digitizerNETBOX
DN2.48x-xx

Ethernet/LXI remote digitizer with 16 bit resolution

Hardware Manual
Software Driver Manual

English version **May 7, 2020**

SPECTRUM INSTRUMENTATION GMBH · AHRENSFELDER WEG 13-17 · 22927 GROSSHANSDORF · GERMANY
PHONE: +49 (0)4102-6956-0 · FAX: +49 (0)4102-6956-66 · E-MAIL: info@spec.de · INTERNET: www.spectrum-instrumentation.com

Introduction

Preface

This manual provides detailed information on the hardware features of your Spectrum instrument. This information includes technical data, specifications, block diagrams and a connector description.

In addition, this guide takes you through the process of installing and recognizing your hardware and also describes the installation of the delivered driver package for each operating system.

Finally this manual provides you with the complete software information of the hardware and the related driver. The reader of this manual will be able to control the instrument from any PC system with one of the supported operating systems and one of the supported operating software packages.

Please note that this manual provides no description for specific driver parts such as those for IVI, LabVIEW or MATLAB. These driver manuals are available on USB-Stick or on the Spectrum website.

For any new information on the board as well as new available options or memory upgrades please contact our website www.spectrum-instrumentation.com. You will also find the current driver package with the latest bug fixes and new features on our site.

Please read this manual carefully before you install any hardware or software. Spectrum is not responsible for any hardware failures resulting from incorrect usage.

General Information

The DN2.48x series allows recording of up to 4 channels in the high speed high resolution segment. Due to the proven design a wide variety of 16 bit digitizerNETBOX products can be offered. These products are available in several versions and different speed grades making it possible for the user to find a individual solution.

The digitizerNETBOX products can be used with maximum sample rates of up to 65 MS/s, 105 MS/s or 180 MS/s using either two or four (SE) channels. The installed memory of up to 2 GSample per digitizer unit will be used for fast data recording. It can completely be used by the current active channels. If using slower sample rates the memory can be switched to a FIFO buffer and data will be transferred online over Ethernet to the PC memory or to hard disk.

Application examples: Laboratory equipment, Super-sonics, LDA/PDA, Radar, Spectroscopy.

digitizerNETBOX Overview

The series of digitizerNETBOX products are remote powerful digitizer instruments with GBit Ethernet connection following the LXI Core 2011 standard. The proven internal digitizer modules, a stable chassis, an embedded remote controller, sufficient air cooling and standard BNC connectors form an unique instrument that opens a lot of new application areas.

The digitizerNETBOX can be either directly connected to a PC or Laptop or it can be connected to a company/institute LAN and can be accessed from any PC within that LAN. Using the digitizerNETBOX offers the following benefits and new possibilities compared to digitizer plug-in cards:

- Use a powerful digitizer without opening the PC and without mounting hardware inside the PC.
- Share the digitizer within a group of engineers that only need the instrument from time to time.
- Place the digitizer directly near the signal sources and control it remotely from the desk.
- Use the instrument at different location without moving a complete system. One just needs the digitizerNETBOX, a few cables and a Laptop.
- Use the digitizer as s mobile data acquisition device with the DC power option (DN2.xxx only).

Internal Digitizer Modules

The digitizerNETBOX products internally consist of either digitizer modules that are accessed and programmed in a similiar way as the Spectrum digitizer cards themselves.

Accessing the digitizerNETBOX by software therefore is nearly identical to accessing the same plug-in cards. Throughout the manual all programming and software usage will be described for the internal digitizer modules.

Differences between plain cards and digitizer modules inside the digitizerNETBOX

Feature	Plain M2i-Express Card	Installed inside digitizerNETBOX DN2.20x, DN2.46x, DN2.47x, DN2.48x, DN2.49x	Installed inside digitizerNETBOX DN6.20x, DN6.46x, DN6.49x
Trigger Input B	Only available as part of option BaseXIO	Available as standard	Available as standard
Timestamp Reference Clock Input	Only available as part of option BaseXIO	Available as standard	Available as standard
Option BaseXIO	Option can be ordered with purchase	Not available	Not available
Option Star-Hub	Option can be ordered and allows to connect 5 or 16 cards	Option installed internally in all digitizerNETBOXes with two internal modules	Option installed internally in all models
Standard Memory	512 MSamples/256 MSamples per card (for 8bit / 16bit samples)	1 GSamples/512 MSamples per module (for 8bit / 16bit samples)	1 GSamples/512 MSamples per module (for 8bit / 16bit samples)
Maximum Memory	2 GSamples/1 GSamples per card (for 8bit / 16bit samples)	2 GSamples/1 GSamples per module (for 8bit / 16bit samples)	2 GSamples/1 GSamples per module (for 8bit / 16bit samples)

Feature	Plain M4i-Express Card	Installed inside digitizerNETBOX DN2.22x and DN2.44x	Installed inside digitizerNETBOX DN6.22x and DN6.44x
Option Star-Hub	Option can be ordered and allows to connect 8 cards	Option installed internally in all digitizerNETBOXes with two internal modules	Option installed internally in all models
Standard Memory	4 GSamples per card: M4i.22xx 2 GSamples per card: M4i.44xx	4 GSamples per module in DN2.22x 2 GSamples per module in DN2.44x	4 GSamples per module in DN6.22x 2 GSamples per module in DN6.44x
Maximum Memory	4 GSamples per card: M4i.22xx 2 GSamples per card: M4i.44xx	4 GSamples per module in DN2.22x 2 GSamples per module in DN2.44x	4 GSamples per module in DN6.22x 2 GSamples per module in DN6.44x

Feature	Plain M2p-Express Card	Installed inside digitizerNETBOX DN2.59x	Installed inside digitizerNETBOX DN6.59x
Option Star-Hub	Option can be ordered and allows to connect either 6 or 16 cards	Option installed internally in all models with two internal modules	Option installed internally in all models
Standard Memory	512 MSamples per card	512 MSamples per module	512 MSamples per module
Maximum Memory	512 MSamples per card	512 MSamples per module	512 MSamples per module

Overview of digitizer modules inside the DN2.48x

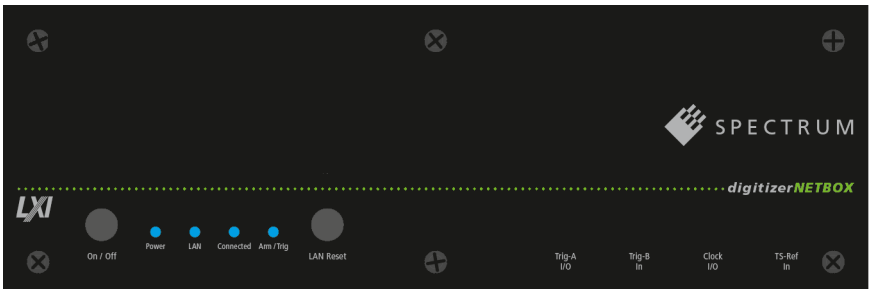
digitizerNETBOX model	Resolution	Single-Ended Differential	Max Speed	Number of Modules	Digitizer Module Type	Internal Star-Hub	Memory per module	Max memory per module
DN2								
Currently not available. Please contact Spectrum for information on availability.								
DN6								
Currently not available. Please contact Spectrum for information on availability.								

As an example: a DN2.472-32 would be recognized and programmed inside the software as 2 cards of M2i.4721-exp and 1 star-hub

Different models of the DN2.48x series

The following overview shows the different available models of the DN2.48x series. They differ in the number of internally mounted digitizer modules and the number of available channels.

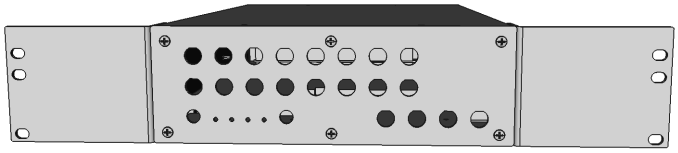
- **DN2.48x-xx**



Additional options for DN2 products

19" Rack Mount Kit

The rack mount kit allows to mount the digitizerNETBOX/generatorNETBOX into a standard 19" rack. The digitizerNETBOX/generatorNETBOX DN2 requires two height units of the 19" rack.



Multiple digitizerNETBOX/generatorNETBOX products can be mounted one on top of the other.

It is not possible to mount two digitizerNETBOX/generatorNETBOX DN2 products side by side into one 19" slot.

DC Power Supply

The DC power supply option is factory mounted and allows the connection of a DC source directly to the digitizerNETBOX/generatorNETBOX.

AC Cable Options

The system is delivered with a connection cable meeting your countries power connection. Other power cables can be ordered separately to connect your products with your local power connection system. A comprehensive list of all world-wide power plugs in use can be found on the IEC (Interbational Electrotechnical Commission) website: http://www.iec.ch/worldplugs/

The following power cable options are available from Spectrum:

001: Universal Type for IEC Plug Type E and Type F

The power cable is suitable for Continental Europe, Korea and others.

Cab-Pwr-001: 180 cm cable to CEE 7/VII

002: IEC Plug Type B

The power cable complies to standards UL 62 and UL 1581 and is suitable for US, Canada, Taiwan and others.

Cab-Pwr-002: 180 cm cable for NEMA5-15P

003: IEC Plug Type G

The power cable is suitable for United Kingdom, Ireland, Hong Kong, Saudi Arabia and others.

Cab-Pwr-003: 180 cm cable to BS 1363A

004: IEC Plug Type J

The power cable is suitable for Switzerland and others.

Cab-Pwr-004: 180 cm cable for SEV type 12

005: IEC Plug Type I

The power cable is suitable for Mainland China, Australia, New Zealand, Argentina and others.

Cab-Pwr-005: 180 cm cable for AS 3112

006: IEC Plug Type M

The power cable is suitable for India, Singapore, South Africa and others.

Cab-Pwr-006: 180 cm cable for IEC 83-B

007: IEC Plug Type K

The power cable is suitable for Denmark and others.

Cab-Pwr-007: 180 cm cable for SR 107-2-D

008: IEC Plug Type H

The power cable is suitable for Israel.

Cab-Pwr-008: 180 cm cable for SI 32

009: IEC Plug Type B

The power cable complies to standard JIS C3306 and is suitable for Japan.

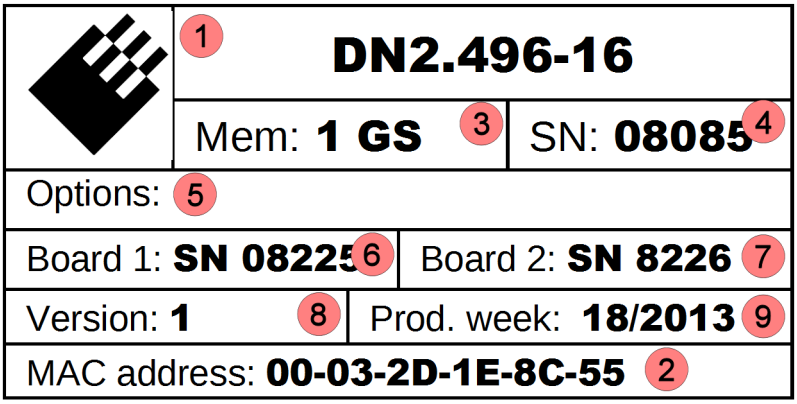
Cab-Pwr-009: 180 cm cable for NEMA5-15P

010: IEC Plug Type L

The power cable is suitable for Italy, Chile and others.

Cab-Pwr-010: 180 cm cable for CEI 23-16

The Spectrum type plate



The Spectrum type plate, which consists of the following components, can be found on the back of all netbox products. Please check whether the printed information is the same as the information on your delivery note. All this information can also be read out by software:

(1) The digitizerNETBOX/generatorNETBOX type, consisting of the abbrevaition for the digitizerNETBOX/generatorNETBOX chassis type (DN2 in this example), the model type (496 in this example) and the number of channels (16 in this example)

(2) The MAC address of the device. The MAX address is fixed and cannot be changed by the user. To check the MAC address by software one can use the integrated web pages of the digitizerNETBOX/generatorNETBOX.

(3) The installed complete data acquisition memory of the digitizerNETBOX/generatorNETBOX. As in our example there are two internal digitizer/generator modules installed the memory is shared between them. Each internal digitizer/generator module has 512 MSamples installed.

(4) The serial number of the digitizerNETBOX/generatorNETBOX itself. This is the serial number also found on the delivery note.

(5) Installed options of the digitizerNETBOX/generatorNETBOX.

(6) The serial number of the first internal digitizer/generator module.

(7) The serial number of the second internal digitizer/generator module.

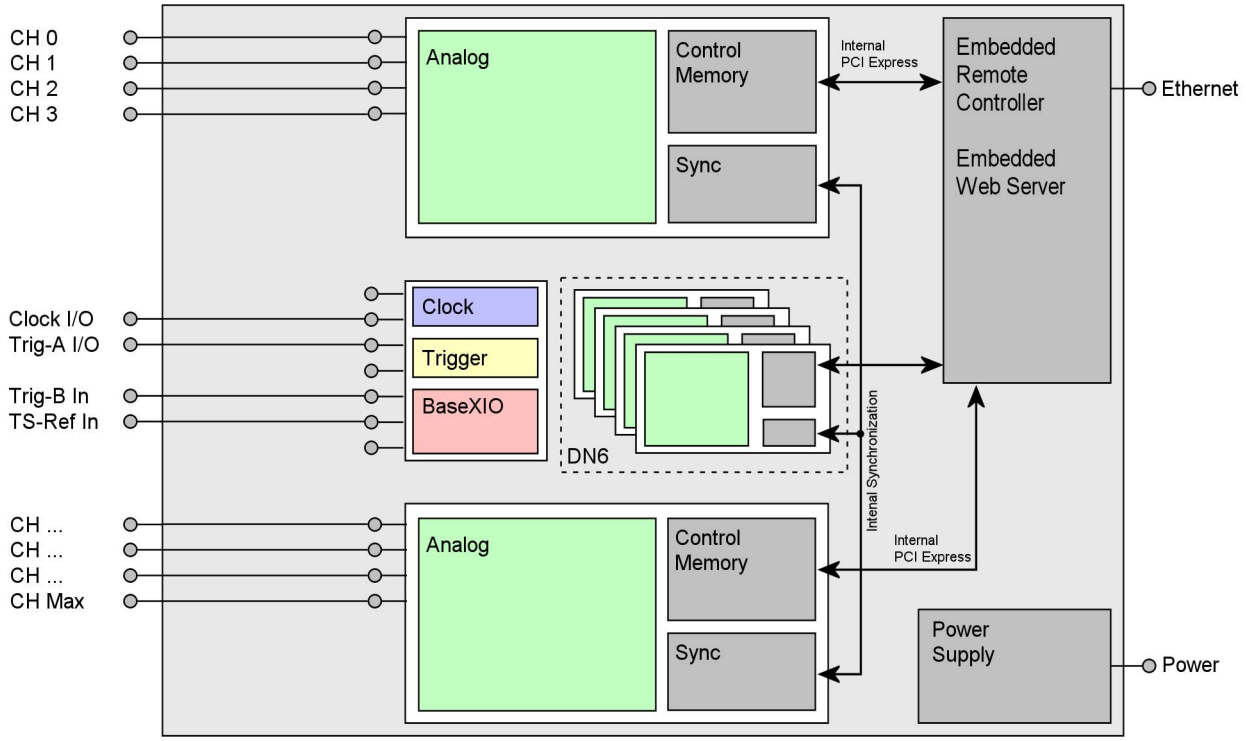
(8) The hardware version of the digitizerNETBOX/generatorNETBOX. The hardware and firmware versions of the installed digitizer/generator modules are found using the Spectrum Control Center.

(9) The date of production of the digitizerNETBOX/generatorNETBOX consisting of the calendar week and the year.

Please always supply us with the above information, especially the serial number in case of support request. That allows us to answer your questions as soon as possible. Thank you.

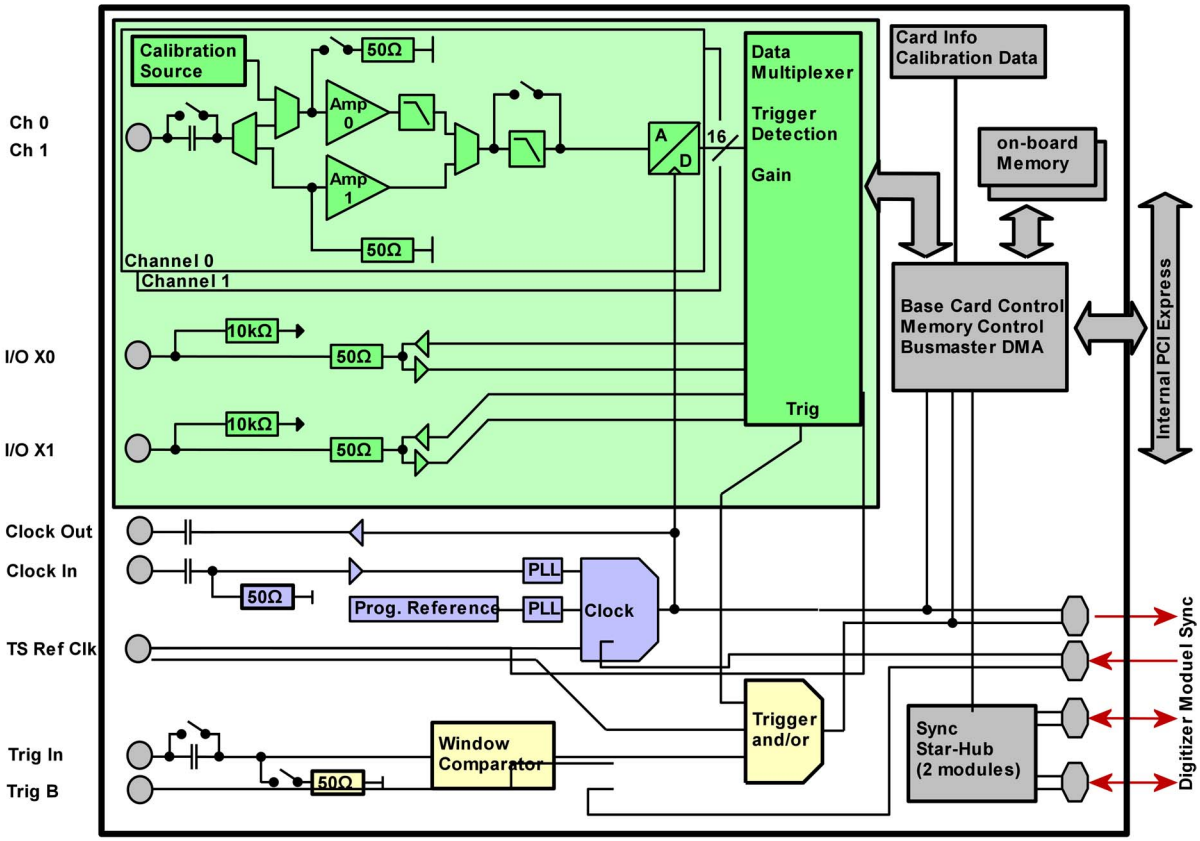
Hardware information

Block diagram of digitizerNETBOX DN2.48x:



- The number of maximum channels and internal digitizer modules and existance of a synchronization Star-Hub is model dependent.

Block diagram of a single internal digitizer module:



Technical Data

Analog Inputs

Resolution		16 bit	
Input Type		Single-ended	
Programmable Input Offset		not available	
ADC Differential non linearity (DNL)	ADC only	≤ 1.0 LSB	
ADC Integral non linearity (INL)	ADC only	≤ 4.0 LSB	
Channel selection	software programmable	1 or 2 channels (maximum is model dependent)	
Bandwidth filter 4830, 4831	activate by software	10 MHz bandwidth with 3rd order Butterworth filtering	
Bandwidth filter 4840, 4841, 4860, 4861	activate by software	20 MHz bandwidth with 3rd order Butterworth filtering	
Input Path Types	software programmable	**50 Ω (HF) Path**	**Buffered (high impedance) Path**
Analog Input impedance	software programmable	50 Ω	1 MΩ || 25 pF or 50 Ω
Input Ranges	software programmable	±500 mV, ±1 V, ±2.5 V, ±5 V	±200 mV, ±500 mV, ±1 V, ±2 V, ±5 V, ±10 V
Input Coupling	software programmable	AC/DC	AC/DC
Offset error (full speed)	after warm-up and calibration	≤ 0.1%	≤ 0.1%
Gain error (full speed)	after warm-up and calibration	≤ 1.0%	≤ 0.1%
Over voltage protection	range ≤ ±1V	2 Vrms	±5 V (1 MΩ), 5 Vrms (50 Ω)
Over voltage protection	range ≥ ±2V	6 Vrms	±30 V (1 MΩ), 5 Vrms (50 Ω)
Max DC voltage if AC coupling active		±30 V	±30 V
Relative input stage delay		Bandwidth filter disabled: 0 ns Bandwidth filter enabled: 14.7 ns	Bandwidth filter disabled: 3.8 ns Bandwidth filter enabled: 18.5 ns
Crosstalk 1 MHz sine signal	input range ±1 V	not available	≤ -100 dB
Crosstalk 20 MHz sine signal	input range ±1 V	not available	≤ -100 dB
Crosstalk 1 MHz sine signal	input range ±5 V	≤ -110 dB	≤ -92 dB
Crosstalk 20 MHz sine signal	input range ±5 V	≤ -102 dB	≤ -92 dB

Trigger

Available trigger modes	software programmable	Channel Trigger, Ext0 (Analog), Ext1 (TT), Software, Window, Re-Arm, Or/And, Delay	
Trigger level resolution	software programmable	10 bits	
Trigger edge	software programmable	Rising edge, falling edge or both edges	
Trigger delay	software programmable	0 to (8GSamples - 8) = 8589934584 Samples in steps of 8 samples	
Multi, Gate: re-arming time		≤ 32 samples (+ programmed pretrigger)	
Pretrigger at Multi, ABA, Gate, FIFO	software programmable	8 up to [8192 Samples / number of active channels] in steps of 8	
Posttrigger	software programmable	8 up to 4 GSamples in steps of 8(defining pretrigger in standard scope mode)	
Memory depth	software programmable	16 up to [installed memory / number of active channels] samples in steps of 8	
Multiple Recording/ABA segment size	software programmable	16 up to [installed memory / 2 / active channels] samples in steps of 16	
Trigger output delay	after trigger input	134 sampling clock cycles	
Internal/External trigger accuracy		1 sample	
External trigger		**Ext0 (Trg)**	**Ext1 (X0) + Ext2 (X1)**
External trigger impedance	software programmable	50 Ω /1 MΩ || 25 pF	10 kΩ to 3.3 V
External trigger coupling	software programmable	AC or DC	fixed DC
Minimum trigger pulse width	(DC / AC)	≥ 2 samples	≥ 2 samples
External trigger bandwidth DC	50 Ω /1 MΩ	DC to 200 MHz / 150 MHz	DC to 125 MHz
External trigger bandwidth AC	50 Ω	20 kHz to 200 MHz	n.a.
External trigger type		Window comparator, ±5 V	TTL level
External trigger level	software programmable	2 levels ±5V in steps of 10 mV	fixed: Low: ≤0.8 V, High: ≥2.0 V
External trigger maximum voltage		5V rms (50 Ω), ±30V (1 MΩ)	-0.3 V to +5.5V
External trigger output impedance		input only	50 Ω
External trigger output levels		input only	Low: ≤0.4 V, High: ≥2.4 V
External trigger output type		input only	3.3 V LVTTL.TTL compatible for high impedance
External trigger output drive strength		input only	Capable of driving 50 Ω loads, ±64 mA output

Clock

Clock Modes	software programmable	internal, external reference clock, sync
Internal clock accuracy		≤ ±32 ppm
Internal clock setup granularity		1 Hz (except the clock setup gaps shwon below)
Clock setup range gaps	clock not programmable	70 MHz to 72 MHz, 140 MHz to 144 MHz, 281 MHz to 287 MHz
External reference clock range	software programmable	≥ 10 MHz and ≤ 1 GHz (fix at runtime)
External reference clock setup granularity		1 kHz
External clock input impedance	software programmable	50 Ω fixed
External clock input coupling		AC coupling
External clock input edge		Rising edge
External clock input to internal ADC clock delay		3.7 ns (8.2 ns if synchronization is used)
External clock input type		Single-ended, sine wave or square wave
External clock input swing		0.3 V peak-peak up to 3.0 V peak-peak
External clock input max DC voltage		±30 V (with max 3.0 V difference between low and high level)
External clock input duty cycke requirement		40% to 60%
External clock output type		Single-ended, 3.3V LVPECL
External clock output coupling		AC coupling
ABA mode clock divider for slow clock	software programmable	8 up to [128k - 8] in steps of 8

Connectors

Analog Inputs		SMA female (one for each single-ended input)	Cable-Type: Cab-3mA-xx-xx
Trigger A Input		SMA female	Cable-Type: Cab-3mA-xx-xx
Trigger B Input/Output	programmable direction	SMA female	Cable-Type: Cab-3mA-xx-xx
Clock Input		SMA female	Cable-Type: Cab-3mA-xx-xx
Clock Output		SMA female	Cable-Type: Cab-3mAxx-xx
Timestamp Reference Clock Input		SMA female	Cable-Type: Cab-3mA-xx-xx

Option digitizerNETBOX/generatorNETBOX embedded server (DN2.xxx-Emb, DN6.xxx-Emb)

CPU	Intel Quad Core 2 GHz	
System memory	4 GByte RAM	
System data storage	Internal 128 GByte SSD	
Development access	Remote Linux command shell (ssh), no graphical interface (GUI) available	
Accessible Hardware	Full access to Spectrum instruments, LAN, front panel LEDs, RAM, SSD	
Integrated operating system	OpenSuse 12.2 with kernel 4.4.7.	
Internal PCIe connection	DN2.20, DN2.46, DN2.47, DN2.49, DN2.59, DN2.60 DN6.46, DN6.49, DN6.59	PCIe x1, Gen1
	DN2.22, DN2.44, DN2.66 DN6.22, DN6.44, DN6.66	PCIe x1, Gen2

Ethernet specific details

LAN Connection		Standard RJ45	
LAN Speed		Auto Sensing: GBit Ethernet, 100BASE-T, 10BASE-T	
LAN IP address	programmable	DHCP (IPv4) with AutoIP fall-back (169.254.x.y), fixed IP (IPv4)	
Sustained Streaming speed		DN2.20, DN2.46, DN2.47, DN2.49, DN2.60 DN6.46, DN6.49	up to 70 MByte/s
		DN2.59, DN2.22, DN2.44, DN2.66 DN6.59, DN6.22, DN6.44, DN6.66	up to 100 MByte/s
Used TCP/UDP Ports		Webserver: 80 VISA Discovery Protocol: 111, 9757 Spectrum Remote Server: 1026, 5025	mDNS Daemon: 5353 UPNP Daemon: 1900

Power connection details

Mains AC power supply	Input voltage: 100 to 240 VAC, 50 to 60 Hz
AC power supply connector	IEC 60320-1-C14 (PC standard coupler)
Power supply cord	power cord included for Schuko contact (CEE 7/7)

Serial connection details (DN2.xxx with hardware ≥ V11)

Serial connection (RS232)	For diagnostic purposes only. Do not use, unless being instructed by a Spectrum support agent.

Certification, Compliance, Warranty

EMC Immunity	Compliant with CE Mark
EMC Emission	Compliant with CE Mark
Product warranty	5 years starting with the day of delivery
Software and firmware updates	Life-time, free of charge

Environmental and Physical Details DN2.xxx

Dimension of Chassis without connectors or bumpers	L x W x H	366 mm x 267 mm x 87 mm
Dimension of Chassis with 19" rack mount option	L x W x H	366 mm x 482.6 mm x 87 mm (2U height)
Weight (1 internal acquisition/generation module)		6.3 kg, with rack mount kit: 6.8 kg
Weight (2 internal acquisition/generation modules)		6.7 kg, with rack mount kit 7.2 kg
Warm up time		20 minutes
Operating temperature		0°C to 40°C
Storage temperature		-10°C to 70°C
Humidity		10% to 90%
Dimension of packing (single DN2)	L x W x H	470 mm x 390 mm x 180 mm
Volume weight of Packing (single DN2)		7.0 kgs

Dynamic Parameters

	M3i.4861 and M3i.4860, 1 or 2 channels 180 MS/s									
Input Path	HF path, AC coupled, fixed 50 Ohm				Buffered path, BW limit			Buffered path, full BW		
Test signal frequency	1 MHz	10 MHz		40 MHz	10 MHz			1 MHz	10 MHz	40 MHz
Input Range	±1V	±500mV	±1V	±1V	±200mV	±500mV	±1V	±500mV	±500mV	±500mV
RMS Noise (zero level)	≤ 8.0 LSB				≤ 10.0 LSB			≤ 10.0 LSB		
THD (typ) (dB)	-80.6	-79.2	-79.3	-77.8	-77.4	-77.7	-75.3	-83.4	-77.7	-47.8
SNR (typ) (dB)	73.1	73.3	73.4	71.9	71.4	72.8	73.1	71.1	72.8	68.6
SFDR (typ), excl. harm. (dB)	92.4	96.0	96.8	87.8	95.8	96.8	96.7	87.6	96.4	88.2
SFDR (typ), incl. harm. (dB)	81.1	80.5	80.5	78.8	79.0	78.7	76.2	85.2	79.0	48.0
SINAD/THD+N (typ) (dB)	72.4	72.3	72.3	70.9	70.4	71.6	73.1	70.9	71.6	47.8
ENOB based on SINAD (bit)	11.7	11.7	11.7	11.5	11.4	11.6	11.5	11.5	11.6	7.6
ENOB based on SNR (bit)	11.9	11.9	11.9	11.7	11.6	11.8	11.8	11.5	11.8	11.1

	M3i.4841 and M3i.4840, 1 or 2 channels 105 MS/s									
Input Path	HF path, AC coupled, fixed 50 Ohm				Buffered path, BW limit			Buffered path, full BW		
Test signal frequency	1 MHz	10 MHz			10 MHz			1 MHz	10 MHz	
Input Range	±1V	±500mV	±1V		±200mV	±500mV	±1V	±500mV	±500mV	
RMS Noise (zero level)	≤ 7.0 LSB				≤ 10.0 LSB			≤ 10.0 LSB		
THD (typ) (dB)	-86.0	-87.3	-88.0		-83.0	-82.1	-76.2	-85.0	-79.8	
SNR (typ) (dB)	74.5	74.7	74.7		71.7	73.9	74.2	73.1	73.0	
SFDR (typ), excl. harm. (dB)	93.0	97.0	97.1		92.8	93.5	93.1	92.5	96.3	
SFDR (typ), incl. harm. (dB)	86.5	91.5	91.7		85.3	85.1	79.0	87.5	81.5	
SINAD/THD+N (typ) (dB)	74.2	74.5	74.5		71.4	73.3	72.1	72.8	72.2	
ENOB based on SINAD (bit)	12.0	12.1	12.1		11.6	11.9	11.7	11.8	11.7	
ENOB based on SNR (bit)	12.1	12.1	12.1		11.6	12.0	12.0	11.9	11.8	

	M3i.4831 and M3i.4830, 1 or 2 channels 65 MS/s									
Input Path	HF path, AC coupled, fixed 50 Ohm				Buffered path, BW limit			Buffered path, full BW		
Test signal frequency	1 MHz	10 MHz			10 MHz			1 MHz	10 MHz	
Input Range	±1V	±500mV	±1V		±200mV	±500mV	±1V	±500mV	±500mV	
RMS Noise (zero level)	≤ 5.0 LSB				≤ 9.0 LSB			≤ 9.0 LSB		
THD (typ) (dB)	-85.0	-86.2	-86.2		-83.5	-80.8	-76.5	-84.1	-80.4	
SNR (typ) (dB)	75.0	75.4	75.2		72.3	74.6	74.8	73.8	74.2	
SFDR (typ), excl. harm. (dB)	94.5	92.0	90.8		88.5	91.4	90.7	88.3	91.0	
SFDR (typ), incl. harm. (dB)	81.5	87.7	87.5		84.7	83.3	78.8	85.2	81.5	
SINAD/THD+N (typ) (dB)	74.6	75.1	74.9		72.0	73.7	72.6	73.4	73.4	
ENOB based on SINAD (bit)	12.0	12.2	12.2		11.7	11.9	11.8	11.9	11.9	
ENOB based on SNR (bit)	12.2	12.2	12.2		11.7	12.1	12.1	12.0	12.0	

A pure sine wave with > 99% amplitude of input range is measured with 50 ohms termination. SNR and RMS noise parameters may differ depending on the quality of the used PC. SNR = Signal to Noise Ratio, THD = Total Harmonic Distortion, SFDR = Spurious Free Dynamic Range, SINAD = Signal Noise and Distortion, ENOB = Effective Number of Bits. Depending on the test signal frequency different filter types are used: 1 MHz signal = 7th order low pass, 10 MHz signal = 6th order band pass, 40 MHz signal = 6th order bandpass.

DN2 Order Information

The digitizerNETBOX is equipped with a large internal memory for data storage and supports standard acquisition (Scope), FIFO acquisition (streaming), Multiple Recording, Gated Sampling, ABA mode and Timestamps. Operating system drivers for Windows/Linux 32 bit and 64 bit, drivers and examples for C/C++, IVI (Scope and Digitizer class), LabVIEW (Windows), MATLAB (Windows and Linux), .NET, Delphi, Java, Python and a Professional license of the oscilloscope software SBench 6 are included.

The system is delivered with a connection cable meeting your countries power connection. Additional power connections with other standards are available as option.

***digitizerNETBOX* DN2 - Ethernet/LXI Interface**

Order no.	A/D Resolution	Bandwidth Standard	Bandwidth ir40m Option	1 Channel	2 Channels	4 Channels	8 Channels	Installed Memory
DN2.221-02	8 Bit	500 MHz	500 MHz	1.25 GS/s	1.25 GS/s			1 x 4 GS
DN2.221-04	8 Bit	500 MHz	500 MHz	1.25 GS/s	1.25 GS/s	1.25 GS/s		1 x 4 GS
DN2.221-08	8 Bit	500 MHz	500 MHz	1.25 GS/s	1.25 GS/s	1.25 GS/s	1.25 GS/s	2 x 4 GS
DN2.222-02	8 Bit	1.5 GHz	1.2 GHz	2.5 GS/s	2.5 GS/s			1 x 4 GS
DN2.222-04	8 Bit	1.5 GHz	1.2 GHz	2.5 GS/s	2.5 GS/s	2.5 GS/s		2 x 4 GS
DN2.223-02	8 Bit	1.5 GHz	1.2 GHz	5 GS/s	5 GS/s			2 x 4 GS
DN2.225-04	8 Bit	1.5 GHz	1.2 GHz	5 GS/s	2.5 GS/s	1.25 GS/s		1 x 4 GS
DN2.225-08	8 Bit	1.5 GHz	1.2 GHz	5 GS/s	5 GS/s	2.5 GS/s	1.25 GS/s	2 x 4 GS

Order no.	A/D Resolution	Bandwidth	Single-Ended Channels	Differential Channels	Sampling Speed	Installed Memory
DN2.441-02	16 Bit	65 MHz	2 channels	-	130 MS/s	1 x 2 GS
DN2.441-04	16 Bit	65 MHz	4 channels	-	130 MS/s	1 x 2 GS
DN2.441-08	16 Bit	65 MHz	8 channels	-	130 MS/s	2 x 2 GS
DN2.442-02	16 Bit	125 MHz	2 channels	-	250 MS/s	1 x 2 GS
DN2.442-04	16 Bit	125 MHz	4 channels	-	250 MS/s	1 x 2 GS
DN2.442-08	16 Bit	125 MHz	8 channels	-	250 MS/s	2 x 2 GS
DN2.445-02	14 Bit	250 MHz	2 channels	-	500 MS/s	1 x 2 GS
DN2.445-04	14 Bit	250 MHz	4 channels	-	500 MS/s	1 x 2 GS
DN2.445-08	14 Bit	250 MHz	8 channels	-	500 MS/s	2 x 2 GS
DN2.447-02[1]	16 Bit	125 MHz	2 channels	-	180 MS/s	1 x 2 GS
DN2.447-04[1]	16 Bit	125 MHz	4 channels	-	180 MS/s	1 x 2 GS
DN2.447-08[1]	16 Bit	125 MHz	8 channels	-	180 MS/s	2 x 2 GS
DN2.448-02[1]	14 Bit	250 MHz	2 channels	-	400 MS/s	1 x 2 GS
DN2.448-04[1]	14 Bit	250 MHz	4 channels	-	400 MS/s	1 x 2 GS
DN2.448-08[1]	14 Bit	250 MHz	8 channels	-	400 MS/s	2 x 2 GS

[1] Export Version

Order no.	A/D Resolution	Bandwidth	Single-Ended Channels	Differential Channels	Sampling Speed	Installed Memory	Available Memory Options
DN2.491-04	16 Bit	5 MHz	4 channels	2 channels	10 MS/s	1 x 512MS	1 x 1GS
DN2.491-08	16 Bit	5 MHz	8 channels	4 channels	10 MS/s	1 x 512MS	1 x 1GS
DN2.491-16	16 Bit	5 MHz	16 channels	8 channels	10 MS/s	2 x 512MS	2 x 1GS
DN2.496-04	16 Bit	30 MHz	4 channels	2 channels	60 MS/s (2 channels) 30 MS/s (4 channels)	1 x 512MS	1 x 1GS
DN2.496-08	16 Bit	30 MHz	8 channels	4 channels	60 MS/s (4 channels) 30 MS/s (8 channels)	1 x 512MS	1 x 1GS
DN2.496-16	16 Bit	30 MHz	16 channels	8 channels	60 MS/s (8 channels) 30 MS/s (16 channels)	2 x 512MS	2 x 1GS

Order no.	A/D Resolution	Bandwidth	Single-Ended Channels	Differential Channels	Sampling Speed	Installed Memory	Available Memory Options
DN2.462-04	16 Bit	100 kHz	4 channels	4 channels	200 kS/s	1 x 512MS	1 x 1GS
DN2.462-08	16 Bit	100 kHz	8 channels	8 channels	200 kS/s	1 x 512MS	1 x 1GS
DN2.462-16	16 Bit	100 kHz	16 channels	-	200 kS/s	2 x 512MS	2 x 1GS
DN2.464-04	16 Bit	500 kHz	4 channels	4 channels	1 MS/s	1 x 512MS	1 x 1GS
DN2.464-08	16 Bit	500 kHz	8 channels	8 channels	1 MS/s	1 x 512MS	1 x 1GS
DN2.464-16	16 Bit	500 kHz	16 channels	-	1 MS/s	2 x 512MS	2 x 1GS
DN2.465-04	16 Bit	1.5 MHz	4 channels	4 channels	3 MS/s	1 x 512MS	1 x 1GS
DN2.465-08	16 Bit	1.5 MHz	8 channels	8 channels	3 MS/s	1 x 512MS	1 x 1GS
DN2.465-16	16 Bit	1.5 MHz	16 channels	-	3 MS/s	2 x 512MS	2 x 1GS

Order no.	A/D Resolution	Bandwidth	Single-Ended Channels	Differential Channels	Sampling Speed	Installed Memory	Available Memory Options
DN2.203-02	8 Bit	90 MHz	2 channels	-	200 MS/s (1 channel) 100 MS/s (2 channels)	1 x 1GS	1 x 2GS
DN2.203-04	8 Bit	90 MHz	4 channels	-	200 MS/s (2 channel) 100 MS/s (4 channels)	1 x 1GS	1 x 2GS
DN2.203-08	8 Bit	90 MHz	8 channels	-	200 MS/s (4 channel) 100 MS/s (8 channels)	2 x 1GS	2 x 2GS

Options

Order no.	Option
DN2.xxx-Rack	19" rack mounting set for self mounting
DN2.xxx-Emb	Extension to Embedded Server: CPU, more memory, SSD. Access via remote Linux secure shell (ssh)
DN2.xxx-1x1GS	Memory extension to 1 x 1 GSample for 46x-04, 46x-08, 49x-04, 49x-08 versions
DN2.xxx-2x1GS	Memory extension to 2 x 1 GSample for 46x-16 and 49x-16 versions
DN2.xxx-DC12	12 VDC internal power supply. Replaces AC power supply. Accepts 9 V to 18 V DC input. Screw terminals.
DN2.xxx-DC24	24 VDC internal power supply. Replaces AC power supply. Accepts 18 V to 36 V DC input. Screw terminals
DN2.xxx-BTPWR	Boot on Power On: the digitizerNETBOX/generatorNETBOX automatically boots if power is switched on.

Calibration

Order no.	Option
DN2.xxx-Recal	Recalibration of complete digitizerNETBOX/generatorNETBOX DN2 including calibration protocol

BNC Cables

The standard adapter cables are based on RG174 cables and have a nominal attenuation of 0.3 dB/m at 100 MHz.

for Connections	Connection	Length	to SMA male	to SMA female	to BNC male	to SMB female	
All	BNC male	80 cm	Cab-9m-3mA-80	Cab-9m-3fA-80	Cab-9m-9m-80	Cab-9m-3f-80	
All	BNC male	200 cm	Cab-9m-3mA-200	Cab-9m-3fA-200	Cab-9m-9m-200	Cab-9m-3f-200	

Standard SMA Cables

The standard adapter cables are based on RG174 cables and have a nominal attenuation of 0.3 dB/m at 100 MHz and 0.5 dB/m at 250 MHz. For high speed signals we recommend the low loss cables series CHF.

for Connections	Connection	Length	to BNC male	to BNC female	to SMB female	to MMCX male	to SMA male	
All	SMA male	80 cm	Cab-3mA-9m-80	Cab-3mA-9f-80	Cab-3f-3mA-80	Cab-1m-3mA-80	Cab-3mA-3mA-80	
All	SMA male	200 cm	Cab-3mA-9m-200	Cab-3mA-9f-200	Cab-3f-3mA-200	Cab-1m-3mA-200	Cab-3mA-3mA-200	
Probes (short)	SMA male	5 cm		Cab-3mA-9f-5				

Low Loss SMA Cables

The low loss adapter cables are based on MF141 cables and have an attenuation of 0.3 dB/m at 500 MHz and 0.5 dB/m at 1.5 GHz. They are recommended for signal frequencies of 200 MHz and above.

Order no.	Option
CHF-3mA-3mA-200	Low loss cables SMA male to SMA male 200 cm
CHF-3mA-9m-200	Low loss cables SMA male to BNC male 200 cm

Additional AC Power Cables

Order no.	IEC Plug	Option
Cab-Pwr-001	Type E & F	AC power cable for Continental Europe, Korea and others with Schuko (CEE 7/VII) connector, 180 cm long
Cab-Pwr-002	Type V	AC power cable for US, Canada, Japan, Taiwan and others with NEMA5-15P connector, 180 cm long
Cab-Pwr-003	Type G	AC power cable for United Kingdom, Ireland, Hong Kong and others with BS 1363A connector, 180 cm long
Cab-Pwr-004	Type J	AC power cable for Switzerland and others with SEV type 12 connector, 180 cm long
Cab-Pwr-005	Type I	AC power cable for Mainland China, Australia, New Zealand and others with AS 3112 connector, 180 cm long
Cab-Pwr-006	Type M	AC power cable for India, Singapore, South Africa and others with 83-B1 connector, 180 cm long
Cab-Pwr-007	Type K	AC power cable for Denmark and others with SR 107-2-D connector, 180 cm long
Cab-Pwr-008	Type H	AC power cable for Israel with SI 32 connector, 180 cm long
Cab-Pwr-010	Type L	AC power cable for Italy, Chile and others with CEI 23-16 connector, 180 cm long

Hardware Installation

Warnings

ESD Precautions

The digitizerNETBOX/generatorNETBOX products internally contain electronic components that can be damaged by electrostatic discharge (ESD). The grounded chassis itself gives a very good protection against ESD.

Before installing the board in your system or protective conductive packaging, discharge yourself by touching a grounded bare metal surface or approved anti-static mat before picking up this ESD sensitive product.

Opening the Chassis

There are no components inside the chassis that need any operating by the user. In contrary there are a lot of components that may be harmed when operated unproperly by a use.

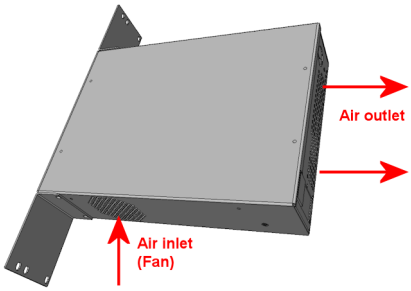
As Spectrum only gives a warranty on the complete instrument, opening the chassis will make you loose the warranty.

Cooling Precautions

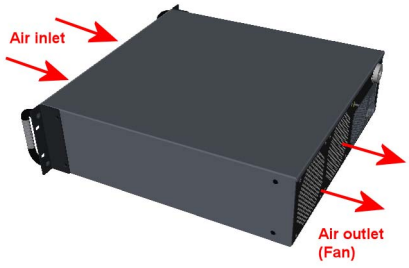
The high performance digitizers/generators of the digitizerNETBOX/generatorNETBOX operate with components having very high power consumption. Therefore the digitizerNETBOX/generatorNETBOX models have sufficient cooling fans.

Make sure that the air inlets and air outlets are free and uncovered and in case of a DN6 ensure that the installed filters at the inlet are cleaned regularly.

DN2 airflow:



DN6 airflow:



Sources of noise

The digitizerNETBOX/generatorNETBOX is using electrical components with very high resolution and high sensitivity. The signal inputs will acquire your signals with a high quality but will also collect spurious noise signals from various sources - especially if using the inputs in high impedance mode. To minimize this effect the cabeling must be made with care.

Keep away the cables from any sources that may inject noise into the signals like other instruments, crossing or even worse running in parallel with other cables with high frequency signals on them. If possible use differential signalling to minimize the effectes of injected noise.

A standard GND screw on the back of the chassis allows to connect the metal chassis to measurement ground to reduce noise based on ground loops and ground level differences.

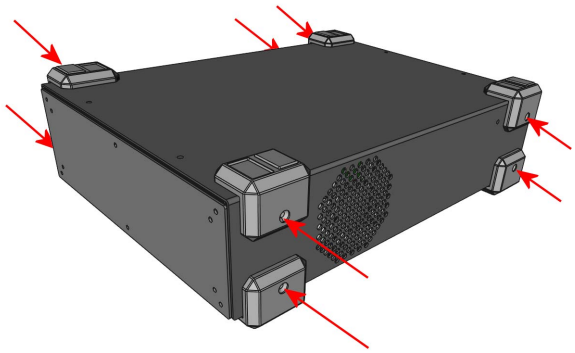
Installing 19“ rack mount option for DN2

This option has to be ordered separately. It can be ordered together with the digitizerNETBOX/generatorNETBOX at the time of purchase or it can be ordered later on, if it is becoming necessary to mount the digitizerNETBOX/generatorNETBOX into a 19“ rack. In any case the digitizerNETBOX/generatorNETBOX comes pre-configured as a standalone unit, which has then to manually be converted to the rackmount configuration by the user.

Step 1

The rackmount option comes with the required Torx T20 size screw driver to un-mount the default screws holding the bumper feet.

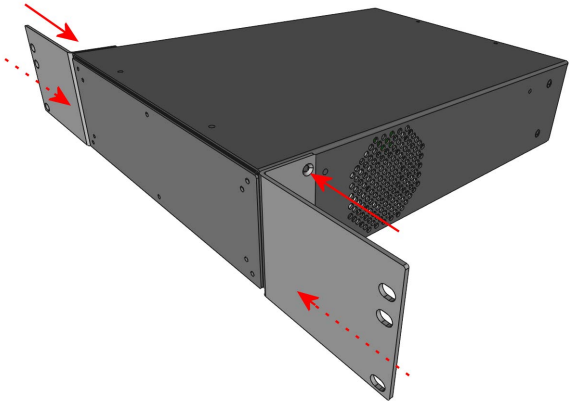
Unscrew these 8 Torx T20 screws with the provided screw driver and keep them together with the un-mounted bumpers for possible later use in case the rackmount option shall be un-mounted again in the future.



Step 2

Mount the 19“ rack mount extension using the four phillips-head screws that are also provided with each rack mount extension. Two screws are required for each rack mounting bracket.

Care should be taken to not overtighten the screws.



Installing 19“ rack mount option for DN6

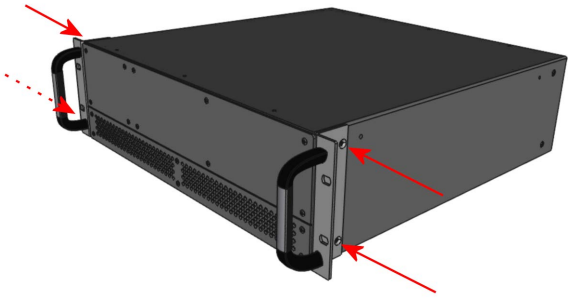
Installing the rack mount extension for the DN6 series follows the same principles as for the DN2 models shown above.

Step 1

Unscrew the existing bumper corner pieces with the provided screw driver and keep them together with the un-mounted bumpers for possible later use in case the rackmount option shall be un-mounted again in the future.

Step 2

Mount the 19“ rack mount extension using the four phillips-head screws that are also provided with each rack mount extension. Two screws are required for each rack mounting bracket. Care should be taken to not overtighten the screws.



In addition to using the provided rack mount extension for fastening the DN6 device within the 19“ rack, the user must take additional measures, suitable for the used rack, to provide adequate mechanical support at the backside of the device.

This support is required for DN6 devices due to their higher weight compared to DN2 devices.

Setup of digitizerNETBOX/generatorNETBOX

Connections

First of all the digitizerNETBOX/generatorNETBOX needs to be connected to both power line and LAN environment:

Power

Connect the power line cable to a matching power source. First connect the cable to the digitizerNETBOX/generatorNETBOX, second connect the cable to the power plug. Please check the technical data section to see the requirements for the power supply.

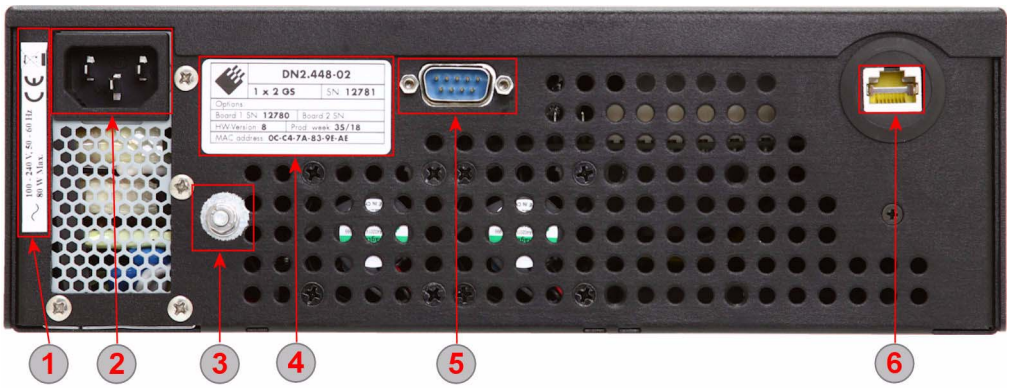
If using a DC power option please be sure to have the external DC power source switched off while connecting the power lines. Only switch on the power supply after all connections have been done and are checked.

Ethernet

Connect the digitizerNETBOX/generatorNETBOX to either your company LAN or directly to your PC. Please use a standard Cat-5 or better Ethernet cable for the connection.

Back Side DN2

The right hand pictures shows the back side of one digitizerNETBOX/generatorNETBOX with standard AC power supply. The different power supply options are described later in this chapter. The picture is taken from a digitizerNETBOX hardware revisoon V8. Older version look different.

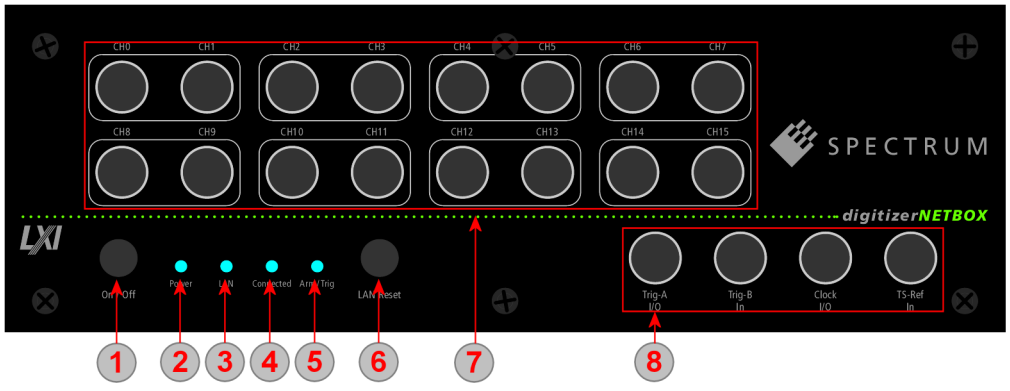


Please see the table below for a description of the different marked areas:

Area	Name	Description
1	Power Label	The label shows the power specification in detail. Please check the listed specification before connection the power line
2	Power Connector (AC)	Standard three pole power connector. A matching power cable is included in the delivery. Separate power cables for other country standards are available upon request.
3	GND Screw	THis screw is directly connected to Chassis ground and be used to add a low resistance ground connection to the system
4	Type plate	The type plate shows exact type, option, serieal number, versions and production week. A more detailed desctiption of the type plate is found in a separate chapter of the manual
5	Debug Port (DSUB)	This port is for debug purposes only. Please only connect a cable when asked by the Spectrum support group. The debug connector is a feature of hardware revision V8 and is not available on earlier versions
6	LAN Connection	A standard Ethernet port. Please connect the device with your PC/Laptop or company LAN before start

Front Panel DN2

The right-hand drawing gives you an overview on one digitizerNETBOX DN2 front panel. Depending on the version of the digitizerNETBOX or generatorNETBOX you have the area 7 may differ in terms of number of channels or grouping of the channels.

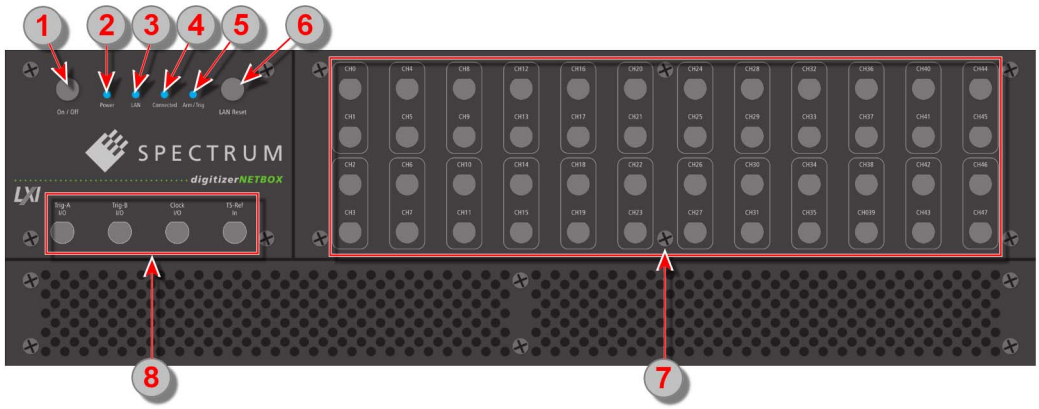


In area 8 a version with 4 BNC connectors is shown. Other versions with 5 SMA, 6 BNC or 7 SMA connectors are also available. Please see the table below for the different connections.

Area	Name	Status	Description
1	Power On/Off	press while device stopped	digitizerNETBOX/generatorNETBOX is started
		short press while device is running	digitizerNETBOX/generatorNETBOX is closing the embedded controller and is going into standby mode
		long press while device is running	digitizerNETBOX/generatorNETBOX is aborted and is going into standby. Please only use this stop method if the digitizerNETBOX/generatorNETBOX is not responding
2	Power LED	LED off	no power connected to the device
		LED orange	power is connected, device is in standby mode
		LED green	device has started and is working
3	LAN LED	LED off	Only off during boot up, turning to either red or green afterwards. If permanently off, contact support.
		LED red	Error while trying to get a LAN connection
		LED green	Device is connected to LAN.
		LED green flashing	Device is connected to LAN. Flashing indicates LAN ID (see webserver)
4	Connected LED	LED off	Device is not in use
		LED green	Device is in use by other PC
5	Arm/Trigger	LED off	No trigger detected, device is waiting for trigger event, or not armed at all
		LED green	Trigger detected, acquisition is running or already finished
6	LAN Reset	press once	Does a reset of the LAN settings to default state. The reset button needs to be pressed for 4 seconds to issue the reset. The reset command is then issued immediatley independent of the current run state of the device.
7	Signal Connections		Connect your input signals here. For differential connections use even channels for positive phase and odd channels for negative phase.
8	Control Connections (4 BNC connector version, for M2i module based products)	Trig-A I/O	Trigger A with programmable input or output. This is the main external trigger
		Trig-B In	Trigger B, input only. This trigger is referenced in the manual as TRIG_XIO0
		Clock I/O	Clock with programmable input or output
		TS-Ref In	Timestamp Reference Clock Input
8	Control Connections (5 SMA connector version, for M3i module based products))	Clock In	External clock input
		Clock Out	External clock output
		Trig-A In	Trigger A, input only. This is the main external trigger. The trigger line is reference in the manual as EXT0
		Trig-B I/O	Trigger B/Multi Purpose X0 with programmable direction. The connection is referenced in the manual as X0
		TS_Ref In	Timestamp Reference Clock Input
8	Control Connections (6 BNC connector version, for M2p module based products))	Clock In	External clock input
		Trig In	Trigger, input only. This is the main external trigger. The trigger line is reference in the manual as EXT0
		X0 Out	Multi Purpose X0, output only. Clock output available. The connection is referenced in the manual as X0
		X1 I/O	Multi Purpose X1 with programmable direction. The connection is referenced in the manual as X1
		X2 I/O	Multi Purpose X2 with programmable direction. The connection is referenced in the manual as X2
		X3 I/O	Multi Purpose X3 with programmable direction. The connection is referenced in the manual as X3
8	Control Connections (7 SMA connector version, for M4i module based products))	Clock In	External clock input
		Clock Out	External clock output
		Trig0 In	Trigger 0, input only. This is the main external trigger. The trigger line is reference in the manual as EXT0
		Trig1 In	Trigger 1, input only. This is the secondary external trigger. This line is reference in the manual as EXT1
		X0 I/O	Multi Purpose X0 with programmable direction. The connection is referenced in the manual as X0
		X1 I/O	Multi Purpose X1 with programmable direction. The connection is referenced in the manual as X1
		X2 I/O	Multi Purpose X2 with programmable direction. The connection is referenced in the manual as X2

Front Panel DN6

The right-hand drawing gives you an overview on one digitizerNETBOX DN6 front panel. Depending on the version of the digitizerNETBOX or generatorNETBOX you have, the area 7 may differ in terms of number of channels or grouping of the channels.



