llb libraries do not cover some special modes of some specific cards. These can always be directly accessed using the driver functions that are located in the spcm_drv_interface.llb library.

Libraries

Library spcm_drv_interface.llb

Overview

All library functions get a cluster containing the driver handle and the current error code. The function is only executed if the error code is zero. This allows easy error routing without the need to check for driver errors after each call. An example is shown below:



On the left one sees the open function generating the cluster that is routed through all other driver calls until it stops in the close function.

In this example we open the driver, read out the card type (shown in the digital indicator „Type") and try to set the sampling rate from the digital control „Samplerate". The sampling rate register number is found in the hardware manual, it is „20000".

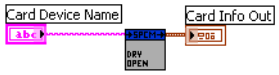
After these two function calls we check for the driver error and display the error message in the string indicator „Error".

Library Functions

The following library functions are available inside the library

spcm_hOpen.vi

Calls the spcm_hOpen function of the driver. The open function tries to open the driver handle. It will return a valid card information cluster containing the card handle and the error code. This card information cluster is routed through all VIs of this library. The function can open real cards as well as demo cards with no difference calls.



Card Device Name	the device name to open. Under windows it can be any name finishing by a number giving the index of the card to open.
Card Info Out	the generated card information cluster. It contains the card handle and the error information. If the open function succeeded the error information will be zero.

spcm_vClose.vi

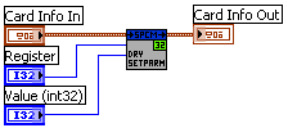
Calls the spcm_vClose function of the driver. The close function closes the card handle allowing further use of this card by other software. If the close function isn't called the card will be locked preventing any other software from accessing this card. The close function is automatically called when the DLL is unloaded. LabVIEW will unload the DLL when closing.



Card Info In	a valid card information cluster containing a valid card handle

spcm_dwSetParam_i32.vi

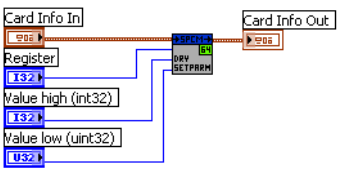
Calls the spcm_dwSetParam_i32 function of the driver. The function will set a software register with a 32 bit integer value. Please use the spcm_dwSetParam_i64m function if the value of the software register exceeds the 32 bit integer range.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to write. Please have a look at the hardware manual to see the valid software registers
Value (int32)	the value to write to the software register limited to 32 bit integer
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error

spcm_dwSetParam_i64m.vi

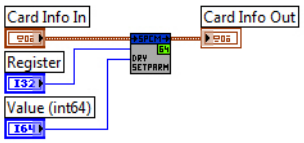
Calls the spcm_dwSetParam_i64m function of the driver. The function will set a software register with a 64 bit integer value. The value to write needs to be given in two 32 bit integer words.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to write. Please have a look at the hardware manual to see the valid software registers
Value high (int32)	the high 32 bit part of the 64 bit value to write to the software register. This part contains the sign bit
Value low (uint32)	the low 32 bit part of the 64 bit value to write. This part is unsigned.
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error

spcm_dwSetParam_i64.vi

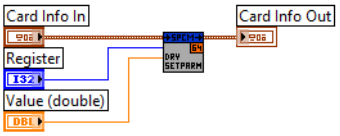
Calls the spcm_dwSetParam_i64 function of the driver. The function will set a software register with a 64 bit integer value.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to write. Please have a look at the hardware manual to see the valid software registers
Value (int64)	the value to write to the software register as a 64 bit integer
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error

spcm_dwSetParam_d64.vi

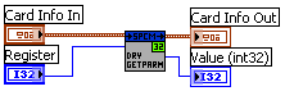
Calls the spcm_dwSetParam_d64 function of the driver. The function will set a software register with a 64 bit double value.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to write. Please have a look at the hardware manual to see the valid software registers
Value (double)	the value to write to the software register as a 64 bit integer
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error

spcm_dwGetParam_i32.vi

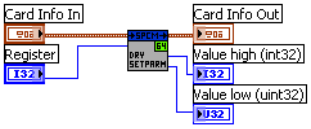
Calls the spcm_dwGetParam_i32 function of the driver. The VI reads a software register with up to 32 bit integer values. If the value exceeds the 32 bit integer range one is requested to use the spcm_dwGetParam_i64m.vi. Using the 32 bit function with a value exceeding the range will result in an error generated.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to read. Please have a look at the hardware manual to see the valid software registers
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Value (inbt32)	the current value of the software register limited to 32 bit integer

spcm_dwGetParam_i64m.vi

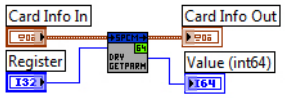
Calls the spcm_dwGetParam_i64m function of the driver. The VI reads a software register with 64 bit integer values. The value is split up in two parts and returned as two 32 bit integer values.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to read. Please have a look at the hardware manual to see the valid software registers
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Value high (int32)	the high 32 bit part of the 64 bit value that is read from the software register. This part contains the sign bit
Value low (uint32)	the low 32 bit part of the 64 bit value that is read. This part is unsigned

spcm_dwGetParam_i64.vi

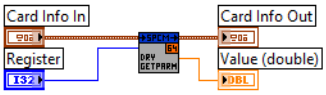
Calls the spcm_dwGetParam_i64 function of the driver. The VI reads a software register with 64 bit integer values.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to read. Please have a look at the hardware manual to see the valid software registers
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Value (int64)	the current value of the software register as a 64 bit integer

spcm_dwGetParam_d64.vi

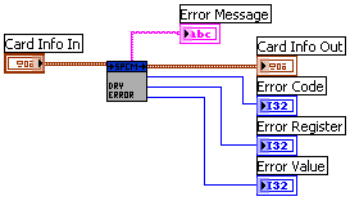
Calls the spcm_dwGetParam_d64 function of the driver. The VI reads a software register with 64 bit double values.



Card Info In	a valid card information cluster containing a valid card handle
Register	the value of the software register to read. Please have a look at the hardware manual to see the valid software registers
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Value (double)	the current value of the software register as a 64 bit double

spcm_dwGetErrorInfo.vi

Calls the spcm_dwGetErrorInfo function of the driver. The function checks for an error code and reads out all error information and the error message if an error has occurred.



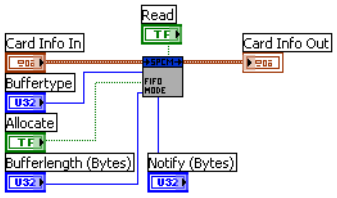
Card Info In	a valid card information cluster containing a valid card handle
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Error Message	the error message from the driver. This error message will help to examine which part of the setup was wrong
Error Code	the error code from the driver. If no error occurred this value is zero
Error Register	the register that generates the error. Please see the hardware manual for a cross reference list of the software registers
Error Value	the value that was written when the error occurred.

Data transfer library functions

The following functions are used for data transfer and FIFO mode control. These functions are located inside the helper DLL spcm_datasort_win32.dll.

dwSetupFIFOMode.vi

This VI handles the FIFO mode of the card and all transfers for timestamps and ABA data. Before starting FIFO transfer one has to allocate a FIFO buffer calling this setup function with the allocate flag set. After finishing the FIFO transfer a second call with the allocate flag cleared will delete the FIFO buffer again. Data can be accessed with the functions explained further below.

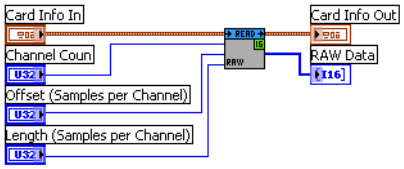


Card Info In	a valid card information cluster containing a valid card handle
Buffertype	the type of FIFO buffer to allocate, a 0 stands for data, a 1 for timestamps and a 2 for slow ABA data
Allocate	allocates the FIFO buffer if true and deletes the FIFO buffer if false
Bufferlength (Bytes)	the length of the FIFO buffer in bytes. Be sure to check the samples format to do the correct calculations on this value
Notify (Bytes)	the notify length in bytes. Every time after this number of bytes have been transferred an interrupt is generated and the user program is informed that new data is available. This value must be a multiple of 4Ki (4096). Please see the hardware manual for further information on the notify size
Read	the flag defines the direction of the following FIFO transfer
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error

dwDataRead_raw16.vi

This VI reads the data from the card in raw format for all cards that have 16 bit wide samples (analog resolution > 8 bit) or digital cards with at least 16 digital channels. Using this function is the fasted way to get data into LabVIEW. Data is unsorted and in no way converted. Please check the hardware manual to see the data ordering in the RAW buffer.

This VI can be used with FIFO mode as well as with standard mode. In FIFO mode it will read out the next free block of data, in standard mode it will read some data directly from the onboard memory.



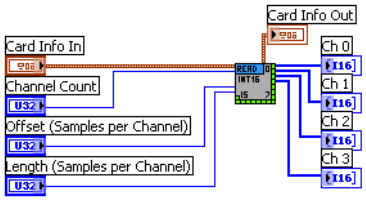
Card Info In	a valid card information cluster containing a valid card handle
Channel Count	the number of channels to be read. This value must match the number of channels that have been acquired!
Offset (Samples/Ch)	the offset from where the reading should start (standard mode). Offset is given in samples per channel, not in bytes
Length (Samples/Ch)	the length of the data to be read starting from offset (standard mode) or from the current buffer position (FIFO mode). The length value is given in samples per channel and must not exceed the previously acquired data
Card Info Out	a copy of the card information cluster input containing an error code if the dll function has returned with an error
RAW Data	An array containing the raw and unsorted data as 16 bit integer values.

dwDataRead_raw8.vi

This VI does exactly the same as the above described but returning 8 bit wide raw data instead of 16 bit. Use this function for all analog cards with 8 bit resolution and digital cards with 8 channels only activated.

dwDataRead_i16.vi

The DataRead function reads data, sorts them and returns up to 16 arrays of data (only 4 shown in the picture on the right). Each array contains data of one analog channel or a bundle of 16 digital channels and can be directly used for display and further calculations.

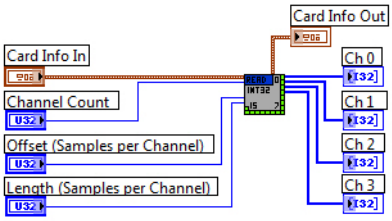


Data is stored as 16 bit integer values independent of the original data format. For 8 bit cards this means that memory storage space is doubled! Each 8 bit sample will be converted to 16 bit integer value.

Card Info In	a valid card information cluster containing a valid card handle
Channel Count	the number of channels to be read. This number must be equal to the number of installed channels on the card. Channels that are not acquired due to a different channel enable mask will be left empty
Offset (Samples/Ch)	the offset from where the reading should start (standard mode). Offset is given in samples per channel, not in bytes
Length (Samples/Ch)	the length of the data to be read starting from offset (standard mode) or from the current buffer position (FIFO mode). The length value is given in samples per channel and must not exceed the previously acquired data
Card Info Out	a copy of the card information cluster input containing an error code if the dll function has returned with an error
Ch0, Ch1,... Ch15	arrays containing the sorted data for one channel

dwDataRead_i32.vi

The DataRead function reads data, sorts them and returns up to 16 arrays of data (only 4 shown in the picture on the right). Each array contains data of one analog channel and can be directly used for display and further calculations.

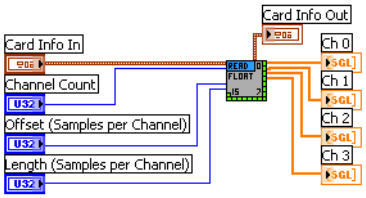


Data is stored as 32 bit integer values independent of the original ADC data format. The 32 bit data format is used for the M4i/M4x cards and digitizerNETBOXes for the block average (requires signal processing firmware option Block Average to be installed on the card) and boxcar average modes.

Card Info In	a valid card information cluster containing a valid card handle
Channel Count	the number of channels to be read. This number must be equal to the number of installed channels on the card. Channels that are not acquired due to a different channel enable mask will be left empty
Offset (Samples/Ch)	the offset from where the reading should start (standard mode). Offset is given in samples per channel, not in bytes
Length (Samples/Ch)	the length of the data to be read starting from offset (standard mode) or from the current buffer position (FIFO mode). The length value is given in samples per channel and must not exceed the previously acquired data
Card Info Out	a copy of the card information cluster input containing an error code if the dll function has returned with an error
Ch0, Ch1,... Ch15	arrays containing the sorted data for one channel

dwDataRead_float.vi

The DataRead function reads data, sorts them, recalculates them to voltage and returns up to 16 arrays of data (only 4 shown in the picture on the right). Each array contains data of one analog channel and can be directly used for display and further calculations.



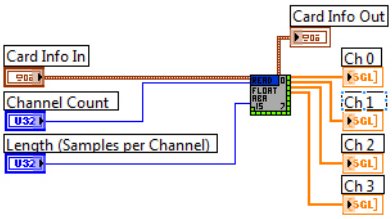
Data is stored as float values with single precision. The sorting functions recalculates the raw integer data to a true voltage level taking the programmed input range and also the programmed offset into account.

Please keep in mind that single values have 4 bytes for each sample. Acquiring 4 channels of 8 bit data with 10 MSamples of memory per each channel would result in a PC memory usage of 4 channels * 10 MSamples * 4 bytes = 160 MBytes when using this sorting function.

Card Info In	a valid card information cluster containing a valid card handle
Channel Count	the number of channels to be read. This number must be equal to the number of installed channels on the card. Channels that are not acquired due to a different channel enable mask will be left empty
Offset (Samples/Ch)	the offset from where the reading should start (standard mode). Offset is given in samples per channel, not in bytes
Length (Samples/Ch)	the length of the data to be read starting from offset (standard mode) or from the current buffer position (FIFO mode). The length value is given in samples per channel and must not exceed the previously acquired data
Card Info Out	a copy of the card information cluster input containing an error code if the dll function has returned with an error
Ch0, Ch1,... Ch15	arrays containing the sorted data for one channel, data format is single precision float containing the real voltage levels of the inputs

dwABARead_float.vi

The ABARead function reads the ABA data samples, sorts them, recalculates them to voltage and returns up to 16 arrays of data (only 4 shown in the picture on the right). Each array contains data of one channel and can be directly used for display and further calculations. Data is stored as float values with single precision. The sorting functions recalculates the raw integer data to a true voltage level taking the programmed input range and also the programmed offset into account.

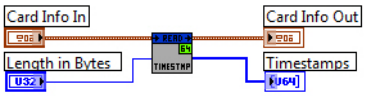


Please keep in mind that single values have 4 bytes for each sample.

Card Info In	a valid card information cluster containing a valid card handle
Channel Count	the number of channels to be read. This number must be equal to the number of installed channels on the card. Channels that are not acquired due to a different channel enable mask will be left empty
Length (Samples/Ch)	the length of the data to be read starting from offset (standard mode) or from the current buffer position (FIFO mode). The length value is given in samples per channel and must not exceed the previously acquired data
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Ch0, Ch1,... Ch15	arrays containing the sorted data for one channel

dwTimestampsRead_64.vi

The TimestampsRead function reads the timestamp data. Each timestamp is 128 bit long and mapped to two consecutive 64 bit (8 bytes) values. Please check the hardware manual for more information about the timestamp data format.

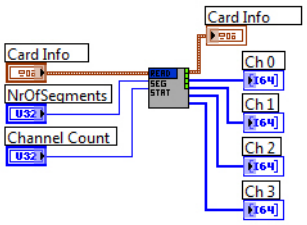


Please keep in mind that single values have 4 bytes for each sample.

Card Info In	a valid card information cluster containing a valid card handle
Length (Bytes)	the length of the data to be read. The length value is given in bytes and must not exceed the previously acquired data
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Timestamps	contains the timestamp data as an array of 64 bit values. Two consecutive 64 bit values are representing one 128 bit timestamp value

dwSegmentStatisticRead.vi (only M4i, M4x and their digitizerNETBOX counterparts)

The SegmentStatisticRead function reads the block statistic data and returns up to 4 arrays of data. Each array contains data of one channel. Six consecutive values of the 64 bit array are representing the statistic values for one segment.



Please also consult the hardware manual for more details on the Segment Statistic mode.

The following shows the arrangement of all statistic values for 2 segments of channel 0:

Requires the Signal Processing Firmware option: Block Statistic (M4i.xxxx-spstat) to be installed on the card.

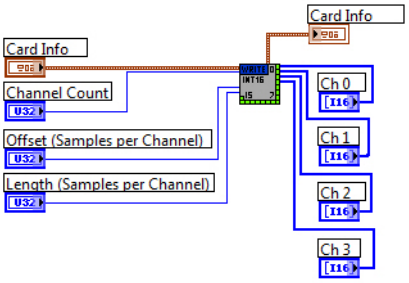
Segment1:
Ch0[0] : Average Value, Ch0[1]: Min Value, Ch0[2]: Max Value, Ch0[3]: MinPos, Ch0[4]: MaxPos, Ch0[5]: Timestamp

Segment2:
Ch0[6] : Average Value, Ch0[7]: Min Value, Ch0[8]: Max Value, Ch0[9]: MinPos, Ch0[10]: MaxPos, Ch0[11]: Timestamp

Card Info In	a valid card information cluster containing a valid card handle
NrOfSegments	the number of segments
Channel Count	the number of channels to be read. This number must be equal to the number of installed channels on the card. Channels that are not acquired due to a different channel enable mask will be left empty
Card Info Out	a copy of the card information cluster input containing an error code if the DLL function has returned with an error
Ch0, Ch1, Ch2 Ch3	arrays containing the statistic data for one channel

dwDataWrite_i16.vi

The DataWrite function writes data given as sorted arrays of int16 channels. Each channel is either an analog channel of up to 16 bit width or a bundle of up to 16 digital channels (2 bytes). Data is multiplexed inside the driver and written to hardware afterwards.



Card Info In	a valid card information cluster containing a valid card handle
Channel Count	the number of channels to be read. This number must be equal to the number of installed channels on the card. Channels that are not written due to a different channel enable mask will be left empty
Offset (Samples/Ch)	the offset where the writing should start (standard mode). Offset is given in samples per channel, not in bytes
Length (Samples/Ch)	the length of the data to be written starting from offset (standard mode) or from the current buffer position (FIFO mode). The length value is given in samples per channel and must not exceed the available amount of empty data space
Card Info Out	a copy of the card information cluster input containing an error code if the dll function has returned with an error
Ch0, Ch1,... Ch15	arrays containing the sorted data for one channel each

Library spcm_card_common.llb

Overview

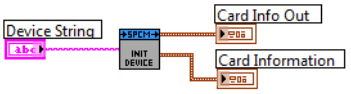
The spcm_card_common.llb library is the library for accessing the Spectrum M4i/M4x/M2p/M5i cards and digitizerNETBOX/generatorNETBOX/hybridNETBOX products. It contains the functions that are common for all of these products. All VIs route the standard card information shown on the right containing the card handle and the current error code. All VIs can simply be placed one after the other as none of these VIs execute their function if an error code is set.



Standard library functions

init device.vi

This VI is the main entrance point for the card. It must be called first to get a valid card handle. The VI tries to open the card that is given with the index and if successful it reads out some standard information from the card shown below as the card information cluster.