In area 8 a version with 4 BNC connectors is shown. Other versions with 5 SMA, 6 BNC or 7 SMA connectors are also available. Please see the table below for the different connections.

Area	Name	Status	Description
1	Power On/Off	press while device stopped	digitizerNETBOX/generatorNETBOX is started
		short press while device is running	digitizerNETBOX/generatorNETBOX is closing the embedded controller and is going into standby mode
		long press while device is running	digitizerNETBOX/generatorNETBOX is aborted and is going into standby. Please only use this stop method if the digitizerNETBOX/generatorNETBOX is not responding
2	Power LED	LED off	no power connected to the device
		LED orange	power is connected, device is in standby mode
		LED green	device has started and is working
3	LAN LED	LED off	Only off during boot up, turning to either red or green afterwards. If permanently off, contact support.
		LED red	Error while trying to get a LAN connection.
		LED green	Device is connected to LAN.
		LED green flashing	Device is connected to LAN. Flashing indicates LAN ID (see webserver).

Area	Name	Status	Description
4	Connected LED	LED off	Device is not in use
		LED green	Device is in use by other PC
5	Arm/Trigger	LED off	No trigger detected, device is waiting for trigger event, or not armed at all
		LED green	Trigger detected, acquisition is running or already finished
6	LAN Reset	press once	Does a reset of the LAN settings to default state. The reset button needs to be pressed for 4 seconds to issue the reset. The reset command is then issued immediatley independent of the current run state of the device.
7	Signal Connections		Connect your input signals here. For differential connections use even channels for positive phase and odd channels for negative phase.
8	Control Connections (4 BNC connector version, for M2i module based products)	Trig-A I/O	Trigger A with programmable input or output. This is the main external trigger
		Trig-B In	Trigger B, input only. This trigger is referenced in the manual as TRIG_XIO0
		Clock I/O	Clock with programmable input or output
		TS-Ref In	Timestamp Reference Clock Input
8	Control Connections (6 BNC connector version, for M2p module based products))	Clock In	External clock input
		Trig In	Trigger, input only. This is the main external trigger. The trigger line is reference in the manual as EXT0
		X0 Out	Multi Purpose X0, output only. Clock output available. The connection is referenced in the manual as X0
		X1 I/O	Multi Purpose X1 with programmable direction. The connection is referenced in the manual as X1
		X2 I/O	Multi Purpose X2 with programmable direction. The connection is referenced in the manual as X2
		X3 I/O	Multi Purpose X3 with programmable direction. The connection is referenced in the manual as X3
8	Control Connections (7 SMA connector version, for M4i module based products)	Clock In	External clock input
		Clock Out	External clock output
		Trig0 In	Trigger 0, input only. This is the main external trigger. The trigger line is reference in the manual as EXT0
		Trig1 In	Trigger 1, input only. This is the secondary external trigger. This line is reference in the manual as EXT1
		X0 I/O	Multi Purpose X0 with programmable direction. The connection is referenced in the manual as X0
		X1 I/O	Multi Purpose X1 with programmable direction. The connection is referenced in the manual as X1
		X2 I/O	Multi Purpose X2 with programmable direction. The connection is referenced in the manual as X2

Ethernet Default Settings

The digitizerNETBOX/generatorNETBOX is started with the following Ethernet configuration:

Setting	Default Setup	
DHCP	enabled	
Auto IP	enabled	
Host Name	Default hostname as netbox type + serial number	Example: DN2_465-08_sn8085

Detecting the digitizerNETBOX

Before accessing the digitizerNETBOX/generatorNETBOX one has to determine the IP address of the digitizerNETBOX/generatorNETBOX. Normally that can be done using one of the two methods described below:

Discovery Function

The digitizerNETBOX/generatorNETBOX responds to the VISA described Discovery function. The next chapter will show how to install and use the Spectrum control center to execute the discovery function and to find the Spectrum hardware. As the discovery function is a standard feature of all LXI devices there are other software packages that can find the digitizerNETBOX/generatorNETBOX using the discovery function:

- Spectrum control center (limited to Spectrum remote products)
- free LXI System Discovery Tool from the LXI consortium (www.lxistandard.org)
- Measurement and Automation Explorer from National Instruments (NI MAX)
- Keysight Connection Expert from Keysight Technologies

Additionally the discovery procedure can also be started from ones own specific application:

```
#define TIMEOUT_DISCOVERY   5000 // timeout value in ms

const uint32 dwMaxNumRemoteCards = 50;

char* pszVisa[dwMaxNumRemoteCards] = { NULL };
char* pszIdn[dwMaxNumRemoteCards]  = { NULL };

const uint32 dwMaxIdnStringLen  = 256;
const uint32 dwMaxVisaStringLen = 50;

// allocate memory for string list
for (uint32 i = 0; i < dwMaxNumRemoteCards; i++)
    {
    pszVisa[i] = new char [dwMaxVisaStringLen];
    pszIdn[i] = new char [dwMaxIdnStringLen];
    memset (pszVisa[i], 0, dwMaxVisaStringLen);
    memset (pszIdn[i], 0, dwMaxIdnStringLen);
    }

// first make discovery - check if there are any LXI compatible remote devices
dwError = spcm_dwDiscovery ((char**)pszVisa, dwMaxNumRemoteCards, dwMaxVisaStringLen, TIMEOUT_DISCOVERY);

// second: check from which manufacturer the devices are
spcm_dwSendIDNRequest ((char**)pszIdn, dwMaxNumRemoteCards, dwMaxIdnStringLen);

// Use the VISA strings of these devices with Spectrum as manufacturer
// for accessing remote devices without previous knowledge of their IP address
```

Finding the digitizerNETBOX/generatorNETBOX in the network

As the digitizerNETBOX/generatorNETBOX is a standard network device it has its own IP address and host name and can be found in the computer network. The standard host name consist of the model type and the serial number of the digitizerNETBOX/generatorNETBOX. The serial number is also found on the type plate on the back of the digitizerNETBOX/generatorNETBOX chassis.

As default DHCP (IPv4) will be used and an IP address will be automatically set. In case no DHCP server is found, an IP will be obtained using the AutoIP feature. This will lead to an IPv4 address of 169.254.x.y (with x and y being assigned to a free IP in the network) using a subnet mask of 255.255.0.0.

The default IP setup can also be restored, by using the „LAN Reset" button on the device.

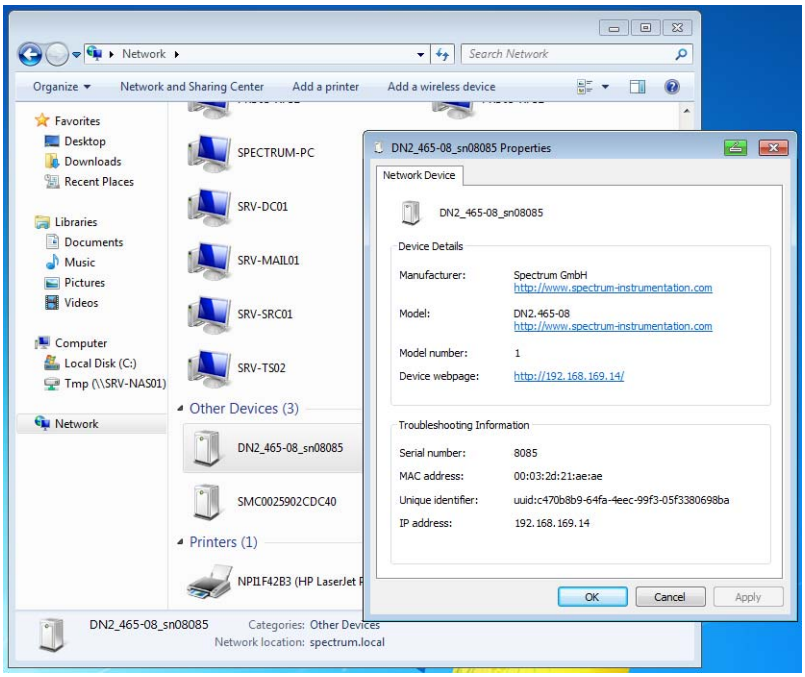
If a fixed IP address should be used instead, the parameters need to be set according to the current LAN requirements.

Windows 7, Windows 8, Windows 10

Under Windows 7, Windows 8 and Windows 10 the digitizerNETBOX and generatorNETBOX devices are listed under the „other devices" tree with their given host name.

A right click on the digitizerNETBOX or generatorNETBOX device opens the properties window where you find further information on the device including the IP address.

From here it is possible to go the website of the device where all necessary information are found to access the device from software.



Troubleshooting

If the above methods do not work please try one of the following steps:

- Ask your network administrator for the IP address of the digitizerNETBOX/generatorNETBOX and access it directly over the IP address.
- Check your local firewall whether it allows access to the device and whether it allows to access the ports listed in the technical data section.
- Check with your network administrator whether the subnet, the device and the ports that are listed in the technical data section are accessible from your system due to company security settings.

Software Driver Installation

Before using the digitizerNETBOX/generatorNETBOX a software package and the appropriate API drivers must be installed that matches the operating system. The installation is done in different ways depending on the used operating system. The driver that is on USB-Stick supports all products of the digitizerNETBOX/generatorNETBOX family as well as all cards of the M2i/M3i/M4i/M4x/M2p series. That means that you can use the same driver for all products of these families.

Needed Software for operating

The digitizerNETBOX/generatorNETBOX comes fully installed and ready to start. However to operate the digitizerNETBOX or generatorNETBOX from the client PC there need to be some software packages to be installed there:

Spectrum driver API

The Spectrum API is installed automatically under Windows when installing the Card Control Center. Under Linux it is necessary to install the matching driver API for your Linux client system before installing the Card Control Center.

Spectrum Card Control Center

This software is the maintenance tool for all Spectrum products. In here the digitizerNETBOX/generatorNETBOX can be searched inside the LAN (Discovery function), all hardware information is found, updates and product tests can be done. The Card Control Center and all of its functions are explained in greater detail later on in this manual.

The card control center is available for Windows and Linux, both 32 bit and 64 bit (Windows 32 bit version also runs on WOW64)

SBench 6

SBench 6 allows to operate the digitizerNETBOX/generatorNETBOX in all hardware modes, displays data, streams to hard disk and allows to make calculations and exports. The digitizerNETBOX/generatorNETBOX is equipped with a full SBench 6 Professional license. Even if you want to operate the digitizerNETBOX/generatorNETBOX from your self written software it is recommended that you install SBench 6 to do first hardware tests and to validate your own software results with the software from the hardware manufacturer. For SBench 6 a dedicated manual is installed with the software package.

SBench 6 is available for Windows and Linux, both 32 bit and 64 bit (Windows 32 bit version also runs on WOW64)

Examples and Drivers

If you intend to operate the digitizerNETBOX/generatorNETBOX from a self written program, be it IVI based, C++, C#, LabVIEW, MATLAB or something else, it is necessary to install the matching drivers and examples for the platform you want to run.

Location

The needed software for operating the digitizerNETBOX/generatorNETBOX can be found on three different locations. Please choose the one most convinient for you.

Install software packages from USB-Stick

The USB-Stick that is delivered together with the digitizerNETBOX/generatorNETBOX contains the complete software and documentation package that is available for your digitizerNETBOX/generatorNETBOX. You find the software packages at the following locations on the USB-Stick:

Software Package	Operating System	Location
Card Control Center	Windows	\Install\Win
SBench 6	Windows	\Install\Win
LabVIEW, MATLAB, IVI	Windows	\Install\Win
C++, C#, VB.NET, Delphi, Python, Java, LabWindows/CVI...	Windows	\Examples\...
Driver API	Linux	/Driver/linux/install_libonly.sh
Card Control Center	Linux	/Install/linux/SBench6
SBench 6	Linux	/Install/linux/spcm_control_center
MATLAB (64bit only)	Linux	/Install/linux
C++, Python, Java	Linux	/Examples/...

Install software packages from the internet

All software packages are found on the downlaod page under www.spectrum-instrumentation.com

Inhere the latest versions and updates are available.

Install software packages from the digitizerNETBOX/generatorNETBOX

For easy installation or for installation on machines that don't have access to a USB thumb drive, all software packages are also available for download directly from the digitizerNETBOX/generatorNETBOX.

Please go to the download page of the integrated webserver and download and execute the software packages.

Linux

Overview

The Spectrum M2i/M3i/M4i/M4x/M2p cards and digitizerNETBOX/generatorNETBOX products are delivered with Linux drivers suitable for Linux installations based on kernel 2.6, 3.x, 4.x or 5.x, single processor (non-SMP) and SMP systems, 32 bit and 64 bit systems. As each Linux distribution contains different kernel versions and different system setup it is in nearly every case necessary, to have a directly matching kernel driver for card level products to run it on a specific system. For digitizerNETBOX/generatorNETBOX products the library is suffcient and no kernel driver has to be installed.

Spectrum delivers pre-compiled kernel driver modules for a number of common distributions with the cards. You may try to use one of these kernel modules for different distributions which have a similar kernel version. Unfortunately this won't work in most cases as most Linux system refuse to load a driver which is not exactly matching. In this case it is possible to get the kernel driver sources from Spectrum. Please contact your local sales representative to get more details on this procedure.

The Standard delivery contains the pre-compiled kernel driver modules for the most popular Linux distributions, like Suse, Debian, Fedora and Ubuntu. The list with all pre-compiled and readily supported distributions and their respective kernel version can be found under:
http://spectrum-instrumentation.com/en/supported-linux-distributions or via the shown QR code.

The Linux drivers have been tested with all above mentioned distributions by Spectrum. Each of these distributions has been installed with the default setup using no kernel updates. A lot more different distributions are used by customers with self compiled kernel driver modules.

Standard Driver Installation

The driver is delivered as installable kernel modules together with libraries to access the kernel driver. The installation script will help you with the installation of the kernel module and the library.

This installation is only needed if you are operating real locally installed cards. For software emulated demo cards, remotely installed cards or for digitizerNETBOX/generatorNETBOX products it is only necessary to install the libraries without a kernel as explained further below.

Login as root

It is necessary to have the root rights for installing a driver.

Call the install.sh <install_path> script

This script will install the kernel module and some helper scripts to a given directory. If you do not specify a directory it will use your home directory as destination. It is possible to move the installed driver files later to any other directory.

The script will give you a list of matching kernel modules. Therefore it checks for the system width (32 bit or 64 bit) and the processor (single or smp). The script will only show matching kernel modules. Select the kernel module matching your system. The script will then do the following steps:

- copy the selected kernel module to the install directory (spcm.o or spcm.ko)
- copy the helper scripts to the install directory (spcm_start.sh and spc_end.sh)
- copy and rename the matching library to /usr/lib (/usr/lib/libspcm_linux.so)

Udev support

Once the driver is loaded it automatically generates the device nodes under /dev. The cards are automatically named to /dev/spcm0, /dev/spcm1,...

You may use all the standard naming and rules that are available with udev.

Start the driver

Starting the driver can be done with the spcm_start.sh script that has been placed in the install directory. If udev is installed the script will only load the driver. If no udev is installed the start script will load the driver and make the required device nodes /dev/spcm0... for accessing the drivers. Please keep in mind that you need root rights to load the kernel module and to make the device nodes!

Using the dedicated start script makes sure that the device nodes are matching your system setup even if new hardware and drivers have been added in between. Background: when loading the device driver it gets assigned a „major" number that is used to access this driver. All device nodes point to this major number instead of the driver name. The major numbers are assigned first come first served. This means that installing new hardware may result in different major numbers on the next system start.

Get first driver info

After the driver has been loaded successfully some information about the installed boards can be found in the /proc/spcm_cards file. Some basic information from the on-board EEProm is listed for every card.

```
cat /proc/spcm_cards
```

Stop the driver

You may want to unload the driver and clean up all device nodes. This can be done using the spcm_end.sh script that has also been placed in the install directory

Standard Driver Update

A driver update is done with the same commands as shown above. Please make sure that the driver has been stopped before updating it. To stop the driver you may use the spcm_end.sh script.

Compilation of kernel driver sources (optional and local cards only)

The driver sources are only available for existing customers on special request and against a signed NDA. The driver sources are not part of the standard delivery. The driver source package contains only the sources of the kernel module, not the sources of the library.

Please do the following steps for compilation and installation of the kernel driver module:

Login as root

It is necessary to have the root rights for installing a driver.

Call the compile script make_spcm_linux_kerneldrv.sh

This script will examine the type of system you use and compile the kernel with the correct settings. If using a kernel 2.4 the makefile expects two symbolic links in your system:

- /usr/src/linux pointing to the correct kernel source directory
- /usr/src/linux/.config pointing to the currently used kernel configuration

The compile script will then automatically call the install script and install the just compiled kernel module in your home directory. The rest of the installation procedure is similar as explained above.

Update of a self compiled kernel driver

If the kernel driver has changed, one simply has to perform the same steps as shown above and recompile the kernel driver module. However the kernel driver module isn't changed very often.

Normally an update only needs new libraries. To update the libraries only you can either download the full Linux driver (spcm_linux_drv_v123b4567) and only use the libraries out of this or one downloads the library package which is much smaller and doesn't contain the pre-compiled kernel driver module (spcm_linux_lib_v123b4567).

The update is done with a dedicated script which only updates the library file. This script is present in both driver archives:

```
sh install_libonly.sh
```

Installing the library only without a kernel (for remote devices)

The kernel driver module only contains the basic hardware functions that are necessary to access locally installed card level products. The main part of the driver is located inside a dynamically loadable library that is delivered with the driver. This library is available in 3 different versions:

- spcm_linux_32bit_stdc++6.so - supporting libstdc++.so.6 on 32 bit systems
- spcm_linux_64bit_stdc++6.so - supporting libstdc++.so.6 on 64 bit systems

The matching version is installed automatically in the /usr/lib directory by the kernel driver install script for card level products. The library is renamed for easy access to libspcm_linux.so.

For digitizerNETBOX/generatorNETBOX products and also for evaluating or using only the software simulated demo cards the library is installed with a separate install script:

```
sh install_libonly.sh
```

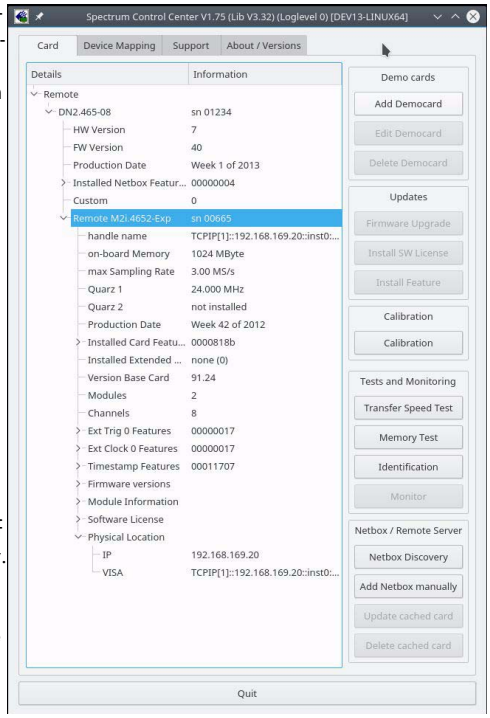
To access the driver library one must include the library in the compilation:

```
gcc -o test_prg -lspcm_linux test.cpp
```

To start programming the cards under Linux please use the standard C/C++ examples which are all running under Linux and Windows.

Control Center

The Spectrum Control Center is also available for Linux and needs to be installed separately. The features of the Control Center are described in a later chapter in deeper detail. The Control Center has been tested under all Linux distributions for which Spectrum delivers pre-compiled kernel modules. The following packages need to be installed to run the Control Center:



- X-Server
- expat
- freetype
- fontconfig
- libpng
- libspcm_linux (the Spectrum linux driver library)

Installation

Use the supplied packages in either *.deb or *.rpm format found in the driver section of the USB-Stick by double clicking the package file root rights from a X-Windows window.

The Control Center is installed under KDE, Gnome or Unity in the system/system tools section. It may be located directly in this menu or under a „More Programs" menu. The final location depends on the used Linux distribution. The program itself is installed as /usr/bin/spcmcontrol and may be started directly from here.

Manual Installation

To manually install the Control Center, first extract the files from the rpm matching your distribution:

```
rpm2cpio spcmcontrol-{Version}.rpm > ~/spcmcontrol-{Version}.cpio
cd ~/
cpio -id < spcmcontrol-{Version}.cpio
```

You get the directory structure and the files contained in the rpm package. Copy the binary spcmcontrol to /usr/bin. Copy the .desktop file to /usr/share/applications. Run ldconfig to update your systems library cache. Finally you can run spcmcontrol.

Troubleshooting

If you get a message like the following after starting spcmcontrol:

```
spcm_control: error while loading shared libraries: libz.so.1: cannot open shared object file: No such file
or directory
```

Run ldd spcm_control in the directory where spcm_control resides to see the dependencies of the program. The output may look like this:

```
libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x4019e000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x401ad000)
libz.so.1 => not found
libdl.so.2 => /lib/libdl.so.2 (0x402ba000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x402be000)
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0x402d0000)
```

As seen in the output, one of the libraries isn't found inside the library cache of the system. Be sure that this library has been properly installed. You may then run ldconfig. If this still doesn't help please add the library path to /etc/ld.so.conf and run ldconfig again.

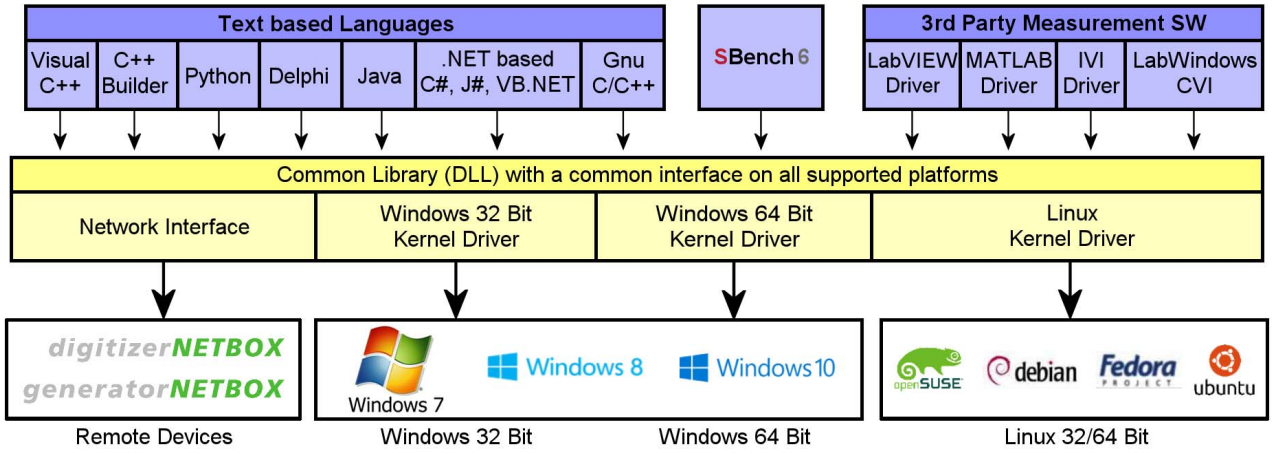
If the libspcm_linux.so is quoted as missing please make sure that you have installed the card driver properly before. If any other library is stated as missing please install the matching package of your distribution.

Software

This chapter gives you an overview about the structure of the drivers and the software, where to find and how to use the examples. It shows in detail, how the drivers are included using different programming languages and deals with the differences when calling the driver functions from them.

This manual only shows the use of the standard driver API. For further information on programming drivers for third-party software like LabVIEW, MATLAB or IVI an additional manual is required that is available on USB-Stick or by download on the internet.

Software Overview



The Spectrum drivers offer you a common and fast API for using all of the board hardware features. This API is the same on all supported operating systems. Based on this API one can write own programs using any programming language that can access the driver API. This manual describes in detail the driver API, providing you with the necessary information to write your own programs.
The drivers for third-party products like LabVIEW or MATLAB are also based on this API. The special functionality of these drivers is not subject of this document and is described with separate manuals available on the USB-Stick or on the website.

Card Control Center

A special card control center is available on USB-Stick and from the internet for all Spectrum M2i/M3i/M4i/M4x/M2p cards and for all digitizerNETBOX or generatorNETBOX products. Windows users find the Control Center installer on the USB-Stick under „Install\win\spcmcontrol_install.exe".

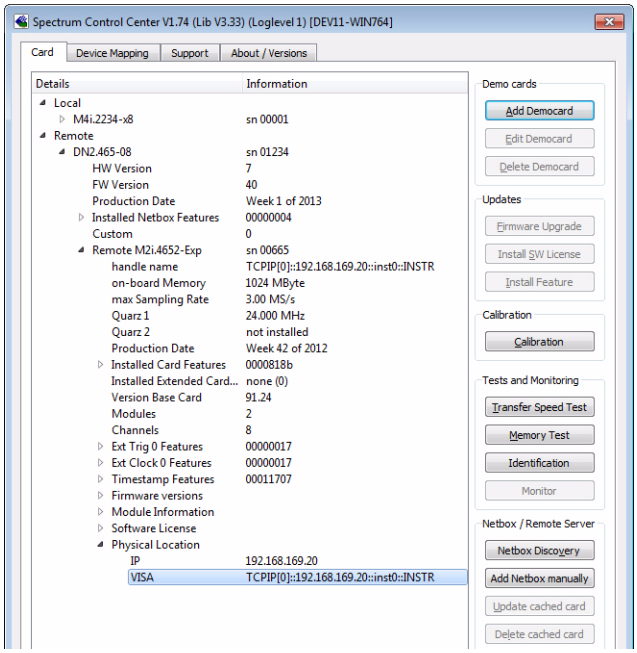
Linux users find the versions for the different stdc++ libraries under /Install/linux/spcm_control_center/ as RPM packages.

When using a digitizerNETBOX/generatorNETBOX the Card Control Center installers for Windows and Linux are also directly available from the integrated webserver.

The Control Center under Windows and Linux is available as an executive program. Under Windows it is also linked as a system control and can be accessed directly from the Windows control panel. Under Linux it is also available from the KDE System Settings, the Gnome or Unity Control Center. The different functions of the Spectrum card control center are explained in detail in the following passages.

To install the Spectrum Control Center you will need to be logged in with administrator rights for your operating system. On all Windows versions, starting with Windows Vista, installations with enabled UAC will ask you to start the installer with administrative rights (run as administrator).

Discovery of Remote Cards and digitizerNETBOX/generatorNETBOX products



The Discovery function helps you to find and identify the Spectrum LXI instruments like digitizerNETBOX/generatorNETBOX available to your computer on the network. The Discovery function will also locate Spectrum card products handled by an installed Spectrum Remote Server somewhere on the network. The function is not needed if you only have locally installed cards.

Please note that only remote products are found that are currently not used by another program. Therefore in a bigger network the number of Spectrum products found may vary depending on the current usage of the products.

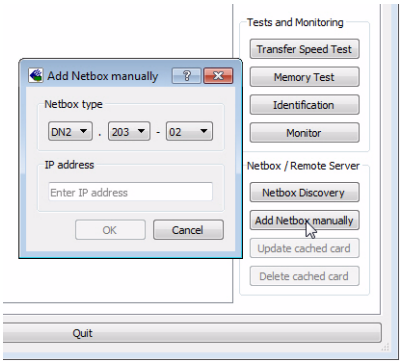
Execute the Discovery function by pressing the „Discovery" button. There is no progress window shown. After the discovery function has been executed the remotely found Spectrum products are listed under the node Remote as separate card level products. Inhere you find all hardware information as shown in the next topic and also the needed VISA resource string to access the remote card.

Please note that these information is also stored on your system and allows Spectrum software like SBench 6 to access the cards directly once found with the Discovery function.

After closing the control center and re-opening it the previously found remote products are shown with the prefix cached, only showing the card type and the serial number. This is the stored information that allows other Spectrum products to access previously found cards. Using the „Update cached cards" button will try to re-open these cards and gather information of it. Afterwards the remote cards may disappear if they're in use from somewhere else or the complete information of the remote products is shown again.

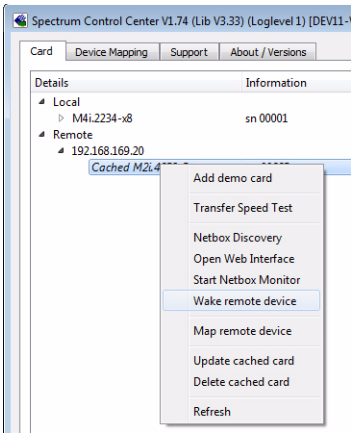
Enter IP Address of digitizerNETBOX/generatorNETBOX manually

If for some reason an automatic discovery is not suitable, such as the case where the remote device is located in a different subnet, it can also be manually acessed by its type and IP address.



Wake On LAN of digitizerNETBOX/generatorNETBOX

Cached digitizerNETBOX/generatorNETBOX products that are currently in standby mode can be woken up by using the „Wake remote device" entry from the context menu.



The Control Center will broadcast a standard Wake On LAN „Magic Packet", that is sent to the device's MAC address.

It is also possible to use any other Wake On LAN software to wake a digitizerNETBOX by sending such a „Magic Packet" to the MAC address, which must be then entered manually.

It is also possible to wake a digitizerNETBOX/generatorNETBOX from your own application software by using the SPC_NETBOX_WAKEONLAN register. To wake a digitizerNETBOX/generatorNETBOX with the MAC address „00:03:2d:20:48", the following command can be issued:

```
spcm_dwSetParam_i64 (NULL, SPC_NETBOX_WAKEONLAN, 0x00032d2048ec);
```

Netbox Monitor

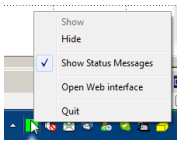
The Netbox Monitor permanently monitors whether the digitizerNETBOX/generatorNETBOX is still available through LAN. This tool is helpful if the digitizerNETBOX is located somewhere in the company LAN or located remotely or directly mounted inside another device. Starting the Netbox Monitor can be done in two different ways:

- Starting manually from the Spectrum Control Center using the context menu as shown above
- Starting from command line. The Netbox Monitor program is automatically installed together with the Spectrum Control Center and is located in the selected install folder. Using the command line tool one can place a simple script into the autostart folder to have the Netbox Monitor running automatically after system boot. The command line tool needs the IP address of the digitizerNETBOX/generatorNETBOX to monitor:

```
NetboxMonitor 192.168.169.22
```

The Netbox Monitor is shown as a small window with the type of digitizerNETBOX/generatorNETBOX in the title and the IP address under which it is accessed in the window itself. The Netbox Monitor runs completely independent of any other software and can be used in parallel to any application software. The background of the IP address is used to display the current status of the device. Pressing the Escape key or alt + F4 (Windows) terminates the Netbox Monitor permanently.

After starting the Netbox Monitor it is also displayed as a tray icon under Windows. The tray icon itself shows the status of the digitizerNETBOX/generatorNETBOX as a color. Please note that the tray icon may be hidden as a Windows default and need to be set to visible using the Windows tray setup.



Left clicking on the tray icon will hide/show the small Netbox Monitor status window. Right clicking on the tray icon as shown in the picture on the right will open up a context menu. In here one can again select to hide/show the Netbox Monitor status window, one can directly open the web interface from here or quit the program (including the tray icon) completely.

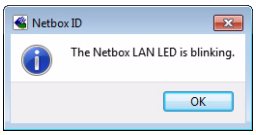
The checkbox „Show Status Message" controls whether the tray icon should emerge a status message on status change. If enabled (which is default) one is notified with a status message if for example the LAN connection to the digitizerNETBOX/generatorNETBOX is lost.

The status colors:

- Green: digitizerNETBOX/generatorNETBOX available and accessible over LAN
- Cyan: digitizerNETBOX/generatorNETBOX is used from my computer
- Yellow: digitizerNETBOX/generatorNETBOX is used from a different computer
- Red: LAN connection failed, digitizerNETBOX/generatorNETBOX is no longer accessible

Device identification

Pressing the *Identification* button helps to identify a certain device in either a remote location, such as inside a 19" rack where the back of the device with the type plate is not easily accessible, or a local device installed in a certain slot. Pressing the button starts flashing a visible LED on the device, until the dialog is closed, for:



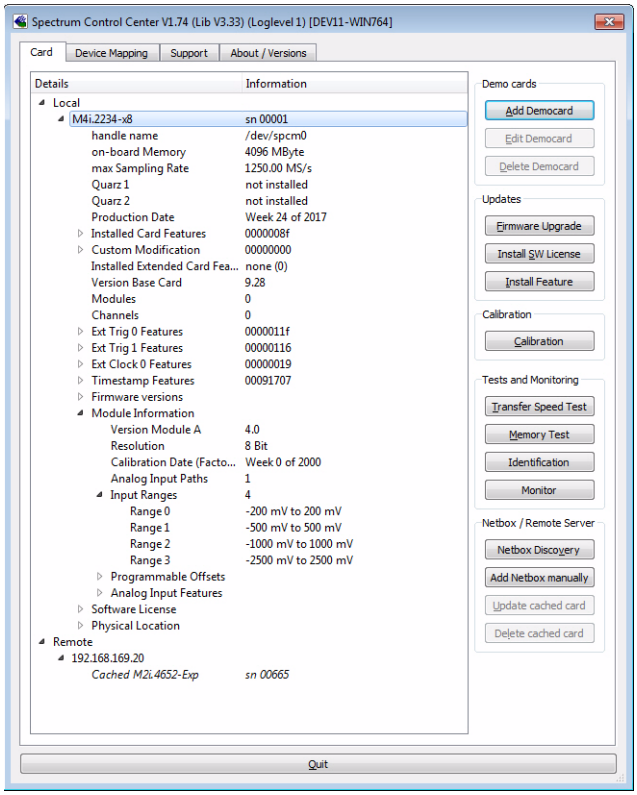
- On a digitizerNETBOX or generatorNETBOX: the LAN LED light on the front plate of the device
- On local or remote M4i, M4x or M2p card: the indicator LED on the card's bracket

This feature is not available for M2i/M3i cards, either local or remote, other than inside a digitizerNETBOX or generatorNETBOX.

Hardware information

Through the control center you can easily get the main information about all the installed Spectrum hardware. For each installed card there is a separate tree of information available. The picture shows the information for one installed card by example. This given information contains:

- Basic information as the type of card, the production date and its serial number, as well as the installed memory, the hardware revision of the base card, the number of available channels and installed acquisition modules.
- Information about the maximum sampling clock and the available quartz clock sources.
- The installed features/options in a sub-tree. The shown card is equipped for example with the option Multiple Recording, Gated Sampling, Timestamp and ABA-mode.
- Detailed Information concerning the installed acquisition modules. In case of the shown analog acquisition card the information consists of the module's hardware revision, of the converter resolution and the last calibration date as well as detailed information on the available analog input ranges, offset compensation capabilities and additional features of the inputs.



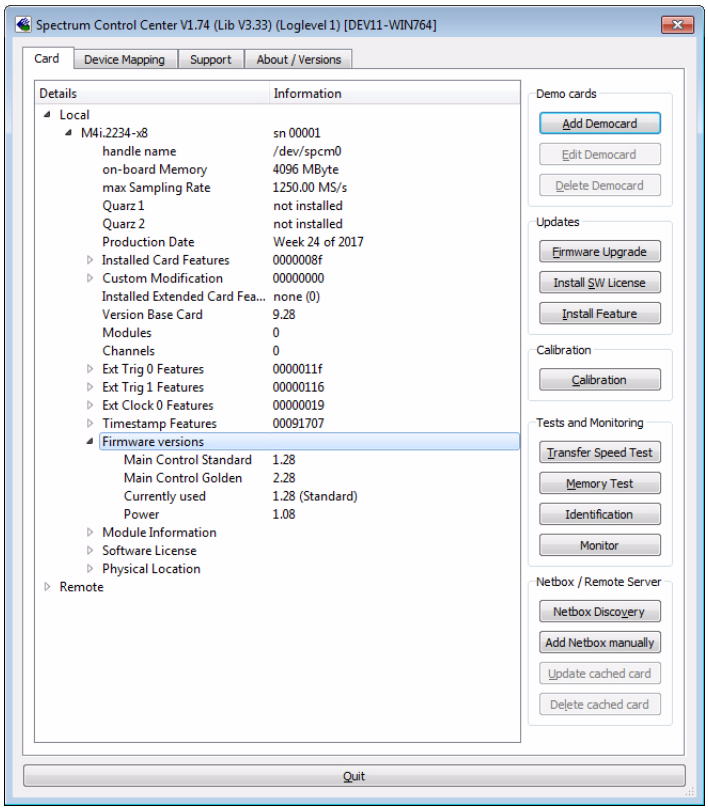
Firmware information

Another sub-tree is informing about the cards firmware version. As all Spectrum cards consist of several programmable components, there is one firmware version per component.

Nearly all of the components firmware can be updated by software. The only exception is the configuration device, which only can receive a factory update.

The procedure on how to update the firmware of your Spectrum card with the help of the card control center is described in a dedicated section later on.

The procedure on how to update the firmware of your digitizerNETBOX/generatorNETBOX with the help of the integrated Webserver is described in a dedicated chapter later on.

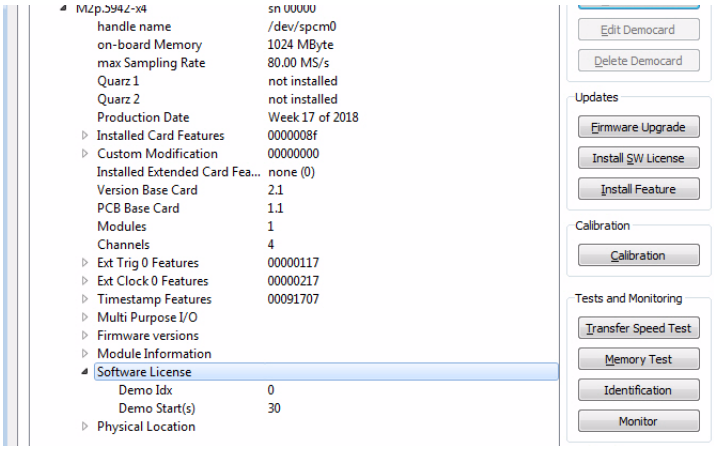


Software License information

This sub-tree is informing about installed possible software licenses.

As a default all cards come with the demo professional license of SBench6, that is limited to 30 starts of the software with all professional features unlocked.

The number of demo starts left can be seen here.



Driver information

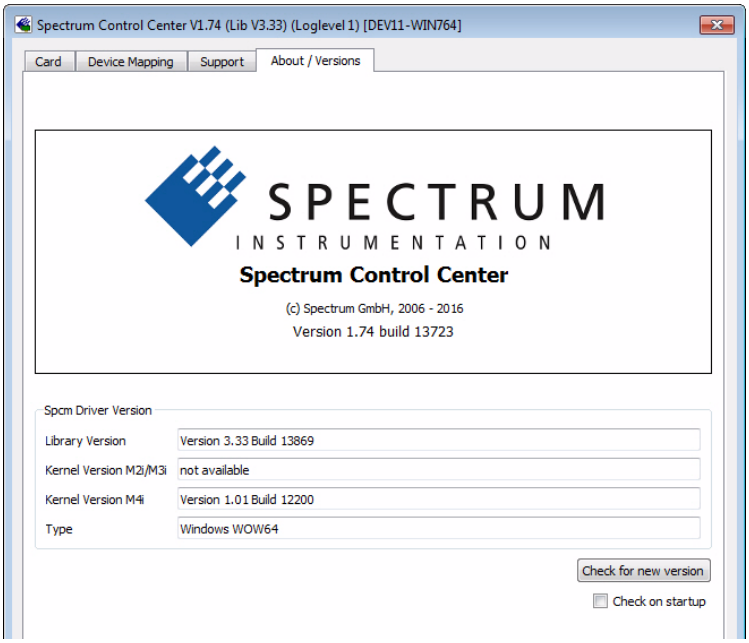
The Spectrum card control center also offers a way to gather information on the installed and used Spectrum driver.

The information on the driver is available through a dedicated tab, as the picture is showing in the example.

The provided information informs about the used type, distinguishing between Windows or Linux driver and the 32 bit or 64 bit type.

It also gives direct information about the version of the installed Spectrum kernel driver, separately for M2i/ M3i cards and M4i/M4x/M2p cards and the version of the library (which is the *.dll file under Windows).

The information given here can also be found under Windows using the device manager form the control panel. For details in driver details within the control panel please stick to the section on driver installation in your hardware manual.

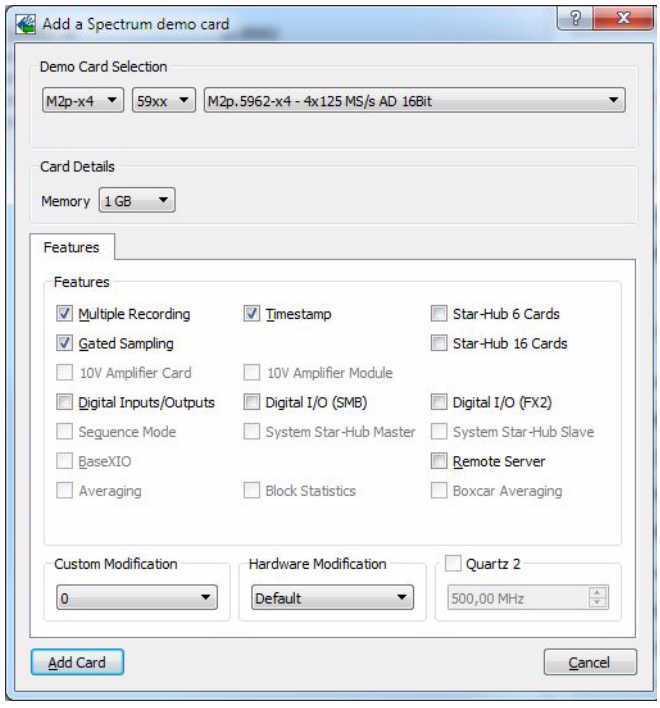


Installing and removing Demo cards

With the help of the card control center one can install demo cards in the system. A demo card is simulated by the Spectrum driver including data production for acquisition cards. As the demo card is simulated on the lowest driver level all software can be tested including SBench, own applications and drivers for third-party products like LabVIEW. The driver supports up to 64 demo cards at the same time. The simulated memory as well as the simulated software options can be defined when adding a demo card to the system.

Please keep in mind that these demo cards are only meant to test software and to show certain abilities of the software. They do not simulate the complete behavior of a card, especially not any timing concerning trigger, recording length or FIFO mode notification. The demo card will calculate data every time directly after been called and give it to the user application without any more delay. As the calculation routine isn't speed optimized, generating demo data may take more time than acquiring real data and transferring them to the host PC.

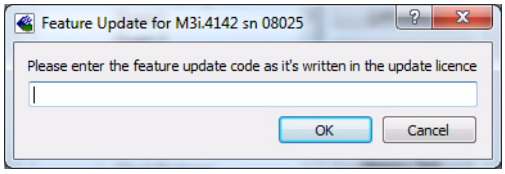
Installed demo cards are listed together with the real hardware in the main information tree as described above. Existing demo cards can be deleted by clicking the related button. The demo card details can be edited by using the edit button. It is for example possible to virtually install additional feature to one card or to change the type to test with a different number of channels.



For installing demo cards on a system without real hardware simply run the Control Center installer. If the installer is not detecting the necessary driver files normally residing on a system with real hardware, it will simply install the Spcm_driver.

Feature upgrade

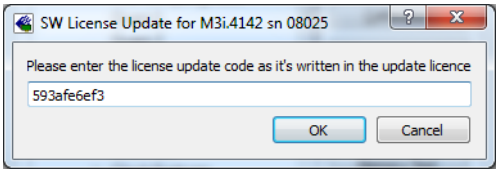
All optional features of the M2i/M3i/M4i/M4x/M2p cards that do not require any hardware modifications can be installed on fielded cards. After Spectrum has received the order, the customer will get a personalized upgrade code. Just start the card control center, click on „install feature" and enter that given code. After a short moment the feature will be installed and ready to use. No restart of the host system is required.



For details on the available options and prices please contact your local Spectrum distributor.

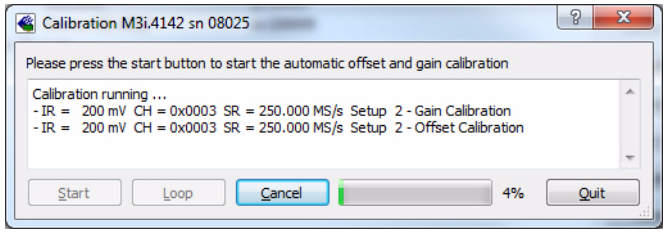
Software License upgrade

The software license for SBench 6 Professional is installed on the hardware. If ordering a software license for a card that has already been delivered you will get an upgrade code to install that software license. The upgrade code will only match for that particular card with the serial number given in the license. To install the software license please click the „Install SW License" button and type in the code exactly as given in the license.



Performing card calibration

The card control center also provides an easy way to access the automatic card calibration routines of the Spectrum A/D converter cards. Depending on the used card family this can affect offset calibration only or also might include gain calibration. Please refer to the dedicated chapter in your hardware manual for details.

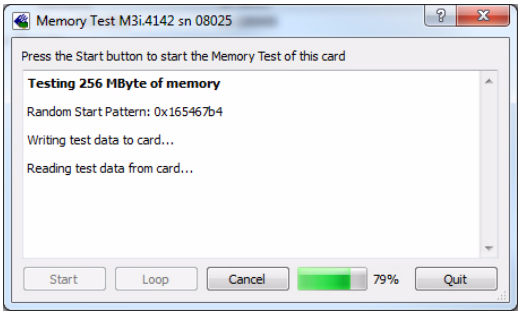


Performing memory test

The complete on-board memory of the Spectrum M2i/M3i/M4i/M4x/M2p cards can be tested by the memory test included with the card control center.

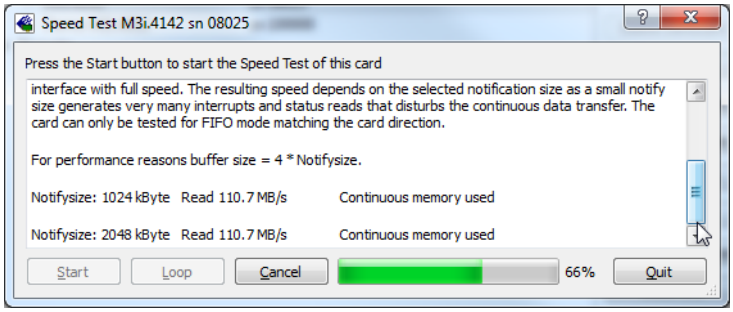
When starting the test, randomized data is generated and written to the on-board memory. After a complete write cycle all the data is read back and compared with the generated pattern.

Depending on the amount of installed on-board memory, and your computer's performance this operation might take a while.



Transfer speed test

The control center allows to measure the bus transfer speed of an installed Spectrum card. Therefore different setup is run multiple times and the overall bus transfer speed is measured. To get reliable results it is necessary that you disable debug logging as shown below. It is also highly recommended that no other software or time-consuming background threads are running on that system. The speed test program runs the following two tests:



- Repetitive Memory Transfers: single DMA data transfers are repeated and measured. This test simulates the measuring of pulse repetition frequency when doing multiple single-shots. The test is done using different block sizes. One can estimate the transfer in relation to the transferred data size on multiple single-shots.
- FIFO mode streaming: this test measures the streaming speed in FIFO mode. The test can only use the same direction of transfer the card has been designed for (card to PC=read for all DAQ cards, PC to card=write for all generator cards and both directions for I/O cards). The streaming speed is tested without using the front-end to measure the maximum bus speed that can be reached.
  The Speed in FIFO mode depends on the selected notify size which is explained later in this manual in greater detail.

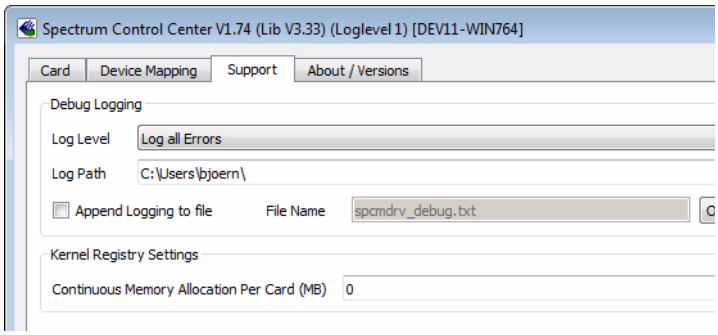
The results are given in MB/s meaning MByte per second. To estimate whether a desired acquisition speed is possible to reach one has to calculate the transfer speed in bytes. There are a few things that have to be put into the calculation:

- 12, 14 and 16 bit analog cards need two bytes for each sample.
- 16 channel digital cards need 2 bytes per sample while 32 channel digital cards need 4 bytes and 64 channel digital cards need 8 bytes.
- The sum of analog channels must be used to calculate the total transfer rate.
- The figures in the Speed Test Utility are given as MBytes, meaning 1024 * 1024 Bytes, 1 MByte = 1048576 Bytes

As an example running a card with 2 14 bit analog channels with 28 MHz produces a transfer rate of [2 channels * 2 Bytes/Sample * 28000000] = 112000000 Bytes/second. Taking the above figures measured on a standard 33 MHz PCI slot the system is just capable of reaching this transfer speed: 108.0 MB/s = 108 * 1024 * 1024 = 113246208 Bytes/second.

Unfortunately it is not possible to measure transfer speed on a system without having a Spectrum card installed.

Debug logging for support cases



For answering your support questions as fast as possible, the setup of the card, driver and firmware version and other information is very helpful.

Therefore the card control center provides an easy way to gather all that information automatically.

Different debug log levels are available through the graphical interface. By default the log level is set to „no logging" for maximum performance.

The customer can select different log levels and the path of the generated ASCII text file. One can also decide to delete the previous log file first before creating a new one automatically or to append different logs to one single log file.

For maximum performance of your hardware, please make sure that the debug logging is set to „no logging" for normal operation. Please keep in mind that a detailed logging in append mode can quickly generate huge log files.

Device mapping

Within the „Device mapping" tab of the Spectrum Control Center, one can enable the re-mapping of Spectrum devices, be it either local cards, remote instruments such as a digitizerNETBOX or generatorNETBOX or even cards in a remote PC and accessed via the Spectrum remote server option.

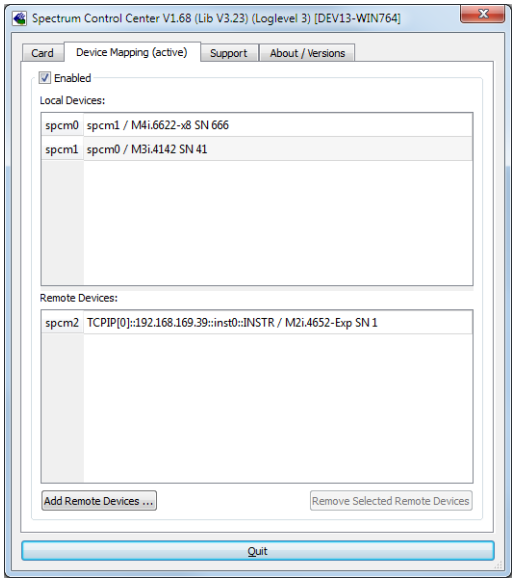
In the left column the re-mapped device name is visible that is given to the device in the right column with its original un-mapped device string.

In this example the two local cards „spcm0" and „spcm1" are re-mapped to „spcm1" and „spcm0" respectively, so that their names are simply swapped.

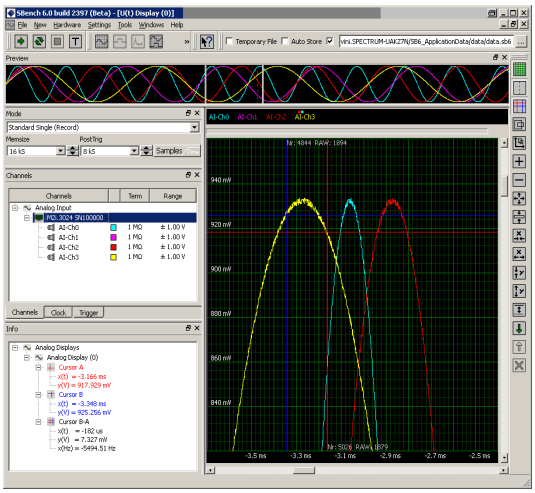
The remote digitizerNETBOX device is mapped to spcm2.

The application software can then use the re-mapped name for simplicity instead of the quite long VISA string.

Changing the order of devices within one group (either local cards or remote devices) can simply be accomplished by draging&dropping the cards to their desired position in the same table.



Accessing the hardware with SBench 6



After the installation of the cards and the drivers it can be useful to first test the card function with a ready to run software before starting with programming. If accessing a digitizerNETBOX/generatorNETBOX a full SBench 6 Professional license is installed on the system and can be used without any limitations. For plug-in card level products a base version of SBench 6 is delivered with the card on USB-Stick also including a 30 starts Professional demo version for plain card products. If you already have bought a card prior to the first SBench 6 release please contact your local dealer to get a SBench 6 Professional demo version. All digitizerNETBOX/generatorNETBOX products come with a pre-installed full SBench 6 Professional.

SBench 6 supports all current acquisition and generation cards and digitizerNETBOX/generatorNETBOX products from Spectrum. Depending on the used product and the software setup, one can use SBench as a digital storage oscilloscope, a spectrum analyzer, a signal generator, a pattern generator, a logic analyzer or simply as a data recording front end. Different export and import formats allow the use of SBench 6 together with a variety of other programs.

On the USB-Stick you'll find an install version of SBench 6 in the directory „/Install/SBench6".

The current version of SBench 6 is available free of charge directly from the Spectrum website: www.spectrum-instrumentation.com. Please go to the download section and get the latest version there.

SBench 6 has been designed to run under Windows 7, Windows 8 and Windows 10 as well as Linux using KDE, Gnome or Unity Desktop.

C/C++ Driver Interface

C/C++ is the main programming language for which the drivers have been designed for. Therefore the interface to C/C++ is the best match. All the small examples of the manual showing different parts of the hardware programming are done with C. As the libraries offer a standard interface it is easy to access the libraries also with other programming languages like Delphi, Basic, Python or Java . Please read the following chapters for additional information on this.

Header files

The basic task before using the driver is to include the header files that are delivered on USB-Stick together with the board. The header files are found in the directory /Driver/c_header. Please don't change them in any way because they are updated with each new driver version to include the new registers and new functionality.

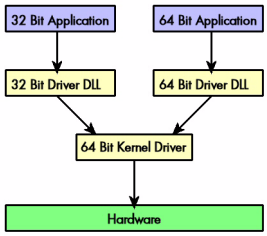
dlltyp.h	Includes the platform specific definitions for data types and function declarations. All data types are based on these definitions. The use of this type definition file allows the use of examples and programs on different platforms without changes to the program source. The header file supports Microsoft Visual C++, Borland C++ Builder and GNU C/C++ directly. When using other compilers it might be necessary to make a copy of this file and change the data types according to this compiler.
regs.h	Defines all registers and commands which are used in the Spectrum driver for the different boards. The registers a board uses are described in the board specific part of the documentation. This header file is common for all cards. Therefore this file also contains a huge number of registers used on other card types than the one described in this manual. Please stick to the manual to see which registers are valid for your type of card.
spcm_drv.h	Defines the functions of the used SpcM driver. All definitions are taken from the file dlltyp.h. The functions themselves are described below.
spcerr.h	Contains all error codes used with the Spectrum driver. All error codes that can be given back by any of the driver functions are also described here briefly. The error codes and their meaning are described in detail in the appendix of this manual.