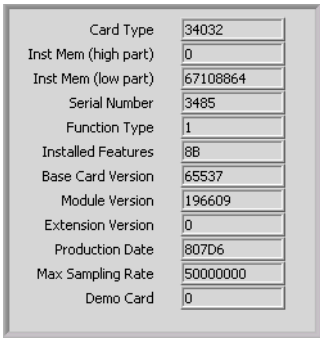
Each card can only be opened by one software at the time. Multiple calls of this initialization function with different index values will open multiple cards. Multiple calls with the same index value will result in an error as the card is opened and locked with the first call.

This function can open real cards as well as demo cards.

Device String	Pass a valid device string to open the device. A device string for a single card with index 0 is „/dev/spcm0". To open a remote instrument such as a digitizerNETBOX, generatorNETBOX, hybridNETBOX or a PC running the remote server option, use the VISA string to open the device. Please check the hardware manual for more information. An easy way to create a valid device string is to use the „select device.vi" described below.
Card Info Out	A filled card info cluster that is routed through all the other functions. If initialization failed the error code will show an initialization error. The card information cluster is shown below in the overview.
Card Information	A filled card information cluster containing all details that are common for all cards

Card information cluster

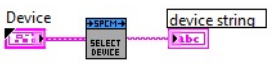
The cluster contains all common information for Spectrums M2i/M3i/M4i/M4x/M2p/M5i cards, as well as digitizerNETBOX and generatorNETBOX products. The information can be used to show card details in the software or to check the correct type or version.



Card Type	the type of card found at that position. Card types are listed in the hardware manual. You may use the translation function in the spcm_tools.llb library to show a real name for the card type
Inst Mem (high + low)	installed on-board memory in bytes, in the example the card has a memory of 64 MiBytes installed
Serial Number	serial number of the card. The serial number is an unique identifier
Function Type	the card function type (like analog input, digital i/o), details can be found in the hardware manual, in our example the card is an analog input card (1)
Installed Features	shows all installed features on the card. The features are returned as a bit-mask, each activated bit stands for one feature installed. In our example bit 4, 3, 1 and 0 are set meaning that feature ABA mode, Timestamp, Gated Sampling and Multiple Recording is installed on the card. All feature codes are explained in the hardware manual
Base card version	the version of the base card split in major and minor version. Please use the translation function from spcm_tools.llb to have a correct version display
Module version	version of the used front-end module, same format as above
Extension version	version of the extension module if one is installed, same format as above
Production date	the production week of the card, the lower 16 bit contain the year, the upper 16 bit the week. Please use the translation function from spcm_tools.llb to have the date printed in correct format
Max Sampling Rate	the maximum sampling rate of the card in hertz. In our example it is 50 MS/s (50000000 Hz). This is the absolute maximum sampling rate that may not be available with all channel combinations!
Demo Card	a simple flag indicating whether this is a virtual demo card or a real card (zero)

select device.vi

This VI builds a device string from the selection of the device cluster.



Devicethe cluster with the device selection as explained below

Device String	outputs a device string depending on your device selection

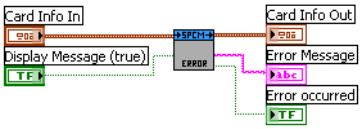
device cluster



Select Device	select the device to open (either a local card or a remote instrument)
Index	selects the index of the device in the system
IP Address	insert the IP address to your remote device here (for remote devices only)

error check and message.vi

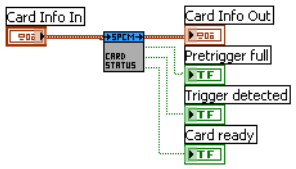
This VI is used to check the card info for an error and to display an error message if requested. To keep programming simple the VI also gives an error flag that can be directly used for case structures



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with cleared error information
Display Message	the flag selects whether the function should display an error message in case that an error occurred. As default this flag is true
Error Message	the error message string that can be used for own error display routines. Can be ignored if the error message is displayed by the VI itself
Error occurred	A flag indicating the an error has been found, error code is not zero. Can be directly used to drive case structures

read card status.vi

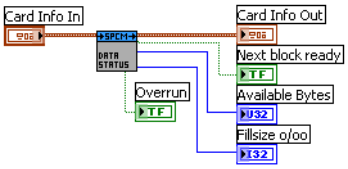
The VI reads the current card status and returns some flags indicating the status. The flags can be directly used to drive case structures or to end while loops.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Pretrigger full	acquistion cards only: the pretrigger area has been filled once, card is armed now and can detect trigger events
Trigger detected	a trigger event has been detected
Card ready	the acquisition/generation of data has been finished

read data status.vi

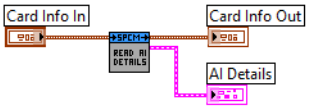
The VI reads the current status of the data transfer. This function is used together with the FIFO mode and controls the transfer and the current transfer status.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Next block ready	is true if a new block of data is ready. That means at least the programmed number of bytes are ready that have been programmed with the dwSetupFIFOMode call as notify size.
Available Bytes	returns the number of bytes that are available for the user and for the copy function
Fillsize o/oo	Gives the current fill size of the hardware FIFO in 1/1000

read AI details.vi

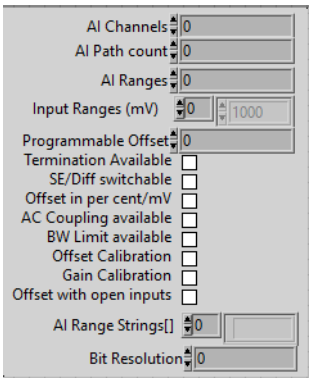
This VI reads out all analog input details from the card. These details are used throughout our examples to setup the analog input clusters according to the specific card that is installed in the system. The VI returns two complete sets of information, one for each input path.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
AI Details	a cluster with complete details of the analog input as described below

Cluster AI details

This cluster is returned by the „read AI details.vi" and contains all information on the analog inputs. All these details are read from the driver. The cluster is mainly used to keep the examples and the programs universal as the analog inputs may differ from card to card in the number of input ranges, the availability of certain features or the offset programming mode.



AI Channels	the number of analog input channels
AI Path count	the number of different input paths per channel
AI Ranges	the number of ranges for each channel for this input path. This is normally fixed for one card series but can differ if special options are ordered. The input ranges are therefore stored in the on-board eeprom and read out with this value and the array just following next in the description
Input Ranges (mV)	an array containing all input ranges as mV values that are available on your card. A 1000 as shown in the example means an input range of ±1000 mV
Termination Available	if true each analog input has a software programmable 50 ohm termination available
SE/Diff switchable	if true each analog input can be changed from single-ended to differential by software command
Offset	in per cent/mV if true the offset is programmed in mV absolute, if false the offset is programmed in per cent of the input range
AC	Coupling available if true the input can be programmed to be AC or DC coupled
BW	Limit available if true the input has a software selectable bandwidth limit (anti-aliasing filter)
Offset calibration	if true the card has a complete on-board automatic offset calibration
Gain calibration	if true the card has a complete on-board automatic gain calibration
Offset with open inputs	if true the card has an on-board automatic offset calibration that needs all signals to be disconnected from the inputs for doing the offset calibration
AI Range Strings	contains a number of input range strings that can be directly used to fill the ring control of the analog input setup cluster as shown in the example further below
Bit Resolution	contains the analog resolution of the ADC

read AO details.vi

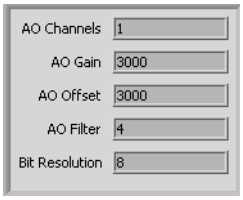
This VI reads out all analog output details from the card. These details are used throughout our examples to setup the analog output clusters according to the specific card that is installed in the system.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
AO Details	a cluster with complete details of the analog outputs as described below

Cluster AO details

This cluster is returned by the „read AO deatils.vi" and contains all information on the analog outputs. All these details are read from the driver. The cluster is mainly used to keep the examples and the programs universal as the analog outputs may differ from card to card .

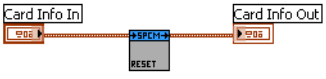


AO Channels	the number of analog output channels (in this example 1 channel)
AI Gain	the maximum bipolar output gain in mV. In our example the card can have up to +/- 3000 mV of output swing.
AO Offset	the maximum bipolar programmable analog offset of each channel in mV
AO Filter	the number of programmable output filters. Please have a look in the hardware manual to see which edge frequencies each filter has on your specific card.
Bit Resolution	contains the analog resolution of the DAC, in our example the DAC has 8 bit resolution

Commands

cmd reset

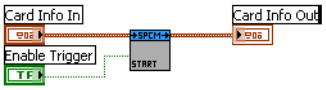
Performs a hardware and software reset of the card



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function

cmd start

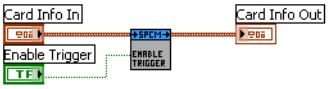
The card is started with the current setup that has to be programmed before using a valid combination of the setup VIs.



Card Info In	a valid card info cluster containing driver handle and error information
Enable Trigger	defines whether the trigger engine should be enabled directly with the start (default) or whether the trigger engine should be enabled with a separate enable trigger command
Card Info Out	a copy of the card info cluster with the error output of this function

cmd en/dis trigger

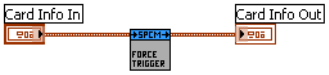
Enables or disables the trigger engine. No trigger detection is done as long as the trigger engine is disabled.



Card Info In	a valid card info cluster containing driver handle and error information
Enable Trigger	a true enables the trigger engine, a false disables it
Card Info Out	a copy of the card info cluster with the error output of this function

cmd force

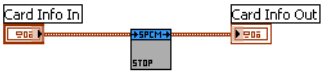
This VI sends a force trigger command that immediately triggers the card if it is waiting for a trigger event



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function

cmd stop

This VI stops the current run, the card data acquisition or generation is aborted

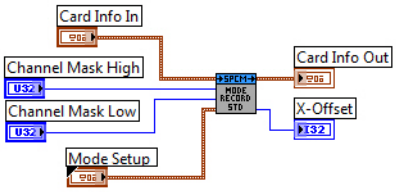


Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function

Aquisition specific library functions

setup mode record standard.vi

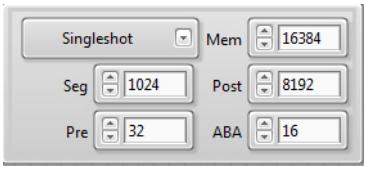
This VI programs all standard acquisition modes and programs all related settings to these modes.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Channel Mask High	upper 32 bit of channel mask for all cards that have more than 32 channels on-board (like some digital I/O cards), can be left unconnected for all cards that have less than 32 channels
Channel Mask Low	lower 32 bit of the channel mask. Each channel corresponds to one bit of the mask. This channel mask defines which channels are used for the next acquisition. Please see the hardware manual to see which restrictions are given for the channel mask selection
Mode Setup	a cluster containing the mode setup as shown below in „Cluster Record Mode Standard Setup“
X-Offset	the x-offset in samples that can be used to scale a waveform graph correctly. The offset is given in relation to the trigger event

Cluster Record Mode Standard Setup / (element_mode.ctl)

This cluster is used to feed the „setup mode record standard.vi“. It contains all standard mode setup. Depending on the selected mode some of the settings are not used.



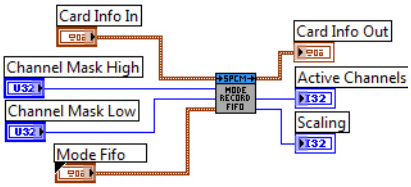
Mode	(top left) selects the standard acquisition mode.
Mem	selects the on-board memory in samples per channel that is used for the next acquisition
Seg	selects the segment size, only valid if Multiple Recording or ABA mode is selected
Post	selects the posttrigger in samples per channel. Depending on the selected mode this value has a little different meaning: *Singleshot:* number of samples to acquire after detection of trigger event *Multiple Recording:* number of samples to acquire after trigger event for each segment *ABA mode:* number of samples to acquire after trigger event for each segment *Gated Sampling:* number of samples to acquire after detection of gate-end signal

Pre	number of samples to acquire before the gate-start signal, therefore only valid if Gated Sampling is selected
ABA	ABA mode only: divides the current sampling rate to form the slow ABA clock to acquire the A-samples

The „gui update mode.vi" within the provided „spcm_gui_common.llb" provides an easy way to update the values in this cluster.

setup mode record fifo.vi

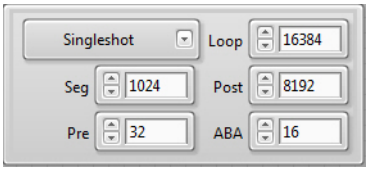
This VI programs all Fifo acquisition modes and programs all related settings to these modes.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Channel Mask High	upper 32 bit of channel mask for all cards that have more than 32 channels on-board (like some digital I/O cards), can be left unconnected for all cards that have less than 32 channels
Channel Mask Low	lower 32 bit of the channel mask. Each channel corresponds to one bit of the mask. This channel mask defines which channels are used for the next acquisition. Please see the hardware manual to see which restrictions are given for the channel mask selection
Mode Fifo	a cluster containing the mode setup as shown below in „Cluster Record Mode FifoSetup"
Scaling	returns a scaling factor to scale bytes to samples per channel. If for example 2 channels are active, each with 14 bit resolution, the scaling factor will be 4 as one channel needs 2 bytes in total to store 1 samples per channel
Active Channels	returns the number of active channels to allow easy multiplexing and de-multiplexing

Cluster Record Mode Fifo Setup / (element_mode_fifo.ctl)

This cluster contains all record FIFO mode related settings. It contains all FIFO mode setup. Depending on the selected mode some of the settings are not used.

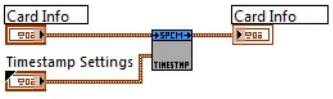


Mode	(top left) selects the FIFO acquisition mode.
Loop	selects the number of segments/gates to acquire, leave zero if the FIFO acquisition should run endlessly
Seg	selects the segment size for Multiple Recording and ABA mode, for singleshot it forms together with Loop the total data length to acquire
Post	selects the posttrigger in samples per channel. Depending on the selected mode this value has a little different meaning: *Multiple Recording:* number of samples to acquire after trigger event for each segment *ABA mode:* number of samples to acquire after trigger event for each segment *Gated Sampling:* number of samples to acquire after detection of gate-end signal
Pre	number of samples to acquire before the trigger or gate-start signal: used only with Gated Sampling or Singleshot selected
ABA	ABA mode only: divides the current sampling rate to form the slow ABA clock to acquire the A-samples

The „gui update mode.vi" within the provided „spcm_gui_common.llb" provides an easy way to update the values in this cluster.

setup timestamps.vi

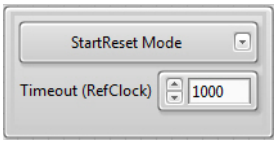
The VI programs the timestamp mode. The timestamp settings is available as a cluster that is explained next.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Timestamps Settings	contains all timestamps related settings as explained below. All these settings are programmed to the card

Cluster Timestamps / (element_timestamps.ctl)

The cluster contains the timestamps setup and is also used throughout our examples. Please have a look at the hardware documentation to see details about the timestamp mode and the different setups.

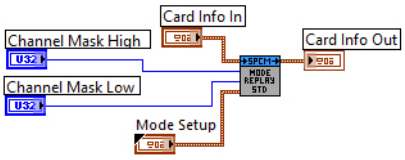


Mode	selects one of the timestamp modes
Timeout (RefClock)	sets a timeout in milliseconds to waiting for a reference clock edge. Only used for the Timestamp Refclock mode.

Replay specific library functions

setup mode replay standard.vi

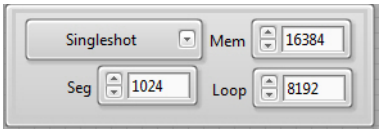
This VI programs all standard output (generation) modes and programs all related settings to these modes. Either the „setup output standard" or the „setup output FIFO" VI needs to be used in any LabVIEW program that performs output.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Channel Mask High	upper 32 bit of channel mask for all cards that have more than 32 channels on-board (like some digital I/O cards), can be left unconnected for all cards that have less than 32 channels
Channel Mask Low	lower 32 bit of the channel mask. Each channel corresponds to one bit of the mask. This channel mask defines which channels are used for the next acquisition. Please see the hardware manual to see which restrictions are given for the channel mask selection
Mode Out	a cluster containing the mode setup as shown below in „Cluster Replay Mode Standard Setup"

Cluster Replay Mode Standard Setup / (element_mode_out.ctl)

This cluster is used to feed the „setup mode record standard.vi". It contains all standard mode setup. Depending on the selected mode some of the settings are not used.

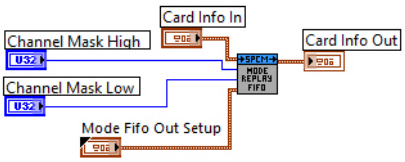


Mode	(top left) selects the standard output mode.
Mem	selects the on-board memory in samples per channel that is used for the next generation
Seg	selects the segment size, only valid if Multiple Replay mode is selected
Loop	Defines the number of loops to output. A zero stands for endless looping, a 1 for one loop until the programmed memory size is one time completely replayed. The meaning of this value differs a little depending on the selected mode: *Singleshot:* Defines the number of single shots that are performed. Each detected trigger event will generate one single shot until the loop counter expires. *Continuous:* Defines the number of loops the programmed memory is replayed after one trigger event. *Multiple Replay:* Values > 1 defines the number of segments that are replayed. Each segment will be replayed after detection of a new trigger event. *Gated Replay:* Values > 1 defines how many gate segments are replayed

The „gui update mode out.vi" within the provided „spcm_gui_common.llb" provides an easy way to update the values in this cluster.

setup mode replay fifo.vi

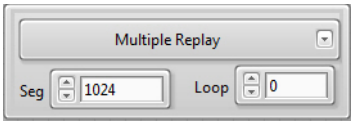
This VI programs all FIFO output modes and programs all related settings to this mode. Either the „setup output standard" or the „setup output FIFO" VI needs to be used in any LabVIEW program that performs output.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Channel Mask High	upper 32 bit of channel mask for all cards that have more than 32 channels on-board (like some digital I/O cards), can be left unconnected for all cards that have less than 32 channels
Channel Mask Low	lower 32 bit of the channel mask. Each channel corresponds to one bit of the mask. This channel mask defines which channels are used for the next generation. Please see the hardware manual to see which restrictions are given for the channel mask selection
Mode Fifo Out Setup	a cluster containing the mode setup as shown below in „Cluster Replay Mode FifoSetup"

Cluster Replay Mode FIFO Setup / (element_mode_fifo_out.ctl)

This cluster contains all output FIFO mode related settings:



Mode	(top left) selects the FIFO output mode.t
Seg	selects the segment size for Multiple Replay, for singleshot it forms together with Loop the total data length to output
Loop	selects the number of segments/gates to output, leave zero if the FIFO generation should run endlessly

The „gui update mode.vi" within the provided „spcm_gui_common.llb" provides an easy way to update the values in this cluster.

Library spcm_card_m2p.llb

Overview

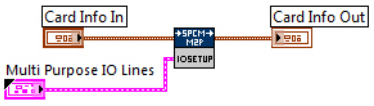
The setup of specific categories like an example the setup of clock settings is done by a corresponding control cluster (user dialogs) and a setup VI. The control cluster elements (user dialogs) are stored as control files (*.ctl). For some control clusters there are also „GUI update" VIs. A „GUI update" VI adjusts the components within the control cluster. As an example some components will be enabled or disabled depending on a specific selection.

All M2p specific „GUI update" VIs and all control cluster elements used here stored in the card specific library „spcm_gui_m2p.llb".