Example for including the header files:

```
// ----- driver includes -----
#include "dlltyp.h"       // 1st include
#include "regs.h"         // 2nd include
#include "spcerr.h"       // 3rd include
#include "spcm_drv.h"     // 4th include
```

Please always keep the order of including the four Spectrum header files. Otherwise some or all of the functions do not work properly or compiling your program will be impossible!

General Information on Windows 64 bit drivers



After installation of the Spectrum 64 bit driver there are two general ways to access the hardware and to develop applications. If you're going to develop a real 64 bit application it is necessary to access the 64 bit driver dll (spcm_win64.dll) as only this driver dll is supporting the full 64 bit address range.

But it is still possible to run 32 bit applications or to develop 32 bit applications even under Windows 64 bit. Therefore the 32 bit driver dll (spcm_win32.dll) is also installed in the system. The Spectrum SBench5 software is for example running under Windows 64 bit using this driver. The 32 bit dll of course only offers the 32 bit address range and is therefore limited to access only 4 GByte of memory. Beneath both drivers the 64 bit kernel driver is running.

Mixing of 64 bit application with 32 bit dll or vice versa is not possible.

Microsoft Visual C++ 6.0, 2005 and newer 32 Bit

Include Driver

The driver files can be directly included in Microsoft C++ by simply using the library file spcm_win32_msvcpp.lib that is delivered together with the drivers. The library file can be found on the USB-Stick in the path /examples/c_cpp/c_header. Please include the library file in your Visual C++ project as shown in the examples. All functions described below are now available in your program.

Examples

Examples can be found on USB-Stick in the path /examples/c_cpp. This directory includes a number of different examples that can be used with any card of the same type (e.g. A/D acquisition cards, D/A acquisition cards). You may use these examples as a base for own programming and modify them as you like. The example directories contain a running workspace file for Microsoft Visual C++ 6.0 (*.dsw) as well as project files for Microsoft Visual Studio 2005 and newer (*.vcproj) that can be directly loaded or imported and compiled.
There are also some more board type independent examples in separate subdirectory. These examples show different aspects of the cards like programming options or synchronization and can be combined with one of the board type specific examples.

As the examples are build for a card class there are some checking routines and differentiation between cards families. Differentiation aspects can be number of channels, data width, maximum speed or other details. It is recommended to change the examples matching your card type to obtain maximum performance. Please be informed that the examples are made for easy understanding and simple showing of one aspect of programming. Most of the examples are not optimized for maximum throughput or repetition rates.

Microsoft Visual C++ 2005 and newer 64 Bit

Depending on your version of the Visual Studio suite it may be necessary to install some additional 64 bit components (SDK) on your system. Please follow the instructions found on the MSDN for further information.

Include Driver

The driver files can be directly included in Microsoft C++ by simply using the library file spcm_win64_msvcpp.lib that is delivered together with the drivers. The library file can be found on the USB-Stick in the path /examples/c_cpp/c_header. All functions described below are now available in your program.

C++ Builder 32 Bit

Include Driver

The driver files can be easily included in C++ Builder by simply using the library file spcm_win32_bcppb.lib that is delivered together with the drivers. The library file can be found on the USB-Stick in the path /examples/c_cpp/c_header. Please include the library file in your C++ Builder project as shown in the examples. All functions described below are now available in your program.

Examples

The C++ Builder examples share the sources with the Visual C++ examples. Please see above chapter for a more detailed documentation of the examples. In each example directory are project files for Visual C++ as well as C++ Builder.

Linux Gnu C/C++ 32/64 Bit

Include Driver

The interface of the linux drivers does not differ from the windows interface. Please include the spcm_linux.lib library in your makefile to have access to all driver functions. A makefile may look like this:

```
COMPILER =   gcc
EXECUTABLE = test_prg
LIBS = -lspcm_linux

OBJECTS =   test.o\
            test2.o

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    $(COMPILER) $(CFLAGS) -o $(EXECUTABLE) $(LIBS) $(OBJECTS)

%.o: %.cpp
    $(COMPILER) $(CFLAGS) -o $*.o -c $*.cpp
```

Examples

The Gnu C/C++ examples share the source with the Visual C++ examples. Please see above chapter for a more detailed documentation of the examples. Each example directory contains a makefile for the Gnu C/C++ examples.

C++ for .NET

Please see the next chapter for more details on the .NET inclusion.

Other Windows C/C++ compilers 32 Bit

Include Driver

To access the driver, the driver functions must be loaded from the 32 bit driver DLL. Most compilers offer special tools to generate a matching library (e.g. Borland offers the implib tool that generates a matching library out of the windows driver DLL). If such a tool is available it is recommended to use it. Otherwise the driver functions need to be loaded from the dll using standard Windows functions. There is one example in the example directory /examples/c_cpp/dll_loading that shows the process.

Example of function loading:

```
hDLL = LoadLibrary ("spcm_win32.dll"); // Load the 32 bit version of the Spcm driver
pfn_spcm_hOpen =  (SPCM_HOPEN*)  GetProcAddress (hDLL, "_spcm_hOpen@4");
pfn_spcm_vClose = (SPCM_VCLOSE*) GetProcAddress (hDLL, "_spcm_vClose@4");
```

Other Windows C/C++ compilers 64 Bit

Include Driver

To access the driver, the driver functions must be loaded from the 64 bit the driver DLL. Most compilers offer special tools to generate a matching library (e.g. Borland offers the implib tool that generates a matching library out of the windows driver DLL). If such a tool is available it is recommended to use it. Otherwise the driver functions need to be loaded from the dll using standard Windows functions. There is one example in the example directory /examples/c_cpp/dll_loading that shows the process for 32 bit environments. The only line that needs to be modified is the one loading the DLL:

Example of function loading:

```
hDLL = LoadLibrary ("spcm_win64.dll"); // Modified: Load the 64 bit version of the Spcm driver here
pfn_spcm_hOpen =  (SPCM_HOPEN*)  GetProcAddress (hDLL, "spcm_hOpen");
pfn_spcm_vClose = (SPCM_VCLOSE*) GetProcAddress (hDLL, "spcm_vClose");
```

Driver functions

The driver contains seven main functions to access the hardware.

Own types used by our drivers

To simplify the use of the header files and our examples with different platforms and compilers and to avoid any implicit type conversions we decided to use our own type declarations. This allows us to use platform independent and universal examples and driver interfaces. If you do not stick to these declarations please be sure to use the same data type width. However it is strongly recommended that you use our defined

type declarations to avoid any hard to find errors in your programs. If you're using the driver in an environment that is not natively supported by our examples and drivers please be sure to use a type declaration that represents a similar data width

Declaration	Type
int8	8 bit signed integer (range from -128 to +127)
int16	16 bit signed integer (range from -32768 to 32767)
int32	32 bit signed integer (range from -2147483648 to 2147483647)
int64	64 bit signed integer (full range)
drv_handle	handle to driver, implementation depends on operating system platform

Declaration	Type
uint8	8 bit unsigned integer (range from 0 to 255)
uint16	16 bit unsigned integer (range from 0 to 65535)
uint32	32 bit unsigned integer (range from 0 to 4294967295)
uint64	64 bit unsigned integer (full range)

Notation of variables and functions

In our header files and examples we use a common and reliable form of notation for variables and functions. Each name also contains the type as a prefix. This notation form makes it easy to see implicit type conversions and minimizes programming errors that result from using incorrect types. Feel free to use this notation form for your programs also-

Declaration	Notation
int8	byName (byte)
int16	nName
int32	lName (long)
int64	llName (long long)
int32*	plName (pointer to long)

Declaration	Notation
uint8	cName (character)
uint16	wName (word)
uint32	dwName (double word)
uint64	qwName (quad word)
char	szName (string with zero termination)

Function spcm_hOpen

This function initializes and opens an installed card supporting the new SpcM driver interface, which at the time of printing, are all cards of the M2i/M3i/M4i/M4x/M2p series and the related digitizerNETBOX/generatorNETBOX devices. The function returns a handle that has to be used for driver access. If the card can't be found or the loading of the driver generated an error the function returns a NULL. When calling this function all card specific installation parameters are read out from the hardware and stored within the driver. It is only possible to open one device by one software as concurrent hardware access may be very critical to system stability. As a result when trying to open the same device twice an error will be raised and the function returns NULL.

Function spcm_hOpen (const char* szDeviceName):

```
drv_handle _stdcall spcm_hOpen (        // tries to open the device and returns handle or error code
    const char* szDeviceName);          // name of the device to be opened
```

Under Linux the device name in the function call needs to be a valid device name. Please change the string according to the location of the device if you don't use the standard Linux device names. The driver is installed as default under /dev/spcm0, /dev/spcm1 and so on. The kernel driver numbers the devices starting with 0.

Under Windows the only part of the device name that is used is the tailing number. The rest of the device name is ignored. Therefore to keep the examples simple we use the Linux notation in all our examples. The tailing number gives the index of the device to open. The Windows kernel driver numbers all devices that it finds on boot time starting with 0.

Example for local installed cards

```
drv_handle  hDrv;                   // returns the handle to the opended driver or NULL in case of error
hDrv = spcm_hOpen ("/dev/spcm0");   // string to the driver to open
if (!hDrv)
    printf ("open of driver failed\n");
```

Example for digitizerNETBOX/generatorNETBOX and remote installed cards

```
drv_handle  hDrv;                   // returns the handle to the opended driver or NULL in case of error
hDrv = spcm_hOpen ("TCPIP::192.168.169.14::INST0::INSTR");
if (!hDrv)
    printf ("open of driver failed\n");
```

If the function returns a NULL it is possible to read out the error description of the failed open function by simply passing this NULL to the error function. The error function is described in one of the next topics.

Function spcm_vClose

This function closes the driver and releases all allocated resources. After closing the driver handle it is not possible to access this driver any more. Be sure to close the driver if you don't need it any more to allow other programs to get access to this device.

Function spcm_vClose:

```
void _stdcall spcm_vClose (             // closes the device
    drv_handle  hDevice);               // handle to an already opened device
```

Example:

```
spcm_vClose (hDrv);
```

Function spcm_dwSetParam

All hardware settings are based on software registers that can be set by one of the functions spcm_dwSetParam. These functions set a register to a defined value or execute a command. The board must first be initialized by the spcm_hOpen function. The parameter lRegister must have a valid software register constant as defined in regs.h. The available software registers for the driver are listed in the board specific part of the documentation below. The function returns a 32 bit error code if an error occurs. If no error occurs the function returns ERR_OK, what is zero.

Function spcm_dwSetParam

```
uint32 _stdcall spcm_dwSetParam_i32 (   // Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    int32       lRegister,              // software register to be modified
    int32       lValue);                // the value to be set

uint32 _stdcall spcm_dwSetParam_i64m (  // Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    int32       lRegister,              // software register to be modified
    int32       lValueHigh,             // upper 32 bit of the value. Containing the sign bit !
    uint32      dwValueLow);            // lower 32 bit of the value.

uint32 _stdcall spcm_dwSetParam_i64 (   // Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    int32       lRegister,              // software register to be modified
    int64       llValue);               // the value to be set
```

Example:

```
if (spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE, 16384) != ERR_OK)
    printf ("Error when setting memory size\n");
```

This example sets the memory size to 16 kSamples (16384). If an error occurred the example will show a short error message

Function spcm_dwGetParam

All hardware settings are based on software registers that can be read by one of the functions spcm_dwGetParam. These functions read an internal register or status information. The board must first be initialized by the spcm_hOpen function. The parameter lRegister must have a valid software register constant as defined in the regs.h file. The available software registers for the driver are listed in the board specific part of the documentation below. The function returns a 32 bit error code if an error occurs. If no error occurs the function returns ERR_OK, what is zero.

Function spcm_dwGetParam

```
uint32 _stdcall spcm_dwGetParam_i32 (   // Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    int32       lRegister,              // software register to be read out
    int32*      plValue);               // pointer for the return value

uint32 _stdcall spcm_dwGetParam_i64m (  // Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    int32       lRegister,              // software register to be read out
    int32*      plValueHigh,            // pointer for the upper part of the return value
    uint32*     pdwValueLow);           // pointer for the lower part of the return value

uint32 _stdcall spcm_dwGetParam_i64 (   // Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    int32       lRegister,              // software register to be read out
    int64*      pllValue);              // pointer for the return value
```

Example:

```
int32 lSerialNumber;
spcm_dwGetParam_i32 (hDrv, SPC_PCISERIALNO, &lSerialNumber);
printf ("Your card has serial number: %05d\n", lSerialNumber);
```

The example reads out the serial number of the installed card and prints it. As the serial number is available under all circumstances there is no error checking when calling this function.

Different call types of spcm_dwSetParam and spcm_dwGetParam: _i32, _i64, _i64m

The three functions only differ in the type of the parameters that are used to call them. As some of the registers can exceed the 32 bit integer range (like memory size or post trigger) it is recommended to use the _i64 function to access these registers. However as there are some

programs or compilers that don't support 64 bit integer variables there are two functions that are limited to 32 bit integer variables. In case that you do not access registers that exceed 32 bit integer please use the _i32 function. In case that you access a register which exceeds 64 bit value please use the _i64m calling convention. Inhere the 64 bit value is split into a low double word part and a high double word part. Please be sure to fill both parts with valid information.

If accessing 64 bit registers with 32 bit functions the behavior differs depending on the real value that is currently located in the register. Please have a look at this table to see the different reactions depending on the size of the register:

Internal register	read/write	Function type	Behavior
32 bit register	read	spcm_dwGetParam_i32	value is returned as 32 bit integer in plValue
32 bit register	read	spcm_dwGetParam_i64	value is returned as 64 bit integer in pllValue
32 bit register	read	spcm_dwGetParam_i64m	value is returned as 64 bit integer, the lower part in plValueLow, the upper part in plValueHigh. The upper part can be ignored as it's only a sign extension
32 bit register	write	spcm_dwSetParam_i32	32 bit value can be directly written
32 bit register	write	spcm_dwSetParam_i64	64 bit value can be directly written, please be sure not to exceed the valid register value range
32 bit register	write	spcm_dwSetParam_i64m	32 bit value is written as llValueLow, the value llValueHigh needs to contain the sign extension of this value. In case of llValueLow being a value >= 0 llValueHigh can be 0, in case of llValueLow being a value < 0, llValueHigh has to be -1.
64 bit register	read	spcm_dwGetParam_i32	If the internal register has a value that is inside the 32 bit integer range (-2G up to (2G - 1)) the value is returned normally. If the internal register exceeds this size an error code ERR_EXCEEDSINT32 is returned. As an example: reading back the installed memory will work as long as this memory is < 2 GByte. If the installed memory is >= 2 GByte the function will return an error.
64 bit register	read	spcm_dwGetParam_i64	value is returned as 64 bit integer value in pllValue independent of the value of the internal register.
64 bit register	read	spcm_dwGetParam_i64m	the internal value is split into a low and a high part. As long as the internal value is within the 32 bit range, the low part plValueLow contains the 32 bit value and the upper part plValueHigh can be ignored. If the internal value exceeds the 32 bit range it is absolutely necessary to take both value parts into account.
64 bit register	write	spcm_dwSetParam_i32	the value to be written is limited to 32 bit range. If a value higher than the 32 bit range should be written, one of the other function types need to used.
64 bit register	write	spcm_dwSetParam_i64	the value has to be split into two parts. Be sure to fill the upper part lValueHigh with the correct sign extension even if you only write a 32 bit value as the driver every time interprets both parts of the function call.
64 bit register	write	spcm_dwSetParam_i64m	the value can be written directly independent of the size.

Function spcm_dwGetContBuf

This function reads out the internal continuous memory buffer in bytes, in case one has been allocated. If no buffer has been allocated the function returns a size of zero and a NULL pointer. You may use this buffer for data transfers. As the buffer is continuously allocated in memory the data transfer will speed up by up to 15% - 25%, depending on your specific kind of card. Please see further details in the appendix of this manual.

```
uint32 _stdcall spcm_dwGetContBuf_i64 ( // Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType,              // type of the buffer to read as listed above under SPCM_BUF_XXXX
    void**      ppvDataBuffer,          // address of available data buffer
    uint64*     pqwContBufLen);         // length of available continuous buffer

uint32 _stdcall spcm_dwGetContBuf_i64m (// Return value is an error code
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType,              // type of the buffer to read as listed above under SPCM_BUF_XXXX
    void**      ppvDataBuffer,          // address of available data buffer
    uint32*     pdwContBufLenH,         // high part of length of available continuous buffer
    uint32*     pdwContBufLenL);        // low part of length of available continuous buffer
```

These functions have been added in driver version 1.36. The functions are not available in older driver versions.

These functions also only have effect on locally installed cards and are neither useful nor usable with any digitizerNETBOX or generatorNETBOX products, because no local kernel driver is involved in such a setup. For remote devices these functions will return a NULL pointer for the buffer and 0 Bytes in length.

Function spcm_dwDefTransfer

The spcm_dwDefTransfer function defines a buffer for a following data transfer. This function only defines the buffer, there is no data transfer performed when calling this function. Instead the data transfer is started with separate register commands that are documented in a later chapter. At this position there is also a detailed description of the function parameters.
Please make sure that all parameters of this function match. It is especially necessary that the buffer address is a valid address pointing to memory buffer that has at least the size that is defined in the function call. Please be informed that calling this function with non valid parameters may crash your system as these values are base for following DMA transfers.

The use of this function is described in greater detail in a later chapter.

Function spcm_dwDefTransfer

```
uint32 _stdcall spcm_dwDefTransfer_i64m(// Defines the transfer buffer by 2 x 32 bit unsigned integer
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType,              // type of the buffer to define as listed above under SPCM_BUF_XXXX
    uint32      dwDirection,            // the transfer direction as defined above
    uint32      dwNotifySize,           // no. of bytes after which an event is sent (0=end of transfer)
    void*       pvDataBuffer,           // pointer to the data buffer
    uint32      dwBrdOffsH,             // high part of offset in board memory
    uint32      dwBrdOffsL,             // low part of offset in board memory
    uint32      dwTransferLenH,         // high part of transfer buffer length
    uint32      dwTransferLenL);        // low part of transfer buffer length

uint32 _stdcall spcm_dwDefTransfer_i64 (// Defines the transfer buffer by using 64 bit unsigned integer values
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType,              // type of the buffer to define as listed above under SPCM_BUF_XXXX
    uint32      dwDirection,            // the transfer direction as defined above
    uint32      dwNotifySize,           // no. of bytes after which an event is sent (0=end of transfer)
    void*       pvDataBuffer,           // pointer to the data buffer
    uint64      qwBrdOffs,              // offset for transfer in board memory
    uint64      qwTransferLen);         // buffer length
```

This function is available in two different formats as the spcm_dwGetParam and spcm_dwSetParam functions are. The background is the same. As long as you're using a compiler that supports 64 bit integer values please use the _i64 function. Any other platform needs to use the _i64m function and split offset and length in two 32 bit words.

Example:

```
int16* pnBuffer = (int16*) pvAllocMemPageAligned (16384);
if (spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA, SPCM_DIR_CARDTOPC, 0, (void*) pnBuffer, 0, 16384) != ERR_OK)
    printf ("DefTransfer failed\n");
```

The example defines a data buffer of 8 kSamples of 16 bit integer values = 16 kByte (16384 byte) for a transfer from card to PC memory. As notify size is set to 0 we only want to get an event when the transfer has finished.

Function spcm_dwInvalidateBuf

The invalidate buffer function is used to tell the driver that the buffer that has been set with spcm_dwDefTransfer call is no longer valid. It is necessary to use the same buffer type as the driver handles different buffers at the same time. Call this function if you want to delete the buffer memory after calling the spcm_dwDefTransfer function. If the buffer already has been transferred after calling spcm_dwDefTransfer it is not necessary to call this function. When calling spcm_dwDefTransfer any further defined buffer is automatically invalidated.

Function spcm_dwInvalidateBuf

```
uint32 _stdcall spcm_dwInvalidateBuf (  // invalidate the transfer buffer
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType);             // type of the buffer to invalidate as
                                        // listed above under SPCM_BUF_XXXX
```

Function spcm_dwGetErrorInfo

The function returns complete error information on the last error that has occurred. The error handling itself is explained in a later chapter in greater detail. When calling this function please be sure to have a text buffer allocated that has at least ERRORTEXTLEN length. The error text function returns a complete description of the error including the register/value combination that has raised the error and a short description of the error details. In addition it is possible to get back the error generating register/value for own error handling. If not needed the buffers for register/value can be left to NULL.

Note that the timeout event (ERR_TIMEOUT) is not counted as an error internally as it is not locking the driver but as a valid event. Therefore the GetErrorInfo function won't return the timeout event even if it had occurred in between. You can only recognize the ERR_TIMEOUT as a direct return value of the wait function that was called.

Function spcm_dwGetErrorInfo

```
uint32 _stdcall spcm_dwGetErrorInfo_i32 (
    drv_handle  hDevice,                // handle to an already opened device
    uint32*     pdwErrorReg,            // address of the error register (can be zero if not of interest)
    int32*      plErrorValue,           // address of the error value    (can be zero if not of interest)
    char        pszErrorTextBuffer[ERRORTEXTLEN]); // text buffer for text error
```

Example:

```
char szErrorBuf[ERRORTEXTLEN];
if (spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE, -1))
    {
    spcm_dwGetErrorInfo_i32 (hDrv, NULL, NULL, szErrorBuf);
    printf ("Set of memsize failed with error message: %s\n", szErrorBuf);
    }
```

Delphi (Pascal) Programming Interface

Driver interface

The driver interface is located in the sub-directory d_header and contains the following files. The files need to be included in the delphi project and have to be put into the „uses" section of the source files that will access the driver. Please do not edit any of these files as they're regularly updated if new functions or registers have been included.

file spcm_win32.pas

The file contains the interface to the driver library and defines some needed constants and variable types. All functions of the delphi library are similar to the above explained standard driver functions:

```
// ----- device handling functions -----
function spcm_hOpen (strName: pchar): int32; stdcall; external 'spcm_win32.dll' name '_spcm_hOpen@4';
procedure spcm_vClose (hDevice: int32); stdcall; external 'spcm_win32.dll' name '_spcm_vClose@4';

function spcm_dwGetErrorInfo_i32 (hDevice: int32; var lErrorReg, lErrorValue: int32; strError: pchar): uint32;
stdcall; external 'spcm_win32.dll' name '_spcm_dwGetErrorInfo_i32@16'

// ----- register access functions -----
function spcm_dwSetParam_i32 (hDevice, lRegister, lValue: int32): uint32;
stdcall; external 'spcm_win32.dll' name '_spcm_dwSetParam_i32@12';

function spcm_dwSetParam_i64 (hDevice, lRegister: int32; llValue: int64): uint32;
stdcall; external 'spcm_win32.dll' name '_spcm_dwSetParam_i64@16';

function spcm_dwGetParam_i32 (hDevice, lRegister: int32; var plValue: int32): uint32;
stdcall; external 'spcm_win32.dll' name '_spcm_dwGetParam_i32@12';

function spcm_dwGetParam_i64 (hDevice, lRegister: int32; var pllValue: int64): uint32;
stdcall; external 'spcm_win32.dll' name '_spcm_dwGetParam_i64@12';

// ----- data handling -----
function spcm_dwDefTransfer_i64 (hDevice, dwBufType, dwDirection, dwNotifySize: int32; pvDataBuffer: Pointer;
llBrdOffs, llTransferLen: int64): uint32;
stdcall; external 'spcm_win32.dll' name '_spcm_dwDefTransfer_i64@36';

function spcm_dwInvalidateBuf (hDevice, lBuffer: int32): uint32;
stdcall; external 'spcm_win32.dll' name '_spcm_dwInvalidateBuf@8';
```

The file also defines types used inside the driver and the examples. The types have similar names as used under C/C++ to keep the examples more simple to understand and allow a better comparison.

file SpcRegs.pas

The SpcRegs.pas file defines all constants that are used for the driver. The constant names are the same names as used under the C/C++ examples. All constants names will be found throughout this hardware manual when certain aspects of the driver usage are explained. It is recommended to only use these constant names for better visibility of the programs:

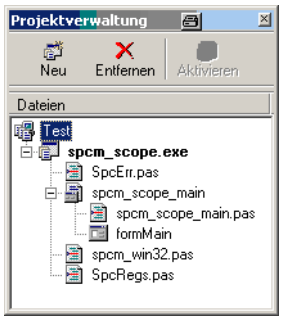
```
const SPC_M2CMD                         = 100;              { write a command }
const     M2CMD_CARD_RESET              = $00000001;        { hardware reset    }
const     M2CMD_CARD_WRITESETUP         = $00000002;        { write setup only }
const     M2CMD_CARD_START              = $00000004;        { start of card (including writesetup) }
const     M2CMD_CARD_ENABLETRIGGER      = $00000008;        { enable trigger engine }
...
```

file SpcErr.pas

The SpeErr.pas file contains all error codes that may be returned by the driver.

Including the driver files

To use the driver function and all the defined constants it is necessary to include the files into the project as shown in the picture on the right. The project overview is taken from one of the examples delivered on USB-Stick. Besides including the driver files in the project it is also necessary to include them in the uses section of the source files where functions or constants should be used:




```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls,

  SpcRegs, SpcErr, spcm_win32;
```

Examples

Examples for Delphi can be found on USB-Stick in the directory /examples/delphi. The directory contains the above mentioned delphi header files and a couple of universal examples, each of them working with a certain type of card. Please feel free to use these examples as a base for your programs and to modify them in any kind.

spcm_scope

The example implements a very simple scope program that makes single acquisitions on button pressing. A fixed setup is done inside the example. The spcm_scope example can be used with any analog data acquisition card from Spectrum. It covers cards with 1 byte per sample (8 bit resolution) as well as cards with 2 bytes per sample (12, 14 and 16 bit resolution)

The program shows the following steps:

- Initialization of a card and reading of card information like type, function and serial number
- Doing a simple card setup
- Performing the acquisition and waiting for the end interrupt
- Reading of data, re-scaling it and displaying waveform on screen

.NET programming languages

Library

For using the driver with a .NET based language Spectrum delivers a special library that encapsulates the driver in a .NET object. By adding this object to the project it is possible to access all driver functions and constants from within your .NET environment.

There is one small console based example for each supported .NET language that shows how to include the driver and how to access the cards. Please combine this example with the different standard examples to get the different card functionality.

Declaration

The driver access methods and also all the type, register and error declarations are combined in the object Spcm and are located in one of the two DLLs either SpcmDrv32.NET.dll or SpcmDrv64.NET.dll delivered with the .NET examples.

For simplicity, either file is simply called „SpcmDrv.NET.dll“ in the following passages and the actual file name must be replaced with either the 32bit or 64bit version according to your application.

Spectrum also delivers the source code of the DLLs as a C# project. These sources are located in the directory SpcmDrv.NET.

```
namespace Spcm
    {
    public class Drv
        {
        [DllImport("spcm_win32.dll")]public static extern IntPtr spcm_hOpen (string szDeviceName);
        [DllImport("spcm_win32.dll")]public static extern void spcm_vClose  (IntPtr hDevice);
...
    public class CardType
        {
        public const int TYP_M2I2020               = unchecked ((int)0x00032020);
        public const int TYP_M2I2021               = unchecked ((int)0x00032021);
        public const int TYP_M2I2025               = unchecked ((int)0x00032025);
...
    public class Regs
        {
        public const int SPC_M2CMD                 = unchecked ((int)100);
        public const int M2CMD_CARD_RESET          = unchecked ((int)0x00000001);
        public const int M2CMD_CARD_WRITESETUP     = unchecked ((int)0x00000002);
...
```

Using C#

The SpcmDrv.NET.dll needs to be included within the Solution Explorer in the References section. Please use right mouse and select „AddReference“. After this all functions and constants of the driver object are available.

Please see the example in the directory CSharp as a start:

```
// ----- open card -----
hDevice = Drv.spcm_hOpen("/dev/spcm0");
if ((int)hDevice == 0)
    {
    Console.WriteLine("Error: Could not open card\n");
    return 1;
    }

// ----- get card type -----
dwErrorCode = Drv.spcm_dwGetParam_i32(hDevice, Regs.SPC_PCITYP, out lCardType);
dwErrorCode = Drv.spcm_dwGetParam_i32(hDevice, Regs.SPC_PCISERIALNR, out lSerialNumber);
```

Example for digitizerNETBOX/generatorNETBOX and remotely installed cards:

```
// ----- open remote card -----
hDevice = Drv.spcm_hOpen("TCPIP::192.168.169.14::INST0::INSTR");
```

Using Managed C++/CLI

The SpcmDrv.NET.dll needs to be included within the project options. Please select „Project" - „Properties" - „References" and finally „Add new Reference". After this all functions and constants of the driver object are available.

Please see the example in the directory CppCLR as a start:

```
// ----- open card -----
hDevice = Drv::spcm_hOpen("/dev/spcm0");
if ((int)hDevice == 0)
    {
    Console::WriteLine("Error: Could not open card\n");
    return 1;
    }

// ----- get card type -----
dwErrorCode = Drv::spcm_dwGetParam_i32(hDevice, Regs::SPC_PCITYP, lCardType);
dwErrorCode = Drv::spcm_dwGetParam_i32(hDevice, Regs::SPC_PCISERIALNR, lSerialNumber);
```

Example for digitizerNETBOX/generatorNETBOX and remotely installed cards:

```
// ----- open remote card -----
hDevice = Drv::spcm_hOpen("TCPIP::192.168.169.14::INST0::INSTR");
```

Using VB.NET

The SpcmDrv.NET.dll needs to be included within the project options. Please select „Project" - „Properties" - „References" and finally „Add new Reference". After this all functions and constants of the driver object are available.

Please see the example in the directory VB.NET as a start:

```
' ----- open card -----
hDevice = Drv.spcm_hOpen("/dev/spcm0")

If (hDevice = 0) Then
    Console.WriteLine("Error: Could not open card\n")
Else

    ' ----- get card type -----
    dwError = Drv.spcm_dwGetParam_i32(hDevice, Regs.SPC_PCITYP, lCardType)
    dwError = Drv.spcm_dwGetParam_i32(hDevice, Regs.SPC_PCISERIALNR, lSerialNumber)
```

Example for digitizerNETBOX/generatorNETBOX and remotely installed cards:

```
' ----- open remote card -----
hDevice = Drv.spcm_hOpen("TCPIP::192.168.169.14::INST0::INSTR")
```

Using J#

The SpcmDrv.NET.dll needs to be included within the Solution Explorer in the References section. Please use right mouse and select „AddReference". After this all functions and constants of the driver object are available.

Please see the example in the directory JSharp as a start:

```
// ----- open card -----
hDevice = Drv.spcm_hOpen("/dev/spcm0");

if (hDevice.ToInt32() == 0)
    System.out.println("Error: Could not open card\n");
else
    {
    // ----- get card type -----
    dwErrorCode = Drv.spcm_dwGetParam_i32(hDevice, Regs.SPC_PCITYP, lCardType);
    dwErrorCode = Drv.spcm_dwGetParam_i32(hDevice, Regs.SPC_PCISERIALNR, lSerialNumber);
```

Example for digitizerNETBOX/generatorNETBOX and remotely installed cards:

```
' ----- open remote card -----
hDevice = Drv.spcm_hOpen("TCPIP::192.168.169.14::INST0::INSTR")
```

Python Programming Interface and Examples

Driver interface

The driver interface contains the following files. The files need to be included in the python project. Please do not edit any of these files as they are regularly updated if new functions or registers have been included. To use pyspcm you need either python 2 (2.4, 2.6 or 2.7) or python 3 (3.x) and ctype, which is included in python 2.6 and newer and needs to be installed separately for Python 2.4.

file pyspcm.py

The file contains the interface to the driver library and defines some needed constants. All functions of the python library are similar to the above explained standard driver functions and use ctypes as input and return parameters:

```
# ----- Windows -----
spcmDll = windll.LoadLibrary ("c:\\windows\\system32\\spcm_win32.dll")

# load spcm_hOpen
spcm_hOpen = getattr (spcmDll, "_spcm_hOpen@4")
spcm_hOpen.argtype = [c_char_p]
spcm_hOpen.restype = drv_handle

# load spcm_vClose
spcm_vClose = getattr (spcmDll, "_spcm_vClose@4")
spcm_vClose.argtype = [drv_handle]
spcm_vClose.restype = None

# load spcm_dwGetErrorInfo
spcm_dwGetErrorInfo_i32 = getattr (spcmDll, "_spcm_dwGetErrorInfo_i32@16")
spcm_dwGetErrorInfo_i32.argtype = [drv_handle, ptr32, ptr32, c_char_p]
spcm_dwGetErrorInfo_i32.restype = uint32

# load spcm_dwGetParam_i32
spcm_dwGetParam_i32 = getattr (spcmDll, "_spcm_dwGetParam_i32@12")
spcm_dwGetParam_i32.argtype = [drv_handle, int32, ptr32]
spcm_dwGetParam_i32.restype = uint32

# load spcm_dwGetParam_i64
spcm_dwGetParam_i64 = getattr (spcmDll, "_spcm_dwGetParam_i64@12")
spcm_dwGetParam_i64.argtype = [drv_handle, int32, ptr64]
spcm_dwGetParam_i64.restype = uint32

# load spcm_dwSetParam_i32
spcm_dwSetParam_i32 = getattr (spcmDll, "_spcm_dwSetParam_i32@12")
spcm_dwSetParam_i32.argtype = [drv_handle, int32, int32]
spcm_dwSetParam_i32.restype = uint32

# load spcm_dwSetParam_i64
spcm_dwSetParam_i64 = getattr (spcmDll, "_spcm_dwSetParam_i64@16")
spcm_dwSetParam_i64.argtype = [drv_handle, int32, int64]
spcm_dwSetParam_i64.restype = uint32

# load spcm_dwSetParam_i64m
spcm_dwSetParam_i64m = getattr (spcmDll, "_spcm_dwSetParam_i64m@16")
spcm_dwSetParam_i64m.argtype = [drv_handle, int32, int32, int32]
spcm_dwSetParam_i64m.restype = uint32

# load spcm_dwDefTransfer_i64
spcm_dwDefTransfer_i64 = getattr (spcmDll, "_spcm_dwDefTransfer_i64@36")
spcm_dwDefTransfer_i64.argtype = [drv_handle, uint32, uint32, uint32, c_void_p, uint64, uint64]
spcm_dwDefTransfer_i64.restype = uint32

spcm_dwInvalidateBuf = getattr (spcmDll, "_spcm_dwInvalidateBuf@8")
spcm_dwInvalidateBuf.argtype = [drv_handle, uint32]
spcm_dwInvalidateBuf.restype = uint32


# ----- Linux -----
# use cdll because all driver access functions use cdecl calling convention under linux
spcmDll = cdll.LoadLibrary ("libspcm_linux.so")

# the loading of the driver access functions is similar to windows:

# load spcm_hOpen
spcm_hOpen = getattr (spcmDll, "spcm_hOpen")
spcm_hOpen.argtype = [c_char_p]
spcm_hOpen.restype = drv_handle

# ...
```

file regs.py

The regs.py file defines all constants that are used for the driver. The constant names are the same names compared to the C/C++ examples. All constant names will be found throughout this hardware manual when certain aspects of the driver usage are explained. It is recommended to only use these constant names for better readability of the programs:

```
SPC_M2CMD = 1001                              # write a command
M2CMD_CARD_RESET = 0x000000011                # hardware reset
M2CMD_CARD_WRITESETUP = 0x000000021           # write setup only
M2CMD_CARD_START = 0x000000041                # start of card (including writesetup)
M2CMD_CARD_ENABLETRIGGER = 0x000000081        # enable trigger engine
...
```

file spcerr.py

The spcerr.py file contains all error codes that may be returned by the driver.

Examples

Examples for Python can be found on USB-Stick in the directory /examples/python. The directory contains the above mentioned header files and some examples, each of them working with a certain type of card. Please feel free to use these examples as a base for your programs and to modify them in any kind.

When allocating the buffer for DMA transfers, use the following function to get a mutable character buffer: ctypes.create_string_buffer(init_or_size[, size])

Java Programming Interface and Examples

Driver interface

The driver interface contains the following Java files (classes). The files need to be included in your Java project. Please do not edit any of these files as they are regularily updated if new functions or registers have been included. The driver interface uses the Java Native Access (JNA) library.

This library is licensed under the LGPL (https://www.gnu.org/licenses/lgpl-3.0.en.html) and has also to be included to your Java project.

To download the latest jna.jar package and to get more information about the JNA project please check the projects GitHub page under: https://github.com/java-native-access/jna

The following files can be found in the „SpcmDrv" folder of your Java examples install path.

SpcmDrv32.java / SpcmDrv64.java

The files contain the interface to the driver library and defines some needed constants. All functions of the driver interface are similar to the above explained standard driver functions. Use the SpcmDrv32.java for 32 bit and the SpcmDrv64.java for 64 bit projects:

```
...

public interface SpcmWin64 extends StdCallLibrary {

    SpcmWin64 INSTANCE = (SpcmWin64)Native.loadLibrary (("spcm_win64"), SpcmWin64.class);

    int spcm_hOpen (String sDeviceName);
    void spcm_vClose (int hDevice);
    int spcm_dwSetParam_i64  (int hDevice, int lRegister, long llValue);
    int spcm_dwGetParam_i64 (int hDevice, int lRegister, LongByReference pllValue);
    int spcm_dwDefTransfer_i64 (int hDevice, int lBufType, int lDirection, int lNotifySize,
                                Pointer pDataBuffer, long llBrdOffs, long llTransferLen);
    int spcm_dwInvalidateBuf (int hDevice, int lBufType);
    int spcm_dwGetErrorInfo_i32 (int hDevice, IntByReference plErrorReg,
                                 IntByReference plErrorValue, Pointer sErrorTextBuffer);
    }

...
```

SpcmRegs.java

The SpcmRegs class defines all constants that are used for the driver. The constants names are the same names compared to the C/C++ examples. All constant names will be found throughout this hardware manual when certain aspects of the driver usage are explained. It is recommended to only use these constant names for better readability of the programs:

```
...

public static final int SPC_M2CMD = 100;
public static final int M2CMD_CARD_RESET = 0x00000001;
public static final int M2CMD_CARD_WRITESETUP = 0x00000002;
public static final int M2CMD_CARD_START = 0x00000004;
public static final int M2CMD_CARD_ENABLETRIGGER = 0x00000008;
...
```

SpcmErrors.java

The SpcmErrors class contains all error codes that may be returned by the driver.

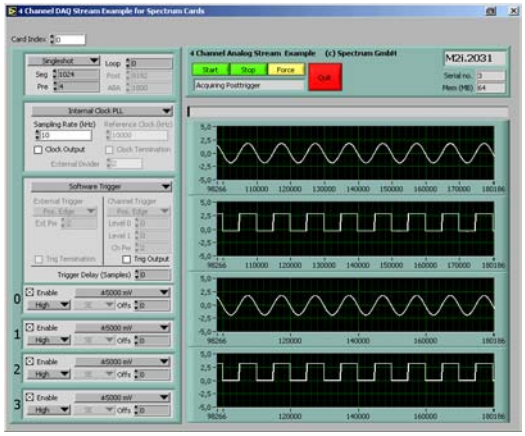
Examples

Examples for Java can be found on USB-Stick in the directory /examples/java. The directory contains the above mentioned header files and some examples, each of them working with a certain type of card. Please feel free to use these examples as a base for your programs and to modify them in any kind.

LabVIEW driver and examples

A full set of drivers and examples is available for LabVIEW for Windows. LabVIEW for Linux is currently not supported. The LabVIEW drivers have their own manual. The LabVIEW drivers, examples and the manual are found on the USB-Stick that has been included in the delivery. The latest version is also available on our webpage www.spectrum-instrumentation.com

Please follow the description in the LabVIEW manual for installation and useage of the LabVIEW drivers for this card.

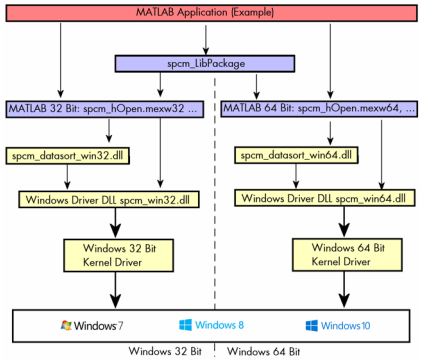


MATLAB driver and examples

A full set of drivers and examples is available for Mathworks MATLAB for Windows (32 bit and 64 bit versions) and also for MATLAB for Linux (64 bit version). There is no additional toolbox needed to run the MATLAB examples and drivers.

The MATLAB drivers have their own manual. The MATLAB drivers, examples and the manual are found on the USB-Stick that has been included in the delivery. The latest version is also available on our webpage www.spectrum-instrumentation.com

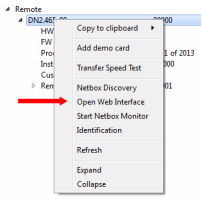
Please follow the description in the MATLAB manual for installation and useage of the MATLAB drivers for this card.



Integrated Webserver

The digitizerNETBOX/generatorNETBOX has an integrated webserver following the LXI standards. The web pages give informtion about the device, allows to set up ethernet details or make firmware updates.

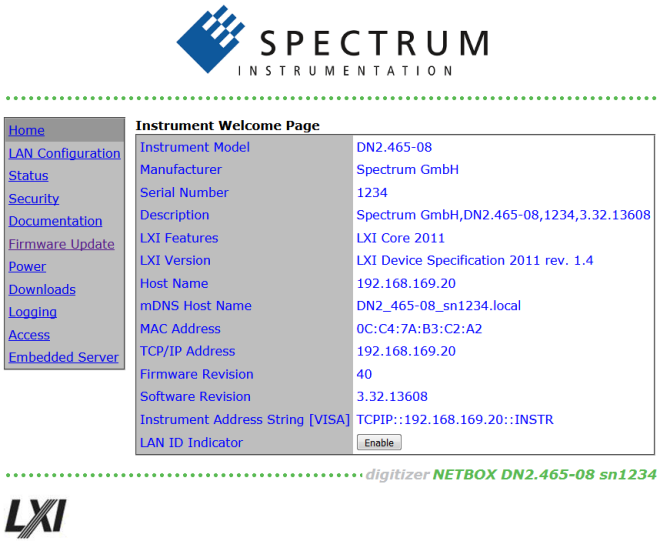
The webserver can be reached in three different ways:



- Directly by typing the IP address into the URL field of a Web Browser.
- By selecting it from the Spectrum Control Center via the context menu on the remote device node (as shown on the screen shot on the right).
- On Windows machines (starting with Windows 7) on the device properties page, as described in the section „Finding the digitizerNETBOX in the network“ earlier in this manual.

Home Screen

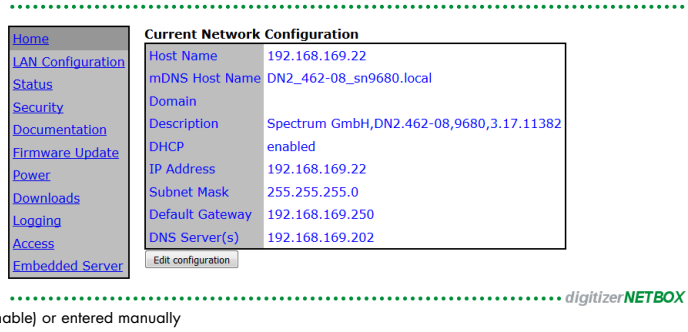
The home screen gives an overview about the instrument showing all main information:



Name	Description
Instrument Model	The specific model code of your digitizerNETBOX or generatorNETBOX
Manufacturer	Manufacturer of the device - Spectrum GmbH
Serial Number	The unique serial number of the product. The serial number is also found on the type plate on the back of the chassis of the digitizerNETBOX/generatorNETBOX.
Description	A free definable description of the specific device that you can edit by yourself in the LAN configuration page. It is recommended to include the location of the device and any other infoamtion that helps your network administrator.
LXI Features	Listing the supported LXI features
LXI Version	Listing the used LXI specification for designing this device
Host Name	The host name given by the DNS server. If the DNS server does not generate a host name, the IP address is shown
mDNS Host Name	The internal mDNS host name which allows to find the device in the network environment. The mDNS host name can also be changed in the LAN configuration page
MAC Address	The unique MAC address of the device which can also be found on the type plate on the back of the device
TCP/IP Address	The current TCP/IP address as given by the DNS
Firmware revision	The revision of the installed firmware files for the digitizerNETBOX/generatorNETBOX itself. The integrated digitizer modules have their own firmware versioning and can be read out by the Spectrum control center
Software Revision	The software revision of the integrated remote server software
Instrument Address String (VISA)	The instrument address string following the VISA notification. Using this address string one can access the digitizerNETBOX/generatorNETBOX from the software. The integrated digitizer modules are numbered starting with INST0 (example: TCPIP::192.168.169.14::INST0::INSTR)
LAN ID Indicator	Pressing this button starts flashing the LAN LED light on the front plate of the device. This helps to find the device inside a 19“ rack where the back of the device with the type plate is not easily accessible.

LAN Configuration

The LAN configuration page allows to change the LAN configuration of the device. This page is password protected if a password is given in the security page.



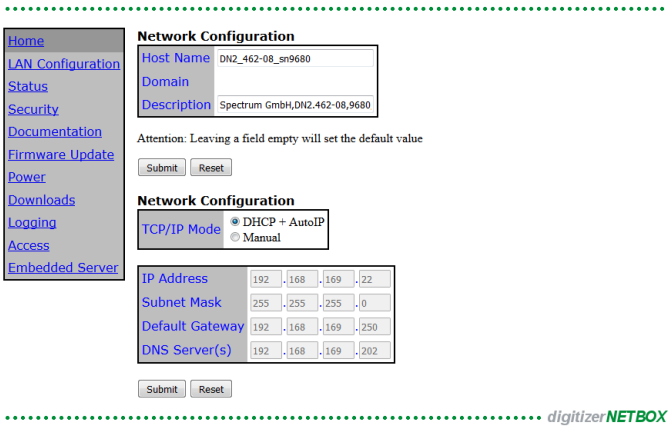
Name	Description
Host Name	The offical host name as given by the DNS
mDNS Host Name	The local host name which can be changed here
Domain	The domain in which the digitizerNETBOX is placed if the DNS server has filled this information correctly
Description	The device description which can be changed here
DHCP	DHCP (Dynamic Host Configuration Protocol) setting
IP Address	The current IP address as given by the DHCP server (DHCP enable) or entered manually
Subnet Mask	The current subnet mask as given by the DHCP server (DHCP enable) or entered manually
Default Gateway	The current default gateway address as given by the DHCP server (DHCP enable) or entered manually
DNS Server(s)	The current DNS server address as given by the DHCP server (DHCP enable) or entered manually

As default DHCP (IPv4) will be used and an IP address will be automatically set. In case no DHCP server is found, an IP will be obtained using the AutoIP feature. This will lead to an IPv4 address of 169.254.x.y (with x and y being assigned to a free IP in the network) using a subnet mask of 255.255.0.0.

The default IP setup can also be restored, by using the „LAN Reset“ button on the device.

If a fixed IP address should be used instead, the parameters need to be set according to the current LAN requirements.

Pressing the „edit configuration“ button will issue a new edit page. If a password is given in the security pages the password must be entered before the edit screen is available



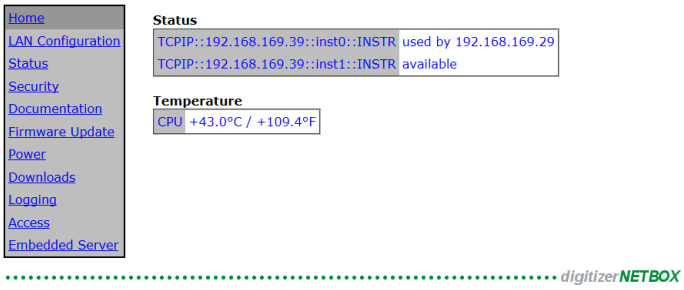
Name	Description
Host Name	Enter a new host name for the mDNS host name. Please note that host names can only contain letters, numbers, minus and underscore, no dots or blanks are allowed
Domain	The domain in which the digitizerNETBOX/generatorNETBOX is placed
Submit Button	After review this button submits the changes and changes host name and description permanently
Reset Button	Discards the changes and returns host name and description to the previous values.
TCP/IP Mode	Select between DHCP + AutoIP to have all configuration done automatically or Manual to enter all IP related settings manual.
IP Address	Only available if manual TCP/IP mode is selected
Subnet Mask	Only available if manual TCP/IP mode is selected
Default Gateway	Only available if manual TCP/IP mode is selected
DNS Server(s)	Only available if manual TCP/IP mode is selected
Submit Button	Submits the changes. If you set the IP details manually please be sure that your device is adressable within your network. In case of a failure the LAN reset button on the front page of the device will set back the LAN configuration to DHCP
Reset Button	Discards the changes and returns IP settings to the previous values

Status

Shows the internal device status. For each internal digitizer/generator module the status whether the module is available or locked by a user is shown. A digitizer/generator module is locked as soon as it is opened from any software on any PC.

In case the instrument is locked, the IP address of the current control PC can be obtained here.

Also the current temperature will be displayed here. DN6.xxxx models of either the digitizerNETBOX or generatorNETBOX will also display the case fan speed here as well (not shown on screen shot).

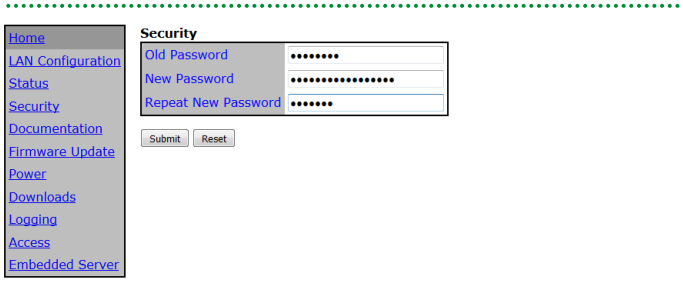


Security

Allows to set a password to protect the device from changes. The password secures access to LAN configuration, power settings like reboot or power down and firmware updates of the instrument. As default no password is set for the configuration.

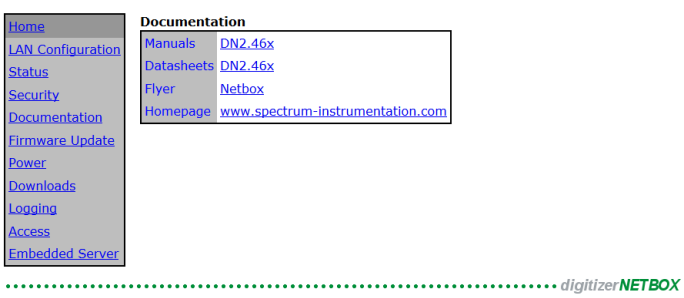
To change the password the old password has to be entered once and the new password twice to avoid typing errors.

In case of a lost password the LAN reset button on the front plate of the digitizerNETBOX/generatorNETBOX will delete the password and set the complete device to the default stage again.



Documentation

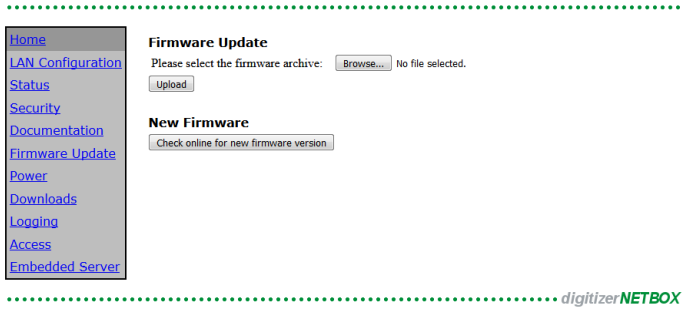
All related documents for the device that may be needed to operate the digitizerNETBOX/generatorNETBOX or to program it are available by download as pdf documents from here.



Firmware Update

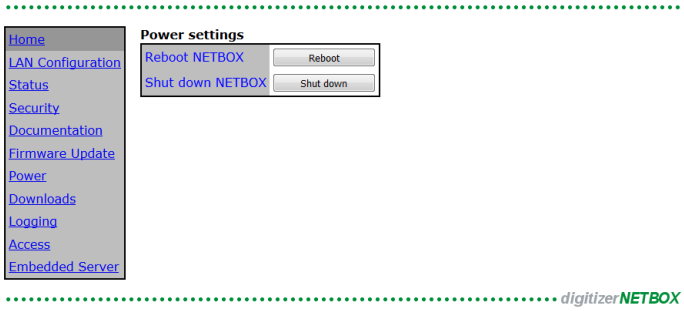
The complete firmware of the device can be updated with a single firmware update file which is available for download directly here by clicking the „check online“ button or on the Spectrum webpage www.spectrum-instrumentation.com. The firmware file contains update files for the following parts:

- firmware files of the integrated digitizer/generator modules
- drivers for the digitizer/generator modules
- software and setup of the underlying operating system
- webserver and integrated web pages and manuals
- remote server software
- initialization scripts and tools



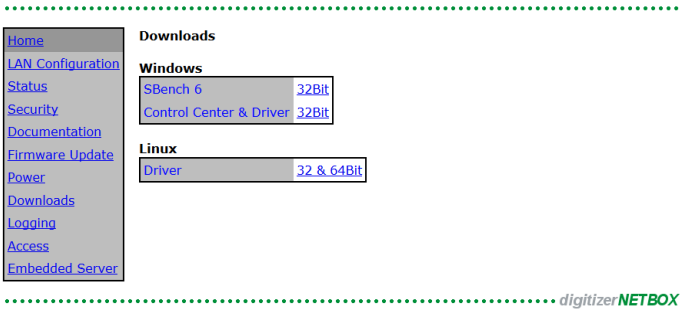
Power

From here the digitizerNETBOX/generatorNETBOX can be remotely shut down or remotely rebooted. Please make sure that no software is currently accessing the digitizerNETBOX or generatorNETBOX before using any of these power options.



Downloads

The websever gives access to all necessary software components for download. All these software installers are also available on the USB-Stick that is delivered with the digitizerNETBOX/generatorNETBOX and on the internet.

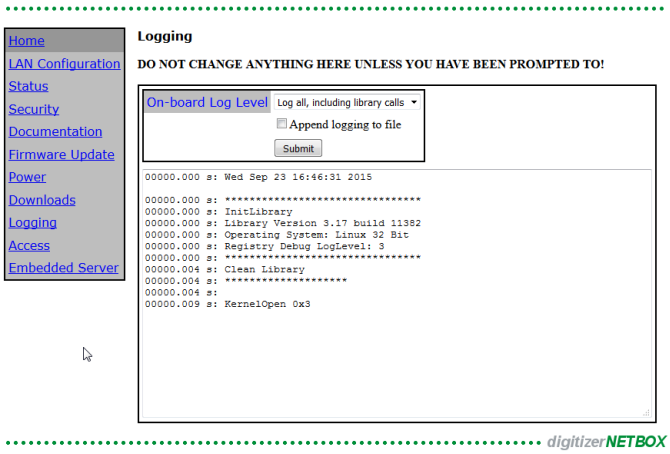


Logging

This is a debug setting only. You shouldn't change any of these settings unless our support team requested you to do so. Operating the digitizerNETBOX/generatorNETBOX with log-level „Log all“ will slow down the operation as each single call is logged as a text entry in the internal log file.

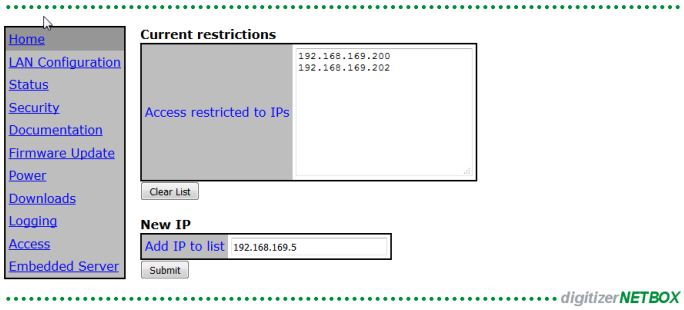
These debug log settings are similar to the ones described in the chapter about the Spectrum control center. Using this logging the internal communication between the remote server and the locally installed Spectrum driver is logged.

Please note that some digitizerNETBOX/generatorNETBOX products (having only one internal digitizer/generator installed) show an error message „KernelOpen /dev/spcm1 failed“. This error message is not an error but simply the remote server trying to open the second internal digitizer that isn't installed.



Access

In here it is possible to restrict the access to the digitizerNETBOX/generatorNETBOX to certain IP addresses. As long as the access list is clear, everybody who has a TCP/IP connection to the digitizerNETBOX/generatorNETBOX can get control of it and use it with any software like SBench 6.



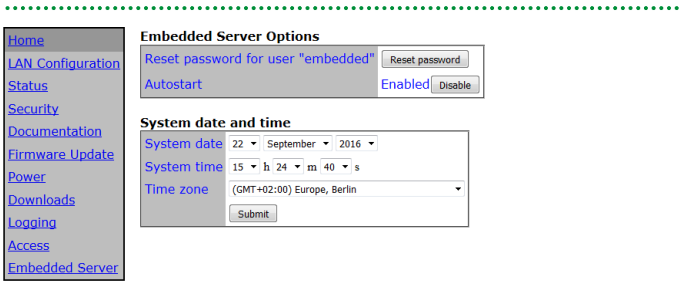
Use the add IP to list field with the submit button to add an IP address to the list. As a default your current IP address is shown in the entry field.