Functions for all M2p cards

setup M2p IO Lines.vi

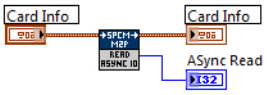
The VI defines the behavior of the multi purpose I/O lines.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Multi Purpose IO Lines	A cluster defining the behaviour

read M2p async io.vi

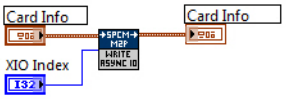
Calling this VI reads the asynchronous input lines if Multi Purpose mode is set to asynchronous input.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
ASync Read	returns the value from the asynchronous read

write M2p async io.vi

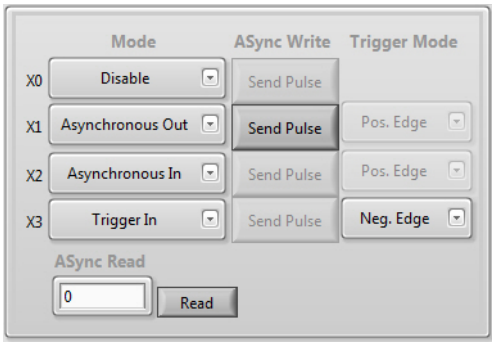
Calling this VI sends a pulse signal over the selected asynchronous output line if Multi Purpose mode is set to asynchronous output.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
XIO Index	index of the I/O line to writea pulse to asynchronously

M2p Cluster Multi Purpose IO Lines / (element_multi_purpose_io.ctl)

This cluster is used for the Setup M2p Multi Purpose I/O lines VI

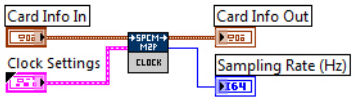


Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Multi Purpose IO Lines	A cluster defining the behavior
ASync Read	returns value from asynchronous read

Functions for all M2p analog (input and output) cards

setup M2p clock.vi

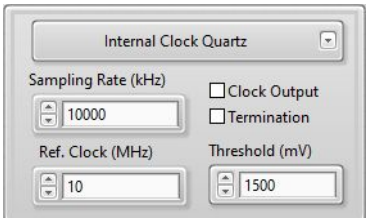
The VI programs the sampling clock and all clock related setup to the card. The clock settings are available as a cluster that is explained next.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Clock Settings	contains all clock related settings as explained below. All these settings are programmed to the card
Sampling Rate (Hz)	contains the current programmed sampling rate that is read back from the driver. This sampling rate may differ from the one that has been programmed before depending on the capabilities of the card and the clock fed as reference clock.

M2p Cluster Clock / (element_m2p_clock.ctl)

The cluster contains the complete clock setup and is also used throughout our examples. Not all of the settings are used for every clock mode. Please have a look at the hardware documentation to see details about the clock mode and the different setups.



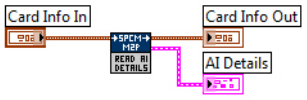
Mode (on top)	selects one of the clock generating modes. The clock mode defines which of the other settings are used and which are ignored.
Sampling rate (kHz)	defines the sampling rate for all internal clock modes in kHz (kS/s) and also for the reference clock mode. The driver set's the nearest matching sampling rate which can be read back using the current clock settings cluster described below
Reference Clock (MHz)	defines the exact frequency of the reference clock that is fed into the external clock connector. This value is only used if the reference clock mode is selected
Threshold (mV)	sets the clock threshold level. The threshold level can be set in a range of -5000 mV to +5000 mV
Clock Output	if enabled the clock connector outputs the currently used internal sampling clock
Termination	switches the 50 ohm clock termination if an external clock source has been selected (ref clock or external clock)

The „m2p gui update clock.vi" within the provided „spcm_gui_m2p.llb" provides an easy way to update the values in this cluster.

Functions for all M2p AI (analog input) cards

read M2p AI details.vi

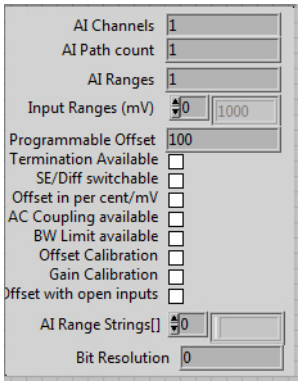
This VI reads out all analog input details from the card. These details are used throughout our examples to setup the analog input clusters according to the specific card that is installed in the system. The VI returns two complete sets of information, one for each input path.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
AI Details	a cluster with complete details of the analog inputs as described below

M2p Cluster AI details

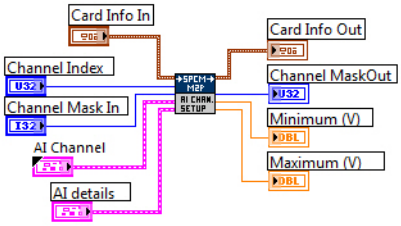
This cluster is returned by the „read M2p AI details.vi" and contains all information on the analog inputs. All these details are read from the driver. The cluster is mainly used to keep the examples and the programs universal as the analog inputs may differ from card to card in the number of input ranges, the availability of certain features or the offset programming mode.



AI Channels	the number of analog input channels (in this example 1 channel)
AI Path	count the number of different input paths per channel
AI Ranges	the number of ranges for each channel for this input path. This is normally fixed for one card series but can differ if special options are ordered. The input ranges are therefore stored in the on-board eeprom and read out with this value and the array just following next in the description
Input Ranges (mV)	an array containing all input ranges as mV values that are available on your card. A 1000 as shown in the example means an input range of ±1000 mV
Termination Available	if true each analog input has a software programmable 50 ohm termination available
SE/Diff switchable	if true each analog input can be changed from single-ended to differential by software command
Offset in per cent/mV	if true the offset is programmed in mV absolute, if false the offset is programmed in per cent of the input range
AC Coupling available	if true the input can be programmed to be AC or DC coupled
BW Limit available	if true the input has a software selectable bandwidth limit (anti aliasing filter)
Offset calibration	if true the card has a complete on-board automatic offset calibration
Gain calibration	if true the card has a complete on-board automatic gain calibration
Offset with open inputs	if true the card has an on-board automatic offset calibration that needs all signals to be disconnected from the inputs for doing the offset calibration
AI Range Strings	contains a number of input range strings that can be directly used to fill the ring control of the analog input setup cluster as shown in the example further below
Bit Resolution	contains the analog resolution of the ADC

setup M2p AI channel.vi

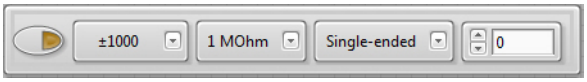
This VI performs the complete analog input setup for one channel. It therefore gets a AI setup cluster and the number of the channel to perform. To keep the setup of the channel mask easy it will also add the correct channel mask bit to the routed channel mask. After calling all analog input setups the channel mask output of the last VI contains the correct channel mask to be set.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Channel Index	the index of the channel which settings should be programmed. The channel indexing starts with zero!
Channel Mask In	In the current channel mask that will be modified by the VI if the channel is activated. The first „setup AI channel" call must be fed with a zero and all following calls need to be fed with the output of the last call
Channel Mask Out	the modified channel mask to be routed to the next call of „setup AI channel"
Input Channel Settings	the cluster with the channel setup as explained below
AI Details	the AI details cluster that were returned by the „M2p read AI details" VI. These cluster is absolutely necessary as this VI can handle all different card types and has to know which functions the card supports
Minimum, Maximum (V)	These outputs can be optionally used to scale a waveform graph. They contain the minimum and maximum value the input channel will generate as a voltage level. The calculation checks the selected input range as well as the selected user offset

M2p Cluster Input Channel Settings / (element_m2p_ai_channel.ctl)

This cluster contains all analog input channel settings and is used together with the „M2p setup AI channel" VI. It support all possible settings that an analog input channel can have. It is recommended to adjust the controls of this cluster according to the analog input details as shown in our example.

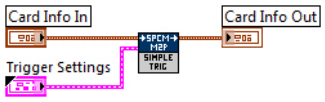


Enable	selects whether this channel should be acquired or not. This input is used by the „setup M2p AI channel" function to set up the channel mask
Range	selects the input range for the channel. It is recommended to use the „AI range strings[]" from the „M2p AI details cluster" to fill this ring element with valid setup
Termination	selects the input termination if the card supports software programmable input termination
SE/Diff	switches the input from single-ended to differential by software if the card supports this feature
Offset	programs the input offset of the channel if the card supports this feature. Depending on the used card the offset can either be a per cent value of the input range or an absolute mV value

The „m2p_gui update ai ch.vi" within the provided „spcm_gui_m2p.llb" provides an easy way to update the values in this cluster.

setup M2p AI simple trigger.vi

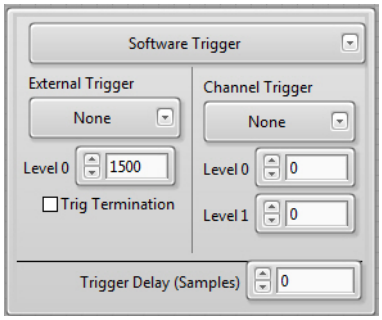
The VI is used to have a simple method for settings triggers. This VI is limited to one trigger source at the time.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Trigger Settings	the cluster containing the simple trigger settings. The cluster itself is described next

M2p Cluster Simple AI Trigger / (element_m2p_ai_trigger.ctl)

This cluster contains the simple AI trigger setup. It covers all Spectrum M2p analog input cards and therefore list's all channels that may be available with any Spectrum card. Please use only the channels that are available on your card as a trigger source as using another channel will result in an error message from the driver. Please note that besides the trigger source on top of the window and the trigger delay all other settings are only used for certain trigger modes.



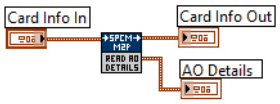
Trigger Source (top)	selects the single trigger source to be used. In our examples the software trigger is selected. External trigger selects the external trigger mode Ext0. If the trigger source is set to External Trigger, otherwise this setting is ignored
Ext0 Mode	selects the external trigger mode (ext0).
Level 0	defines the level 0 of external analog trigger (ext0) in mV
Trig Termination	switches the 50 ohm trigger termination if external trigger source (ext0) has been selected if the trigger source is set to Secondary External Trigger, otherwise this setting is ignored
Channel Trigger Mode	selects the channel trigger mode if one of the channel trigger sources has been selected
Level 0	define the trigger level 0 (upper level) as integer value. Please check the re-calculation and the valid range of the trigger level in the hardware manual
Level 1	define the trigger level 1 (lower level) as integer value. Only available for certain channel trigger modes that need two trigger levels. Please check the hardware manual for details
Trigger Delay	programs the trigger delay in samples. Is used for all trigger sources and trigger modes

The „m2p_gui update ai trig.vi" within the provided „spcm_gui_m2p.llb" provides an easy way to update the values in this cluster.

Functions for all M2p AO (analog output) cards

read M2p AO details.vi

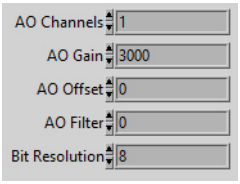
This VI reads out all analog output details from the card. These details are used throughout our examples to setup the analog output clusters according to the specific card that is installed in the system.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
AO Details	a cluster with complete details of the analog outputs as described below

M2p Cluster AO details

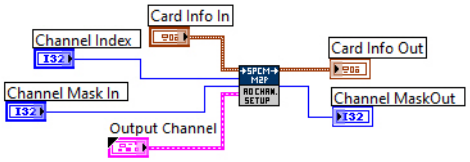
This cluster is returned by the „read M2p AO details.vi“ and contains all information on the analog outputs. All these details are read from the driver. The cluster is mainly used to keep the examples and the programs universal as the analog outputs may differ from card to card.



AO Channels	the number of analog output channels
AO Gain	the maximum bipolar output gain in mV that can be set
AO Offset	the maximum bipolar programmable analog offset of each channel in mV
AO Filter	the number of programmable output filters. Please have a look in the hardware manual to see which edge frequencies each filter has on your specific card.
Bit Resolution	contains the analog resolution of the DAC

setup M2p AO channel.vi

This VI performs the complete analog output setup for one channel. It therefore gets a AO setup cluster and the number of the channel to perform. To keep the setup of the channel mask easy it will also add the correct channel mask bit to the routed channel mask. After calling all analog output setups the channel mask output of the last VI contains the correct channel mask to be set.



Card Info	In a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Channel Index	the index of the channel which settings should be programmed. The channel indexing starts with zero!
Channel Mask In	the current channel mask that will be modified by the VI if the channel is activated. The first „setup AO channel“ call must be fed with a zero and all following calls need to be fed with the output of the last call
Channel Mask Out	the modified channel mask to be routed to the next call of „setup AO channel“
Output Channel	settings cluster with the channel setup as explained below AO Details the AO details cluster that was returned by the „read AO details“ VI. This cluster is absolutely necessary as this VI can handle all different card types and has to know which functions the card supports

M2p Cluster Output Channel Settings / (element_m2p_ao_channel.ctl)

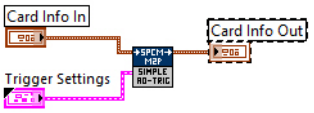
This cluster contains all analog output channel settings and is used together with the „M2p setup AO channel“ VI. It supports all possible settings that an analog output channel can have. It is recommended to adjust the controls of this cluster according to the analog output details as shown in our example.



Enable	selects whether this channel should be acquired or not. This input is used by the „M2p setup AO channel“ function to set up the channel mask
Output Mode	defines the output mode of the selected channel. Depending on the card version and the channel selection some special output modes are available. The output modes are described in detail in the hardware manual.
Gain	selects the programmed bipolar output gain as mV.
Offset	selects the programmed output offset as mV.
Filter	selects the index of the used output filter for this channel.
Stop Level	selects the stop level for this channel

setup M2p AO simple trigger.vi

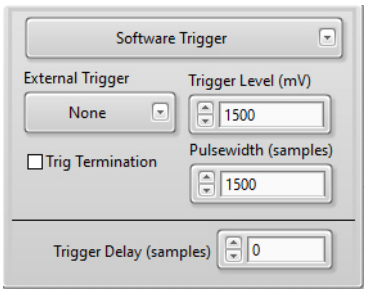
The VI is used to have a simple method for setting triggers. This VI is limited to one trigger source at the time.



Card Info In	a valid card info cluster containing driver handle and error information
Card Info Out	a copy of the card info cluster with the error output of this function
Trigger Settings	the cluster containing the simple trigger settings. The cluster itself is described next

M2p Cluster Simple AO Trigger / (element_m2p_ao_trigger.ctl)

This cluster contains the simple AO trigger setup. Please note that besides the trigger source on top of the window and the trigger delay all other settings are only used for certain trigger modes.



Trigger Source (top)	selects the single trigger source to be used. In our examples the software trigger is selected. External trigger selects the external trigger mode Ext0. If the trigger source is set to External Trigger, otherwise this setting is ignored
External Trigger	selects the external trigger mode (ext0).
Trigger Level	defines the level of external analog trigger (ext0) in mV
Trig Termination	switches the 50 ohm trigger termination if external trigger source (ext0) has been selected
Pulsewidth	sets the pulsewidth counter in samples
Trigger Delay	programs the trigger delay in samples. Is used for all trigger sources and trigger modes

Examples

This chapter gives you a brief introduction to the examples that are delivered with the Spectrum LabVIEW driver. Please keep in mind that these are only examples to show how the LabVIEW driver can be used. Although most of these examples can also be used as complete and comfortable stand-alone programs that is not their specifically intended use case. Therefore there might be some limits in the examples and some settings are not checked on a LabVIEW example level, but only on the level of the standard driver.

These examples are not tailored to the older generation M2i and M3i cards. For these older families please use their respective dedicated examples.



General structure of the Spectrum LabVIEW examples

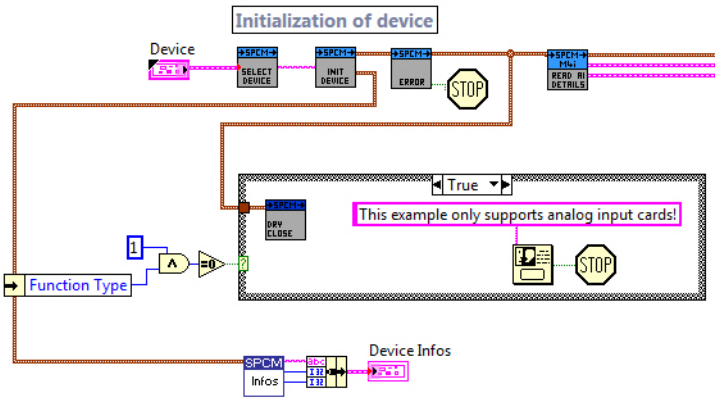
All Spectrum LabVIEW examples, staring with those for the M4i, M4x, M2p and M5i cards (and related digitizerNETBOX and generatorNETBOX products) follow the same principle of operation. Their structure is almost identical, no matter what device the specific example is targeted for and their key components are explained in this passage.

Initialization

The initialization of the device is the first step in every example.

First the device is opened and some basic information is read out.

Afterwards it is then checked, whether this specific example is compatible with the currently installed hardware. If this is not the case, an error is shown and the execution of the example is stopped.



GUI Event Loop

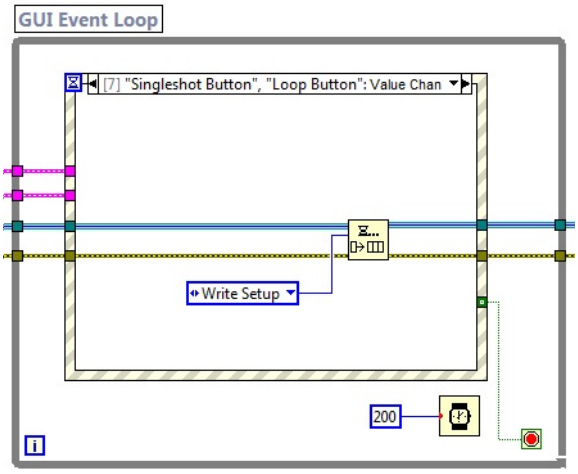
The GUI (graphical user interface) event loop detects any actions within any of the examples setup dialogs. In case that a changed value or setting is detected, the proper steps will be taken.

One of these steps is to update all the dialogs and their elements, using the VIs included in the GUI related *.llb libraries.

These updates can for example be to change the visibility or availability of certain control elements as a result of settings that have been made to other controls, settings or values.