After having setup an access list everybody else who is not on the access restricted IP list can still see the digitizerNETBOX or generatorNETBOX in the network and use the discovery function but access to the internal digitizers/generators is restricted and no longer possible.

Use this option together with the password option to completely secure the digitizerNETBOX/generatorNETBOX from unwanted access.

Embedded Server

The embedded server is an option and is only available if ordered with and installed on your particular digitizerNETBOX/generatorNETBOX. Please see the dedicated Embedded Server Option chapter for more information on this feature.

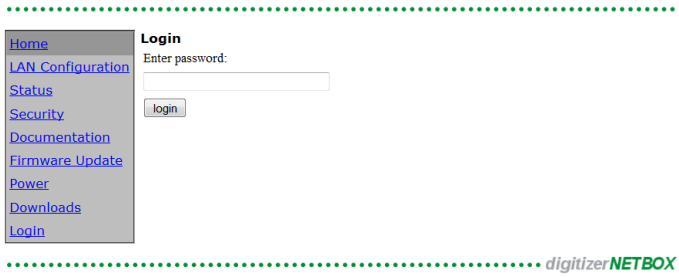


Using the „Reset password" button the password for the user „embedded" is reset to the default password which is also „embedded"

The autostart feature allows the user to automatically start scripts, programs or services on the device during boot process. If something fails with the start, the autostart feature can be disabled using the „Autostart [Disable]" button. After fixing the automatically starting programs one can enable the autostart feature again.

Login/Logout

As soon as a password has been entered in the security settings a login/logout command is available from the webpage menu.



After entering the password once the login stays valid until a logout or until closing the web browser.

IVI Driver

The IVI Foundation is an open consortium founded in 1998 to promote standards for programming test instruments. Composed primarily of instrument manufacturers, end-users, software vendors, and system integrators, the Foundation strives to create specifications that govern the development of instrument drivers.

-> http://IVIfoundation.org

About IVI

The IVI standards define an open driver architecture, a set of instrument classes, and shared software components. Together these provide critical elements needed for instrument interchangeability.

Benefits

IVI offers several benefits to measurement system designers:

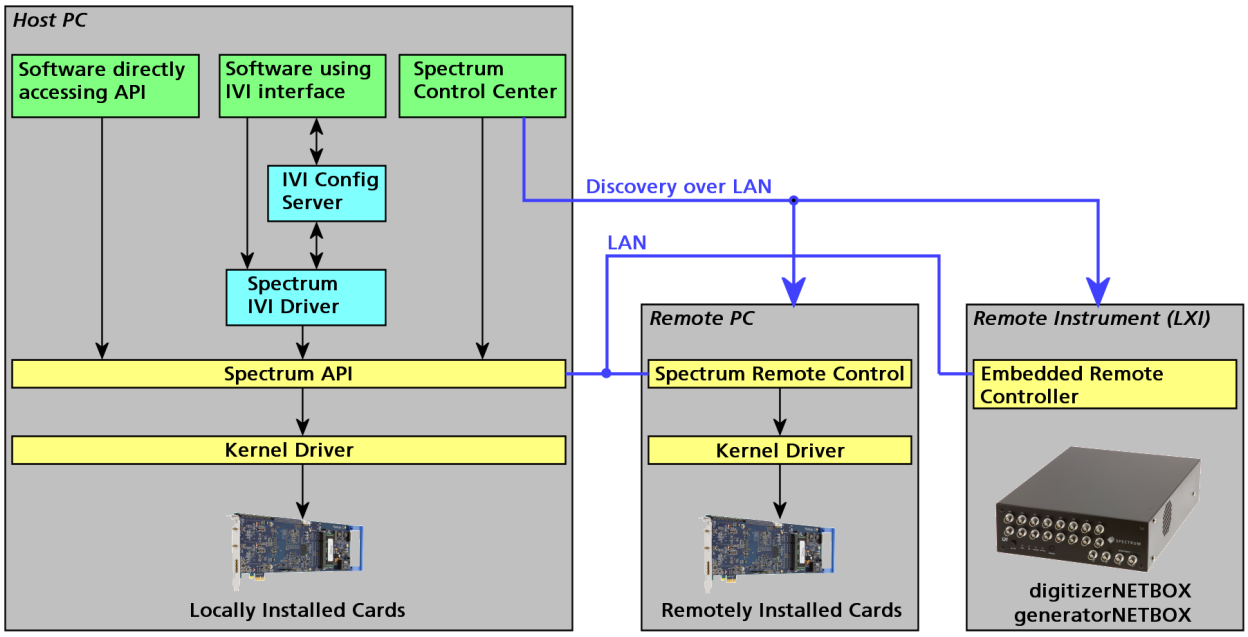
- IVI's defined Application Programming Interfaces (APIs) standardize common measurement functions reducing the time needed to learn a new IVI instrument.
- Instrument simulation allows developers to run code without an instrument. This feature reduces the need for sometimes scarce measurement hardware resources and it can simplify testing of measurement applications.
- IVI drivers feature enhanced ease of use in popular Application Development Environments. IVI's standard APIs, combined with IVI driver wrappers where appropriate, provide fast, intuitive access to driver functions.
- IVI drivers provide for interchangeability. Interchangeability reduces the time and effort needed to integrate measurement devices into new or existing systems

Interchangeability

Systems designed with IVI drivers enjoy the benefits of standardized code that can be interchanged into other systems. This code also supports interchange of measurement devices – helping to prevent hardware obsolescence. Interchangeability is supported on three levels: The IVI architecture specifications allow architectural interchangeability – that is a standard driver architecture that can be reused. The class specifications provide syntactic interchangeability which supports instrument exchange with minimal code changes. The highest level of interchangeability is achieved by using the IVI signal specifications.

General Concept of the Spectrum IVI driver

The Spectrum IVI driver is based on the standard Spectrum API and can be used with any Spectrum products specified below in the supported hardware chapter. The Spectrum products to be accessed with the IVI driver can be locally installed data acquisition cards, remotely installed data acquisition cards or remote LXI instruments like a digitizerNETBOX or generatorNETBOX.



Supported Spectrum Hardware

All Spectrum analog data acquisition hardware based on the SPCM driver structure is supported by the IVI driver. There is only one IVI driver for all hardware.

Supported data acquisition card families:

- M2i.20xx and M2i.20xx-exp family
- M3i.21xx and M3i.21xx-exp family
- M4i.22xx-x8 and M4x.22xx-x4 family
- M2i.30xx and M2i.30xx-exp family
- M2i.31xx and M2i.31xx-exp family
- M3i.32xx and M3i.32xx-exp family
- M2i.40xx and M2i.40xx-exp family
- M3i.41xx and M3i.41xx-exp family
- M4i.44xx-x8 and M4x.44xx-x4 family
- M2i.46xx and M2i.46xx-exp family
- M2i.47xx and M2i.47xx-exp family
- M3i.48xx and M3i.48xx-exp family
- M2i.49xx and M2i.49xx-exp family
- M2p.59xx-x4 family
- M2p.65xx-x4 family
- M2i.60xx and M2i.60xx-exp family
- M4i.66xx-x8 and M4x.66xx-x4 family

Supported digitizerNETBOX families

- DN2.20x-xx family
- DN2.22x-xx and DN6.22x-xx family
- DN2.44x-xx and DN6.44x-xx family
- DN2.46x-xx and DN6.46x-xx family
- DN2.49x-xx and DN6.49x-xx family
- DN2.59x-xx and DN6.59x-xx family

Supported generatorNETBOX families

- DN2.60x-xx family
- DN2.65x-xx and DN6.65x-xx family
- DN2.66x-xx and DN6.66x-xx family

IVI Compliance

General information on the Spectrum IVI driver:

IVI class specification version	Version 3.3
IVI-C interface	supported
IVI-COM interface	supported
IVI.NET interface	not supported

The following IVI classes are supported by different instrument types:

IVI Class	Supported by Spectrum hardware	IVI specific driver function prefix
IVIScope	Supported by all digitizerNETBOX devices and analog data acquisition cards listed above	SpecScope_
IVIDigitizer	Supported by all digitizerNETBOX devices and analog data acquisition cards listed above	SpecDigitizer_
IVIFgen	Supported by all generatorNETBOX devices and analog data generator cards listed above	SpecFGen_

Supported Operating Systems

32 bit operating systems	64 bit operating systems
Winodws 7	Windows 7
Windows 8	Windows 8
Windows 10	Windows 10

Supported Standard Driver Features

Feature	Supported	Description of the Feature
State caching	yes standard feature of the API which is permanently active	To minimize the number of I/O calls needed to configure an instrument to a new state, IVI specific drivers may implement state caching. IVI specific drivers can choose to implement state caching for all, some, or none of the instrument settings. If the user enables state caching and the IVI specific driver implements caching for hardware configuration attributes, driver functions perform instrument I/O when the current state of the instrument settings is different from what the user requests.
Range checking	yes standard feature of the API which is permanently active	If range checking is enabled, an IVI specific driver checks that input parameters are within the valid range for the instrument.
Instrument Status Checking	yes standard feature of the API which is permanently active	If instrument status checking is enabled, an IVI specific driver automatically checks the status of the instrument after most operations. If the instrument indicates that it has an error, the driver returns a special error code. The user then calls the Error Query function to retrieve the instrument specific error code from the instrument.
Multithread Safety	yes	IVI drivers are multithread safe. Multithread safety means that multiple threads in the same process can use the same IVI driver session and that different sessions of the same IVI driver can run simultaneously on different threads.
Simulation	yes	If simulation is enabled, an IVI specific driver does not perform instrument I/O, and the driver creates simulated data for output parameters. This allows the user to execute instrument driver calls in the application program even though the instrument is not available.

IVIScope Supported Class Capabilities

Feature	Supported	Description of Feature
IVIScopeBase	yes	Base Capabilities of the IVIScope specification. This group includes the capability to acquire waveforms using edge triggering.
IVIScopeInterpolation	no	Extension: IVIScope with the ability to configure the oscilloscope to interpolate missing points in a waveform.
IVIScopeTVTrigger	no	Extension: IVIScope with the ability to trigger on standard television signals.
IVIScopeRuntTrigger	no	Extension: IVIScope with the ability to trigger on runts.
IVIScopeGlitchTrigger	no	Extension: IVIScope with the ability to trigger on glitches.
IVIScopeWidthTrigger	no	Extension: IVIScope with the ability to trigger on a variety of conditions regarding pulse widths.
IVIScopeAcLineTrigger	no	Extension: IVIScope with the ability to trigger on zero crossings of a network supply voltage.
IVIScopeWaveformMeas	no	Extension: IVIScope with the ability to calculate waveform measurements, such as rise time or frequency.
IVIScopeMinMaxWaveform	no	Extension: IVIScope with the ability to acquire a minimum and maximum waveforms that correspond to the same time range.
IVIScopeProbeAutoSense	no	Extension: IVIScope with the ability to automatically sense the probe attenuation of an attached probe.
IVIScopeContinuous Acquisition	no	Extension: IVIScope with the ability to continuously acquire data from the input and display it on the screen.
IVIScopeAverage Acquisition	no	Extension: IVIScope with the ability to create a waveform that is the average of multiple waveform acquisitions.
IVIScopeSampleMode	no	Extension: IVIScope with the ability to return the actual sample mode.
IVIScopeTrigger Modifier	no	Extension: IVIScope with the ability to modify the behavior of the triggering subsystem in the absence of a expected trigger.
IVIScopeAutoSetup	no	Extension: IVIScope with the automatic configuration ability.

IVIDigitizer Supported Class Capabilities

Feature	Supported	Description of Feature
IVIDigitizerBase	yes	Base Capabilities of the IVIDigitizer specification. This group includes the capability to acquire waveforms using edge triggering.
IVIDigitizerMultiRecordAcquisition	yes	Extension: IVIDigitizer with the ability to do multi-record acquisitions.
IVIDigitizerBoardTemperature	no	Extension: IVIDigitizer with the ability to report the temperature of the digitizer.
IVIDigitizerChannelFilter	no	Extension: IVIDigitizer with the ability to control the channel input filter frequency.
IVIDigitizerChannelTemperature	no	Extension: IVIDigitizer with the ability to report the temperature of indIVIdual digitizer channels.
IVIDigitizerTimeInterleavedChannels	no	Extension: IVIDigitizer with the ability to combine two or more input channels to achieve higher acquisitions rates and/or record lengths.
IVIDigitizerDataInterleavedChannels	no	Extension: IVIDigitizer with the ability to interleave the data from two or more input channels, usually to create complex (I/Q) data.
IVIDigitizerReferenceOscillator	no	Extension: IVIDigitizer with the ability to use an external reference oscillator.
IVIDigitizerSampleClock	yes	Extension: IviDigitizer with the ability to use an external sample clock.
IVIDigitizerSampleMode	no	Extension: IVIDigitizer with the ability to control whether the digitizer is using real-time or equivalent-time sampling.
IVIDigitizerSelfCalibration	yes	Extension: IVIDigitizer with the ability to perform self calibration.
IVIDigitizerDownconversion	no	Extension: IVIDigitizer with the ability to do frequency translation or downconversion in hardware.
IVIDigitizerArm	no	Extension: IVIDigitizer with the ability to arm on positive or negative edges.
IVIDigitizerMultiArm	no	Extension: IVIDigitizer with the ability to arm on one or more sources.
IVIDigitizerGlitchArm	no	Extension: IVIDigitizer with the ability to arm on glitches.
IVIDigitizerRuntArm	no	Extension: IVIDigitizer with the ability to arm on runts.
IVIDigitizerSoftwareArm	no	Extension: IVIDigitizer with the ability to arm acquisitions.
IVIDigitizerTVArm	no	Extension: IVIDigitizer with the ability to arm on standard TV signals.
IVIDigitizerWidthArm	no	Extension: IVIDigitizer with the ability to arm on a variety of conditions regarding pulse widths.
IVIDigitizerWindowArm	no	Extension: IVIDigitizer with the ability to arm on signals entering or leaving a defined voltage range.
IVIDigitizerTriggerModifier	no	Extension: IVIDigitizer with the ability to perform an alternative triggering function in the event that the specified trigger event doesn't occur.
IVIDigitizerMultiTrigger	yes	Extension: IVIDigitizer with the ability to trigger on one or more sources.
IVIDigitizerPretriggerSamples	yes	Extension: IVIDigitizer with the ability to specify a number of samples to fill up the data buffer with pre-trigger data.
IVIDigitizerTriggerHoldoff	no	Extension: IVIDigitizer with the ability to specify a length of time after the digitizer detects a trigger during which the digitizer ignores additional triggers.
IVIDigitizerGlitchTrigger	no	Extension: IVIDigitizer with the ability to trigger on glitches.
IVIDigitizerRuntTrigger	no	Extension: IVIDigitizer with the ability to trigger on runts.
IVIDigitizerSoftwareTrigger	no	Extension: IVIDigitizer with the ability to trigger acquisitions.
IVIDigitizerTVTrigger	no	Extension: IVIDigitizer with the ability to trigger on standard television signals.
IVIDigitizerWidthTrigger	no	Extension: IVIDigitizer with the ability to trigger on a variety of conditions regarding pulse widths.
IVIDigitizerWindowTrigger	yes	Extension: IVIDigitizer with the ability to trigger on signals entering or leaving a defined voltage range.

IVIFGen Supported Class Capabilities

Feature	Supported	Description of Feature
IviFgenBase	yes	Base Capabilities.
IviFgenArbFrequency	no	Extension: IVIFgen with the ability to generate arbitrary waveforms with user-defined sample rate.
IviFgenArbWfm	yes	Extension: IVIFgen with the ability to generate user-defined arbitrary waveforms.
IviFgenArbSeq	no	Extension: IVIFgen with the ability to generate of arbitrary sequences
IviFgenBurst	no	Extension: IVIFgen with the ability to generate discrete numbers of waveform cycles.
IviFgenInternalTrigger	no	Extension: IVIFgen with the ability to use internally generated triggers
IviFgenModulateAM	no	Extension: IVIFgen with the ability to apply amplitude modulation to an output signal
IviFgenModulateFM	no	Extension: IVIFgen with the ability to apply frequency modulation to an output signal
IviFgenSoftwareTrigger	no	Extension: IVIFgen with the ability to generate signals based on software triggers
IviFgenStdFunc	yes	Extension: IVIFgen with the ability to generate standard waveforms
IviFgenTrigger	no	Extension: IVIFgen with the ability to use user-definable trigger sources

Find more Information on IVI

The official IVI foundation webpage offers a lot of additional information on setup and programming of the IVI drivers using different environments.

General Information on IVI

->http://ivifoundation.org

The website of the IVI foundation offers several documents and detailed explanations for the useage of IVI drivers and the benefits.

IVI Getting Started Guides and Videos

-> http://ivifoundation.org/resources/default.aspx

In here you find getting started guides and videos for different environments:

- Using IVI with Visual C++
- Using IVI Visual C# and Visual Basic .NET
- Using IVI with LabVIEW
- Using IVI with LabWindows/CVI
- Using IVI with MATLAB
- Using IVI with Measure Foundry
- Using IVI with Visual Basic 6.0
- Using IVI with Keysight VEE Pro

Installation

Installer

The Spectrum IVI Driver Installer is shipped as an executable containing all IVI related software parts. There is only one installer for both 32 bit and 64 bit environments. The insaller automatically detects the components that are necessary to install.

Please be sure to have the latest drivers available. You find the current driver archieves on the Spectrum webpage www.spectrum-instrumentation.com available for download.

Shared Components

To improve users' experience when they combine drivers and other software from various vendors, it is important to have some key software components common to all implementations. In order to accomplish this, the IVI Foundation provides a standard set of shared components that must be used by all compliant drivers and ancillary software. These components provide services to drivers and driver clients that need to be common to all drivers, for instance, the administration of system-wide configuration.

The IVI shared components are available directly at the IVI Foundation homepage www.ivifoundation.org. Please download the lates version of the IVI shared components there.

The IVI Shared Component installer creates a directory structure to house the IVI Shared Components as well as IVI drivers themselves. The root of this directory structure is referred to as the IVI install directory [IVIInstallDir] and is typically located under [program files]\IVI Foundation\IVI.

Installation Procedure

Please stick to this installation order to avoid any problems with the drivers:

Spectrum Card locally installed

- Install card into the system as described in the hardware manual
- Start the system and let Windows install the hardware driver from USB-Stick or from your download folder
- Install the Spectrum Control Center
- Install the IVI shared components from www.ivifoundation.org
- Install the IVI driver package

Spectrum Card remotely installed

- Install card into the remote system as described in the hardware manual
- Start the remote system and let Windows install the hardware driver from USB-Stick or from your download folder
- Install the Spectrum Remote Package onto the remote PC as described in the manual
- Install the Spectrum Control Center on the host system
- Setup the remote connection inside the Control Center as described in the hardware manual
- Install the IVI shared components from www.ivifoundation.org
- Install the IVI driver package on the host system

Spectrum digitizerNETBOX/generatorNETBOX remotely controlled

- Connect the digitizerNETBOX/generatorNETBOX to your LAN or directly to your host PC
- Install the Spectrum Control Center on the host system
- Setup the remote connection inside the Control Center as described in the hardware manual
- Install the IVI shared components from www.ivifoundation.org
- Install the IVI driver package on the host system

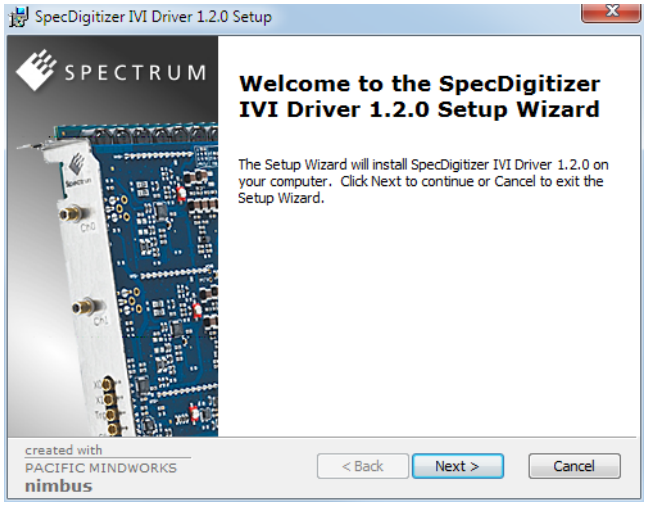
No Spectrum hardware available, only simulated cards

- Install the Spectrum Control Center on the system
- Setup one or more demo cards inside the Spectrum Control Center
- Install the IVI shared components from www.ivifoundation.org
- Install the IVI driver package on the host system

Installation of the IVI driver package

Please start the installation by doubleclicking the install file

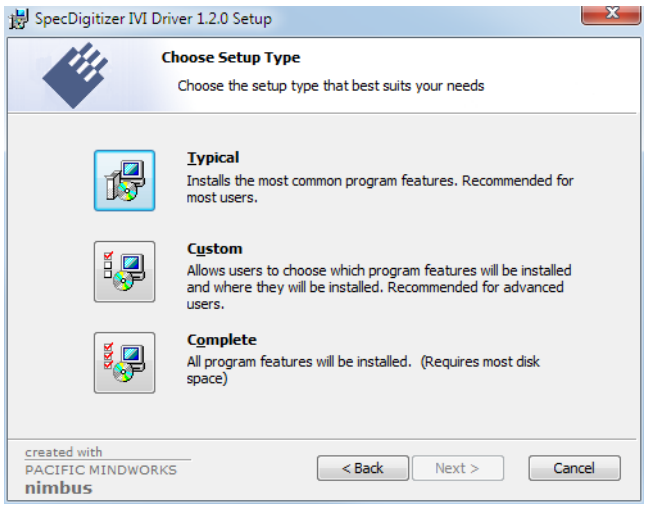
There is one installer for the IVI scope class driver and one installer for the IVI digitizer class driver. You may install one of them or both.



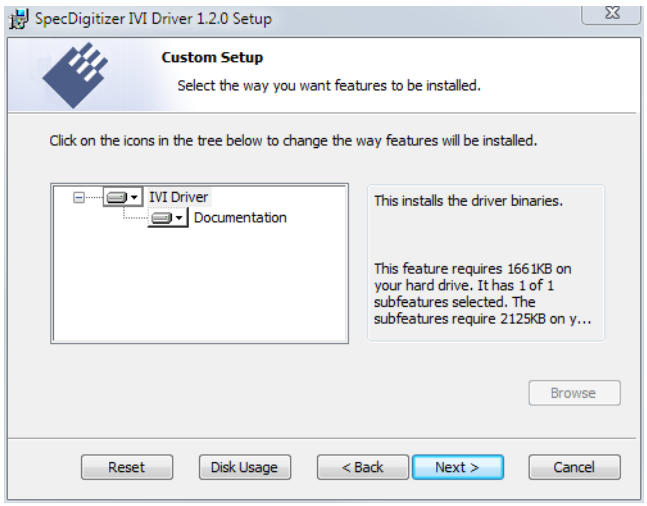
Select the setup type for the installation:

- Typical setup will install the most common program features
- Custom setup allows user to choose which program features will be installed.
- Complete setup will install all prgra, features.

Typical and Complete setup runs without any further user interaction and install the needed components of the driver.



The custom setup allows users to deselect certain parts of the driver package



Configuration Store

General Information

The IVI Configuration Server is the run-time module that is responsible for providing system database services to IVI based measurement system applications. Specifically, it provides system initialization and configuration information. The IVI Configuration Server is used by several of the IVI compliant modules. For instance, the Configuration Server indicates which physical instrument and IVI driver will be used by a particular application to provide a particular measurement capability.

Since a typical system intermixes instruments and drivers from multiple vendors this system configuration service needs to be accessed in a vendor independent fashion. Therefore, the IVI Configuration Server is an IVI shared component (that is, the code is owned by the IVI Foundation). The IVI Configuration Server is provided by the IVI Foundation because the architecture requires a single Configuration Server be installed on any system, therefore having a single shared implementation eliminates potential conflicts from divergent implementations.

The IVI Configuration Server is a single executable and one or more XML configuration stores (databases) made up of the following basic components:

- The physical database (known as the configuration store). A physical configuration store is a single XML file. APIs are available to read and write the data to arbitrary files, thus providing complex applications with the ability to directly manage system configurations.
- The API (and its implementation) used to read information from the configuration store(s). The IVI modules typically use this API when they are instantiated and configured.
- The API (and its implementation) to write information to the configuration store(s). This API is typically used by GUI or other applications that set up the initial configuration.
- The API (and its implementation) used to bind an instance of the Configuration Server code to a particular copy of the configuration information stored on a system. This includes appropriate algorithms for gaining access to the master configuration store.

Repeated Capabilities

In many instruments there are capabilities that are duplicated either identically or very similarly across the instrument. Such capabilities are called repeated capabilities. The IVI class-compliant APIs represent repeated capabilities by a parameter that indicates which instance of the duplicate capability this function is intended to access. The IVI C APIs include this parameter as an additional parameter to function calls.

The IVI Configuration Server provides a way for software modules to publish the functionality that is duplicated and the strings that the software module recognizes to access the repeated capabilities. The IVI Configuration Server also provides a way for the client to supply aliases for the physical identifiers recognized by the drivers.

The Spectrum IVI driver for example uses the channel index as repeated capability allowing to give channel names as an identifier.

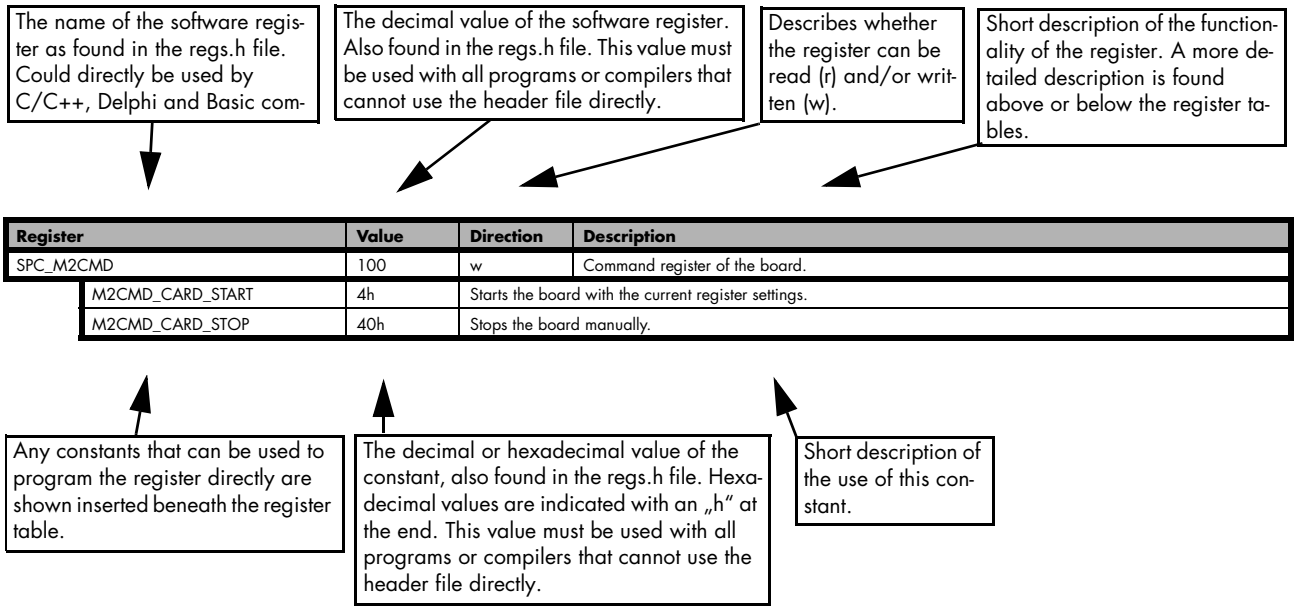
Programming the Board

Overview

The following chapters show you in detail how to program the different aspects of the board. For every topic there's a small example. For the examples we focused on Visual C++. However as shown in the last chapter the differences in programming the board under different programming languages are marginal. This manual describes the programming of the whole hardware family. Some of the topics are similar for all board versions. But some differ a little bit from type to type. Please check the given tables for these topics and examine carefully which settings are valid for your special kind of board.

Register tables

The programming of the boards is totally software register based. All software registers are described in the following form:



Register	Value	Direction	Description
SPC_M2CMD	100	w	Command register of the board.
M2CMD_CARD_START	4h	Starts the board with the current register settings.	
M2CMD_CARD_STOP	40h	Stops the board manually.	

If no constants are given below the register table, the dedicated register is used as a switch. All such registers are activated if written with a "1" and deactivated if written with a "0".

Programming examples

In this manual a lot of programming examples are used to give you an impression on how the actual mentioned registers can be set within your own program. All of the examples are located in a separated colored box to indicate the example and to make it easier to differ it from the describing text.

All of the examples mentioned throughout the manual are written in C/C++ and can be used with any C/C++ compiler for Windows or Linux.

Complete C/C++ Example

```
#include "../c_header/dlltyp.h"
#include "../c_header/regs.h"
#include "../c_header/spcm_drv.h"

#include <stdio.h>

int main()
    {
    drv_handle hDrv;                                        // the handle of the device
    int32 lCardType;                                        // a place to store card information

    hDrv = spcm_hOpen ("/dev/spcm0");                       // Opens the board and gets a handle
    if (!hDrv)                                              // check whether we can access the card
        return -1;

    spcm_dwGetParam_i32 (hDrv, SPC_PCITYP,  &lCardType);    // simple command, read out of card type
    printf ("Found card M2i/M3i/M4i/M4x/M2p.%04x in the system\n", lCardType & TYP_VERSIONMASK);
    spcm_vClose (hDrv);

    return 0;
    }
```

Initialization

Before using the card it is necessary to open the kernel device to access the hardware. It is only possible to use every device exclusively using the handle that is obtained when opening the device. Opening the same device twice will only generate an error code. After ending the driver use the device has to be closed again to allow later re-opening. Open and close of driver is done using the spcm_hOpen and spcm_vClose function as described in the “Driver Functions” chapter before.

Open/Close Example

```
drv_handle hDrv;                                        // the handle of the device

hDrv = spcm_hOpen ("/dev/spcm0");                       // Opens the board and gets a handle
if (!hDrv)                                              // check whether we can access the card
    {
    printf “Open failed\n”);
    return -1;
    }

... do any work with the driver

spcm_vClose (hDrv);
return 0;
```

Initialization of Remote Products

The only step that is different when accessing remotely controlled cards or digitizerNETBOXes is the initialization of the driver. Instead of the local handle one has to open the VISA string that is returned by the discovery function. Alternatively it is also possible to access the card directly without discovery function if the IP address of the device is known.

```
drv_handle hDrv;                                        // the handle of the device

hDrv = spcm_hOpen ("TCPIP::192.168.169.14::INSTR");     // Opens the remote board and gets a handle
if (!hDrv)                                              // check whether we can access the card
    {
    printf “Open of remote card failed\n”);
    return -1;
    }

...
```

Multiple cards are opened by indexing the remote card number:

```
hDrv = spcm_hOpen ("TCPIP::192.168.169.14::INSTR");         // Opens the remote board #0
                                                            // or alternatively
hDrv = spcm_hOpen ("TCPIP::192.168.169.14::INST0::INSTR");  // Opens the remote board #0
                                                            // all other boards require an index:
hDrv = spcm_hOpen ("TCPIP::192.168.169.14::INST1::INSTR");  // Opens the remote board #1
hDrv = spcm_hOpen ("TCPIP::192.168.169.14::INST2::INSTR");  // Opens the remote board #2
```

Error handling

If one action caused an error in the driver this error and the register and value where it occurs will be saved.

The driver is then locked until the error is read out using the error function spcm_dwGetErrorInfo_i32. Any calls to other functions will just return the error code ERR_LASTERR showing that there is an error to be read out.

This error locking functionality will prevent the generation of unseen false commands and settings that may lead to totally unexpected behavior. For sure there are only errors locked that result on false commands or settings. Any error code that is generated to report a condition to the user won’t lock the driver. As example the error code ERR_TIMEOUT showing that the a timeout in a wait function has occurred won’t lock the driver and the user can simply react to this error code without reading the complete error function.

As a benefit from this error locking it is not necessary to check the error return of each function call but just checking the error function once at the end of all calls to see where an error occurred. The enhanced error function returns a complete error description that will lead to the call that produces the error.

Example for error checking at end using the error text from the driver:

```
char szErrorText[ERRORTEXTLEN];

spcm_dwSetParam_i64 (hDrv, SPC_SAMPLERATE, 1000000);                    // correct command
spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE, -345);                          // faulty command
spcm_dwSetParam_i64 (hDrv, SPC_POSTTRIGGER, 1024);                      // correct command
if (spcm_dwGetErrorInfo_i32 (hDrv, NULL, NULL, szErrorText) != ERR_OK)  // check for an error
    {
    printf (szErrorText);                                               // print the error text
    spcm_vClose (hDrv);                                                 // close the driver
    exit (0);                                                           // and leave the program
    }
```

This short program then would generate a printout as:

```
Error ocurred at register SPC_MEMSIZE with value -345: value not allowed
```

All error codes are described in detail in the appendix. Please refer to this error description and the description of the software register to examine the cause for the error message.

Any of the parameters of the spcm_dwGetErrorInfo_i32 function can be used to obtain detailed information on the error. If one is not interested in parts of this information it is possible to just pass a NULL (zero) to this variable like shown in the example. If one is not interested in the error text but wants to install its own error handler it may be interesting to just read out the error generating register and value.

Example for error checking with own (simple) error handler:

```
uint32 dwErrorReg;
int32  lErrorValue;
uint32 dwErrorCode;

spcm_dwSetParam_i64 (hDrv, SPC_SAMPLERATE, 1000000);                    // correct command
spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE, -345);                          // faulty command
spcm_dwSetParam_i64 (hDrv, SPC_POSTTRIGGER, 1024);                      // correct command
dwErrorCode = spcm_dwGetErrorInfo_i32 (hDrv, &dwErrorReg, &lErrorValue, NULL);
if (dwErrorCode)                                                        // check for an error
    {
    printf ("Errorcode: %d in register %d at value %d\n", lErrorCode, dwErrorReg, lErrorValue);
    spcm_vClose (hDrv);                                                 // close the driver
    exit (0);                                                           // and leave the program
    }
```

Gathering information from the card

When opening the card the driver library internally reads out a lot of information from the on-board eeprom. The driver also offers additional information on hardware details. All of this information can be read out and used for programming and documentation. This chapter will show all general information that is offered by the driver. There is also some more information on certain parts of the card, like clock machine or trigger machine, that is described in detail in the documentation of that part of the card.

All information can be read out using one of the spcm_dwGetParam functions. Please stick to the "Driver Functions" chapter for more details on this function.

Card type

The card type information returns the specific card type that is found under this device. When using multiple cards in one system it is highly recommended to read out this register first to examine the ordering of cards. Please don't rely on the card ordering as this is based on the BIOS, the bus connections and the operating system.

Register	Value	Direction	Description
SPC_PCITYP	2000	read	Type of board as listed in the table below.

One of the following values is returned, when reading this register. Each card has its own card type constant defined in regs.h. Please note that when reading the card information as a hex value, the lower word shows the digits of the card name while the upper word is a indication for the used bus type.

.

Card type	Card type as defined in regs.h	Value hexadecimal	Value decimal		Card type	Card type as defined in regs.h	Value hexadecimal	Value decimal
M3i.4830	TYP_M3I4830	54830h	346160		M3i.4141	TYP_M3I4841	54841h	346177
M3i.4831	TYP_M3I4831	54831h	346161		M3i.4160	TYP_M3I4860	54860h	346208
M3i.4840	TYP_M3I4840	54840h	346176		M3i.4161	TYP_M3I4861	54861h	346209
M3i.4830-exp	TYP_M3I4830EXP	64830h	411696		M3i.4841-exp	TYP_M3I4841EXP	64841h	411713
M3i.4831-exp	TYP_M3I4831EXP	64831h	411697		M3i.4860-exp	TYP_M3I4860EXP	64860h	411744
M3i.4840-exp	TYP_M3I4840EXP	64840h	411712		M3i.4861-exp	TYP_M3I4861EXP	64861h	411745

Hardware version

Since all of the boards from Spectrum are modular boards, they consist of one base board and one or two piggy-back front-end modules and eventually of an extension module like the star-hub. Each of these three kinds of hardware has its own version register. Normally you do not need this information but if you have a support question, please provide the revision together with it.

Register	Value	Direction	Description
SPC_PCIVERSION	2010	read	Base card version: the upper 16 bit show the hardware (PCB) version, the lower 16 bit show the firmware version.
SPC_PCIMODULEVERSION	2012	read	Module version: the upper 16 bit show the hardware (PCB) version, the lower 16 bit show the firmware version.

If your board has a additional piggy-back extension module mounted you can get the hardware version with the following register.

Register	Value	Direction	Description
SPC_PCIEXTVERSION	2011	read	Extension module version: the upper 16 bit show the hardware (PCB) version, the lower 16 bit show the firmware version.

Firmware versions

All the cards from Spectrum typically contain multiple programmable devices such as FPGAs, CPLDs and the like. Each of these have their own dedicated firmware version. This version information is readable for each device through the various version registers. Normally you do not need this information but if you have a support question, please provide us with this information. Please note that number of devices and hence the readable firmware information is card series dependent:

Register	Value	Direction	Description	Available for				
				M2i	M3i	M4i	M4x	M2p
SPCM_FW_CTRL	210000	read	Main control FPGA version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the standard release firmware, the type has always a value of 1.	X	X	X	X	X
SPCM_FW_CTRL_GOLDEN	210001	read	Main control FPGA golden version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the golden (recovery) firmware, the type has always a value of 2.	–	–	X	X	X
SPCM_FW_CLOCK	210010	read	Clock distribution version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the standard release firmware, the type has always a value of 1.	X	–	–	–	–
SPCM_FW_CONFIG	210020	read	Configuration controller version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the standard release firmware, the type has always a value of 1.	X	X	–	–	–
SPCM_FW_MODULEA	210030	read	Front-end module A version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the standard release firmware, the type has always a value of 1.	X	X	X	X	X
SPCM_FW_MODULEB	210031	read	Front-end module B version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the standard release firmware, the type has always a value of 1. The version is zero if no second front-end module is installed on the card.	X	–	–	–	X
SPCM_FW_MODEXTRA	210050	read	Extension module (Star-Hub) version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the standard release firmware, the type has always a value of 1. The version is zero if no sextension module is installed on the card.	X	X	X	–	X
SPCM_FW_POWER	210060	read	Power controller version: the upper 16 bit show the firmware type, the lower 16 bit show the firmware version. For the standard release firmware, the type has always a value of 1.	–	–	X	X	X

Cards that do provide a golden recovery image for the main control FPGA, the currently booted firmware can additionally read out:

Register	Value	Direction	Description	M2i	M3i	M4i	M4x	M2p
SPCM_FW_CTRL_ACTIVE	210002	read	Cards that do provide a golden (recovery) fiwmware additionally have a register to read out the version information of the currently loaded firmware version string, do determine if it is standard or golden. The hexadecimal 32bit format is: TVVVUUUUh T: the currently booted type (1: standard, 2: golden) V: the version U: unused, in production versions always zero	–	–	X	X	X

Production date

This register informs you about the production date, which is returned as one 32 bit long word. The lower word is holding the information about the year, while the upper word informs about the week of the year.

Register	Value	Direction	Description
SPC_PCIDATE	2020	read	Production date: week in bits 31 to 16, year in bits 15 to 0

The following example shows how to read out a date and how to interpret the value:

```
spcm_dwGetParam_i32 (hDrv, SPC_PCIDATE,     &lProdDate);
printf ("Production: week &d of year &d\n", (lProdDate >> 16) & 0xffff, lProdDate & 0xffff);
```

Last calibration date (analog cards only)

This register informs you about the date of the last factory calibration. When receiving a new card this date is similar to the delivery date when the production calibration is done. When returning the card to calibration this information is updated. This date is not updated when just doing an on-board calibration by the user. The date is returned as one 32 bit long word. The lower word is holding the information about the year, while the upper word informs about the week of the year.

Register	Value	Direction	Description
SPC_CALIBDATE	2025	read	Last calibration date: week in bit 31 to 16, year in bit 15 to 0

Serial number

This register holds the information about the serial number of the board. This number is unique and should always be sent together with a support question. Normally you use this information together with the register SPC_PCITYP to verify that multiple measurements are done with the exact same board.

Register	Value	Direction	Description
SPC_PCISERIALNO	2030	read	Serial number of the board

Maximum possible sampling rate

This register gives you the maximum possible sampling rate the board can run. The information provided here does not consider any restrictions in the maximum speed caused by special channel settings. For detailed information about the correlation between the maximum sampling rate and the number of activated channels please refer to the according chapter.

Register	Value	Direction	Description
SPC_PCISAMPLERATE	2100	read	Maximum sampling rate in Hz as a 64 bit integer value

Installed memory

This register returns the size of the installed on-board memory in bytes as a 64 bit integer value. If you want to know the amount of samples you can store, you must regard the size of one sample of your card. All 8 bit A/D and D/A cards use only one byte per sample, while all other A/D and D/A cards with 12, 14 and 16 bit resolution use two bytes to store one sample. All digital cards need one byte to store 8 data bits.

Register	Value	Direction	Description
SPC_PCIMEMSIZE	2110	read _i32	Installed memory in bytes as a 32 bit integer value. Maximum return value will 1 GByte. If more memory is installed this function will return the error code ERR_EXCEEDINT32.
SPC_PCIMEMSIZE	2110	read _i64	Installed memory in bytes as a 64 bit integer value

The following example is written for a „two bytes" per sample card (12, 14 or 16 bit board), on any 8 bit card memory in MSamples is similar to memory in MBytes.

```
spcm_dwGetParam_i64 (hDrv, SPC_PCIMEMSIZE,     &llInstMemsize);
printf ("Memory on card: %d MBytes\n", (int32) (llInstMemsize /1024/1024));
printf ("              : %d MSamples\n", (int32) (llInstMemsize /1024/1024/2));
```

Installed features and options

The SPC_PCIFEATURES register informs you about the features, that are installed on the board. If you want to know about one option being installed or not, you need to read out the 32 bit value and mask the interesting bit. In the table below you will find every feature that may be installed on a M2i/M3i/M4i/M4x/M2p card. Please refer to the ordering information to see which of these features are available for your card series.

Register	Value	Direction	Description
SPC_PCIFEATURES	2120	read	PCI feature register. Holds the installed features and options as a bitfield. The read value must be masked out with one of the masks below to get information about one certain feature.
SPCM_FEAT_MULTI	1h		Is set if the feature Multiple Recording / Multiple Replay is available.
SPCM_FEAT_GATE	2h		Is set if the feature Gated Sampling / Gated Replay is available.
SPCM_FEAT_DIGITAL	4h		Is set if the feature Digital Inputs / Digital Outputs is available.
SPCM_FEAT_TIMESTAMP	8h		Is set if the feature Timestamp is available.
SPCM_FEAT_STARHUB6_EXTM	20h		Is set on the card, that carries the star-hub extension or piggy-back module for synchronizing up to 6 cards (M2p).
SPCM_FEAT_STARHUB8_EXTM	20h		Is set on the card, that carries the star-hub extension or piggy-back module for synchronizing up to 8 cards (M4i).
SPCM_FEAT_STARHUB4	20h		Is set on the card, that carries the star-hub piggy-back module for synchronizing up to 4 cards (M3i).
SPCM_FEAT_STARHUB5	20h		Is set on the card, that carries the star-hub piggy-back module for synchronizing up to 5 cards (M2i).
SPCM_FEAT_STARHUB16_EXTM	40h		Is set on the card, that carries the star-hub piggy-back module for synchronizing up to 16 cards (M2p).
SPCM_FEAT_STARHUB8	40h		Is set on the card, that carries the star-hub piggy-back module for synchronizing up to 8 cards (M3i).
SPCM_FEAT_STARHUB16	40h		Is set on the card, that carries the star-hub piggy-back module for synchronizing up to 16 cards (M2i).
SPCM_FEAT_ABA	80h		Is set if the feature ABA mode is available.
SPCM_FEAT_BASEXIO	100h		Is set if the extra BaseXIO option is installed. The lines can be used for asynchronous digital I/O, extra trigger or timestamp reference signal input.
SPCM_FEAT_AMPLIFIER_10V	200h		Arbitrary Waveform Generators only: card has additional set of calibration values for amplifier card.
SPCM_FEAT_STARHUBSYSMASTER	400h		Is set in the card that carries a System Star-Hub Master card to connect multiple systems (M2i).
SPCM_FEAT_DIFFMODE	800h		M2i.30xx series only: card has option -diff installed for combining two SE channels to one differential channel.
SPCM_FEAT_SEQUENCE	1000h		Only available for output cards or I/O cards: Replay sequence mode available.
SPCM_FEAT_AMPMODULE_10V	2000h		Is set on the card that has a special amplifier module for mounted (M2i.60xx/61xx only).
SPCM_FEAT_STARHUBSYSSLAVE	4000h		Is set in the card that carries a System Star-Hub Slave module to connect with System Star-Hub master systems (M2i).
SPCM_FEAT_NETBOX	8000h		The card is physically mounted within a digitizerNETBOX or generatorNETBOX.
SPCM_FEAT_REMOTESERVER	10000h		Support for the Spectrum Remote Server option is installed on this card.
SPCM_FEAT_SCAPP	20000h		Support for the SCAPP option allowing CUDA RDMA access to supported graphics cards for GPU calculations (M4i and M2p)
SPCM_FEAT_DIG16_SMB	40000h		M2p: Set if option M2p.xxxx-DigSMB is installed, adding16 additional digital I/Os via SMB connectors.
SPCM_FEAT_DIG16_FX2	80000h		M2p: Set if option M2p.xxxx-DigFX2 is installed, adding16 additional digital I/Os via FX2 multipin connectors.
SPCM_FEAT_DIGITALBWFILTER	100000h		A digital (boxcar) bandwidth filter is available that can be globally enabled/disabled for all channels.
SPCM_FEAT_CUSTOMMOD_MASK	F0000000h		The upper 4 bit of the feature register is used to mark special custom modifications. This is only used if the card has been specially customized. Please refer to the extra documentation for the meaning of the custom modifications. (M2i/M3i). For M4i, M4x and M2p cards see „Custom modifications" chapter instead.

The following example demonstrates how to read out the information about one feature.

```
spcm_dwGetParam_i32 (hDrv, SPC_PCIFEATURES, &lFeatures);
if (lFeatures & SPCM_FEAT_DIGITAL)
    printf("Option digital inputs/outputs is installed on your card");
```

The following example demonstrates how to read out the custom modification code.

```
spcm_dwGetParam_i32 (hDrv, SPC_PCIFEATURES, &lFeatures);
lCustomMod = (lFeatures >> 28) & 0xF;
if (lCustomMod != 0)
    printf("Custom modification no. %d is installed.", lCustomMod);
```

Installed extended Options and Features

Some cards (such as M4i/M4x/M2p cards) can have advanced features and options installed. This can be read out with with the following register:

Register	Value	Direction	Description
SPC_PCIEXTFEATURES	2121	read	PCI extended feature register. Holds the installed extended features and options as a bitfield. The read value must be masked out with one of the masks below to get information about one certain feature.

SPCM_FEAT_EXTFW_SEGSTAT	1h	Is set if the firmware option „Block Statistics“ is installed on the board, which allows certain statistics to be on-board calculated for data being recorded in segmented memory modes, such as Multiple Recording or ABA.
SPCM_FEAT_EXTFW_SEGAVERAGE	2h	Is set if the firmware option „Block Average“ is installed on the board, which allows on-board hardware averaging of data being recorded in segmented memory modes, such as Multiple Recording or ABA.
SPCM_FEAT_EXTFW_BOXCAR	4h	Is set if the firmware mode „Boxcar Average“ is supported in the installed firmware version.

Miscellaneous Card Information

Some more detailed card information, that might be useful for the application to know, can be read out with the following registers:

Register	Value	Direction	Description
SPC_MIINST_MODULES	1100	read	Number of the installed front-end modules on the card.
SPC_MIINST_CHPERMODULE	1110	read	Number of channels installed on one front-end module.
SPC_MIINST_BYTESPERSAMPLE	1120	read	Number of bytes used in memory by one sample.
SPC_MIINST_BITSPERSAMPLE	1125	read	Resolution of the samples in bits.
SPC_MIINST_MAXADCVALUE	1126	read	Decimal code of the full scale value.
SPC_MIINST_MINEXTCLOCK	1145	read	Minimum external clock that can be fed in for direct external clock (if available for card model).
SPC_MIINST_MAXEXTCLOCK	1146	read	Maximum external clock that can be fed in for direct external clock (if available for card model).
SPC_MIINST_MINEXTREFCLOCK	1148	read	Minimum external clock that can be fed in as a reference clock.
SPC_MIINST_MAXEXTREFCLOCK	1149	read	Maximum external clock that can be fed in as a reference clock.
SPC_MIINST_ISDEMOCARD	1175	read	Returns a value other than zero, if the card is a demo card.

Function type of the card

This register register returns the basic type of the card:

Register	Value	Direction	Description
SPC_FNCTYPE	2001	read	Gives information about what type of card it is.
SPCM_TYPE_AI	1h	Analog input card (analog acquisition; the M2i.4028 and M2i.4038 also return this value)	
SPCM_TYPE_AO	2h	Analog output card (arbitrary waveform generators)	
SPCM_TYPE_DI	4h	Digital input card (logic analyzer card)	
SPCM_TYPE_DO	8h	Digital output card (pattern generators)	
SPCM_TYPE_DIO	10h	Digital I/O (input/output) card, where the direction is software selectable.	

Used type of driver

This register holds the information about the driver that is actually used to access the board. Although the driver interface doesn't differ between Windows and Linux systems it may be of interest for a universal program to know on which platform it is working.

Register	Value	Direction	Description
SPC_GETDRVTYPE	1220	read	Gives information about what type of driver is actually used
DRVTYP_LINUX32	1	Linux 32bit driver is used	
DRVTYP_WDM32	4	Windows WDM 32bit driver is used (XP/Vista/Windows 7/Windows 8/Windows 10).	
DRVTYP_WDM64	5	Windows WDM 64bit driver is used by 64bit application (XP64/Vista/Windows 7/Windows 8/Windows 10).	
DRVTYP_WOW64	6	Windows WDM 64bit driver is used by 32bit application (XP64/Vista/Windows 7/Windows 8/ Windows 10).	
DRVTYP_LINUX64	7	Linux 64bit driver is used	

Driver version

This register holds information about the currently installed driver library. As the drivers are permanently improved and maintained and new features are added user programs that rely on a new feature are requested to check the driver version whether this feature is installed.

Register	Value	Direction	Description
SPC_GETDRVVERSION	1200	read	Gives information about the driver library version

The resulting 32 bit value for the driver version consists of the three version number parts shown in the table below:

Driver Major Version	Driver Minor Version	Driver Build
8 Bit wide: bit 24 to bit 31	8 Bit wide, bit 16 to bit 23	16 Bit wide, bit 0 to bit 15

Kernel Driver version

This register informs about the actually used kernel driver. Windows users can also get this information from the device manager. Please refer to the „Driver Installation“ chapter. On Linux systems this information is also shown in the kernel message log at driver start time.

Register	Value	Direction	Description
SPC_GETKERNELVERSION	1210	read	Gives information about the kernel driver version.

The resulting 32 bit value for the driver version consists of the three version number parts shown in the table below:

Driver Major Version	Driver Minor Version	Driver Build
8 Bit wide: bit 24 to bit 31	8 Bit wide, bit 16 to bit 23	16 Bit wide, bit 0 to bit 15

The following example demonstrates how to read out the kernel and library version and how to print them.

```
spcm_dwGetParam_i32 (hDrv, SPC_GETDRVVERSION,    &lLibVersion);
spcm_dwGetParam_i32 (hDrv, SPC_GETKERNELVERSION, &lKernelVersion);
printf("Kernel V %d.%d build %d\n",lKernelVersion >> 24, (lKernelVersion >> 16) & 0xff, lKernelVersion & 0xffff);
printf("Library V %d.%d build %d\n",lLibVersion >> 24, (lLibVersion >> 16) & 0xff, lLibVersion & 0xffff);
```

This small program will generate an output like this:

```
Kernel V 1.11 build 817
Library V 1.1 build 854
```

Reset

Every Spectrum card can be reset by software. Concerning the hardware, this reset is the same as the power-on reset when starting the host computer. In addition to the power-on reset, the reset command also brings all internal driver settings to a defined default state. A software reset is automatically performed, when the driver is first loaded after starting the host system.

It is recommended, that every custom written program performs a software reset first, to be sure that the driver is in a defined state independent from possible previous setting.

Performing a board reset can be easily done by the related board command mentioned in the following table.

Register	Value	Direction	Description
SPC_M2CMD	100	w	Command register of the board.
M2CMD_CARD_RESET	1h	A software and hardware reset is done for the board. All settings are set to the default values. The data in the board's on-board memory will be no longer valid. Any output signals like trigger or clock output will be disabled.	

digitizerNETBOX/generatorNETBOX specific registers

Information about the digitizerNETBOX/generatorNETBOX, in which the card is installed, can be read out via the card handle.

The following digitizerNETBOX/generatorNETBOX specific information registers can be used:

Register	Value	Direction	Description
SPC_NETBOX_TYPE	400000	read	Hex coded version of the digitizerNETBOX/generatorNETBOX, example 02490110h: bit 24 to 31: Series: example 02h = DN2 bit 16 to 23: Family: example 49h = 49 bit 8 to 15: Speed grade: example 01h = 1 bit 0 to 7: Channels: example 10h = 16 Decoded example: DN2.491-16
SPC_NETBOX_SERIALNO	400001	read	Serial number of the digitizerNETBOX/generatorNETBOX itself. In most cases the serial numbers of the digitizerNETBOX/generatorNETBOX and the embedded cards are consecutive but there is no guarantee for this.
SPC_NETBOX_PRODUCTIONDATE	400002	read	Production date: week in bit 31 to 16, year in bit 15 to 0
SPC_NETBOX_HWVERSION	400003	read	The hardware version of the digitizerNETBOX/generatorNETBOX products
SPC_NETBOX_SWVERSION	400004	read	The software version of the installed remote server
SPC_NETBOX_FEATURES	400005	read	Features of the digitizerNETBOX/generatorNETBOX. Holds the installed features and options as a bitfield. The read value must be masked out with one of the masks below to get information about one certain feature.
NETBOX_FEAT_DCPOWER	1h	Is set if one of the DC power options are installed in the system.	
NETBOX_FEAT_BOOTATPOWERON	2h	Is set if the special feature automatic boot on power on is installed. This would allow remote devices to automatically reboot after a failure of the power supply.	
NETBOX_FEAT_EMBEDDEDSERVER	4h	Is set if the option Embedded Server is installed.	

Register	Value	Direction	Description
SPC_NETBOX_CUSTOM	400006	read	Custom code for custom modifications of the digitizerNETBOX/generatorNETBOX.
SPC_NETBOX_WAKEONLAN	400007	write	This command is issed to wake a digitizerNETBOX/generatorNETBOX that is currently in standby-mode with a special wake-on-lan message. Please note that the card handle is NULL in this case as there is no opened card here. The argument is the MAC address of that device
SPC_NETBOX_MACADDRESS	400008	read	Reads out the MAC address of the digitizerNETBOX/generatorNETBOX.
SPC_NETBOX_LANIDFLASH	400009	write	By writing 1 to this register, one can start the autoamtic flashing of the LAN Id to detect a particulat digitizerNETBOX/generatorNETBOX that is installed in a Rack of multiple digitizerNETBOX or generatorNETBOX devices. Writing a 0 to this register will stop the flashing again.
SPC_NETBOX_TEMPERATURE	400010	read	Read out the temperature inside the digitizerNETBOX/generatorNETBOX (same as displayed in the webinterface status information) in Kelvin.
SPC_NETBOX_SHUTDOWN	400011	write	Remotely shut down the digitizerNETBOX/generatorNETBOX. Value must be set to 0.
SPC_NETBOX_RESTART	400012	write	Remotely restart the digitizerNETBOX/generatorNETBOX. Value must be set to 0.

Analog Inputs

Channel Selection

One key setting that influences all other possible settings is the channel enable register. A unique feature of the Spectrum cards is the possibility to program the number of channels you want to use. All on-board memory can then be used by these activated channels.

This description shows you the channel enable register for the complete card family. However, your specific board may have less channels depending on the card type that you have purchased and therefore does not allow you to set the maximum number of channels shown here.

Register	Value	Direction	Description
SPC_CHENABLE	11000	read/write	Sets the channel enable information for the next board run.
CHANNEL0	1	Activates channel 0	
CHANNEL1	2	Activates channel 1	

The channel enable register is set as a bitmap. That means one bit of the value corresponds to one channel to be activated. To activate more than one channel the values have to be combined by a bitwise OR.

Example showing how to activate 2 channels:

```
spcm_dwSetParam_i32 (hDrv, SPC_CHENABLE, CHANNEL0 | CHANNEL1);
```

The following table shows all allowed settings for the channel enable register.

Channels to activate				
Ch0	Ch1	Values to program	Value as hex	Value as decimal
X		CHANNEL0	1h	1
	X	CHANNEL1	2h	2
X	X	CHANNEL0 | CHANNEL1	3h	3

Any channel activation mask that is not shown here is not valid. If programming another channel activation the driver will return with an error.

To help user programs it is also possible to read out the number of activated channels that correspond to the currently programmed bitmap.