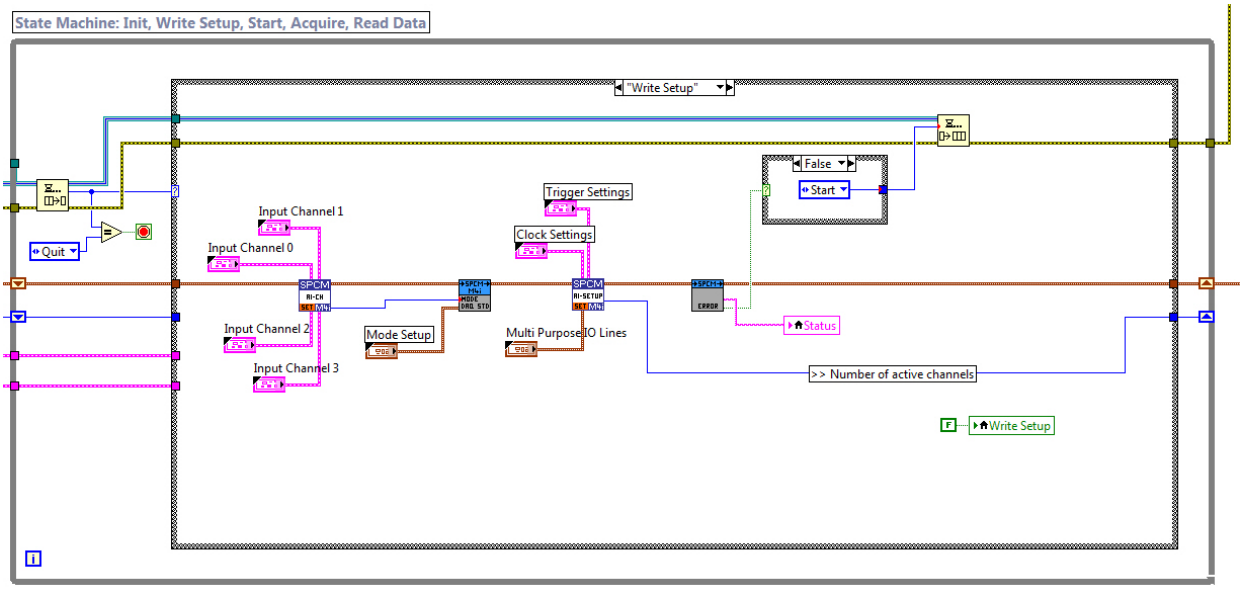
One other step is to trigger or proceed states in the State Machine (see below).

One action that would for example cause a change of states would be issued to the State Machine would be in the case that the user has pressed the acquisition start or stop button.



State Machine

The sequence of different steps/commands/actions that are required for a certain example are implemented with the help of a „State Machine“:



There are different states, such as „Write Setup“ or „Read Data“ that will be processed in a certain order to do an acquisition or generation of data.

The existing sequences within an example can be changed or extended quite easily by the user by adding additional sequence steps to this State Machine.

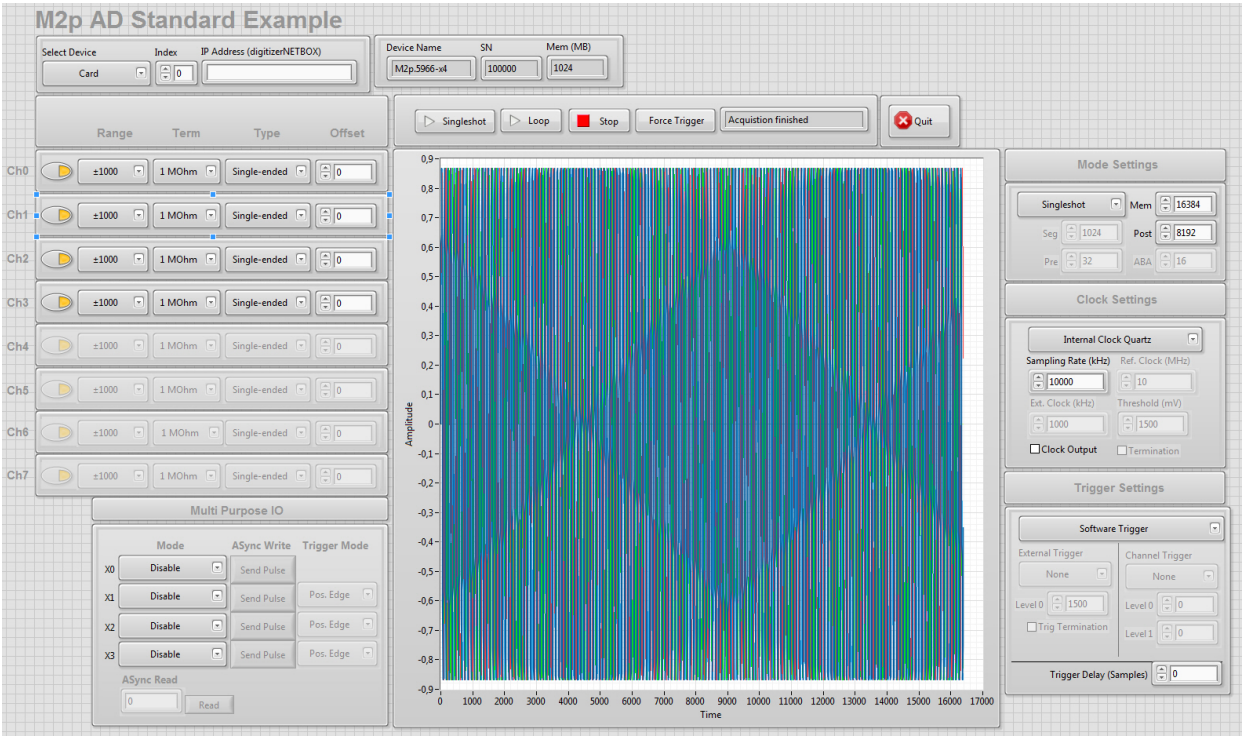
The events that cause the State Machine to proceed states are either events from within the State Machine itself, or can also be triggered by the GUI event loop as described above.

Examples for M2p cards and NETBOX products

Analog acquisition Examples

There are multiple analog acquisition examples available when installing the Spectrum LabVIEW driver. Because not every one of all the examples is compatible with or suited for every of the many Spectrum A/D cards and digitizerNETBOX products, the LabVIEW installer will only install the examples matching the device based upon your choice of device during the installation process.

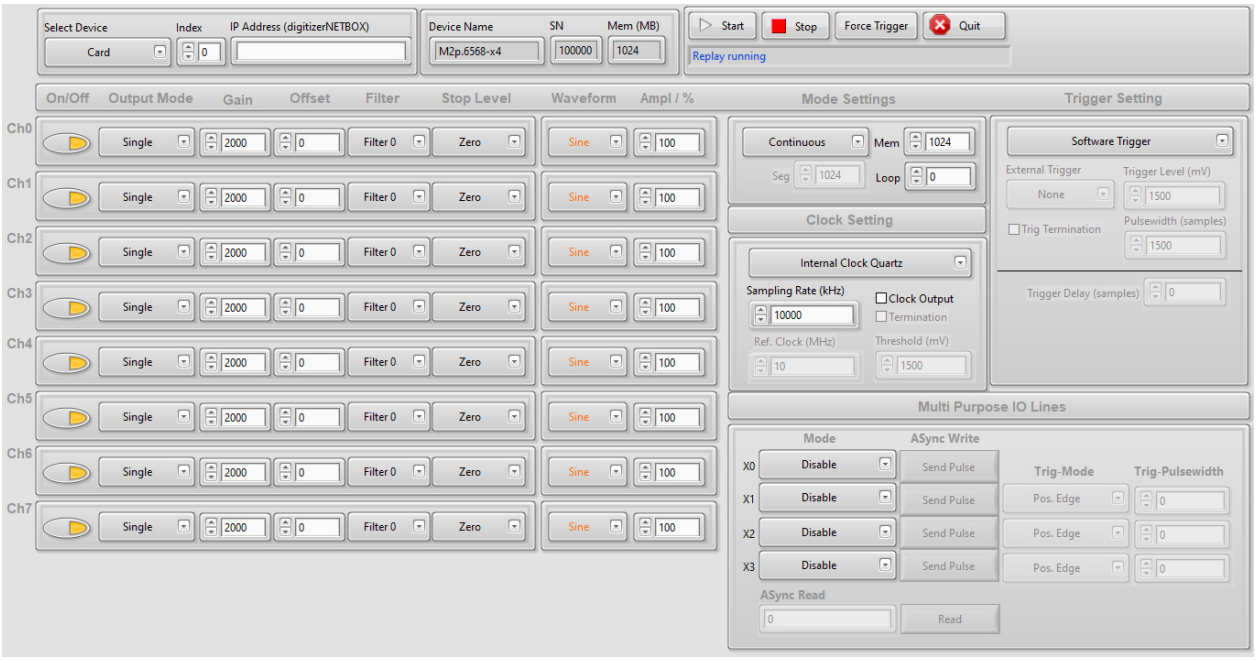
The following screenshot shows the user interface of the „M2p_AD_Std.vi“ example, that has been modeled after the typical user interface of a standalone data logger or multi-channel digital storage oscilloscope.



Analog generation/replay Examples

There are multiple analog generation/replay examples available when installing the Spectrum LabVIEW driver. Because not every one of all the examples is compatible with or suited for every of the many Spectrum D/A cards and generatorNETBOX products, the LabVIEW installer will only install the examples matching the device based upon your choice of device during the installation process.

The following screenshot shows the user interface of the „M2p_DA_Std.vi“ example, that has been modeled after the typical user interface of a standalone signal generator.



Error Codes

The following error codes could occur when a driver function has been called. Please check carefully the allowed setup for the register and change the settings to run the program.