Register	Value	Direction	Description
SPC_CHCOUNT	11001	read	Reads back the number of currently activated channels.

Reading out the channel enable information can be done directly after setting it or later like this:

```
spcm_dwSetParam_i32 (hDrv, SPC_CHENABLE, CHANNEL0 | CHANNEL1);
spcm_dwGetParam_i32 (hDrv, SPC_CHENABLE, &lActivatedChannels);
spcm_dwGetParam_i32 (hDrv, SPC_CHCOUNT, &lChCount);

printf ("Activated channels bitmask is: 0x%08x\n", lActivatedChannels);
printf ("Number of activated channels with this bitmask: %d\n", lChCount);
```

Assuming that the two channels are available on your card the program will have the following output:

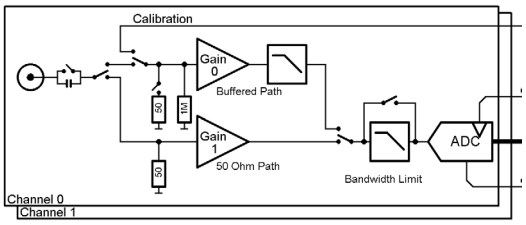
```
Activated channels bitmask is: 0x00000003
Number of activated channels with this bitmask: 2
```

Important note on channel selection

As some of the manuals passages are used in more than one hardware manual most of the registers and channel settings throughout this handbook are described for the maximum number of possible channels that are available on one card of the current series. There can be less channels on your actual type of board or bus-system. Please refer to the technical data section to get the actual number of available channels.

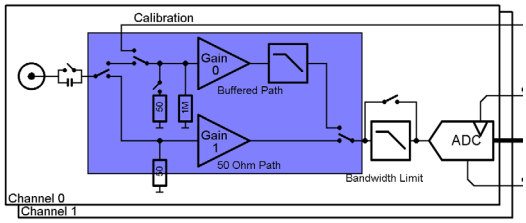
Setting up the inputs

This analog acquisition board uses separate input stages and converters on each channel. This gives you the possibility to set up the desired and concerning your application best suiting input range also separately for each channel. All input stage related settings can easily be set by the corresponding input registers. The table below shows the available input stage registers and possible standard values for your type of board. As there are also modified versions available with different input ranges it is recommended to read out the currently available input ranges as shown later in this chapter.



Input Path

Each input stage consists of different input paths each with different available settings and features. Please refer to the technical data section to get details on the differences of the input paths.



Offering different input paths gives the choice to adopt the cards input stage to the specific application in the best technical way by either using a high frequency 50 ohm path to have full bandwidth and best dynamic performance or by using a buffered path with all features but limited bandwidth and dynamic performance.

All following settings are related to the selected input path. To read available features like input ranges or termination settings it is first necessary to set the input path for which the features are to be read.

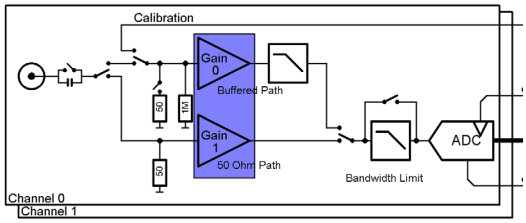
Register	Value	Direction	Description
SPC_READAIPATHCOUNT	3120	read	Returns the number of available analog input paths
SPC_READAIPATH	3121	read/write	Selects the input path which is used to read out the features. Please note that this settings does not change the current path selection.

The following registers show the available input path settings

Register	Value	Direction	Description
SPC_PATH0	30090	read/write	Selects the analog input path for channel 0 (default path is path 0)
SPC_PATH1	30190	read/write	Selects the analog input path for channel 1 (default path is path 0)
	0	Input Path 0: Buffered inputs	
	1	Input Path 1: HF input with fixed 50 ohm termination	

Input ranges

This analog acquisition board has several different input ranges for each channel. This gives you the possibility to set up the desired and concerning your application best suiting input range also separately for each channel. The input ranges can easily be set by the corresponding input registers. The table below shows the available input registers and possible standard ranges for your type of board. As there are also modified versions available with different input ranges it is recommended to read out the currently available input ranges as shown later in this chapter.



Please note that the available ranges need to be read out separately for each input path. Please set the register SPC_READAIPATH as shown above to select the input path for which the settings should be read. The available Input rages are read out using the following registers.

Register	Value	Direction	Description
SPC_READAIPATH	3121	read/write	Selects the input path which is used to read out the features.
SPC_READIRCOUNT	3000	read	Returns the number of available input ranges for the input path selected by SPC_READAIPATH
SPC_READRANGEMIN0	4000	read	Reads the lower border of input range 0 in mV
SPC_READRANGEMIN1	4001	read	Reads the lower border of input range 1 in mV
...	...	...	
SPC_READRANGEMAX0	4100	read	Reads the upper border of input range 0 in mV
SPC_READRANGEMAX1	4101	read	Reads the upper border of input range 1 in mV
...	...	...	

The following example reads out the number of available input ranges and reads and prints the minimum and maximum value of all input ranges.

```
spcm_dwGetParan_i32 (hDrv, SPC_READAIPATHCOUNT,   &lNumOfPaths);
for (lPath = 0; lPath < lNumOfPaths; lPath++)
    {
    spcm_dwSetParan_i32 (hDrv, SPC_READAIPATH,     lPath)
    spcm_dwGetParam_i32 (hDrv, SPC_READIRCOUNT,    &lNumberOfRanges);
    for (i = 0; i < lNumberOfRanges; i++)
        {
        spcm_dwGetParam_i32 (hDrv, SPC_READRANGEMIN0 + i, &lMinimumInputRage);
        spcm_dwGetParam_i32 (hDrv, SPC_READRANGEMAX0 + i, &lMaximumInputRange);
        printf („Path %d Range %d: %d mV to %d mV\n“, lPath, i, lMinimumInputRange, lMaximumInputRange);
        }
    }
```

The input range is selected individually for each channel. Please note that the correct input path needs to be set

Register	Value	Direction	Description
SPC_AMP0	30010	read/write	Defines the input range of channel0.
SPC_AMP1	30110	read/write	Defines the input range of channel1.

Standard Input ranges of path 0 (Buffered):

200	± 200 mV calibrated input range for the appropriate channel.
500	± 500 mV calibrated input range for the appropriate channel.
1000	± 1 V calibrated input range for the appropriate channel.
2000	± 2 V calibrated input range for the appropriate channel.
5000	± 5 V calibrated input range for the appropriate channel.
10000	± 10 V calibrated input range for the appropriate channel.

Standard Input ranges of path 1 (HF, 50 ohm terminated):

500	± 500 mV calibrated input range for the appropriate channel.
1000	± 1 V calibrated input range for the appropriate channel.
2500	± 2.5 V calibrated input range for the appropriate channel.
5000	± 5 V calibrated input range for the appropriate channel.

Read out of input features

Each input path (if multiple paths are available on the card) has different features that can be read out to make the software more general. If you only operate one single card type in your software it is not necessary to read out these features.

Please note that the input features are read out for the currently selected read AI path done by register SPC_READAIPATH. Please also note that the following table shows all input features settings that are available throughout all Spectrum acquisition cards. Some of these features are not installed on your specific hardware. The column(s) for the input paths show which settings are available for which input path (if multiple paths are available on the card) on a standard card:

Register	Value	Direction	Description
SPC_READAIPATH	3121	read/write	Selects the input path which is used to read out the features. Please note that this settings does not change the current path selection.
SPC_READAIFEATURES	3101	read	Returns a bit map with the available features of that input path. The possible return values are listed below.

	Value	Path 0	Path 1	Description
SPCM_AI_TERM	00000001h	x	fixed	Programmable input termination available
SPCM_AI_SE	00000002h	fixed	fixed	Input is single-ended. If available together with SPC_AI_DIFF: input type is software selectable
SPCM_AI_DIFF	00000004h			Input is differential. If available together with SPC_AI_SE: input type is software selectable
SPCM_AI_OFFSPERCENT	00000008h	x	x	Input offset programmable in per cent of input range
SPCM_AI_OFFSMV	00000010h			Input offset programmable in mV
SPCM_AI_OVERRANGEDETECT	00000020h			Programmable overrange detection available
SPCM_AI_DCCOUPLING	00000040h	x	x	Input is DC coupled. If available together with AC coupling: coupling is software selectable
SPCM_AI_ACCOUPLING	00000080h	x	x	Input is AC coupled. If available together with DC coupling: coupling is software selectable
SPCM_AI_LOWPASS	00000100h	x	x	Input has a selectable low pass filter (bandwidth limit)
SPCM_AI_ACDC_OFFS_COMP	00000200h			Input has a selectable offset compensation for HF-Path with AC/DC coupling/source missmatch.
SPCM_AI_AUTOCALOFFS	00001000h	x	x	Input offset can be auto calibrated on the card
SPCM_AI_AUTOCALGAIN	00002000h	x		Input gain can be auto calibrated on the card
SPCM_AI_AUTOCALOFFSNOIN	00004000h			Input offset can auto calibrated on the card if inputs are left open
SPCM_AI_INDIVPULSEWIDTH	00010000h			Trigger pulsewidth is individually per channel programmable

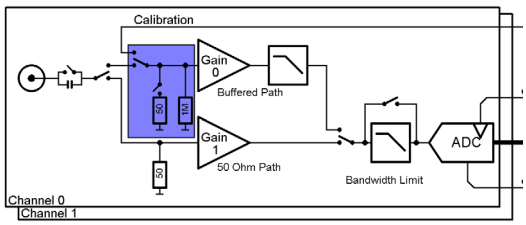
The following example shows a setup of path and input range of a two channel card.

Please note that this is a general example and the number of input channels may not match your card channels.

```
spcm_dwSetParam_i32 (hDrv, SPC_PATH0 ,      0); // Set up channel0 to input path 0 (buffered)
spcm_dwSetParam_i32 (hDrv, SPC_AMP0 ,    1000); // Set up channel0 to the range of ± 1.0 V
spcm_dwSetParam_i32 (hDrv, SPC_PATH1 ,      1); // Set up channel1 to input path 1 (HF, 50 ohm terminated)
spcm_dwSetParam_i32 (hDrv, SPC_AMP1 ,     500); // Set up channel1 to the range of ± 0.5 V
```

Input termination

The Spectrum analog acquisition cards of the M3i series offer an input path with fixed 50 ohm termination (HF path, 50 ohm path) as well as a second input path with all features to be programmed by the user (buffered path). If the HF path with fixed 50 ohm termination is activated this register will have no functionality.

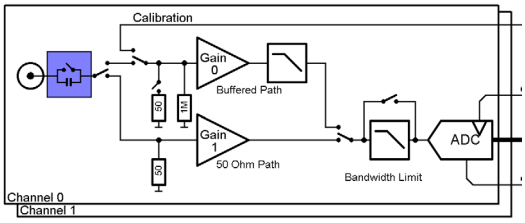


The buffered input path can be terminated separately with 50 Ohm by software programming. If you do so, please make sure that your signal source is able to deliver the higher output currents. If no termination is used, the inputs have an impedance of 1 Megaohm. The following table shows the corresponding register to set the input termination.

Register	Value	Direction	Description
SPC_50OHM0	30030	read/write	A „1“ sets the 50 ohm termination for channel0. A „0“ sets the termination to1 MOhm.
SPC_50OHM1	30130	read/write	A „1“ sets the 50 ohm termination for channel1. A „0“ sets the termination to1 MOhm.

Input coupling

All inputs can be set separately switched to AC or DC coupling. Please refer to the technical data section to see the signal frequency range that is available for the different settings.



Using the AC coupling will eliminate all DC and low frequency parts of the input signal and allows best quality measurings in the frequency domain even if the DC level of the signal varies over the time.

The following table shows the corresponding register to set the input coupling.

Register	Value	Direction	Description
SPC_ACDC0	30020	read/write	A „1“ sets the AC coupling for channel0. A „0“ sets the DC coupling (default is DC)
SPC_ACDC1	30120	read/write	A „1“ sets the AC coupling for channel1. A „0“ sets the DC coupling (default is DC)

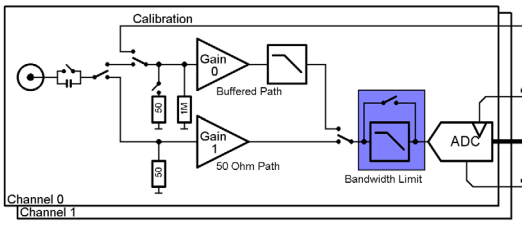
AC/DC offset compensation

When using the HF-Path of the input channel, an offset voltage will be visible in case DC coupling is selected for the channel and the signal source is externally AC coupled. This offset can be compensated for by setiing the compensation registers:

Register	Value	Direction	Description
SPC_ACDC_OFFS_COMPENSATION0	30021	read/write	A „1“ enables the compensation. A „0“ disables the compensation (default).
SPC_ACDC_OFFS_COMPENSATION1	30121	read/write	A „1“ enables the compensation. A „0“ disables the compensation (default).

Anti aliasing filter (Bandwidth limit)

All inputs have a separate selectable anti aliasing filter (bandwidth limit) that will cut of any aliasing effectes and that will reduce signal noise.



Please note that this bandwidth limit filter will also cut of any distortion or high frequency spurious signals parts that are within the frequency spectrum of the input.

Please refer to the technical data section to see the cut off frequency and the type of filter used. The following table shows the corresponding register to activate the bandwidth limit.

Register	Value	Direction	Description
SPC_FILTER0	30080	read/write	A „1" selects the bandwidth limit for channel 0. A „0" set the channel to full bandwidth (default is full)
SPC_FILTER1	30180	read/write	A „1" selects the bandwidth limit for channel 1. A „0" set the channel to full bandwidth (default is full

Enhanced Status Register

The enhanced status register shows detected channel overrange events during the last acquisition. It can only be read out after the acquisition has stopped. If the input signal on the channel exceeds the programmed input range even for just one time the overrange register is set in hardware.

Register	Value	Direction	Description
SPC_ENHANCEDSTATUS	20900	read	Reads out the enhanced status information of the card.
SPC_ENHSTAT_OVERRANGE0	00000001h	Bit is set if an overrange event has occurred on channel 0.	
SPC_ENHSTAT_OVERRANGE1	00000002h	Bit is set if an overrange event has occurred on channel 1.	

Automatic on-board calibration of the offset and gain settings

All of the channels are calibrated in factory before the board is shipped. These values are stored in the on-board EEProm under the default settings. If you have asymmetrical signals, you can adjust the offset easily with the corresponding registers of the inputs as shown before.

To start the automatic offset adjustment, simply write the register, mentioned in the following table.

Before you start an automatic offset adjustment make sure, that no signal is connected to any input. Leave all the input connectors open and then start the adjustment. All the internal settings of the driver are changed, while the automatic offset compensation is in progress.

Register	Value	Direction	Description
SPC_ADJ_AUTOADJ	50020	write	Performs the automatic offset compensation in the driver either for all input ranges or only the actual.
ADJ_ALL	0	Automatic offset adjustment for all input ranges.	

As all settings are temporarily stored in the driver, the automatic adjustment will only affect these values. After exiting your program, all calibration information will be lost. To give you a possibility to save your own settings, most Spectrum card have at least one set of user settings that can be saved within the on-board EEPROM. The default settings of the offset and gain values are then read-only and cannot be written to the EEPROM by the user. If the card has no user settings the default settings may be overwritten.

You can easily either save adjustment settings to the EEPROM with SPC_ADJ_SAVE or recall them with SPC_ADJ_LOAD. These two registers are shown in the table below. The values for these EEPROM access registers are the sets that can be stored within the EEPROM. The amount of sets available for storing user offset settings depends on the type of board you use. The table below shows all the EEPROM sets, that are available for your board.

Register	Value	Direction	Description
SPC_ADJ_LOAD	50000	write	Loads the specified set of settings from the EEPROM. The default settings are automatically loaded, when the driver is started.
		read	Reads out, what kind of settings have been loaded last.
SPC_ADJ_SAVE	50010	write	Stores the current settings to the specified set in the EEPROM.
		read	Reads out, what kind of settings have been saved last.
ADJ_DEFAULT	0	Default settings, no user settings available	

If you want to make an offset and gain adjustment on all the channels and store the data to the ADJ_DEFAULT set of the EEPROM you can do this the way, the following example shows.

```
spcm_dwSetParam_i32 (hDrv, SPC_ADJ_AUTOADJ,    ADJ_ALL  ); // Activate offset/gain adjustment on all channels
spcm_dwSetParam_i32 (hDrv, SPC_ADJ_SAVE   ,    ADJ_DEFAULT); // and store values to DEFAULT set in the EEPROM
```

Acquisition modes

Your card is able to run in different modes. Depending on the selected mode there are different registers that each define an aspect of this mode. The single modes are explained in this chapter. Any further modes that are only available if an option is installed on the card is documented in a later chapter.

Overview

This chapter gives you a general overview on the related registers for the different modes. The use of these registers throughout the different modes is described in the following chapters.

Setup of the mode

The mode register is organized as a bitmap. Each mode corresponds to one bit of this bitmap. When defining the mode to use, please be sure just to set one of the bits. All other settings will return an error code.

The main difference between all standard and all FIFO modes is that the standard modes are limited to on-board memory and therefore can run with full sampling rate. The FIFO modes are designed to transfer data continuously over the bus to PC memory or to hard disk and can therefore run much longer. The FIFO modes are limited by the maximum bus transfer speed the PC can use. The FIFO mode uses the complete installed on-board memory as a FIFO buffer.

However as you'll see throughout the detailed documentation of the modes the standard and the FIFO mode are similar in programming and behavior and there are only a very few differences between them.

Register	Value	Direction	Description
SPC_CARDMODE	9500	read/write	Defines the used operating mode, a read command will return the currently used mode.
SPC_AVAILCARDMODES	9501	read	Returns a bitmap with all available modes on your card. The modes are listed below.

Acquisition modes

Mode	Value	Available for series	Description
SPC_REC_STD_SINGLE	1h	M2i/M3i	Data acquisition to on-board memory for one single trigger event.
SPC_REC_STD_MULTI	2h	M2i/M3i	Data acquisition to on-board memory for multiple trigger events. Each recorded segment has the same size. This mode is described in greater detail in a special chapter about the Multiple Recording option.
SPC_REC_STD_ABA	8h	M2i/M3i	Data acquisition to on-board memory for multiple trigger events. While the multiple trigger events are stored with programmed sampling rate the inputs are sampled continuously with a slower sampling speed. The mode is described in a special chapter about ABA mode option.
SPC_REC_FIFO_SINGLE	10h	M2i/M3i	Continuous data acquisition for one single trigger event. The on-board memory is used completely as FIFO buffer.
SPC_REC_FIFO_MULTI	20h	M2i/M3i	Continuous data acquisition for multiple trigger events.
SPC_REC_FIFO_ABA	80h	M2i/M3i	Continuous data acquisition for multiple trigger events together with continuous data acquisition with a slower sampling clock.

Commands

The data acquisition/data replay is controlled by the command register. The command register controls the state of the card in general and also the state of the different data transfers. Data transfers are explained in an extra chapter later on.

The commands are split up into two types of commands: execution commands that fulfill a job and wait commands that will wait for the occurrence of an interrupt. Again the commands register is organized as a bitmap allowing you to set several commands together with one call. As not all of the command combinations make sense (like the combination of reset and start at the same time) the driver will check the given command and return an error code ERR_SEQUENCE if one of the given commands is not allowed in the current state.

Register	Value	Direction	Description
SPC_M2CMD	100	write only	Executes a command for the card or data transfer.

Card execution commands

M2CMD_CARD_RESET	1h	Performs a hard and software reset of the card as explained further above.
M2CMD_CARD_WRITESETUP	2h	Writes the current setup to the card without starting the hardware. This command may be useful if changing some internal settings like clock frequency and enabling outputs.
M2CMD_CARD_START	4h	Starts the card with all selected settings. This command automatically writes all settings to the card if any of the settings has been changed since the last one was written. After card has been started, only some of the settings might be changed while the card is running, such as e.g. output level and offset for D/A replay cards.
M2CMD_CARD_ENABLETRIGGER	8h	The trigger detection is enabled. This command can be either sent together with the start command to enable trigger immediately or in a second call after some external hardware has been started.
M2CMD_CARD_FORCETRIGGER	10h	This command forces a trigger even if none has been detected so far. Sending this command together with the start command is similar to using the software trigger.
M2CMD_CARD_DISABLETRIGGER	20h	The trigger detection is disabled. All further trigger events are ignored until the trigger detection is again enabled. When starting the card the trigger detection is started disabled.
M2CMD_CARD_STOP	40h	Stops the current run of the card. If the card is not running this command has no effect.

Card wait commands

These commands do not return until either the defined state has been reached which is signaled by an interrupt from the card or the timeout counter has expired. If the state has been reached the command returns with an ERR_OK. If a timeout occurs the command returns with ERR_TIMEOUT. If the card has been stopped from a second thread with a stop or reset command, the wait function returns with ERR_ABORT.

M2CMD_CARD_WAITPREFULL	1000h	Acquisition modes only: the command waits until the pretrigger area has once been filled with data. After pretrigger area has been filled the internal trigger engine starts to look for trigger events if the trigger detection has been enabled.
M2CMD_CARD_WAITTRIGGER	2000h	Waits until the first trigger event has been detected by the card. If using a mode with multiple trigger events like Multiple Recording or Gated Sampling there only the first trigger detection will generate an interrupt for this wait command.
M2CMD_CARD_WAITREADY	4000h	Waits until the card has completed the current run. In an acquisition mode receiving this command means that all data has been acquired. In a generation mode receiving this command means that the output has stopped.

Wait command timeout

If the state for which one of the wait commands is waiting isn't reached any of the wait commands will either wait forever if no timeout is defined or it will return automatically with an ERR_TIMEOUT if the specified timeout has expired.

Register	Value	Direction	Description
SPC_TIMEOUT	295130	read/write	Defines the timeout for any following wait command in a millisecond resolution. Writing a zero to this register disables the timeout.

As a default the timeout is disabled. After defining a timeout this is valid for all following wait commands until the timeout is disabled again by writing a zero to this register.

A timeout occurring should not be considered as an error. It did not change anything on the board status. The board is still running and will complete normally. You may use the timeout to abort the run after a certain time if no trigger has occurred. In that case a stop command is necessary after receiving the timeout. It is also possible to use the timeout to update the user interface frequently and simply call the wait function afterwards again.

Example for card control:

```
// card is started and trigger detection is enabled immediately
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER);

// we wait a maximum of 1 second for a trigger detection. In case of timeout we force the trigger
spcm_dwSetParam_i32 (hDrv, SPC_TIMEOUT, 1000);
if (spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_WAITTRIGGER) == ERR_TIMEOUT)
    {
    printf ("No trigger detected so far, we force a trigger now!\n");
    spcm_dwSetParam (hdrv, SPC_M2CMD, M2CMD_CARD_FORCETRIGGER);
    }

// we disable the timeout and wait for the end of the run
spcm_dwSetParam_i32 (hDrv, SPC_TIMEOUT, 0);
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_WAITREADY);
printf ("Card has stopped now!\n");
```

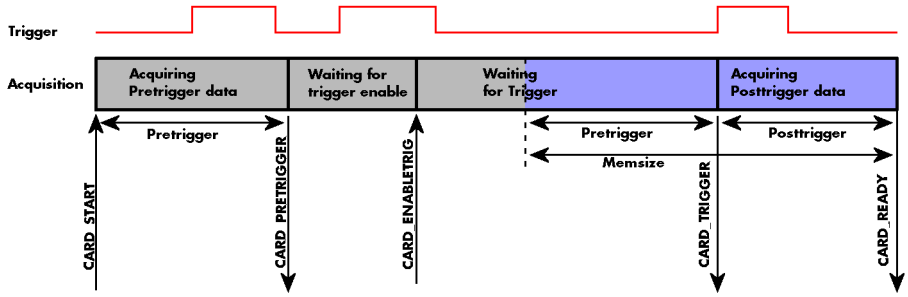
Card Status

In addition to the wait for an interrupt mechanism or completely instead of it one may also read out the current card status by reading the SPC_M2STATUS register. The status register is organized as a bitmap, so that multiple bits can be set, showing the status of the card and also of the different data transfers.

Register	Value	Direction	Description
SPC_M2STATUS	110	read only	Reads out the current status information
M2STAT_CARD_PRETRIGGER	1h	Acquisition modes only: the pretrigger area has been filled.	
M2STAT_CARD_TRIGGER	2h	The first trigger has been detected.	
M2STAT_CARD_READY	4h	The card has finished its run and is ready.	
M2STAT_CARD_SEGMENT_PRETRG	8h	Multi/ABA/Gated acquisition of M4i/M4x/M2p only: the pretrigger area of one segment has been filled.	

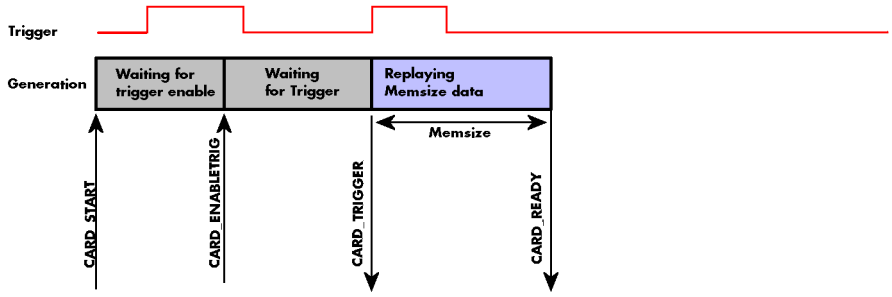
Acquisition cards status overview

The following drawing gives you an overview of the card commands and card status information. After start of card with M2CMD_CARD_START the card is acquiring pretrigger data until one time complete pretrigger data has been acquired. Then the status bit M2STAT_CARD_PRETRIGGER is set. Either the trigger has been enabled together with the start command or the card now waits for trigger enable command M2CMD_CARD_ENABLETRIGGER. After receiving this command the trigger engine is enabled and card checks for a trigger event. As soon as the trigger event is received the status bit M2STAT_CARD_TRIGGER is set and the card acquires the programmed posttrigger data. After all post trigger data has been acquired the status bit M2STAT_CARD_READY is set and data can be read out:



Generation card status overview

This drawing gives an overview of the card commands and status information for a simple generation mode. After start of card with the M2CMD_CARD_START the card is armed and waiting. Either the trigger has been enabled together with the start command or the card now waits for trigger enable command M2CMD_CARD_ENABLETRIGGER. After receiving this command the trigger engine is enabled and card checks for a trigger event. As soon as the trigger event is received the status bit M2STAT_CARD_TRIGGER is set and the card starts with the data replay. After replay has been finished - depending on the programmed mode - the status bit M2STAT_CARD_READY is set and the card stops.



Data Transfer

Data transfer consists of two parts: the buffer definition and the commands/status information that controls the transfer itself. Data transfer shares the command and status register with the card control commands and status information. In general the following details on the data transfer are valid for any data transfer in any direction:

- The memory size register (SPC_MEMSIZE) must be programmed before starting the data transfer.
- When the hardware buffer is adjusted from its default (see „Output latency" section later in this manual), this must be done before defining the transfer buffers in the next step via the spcm_dwDefTransfer function.
- Before starting a data transfer the buffer must be defined using the spcm_dwDefTransfer function.
- Each defined buffer is only used once. After transfer has ended the buffer is automatically invalidated.
- If a buffer has to be deleted although the data transfer is in progress or the buffer has at least been defined it is necessary to call the spcm_dwInvalidateBuf function.

Definition of the transfer buffer

Before any data transfer can start it is necessary to define the transfer buffer with all its details. The definition of the buffer is done with the spcm_dwDefTransfer function as explained in an earlier chapter.

```
uint32 _stdcall spcm_dwDefTransfer_i64 (// Defines the transfer buffer by using 64 bit unsigned integer values
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType,              // type of the buffer to define as listed below under SPCM_BUF_XXXX
    uint32      dwDirection,            // the transfer direction as defined below
    uint32      dwNotifySize,           // number of bytes after which an event is sent (0=end of transfer)
    void*       pvDataBuffer,           // pointer to the data buffer
    uint64      qwBrdOffs,              // offset for transfer in board memory
    uint64      qwTransferLen);         // buffer length
```

This function is used to define buffers for standard sample data transfer as well as for extra data transfer for additional ABA or timestamp information. Therefore the dwBufType parameter can be one of the following:

SPCM_BUF_DATA	1000	Buffer is used for transfer of standard sample data
SPCM_BUF_ABA	2000	Buffer is used to read out slow ABA data. Details on this mode are described in the chapter about the ABA mode option
SPCM_BUF_TIMESTAMP	3000	Buffer is used to read out timestamp information. Details on this mode are described in the chapter about the timestamp option.

The dwDirection parameter defines the direction of the following data transfer:

SPCM_DIR_PCTOCARD	0	Transfer is done from PC memory to on-board memory of card
SPCM_DIR_CARDTOPC	1	Transfer is done from card on-board memory to PC memory.
SPCM_DIR_CARDTOGPU	2	RDMA transfer from card memory to GPU memory, SCAPP option needed, Linux only
SPCM_DIR_GPUTOCARD	3	RDMA transfer from GPU memory to card memory, SCAPP option needed, Linux only

The direction information used here must match the currently used mode. While an acquisition mode is used there's no transfer from PC to card allowed and vice versa. It is possible to use a special memory test mode to come beyond this limit. Set the SPC_MEMTEST register as defined further below.

The dwNotifySize parameter defines the amount of bytes after which an interrupt should be generated. If leaving this parameter zero, the transfer will run until all data is transferred and then generate an interrupt. Filling in notify size > zero will allow you to use the amount of data that has been transferred so far. The notify size is used on FIFO mode to implement a buffer handshake with the driver or when transferring large amount of data where it may be of interest to start data processing while data transfer is still running. Please see the chapter on handling positions further below for details.

The Notify size sticks to the page size which is defined by the PC hardware and the operating system. Therefore the notify size must be a multiple of 4 kByte. For data transfer it may also be a fraction of 4k in the range of 16, 32, 64, 128, 256, 512, 1k or 2k. No other values are allowed. For ABA and timestamp the notify size can be 2k as a minimum. If you need to work with ABA or timestamp data in smaller chunks please use the polling mode as described later.

The pvDataBuffer must point to an allocated data buffer for the transfer. Please be sure to have at least the amount of memory allocated that you program to be transferred. If the transfer is going from card to PC this data is overwritten with the current content of the card on-board memory.

The pvDataBuffer needs to be aligned to a page size (4096 bytes). Please use appropriate software commands when allocating the data buffer. Using a non-aligned buffer may result in data corruption.

When not doing FIFO mode one can also use the qwBrdOffs parameter. This parameter defines the starting position for the data transfer as byte value in relation to the beginning of the card memory. Using this parameter allows it to split up data transfer in smaller chunks if one has acquired a very large on-board memory.

The qwTransferLen parameter defines the number of bytes that has to be transferred with this buffer. Please be sure that the allocated memory has at least the size that is defined in this parameter. In standard mode this parameter cannot be larger than the amount of data defined with memory size.

Memory test mode

In some cases it might be of interest to transfer data in the opposite direction. Therefore a special memory test mode is available which allows random read and write access of the complete on-board memory. While memory test mode is activated no normal card commands are processed:

Register	Value	Direction	Description
SPC_MEMTEST	200700	read/write	Writing a 1 activates the memory test mode, no commands are then processed. Writing a 0 deactivates the memory test mode again.

Invalidation of the transfer buffer

The command can be used to invalidate an already defined buffer if the buffer is about to be deleted by user. This function is automatically called if a new buffer is defined or if the transfer of a buffer has completed

```
uint32 _stdcall spcm_dwInvalidateBuf (  // invalidate the transfer buffer
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType);            // type of the buffer to invalidate as listed above under SPCM_BUF_XXXX
```

The dwBufType parameter need to be the same parameter for which the buffer has been defined:

SPCM_BUF_DATA	1000	Buffer is used for transfer of standard sample data
SPCM_BUF_ABA	2000	Buffer is used to read out slow ABA data. Details on this mode are described in the chapter about the ABA mode option. The ABA mode is only available on analog acquisition cards.
SPCM_BUF_TIMESTAMP	3000	Buffer is used to read out timestamp information. Details on this mode are described in the chapter about the timestamp option. The timestamp mode is only available on analog or digital acquisition cards.

Commands and Status information for data transfer buffers.

As explained above the data transfer is performed with the same command and status registers like the card control. It is possible to send commands for card control and data transfer at the same time as shown in the examples further below.

Register	Value	Direction	Description
SPC_M2CMD	100	write only	Executes a command for the card or data transfer
M2CMD_DATA_STARTDMA	10000h		Starts the DMA transfer for an already defined buffer. In acquisition mode it may be that the card hasn't received a trigger yet, in that case the transfer start is delayed until the card receives the trigger event
M2CMD_DATA_WAITDMA	20000h		Waits until the data transfer has ended or until at least the amount of bytes defined by notify size are available. This wait function also takes the timeout parameter described above into account.
M2CMD_DATA_STOPDMA	40000h		Stops a running DMA transfer. Data is invalid afterwards.

The data transfer can generate one of the following status information:

Register	Value	Direction	Description
SPC_M2STATUS	110	read only	Reads out the current status information
M2STAT_DATA_BLOCKREADY	100h		The next data block as defined in the notify size is available. It is at least the amount of data available but it also can be more data.
M2STAT_DATA_END	200h		The data transfer has completed. This status information will only occur if the notify size is set to zero.
M2STAT_DATA_OVERRUN	400h		The data transfer had on overrun (acquisition) or underrun (replay) while doing FIFO transfer.
M2STAT_DATA_ERROR	800h		An internal error occurred while doing data transfer.

Example of data transfer

```
void* pvData = pvAllocMemPageAligned (1024);

// transfer data from PC memory to card memory (on replay cards) ...
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA,  SPCM_DIR_PCTOCARD , 0, pvData, 0, 1024);
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);

// ... or transfer data from card memory to PC memory (acquisition cards)
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA,  SPCM_DIR_CARDTOPC , 0, pvData, 0, 1024);
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);

// explicitely stop DMA tranfer prior to invalidating buffer
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STOPDMA);
spcm_dwInvalidateBuf (hDrv, SPCM_BUF_DATA);
vFreeMemPageAligned (pvData, 1024);
```

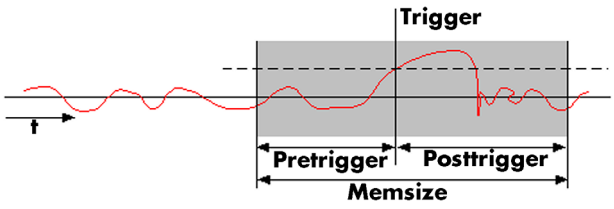
To keep the example simple it does no error checking. Please be sure to check for errors if using these command in real world programs!

Users should take care to explicitly send the M2CMD_DATA_STOPDMA command prior to invalidating the buffer, to avoid crashes due to race conditions when using higher-latency data transportation layers, such as to remote Ethernet devices.

Standard Single acquisition mode

The standard single mode is the easiest and mostly used mode to acquire analog data with a Spectrum acquisition card. In standard single recording mode the card is working totally independent from the PC, after the card setup is done. The advantage of the Spectrum boards is that regardless to the system usage the card will sample with equidistant time intervals.
The sampled and converted data is stored in the on-board memory and is held there for being read out after the acquisition. This mode allows sampling at very high conversion rates without the need to transfer the data into the memory of the host system at high speed.
After the recording is done, the data can be read out by the user and is transferred via the bus into PC memory.

This standard recording mode is the most common mode for all analog and digital acquisition and oscilloscope boards. The data is written to a programmed amount of the on-board memory (memsize). That part of memory is used as a ring buffer, and recording is done continuously until a trigger event is detected. After the trigger event, a certain programmable amount of data is recorded (post trigger) and then the recording finishes. Due to the continuous ring buffer recording, there are also samples prior to the trigger event in the memory (pretrigger).



When the card is started the pre trigger area is filled up with data first. While doing this the board's trigger detection is not armed. If you use a huge pre trigger size and a slow sample rate it can take some time after starting the board before a trigger event will be detected.

Card mode

The card mode has to be set to the correct mode SPC_REC_STD_SINGLE.

Register	Value	Direction	Description
SPC_CARDMODE	9500	read/write	Defines the used operating mode, a read command will return the currently used mode.
SPC_REC_STD_SINGLE	1h	Data acquisition to on-board memory for one single trigger event.	

Memory, Pre- and Posttrigger

At first you have to define, how many samples are to be recorded at all and how many of them should be acquired after the trigger event has been detected.

Register	Value	Direction	Description
SPC_MEMSIZE	10000	read/write	Sets the memory size in samples per channel.
SPC_POSTTRIGGER	10100	read/write	Sets the number of samples to be recorded per channel after the trigger event has been detected.

You can access these settings by the register SPC_MEMSIZE, which sets the total amount of data that is recorded, and the register SPC_POSTTRIGGER, that defines the number of samples to be recorded after the trigger event has been detected. The size of the pretrigger results on the simple formula:

pretrigger = memsize - posttrigger

The maximum memsize that can be use for recording is of course limited by the installed amount of memory and by the number of channels to be recorded. Please have a look at the topic "Limits of pre, post memsize, loops" later in this chapter.

Example

The following example shows a simple standard single mode data acquisition setup with the read out of data afterwards. To keep this example simple there is no error checking implemented.

```
int32 lMemsize = 16384;                                          // recording length is set to 16 kSamples

spcm_dwSetParam_i32 (hDrv, SPC_CHENABLE, CHANNEL0);                // only one channel activated
spcm_dwSetParam_i32 (hDrv, SPC_CARDMODE, SPC_REC_STD_SINGLE);     // set the standard single recording mode
spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE, lMemsize);                 // recording length
spcm_dwSetParam_i64 (hDrv, SPC_POSTTRIGGER, 8192);                 // samples to acquire after trigger = 8k

// now we start the acquisition and wait for the interrupt that signalizes the end
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER | M2CMD_CARD_WAITREADY);

void* pvData = pvAllocMemPageAligned (2 * lMemsize); // assuming 2 bytes per sample

// read out the data
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA,  SPCM_DIR_CARDTOPC , 0, pvData, 0, 2 * lMemsize);
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);
```

FIFO Single acquisition mode

The FIFO single mode does a continuous data acquisition using the on-board memory as a FIFO buffer and transferring data continuously to PC memory. One can make on-line calculations with the acquired data, store the data continuously to disk for later use or even have a data logger functionality with on-line data display.

Card mode

The card mode has to be set to the correct mode SPC_REC_FIFO_SINGLE.

Register	Value	Direction	Description
SPC_CARDMODE	9500	read/write	Defines the used operating mode, a read command will return the currently used mode.
SPC_REC_FIFO_SINGLE	10h	Continuous data acquisition to PC memory. Complete on-board memory is used as FIFO buffer.	

Length and Pretrigger

Even in FIFO mode it is possible to program a pretrigger area. In general FIFO mode can run forever until it is stopped by an explicit user command or one can program the total length of the transfer by two counters Loop and Segment size

Register	Value	Direction	Description
SPC_PRETRIGGER	10030	read/write	Programs the number of samples to be acquired before the trigger event detection
SPC_SEGMENTSIZE	10010	read/write	Length of segments to acquire.
SPC_LOOPS	10020	read/write	Number of segments to acquire in total. If set to zero the FIFO mode will run continuously until it is stopped by the user.

The total amount of samples per channel that is acquired can be calculated by [SPC_LOOPS * SPC_SEGMENTSIZE]. Please stick to the below mentioned limitations of the registers.

Difference to standard single acquisition mode

The standard modes and the FIFO modes differ not very much from the programming side. In fact one can even use the FIFO mode to get the same behavior like the standard mode. The buffer handling that is shown in the next chapter is the same for both modes.

Pretrigger

When doing standard single acquisition memory is used as a circular buffer and the pre trigger can be up to the [installed memory] - [minimum post trigger]. Compared to this the pre trigger in FIFO mode is limited by a special pre trigger FIFO and hence considerably shorter.

Length of acquisition.

In standard mode the acquisition length is defined before the start and is limited to the installed on-board memory whilst in FIFO mode the acquisition length can either be defined or it can run continuously until user stops it.

Example FIFO acquisition

The following example shows a simple FIFO single mode data acquisition setup with the read out of data afterwards. To keep this example simple there is no error checking implemented.

```
spcm_dwSetParam_i32 (hDrv, SPC_CHENABLE, CHANNEL0);                    // only one channel activated
spcm_dwSetParam_i32 (hDrv, SPC_CARDMODE, SPC_REC_FIFO_SINGLE);         // set the FIFO single recording mode
spcm_dwSetParam_i64 (hDrv, SPC_PRETRIGGER, 1024);                      // 1 kSample of data before trigger

// in FIFO mode we need to define the buffer before starting the transfer
void* pvData = pvAllocMemPageAligned (llBufsizeInSamples * 2);         // 2 bytes per sample
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA, SPCM_DIR_CARDTOPC, 4096,
                            pvData, 0, 2 * llBufsizeInSamples);

// now we start the acquisition and wait for the first block
dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER);
dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);

// we acquire data in a loop. As we defined a notify size of 4k we'll get the data in >=4k chuncks
llTotalBytes = 0;
while (!dwError)
    {
    spcm_dwGetParam_i64 (hDrv, SPC_DATA_AVAIL_USER_LEN, &llAvailBytes); // read out the available bytes
    llTotalBytes += llAvailBytes;

    // here is the right position to do something with the data (printf is limited to 32 bit variables)
    printf ("Currently Available: %lld, total: %lld\n", llAvailBytes, llTotalBytes);

    // now we free the number of bytes and wait for the next buffer
    spcm_dwSetParam_i64 (hDrv, SPC_DATA_AVAIL_CARD_LEN, llAvailBytes);
    dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_WAITDMA);
    }
```

Limits of pre trigger, post trigger, memory size

The maximum memory size parameter is only limited by the number of activated channels and by the amount of installed memory. Please keep in mind that each sample needs 2 bytes of memory to be stored. Minimum memory size as well as minimum and maximum post trigger limits are independent of the activated channels or the installed memory.

Due to the internal organization of the card memory there is a certain stepsize when setting these values that has to be taken into account. The following table gives you an overview of all limits concerning pre trigger, post trigger, memory size, segment size and loops. The table shows all values in relation to the installed memory size in samples. If more memory is installed the maximum memory size figures will increase according to the complete installed memory

Activated Channels	Used Mode	Memory size SPC_MEMSIZE			Pre trigger SPC_PRETRIGGER			Post trigger SPC_POSTTRIGGER			Segment size SPC_SEGMENTSIZE			Loops SPC_LOOPS		
		Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step
1 channel	Standard Single	16	Mem	8	defined by post trigger			8	8G - 8	8	not used			not used		
	Standard Multi/ABA	16	Mem	8	8	8k	8	8	Mem/2	8	16	Mem/2	8	not used		
	FIFO Single	not used			8	8k	8	not used			16	8G - 8	8	0 (∞)	4G - 1	1
	FIFO Multi/ABA	not used			8	8k	8	8	8G - 8	8	16	pre+post	8	0 (∞)	4G - 1	1
2 channels	Standard Single	16	Mem/2	8	defined by post trigger			8	8G - 8	8	not used			not used		
	Standard Multi/ABA	16	Mem/2	8	8	4k	8	8	Mem/4	8	16	Mem/4	8	not used		
	FIFO Single	not used			8	4k	8	not used			16	8G - 8	8	0 (∞)	4G - 1	1
	FIFO Multi/ABA	not used			8	4k	8	8	8G - 8	8	16	pre+post	8	0 (∞)	4G - 1	1

All figures listed here are given in samples. An entry of [32G - 8] means [32 GSamples - 8] = 34,359,738,360 samples.

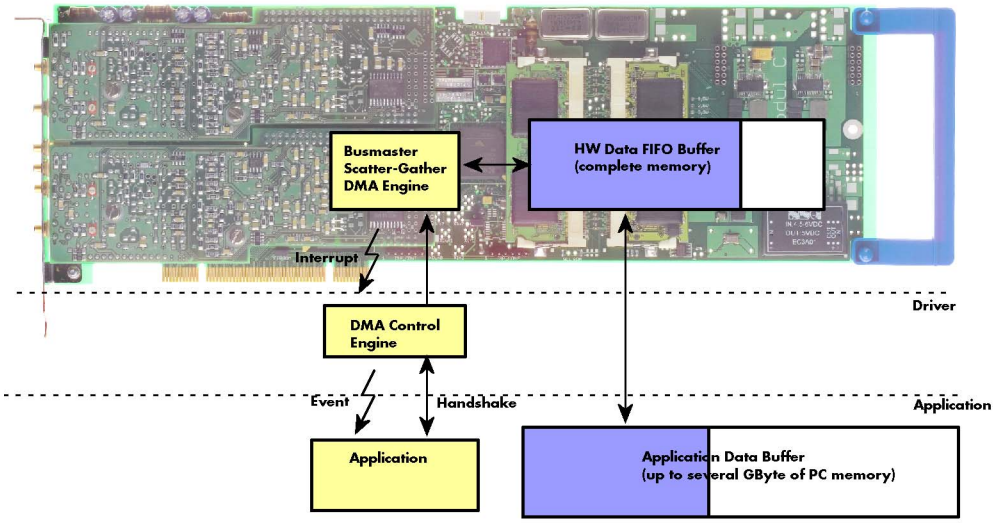
The given memory and memory / divider figures depend on the installed on-board memory as listed below:

	Installed Memory				
	128 MSample	**256 MSample**	**512 MSample**	**1 GSample**	**2 GSample**
Mem	128 MSample	256 MSample	512 MSample	1 GSample	2 GSample
Mem / 2	64 MSample	128 MSample	256 MSample	512 MSample	1 GSample
Mem / 4	32 MSample	64 MSample	128 MSample	256 MSample	512 MSample

Please keep in mind that this table shows all values at once. Only the absolute maximum and minimum values are shown. There might be additional limitations. Which of these values is programmed depends on the used mode. Please read the detailed documentation of the mode.

Buffer handling

To handle the huge amount of data that can possibly be acquired with the M2i/M3i series cards, there is a very reliable two step buffer strategy set up. The on-board memory of the card can be completely used as a real FIFO buffer. In addition a part of the PC memory can be used as an additional software buffer. Transfer between hardware FIFO and software buffer is performed interrupt driven and automatically by the driver to get best performance. The following drawing will give you an overview of the structure of the data transfer handling:



A data buffer handshake is implemented in the driver which allows to run the card in different data transfer modes. The software transfer buffer is handled as one large buffer which is on the one side controlled by the driver and filled automatically by busmaster DMA from/to the hardware FIFO buffer and on the other hand it is handled by the user who set's parts of this software buffer available for the driver for further transfer. The handshake is fulfilled with the following 3 software registers:

Register	Value	Direction	Description
SPC_DATA_AVAIL_USER_LEN	200	read	Returns the number of currently to the user available bytes inside a sample data transfer.
SPC_DATA_AVAIL_USER_POS	201	read	Returns the position as byte index where the currently available data samples start.
SPC_DATA_AVAIL_CARD_LEN	202	write	Writes the number of bytes that the card can now use for sample data transfer again

Internally the card handles two counters, a user counter and a card counter. Depending on the transfer direction the software registers have slightly different meanings:

Transfer direction	Register	Direction	Description
Write to card	SPC_DATA_AVAIL_USER_LEN	read	This register contains the currently available number of bytes that are free to write new data to the card. The user can now fill this amount of bytes with new data to be transferred.
	SPC_DATA_AVAIL_CARD_LEN	write	After filling an amount of the buffer with new data to transfer to card, the user tells the driver with this register that the amount of data is now ready to transfer.
Read from card	SPC_DATA_AVAIL_USER_LEN	read	This register contains the currently available number of bytes that are filled with newly transferred data. The user can now use this data for own purposes, copy it, write it to disk or start calculations with this data.
	SPC_DATA_AVAIL_CARD_LEN	write	After finishing the job with the new available data the user needs to tell the driver that this amount of bytes is again free for new data to be transferred.
Any direction	SPC_DATA_AVAIL_USER_POS	read	The register holds the current byte index position where the available bytes start. The register is just intended to help you and to avoid own position calculation
Any direction	SPC_FILLSIZEPROMILLE	read	The register holds the current fill size of the on-board memory (FIFO buffer) in promille (1/1000) of the full on-board memory. Please note that the hardware reports the fill size only in 1/16 parts of the full memory. The reported fill size is therefore only shown in 1000/16 = 63 promille steps.

Directly after start of transfer the SPC_DATA_AVAIL_USER_LEN is every time zero as no data is available for the user and the SPC_DATA_AVAIL_CARD_LEN is every time identical to the length of the defined buffer as the complete buffer is available for the card for transfer.

The counter that is holding the user buffer available bytes (SPC_DATA_AVAIL_USER_LEN) is sticking to the defined notify size at the DefTransfer call. Even when less bytes already have been transferred you won't get notice of it if the notify size is programmed to a higher value.

Remarks

- The transfer between hardware FIFO buffer and application buffer is done with scatter-gather DMA using a busmaster DMA controller located on the card. Even if the PC is busy with other jobs data is still transferred until the application data buffer is completely used.
- Even if application data buffer is completely used there's still the hardware FIFO buffer that can hold data until the complete on-board memory is used. Therefore a larger on-board memory will make the transfer more reliable against any PC dead times.
- As you see in the above picture data is directly transferred between application data buffer and on-board memory. Therefore it is absolutely critical to delete the application data buffer without stopping any DMA transfers that are running actually. It is also absolutely critical to define the application data buffer with an unmatching length as DMA can than try to access memory outside the application data

area.

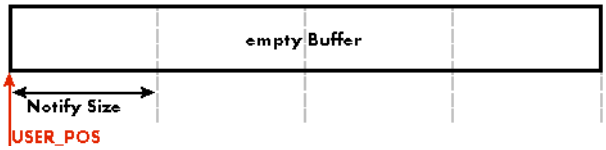
- As shown in the drawing above the DMA control will announce new data to the application by sending an event. Waiting for an event is done internally inside the driver if the application calls one of the wait functions. Waiting for an event does not consume any CPU time and is therefore highly desirable if other threads do a lot of calculation work. However it is not necessary to use the wait functions and one can simply request the current status whenever the program has time to do so. When using this polling mode the announced available bytes still stick to the defined notify size!
- If the on-board FIFO buffer has an overrun (card to PC) or an underrun (PC to card) data transfer is stopped. However in case of transfer from card to PC there is still a lot of data in the on-board memory. Therefore the data transfer will continue until all data has been transferred although the status information already shows an overrun.
- Getting best bus transfer performance is done using a „continuous buffer". This mode is explained in the appendix in greater detail.

The Notify size sticks to the page size which is defined by the PC hardware and the operating system. Therefore the notify size must be a multiple of 4 kByte. For data transfer it may also be a fraction of 4k in the range of 16, 32, 64, 128, 256, 512, 1k or 2k. No other values are allowed. For ABA and timestamp the notify size can be 2k as a minimum. If you need to work with ABA or timestamp data in smaller chunks please use the polling mode as described later.

The following graphs will show the current buffer positions in different states of the transfer. The drawings have been made for the transfer from card to PC. However all the block handling is similar for the opposite direction, just the empty and the filled parts of the buffer are inverted.

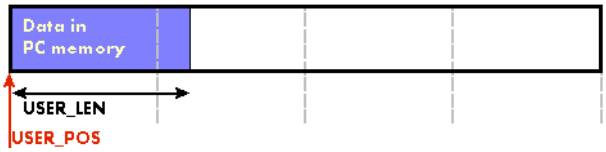
Step 1: Buffer definition

Directly after buffer definition the complete buffer is empty (card to PC) or completely filled (PC to card). In our example we have a notify size which is 1/4 of complete buffer memory to keep the example simple. In real world use it is recommended to set the notify size to a smaller stepsize.



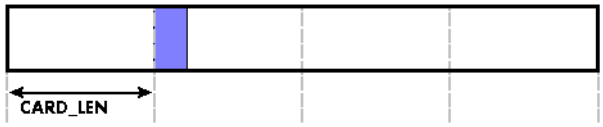
Step 2: Start and first data available

In between we have started the transfer and have waited for the first data to be available for the user. When there is at least one block of notify size in the memory we get an interrupt and can proceed with the data. Any data that already was transferred is announced. The USER_POS is still zero as we are right at the beginning of the complete transfer.



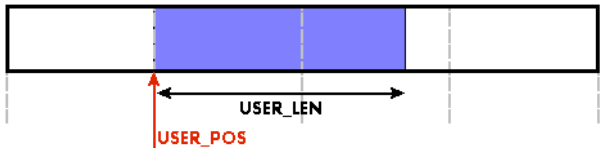
Step 3: set the first data available for card

Now the data can be processed. If transfer is going from card to PC that may be storing to hard disk or calculation of any figures. If transfer is going from PC to card that means we have to fill the available buffer again with data. After the amount of data that has been processed by the user application we set it available for the card and for the next step.



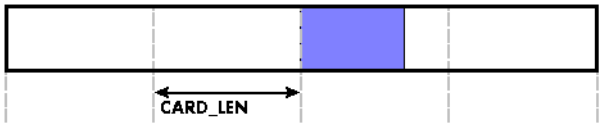
Step 4: next data available

After reaching the next border of the notify size we get the next part of the data buffer to be available. In our example at the time when reading the USER_LEN even some more data is already available. The user position will now be at the position of the previous set CARD_LEN.



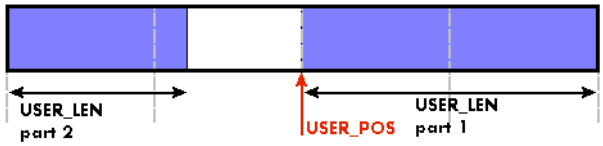
Step 5: set data available again

Again after processing the data we set it free for the card use.
In our example we now make something else and don't react to the interrupt for a longer time. In the background the buffer is filled with more data.



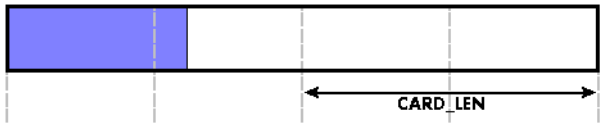
Step 6: roll over the end of buffer

Now nearly the complete buffer is filled. Please keep in mind that our current user position is still at the end of the data part that we processed and marked in step 4 and step 5. Therefore the data to process now is split in two parts. Part 1 is at the end of the buffer while part 2 is starting with address 0.



Step 7: set the rest of the buffer available

Feel free to process the complete data or just the part 1 until the end of the buffer as we do in this example. If you decide to process complete buffer please keep in mind the roll over at the end of the buffer.



This buffer handling can now continue endless as long as we manage to set the data available for the card fast enough. The USER_POS and USER_LEN for step 8 would now look exactly as the buffer shown in step 2.

Buffer handling example for transfer from card to PC (Data acquisition)

```
int8* pcData = (int8*) pvAllocMemPageAligned (llBufferSizeInBytes);

// we now define the transfer buffer with the minimum notify size of one page = 4 kByte
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA,  SPCM_DIR_CARDTOPC , 4096, (void*) pcData, 0, llBufferSizeInBytes);

// we start the DMA transfer
dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA);

do
    {
    if (!dwError)
        {
        // we wait for the next data to be available. Afte this call we get at least 4k of data to proceed
        dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_WAITDMA);

        // if there was no error we can proceed and read out the available bytes that are free again
        spcm_dwGetParam_i64 (hDrv, SPC_DATA_AVAIL_USER_LEN, &llAvailBytes);
        spcm_dwGetParam_i64 (hDrv, SPC_DATA_AVAIL_USER_POS, &llBytePos);

        printf ("We now have %lld new bytes available\n", llAvailBytes);
        printf ("The available data starts at position %lld\n", llBytesPos);

        // we take care not to go across the end of the buffer, handling the wrap-around
        if ((llBytePos + llAvailBytes) >= llBufferSizeInBytes)
            llAvailBytes = llBufferSizeInBytes - llBytePos;

        // our do function gets a pointer to the start of the available data section and the length
        vDoSomething (&pcData[llBytesPos], llAvailBytes);

        // the buffer section is now immediately set available for the card
        spcm_dwSetParam_i64 (hDrv, SPC_DATA_AVAIL_CARD_LEN, llAvailBytes);
        }
    }
while (!dwError); // we loop forever if no error occurs
```

Buffer handling example for transfer from PC to card (Data generation)

```
int8* pcData = (int8*) pvAllocMemPageAligned (llBufferSizeInBytes);

// before starting transfer we first need to fill complete buffer memory with meaningful data
vDoGenerateData (&pcData[0], llBufferSizeInBytes);

// we now define the transfer buffer with the minimum notify size of one page = 4 kByte
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA,  SPCM_DIR_PCTOCARD , 4096, (void*) pcData, 0, llBufferSizeInBytes);

// and transfer some data to the hardware buffer before the start of the card
spcm_dwSetParam_i32 (hDrv, SPC_DATA_AVAIL_CARD_LEN, llBufferSizeInBytes);
dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);

do
    {
    if (!dwError)
        {
        // if there was no error we can proceed and read out the current amount of available data
        spcm_dwGetParam_i64 (hDrv, SPC_DATA_AVAIL_USER_LEN, &llAvailBytes);
        spcm_dwGetParam_i64 (hDrv, SPC_DATA_AVAIL_USER_POS, &llBytePos);

        printf ("We now have %lld free bytes available\n", llAvailBytes);
        printf ("The available data starts at position %lld\n", llBytesPos);

        // we take care not to go across the end of the buffer, handling the wrap-around
        if ((llBytePos + llAvailBytes) >= llBufferSizeInBytes)
            llAvailBytes = llBufferSizeInBytes - llBytePos;

        // our do function gets a pointer to the start of the available data section and the length
        vDoGenerateData (&pcData[llBytesPos], llAvailBytes);

        // now we mark the number of bytes that we just generated for replay
        // and wait for the next free buffer
        spcm_dwSetParam_i64 (hDrv, SPC_DATA_AVAIL_CARD_LEN, llAvailBytes);
        dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_WAITDMA);
        }
    }
while (!dwError); // we loop forever if no error occurs
```

Please keep in mind that you are using a continuous buffer writing/reading that will start again at the zero position if the buffer length is reached. However the DATA_AVAIL_USER_LEN register will give you the complete amount of available bytes even if one part of the free area is at the end of the buffer and the second half at the beginning of the buffer.

Data organisation

Data is organized in a multiplexed way in the transfer buffer. If using 2 channels data of first activated channel comes first, then data of second channel.

Activated Channels	Ch0	Ch1			Samples ordering in buffer memory starting with data offset zero															
1 channel	X				A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
1 channel		X			B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
2 channels	X	X			A0	B0	A1	B1	A2	B2	A3	B3	A4	B4	A5	B5	A6	B6	A7	B7	A8

The samples are re-named for better readability. A0 is sample 0 of channel 0, B4 is sample 4 of channel 1, and so on

Sample format

The 16 bit A/D samples are stored in twos complement in the 16 bit data word. 16bit resolution means that data is ranging from -32768…to…-32767.

Bit	Standard Mode
D15	ADx Bit 15 (MSB)
D14	ADx Bit 14
D13	ADx Bit 13
D12	ADx Bit 12
D11	ADx Bit 11
D10	ADx Bit 10
D9	ADx Bit 9
D8	ADx Bit 8
D7	ADx Bit 7
D6	ADx Bit 6
D5	ADx Bit 5
D4	ADx Bit 4
D3	ADx Bit 3
D2	ADx Bit 2
D1	ADx Bit 1
D0	ADx Bit 0 (LSB)

Converting ADC samples to voltage values

The Spectrum driver also contains a register that holds the value of the decimal value of the full scale representation of the installed ADC. This value should be used when converting ADC values (in LSB) into real-world voltage values, because this register also automatically takes any specialities into account, such as slightly reduced ADC resolution with reserved codes for gain/offset compensation.

Register	Value	Direction	Description
SPC_MIINST_MAXADCVALUE	1126	read	Contains the decimal code (in LSB) of the ADC full scale value.

In case of a board that uses an 8 bit ADC that provides the full ADC code (without reserving any bits) the returned value would be 128. The the peak value for a ±1.0 V input range would be 1.0 V (or 1000 mv).

A returned sample value of for example +49 (decimal, two's complement, signed representation) would then convert to:

$$V_{in} = 49 \times \frac{1000\ \text{mV}}{128} = 382.81\ \text{mV}$$

A returned sample value of for example -55 (decimal) would then convert to:

$$V_{in} = -55 \times \frac{1000\ \text{mV}}{128} = -429.69\ \text{mV}$$

When converting samples that contain any additional data such as for example additional digital channels or overrange bits, this extra information must be first masked out and a proper sign-extension must be performed, before these values can be used as a signed two's complement value for above formulas.

Clock generation

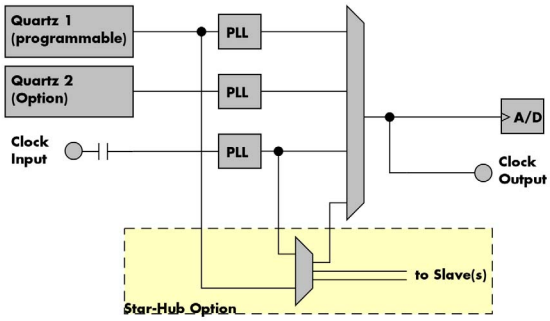
Overview

The different clock modes

The Spectrum M3i/M4i cards offer a wide variety of different clock modes to match all the customers needs. All of the clock modes are described in detail with programming examples in this chapter.

The figure is showing an overview of the complete engine used on all M3i/M4i cards for clock generation.

The purpose of this chapter is to give you a guide to the best matching clock settings for your specific application and needs.



Standard internal sample rate (programmable reference quartz 1)

This is the easiest and most common way to generate a sample rate with no need for additional external clock signals. The sample rate has a very fine resolution, low jitter and a high accuracy. The Quartz 1 is a high quality software programmable clock device acting as a reference to the internal PLL. The specification is found in the technical data section of this manual.

Quartz2 with PLL (option)

This optional second Quartz 2 is for special customer needs, either for a special direct sampling clock or as a very precise reference for the PLL. Please feel free to contact Spectrum for your special needs. The Quarz 2 clock footprint can be equipped with a wide variety of clock sources that are available on the market.

External Clock (reference clock)

Any clock can be fed in that matches the specification of the board. The external clock signal can be used to synchronize the board on a system clock or to feed in an exact matching sample rate. The external clock is divided/multiplied using a PLL allowing a wide range of external clock modes.

Synchronization clock (option Star-Hub)

The star-hub option allows the synchronization of up to 8 cards of the M3i/M4i series from Spectrum with a minimal phase delay between the different cards. The clock is distributed from the master card to all connected cards. As a source it is possible to either use the programmable Quarz 1 clock or the external clock input of the master card. For details on the synchronization option please take a look at the dedicated chapter later in this manual.

Clock Mode Register

The selection of the different clock modes has to be done by the SPC_CLOCKMODE register. All available modes, can be read out by the help of the SPC_AVAILCLOCKMODES register.

Register	Value	Direction	Description
SPC_AVAILCLOCKMODES	20201	read	Bitmask, in which all bits of the below mentioned clock modes are set, if available.
SPC_CLOCKMODE	20200	read/write	Defines the used clock mode or reads out the actual selected one.
SPC_CM_INTPLL	1	Enables internal programmable high precision Quartz 1 for sample clock generation	
SPC_CM_QUARTZ2	4	Enables optional Quartz 2 as reference for sample clock generation	
SPC_CM_EXTREFCLOCK	32	Enables internal PLL with external reference for sample clock generation	

The different clock modes and all other related or required register settings are described on the following pages.

Details on the different clock modes

Standard internal sampling clock (PLL)

The internal sampling clock is generated in default mode by a programmable high precision quartz. You need to select the clock mode by the dedicated register shown in the table below:

Register	Value	Direction	Description
SPC_CLOCKMODE	20200	read/write	Defines the used clock mode
SPC_CM_INTPLL	1	Enables internal programmable high precision Quartz 1 for sample clock generation	

The user does not have to care about how the desired sampling rate is generated by multiplying and dividing internally. You simply write the desired sample rate to the according register shown in the table below and the driver makes all the necessary calculations. If you want to make sure the sample rate has been set correctly you can also read out the register and the driver will give you back the sampling rate that is matching your desired one best.

Register	Value	Direction	Description
SPC_SAMPLERATE	20000	write	Defines the sample rate in Hz for internal sample rate generation.
		read	Read out the internal sample rate that is nearest matching to the desired one.

Independent of the used clock source it is possible to enable the clock output. The clock will be available on the external clock output connector and can be used to synchronize external equipment with the board.

Register	Value	Direction	Description
SPC_CLOCKOUT	20110	read/write	Writing a „1“ enables clock output on external clock output connector. Writing a „0“ disables the clock output (tristate)
SPC_CLOCKOUTFREQUENCY	20111	read	Allows to read out the frequency of an internally synthesized clock present at the clock output.

Example on writing and reading internal sampling rate

```
spcm_dwSetParam_i32 (hDrv, SPC_CLOCKMODE, SPC_CM_INTPLL); // Enables internal programmable quartz 1
spcm_dwSetParam_i64 (hDrv, SPC_SAMPLERATE,  62500000);    // Set internal sampling rate to 62.5 MHz
spcm_dwSetParam_i32 (hDrv, SPC_CLOCKOUT,  1);             // enable the clock output of the card
spcm_dwGetParam_i64 (hDrv, SPC_SAMPLERATE, &lSamplerate); // Read back the programmed sample rate and print
printf („Sample rate = %d\n“, lSamplerate);               // it. Output should be „Sample rate = 62500000“
```

Minimum internal sampling rate

The minimum and the maximum internal sampling rates depend on the specific type of board. Both values can be found in the technical data section of this manual.

Using Quartz2 with PLL (optional, M4i cards only)

In some cases it is necessary to use a special high precision frequency for sampling rate generation. For these applications all cards of the M3i/M4i series can be equipped with a special customer quartz. Please contact Spectrum for details on available oscillators. If your card is equipped with a second oscillator you can enable it for sampling rate generation with the following register:

Register	Value	Direction	Description
SPC_CLOCKMODE	20200	read/write	Defines the used clock mode
SPC_CM_QUARTZ2	4	Enables optional quartz2 for sample clock generation	

The quartz 2 clock is routed through a PLL to allow the generation of sampling rates based on this reference clock. As with internal PLL mode it’s also possible to program the clock mode first, set a desired sampling rate with the SPC_SAMPLERATE register and to read it back. The result will then again be the best matching sampling rate.

Independent of the used clock source it is possible to enable the clock output. The clock will be available on the external clock output connector and can be used to synchronize external equipment with the board.

Register	Value	Direction	Description
SPC_CLOCKOUT	20110	read/write	Writing a „1“ enables clock output on external clock output connector. Writing a „0“ disables the clock output (tristate)
SPC_CLOCKOUTFREQUENCY	20111	read	Allows to read out the frequency of an internally synthesized clock present at the clock output.

External clock (reference clock)

The external clock input is fed through a PLL to the clock system. Therefore the input will act as a reference clock input thus allowing to either use a copy of the external clock or to generate any sampling clock within the allowed range from the reference clock. Please note the limited setup granularity in comparison to the internal sampling clock generation. Details are found in the technical data section.

Register	Value	Direction	Description
SPC_CLOCKMODE	20200	read/write	Defines the used clock mode
SPC_CM_EXTREFCLOCK	32	Enables internal PLL with external reference for sample clock generation	

Due to the fact that the driver needs to know the external fed in frequency for an exact calculation of the sampling rate you must set the SPC_REFERENCECLOCK register accordingly as shown in the table below. The driver then automatically sets the PLL to achieve the desired sampling rate. Please be aware that the PLL has some internal limits and not all desired sampling rates may be reached with every reference clock.

Register	Value	Direction	Description
SPC_REFERENCECLOCK	20140	read/write	Programs the external reference clock in the range stated in the technical data section.
External sampling rate in Hz as an integer value		You need to set up this register exactly to the frequency of the external fed in clock.	

Example of reference clock:

```
spcm_dwSetParam_i32 (hDrv, SPC_CLOCKMODE, SPC_CM_EXTREFCLOCK);   // Set to reference clock mode
spcm_dwSetParam_i32 (hDrv, SPC_REFERENCECLOCK, 10000000);        // Reference clock that is fed in is 10 MHz
spcm_dwSetParam_i64 (hDrv, SPC_SAMPLERATE,     65200000);        // We want to have 62.5 MHz as sampling rate
```

PLL Locking Error

The external clock signal is routed to a PLL to generate any sampling clock from this external clock. Due to the internal structure of the card the PLL is even used if a copy of the clock fed in externally is used for sampling (SPC_REFERENCECLOCK = SPC_SAMPLERATE). The PLL needs a stable and defined external clock with no gaps and no variation in the frequency. The external clock must be present when issuing the start command. It is not possible to start the card with external clock activated and no external clock available.

When starting the card all settings are written to hardware and the PLL is programmed to generate the desired sampling clock. If there has been any change to the clock setting the PLL then tries to lock on the external clock signal to generate the sampling clock. This locking will normally need 10 to 20 ms until the sampling clock is stable. Some clock settings may also need 200 ms to lock the PLL. This waiting time is automatically added at card start.

However if the PLL can not lock on the external clock either because there is no clock available or it hasn't sufficient signal levels or the clock is not stable the driver will return with an error code ERR_CLOCKNOTLOCKED. In that case it is necessary to check the external clock connection. Please see the example below:

```
// settings done to external clock like shown above.
if (spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER) == ERR_CLOCKNOTLOCKED)
    {
    printf („External clock not locked. Please check connection\n");
    return -1;
    }
```

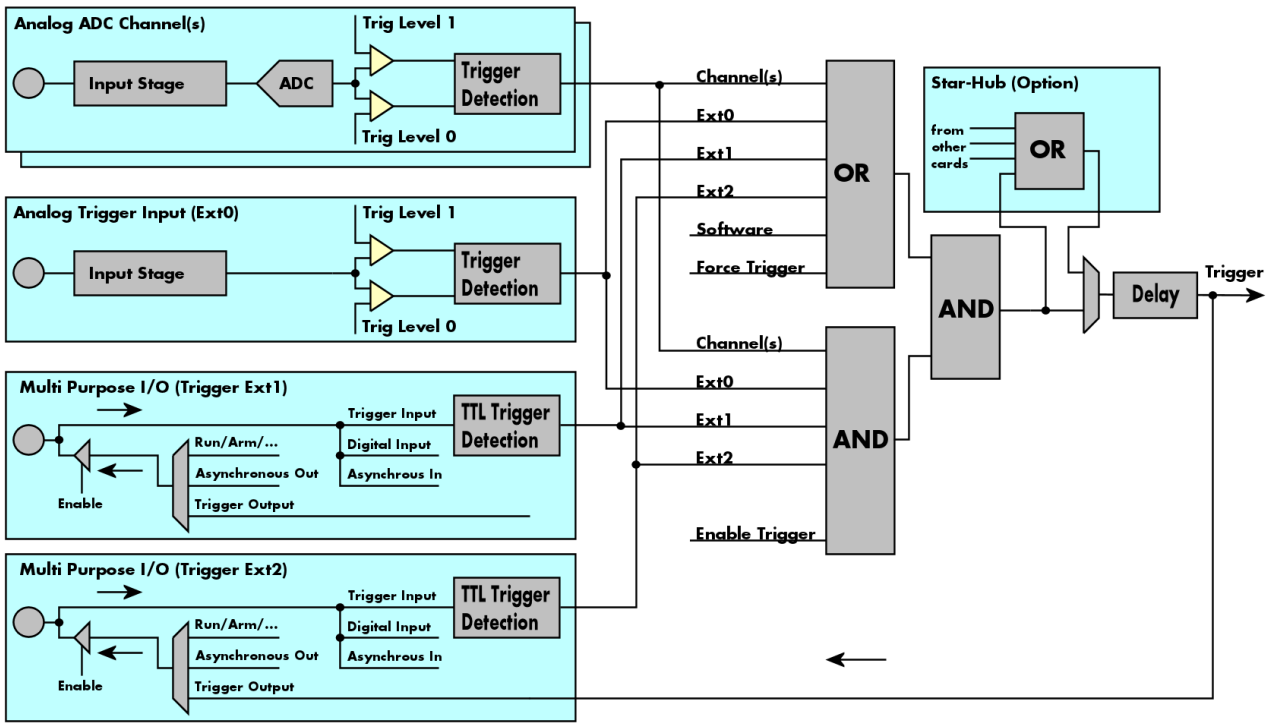
Trigger modes and appendant registers

General Description

The trigger modes of the Spectrum M3i series A/D cards are very extensive and give you the possibility to detect nearly any trigger event you can think of.

You can choose between more than 10 external trigger modes and up to 20 internal trigger modes (on analog acquisition cards) including software and channel trigger, depending on your type of board. Many of the channel trigger modes can be independently set for each input channel (on A/D boards only) resulting in a even bigger variety of modes. This chapter is about to explain all of the different trigger modes and setting up the card's registers for the desired mode.

Trigger Engine Overview



The trigger engine of the M3i card series allows to combine several different trigger sources with OR and AND combination, with a trigger delay or even with an OR combination across several cards when using the Star-Hub option. The above drawing gives a complete overview of the trigger engine and shows all possible features that are available.

Each analog input channel has two trigger level comparators to detect edges as well as windowed triggers. The card has a total of three different additional external trigger sources. One main trigger source which also has two analog level comparators also allowing to use edge and windowed trigger detection and two multi purpose in/outputs that can be software programmed to either additional trigger inputs or trigger outputs or to some extended status signals.

The Enable trigger allows the user to enable or disable all trigger sources (including channel trigger and external trigger) with a single software command. The enable trigger command will not work on force trigger.

When the card is waiting for a trigger event, either a channel trigger or an external trigger the force trigger command allows to force a trigger event with a single software command. The force trigger overrides the enable trigger command.

Before the trigger event is finally generated, it is wired through a programmable trigger delay. This trigger delay will also work when used in a synchronized system thus allowing each card to individually delay its trigger recognition.

Multi Purpose I/O Lines

The M3i series has two multi purpose I/O lines that can be used for a wide variety of functions to help the interconnection with external equipment. The functionality of these multi purpose I/O lines can be software programmed and each of these lines can either be used for input or output.

The multi purpose I/O lines may be used for additional trigger inputs allowing to combine and gate external triggers, for trigger output, for internal arm/run signals output, for asynchronous I/O to control external equipment or as additional digital input lines that are sampled synchronously with the analog data.

The multi purpose I/O lines are available on the front plate and labelled with X0 (line 0 = X0 = Ext1) and X1 (line 1 = X1 = Ext2). As default these lines are switched off.

Please be careful when programming these lines as an output signal being connected with an external signal source may damage components either on the external equipment or on the card itself.

Programming the behaviour

Each multi purpose I/O line can be individually programmed. Please check the available modes by reading the SPCM_X0_AVAILMODES and SPCM_X1_AVAILMODES register first. The available modes may differ from card to card and may be enhanced with new driver/firmware versions to come.

Register	Value	Direction	Description
SPCM_X0_AVAILMODES	47210	read	Bitmask with all bits of the below mentioned modes showing the available modes for (X0 = Ext1)
SPCM_X1_AVAILMODES	47211	read	Bitmask with all bits of the below mentioned modes showing the available modes for (X1 = Ext2)
SPCM_X0_MODE	47200	read/write	Defines the mode for (X0 = Ext1). Only one mode selection is possible to be set at a time
SPCM_X1_MODE	47201	read/write	Defines the mode for (X1 = Ext2). Only one mode selection is possible to be set at a time
SPCM_XMODE_DISABLE	00000000h	No mode selected. Output is tristate (default setup)	
SPCM_XMODE_ASYNCIN	00000001h	Connector is programmed for asynchronous input. Use SPCM_XX_ASYNCIO to read data asynchronous as shown in next chapter.	
SPCM_XMODE_ASYNCOUT	00000002h	Connector is programmed for asynchronous output. Use SPCM_XX_ASYNCIO to write data asynchronous as shown in next chapter.	
SPCM_XMODE_DIGIN	00000004h	Connector is programmed for digital input. Digital channel X0/X1 is written as D14/D15 of data stream during acquisition (12 and 14 bit analog input cards only). Please check the „Sample format" chapter for more details. Please note that automatic sign extension of analog data is switched off as soon as one digital input line is activated.	
SPCM_XMODE_TRIGIN	00000010h	Connector is programmed as additional TTL trigger input. X0/X1 is available as Ext1/Ext2 trigger input. Please be sure to also set the corresponding trigger OR/AND masks to use this trigger input for trigger detection.	
SPCM_XMODE_TRIGOUT	00000020h	Connector is programmed as trigger output and shows the trigger detection. The trigger output is HIGH as long as postcounter is running. After reaching postcounter zero it will become LOW again. In standard FIFO mode the trigger output is HIGH until FIFO mode is stopped.	
SPCM_XMODE_OVROUT	00000040h	Shows the overrange status of the channels at the output. If the analog data of one channel exceeds the input range the overrange signal is set to high level for that time. The overrange status of channel 0 is output on X0 and the overrange status of channel 1 is output on X1.	
SPCM_XMODE_RUNSTATE	00000100h	Connector shows the current run state of the card. If acquisition/output is running the signal is HIGH. If card has stopped the signal is LOW.	
SPCM_XMODE_ARMSTATE	00000200h	Connector shows the current ARM state of the card. If the card is armed and ready to receive a trigger the signal is HIGH. If the card isn't running or the card is still acquiring pretrigger data or the trigger has been detected the signal is LOW.	
SPCM_XMODE_DIRECTTRIGOUT	00000400h	Connector is programmed as direct trigger output and shows the trigger recognition of an fed in trigger event which will lead to a card trigger event. This mode shows the upcoming detection even before the card itself will have triggered. The trigger output is HIGH as long as postcounter is running. After reaching postcounter zero it will become LOW again. This mode ensures that the card is armed and therefor the signaled event trigger event will lead to a card trigger. Please see below for a usage example for this mode.	
SPCM_XMODE_DIRECTTRIGOUT_LR	00000800h	Nearly identical to SPCM_XMODE_DIRECTTRIGOUT, but in contrast the above mode, this mode does not make sure that the card is armed. The user has to take care that the repetition time of the fed in trigger event is longer than the recording and re-arm time, otherwise the direct trigger might be generated although this event cannot be a properly detected card trigger. Please see below for a usage example for this mode.	

Please note that a change to the SPCM_X0_MODE or SPCM_X1_MODE will only be updated with the next call to either the M2CMD_CARD_START or M2CMD_CARD_WRITESETUP register. For further details please see the relating chapter on the M2CMD_CARD registers.

Using asynchronous I/O

To use asynchronous I/O on the multi purpose I/O lines it is first necessary to switch these lines to the desired asynchronous mode by programming the above explained mode registers. As a special feature asynchronous input can also be read if the mode is set to trigger input or digital input.

Register	Value	Direction	Description
SPCM_XX_ASYNCIO	47220	read/write	Connector X0 is linked to bit 0 of the register while connector X1 is linked to bit 1 of this register. Data is written/read immediately without any relation to the currently used sampling rate or mode. If a line is programmed to output, reading this line asynchronously will return the current output level.

Example of asynchronous write and read. We write a high pulse on output X1 and wait for a high level answer on input X0:

```
spcm_dwSetParam_i32 (hDrv, SPCM_X0_MODE, SPCM_XMODE_ASYNCIN);   // X0 set to asynchronous input
spcm_dwSetParam_i32 (hDrv, SPCM_X1_MODE, SPCM_XMODE_ASYNCOUT);  // X1 set to asynchronous output

spcm_dwSetParam_i32 (hDrv, SPCM_XX_ASYNCIO, 0);                 // programming a high pulse on output
spcm_dwSetParam_i32 (hDrv, SPCM_XX_ASYNCIO, 2);
spcm_dwSetParam_i32 (hDrv, SPCM_XX_ASYNCIO, 0);

do {
    spcm_dwGetParam_i32 (hDrv, SPCM_XX_ASYNCIO, &lAsyncIn);     // read input in a loop
} while ((lAsyncIn & 1) == 0)                                   // until X0 is going to high level
```

Special behaviour of trigger output

As the driver of the M3i series is the same as the driver for the M2i series and some old software may rely on register structure of the M2i card series there is a special compatible trigger output register that will work according to the M2i series style. It is not recommended to use this register unless you're writing software for both card series:

Register	Value	Direction	Description
SPC_TRIG_OUTPUT	40100	read/write	M2i style trigger output programming. Write a „1" to enable X1 trigger output (SPCM_X1_MODE = SPCM_XMODE_TRIGOUT) and X0 run state (SPCM_X0_MODE = SPCM_XMODE_RUNSTATE). Write a „0" to disable both outputs (SPCM_X0_MODE = SPCM_X1_MODE = SPCM_XMODE_DISABLE)

The SPC_TRIG_OUTPUT register overrides the multi purpose I/O settings done by SPCM_X0_MODE and SPCM_X1_MODE and vice versa. Please do not use both methods in one program.

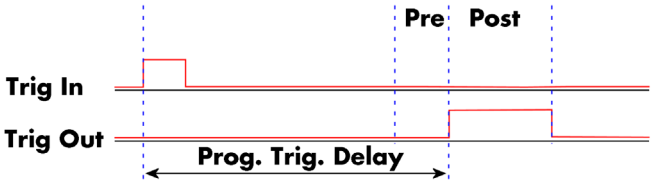
Special direct trigger output modes

The trigger output of the cards can be used to start external equipment. To cope requirements for different applications, all M3i cards support different output modes.

„Standard" Trigger Output

In this mode the output signal indicates the internal trigger event after the trigger delay, and therefore the begin of post trigger area. The trigger output and the recording can be delayed by programming the user trigger delay.

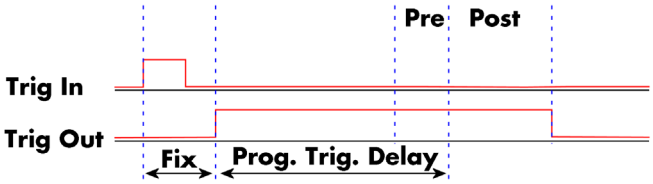
For details on the trigger delay, please see the related chapter in this manual.



„Direct" Trigger Output

In this mode the output signal indicates that the external fed in trigger event (external or channel trigger) will lead to a recording after a fix delay and the optional programmed trigger delay. The start of the recording can be delayed by programming the user trigger delay.
This can be useful when the trigger output is to be used to start the device under test, whilst avoiding the need to record unneeded data in the pre-trigger area.

For details on the trigger delay, please see the related chapter in this manual.

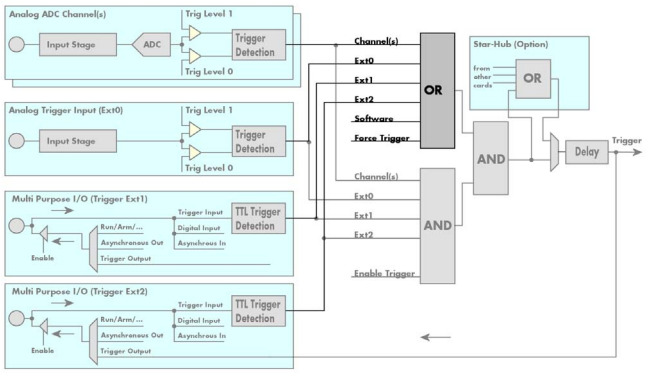


Using the direct trigger output modes requires the following driver and firmware version depending on your card. Please update your system to the newest versions to run these modes mode.

- Driver version V2.06 (or newer)
- Base Ctrl firmware version V6 (or newer)
- M3i.21xx cards : Modul Ctrl firmware version V1 (or newer)
- M3i.32xx cards : Modul Ctrl firmware version V6 (or newer)
- M3i.41xx cards : Modul Ctrl firmware version V6 (or newer)

Trigger masks

Trigger OR mask



The purpose of this passage is to explain the trigger OR mask (see left figure) and all the appendant software registers in detail.

The OR mask shown in the overview before as one object, is separated into two parts: a general OR mask for external trigger (external analog and multi purpose TTL trigger) and software trigger and a channel OR mask.

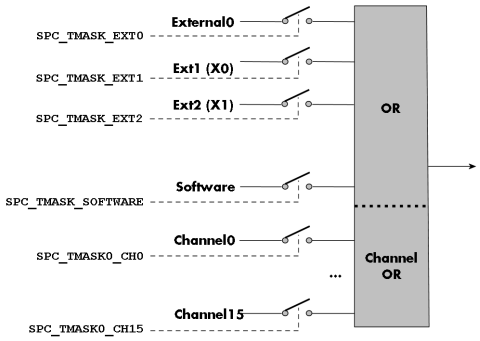
Every trigger source of the M3i series cards is wired to one of the above mentioned OR masks. The user then can program which trigger source will be recognized, and which one won't.

This selection for the general mask is realized with the SPC_TRIG_ORMASK register in combination with constants for every possible trigger source.

This selection for the channel mask is realized with the SPC_TRIG_CH_ORMASK0 register in combination with constants for every possible channel trigger source.

In either case the sources are coded as a bitfield, so that they can be combined by one access to the driver with the help of a bitwise OR.

The table below shows the relating register for the general OR mask and the possible constants that can be written to it.



Register	Value	Direction	Description
SPC_TRIG_AVAILORMASK	40400	read	Bitmask, in which all bits of the below mentioned sources for the OR mask are set, if available.
SPC_TRIG_ORMASK	40410	read/write	Defines the events included within the trigger OR mask of the card.
SPC_TMASK_NONE	0	No trigger source selected	
SPC_TMASK_SOFTWARE	1h	Enables the software trigger for the OR mask. The card will trigger immediately after start.	
SPC_TMASK_EXT0	2h	Enables the external (analog) trigger 0 for the OR mask. The card will trigger when the programmed condition for this input is valid.	
SPC_TMASK_EXT1	4h	Enables the external (TTL) trigger 1 for the OR mask. Please note that the mode of the multi purpose connector X0 must be programmed to trigger input if using the Ext1 trigger (SPCM_X0_MODE=SPCM_XMODE_TRIGIN). The card will trigger when the programmed condition for this input is valid.	
SPC_TMASK_EXT2	8h	Enables the external (TTL) trigger 2 for the OR mask. Please note that the mode of the multi purpose connector X1 must be programmed to trigger input if using the Ext2 trigger (SPCM_X1_MODE=SPCM_XMODE_TRIGIN). The card will trigger when the programmed condition for this input is valid.	

Please note that as default the SPC_TRIG_ORMASK is set to SPC_TMASK_SOFTWARE. When not using any trigger mode requiring values in the SPC_TRIG_ORMASK register, this mask should explicitely cleared, as otherwise the software trigger will override other modes.

The following example shows, how to setup the OR mask, for an external trigger. As an example a simple edge detection has been chosen. The explanation and a detailed description of the different trigger modes for the external trigger inputs will be shown in the dedicated passage within this chapter.

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_EXT0); // Enable external trigger within the OR mask
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_LEVEL0, 1800);      // Trigger level set to 1.8 V
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_MODE, SPC_TM_POS);  // Setting up external trigger for rising edges
```

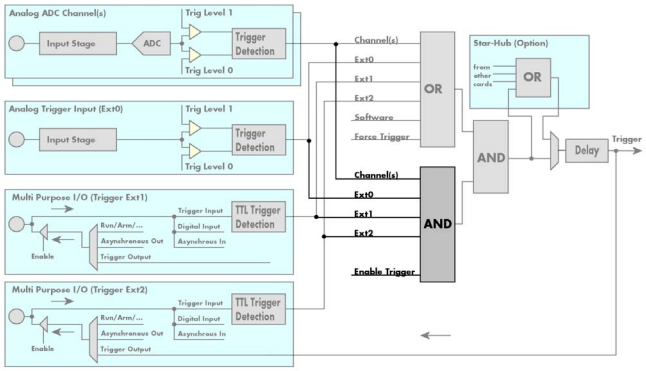
The table below is showing the registers for the channel OR mask and the possible constants that can be written to it.

Register	Value	Direction	Description
SPC_TRIG_CH_AVAILORMASK0	40450	read	Bitmask, in which all bits of the below mentioned sources/channels (0…31) for the channel OR mask are set, if available.
SPC_TRIG_CH_ORMASK0	40460	read/write	Includes the analog or digital channels (0…31) within the channel trigger OR mask of the card.
SPC_TMASK0_CH0	00000001h	Enables channel0 for recognition within the channel OR mask.	
SPC_TMASK0_CH1	00000002h	Enables channel1 for recognition within the channel OR mask.	
SPC_TMASK0_CH2	00000004h	Enables channel2 for recognition within the channel OR mask.	
SPC_TMASK0_CH3	00000008h	Enables channel3 for recognition within the channel OR mask.	
…	…	…	
SPC_TMASK0_CH28	10000000h	Enables channel28 for recognition within the channel OR mask.	
SPC_TMASK0_CH29	20000000h	Enables channel29 for recognition within the channel OR mask.	
SPC_TMASK0_CH30	40000000h	Enables channel30 for recognition within the channel OR mask.	
SPC_TMASK0_CH31	80000000h	Enables channel31 for recognition within the channel OR mask.	

The following example shows, how to setup the OR mask for channel trigger. As an example a simple edge detection has been chosen. The explanation and a detailed description of the different trigger modes for the external TTL trigger inputs will be shown in the dedicated passage within this chapter.

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_NONE);     // disable default software trigger
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH_ORMASK0, SPC_TMASK_CH0); // Enable channel0 trigger within the OR mask
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH0_LEVEL0, 0);             // Trigger level is zero crossing
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_MODE,  SPC_TM_POS);    // Setting up external trigger for rising edges
```

Trigger AND mask



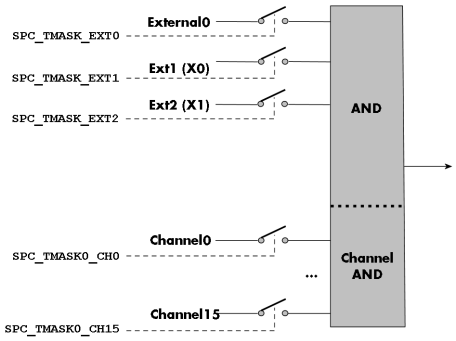
The purpose of this passage is to explain the trigger AND mask (see left figure) and all the appendant software registers in detail.

The AND mask shown in the overview before as one object, is separated into two parts: a general AND mask for external trigger and software trigger and a channel AND mask.

Every trigger source of the M3i series cards except the software trigger is wired to one of the above mentioned AND masks. The user then can program which trigger source will be recognized, and which one won't.

This selection for the general mask is realized with the SPC_TRIG_ANDMASK register in combination with constants for every possible trigger source.

This selection for the channel mask is realized with the SPC_TRIG_CH_ANDMASK0 register in combination with constants for every possible channel trigger source. In either case the sources are coded as a bit-field, so that they can be combined by one access to the driver with the help of a bitwise OR.



The table below shows the relating register for the general AND mask and the possible constants that can be written to it.

Register	Value	Direction	Description
SPC_TRIG_AVAILANDMASK	40420	read	Bitmask, in which all bits of the below mentioned sources for the AND mask are set, if available.
SPC_TRIG_ANDMASK	40430	read/write	Defines the events included within the trigger AND mask of the card.
SPC_TMASK_NONE	0	No trigger source selected	
SPC_TMASK_EXT0	2h	Enables the external (analog) trigger 0 for the AND mask. The card will trigger when the programmed condition for this input is valid.	
SPC_TMASK_EXT1	4h	Enables the external (TTL) trigger 1 for the AND mask. Please note that the mode of the multi purpose connector X0 must be programmed to trigger input if using the Ext1 trigger (SPCM_X0_MODE=SPCM_XMODE_TRIGIN). The card will trigger when the programmed condition for this input is valid.	
SPC_TMASK_EXT2	8h	Enables the external (TTL) trigger 1 for the AND mask. Please note that the mode of the multi purpose connector X1 must be programmed to trigger input if using the Ext2 trigger (SPCM_X1_MODE=SPCM_XMODE_TRIGIN). The card will trigger when the programmed condition for this input is valid.	

The following example shows, how to setup the AND mask, for an external trigger. As an example a simple high level detection has been chosen. The explanation and a detailed description of the different trigger modes for the external trigger inputs will be shown in the dedicated passage within this chapter.

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_NONE);  // disable default software trigger
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ANDMASK, SPC_TMASK_EXT0); // Enable external trigger within the AND mask
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_LEVEL0, 2000);       // Trigger level is 2.0 V (2000 mV)
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_MODE, SPC_TM_HIGH  );// Setting up external trigger for HIGH level
```

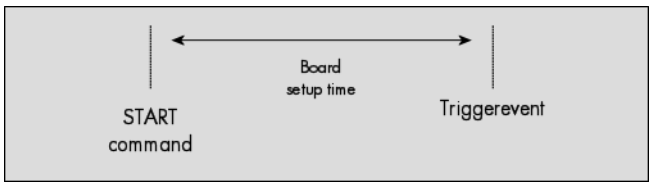
The table below is showing the constants for the channel AND mask and all the constants for the different channels.

Register	Value	Direction	Description
SPC_TRIG_CH_AVAILANDASK0	40470	read	Bitmask, in which all bits of the below mentioned sources/channels (0…31) for the channel AND mask are set, if available.
SPC_TRIG_CH_ANDMASK0	40480	read/write	Includes the analog or digital channels (0…31) within the channel trigger AND mask of the card.
SPC_TMASK0_CH0	00000001h	Enables channel0 for recognition within the channel OR mask.	
SPC_TMASK0_CH1	00000002h	Enables channel1 for recognition within the channel OR mask.	
SPC_TMASK0_CH2	00000004h	Enables channel2 for recognition within the channel OR mask.	
SPC_TMASK0_CH3	00000008h	Enables channel3 for recognition within the channel OR mask.	
…	…	…	
SPC_TMASK0_CH28	10000000h	Enables channel28 for recognition within the channel OR mask.	
SPC_TMASK0_CH29	20000000h	Enables channel29 for recognition within the channel OR mask.	
SPC_TMASK0_CH30	40000000h	Enables channel30 for recognition within the channel OR mask.	
SPC_TMASK0_CH31	80000000h	Enables channel31 for recognition within the channel OR mask.	

The following example shows, how to setup the AND mask for a channel trigger. As an example a simple level detection has been chosen. The explanation and a detailed description of the different trigger modes for the channel trigger inputs will be shown in the dedicated passage within this chapter.

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_NONE);     // disable default software trigger
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH_ANDMASK0, SPC_TMASK_CH0); // Enable channel0 trigger within the AND mask
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH0_LEVEL0, 0);              // channel level to detect is zero level
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH0_MODE, SPC_TM_HIGH);      // Setting up ch0 trigger for HIGH levels
```

Software trigger



The software trigger is the easiest way of triggering any Spectrum board. The acquisition or replay of data will start immediately after the card is started and the trigger engine is armed. The resulting delay upon start includes the time the board needs for its setup and the time for recording the pre-trigger area (for acquisition cards).
For enabling the software trigger one simply has to include the software event within the trigger OR mask, as the following table is showing:

Register	Value	Direction	Description
SPC_TRIG_ORMASK	40410	read/write	Defines the events included within the trigger OR mask of the card.
SPC_TMASK_SOFTWARE	1h	Sets the trigger mode to software, so that the recording/replay starts immediately.	

Example for setting up the software trigger:

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_SOFTWARE); // Internal software trigger mode is used
```

Force- and Enable trigger

In addition to the software trigger (free run) it is also possible to force a trigger event by software while the board is waiting for a real physical trigger event. The forcetrigger command will only have any effect, when the board is waiting for a trigger event. The command for forcing a trigger event is shown in the table below.

Issuing the forcetrigger command will every time only generate one trigger event. If for example using Multiple Recording that will result in only one segment being acquired by forcetrigger. After execution of the forcetrigger command the trigger engine will fall back to the trigger mode that was originally programmed and will again wait for a trigger event.

Register	Value	Direction	Description
SPC_M2CMD	100	write	Command register of the M2i/M3i/M4i/M4x/M2p series cards.

M2CMD_CARD_FORCETRIGGER	10h	Forces a trigger event if the hardware is still waiting for a trigger event.

The example shows, how to use the forcetrigger command:

```
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_FORCETRIGGER); // Force trigger is used.
```

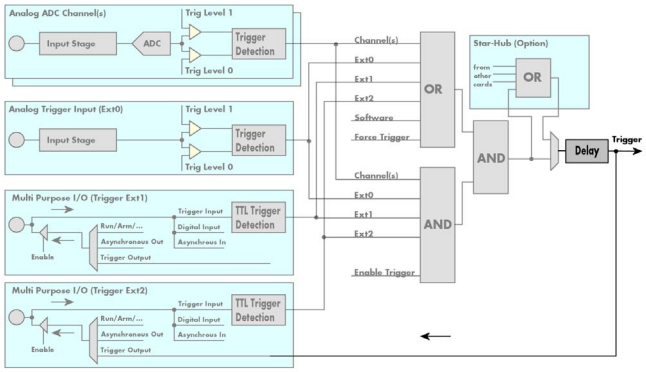
It is also possible to enable (arm) or disable (disarm) the card's whole triggerengine by software. By default the trigger engine is disabled.

Register	Value	Direction	Description
SPC_M2CMD	100	write	Command register of the M2i/M3i/M4i/M4x/M2p series cards.
M2CMD_CARD_ENABLETRIGGER	8h	Enables the trigger engine. Any trigger event will now be recognized.	
M2CMD_CARD_DISABLETRIGGER	20h	Disables the trigger engine. No trigger events will be recognized, except force trigger.	

The example shows, how to arm and disarm the card's trigger engine properly:

```
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_ENABLETRIGGER);  // Trigger engine is armed.
...
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_DISABLETRIGGER); // Trigger engine is disarmed.
```

Trigger delay



All of the Spectrum M3i series cards allow the user to program an additional trigger delay. As shown in the trigger overview section, this delay is the last element in the trigger chain. Therefore the user does not have to care for the sources when programming the trigger delay.

As shown in the overview the trigger delay is located after the star-hub connection meaning that every M3i card being synchronized can still have its own trigger delay programmed. The Star-Hub will combine the original trigger events before the result is being delayed.

The delay is programmed in samples. The resulting time delay will therefore be [Programmed Delay] / [Sampling Rate].

The following table shows the related register and the possible values. A value of 0 disables the trigger delay.

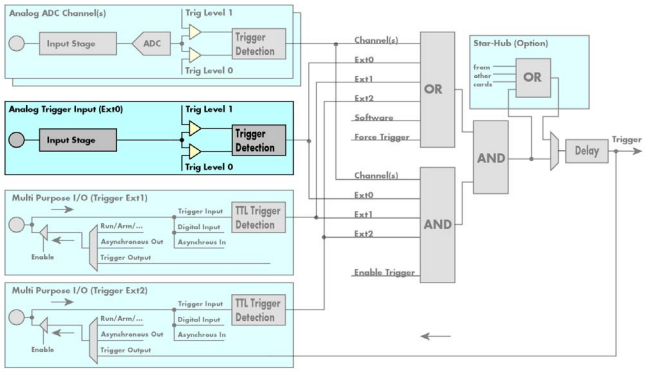
Register	Value	Direction	Description
SPC_TRIG_AVAILDELAY	40800	read	Contains the maximum available delay as a decimal integer value.
SPC_TRIG_DELAY	40810	read/write	Defines the delay for the detected trigger events.
0		No additional delay will be added. The resulting internal delay is mentioned in the technical data section.	
8…[8G -8] in steps of 8 (12, 14 and 16 bit cards)		Defines the additional trigger delay in number of sample clocks. The trigger delay is a full 33 bit counter and can therefore be programmed up to (8GSamples - 8) = 8589934584. Stepsize is 8 samples for 12, 14 and 16 bit cards.	
16…[8G -16] in steps of 16 (8 bit cards)		Defines the additional trigger delay in number of sample clocks. The trigger delay is a full 33 bit counter and can therefore be programmed up to (8GSamples - 16) = 8589934576. Stepsize is 16 samples for 8 bit cards.	

The example shows, how to use the trigger delay command:

```
spcm_dwSetParam_i64 (hDrv, SPC_TRIG_DELAY, 2000); // A detected trigger event will be
                                                  // delayed for 2000 sample clocks.
```

Using the delay trigger does not affect the ratio between pre trigger and post trigger recorded number of samples, but only shifts the trigger event itself. For changing these values, please take a look in the relating chapter about „Acquisition Modes".

External (analog) trigger



The M3i series has one main external trigger input consisting of an input stage with programmable termination and programmable AC/DC coupling and two comparators that can be programmed in the range of +/- 5000 mV. Using two comparators offers a wide range of different trigger modes that are support like edge, level, re-arm and window trigger.

The external analog trigger can be easily combined with channel trigger or with one or two of the multi purpose connectors being programmed as additional external TTL trigger inputs. The programming of the masks and the multi purpose I/O behaviour is shown in the chapters above.

Trigger Mode

Please find the external (analog) trigger input modes below. A detailed description of the modes follows in the next chapters..

Register	Value	Direction	Description
SPC_TRIG_EXT0_AVAILMODES	40500	read	Bitmask shoeing all available trigger modes for external 0 (Ext0) = main analog trigger input
SPC_TRIG_EXT0_MODE	40510	read/write	Defines the external trigger mode for the external MMCX connector trigger input. The trigger need to be added to either OR or AND mask input to be activated.
SPC_TM_NONE	00000000h	Channel is not used for trigger detection. This is as with the trigger masks another possibility for disabling channels.	
SPC_TM_POS	00000001h	Trigger detection for positive edges (crossing level 0 from below to above)	
SPC_TM_NEG	00000002h	Trigger detection for negative edges (crossing level 0 from above to below)	
SPC_TM_POS | SPC_TM_REARM	01000001h	Trigger detection for positive edges on lebel 0. Trigger is armed when crossing level 1 to avoid false trigger on noise	
SPC_TM_NEG | SPC_TM_REARM	01000002h	Trigger detection for negative edges on lebel 1. Trigger is armed when crossing level 0 to avoid false trigger on noise	
SPC_TM_BOTH	00000004h	Trigger detection for positive and negative edges (any crossing of level 0)	
SPC_TM_HIGH	00000008h	Trigger detection for HIGH levels (signal above level 0)	
SPC_TM_LOW	00000010h	Trigger detection for LOW levels (signal below level 0)	
SPC_TM_WINENTER	00000020h	Window trigger for entering area between level 0 and level 1	
SPC_TM_WINLEAVE	00000040h	Window trigger for leaving area between level 0 and level 1	
SPC_TM_INWIN	00000080h	Window trigger for signal inside window between level 0 and level 1	
SPC_TM_OUTSIDEWIN	00000100h	Window trigger for signal outside window between level 0 and level 1	

For all external edge and level trigger modes, the OR mask must contain the corresponding input, as the following table shows:

Register	Value	Direction	Description
SPC_TRIG_ORMASK	40410	read/write	Defines the OR mask for the different trigger sources.
SPC_TMASK_EXT0	2h	Enable external trigger input for the OR mask	
SPC_TMASK_XIO0	100h	Enable extra TTL input 0 for the OR mask. On plain cards this input is only available if the option BaseXIO is installed. As part of the digitizerNETBOX this input is available as connector Trigger B.	
SPC_TMASK_XIO1	200h	Enable extra TTL input 1 for the OR mask. These trigger inputs are only available, when option BaseXIO is installed.	

Trigger Input Termination

The external trigger input is a high impedance input with 1 MOhm termination aginst GND. It is possible to program a 50 Ohm termination by software to terminate fast trigger signals correctly. If you enable the termination, please make sure, that your trigger source is capable to deliver the needed current. Please check carefully whether the source is able to fulfil the trigger input specification given in the technical data section.

Register	Value	Direction	Description
SPC_TRIG_TERM	40110	read/write	A „1“ sets the 50 Ohm termination for external trigger signals. A „0“ sets the high impedance termination

Please note that the signal levels will drop by 50% if using the 50 ohm termination and your source also has 50 ohm output impedance (both terminations will then work as a 1:2 divider). In that case it will be necessary to reprogram the trigger levels to match the new signal levels. In case of problems receiving a trigger please check the signal level of your source while connected to the terminated input.

Trigger Input Coupling

The external trigger input can be switched by software between AC and DC coupling. Please see the technical data section for details on the AC bandwidth.

Register	Value	Direction	Description
SPC_TRIG_EXT0_ACDC	40120	read/write	A „1" sets the AC coupling for the external trigger input. A „0" sets the DC coupling (default)

Trigger level

All of the external (analog) trigger modes listed above require at least one trigger level to be set (except SPC_TM_NONE of course). Some like the window or the re-arm triggers require even two levels (upper and lower level) to be set. The meaning of the trigger levels is depending on the selected mode and can be found in the detailled trigger mode description that follows.

Trigger levels for the external (analog) trigger to be programmed in mV:

Register	Value	Direction	Description	Range
SPC_TRIG_EXT_AVAIL0_MIN	42340	read	returns the minimum trigger level to be programmed in mV	
SPC_TRIG_EXT_AVAIL0_MAX	42341	read	returns the maximum trigger level to be programmed in mV	
SPC_TRIG_EXT_AVAIL0_STEP	42342	read	returns the step size of trigger level to be programmed in mV	
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Trigger level 0 for external trigger	-5000 mV to +5000 mV
SPC_TRIG_EXT0_LEVEL1	42321	read/write	Trigger level 1 for external trigger	-5000 mV to +5000 mV

Detailed description of the external analog trigger modes

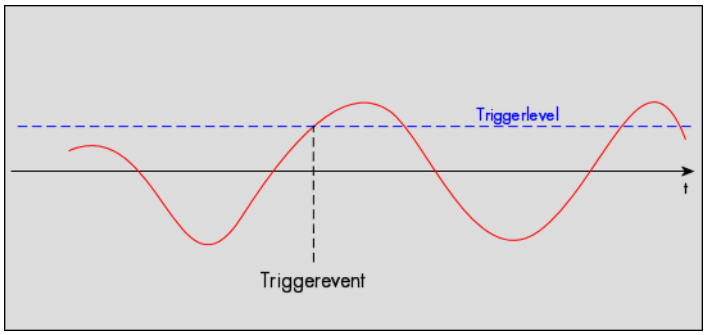
For all external analog trigger modes shown below, either the OR mask or the AND must contain the external trigger to activate the external input as trigger source:.

Register	Value	Direction	Description
SPC_TRIG_ORMASK	40410	read/write	Defines the events included within the trigger OR mask of the card.
SPC_TRIG_ANDMASK	40430	read/write	Defines the events included within the trigger AND mask of the card.
SPC_TMASK_EXT0	2h	Enables the external (analog) trigger 0 for the mask.	

Trigger on positive edge

The trigger input is continuously sampled with the selected sample rate. If the programmed trigger level is crossed by the trigger signal from lower values to higher values (rising edge) then the trigger event will be detected.

This edge triggered external trigger mode correspond to the trigger possibilities of usual oscilloscopes.

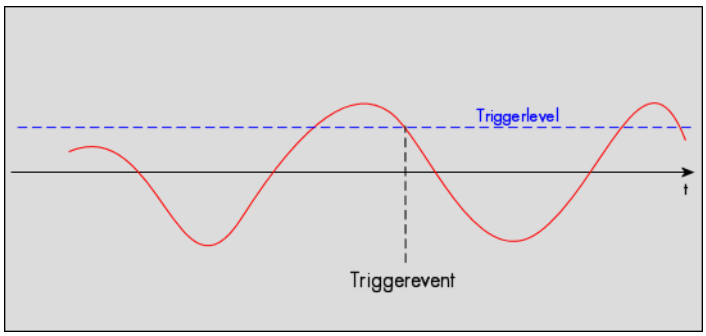


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_POS	1h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the desired trigger level in mV	mV

Trigger on negative edge

The trigger input is continuously sampled with the selected sample rate. If the programmed trigger level is crossed by the trigger signal from higher values to lower values (falling edge) then the trigger event will be detected.

This edge triggered external trigger mode correspond to the trigger possibilities of usual oscilloscopes.

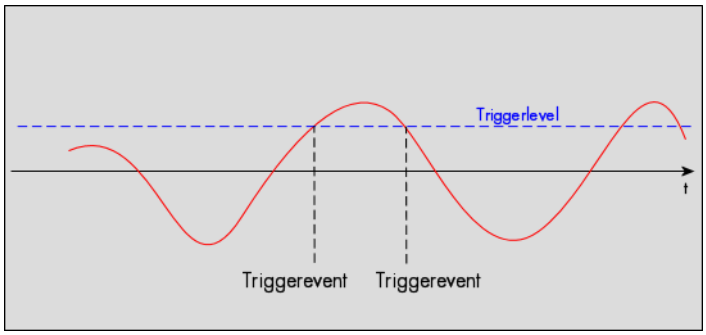


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_NEG	2h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the desired trigger level in mV	mV

Trigger on positive and negative edge

The trigger input is continuously sampled with the selected sample rate. If the programmed trigger level is crossed by the trigger signal (either rising or falling edge) the trigger event will be detected.

Thise edge triggered external trigger mode correspond to the trigger possibilities of usual oscilloscopes.

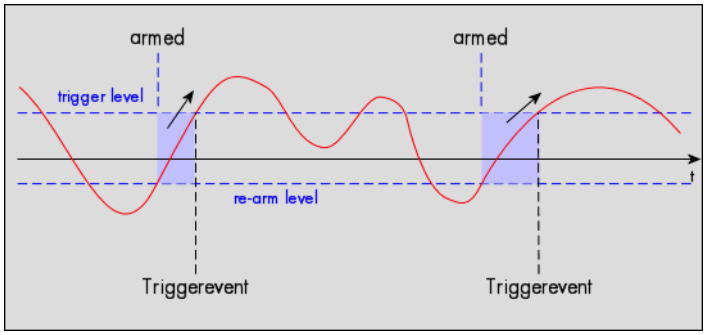


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_BOTH	4h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the desired trigger level in mV	mV

Re-arm trigger on positive edge

The trigger input is continuously sampled with the selected sample rate. If the programmed re-arm level is crossed from lower to higher values, the trigger engine is armed and waiting for trigger. If the programmed trigger level is crossed by the trigger signal from lower values to higher values (rising edge) then the trigger event will be detected and the trigger engine will be disarmed. A new trigger event is only detected if the trigger engine is armed again.

The re-arm trigger modes can be used to prevent the board from triggering on wrong edges in noisy signals.

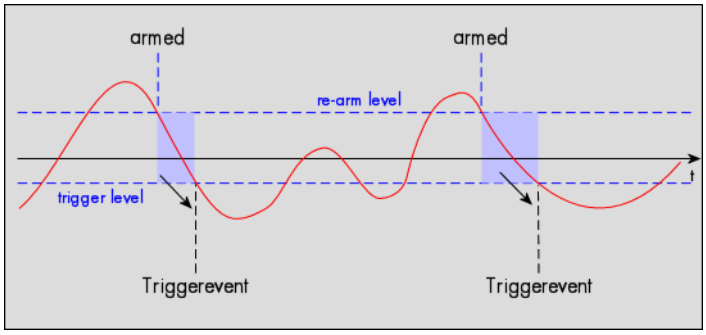


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_POS | SPC_TM_REARM	01000001h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the desired trigger level in mV	mV
SPC_TRIG_EXT0_LEVEL1	42330	read/write	Defines the re-arm level in mV	mV

Re-arm trigger on negative edge

The trigger input is continuously sampled with the selected sample rate. If the programmed re-arm level is crossed from higher to lower values, the trigger engine is armed and waiting for trigger. If the programmed trigger level is crossed by the trigger signal from higher values to lower values (falling edge) then the trigger event will be detected and the trigger engine will be disarmed. A new trigger event is only detected, if the trigger engine is armed again.

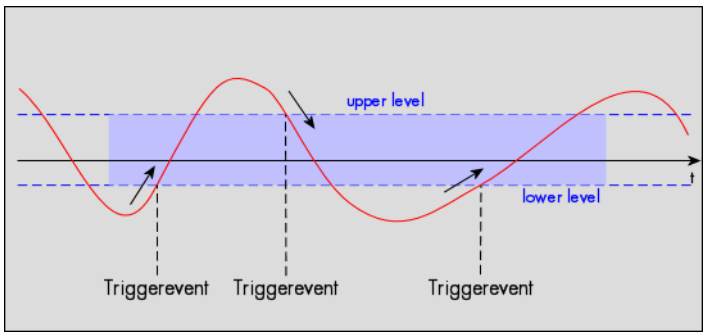
The re-arm trigger modes can be used to prevent the board from triggering on wrong edges in noisy signals.



Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_NEG | SPC_TM_REARM	01000002h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Defines the re-arm level in mV	mV
SPC_TRIG_EXT0_LEVEL1	42330	read/write	Set it to the desired trigger level in mV	mV

Window trigger for entering signals

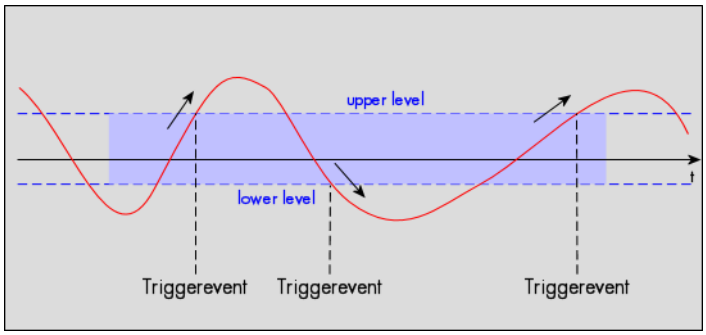
The trigger input is continuously sampled with the selected sample rate. The upper and the lower level define a window. Every time the signal enters the window from the outside, a trigger event will be detected.



Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_WINENTER	00000020h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the upper trigger level in mV	mV
SPC_TRIG_EXT0_LEVEL1	42330	read/write	Set it to the lower trigger level in mV	mV

Window trigger for leaving signals

The trigger input is continuously sampled with the selected sample rate. The upper and the lower level define a window. Every time the signal leaves the window from the inside, a trigger event will be detected.

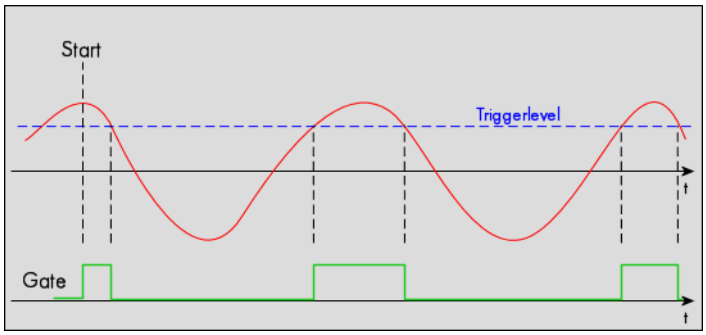


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_WINLEAVE	00000040h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the upper trigger level in mV	mV
SPC_TRIG_EXT0_LEVEL1	42330	read/write	Set it to the lower trigger level in mV	mV

High level trigger

This trigger mode will generate an internal gate signal that can be useful in conjunction with a second trigger mode to gate that second trigger. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the high level (acting like positive edge trigger) or if the trigger signal is already above the programmed level at the start it will immediately detect a trigger event.

The trigger input is continuously sampled with the selected sample rate. The trigger event will be detected if the trigger input is above the programmed trigger level.

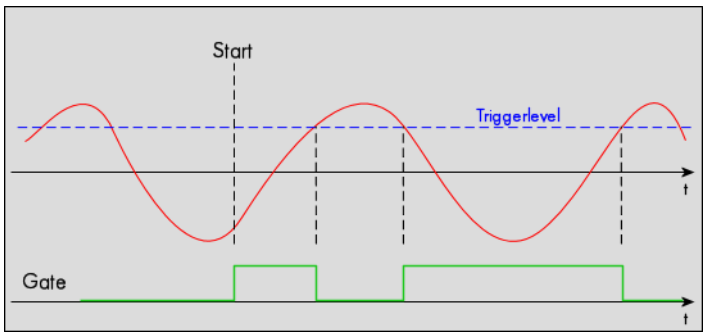


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_HIGH	00000008h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the upper trigger level in mV	mV

Low level trigger

This trigger mode will generate an internal gate signal that can be useful in conjunction with a second trigger mode to gate that second trigger. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the low level (acting like negative edge trigger) or if the trigger signal is already above the programmed level at the start it will immediately detect a trigger event.

The trigger input is continuously sampled with the selected sample rate. The trigger event will be detected if the trigger input is below the programmed trigger level.

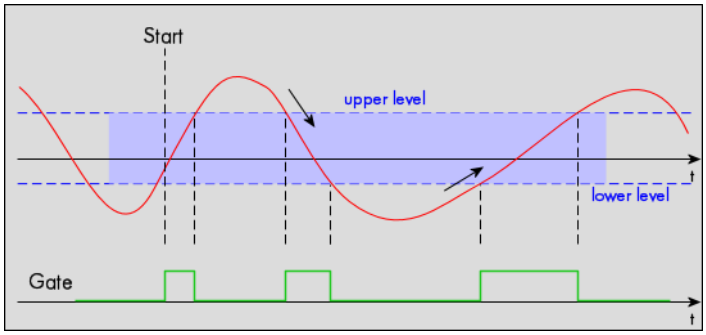


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_LOW	00000010h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the upper trigger level in mV	mV

In window trigger

This trigger mode will generate an internal gate signal that can be useful in conjunction with a second trigger mode to gate that second trigger. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the window defined by the two trigger levels (acting like window enter trigger) or if the trigger signal is already inside the programmed window at the start it will immediately detect a trigger event.

The trigger input is continuously sampled with the selected sample rate. The trigger event will be detected if the trigger input is inside the programmed trigger window.

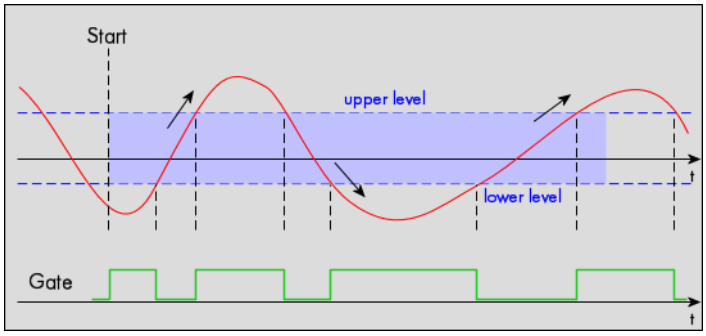


Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_INWIN	00000080h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the upper trigger level in mV	mV
SPC_TRIG_EXT0_LEVEL1	42330	read/write	Set it to the lower trigger level in mV	mV

Outside window trigger

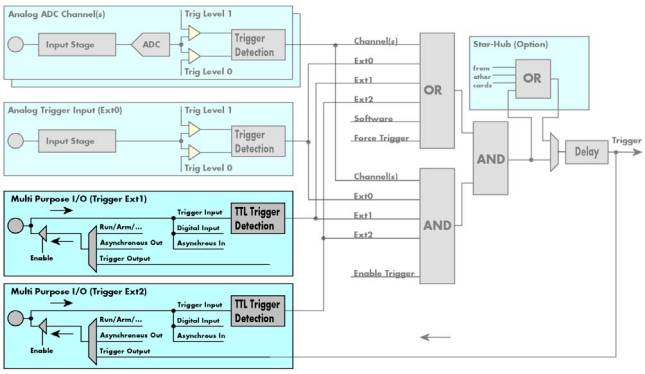
This trigger mode will generate an internal gate signal that can be useful in conjunction with a second trigger mode to gate that second trigger. If using this mode as a single trigger source the card will detect a trigger event at the time when leaving the window defined by the two trigger levels (acting like leaving window trigger) or if the trigger signal is already outside the programmed windo at the start it will immediately detect a trigger event.

The trigger input is continuously sampled with the selected sample rate. The trigger event will be detected if the trigger input is outside the programmed trigger window.



Register	Value	Direction	set to	Value
SPC_TRIG_EXT0_MODE	40510	read/write	SPC_TM_OUTSIDEWIN	00000100h
SPC_TRIG_EXT0_LEVEL0	42320	read/write	Set it to the upper trigger level in mV	mV
SPC_TRIG_EXT0_LEVEL1	42330	read/write	Set it to the lower trigger level in mV	mV

External (TTL) trigger using multi purpose I/O connectors



The M3i card series has two additional multi purpose lines that can be programmed as additional TTL trigger inputs to be combined either with the main (analog) external trigger or with some of the channel trigger modes explained later in this manual.

Please keep in mind that the multi purpose I/O lines need to be switched to trigger input prior to being operated as trigger input. The programming of the masks and the multi purpose I/O behaviour is shown in the chapters before.

TTL Trigger Mode

Please find the multi purpose TTL trigger input modes below. A detailed description of the modes follows in the next chapters..

Register	Value	Direction	Description
SPC_TRIG_EXT1_AVAILMODES	40501	read	Bitmask shwoing all available trigger modes for external 1 (Ext1) = multi purpose X0
SPC_TRIG_EXT2_AVAILMODES	40502	read	Bitmask shwoing all available trigger modes for external 2 (Ext2) = multi purpose X1
SPC_TRIG_EXT1_MODE	40511	read/write	Defines the external trigger mode for the multi purpose X0 MMCX connector trigger input. The trigger need to be added to either OR or AND mask input to be activated.
SPC_TRIG_EXT2_MODE	40512	read/write	Defines the external trigger mode for the multi purpose X1 MMCX connector trigger input. The trigger need to be added to either OR or AND mask input to be activated.
SPC_TM_NONE	00000000h	Channel is not used for trigger detection. This is as with the trigger masks another possibility for disabling channels.	
SPC_TM_POS	00000001h	Trigger detection for positive edges	
SPC_TM_NEG	00000002h	Trigger detection for negative edges	
SPC_TM_BOTH	00000004h	Trigger detection for positive and negative edges	
SPC_TM_HIGH	00000008h	Trigger detection for HIGH levels	
SPC_TM_LOW	00000010h	Trigger detection for LOW levels	

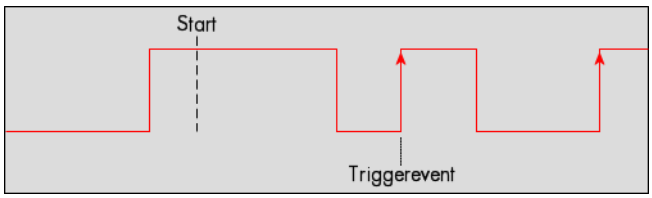
For all external edge and level trigger modes, the OR mask must contain the corresponding input, as the following table shows:

Register	Value	Direction	Description
SPC_TRIG_ORMASK	40410	read/write	Defines the OR mask for the different trigger sources.
SPC_TMASK_EXT1	4h	Enable multi purpose X0 external trigger input for the OR mask	
SPC_TMASK_EXT2	8h	Enable multi purpose X1 external trigger input for the OR mask	

Edge and level triggers

Rising edge TTL trigger

This mode is for detecting the rising edges of an external TTL signal. The board will trigger on the first rising edge that is detected after starting the board. The next triggerevent will then be detected, if the actual recording/replay has finished and the board is armed and waiting for a trigger again.



Register	Value	Direction	Description
SPC_TRIG_EXT1_MODE	40511	read/write	Sets the trigger mode for multi purpose X0 trigger input.
SPC_TRIG_EXT2_MODE	40512	read/write	Sets the trigger mode for multi purpose X1 trigger input.
SPC_TM_POS	1h	Sets the trigger mode for external TTL trigger to detect positive edges.	

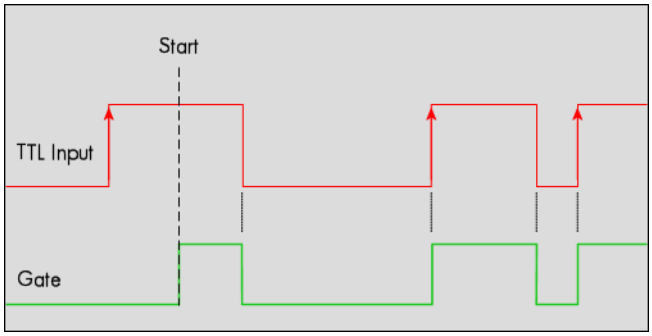
Example on how to set up the board for positive TTL trigger:

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_MODE, SPC_TM_POS);// Set up ext. TTL trigger to detect positive edges
```