Table 1: Spectrum API: driver error codes and error description

error name	value (hex)	value (dec.)	error description
ERR_OK	0h	0	Execution OK, no error.
ERR_INIT	1h	1	An error occurred when initializing the given card. Either the card has already been opened by another process or an hardware error occurred.
ERR_TYP	3h	3	Initialization only: The type of board is unknown. This is a critical error. Please check whether the board is correctly plugged in the slot and whether you have the latest driver version.
ERR_FNCNOTSUPPORTED	4h	4	This function is not supported by the hardware version.
ERR_BRDREMAP	5h	5	The board index re map table in the registry is wrong. Either delete this table or check it carefully for double values.
ERR_KERNELVERSION	6h	6	The version of the kernel driver is not matching the version of the DLL. Please do a complete re-installation of the hardware driver. This error normally only occurs if someone copies the driver library and the kernel driver manually.
ERR_HWDRVVERSION	7h	7	The hardware needs a newer driver version to run properly. Please install the driver that was delivered together with the card.
ERR_ADRRANGE	8h	8	One of the address ranges is disabled (fatal error), can only occur under Linux.
ERR_INVALIDHANDLE	9h	9	The used handle is not valid.
ERR_BOARDNOTFOUND	Ah	10	A card with the given name has not been found.
ERR_BOARDINUSE	Bh	11	A card with given name is already in use by another application.
ERR_EXPHW64BITADR	Ch	12	Express hardware version not able to handle 64 bit addressing -> update needed.
ERR_FWVERSION	Dh	13	Firmware versions of synchronized cards or for this driver do not match -> update needed.
ERR_SYNCPROTOCOL	Eh	14	Synchronization protocol versions of synchronized cards do not match -> update needed.
ERR_KERNEL	Fh	15	Some error occurred in the kernel driver. On Linux check output of dmesg for details.
ERR_LASTERR	10h	16	Old error waiting to be read. Please read the full error information before proceeding. The driver is locked until the error information has been read.
ERR_BOARDINUSE	11h	17	Board is already used by another application. It is not possible to use one hardware from two different programs at the same time.
ERR_ABORT	20h	32	Abort of wait function. This return value just tells that the function has been aborted from another thread. The driver library is not locked if this error occurs.
ERR_BOARDLOCKED	30h	48	The card is already in access and therefore locked by another process. It is not possible to access one card through multiple processes. Only one process can access a specific card at the time.
ERR_DEVICE_MAPPING	32h	50	The device is mapped to an invalid device. The device mapping can be accessed via the Control Center.
ERR_NETWORKSETUP	40h	64	The network setup of a digitizerNETBOX has failed.
ERR_NETWORKTRANSFER	41h	65	The network data transfer from/to a digitizerNETBOX has failed.
ERR_FWPOWERCYCLE	42h	66	Power cycle (PC off/on) is needed to update the card's firmware (a simple OS reboot is not sufficient !).
ERR_NETWORKTIMEOUT	43h	67	A network time out has occurred.
ERR_BUFFERSIZE	44h	68	The buffer size is not sufficient (too small).
ERR_RESTRICTEDACCESS	45h	69	The access to the card has been intentionally restricted.
ERR_INVALIDPARAM	46h	70	An invalid parameter has been used for a certain function.
ERR_TEMPERATURE	47h	71	The temperature of at least one of the card's sensors measures a temperature, that is too high for the hardware.
ERR_REG	100h	256	The register is not valid for this type of board.
ERR_VALUE	101h	257	The value for this register is not in a valid range. The allowed values and ranges are listed in the board specific documentation.
ERR_FEATURE	102h	258	Feature (option) is not installed on this board. It's not possible to access this feature if it's not installed.
ERR_SEQUENCE	103h	259	Command sequence is not allowed. Please check the manual carefully to see which command sequences are possible.
ERR_READABORT	104h	260	Data read is not allowed after aborting the data acquisition.
ERR_NOACCESS	105h	261	Access to this register is denied. This register is not accessible for users.
ERR_TIMEOUT	107h	263	A time out occurred while waiting for an interrupt. This error does not lock the driver.
ERR_CALLTYPE	108h	264	The access to the register is only allowed with one 64 bit access but not with the multiplexed 32 bit (high and low double word) version.
ERR_EXCEEDSINT32	109h	265	The return value is int32 but the software register exceeds the 32 bit integer range. Use double int32 or int64 accesses instead, to get correct return values.
ERR_NOWRITEALLOWED	10Ah	266	The register that should be written is a read-only register. No write accesses are allowed.
ERR_SETUP	10Bh	267	The programmed setup for the card is not valid. The error register will show you which setting generates the error message. This error is returned if the card is started or the setup is written.
ERR_CLOCKNOTLOCKED	10Ch	268	Synchronization to external clock failed: no signal connected or signal not stable. Please check external clock or try to use a different sampling clock to make the PLL locking easier.
ERR_MEMINIT	10Dh	269	On-board memory initialization error. Power cycle the PC and try another PCIe slot (if possible). In case that the error persists, please contact Spectrum support for further assistance.
ERR_POWERSUPPLY	10Eh	270	On-board power supply error. Power cycle the PC and try another PCIe slot (if possible). In case that the error persists, please contact Spectrum support for further assistance.
ERR_ADCCOMMUNICATION	10Fh	271	Communication with ADC failed. Power cycle the PC and try another PCIe slot (if possible). In case that the error persists, please contact Spectrum support for further assistance.
ERR_CHANNEL	110h	272	The channel number may not be accessed on the board: Either it is not a valid channel number or the channel is not accessible due to the current setup (e.g. Only channel 0 is accessible in interlace mode).
ERR_NOTIFYSIZE	111h	273	The notify size of the last spcm_dwDefTransfer call is not valid. The notify size must be a multiple of the page size of 4096. For data transfer it may also be a fraction of 4Ki in the range of 16, 32, 64, 128, 256, 512, 1Ki or 2Ki. For ABA and timestamp the notify size can be 2Ki as a minimum.

Table 1: Spectrum API: driver error codes and error description

error name	value (hex)	value (dec.)	error description
ERR_RUNNING	120h	288	The board is still running, this function is not available now or this register is not accessible now.
ERR_ADJUST	130h	304	Automatic card calibration has reported an error. Please check the card inputs.
ERR_PRETRIGGERLEN	140h	320	The calculated pretrigger size (resulting from the user defined posttrigger values) exceeds the allowed limit.
ERR_DIRMISMATCH	141h	321	The direction of card and memory transfer mismatch. In normal operation mode it is not possible to transfer data from PC memory to card if the card is an acquisition card nor it is possible to transfer data from card to PC memory if the card is a generation card.
ERR_POSTEXCDSEGMENT	142h	322	The posttrigger value exceeds the programmed segment size in multiple recording/ABA mode. A delay of the multiple recording segments is only possible by using the delay trigger!
ERR_SEGMENTINMEM	143h	323	Memsize is not a multiple of segment size when using Multiple Recording/Replay or ABA mode. The programmed segment size must match the programmed memory size.
ERR_MULTIPLEPW	144h	324	Multiple pulsewidth counters used but card only supports one at the time.
ERR_NOCHANNELPWOR	145h	325	The channel pulsewidth on this card can't be used together with the OR conjunction. Please use the AND conjunction of the channel trigger sources.
ERR_ANDORMASKOVRLAP	146h	326	Trigger AND mask and OR mask overlap in at least one channel. Each trigger source can only be used either in the AND mask or in the OR mask, no source can be used for both.
ERR_ANDMASKEDGE	147h	327	One channel is activated for trigger detection in the AND mask but has been programmed to a trigger mode using an edge trigger. The AND mask can only work with level trigger modes.
ERR_ORMASKLEVEL	148h	328	One channel is activated for trigger detection in the OR mask but has been programmed to a trigger mode using a level trigger. The OR mask can only work together with edge trigger modes.
ERR_EDGEPERMOD	149h	329	This card is only capable to have one programmed trigger edge for each module that is installed. It is not possible to mix different trigger edges on one module.
ERR_DOLEVELMINDIFF	14Ah	330	The minimum difference between low output level and high output level is not reached.
ERR_STARHUBENABLE	14Bh	331	The card holding the star-hub must be enabled when doing synchronization.
ERR_PATPWSMALLEDGE	14Ch	332	Combination of pattern with pulsewidth smaller and edge is not allowed.
ERR_XMODESETUP	14Dh	333	The chosen setup for (SPCM_X0_MODE .. SPCM_X19_MODE) is not valid. See hardware manual for details.
ERR_AVRG_LSA	14Eh	334	Setup for Average LSA Mode not valid. Check Threshold and Replacement values for chosen AVRGMODE.
ERR_PCICHECKSUM	203h	515	The check sum of the card information has failed. This could be a critical hardware failure. Restart the system and check the connection of the card in the slot.
ERR_MEMALLOC	205h	517	Internal memory allocation failed. Please restart the system and be sure that there is enough free memory.
ERR_EEPROMLOAD	206h	518	Time out occurred while loading information from the on-board EEProm. This could be a critical hardware failure. Please restart the system and check the PCI connector.
ERR_CARDNOSUPPORT	207h	519	The card that has been found in the system seems to be a valid Spectrum card of a type that is supported by the driver but the driver did not find this special type internally. Please get the latest driver from www.spectrum-instrumentation.com and install this one.
ERR_CONFIGACCESS	208h	520	Internal error occurred during config writes or reads. Please contact Spectrum support for further assistance.
ERR_FIFOHWOVERRUN	301h	769	FIFO acquisition: Hardware buffer overrun in FIFO mode. The complete on-board memory has been filled with data and data wasn't transferred fast enough to PC memory. FIFO replay: Hardware buffer underrun in FIFO mode. The complete on-board memory has been replayed and data wasn't transferred fast enough from PC memory. If acquisition or replay throughput is lower than the theoretical bus throughput, check the application buffer setup.
ERR_FIFOFINISHED	302h	770	FIFO transfer has been finished, programmed data length has been transferred completely.
ERR_TIMESTAMP_SYNC	310h	784	Synchronization to timestamp reference clock failed. Please check the connection and the signal levels of the reference clock input.
ERR_STARHUB	320h	800	The auto routing function of the Star-Hub initialization has failed. Please check whether all cables are mounted correctly.
ERR_INTERNAL_ERROR	FFFFh	65535	Internal hardware error detected. Please check for driver and firmware update of the card.



SPECTRUM
INSTRUMENTATION

Spectrum Instrumentation GmbH
Ahrensfelder Weg 13-17 | 22927 Grosshansdorf | Germany
Phone +49 (0)4102-69 56-0 | Fax +49 (0)4102-69 56-66
info@spec.de

SPECTRUM
INSTRUMENTATION CORP.

Spectrum Instrumentation Corp
401 Hackensack Ave, 4th Floor | Hackensack, NJ 07601 | USA
Phone +1 (201) 562-1999 | Fax +1 (201) 342-7598
sales@spectrum-instrumentation.com

spectrum-instrumentation.com