HIGH level TTL trigger

This trigger mode will generate an internal gate signal that can be very good used together with a second trigger mode to gate the trigger. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the high level (acting like positive edge trigger) or if the trigger signal is already at high level at the start it will immediately detect a trigger event.

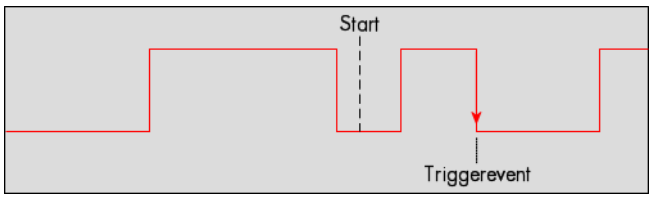
The trigger input is continuously sampled with the selected sample rate. The trigger event will be detected if the trigger input is at TTL high level.



Register	Value	Direction	Description
SPC_TRIG_EXT1_MODE	40511	read/write	Sets the trigger mode for multi purpose X0 trigger input.
SPC_TRIG_EXT2_MODE	40512	read/write	Sets the trigger mode for multi purpose X1 trigger input.
SPC_TM_HIGH	8h	Sets the trigger mode for external TTL trigger to detect HIGH levels.	

Negative TTL trigger

This mode is for detecting the falling edges of an external TTL signal. The board will trigger on the first falling edge that is detected after starting the board. The next triggerevent will then be detected, if the actual recording/replay has finished and the board is armed and waiting for a trigger again.

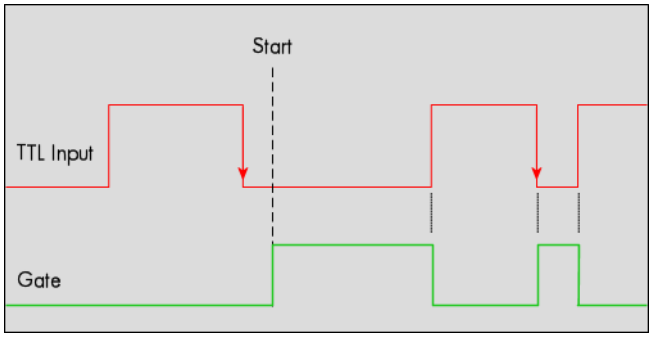


Register	Value	Direction	Description
SPC_TRIG_EXT1_MODE	40511	read/write	Sets the trigger mode for multi purpose X0 trigger input.
SPC_TRIG_EXT2_MODE	40512	read/write	Sets the trigger mode for multi purpose X1 trigger input.
SPC_TM_NEG	2h	Sets the trigger mode for external TTL trigger to detect negative edges.	

LOW level TTL trigger

This trigger mode will generate an internal gate signal that can be very good used together with a second trigger mode to gate the trigger. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the low level (acting like negative edge trigger) or if the trigger signal is already at low level at the start it will immediately detect a trigger event.

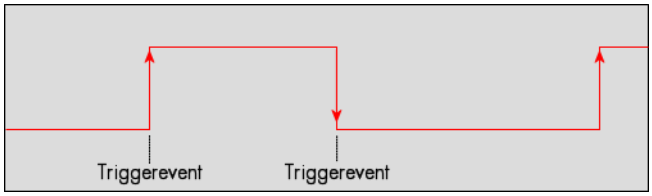
The trigger input is continuously sampled with the selected sample rate. The trigger event will be detected if the trigger input is at TTL low level.



Register	Value	Direction	Description
SPC_TRIG_EXT1_MODE	40511	read/write	Sets the trigger mode for multi purpose X0 trigger input.
SPC_TRIG_EXT2_MODE	40512	read/write	Sets the trigger mode for multi purpose X1 trigger input.
SPC_TM_LOW	10h	Sets the trigger mode for external TTL trigger to detect LOW levels.	

Positive and negative TTL trigger (both edges)

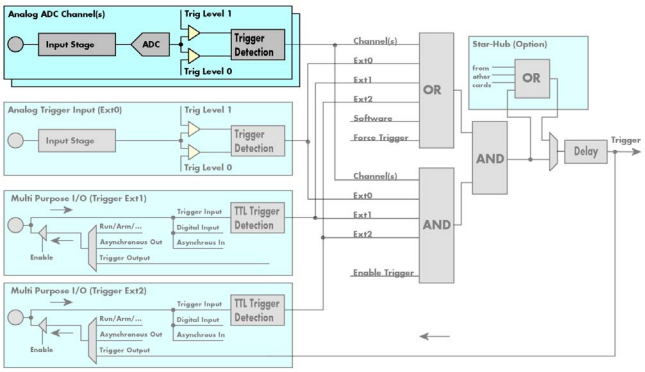
This mode is for detecting the rising and falling edges of an external TTL signal. The board will trigger on the first rising or falling edge that is detected after starting the board. The next trigger-event will then be detected, if the actual recording/replay has finished and the board is armed and waiting for a trigger again.



Register	Value	Direction	Description
SPC_TRIG_EXT1_MODE	40511	read/write	Sets the trigger mode for multi purpose X0 trigger input.
SPC_TRIG_EXT2_MODE	40512	read/write	Sets the trigger mode for multi purpose X1 trigger input.
SPC_TM_BOTH	4h	Sets the trigger mode for external TTL trigger to detect positive and negative edges.	

Channel Trigger

Overview of the channel trigger registers



The channel trigger modes are the most common modes, compared to external equipment like oscilloscopes. The huge variety of different channel trigger modes enable you to observe nearly any part of the analog signal. This chapter is about to explain the different modes in detail. To enable the channel trigger, you have to set the channel triggermode register accordingly. Therefore you have to choose, if you either want only one channel to be the trigger source, or if you want to combine two or more channels to a logical OR or a logical AND trigger.

For all channel trigger modes, the OR mask must contain the corresponding input channels (channel 0 taken as example here):.

Register	Value	Direction	Description
SPC_TRIG_CH_ORMASK0	40460	read/write	Defines the OR mask for the channel trigger sources.
SPC_TMASK0_CH0	1h	Enables channel0 input for the channel OR mask	

The following table shows the according registers for the two general channel trigger modes. It lists the maximum of the available channel mode registers for your card's series. So it can be that you have less channels installed on your specific card and therefore have less valid channel mode registers. If you try to set a channel, that is not installed on your specific card, a error message will be returned.

Register	Value	Direction	Description
SPC_TRIG_CH_AVAILMODES	40600	read	Bitmask, in which all bits of the below mentioned modes for the channel trigger are set, if available.
SPC_TRIG_CH0_MODE	40610	read/write	Sets the trigger mode for channel 0. Channel 0 must be enabled in the channel OR/AND mask.
SPC_TRIG_CH1_MODE	40611	read/write	Sets the trigger mode for channel 1. Channel 1 must be enabled in the channel OR/AND mask.
SPC_TM_NONE	00000000h	Channel is not used for trigger detection. This is as with the trigger masks another possibility for disabling channels.	
SPC_TM_POS	00000001h	Enables the trigger detection for positive edges	
SPC_TM_NEG	00000002h	Enables the trigger detection for negative edges	
SPC_TM_BOTH	00000004h	Enables the trigger detection for positive and negative edges	
SPC_TM_POS | SPC_TM_REARM	01000001h	Trigger detection for positive edges on lebel 0. Trigger is armed when crossing level 1 to avoid false trigger on noise	
SPC_TM_NEG | SPC_TM_REARM	01000002h	Trigger detection for negative edges on lebel 1. Trigger is armed when crossing level 0 to avoid false trigger on noise	
SPC_TM_LOW	00000010h	Enables the trigger detection for LOW levels	
SPC_TM_HIGH	00000008h	Enables the trigger detection for HIGH levels	
SPC_TM_WINENTER	00000020h	Enables the window trigger for entering signals	
SPC_TM_WINLEAVE	00000040h	Enables the window trigger for leaving signals	
SPC_TM_INWIN	00000080h	Enables the window trigger for inner signals	
SPC_TM_OUTSIDEWIN	00000100h	Enables the window trigger for outer signals	

If you want to set up a two channel board to detect only a positive edge on channel 0, you would have to setup the board like the following example. Both of the examples either for the single trigger source and the OR trigger mode do not include the necessary settings for the trigger levels. These settings are detailed described in the following paragraphs.

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_NONE);     // disable software trigger
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH_ORMASK0, SPC_TMASK0_CH0); // Enable channel 0 in the OR mask
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH0_MODE, SPC_TM_POS );     // Set triggermode of channel 0 to positive edge
```

If you want to set up a two channel board to detect a trigger event on either a positive edge on channel 0 or a negative edge on channel 1 you would have to set up your board as the following example shows.

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_NONE);   // disable software trigger
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH_ORMASK0, SPC_TMASK0_CH0 | SPC_TMASK0_CH1);   // Enable channel 0 + 1
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH0_MODE, SPC_TM_POS );    // Set triggermode of channel 0 to positive edge
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH1_MODE, SPC_TM_NEG );    // Set triggermode of channel 1 to negative edge
```

Channel trigger level

All of the channel trigger modes listed above require at least one trigger level to be set (except SPC_TM_NONE of course). Some modes like the window triggers require even two levels (upper and lower level) to be set.

After the data has been sampled, the upper N data bits are compared with the N bits of the trigger levels. The following table shows the level registers and the possible values they can be set to for your specific card.

As the trigger levels are compared to the digitized data, the trigger levels depend on the channels input range. For every input range available to your board there is a corresponding range of trigger levels. On the different input ranges the possible stepsize for the trigger levels differs as well as the maximum and minimum values. The table further below gives you the absolute trigger levels for your specific card series.

10 bit resolution for the trigger levels:

Register	Value	Direction	Description	Range
SPC_TRIG_CH0_LEVEL0	42200	read/write	Trigger level 0 channel 0: main trigger level / upper level if 2 levels used	-511 to +511
SPC_TRIG_CH1_LEVEL0	42201	read/write	Trigger level 0 channel 1: main trigger level / upper level if 2 levels used	-511 to +511
SPC_TRIG_CH0_LEVEL1	42300	read/write	Trigger level 1 channel 0: auxiliary trigger level / lower level if 2 levels used	-511 to +511
SPC_TRIG_CH1_LEVEL1	42301	read/write	Trigger level 1 channel 1: auxiliary trigger level / lower level if 2 levels used	-511 to +511

Trigger level representation depending on selected input range

	Input ranges						
Triggerlevel	**±200 mV**	**±500 mV**	**±1 V**	**±2 V**	**±2.5 V**	**±5 V**	**±10 V**
Path 0 (Buffered)	**x**	**x**	**x**	**x**	**n.a.**	**x**	**x**
Path 1 (HF, 50 Ohms)	**n.a.**	**x**	**x**	**n.a.**	**x**	**x**	**n.a.**
511	+199.6 mV	+499.0 mV	+998.0 mV	+1.996 V	+2.495 V	+4.99 V	+9.98 V
510	+199.2 mV	+498.0 mV	+996.0 mV	+1.992 V	+2.490 V	+4.98 V	+9.96 V
…							
256	+100.0 mV	+250.0 mV	+500.0 mV	+1.00 V	+1.25 V	+2.50 V	+5.00 V
…							
2	+0.8 mV	+2.0 mV	+4.0 mV	+7.8 mV	+9.8 mV	+19.6 mV	+39.0 mV
1	+0.4 mV	+1.0 mV	+2.0 mV	+3.9 mV	+4.9 mV	+9.8 mV	+19.5 mV
0	0 V	0 V	0 V	0 V	0V	0 V	0 V
-1	-0.4 mV	-1.0 mV	-2.0 mV	-3.9 mV	-4.9 mV	-9.8 mV	-19.5 mV
-2	-0.8 mV	-2.0 mV	-4.0 mV	-7.8 mV	-9.8 mV	-19.6 mV	-39.0 mV
…							
-256	-100.0 mV	-250.0 mV	-500.0 mV	-1.00 V	-1.25 V	-2.50 V	-5.00 V
…							
-510	-199.2 mV	-498.0 mV	-996.0 mV	-1.992 V	-2.490 V	-4.98 V	-9.96 V
-511	-199.6 mV	-499.0 mV	-998.0 mV	-1.996 V	-2.495 V	-4.99 V	-9.98 V
Step size	**0.4 mV**	**1.0 mV**	**2.0 mV**	**3.9 mV**	**4.9 mV**	**9.8 mV**	**19.5 mV**

The following example shows, how to set up a one channel board to trigger on channel 0 with rising edge. It is assumed, that the input range of channel 0 is set to the the ±200 mV range. The decimal value for SPC_TRIG_CH0_LEVEL0 corresponds then with 16.0 mV, which is the resulting trigger level.

```
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_NONE);     // disable default software trigger
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH0_MODE,   SPC_TM_POS);     // Setting up channel trig (rising edge)
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH0_LEVEL0, 40);             // Sets triggerlevel to 16.0 mV
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_CH_ORMASK0, SPC_TMASK0_CH0); // and enable it within the OR mask
```

Reading out the number of possible trigger levels

The Spectrum driver also contains a register that holds the value of the maximum possible different trigger levels considering the above mentioned exclusion of the most negative possible value. This is useful, as new drivers can also be used with older hardware versions, because you can check the trigger resolution during run time. The register is shown in the following table:

Register	Value	Direction	Description
SPC_READTRGLVLCOUNT	2500	r	Contains the number of different possible trigger levels meaning ± of the value.

In case of a board that uses 8 bits for trigger detection the returned value would be 127, as either the zero and 127 positive and negative values are possible.The resulting trigger step width in mV can easily be calculated from the returned value. It is assumed that you know the actually selected input range.

$$\text{Trigger step width} = \frac{\text{Input Range}_{max}}{\text{Number of trigger levels} + 1}$$

To give you an example on how to use this formula we assume, that the ±1.0 V input range is selected and the board uses 8 bits for trigger detection. The result would be 7.81 mV, which is the step width for your type of board within the actually chosen input range.

$$\text{Trigger step width} = \frac{+1000\ \text{mV}}{127 + 1}$$

Detailed description of the channel trigger modes

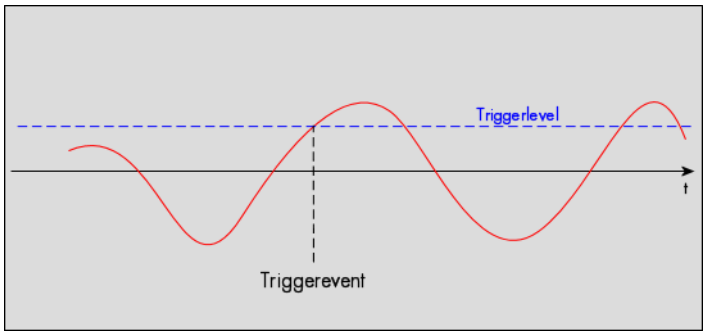
For all channel trigger modes, the OR mask must contain the corresponding input channels (channel 0 taken as example here):

Register	Value	Direction	Description
SPC_TRIG_CH_ORMASK0	40460	read/write	Defines the OR mask for the channel trigger sources.
SPC_TMASK0_CH0	1h	Enables channel0 input for the channel OR mask	

Channel trigger on positive edge

The analog input is continuously sampled with the selected sample rate. If the programmed trigger level is crossed by the channel's signal from lower values to higher values (rising edge) then the trigger event will be detected.

These edge triggered channel trigger modes correspond to the trigger possibilities of usual oscilloscopes.

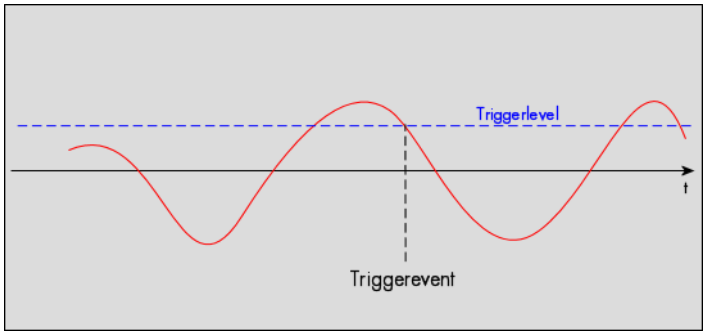


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_POS	1h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the desired trigger level relatively to the channel's input range.	board dependent

Channel trigger on negative edge

The analog input is continuously sampled with the selected sample rate. If the programmed trigger level is crossed by the channel's signal from higher values to lower values (falling edge) then the trigger event will be detected.

These edge triggered channel trigger modes correspond to the trigger possibilities of usual oscilloscopes.

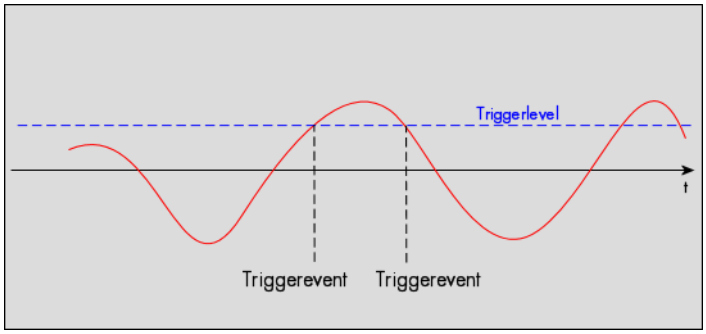


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_NEG	2h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the desired trigger level relatively to the channel's input range.	board dependent

Channel trigger on positive and negative edge

The analog input is continuously sampled with the selected sample rate. If the programmed trigger level is crossed by the channel's signal (either rising or falling edge) the trigger event will be detected.

These edge triggered channel trigger modes correspond to the trigger possibilities of usual oscilloscopes.

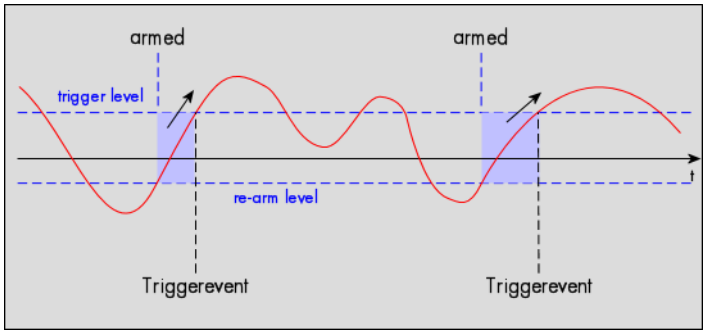


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_BOTH	4h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the desired trigger level relatively to the channel's input range.	board dependent

Channel re-arm trigger on positive edge

The analog input is continuously sampled with the selected sample rate. If the programmed re-arm level is crossed from lower to higher values, the trigger engine is armed and waiting for trigger. If the programmed trigger level is crossed by the channel's signal from lower values to higher values (rising edge) then the trigger event will be detected and the trigger engine will be disarmed. A new trigger event is only detected if the trigger engine is armed again.

The re-arm trigger modes can be used to prevent the board from triggering on wrong edges in noisy signals.

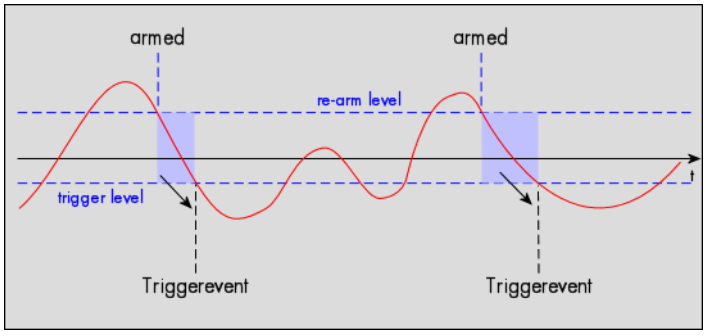


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_POS | SPC_TM_REARM	01000001h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the desired trigger level relatively to the channel's input range.	board dependent
SPC_TRIG_CH0_LEVEL1	42300	read/write	Defines the re-arm level relatively to the channel's input range	board dependent

Channel re-arm trigger on negative edge

The analog input is continuously sampled with the selected sample rate. If the programmed re-arm level is crossed from higher to lower values, the trigger engine is armed and waiting for trigger. If the programmed trigger level is crossed by the channel's signal from higher values to lower values (falling edge) then the trigger event will be detected and the trigger engine will be disarmed. A new trigger event is only detected, if the trigger engine is armed again.

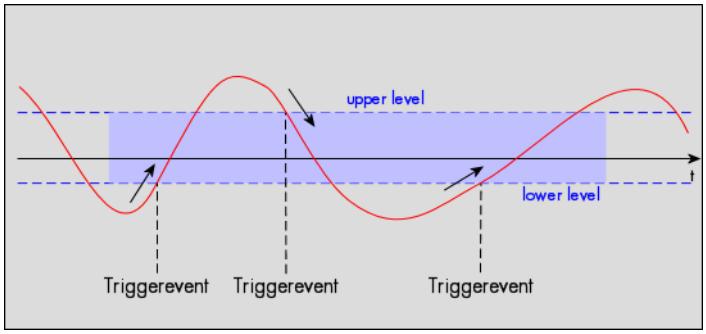
The re-arm trigger modes can be used to prevent the board from triggering on wrong edges in noisy signals.



Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_NEG | SPC_TM_REARM	01000002h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Defines the re-arm level relatively to the channel's input range	board dependent
SPC_TRIG_CH0_LEVEL1	42300	read/write	Set it to the desired trigger level relatively to the channel's input range.	board dependent

Channel window trigger for entering signals

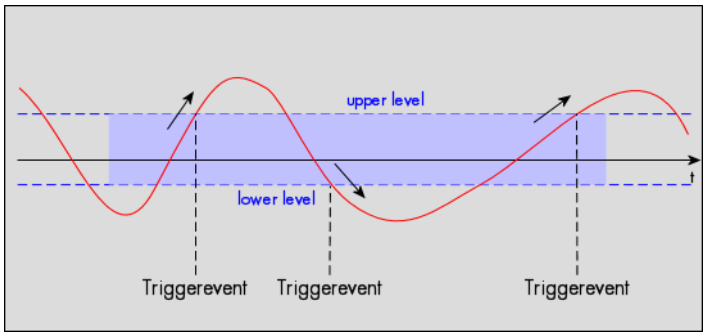
The analog input is continuously sampled with the selected sample rate. The upper and the lower level define a window. Every time the signal enters the window from the outside, a trigger event will be detected.



Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_WINENTER	00000020h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the upper trigger level relatively to the channel's input range.	board dependent
SPC_TRIG_CH0_LEVEL1	42300	read/write	Set it to the lower trigger level relatively to the channel's input range.	board dependent

Channel window trigger for leaving signals

The analog input is continuously sampled with the selected sample rate. The upper and the lower level define a window. Every time the signal leaves the window from the inside, a trigger event will be detected.

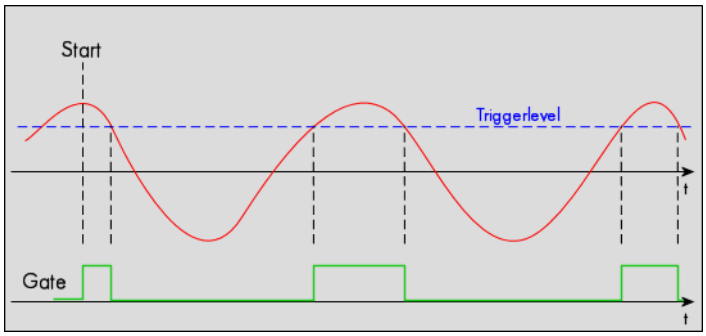


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_WINLEAVE	00000040h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the upper trigger level relatively to the channel's input range.	board dependent
SPC_TRIG_CH0_LEVEL1	42300	read/write	Set it to the lower trigger level relatively to the channel's input range.	board dependent

High level trigger

This trigger mode will generate an internal gate signal that can be useful for masking a second trigger event generated by a different mode. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the high level (acting like positive edge trigger) or if the analog signal is already above the programmed level at the start it will immediately detect a trigger event.

The channel is continuously sampled with the selected sample rate. The trigger event will be detected if the analog signal is above the programmed trigger level.

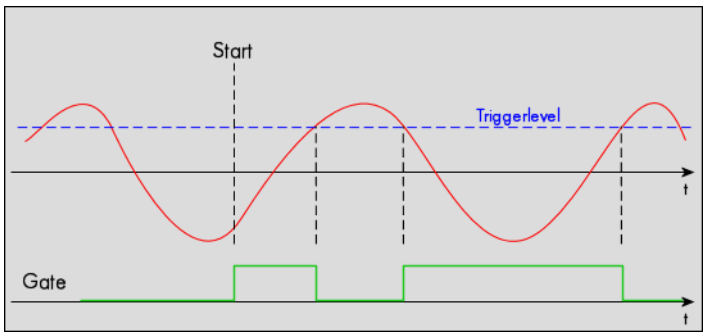


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_HIGH	00000008h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the upper trigger level relatively to the channel's input range.	board dependent

Low level trigger

This trigger mode will generate an internal gate signal that can be useful for masking a second trigger event generated by a different mode. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the low level (acting like negative edge trigger) or if the signal is already above the programmed level at the start it will immediately detect a trigger event.

The channel is continuously sampled with the selected sample rate. The trigger event will be detected if the analog signal is below the programmed trigger level.

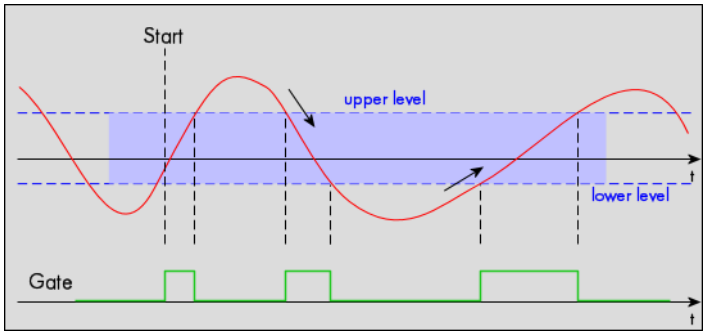


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_LOW	00000010h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the upper trigger level relatively to the channel's input range.	board dependent

In window trigger

This trigger mode will generate an internal gate signal that can be useful for masking a second trigger event generated by a different mode. If using this mode as a single trigger source the card will detect a trigger event at the time when entering the window defined by the two trigger levels (acting like window enter trigger) or if the signal is already inside the programmed window at the start it will immediately detect a trigger event.

The channel is continuously sampled with the selected sample rate. The trigger event will be detected if the analog signal is inside the programmed trigger window.

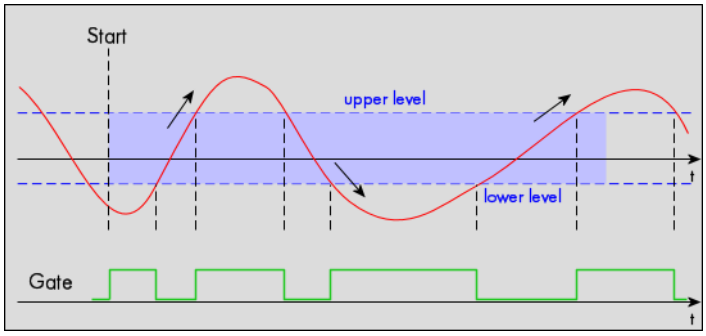


Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_INWIN	00000080h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the upper trigger level relatively to the channel's input range.	board dependent
SPC_TRIG_CH0_LEVEL1	42300	read/write	Set it to the lower trigger level relatively to the channel's input range.	board dependent

Outside window trigger

This trigger mode will generate an internal gate signal that can be useful for masking a second trigger event generated by a different mode. If using this mode as a single trigger source the card will detect a trigger event at the time when leaving the window defined by the two trigger levels (acting like leaving window trigger) or if the signal is already outside the programmed window at the start it will immediately detect a trigger event.

The channel is continuously sampled with the selected sample rate. The trigger event will be detected if the analog signal is outside the programmed trigger window.



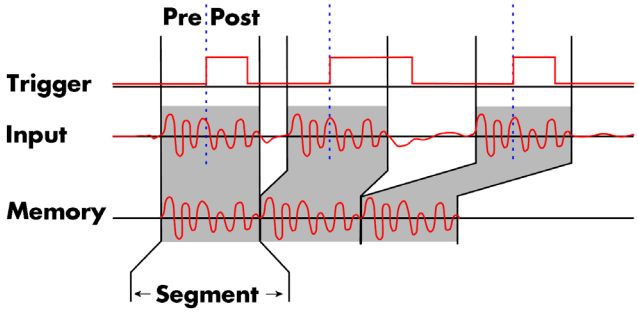
Register	Value	Direction	set to	Value
SPC_TRIG_CH0_MODE	40610	read/write	SPC_TM_OUTSIDEWIN	00000100h
SPC_TRIG_CH0_LEVEL0	42200	read/write	Set it to the upper trigger level relatively to the channel's input range.	board dependent
SPC_TRIG_CH0_LEVEL1	42300	read/write	Set it to the lower trigger level relatively to the channel's input range.	board dependent

Mode Multiple Recording

The Multiple Recording mode allows the acquisition of data blocks with multiple trigger events without restarting the hardware.

The on-board memory will be divided into several segments of the same size. Each segment will be filled with data when a trigger event occurs (acquisition mode).

As this mode is totally controlled in hardware there is a very small re-arm time from end of one segment until the trigger detection is enabled again. You'll find that re-arm time in the technical data section of this manual.



The following table shows the register for defining the structure of the segments to be recorded with each trigger event.

Register	Value	Direction	Description
SPC_POSTTRIGGER	10100	read/write	Acquisition only: defines the number of samples to be recorded per channel after the trigger event.
SPC_SEGMENTSIZE	10010	read/write	Size of one Multiple Recording segment: the total number of samples to be recorded per channel after detection of one trigger event including the time recorded before the trigger (pre trigger).

Each segment in acquisition mode can consist of pretrigger and/or posttrigger samples. The user always has to set the total segment size and the posttrigger, while the pretrigger is calculated within the driver with the formula: [pretrigger] = [segment size] - [posttrigger].

When using Multiple Recording the maximum pretrigger is limited depending on the number of active channels. When the calculated value exceeds that limit, the driver will return the error ERR_PRETRIGGERLEN. Please have a look at the table further below to see the maximum pretrigger length that is possible.

Recording modes

Standard Mode

With every detected trigger event one data block is filled with data. The length of one multiple recording segment is set by the value of the segment size register SPC_SEGMENTSIZE. The total amount of samples to be recorded is defined by the memsize register.
Memsize must be set to a a multiple of the segment size. The table below shows the register for enabling Multiple Recording. For detailed information on how to setup and start the standard acquisition mode please refer to the according chapter earlier in this manual.

Register	Value	Direction	Description
SPC_CARDMODE	9500	read/write	Defines the used operating mode
SPC_REC_STD_MULTI	2	Enables Multiple Recording for standard acquisition.	

The total number of samples to be recorded to the on-board memory in Standard Mode is defined by the SPC_MEMSIZE register.

Register	Value	Direction	Description
SPC_MEMSIZE	10000	read/write	Defines the total number of samples to be recorded per channel.

FIFO Mode

The Multiple Recording in FIFO Mode is similar to the Multiple Recording in Standard Mode. In contrast to the standard mode it is not necessary to program the number of samples to be recorded. The acquisition is running until the user stops it. The data is read block by block by the driver as described under FIFO single mode example earlier in this manual. These blocks are online available for further data processing by the user program. This mode significantly reduces the amount of data to be transferred on the PCI bus as gaps of no interest do not have to be transferred. This enables you to use faster sample rates than you would be able to in FIFO mode without Multiple Recording.
The advantage of Multiple Recording in FIFO mode is that you can stream data online to the host system. You can make real-time data processing or store a huge amount of data to the hard disk. The table below shows the dedicated register for enabling Multiple Recording. For detailed information how to setup and start the board in FIFO mode please refer to the according chapter earlier in this manual.

Register	Value	Direction	Description
SPC_CARDMODE	9500	read/write	Defines the used operating mode
SPC_REC_FIFO_MULTI	32	Enables Multiple Recording for FIFO acquisition.	

The number of segments to be recorded must be set separately with the register shown in the following table:

Register	Value	Direction	Description
SPC_LOOPS	10020	read/write	Defines the number of segments to be recorded
0	Recording will be infinite until the user stops it.		
1 ... [4G - 1]	Defines the total segments to be recorded.		

Limits of pre trigger, post trigger, memory size

The maximum memory size parameter is only limited by the number of activated channels and by the amount of installed memory. Please keep in mind that each sample needs 2 bytes of memory to be stored. Minimum memory size as well as minimum and maximum post trigger limits are independent of the activated channels or the installed memory.

Due to the internal organization of the card memory there is a certain stepsize when setting these values that has to be taken into account. The following table gives you an overview of all limits concerning pre trigger, post trigger, memory size, segment size and loops. The table shows all values in relation to the installed memory size in samples. If more memory is installed the maximum memory size figures will increase according to the complete installed memory

Activated Channels	Used Mode	Memory size SPC_MEMSIZE			Pre trigger SPC_PRETRIGGER			Post trigger SPC_POSTTRIGGER			Segment size SPC_SEGMENTSIZE			Loops SPC_LOOPS		
		Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step
1 channel	Standard Single	16	Mem	8	defined by post trigger			8	8G - 8	8	not used			not used		
	Standard Multi/ABA	16	Mem	8	8	8k	8	8	Mem/2	8	16	Mem/2	8	not used		
	FIFO Single	not used			8	8k	8	not used			16	8G - 8	8	0 (∞)	4G - 1	1
	FIFO Multi/ABA	not used			8	8k	8	8	8G - 8	8	16	pre+post	8	0 (∞)	4G - 1	1
2 channels	Standard Single	16	Mem/2	8	defined by post trigger			8	8G - 8	8	not used			not used		
	Standard Multi/ABA	16	Mem/2	8	8	4k	8	8	Mem/4	8	16	Mem/4	8	not used		
	FIFO Single	not used			8	4k	8	not used			16	8G - 8	8	0 (∞)	4G - 1	1
	FIFO Multi/ABA	not used			8	4k	8	8	8G - 8	8	16	pre+post	8	0 (∞)	4G - 1	1

All figures listed here are given in samples. An entry of [32G - 8] means [32 GSamples - 8] = 34,359,738,360 samples.

The given memory and memory / divider figures depend on the installed on-board memory as listed below:

	Installed Memory				
	128 MSample	**256 MSample**	**512 MSample**	**1 GSample**	**2 GSample**
Mem	128 MSample	256 MSample	512 MSample	1 GSample	2 GSample
Mem / 2	64 MSample	128 MSample	256 MSample	512 MSample	1 GSample
Mem / 4	32 MSample	64 MSample	128 MSample	256 MSample	512 MSample

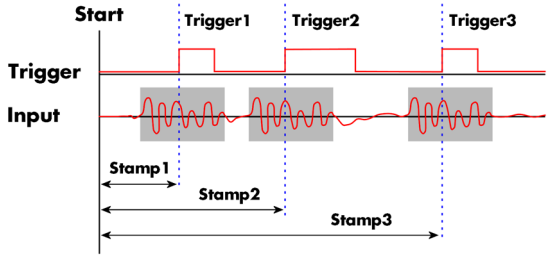
Please keep in mind that this table shows all values at once. Only the absolute maximum and minimum values are shown. There might be additional limitations. Which of these values is programmed depends on the used mode. Please read the detailed documentation of the mode.

Multiple Recording and Timestamps

Multiple Recording is well matching with the timestamp option. If timestamp recording is activated each trigger event and therefore each Multiple Recording segment will get timestamped as shown in the drawing on the right.

Please keep in mind that the trigger events are timestamped, not the beginning of the acquisition. The first sample that is available is at the time position of [Timestamp - Pretrigger].

The programming details of the timestamp option is explained in an extra chapter.



Trigger Modes

When using Multiple Recording all of the card's trigger modes can be used except the software trigger. For detailed information on the available trigger modes, please take a look at the relating chapter earlier in this manual.

Trigger Counter

The number of acquired trigger events in Multiple Recording mode is counted in hardware and can be read out while the acquisition is running or after the acquisition has finished. The trigger events are counted both in standard mode as well as in FIFO mode.

Register	Value	Direction	Description
SPC_TRIGGERCOUNTER	200905	read	Returns the number of trigger events that has been acquired since the acquisition start. The internal trigger counter has 48 bits. It is therefore necessary to read out the trigger counter value with 64 bit access or 2 x 32 bit access if the number of trigger events exceed the 32 bit range.

The trigger counter feature needs at least driver version V2.17 and firmware version V20 (M2i series), V10 (M3i series), V6 (M4i/M4x series) or V1 (M2p series). Please update the driver and the card firmware to these versions to use this feature. Trying to use this feature without the proper firmware version will issue a driver error.

Using the trigger counter information one can determine how many Multiple Recording segments have been acquired and can perform a memory flush by issuing Force trigger commands to read out all data. This is helpful if the number of trigger events is not known at the start of the acquisition. In that case one will do the following steps:

- Program the maximum number of segments that one expects or use the FIFO mode with unlimited segments
- Set a timeout to be sure that there are no more trigger events acquired. Alternatively one can manually proceed as soon as it is clear from the application that all trigger events have been acquired
- Read out the number of acquired trigger segments
- Issue a number of Force Trigger commands to fill the complete memory (standard mode) or to transfer the last FIFO block that contains valid data segments
- Use the trigger counter value to split the acquired data into valid data with a real trigger event and invalid data with a force trigger event.

Programming examples

The following example shows how to set up the card for Multiple Recording in standard mode.

```
spcm_dwSetParam_i32 (hDrv, SPC_CARDMODE, SPC_REC_STD_MULTI);  // Enables Standard Multiple Recording

spcm_dwSetParam_i64 (hDrv, SPC_SEGMENTSIZE,   1024);          // Set the segment size to 1024 samples
spcm_dwSetParam_i64 (hDrv, SPC_POSTTRIGGER,    768);        // Set the posttrigger to 768 samples and therefore
                                                            // the pretrigger will be 256 samples
spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE,     4096);        // Set the total memsize for recording to 4096 samples
                                                            // so that actually four segments will be recorded

spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_MODE,  SPC_TM_POS);  // Set triggermode to ext. TTL mode (rising edge)
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_EXT0);  // and enable it within the trigger OR-mask
```

The following example shows how to set up the card for Multiple Recording in FIFO mode.

```
spcm_dwSetParam_i32 (hDrv, SPC_CARDMODE, SPC_REC_FIFO_MULTI); // Enables FIFO Multiple Recording

spcm_dwSetParam_i64 (hDrv, SPC_SEGMENTSIZE,   2048);          // Set the segment size to 2048 samples
spcm_dwSetParam_i64 (hDrv, SPC_POSTTRIGGER,   1920);        // Set the posttrigger to 1920 samples and therefore
                                                            // the pretrigger will be 128 samples
spcm_dwSetParam_i64 (hDrv, SPC_LOOPS           256);          // 256 segments will be recorded

spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_MODE,  SPC_TM_NEG);  // Set triggermode to ext. TTL mode (falling edge)
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_EXT0);  // and enable it within the trigger OR-mask
```

Timestamps

General information

The timestamp function is used to record trigger events relative to the beginning of the measurement, relative to a fixed time-zero point or synchronized to an external reset clock. The reset clock can come from a radio clock a GPS signal or from any other external machine.

The timestamp is internally realized as a very wide counter that is running with the currently used sampling rate. The counter is reset either by explicit software command or depending on the mode by the start of the card. On receiving the trigger event the current counter value is stored in an extra FIFO memory.

This function is designed as an enhancement to the Multiple Recording mode and is also used together with the ABA mode option but can also be used without these options with plain single acquisitions.

Each recorded timestamp consists of the number of samples that has been counted since the last counter reset has been done. The actual time in relation to the reset command can be easily calculated by the formula on the right. Please note that the timestamp recalculation depends on the currently used sampling rate. Please have a look at the clock chapter to see how to read out the sampling rate.

$$t = \frac{\text{Timestamp}}{\text{Sampling rate}}$$

If you want to know the time between two timestamps, you can simply calculate this by the formula on the right.

$$\Delta t = \frac{\text{Timestamp}_{n+1} - \text{Timestamp}_{n}}{\text{Sampling rate}}$$

The following registers can be used for the timestamp option:

Register	Value	Direction	Description
SPC_TIMESTAMP_STARTTIME	47030	read/write	Return the reset time when using reference clock mode. Hours are placed in bit 16 to 23, minutes are placed in bit 8 to 15, seconds are placed in bit 0 to 7
SPC_TIMESTAMP_STARTDATE	47031	read/write	Return the reset date when using reference clock mode. The year is placed in bit 16 to 31, the month is placed in bit 8 to 15 and the day of month is placed in bit 0 to 7
SPC_TIMESTAMP_TIMEOUT	47045	read/write	Set's a timeout in milli seconds for waiting of an reference clock edge
SPC_TIMESTAMP_AVAILMODES	47001	read	Returns all available modes as a bitmap. Modes are listed below
SPC_TIMESTAMP_CMD	47000	read/write	Programs a timestamp mode and performs commands as listed below
SPC_TSMODE_DISABLE	0	Timestamp is disabled.	
SPC_TS_RESET	1h	The counters are reset and the local PC time is stored for read out by SPC_TIMESTAMP_STARTTIME and SPC_TIMESTAMP_STARTDATE registers.	
SPC_TSMODE_STANDARD	2h	Standard mode, counter is reset by explicit reset command.	
SPC_TSMODE_STARTRESET	4h	Counter is reset on every card start, all timestamps are in relation to card start.	
SPC_TSCNT_INTERNAL	100h	Counter is running with complete width on sampling clock	
SPC_TSCNT_REFCLOCKPOS	200h	Counter is split, upper part is running with external reference clock positive edge, lower part is running with sampling clock	
SPC_TSCNT_REFCLOCKNEG	400h	Counter is split, upper part is running with external reference clock negative edge, lower part is running with sampling clock	
SPC_TSXIOINC_ENABLE	2000h	Enables the trigger synchronous acquisition of the two 10 bit incremental counters with every stored timestamp in the upper 20 bit of the timestamp data.	
SPC_TSXIOACQ_ENABLE	1000h	Enables the trigger synchronous acquisition of the BaseXIO inputs with every stored timestamp in the upper byte.	
SPC_TSXIOACQ_DISABLE	0	The timestamp is filled up with leading zeros as a sign extension for positive values.	
SPC_TSFEAT_NONE	0	No additional timestamp is created. The total number of stamps is only trigger related.	
SPC_TSFEAT_STORE1STABA	10000h	Enables the creation of one additional timestamp for the first A area sample when using the optional ABA (dual-time-base) mode.	

Writes to the SPC_TS_RESET register can only have an effect on the counters, if the cards clock generation is already active. This is the case when the card either has already done an acquisition after the last reset or if the clock setup has already been actively transferred to the card by issuing the M2CMD_CARD_WRITESETUP command.

Example for setting timestamp mode:

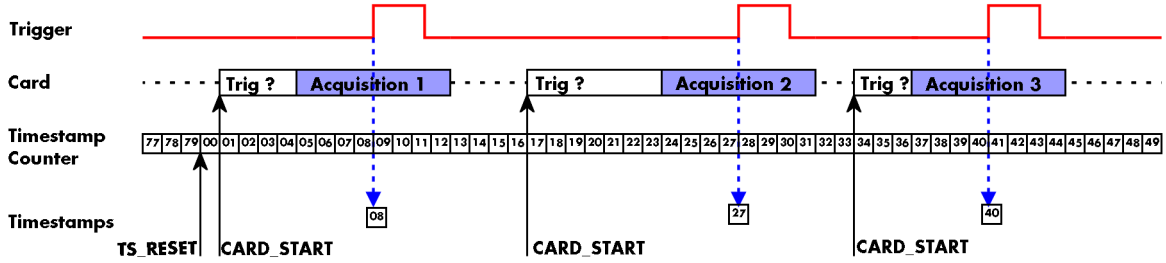
The timestamp mode consists of one of the mode constants, one of the counter and one of the feature constants:

```
// setting timestamp mode to standard using internal clocking
spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, SPC_TSMODE_STANDARD | SPC_TSCNT_INTERNAL | SPC_TSFEAT_NONE);

// setting timestamp mode to start reset mode using internal clocking
spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, SPC_TSMODE_STARTRESET | SPC_TSCNT_INTERNAL | SPC_TSFEAT_NONE);

// setting timestamp mode to standard using external reference clock with positive edge
spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, SPC_TSMODE_STANDARD | SPC_TSCNT_REFCLOCKPOS | SPC_TSFEAT_NONE);
```

Timestamp modes

Standard mode

In standard mode the timestamp counter is set to zero once by writing the TS_RESET commando to the command register. After that command the counter counts continuously independent of start and stop of acquisition. The timestamps of all recorded trigger events are referenced to this common zero time. With this mode you can calculate the exact time difference between different recordings and also within one acquisition (if using Multiple Recording or Gated Sampling).



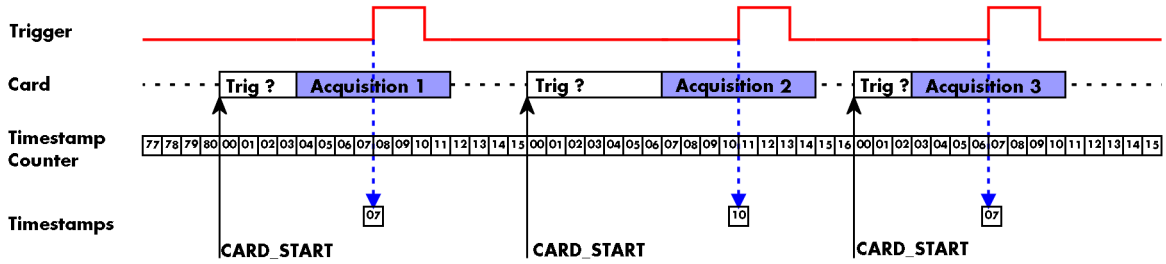
The following table shows the valid values that can be written to the timestamp command register for this mode:

Register	Value	Direction	Description
SPC_TIMESTAMP_CMD	47000	read/write	Programs a timestamp mode and performs commands as listed below
SPC_TSMODE_DISABLE	0	Timestamp is disabled.	
SPC_TS_RESET	1h	The timestamp counter is set to zero	
SPC_TSMODE_STANDARD	2h	Standard mode, counter is reset by explicit reset command.	
SPC_TSCNT_INTERNAL	100h	Counter is running with complete width on sampling clock	

Please keep in mind that this mode only work sufficiently as long as you don't change the sampling rate between two acquisitions that you want to compare.

StartReset mode

In StartReset mode the timestamp counter is set to zero on every start of the card. After starting the card the counter counts continuously. The timestamps of one recording are referenced to the start of the recording. This mode is very useful for Multiple Recording and Gated Sampling (see according chapters for detailed information on these two optional modes)



The following table shows the valid values that can be written to the timestamp command register.

Register	Value	Direction	Description
SPC_TIMESTAMP_CMD	47000	read/write	Programs a timestamp mode and performs commands as listed below
SPC_TSMODE_DISABLE	0	Timestamp is disabled.	
SPC_TSMODE_STARTRESET	4h	Counter is reset on every card start, all timestamps are in relation to card start.	
SPC_TSCNT_INTERNAL	100h	Counter is running with complete width on sampling clock	

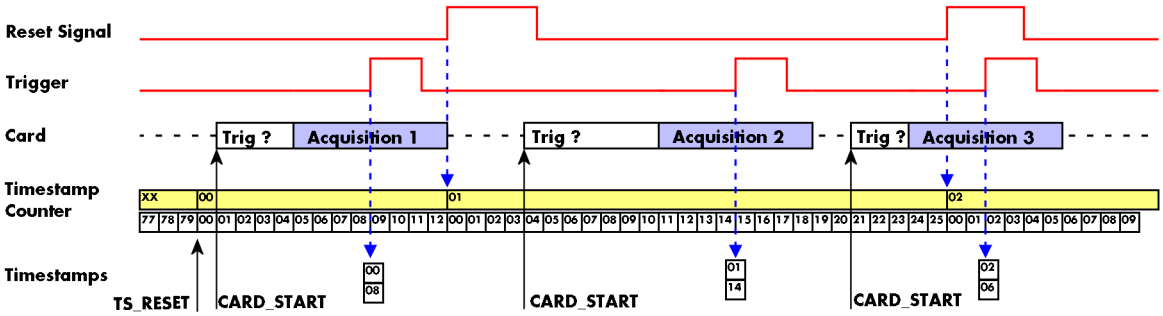
Refclock mode

The counter is split in a HIGH and a LOW part and an additional external signal, that affects both parts of the counter, needs to be fed in externally. The external reference clock signal will reset the LOW part of the counter and increase the HIGH part of the counter. The upper counter will hold the number of the clock edges that have occurred on the external reference clock signal and the lower counter will hold the position within the current reference clock period with the resolution of the sampling rate.

This mode can be used to obtain an absolute time reference when using an external radio clock or a GPS receiver. In that case the higher part is counting the seconds since the last reset and the lower part is counting the position inside the second using the current sampling rate.

Please keep in mind that as this mode uses an additional external signal. If using plain M2i cards the option BaseXIO needs to be installed on the card. Otherwise there is no additional reference clock input available and this mode has no functionality. If using a digitizerNETBOX this additional timestamp reference clock input is available as a standard and no option is needed to use this mode.

The counting is initialized with the timestamp reset command. Both counter parts will then be set to zero.



The following table shows the valid values that can be written to the timestamp command register for this mode:

Register	Value	Direction	Description
SPC_TIMESTAMP_STARTTIME	47030	read/write	Return the reset time when using reference clock mode. Hours are placed in bit 16 to 23, minutes are placed in bit 8 to 15, seconds are placed in bit 0 to 7
SPC_TIMESTAMP_STARTDATE	47031	read/write	Return the reset date when using reference clock mode. The year is placed in bit 16 to 31, the month is placed in bit 8 to 15 and the day of month is placed in bit 0 to 7
SPC_TIMESTAMP_TIMEOUT	47045	read/write	Sets a timeout in milli seconds for waiting for a reference clock edge
SPC_TIMESTAMP_CMD	47000	read/write	Programs a timestamp mode and performs commands as listed below
SPC_TSMODE_DISABLE	0	Timestamp is disabled.	
SPC_TS_RESET	1h	The counters are reset. If reference clock mode is used this command waits for the edge the timeout time.	
SPC_TSMODE_STANDARD	2h	Standard mode, counter is reset by explicit reset command.	
SPC_TSMODE_STARTRESET	4h	Counter is reset on every card start, all timestamps are in relation to card start.	
SPC_TSCNT_REFCLOCKPOS	200h	Counter is split, upper part is running with external reference clock positive edge, lower part is running with sampling clock	
SPC_TSCNT_REFCLOCKNEG	400h	Counter is split, upper part is running with external reference clock negative edge, lower part is running with sampling clock	

To synchronize the external reference clock signal with the PC clock it is possible to perform a timestamp reset command which waits a specified time for the occurrence of the external clock edge. As soon as the clock edge is found the function stores the current PC time and date which can be used to get the absolute time. As the timestamp reference clock can also be used with other clocks that don't need to be synchronized with the PC clock the waiting time can be programmed using the SPC_TIMESTAMP_TIMEOUT register.

Example for initialization of timestamp reference clock and synchronization of a seconds signal with the PC clock:

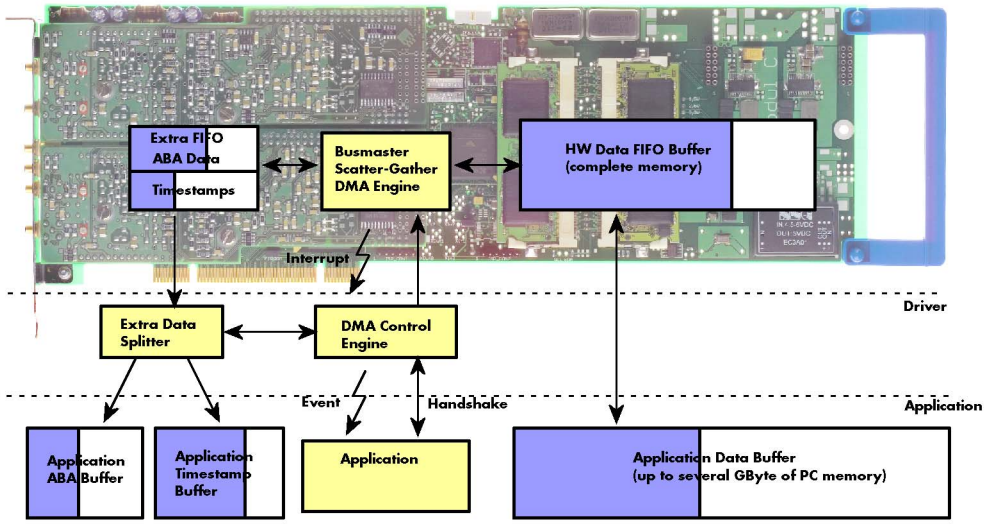
```
spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, SPC_TSMODE_STANDARD | SPC_TSCNT_REFCLOCKPOS);
spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_TIMEOUT, 1500);
if (ERR_TIMEOUT == spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, SPC_TS_RESET))
    printf ("Synchronization with external clock signal failed\n");

// now we read out the stored synchronization clock and date
int32 lSyncDate, lSyncTime;
spcm_dwGetParam_i32 (hDrv, SPC_TIMESTAMP_STARTDATE, &lSyncDate);
spcm_dwGetParam_i32 (hDrv, SPC_TIMESTAMP_STARTTIME, &lSyncTime);

// and print the start date and time information (European format: day.month.year hour:minutes:seconds)
printf ("Start date: %02d.%02d.%04d\n", lSyncDate & 0xff, (lSyncDate >> 8) & 0xff, (lSyncDate >> 16) & 0xffff);
printf ("Start time: %02d:%02d:%02d\n", (lSyncTime >> 16) & 0xff, (lSyncTime >> 8) & 0xff, lSyncTime & 0xff);
```

Reading out the timestamps

General

The timestamps are stored in an extra FIFO that is located in hardware on the card. This extra FIFO can read out timestamps using DMA transfer similar to the DMA transfer of the main sample data DMA transfer. The card has two completely independent busmaster DMA engines in hardware allowing the simultaneous transfer of both timestamp and sample data.

As seen in the picture the extra FIFO is holding ABA and timestamp data as the same time. Nevertheless it is not necessary to care for the shared FIFO as the extra FIFO data is splitted inside the driver in the both data parts.



The only part that is similar for both kinds of data transfer is the handling of the DMA engine. This is similar to the main sample data transfer engine. Therefore additional information can be found in the chapter explaining the main data transfer.

Commands and Status information for extra transfer buffers.

As explained above the data transfer is performed with the same command and status registers like the card control and sample data transfer. It is possible to send commands for card control, data transfer and extra FIFO data transfer at the same time

Register	Value	Direction	Description
SPC_M2CMD	100	write only	Executes a command for the card or data transfer
M2CMD_EXTRA_STARTDMA	100000h	Starts the DMA transfer for an already defined buffer.	
M2CMD_EXTRA_WAITDMA	200000h	Waits until the data transfer has ended or until at least the amount of bytes defined by notify size are available. This wait function also takes the timeout parameter into account.	
M2CMD_EXTRA_STOPDMA	400000h	Stops a running DMA transfer. Data is invalid afterwards.	
M2CMD_EXTRA_POLL	800000h	Polls data without using DMA. As DMA has some overhead and has been implemented for fast data transfer of large amounts of data it is in some cases more simple to poll for available data. Please see the detailed examples for this mode. It is not possible to mix DMA and polling mode.	

The extra FIFO data transfer can generate one of the following status information:.

Register	Value	Direction	Description
SPC_M2STATUS	110	read only	Reads out the current status information
M2STAT_EXTRA_BLOCKREADY	1000h	The next data block as defined in the notify size is available. It is at least the amount of data available but it also can be more data.	
M2STAT_EXTRA_END	2000h	The data transfer has completed. This status information will only occur if the notify size is set to zero.	
M2STAT_EXTRA_OVERRUN	4000h	The data transfer had on overrun (acquisition) or underrun (replay) while doing FIFO transfer.	
M2STAT_EXTRA_ERROR	8000h	An internal error occurred while doing data transfer.	

Data Transfer using DMA

Data transfer consists of two parts: the buffer definition and the commands/status information that controls the transfer itself. Extra data transfer shares the command and status register with the card control, data transfer commands and status information.

The DMA based data transfer mode is activated as soon as the M2CMD_EXTRA_STARTDMA is given. Please see next chapter to see how the polling mode works.

Definition of the transfer buffer

Before any data transfer can start it is necessary to define the transfer buffer with all its details. The definition of the buffer is done with the spcm_dwDefTransfer function as explained in an earlier chapter. The following example will show the definition of a transfer buffer for timestamp data, definition for ABA data is similar:

```
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_CARDTOPC, 0, pvBuffer, 0, lLenOfBufferInBytes);
```

In this example the notify size is set to zero, meaning that we don't want to be notified until all extra data has been transferred. Please have a look at the sample data transfer in an earlier chapter to see more details on the notify size.

Please note that extra data transfer is only possible from card to PC and there's no programmable offset available for this transfer.

Buffer handling

A data buffer handshake is implemented in the driver which allows to run the card in different data transfer modes. The software transfer buffer is handled as one large buffer for each kind of data (timestamp and ABA) which is on the one side controlled by the driver and filled automatically by busmaster DMA from the hardware extra FIFO buffer and on the other hand it is handled by the user who set's parts of this software buffer available for the driver for further transfer. The handshake is fulfilled with the following 3 software registers:

Register	Value	Direction	Description
SPC_ABA_AVAIL_USER_LEN	210	read	This register contains the currently available number of bytes that are filled with newly transferred slow ABA data. The user can now use this ABA data for own purposes, copy it, write it to disk or start calculations with this data.
SPC_ABA_AVAIL_USER_POS	211	read	The register holds the current byte index position where the available ABA bytes start. The register is just intended to help you and to avoid own position calculation
SPC_ABA_AVAIL_CARD_LEN	212	write	After finishing the job with the new available ABA data the user needs to tell the driver that this amount of bytes is again free for new data to be transferred.
SPC_TS_AVAIL_USER_LEN	220	read	This register contains the currently available number of bytes that are filled with newly transferred timestamp data. The user can now use these timestamps for own purposes, copy it, write it to disk or start calculations with the timestamps.
SPC_TS_AVAIL_USER_POS	221	read	The register holds the current byte index position where the available timestamp bytes start. The register is just intended to help you and to avoid own position calculation
SPC_TS_AVAIL_CARD_LEN	222	write	After finishing the job with the new available timestamp data the user needs to tell the driver that this amount of bytes is again free for new data to be transferred.

Directly after start of transfer the SPC_XXX_AVAIL_USER_LEN is every time zero as no data is available for the user and the SPC_XXX_AVAIL_CARD_LEN is every time identical to the length of the defined buffer as the complete buffer is available for the card for transfer.

The counter that is holding the user buffer available bytes (SPC_XXX_AVAIL_USER_LEN) is sticking to the defined notify size at the DefTransfer call. Even when less bytes already have been transferred you won't get notice of it if the notify size is programmed to a higher value.

Remarks

- The transfer between hardware FIFO buffer and application buffer is done with scatter-gather DMA using a busmaster DMA controller located on the card. Even if the PC is busy with other jobs data is still transferred until the application buffer is completely used.
- As shown in the drawing above the DMA control will announce new data to the application by sending an event. Waiting for an event is done internally inside the driver if the application calls one of the wait functions. Waiting for an event does not consume any CPU time and is therefore highly requested if other threads do lot of calculation work. However it is not necessary to use the wait functions and one can simply request the current status whenever the program has time to do so. When using this polling mode the announced available

bytes still stick to the defined notify size!
- If the on-board FIFO buffer has an overrun data transfer is stopped immediately.

Buffer handling example for DMA timestamp transfer (ABA transfer is similar, just using other registers)

```
int8* pcData = (int8*) pvAllocMemPageAligned (lBufSizeInBytes);

// we now define the transfer buffer with the minimum notify size of one page = 4 kByte
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR_CARDTOPC, 4096, (void*) pcData, 0, lBufSizeInBytes);

do
    {
    // we wait for the next data to be available. After this call we get at least 4k of data to proceed
    dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_STARTDMA | M2CMD_EXTRA_WAITDMA);

    if (!dwError)
        {

        // if there was no error we can proceed and read out the current amount of available data
        spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lAvailBytes);
        spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_POS, &lBytePos);

        printf ("We now have %d new bytes available\n", lAvailBytes);
        printf ("The available data starts at position %d\n", lBytesPos);

        // we take care not to go across the end of the buffer
        if ((lBytePos + lAvailBytes) >= lBufSizeInBytes)
            lAvailBytes = lBufSizeInBytes - lBytePos;

        // our do function get's a pointer to the start of the available data section and the length
        vProcessTimestamps (&pcData[lBytesPos], lAvailBytes);

        // the buffer section is now immediately set available for the card
        spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lAvailBytes);
        }
    }
while (!dwError); // we loop forever if no error occurs
```

The extra FIFO has a quite small size compared to the main data buffer. As the transfer is done initiated by the hardware using busmaster DMA this is not critical as long as the application data buffers are large enough and as long as the extra transfer is started BEFORE starting the card.

Data Transfer using Polling

When using M2i cards the Polling mode needs driver version V1.25 and firmware version V11 to run. Please update your system to the newest versions to run this mode. Polling mode for M3i cards is included starting with the first delivered card version.

If the extra data is quite slow and the delay caused by the notify size on DMA transfers is inacceptable for your application it is possible to use the polling mode. Please be aware that the polling mode uses CPU processing power to get the data and that there might be an overrun if your CPU is otherwise busy. You should only use polling mode in special cases and if the amount of data to transfer is not too high.

Most of the functionality is similar to the DMA based transfer mode as explained above.

The polling data transfer mode is activated as soon as the M2CMD_EXTRA_POLL is executed.

Definition of the transfer buffer

is similar to the above explained DMA buffer transfer. The value „notify size" is ignored and should be set to 4k (4096).

Buffer handling

The buffer handling is also similar to the DMA transfer. As soon as one of the registers SPC_TS_AVAIL_USER_LEN or SPC_ABA_AVAIL_USER_LEN is read the driver will read out all available data from the hardware and will return the number of bytes that has been read. In minimum this will be one DWORD = 4 bytes.

Buffer handling example for polling timestamp transfer (ABA transfer is similar, just using other registers)

```
int8* pcData = (int8*) pvAllocMemPageAligned (lBufSizeInBytes);

// we now define the transfer buffer with the minimum notify size of one page = 4 kByte
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR_CARDTOPC, 4096, (void*) pcData, 0, lBufSizeInBytes);

// we start the polling mode
dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_POLL);

// this is pur polling loop
 do
    {
    spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lAvailBytes);
    spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_POS, &lBytePos);

    if (lAvailBytes > 0)
        {
        printf ("We now have %d new bytes available\n", lAvailBytes);
        printf ("The available data starts at position %d\n", lBytesPos);

        // we take care not to go across the end of the buffer
        if ((lBytePos + lAvailBytes) >= lBufSizeInBytes)
            lAvailBytes = lBufSizeInBytes - lBytePos;

        // our do function get's a pointer to the start of the available data section and the length
        vProcessTimestamps (&pcData[lBytesPos], lAvailBytes);

        // the buffer section is now immediately set available for the card
        spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lAvailBytes);
        }
    }
while (!dwError); // we loop forever if no error occurs
```

Comparison of DMA and polling commands

This chapter shows you how small the difference in programming is between the DMA and the polling mode:

	DMA mode	Polling mode
Define the buffer	spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR...);	spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR...);
Start the transfer	spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_STARTDMA)	spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_POLL)
Wait for data	spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_WAITDMA)	not in polling mode
Available bytes?	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);
Min available bytes	programmed notify size	4 bytes
Current position?	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);
Free buffer for card	spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lBytes);	spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lBytes);

Data format

Each timestamp is 56 bit long and internally mapped to 64 bit (8 bytes). The counter value contains the number of clocks that have been recorded with the currently used sampling rate since the last counter-reset has been done. The matching time can easily be calculated as described in the general information section at the beginning of this chapter.

The values the counter is counting and that are stored in the timestamp FIFO represent the moments the trigger event occurs internally. Compared to the real external trigger event, these values are delayed. This delay is fix and therefore can be ignored, as it will be identical for all recordings with the same setup.

Standard data format

When internally mapping the timestamp from 56 bit to a 64 bit value the leading 8 bits are filled up with zeros (as a sign extension for positive values), to have the stamps ready for calculations as a unsigned 64 bit wide integer value.

Timestamp Mode	8th byte	7th byte	6th byte	5th byte	4th byte	3rd byte	2nd byte	1st byte
Standard/StartReset	0h	56 bit wide Timestamp						
Refclock mode	0h	24 bit wide Refclock edge counter (seconds counter)			32bit wide sample counter			

Extended BaseXIO-Data format

Sometimes it is useful to store the level of additional external static signals together with a recording, such as e.g. control inputs of an external input multiplexer or settings of an external. When programming a special flag the upper byte of every 64 bit timestamp value is not (as in standard data mode) filled up with leading zeros, but with the values of the BaseXIO digital inputs. The following table shows the resulting 64 bit timestamps.

Timestamp Mode	8th byte	7th byte	6th byte	5th byte	4th byte	3rd byte	2nd byte	1st byte
Standard / StartReset	XIO7…XIO0	56 bit wide Timestamp						
Refclock mode	XIO7…XIO0	24 bit wide Refclock edge counter (seconds counter)			32bit wide sample counter			

The BaseXIO-Data sampling option requires the option BaseXIO to be installed. All enhanced timestamps are no longer sign extended integer 64 values so that before using these stamps for calculations (such as calculating the difference between two stamps) one has to mask out the leading byte of the stamps first.

Extended BaseXIO incremental encoder counter format

Some applications require to relate the generated timestamps to a rotary position coming from incremental encoders. Therefore this dedicated timestamp mode provides storage of values from two 10 bit counters (leading to a range from 0 to 1023) with separate „Count" and „Reset" lines is provided. This mode requires therefore four external TTL lines which are only available, if the BaseXIO option is installed. Details of the BaseXIO pinout can be found in the BaseXIO chapter in this manual.
With each rising edge of the count input the counter is incremented by one step. Because that input is synchronized into the internal clock domain, please make sure that the HIGH and the LOW time are at least 16 times of the sampling period time long. So when sampling with 100 MS/s (10 ns period time) the pulse has at least a length of 160 ns. A HIGH event on the reset line (assertion) will set the counter asynchronously back to zero. The de-assertion of the reset line is synchronized to the same internal clock domain as the count inputs to prevent runt pulses from corrupting the counter reset.
When enabling the M3i encoder option the total 64 bits of one timestamp are divided up into up to four parts:

Timestamp Mode	8th byte	7th byte	6th byte	5th byte	4th byte	3rd byte	2nd byte	1st byte
Standard / StartReset	10 bit wide Incremental Counter 1	10 bit wide Incremental Counter 2	44bit wide sample counter					
Refclock mode	10 bit wide Incremental Counter 1	10 bit wide Incremental Counter 2	14 bit wide Refclock edge counter (seconds counter)	30bit wide sample counter				

Using the incremental encoder counter requires the driver V2.16 (or newer) and the M3i main control firmware version V14 (or newer). Please update your system to the latest versions to run this mode.

The BaseXIO-Data sampling option requires the option BaseXIO to be installed. All enhanced timestamps are no longer sign extended integer 64 values so that before using these stamps for calculations (such as calculating the difference between two stamps) one has to mask out the incremental counter and the Reference clock counter values of the stamps first.

Selecting the timestamp data format

The selection between the different data format for the timestamps is done with a flag that is written to the timestamp command register. As this register is organized as a bitfield, the data format selection is available for all possible timestamp modes.

Register	Value	Direction	Description
SPC_TIMESTAMP_CMD	47100	r/w	
SPC_TSXIOINC_ENABLE	2000h		Enables the trigger synchronous acquisition of the two 10 bit incremental counters with every stored timestamp in the upper 20 bit of the timestamp data.
SPC_TSXIOACQ_ENABLE	1000h		Enables the trigger synchronous acquisition of the BaseXIO inputs with every stored timestamp in the upper byte.
SPC_TSXIOACQ_DISABLE	0		The timestamp is filled up with leading zeros as a sign extension for positive values.

Combination of Memory Segmentation Options with Timestamps

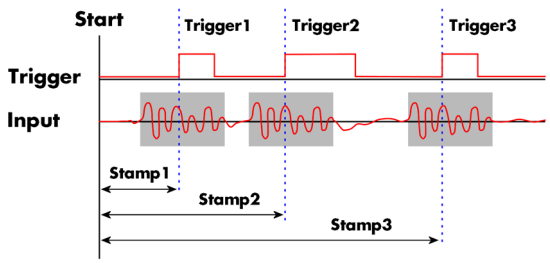
This topic should give you a brief overview how the timestamp option interacts with the options Multiple Recording and ABA mode for which the timestamps option has been made.

Multiple Recording and Timestamps

Multiple Recording is well matching with the timestamp option. If timestamp recording is activated each trigger event and therefore each Multiple Recording segment will get timestamped as shown in the drawing on the right.

Please keep in mind that the trigger events are timestamped, not the beginning of the acquisition. The first sample that is available is at the time position of [Timestamp - Pretrigger].

The programming details of the timestamp option is explained in an extra chapter.

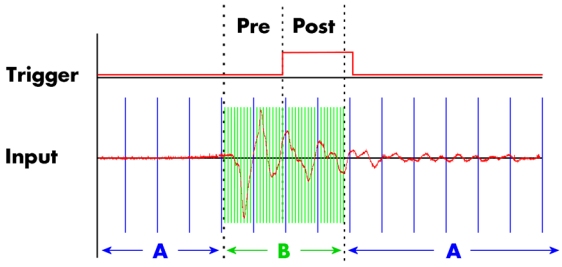


The following example shows the setup of the Multiple Recording mode together with activated timestamps recording and a short display of the acquired timestamps. The example doesn't care for the acquired data itself and doesn't check for error:

```
// setup of the Multiple Recording mode
spcm_dwSetParam_i32 (hDrv, SPC_CARDMODE, SPC_REC_STD_MULTI);  // Enables Standard Multiple Recording
spcm_dwSetParam_i64 (hDrv, SPC_SEGMENTSIZE,    1024);         // Segment size is 1 kSample, Posttrigger is 768
spcm_dwSetParam_i64 (hDrv, SPC_POSTTRIGGER,     768);         // samples and pretrigger therefore 256 samples.
spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE,        4096);         // 4 kSamples in total acquired -> 4 segments

// setup the Timestamp mode and make a reset of the timestamp counter
spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, SPC_TSMODE_STANDARD | SPC_TSCNT_INTERNAL);
spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, SPC_TSMODE_RESET);

// now we define a buffer for timestamp data and start acquistion, each timestamp is 64 bit = 8 bytes
int64* pllStamps = (int64*) pvAllocMemPageAligned (8 * 4);
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR_CARDTOPC, 0, (void*) pllStamps, 0, 4 * 8);
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER | M2CMD_EXTRA_STARTDMA);

// we wait for the end timestamps transfer which will be received if all segments have been recorded
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_WAITDMA);

// as we now have the timestamps we just print them and calculate the time in milli seconds
int64 llSamplerate;
double dTime_ms;
int32 lOver;
spcm_dwGetParam_i64 (hDrv, SPC_SAMPLERATE, &llSamplerate);
spcm_dwGetParam_i32 (hDrv, SPC_OVERSAMPLINGFACTOR, &lOver);

for (int i = 0; i < 4; i++)
    {
    dTime_ms = 1000.0 * pllStamps[i] / llSamplerate / lOver);

    printf ("#%d: %I64d samples = %.3f ms\n", i, pllStamps[i], dTime_ms);
    }
```

ABA Mode and Timestamps

The ABA mode is well matching with the timestamp option. If timestamp recording is activated, each trigger event and therefore each B time base segment will get time tamped as shown in the drawing on the right.



Please keep in mind that the trigger events - located in the B area - are time tamped, not the beginning of the acquisition. The first B sample that is available is at the time position of [Timestamp - Pretrigger].

The first A area sample is related to the card start and therefore in a fixed but various settings dependent relation to the timestamped B sample. To bring exact relation between the first A area sample (and therefore all area A samples) and the B area samples it is possible to let the card stamp the first A area sample automatically after the card start. The following table shows the register to enable this mode:

Register	Value	Direction	Description
SPC_TIMESTAMP_CMD	47000	read/write	Programs a timestamp setup including mode and additional features
SPC_TSFEAT_MASK	F0000h		Mask for the feature relating bits of the SPC_TIMESTAMP_CMD bitmask.
SPC_TSFEAT_STORE1STABA	10000h		Enables storage of one additional timestamp for the first A area sample (B time base related) in addition to the trigger related timestamps.
SPC_TSFEAT_NONE	0h		No additional timestamp is created. The total number of stamps is only trigger related.

This mode is compatible with all existing timestamp modes. Please keep in mind that the timestamp counter is running with the B area time-base.

```
// normal timestamp setup (e.g. setting timestamp mode to standard using internal clocking)
uint32 dwTimestampMode = (SPC_TSMODE_STANDARD | SPC_TSMODE_DISABLE);

// additionally enable index of the first A area sample
dwTimestampMode |= SPC_TSFEAT_STORE1STABA;

spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, dwTimestampMode);
```

The programming details of the ABA mode and timestamp modes are each explained in an dedicated chapter in this manual.

Using the cards in ABA mode with the timestamp feature to stamp the first A are sample requires the following driver and firmware version depending on your card:

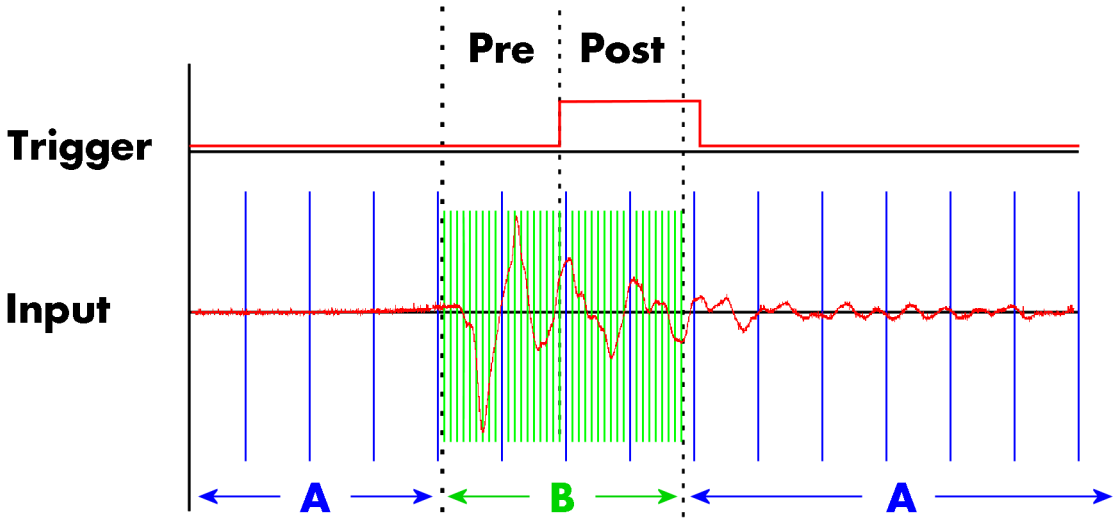
M2i: driver version V2.06 (or newer) and firmware version V16 (or newer)
M3i: driver version V2.06 (or newer) and firmware version V6 (or newer)

Please update your system to the newest versions to run this mode.

ABA mode (dual timebase)

General information

The ABA mode allows the acquisition of data with a dual timebase. In case of trigger event the inputs are sampled very fast with the programmed sampling rate. This part is similar to the Multiple Recording option. But instead of having no data in between the segments one has the opportunity to continuously sample the inputs with a slower sampling rate the whole time. Combining this with the recording of the timestamps gives you a complete acquisition with a dual timebase as shown in the drawing.



As seen in the drawing the area around the trigger event is sampled between pretrigger and posttrigger with full sampling speed (area B of the acquisition). Outside of this area B the input is sampled with the slower ABA clock (area A of the acquisition). As changing sampling clock on the fly is not possible there is no real change in the sampling speed but area A runs continuously with a slow sampling speed without stopping when the fast sampling takes place. As a result one gets a continuous slow sampled acquisition (area A) with some fast sampled parts (area B)

The ABA mode is available for standard recording as well as for FIFO recording. In case of FIFO recording ABA and the acquisition of the fast sampled segments will run continuously until it is stopped by the user.

A second possible application for the ABA mode is the use of the ABA data for slow monitoring of the inputs while waiting for an acquisition. In that case one wouldn't record the timestamps but simply monitor the current values by acquiring ABA data.

The ABA mode needs a second clock base. As explained above the acquisition is not changing the sampling clock but runs the slower acquisition with a divided clock. The ABA memory setup including the divider value can be programmed with the following registers

Register	Value	Direction	Description
SPC_SEGMENTSIZE	10010	read/write	Size of one B-segment: the total number of samples to be recorded per channel after detection of one trigger event including the time recorded before the trigger (pre trigger).
SPC_POSTTRIGGER	10030	read/write	Defines the number of samples to be recorded after each trigger event per channel.
SPC_ABADIVIDER	10040	read/write	Programs the divider which is used to sample slow ABA data: For 12 bit and 14 bit cards : between 8 and 131064 in steps of 8 For 8 bit: between 16 and 262128 in steps of 16

The resulting ABA clock is then calculated by sampling rate / ABA divider.

Each segment can consist of pretrigger and/or posttrigger samples. The user always has to set the total segment size and the posttrigger, while the pretrigger is calculated within the driver with the formula: [pretrigger] = [segment size] - [posttrigger].

When using ABA mode or Multiple Recording the maximum pretrigger is limited depending on the number of active channels. When the calculated value exceeds that limit, the driver will return the error ERR_PRETRIGGERLEN.

Standard Mode

With every detected trigger event one data block is filled with data. The length of one ABA segment is set by the value of the segmentsize register. The total amount of samples to be recorded is defined by the memsize register.
Memsize must be set to a a multiple of the segment size. The table below shows the register for enabling standard ABA mode. For detailed information on how to setup and start the standard acquisition mode please refer to the according chapter earlier in this manual.

Register	Value	Direction	Description
SPC_CARDMODE	9500	read/write	Defines the used operating mode

SPC_REC_STD_ABA	8h	Data acquisition to on-board memory for multiple trigger events. While the multiple trigger events are stored with programmed sampling rate the inputs are sampled continuously with a slower sampling speed.

The total number of samples to be recorded to the on-board memory in standard mode is defined by the SPC_MEMSIZE register.

Register	Value	Direction	Description
SPC_MEMSIZE	10000	read/write	Defines the total number of samples to be recorded per channel.

FIFO Mode

The ABA FIFO Mode is similar to the Multiple Recording FIFO mode. In contrast to the standard mode it is not necessary to program the number of samples to be recorded. The acquisition is running until the user stops it. The data is read block by block by the driver as described under Single FIFO mode example earlier in this manual. These blocks are online available for further data processing by the user program. This mode significantly reduces the average data transfer rate on the PCI bus. This enables you to use faster sample rates then you would be able to in FIFO mode without ABA.

Register		Value	Direction	Description
SPC_CARDMODE		9500	read/write	Defines the used operating mode
	SPC_REC_FIFO_ABA	80h	Continuous data acquisition for multiple trigger events together with continuous data acquisition with a slower sampling clock.	

The number of segments to be recorded must be set separately with the register shown in the following table:

Register	Value	Direction	Description
SPC_LOOPS	10020	read/write	Defines the number of segments to be recorded
0		Recording will be infinite until the user stops it.	
1 ... [4G - 1]		Defines the total segments to be recorded.	

Limits of pre trigger, post trigger, memory size

The maximum memory size parameter is only limited by the number of activated channels and by the amount of installed memory. Please keep in mind that each sample needs 2 bytes of memory to be stored. Minimum memory size as well as minimum and maximum post trigger limits are independent of the activated channels or the installed memory.

Due to the internal organization of the card memory there is a certain stepsize when setting these values that has to be taken into account. The following table gives you an overview of all limits concerning pre trigger, post trigger, memory size, segment size and loops. The table shows all values in relation to the installed memory size in samples. If more memory is installed the maximum memory size figures will increase according to the complete installed memory

Activated Channels	Used Mode	Memory size SPC_MEMSIZE			Pre trigger SPC_PRETRIGGER			Post trigger SPC_POSTTRIGGER			Segment size SPC_SEGMENTSIZE			Loops SPC_LOOPS		
		Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step
1 channel	Standard Single	16	Mem	8	defined by post trigger			8	8G - 8	8	not used			not used		
	Standard Multi/ABA	16	Mem	8	8	8k	8	8	Mem/2	8	16	Mem/2	8	not used		
	FIFO Single	not used			8	8k	8	not used			16	8G - 8	8	0 (∞)	4G - 1	1
	FIFO Multi/ABA	not used			8	8k	8	8	8G - 8	8	16	pre+post	8	0 (∞)	4G - 1	1
2 channels	Standard Single	16	Mem/2	8	defined by post trigger			8	8G - 8	8	not used			not used		
	Standard Multi/ABA	16	Mem/2	8	8	4k	8	8	Mem/4	8	16	Mem/4	8	not used		
	FIFO Single	not used			8	4k	8	not used			16	8G - 8	8	0 (∞)	4G - 1	1
	FIFO Multi/ABA	not used			8	4k	8	8	8G - 8	8	16	pre+post	8	0 (∞)	4G - 1	1

All figures listed here are given in samples. An entry of [32G - 8] means [32 GSamples - 8] = 34,359,738,360 samples.

The given memory and memory / divider figures depend on the installed on-board memory as listed below:

	Installed Memory				
	128 MSample	**256 MSample**	**512 MSample**	**1 GSample**	**2 GSample**
Mem	128 MSample	256 MSample	512 MSample	1 GSample	2 GSample
Mem / 2	64 MSample	128 MSample	256 MSample	512 MSample	1 GSample
Mem / 4	32 MSample	64 MSample	128 MSample	256 MSample	512 MSample

Please keep in mind that this table shows all values at once. Only the absolute maximum and minimum values are shown. There might be additional limitations. Which of these values is programmed depends on the used mode. Please read the detailed documentation of the mode.

Example for setting ABA mode:

The following example will program the standard ABA mode, will set the fast sampling rate to 100 MHz and acquire 2k segments with 1k pretrigger and 1k posttrigger on every rising edge of the trigger input. Meanwhile the inputs are sampled continuously with the ABA mode with a ABA divider set to 5000 resulting in a slow sampling clock for the A area of 100 MHz / 5000 = 20 kHz:

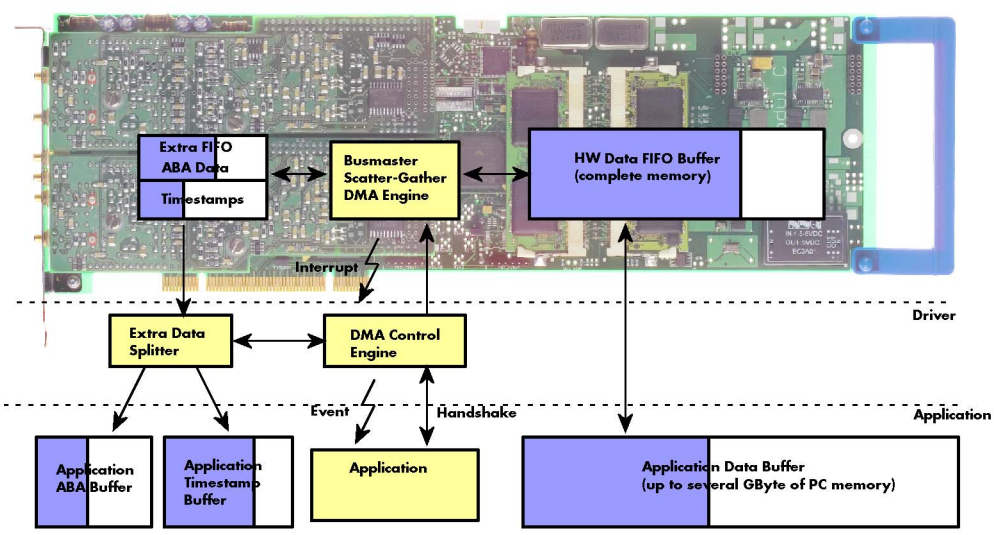
```
// setting the fast sampling clock as internal 100 MHz
spcm_dwSetParam_i32 (hDrv, SPC_CLOCKMODE, SPC_CM_INTPLL);
spcm_dwSetParam_i64 (hDrv, SPC_SAMPLERATE, 100000000);

// enable the ABA mode and set the ABA divider to 5000 -> 100 MHz / 5000 = 20 kHz
spcm_dwSetParam_i32 (hDrv, SPC_CARDMODE, SPC_REC_STD_ABA);
spcm_dwSetParam_i32 (hDrv, SPC_ABADIVIDER, 5000);

// define the segmentsize, pre and posttrigger and the total amount of data to acquire
spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE, 16384);
spcm_dwSetParam_i64 (hDrv, SPC_SEGMENTSIZE, 2048);
spcm_dwSetParam_i64 (hDrv, SPC_POSTTRIGGER, 1024);

// set the trigger mode to external with positive edge
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_ORMASK, SPC_TMASK_EXT0);
spcm_dwSetParam_i32 (hDrv, SPC_TRIG_EXT0_MODE, SPC_TM_POS);
```

Reading out ABA data

General

The slow „A" data is stored in an extra FIFO that is located in hardware on the card. This extra FIFO can read out slow „A" data using DMA transfer similar to the DMA transfer of the main sample data DMA transfer. The card has two completely independent busmaster DMA engines in hardware allowing the simultaneous transfer of both „A" and sample data. The sample data itself is read out as explained before using the standard DMA routine.

As seen in the picture the extra FIFO is holding ABA and timestamp data as the same time. Nevertheless it is not necessary to care for the shared FIFO as the extra FIFO data is splitted inside the driver in the both data parts.



The only part that is similar for both kinds of data transfer is the handling of the DMA engine. This is similar to the main sample data transfer engine. Therefore additional information can be found in the chapter explaining the main data transfer.

Commands and Status information for extra transfer buffers.

As explained above the data transfer is performed with the same command and status registers like the card control and sample data transfer. It is possible to send commands for card control, data transfer and extra FIFO data transfer at the same time

Register	Value	Direction	Description
SPC_M2CMD	100	write only	Executes a command for the card or data transfer
M2CMD_EXTRA_STARTDMA	100000h		Starts the DMA transfer for an already defined buffer.
M2CMD_EXTRA_WAITDMA	200000h		Waits until the data transfer has ended or until at least the amount of bytes defined by notify size are available. This wait function also takes the timeout parameter into account.
M2CMD_EXTRA_STOPDMA	400000h		Stops a running DMA transfer. Data is invalid afterwards.
M2CMD_EXTRA_POLL	800000h		Polls data without using DMA. As DMA has some overhead and has been implemented for fast data transfer of large amounts of data it is in some cases more simple to poll for available data. Please see the detailed examples for this mode. It is not possible to mix DMA and polling mode.

The extra FIFO data transfer can generate one of the following status information:.

Register	Value	Direction	Description
SPC_M2STATUS	110	read only	Reads out the current status information
M2STAT_EXTRA_BLOCKREADY	1000h		The next data block as defined in the notify size is available. It is at least the amount of data available but it also can be more data.
M2STAT_EXTRA_END	2000h		The data transfer has completed. This status information will only occur if the notify size is set to zero.
M2STAT_EXTRA_OVERRUN	4000h		The data transfer had on overrun (acquisition) or underrun (replay) while doing FIFO transfer.
M2STAT_EXTRA_ERROR	8000h		An internal error occurred while doing data transfer.

Data Transfer using DMA

Data transfer consists of two parts: the buffer definition and the commands/status information that controls the transfer itself. Extra data transfer shares the command and status register with the card control, data transfer commands and status information.

The DMA based data transfer mode is activated as soon as the M2CMD_EXTRA_STARTDMA is given. Please see next chapter to see how the polling mode works.

Definition of the transfer buffer

Before any data transfer can start it is necessary to define the transfer buffer with all its details. The definition of the buffer is done with the spcm_dwDefTransfer function as explained in an earlier chapter. The following example will show the definition of a transfer buffer for timestamp data, definition for ABA data is similar:

```
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_CARDTOPC, 0, pvBuffer, 0, lLenOfBufferInBytes);
```

In this example the notify size is set to zero, meaning that we don't want to be notified until all extra data has been transferred. Please have a look at the sample data transfer in an earlier chapter to see more details on the notify size.

Please note that extra data transfer is only possible from card to PC and there's no programmable offset available for this transfer.

Buffer handling

A data buffer handshake is implemented in the driver which allows to run the card in different data transfer modes. The software transfer buffer is handled as one large buffer for each kind of data (timestamp and ABA) which is on the one side controlled by the driver and filled automatically by busmaster DMA from the hardware extra FIFO buffer and on the other hand it is handled by the user who set's parts of this software buffer available for the driver for further transfer. The handshake is fulfilled with the following 3 software registers:

Register	Value	Direction	Description
SPC_ABA_AVAIL_USER_LEN	210	read	This register contains the currently available number of bytes that are filled with newly transferred slow ABA data. The user can now use this ABA data for own purposes, copy it, write it to disk or start calculations with this data.
SPC_ABA_AVAIL_USER_POS	211	read	The register holds the current byte index position where the available ABA bytes start. The register is just intended to help you and to avoid own position calculation
SPC_ABA_AVAIL_CARD_LEN	212	write	After finishing the job with the new available ABA data the user needs to tell the driver that this amount of bytes is again free for new data to be transferred.
SPC_TS_AVAIL_USER_LEN	220	read	This register contains the currently available number of bytes that are filled with newly transferred timestamp data. The user can now use these timestamps for own purposes, copy it, write it to disk or start calculations with the timestamps.
SPC_TS_AVAIL_USER_POS	221	read	The register holds the current byte index position where the available timestamp bytes start. The register is just intended to help you and to avoid own position calculation
SPC_TS_AVAIL_CARD_LEN	222	write	After finishing the job with the new available timestamp data the user needs to tell the driver that this amount of bytes is again free for new data to be transferred.

Directly after start of transfer the SPC_XXX_AVAIL_USER_LEN is every time zero as no data is available for the user and the SPC_XXX_AVAIL_CARD_LEN is every time identical to the length of the defined buffer as the complete buffer is available for the card for transfer.

The counter that is holding the user buffer available bytes (SPC_XXX_AVAIL_USER_LEN) is sticking to the defined notify size at the DefTransfer call. Even when less bytes already have been transferred you won't get notice of it if the notify size is programmed to a higher value.

Remarks

- The transfer between hardware FIFO buffer and application buffer is done with scatter-gather DMA using a busmaster DMA controller located on the card. Even if the PC is busy with other jobs data is still transferred until the application buffer is completely used.
- As shown in the drawing above the DMA control will announce new data to the application by sending an event. Waiting for an event is done internally inside the driver if the application calls one of the wait functions. Waiting for an event does not consume any CPU time and is therefore highly requested if other threads do lot of calculation work. However it is not necessary to use the wait functions and one can simply request the current status whenever the program has time to do so. When using this polling mode the announced available

bytes still stick to the defined notify size!
- If the on-board FIFO buffer has an overrun data transfer is stopped immediately.

Buffer handling example for DMA timestamp transfer (ABA transfer is similar, just using other registers)

```
int8* pcData = (int8*) pvAllocMemPageAligned (lBufSizeInBytes);

// we now define the transfer buffer with the minimum notify size of one page = 4 kByte
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR_CARDTOPC, 4096, (void*) pcData, 0, lBufSizeInBytes);

do
    {
    // we wait for the next data to be available. After this call we get at least 4k of data to proceed
    dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_STARTDMA | M2CMD_EXTRA_WAITDMA);

    if (!dwError)
        {

        // if there was no error we can proceed and read out the current amount of available data
        spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lAvailBytes);
        spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_POS, &lBytePos);

        printf ("We now have %d new bytes available\n", lAvailBytes);
        printf ("The available data starts at position %d\n", lBytesPos);

        // we take care not to go across the end of the buffer
        if ((lBytePos + lAvailBytes) >= lBufSizeInBytes)
            lAvailBytes = lBufSizeInBytes - lBytePos;

        // our do function get's a pointer to the start of the available data section and the length
        vProcessTimestamps (&pcData[lBytesPos], lAvailBytes);

        // the buffer section is now immediately set available for the card
        spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lAvailBytes);
        }
    }
while (!dwError); // we loop forever if no error occurs
```

The extra FIFO has a quite small size compared to the main data buffer. As the transfer is done initiated by the hardware using busmaster DMA this is not critical as long as the application data buffers are large enough and as long as the extra transfer is started BEFORE starting the card.

Data Transfer using Polling

When using M2i cards the Polling mode needs driver version V1.25 and firmware version V11 to run. Please update your system to the newest versions to run this mode. Polling mode for M3i cards is included starting with the first delivered card version.

If the extra data is quite slow and the delay caused by the notify size on DMA transfers is inacceptable for your application it is possible to use the polling mode. Please be aware that the polling mode uses CPU processing power to get the data and that there might be an overrun if your CPU is otherwise busy. You should only use polling mode in special cases and if the amount of data to transfer is not too high.

Most of the functionality is similar to the DMA based transfer mode as explained above.

The polling data transfer mode is activated as soon as the M2CMD_EXTRA_POLL is executed.

Definition of the transfer buffer

is similar to the above explained DMA buffer transfer. The value „notify size" is ignored and should be set to 4k (4096).

Buffer handling

The buffer handling is also similar to the DMA transfer. As soon as one of the registers SPC_TS_AVAIL_USER_LEN or SPC_ABA_AVAIL_USER_LEN is read the driver will read out all available data from the hardware and will return the number of bytes that has been read. In minimum this will be one DWORD = 4 bytes.

Buffer handling example for polling timestamp transfer (ABA transfer is similar, just using other registers)

```
int8* pcData = (int8*) pvAllocMemPageAligned (lBufSizeInBytes);

// we now define the transfer buffer with the minimum notify size of one page = 4 kByte
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR_CARDTOPC, 4096, (void*) pcData, 0, lBufSizeInBytes);

// we start the polling mode
dwError = spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_POLL);

// this is pur polling loop
 do
    {
    spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lAvailBytes);
    spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_POS, &lBytePos);

    if (lAvailBytes > 0)
        {
        printf ("We now have %d new bytes available\n", lAvailBytes);
        printf ("The available data starts at position %d\n", lBytesPos);

        // we take care not to go across the end of the buffer
        if ((lBytePos + lAvailBytes) >= lBufSizeInBytes)
            lAvailBytes = lBufSizeInBytes - lBytePos;

        // our do function get's a pointer to the start of the available data section and the length
        vProcessTimestamps (&pcData[lBytesPos], lAvailBytes);

        // the buffer section is now immediately set available for the card
        spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lAvailBytes);
        }
    }
while (!dwError); // we loop forever if no error occurs
```

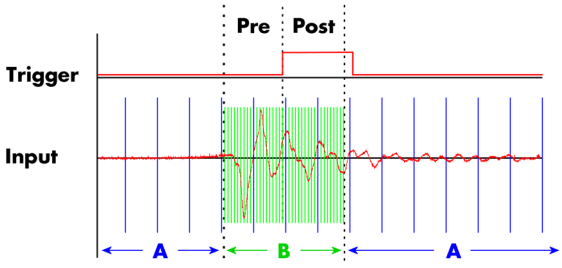
Comparison of DMA and polling commands

This chapter shows you how small the difference in programming is between the DMA and the polling mode:

	DMA mode	Polling mode
Define the buffer	spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR...);	spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_TIMESTAMP, SPCM_DIR...);
Start the transfer	spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_STARTDMA)	spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_POLL)
Wait for data	spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_EXTRA_WAITDMA)	not in polling mode
Available bytes?	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);
Min available bytes	programmed notify size	4 bytes
Current position?	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);	spcm_dwGetParam_i32 (hDrv, SPC_TS_AVAIL_USER_LEN, &lBytes);
Free buffer for card	spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lBytes);	spcm_dwSetParam_i32 (hDrv, SPC_TS_AVAIL_CARD_LEN, lBytes);

ABA Mode and Timestamps

The ABA mode is well matching with the timestamp option. If timestamp recording is activated, each trigger event and therefore each B time base segment will get time tamped as shown in the drawing on the right.



Please keep in mind that the trigger events - located in the B area - are time tamped, not the beginning of the acquisition. The first B sample that is available is at the time position of [Timestamp - Pretrigger].

The first A area sample is related to the card start and therefore in a fixed but various settings dependent relation to the timestamped B sample. To bring exact relation between the first A area sample (and therefore all area A samples) and the B area samples it is possible to let the card stamp the first A area sample automatically after the card start. The following table shows the register to enable this mode:

Register	Value	Direction	Description
SPC_TIMESTAMP_CMD	47000	read/write	Programs a timestamp setup including mode and additional features
SPC_TSFEAT_MASK	F0000h	Mask for the feature relating bits of the SPC_TIMESTAMP_CMD bitmask.	
SPC_TSFEAT_STORE1STABA	10000h	Enables storage of one additional timestamp for the first A area sample (B time base related) in addition to the trigger related timestamps.	
SPC_TSFEAT_NONE	0h	No additional timestamp is created. The total number of stamps is only trigger related.	

This mode is compatible with all existing timestamp modes. Please keep in mind that the timestamp counter is running with the B area timebase.

```
// normal timestamp setup (e.g. setting timestamp mode to standard using internal clocking)
uint32 dwTimestampMode = (SPC_TSMODE_STANDARD | SPC_TSMODE_DISABLE);

// additionally enable index of the first A area sample
dwTimestampMode |= SPC_TSFEAT_STORE1STABA;

spcm_dwSetParam_i32 (hDrv, SPC_TIMESTAMP_CMD, dwTimestampMode);
```

The programming details of the ABA mode and timestamp modes are each explained in an dedicated chapter in this manual.

Using the cards in ABA mode with the timestamp feature to stamp the first A are sample requires the following driver and firmware version depending on your card:

M2i: driver version V2.06 (or newer) and firmware version V16 (or newer)
M3i: driver version V2.06 (or newer) and firmware version V6 (or newer)

Please update your system to the newest versions to run this mode.

Option Star-Hub (M3i and M4i only)

Star-Hub introduction

The purpose of the Star-Hub is to extend the number of channels available for acquisition or generation by interconnecting multiple cards and running them simultaneously.

The Star-Hub option allows to synchronize several cards of the same M3i/M4i series that are mounted within one host system (PC):

- For the M3i series there are the two different versions available: a small version with 4 connectors (option SH4) for synchronizing up to four cards and a big version with 8 connectors (option SH8) for synchronizing up to eight cards.
- For the M4i series there are the two different mechanical versions available, with 8 connectors for synchronizing up to eight cards.

The Star-Hub allows synchronizing cards of the same family only. It is not possible to synchronize cards of different families!

Both versions are implemented as a piggy -back module that is mounted to one of the cards. For details on how to install several cards including the one carrying the Star-Hub module, please refer to the section on hardware installation.

Either which of the two available Star-Hub options is used, there will be no phase delay between the sampling clocks of the synchronized cards and either no delay between the trigger events. The card holding the Star-Hub is automatically also the clock master. Any one of the synchronized cards can be part of the trigger generation.

Star-Hub trigger engine

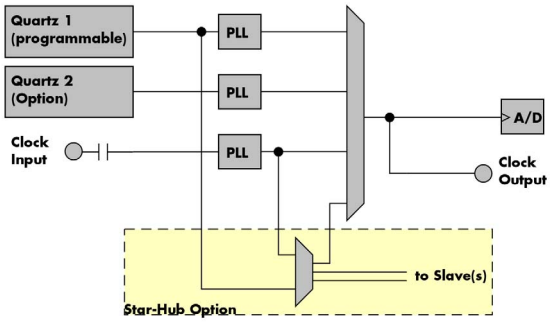
The trigger bus between an M3i/M4i card and the Star-Hub option consists of several lines. Some of them send the trigger information from the card's trigger engine to the Star-Hub and some receives the resulting trigger from the Star-Hub. All trigger events from the different cards connected are combined with OR on the Star-Hub.

While the returned trigger is identical for all synchronized cards, the sent out trigger of every single card depends on their trigger settings.

Star-Hub clock engine

The card holding the Star-Hub is the clock master for the complete system. If you need to feed in an external clock to a synchronized system the clock has to be connected to the master card. Slave cards cannot generate a Star-Hub system clock. As shown in the drawing on the right the clock master can use either the programmable quartz 1 or the external clock input to be broadcast to all other cards.

All cards including the clock master itself receive the distributed clock with equal phase information. This makes sure that there is no phase delay between the cards.



Software Interface

The software interface is similar to the card software interface that is explained earlier in this manual. The same functions and some of the registers are used with the Star-Hub. The Star-Hub is accessed using its own handle which has some extra commands for synchronization setup. All card functions are programmed directly on card as before. There are only a few commands that need to be programmed directly to the Star-Hub for synchronization.

The software interface as well as the hardware supports multiple Star-Hubs in one system. Each set of cards connected by a Star-Hub then runs totally independent. It is also possible to mix cards that are connected with the Star-Hub with other cards that run independent in one system.

Star-Hub Initialization

The interconnection between the Star-Hubs is probed at driver load time and does not need to be programmed separately. Instead the cards can be accessed using a logical index. This card index is only based on the ordering of the cards in the system and is not influenced by the current cabling. It is even possible to change the cable connections between two system starts without changing the logical card order that is used for Star-Hub programming.

The Star-Hub initialization must be done AFTER initialization of all cards in the system. Otherwise the interconnection won't be received properly.

The Star-Hubs are accessed using a special device name „sync“ followed by the index of the star-hub to access. The Star-Hub is handled completely like a physical card allowing all functions based on the handle like the card itself.

Example with 4 cards and one Star-Hub (no error checking to keep example simple)

```
drv_handle  hSync;
drv_handle  hCard[4];

for (i = 0; i < 4; i++)
    {
    sprintf (s, "/dev/spcm%d", i);
    hCard[i] = spcm_hOpen (s);
    }
hSync = spcm_hOpen ("sync0");

...

spcm_vClose (hSync);
for (i = 0; i < 4; i++)
    spcm_vClose (hCard[i]);
```

Example for a digitizerNETBOX with two internal digitizer/generator modules, This example is also suitable for accessing a remote server with two cards installed:

```
drv_handle  hSync;
drv_handle  hCard[2];

for (i = 0; i < 2; i++)
    {
    sprintf (s, "TCPIP::192.168.169.14::INST%d::INSTR", i);
    hCard[i] = spcm_hOpen (s);
    }
hSync = spcm_hOpen ("sync0");

...

spcm_vClose (hSync);
for (i = 0; i < 2; i++)
    spcm_vClose (hCard[i]);
```

When opening the Star-Hub the cable interconnection is checked. The Star-Hub may return an error if it sees internal cabling problems or if the connection between Star-Hub and the card that holds the Star-Hub is broken. It can’t identify broken connections between Star-Hub and other cards as it doesn’t know that there has to be a connection.

The synchronization setup is done using bit masks where one bit stands for one recognized card. All cards that are connected with a Star-Hub are internally numbered beginning with 0. The number of connected cards as well as the connections of the star-hub can be read out after initialization. For each card that is connected to the star-hub one can read the index of that card:

Register	Value	Direction	Description
SPC_SYNC_READ_NUMCONNECTORS	48991	read	Number of connectors that the Star-Hub offers at max. (available with driver V5.6 or newer)
SPC_SYNC_READ_SYNCCOUNT	48990	read	Number of cards that are connected to this Star-Hub
SPC_SYNC_READ_CARDIDX0	49000	read	Index of card that is connected to star-hub logical index 0 (mask 0x0001)
SPC_SYNC_READ_CARDIDX1	49001	read	Index of card that is connected to star-hub logical index 1 (mask 0x0002)
...		read	...
SPC_SYNC_READ_CARDIDX7	49007	read	Index of card that is connected to star-hub logical index 7 (mask 0x0080)
SPC_SYNC_READ_CARDIDX8	49008	read	M2i only: Index of card that is connected to star-hub logical index 8 (mask 0x0100)
...		read	...
SPC_SYNC_READ_CARDIDX15	49015	read	M2i only: Index of card that is connected to star-hub logical index 15 (mask 0x8000)
SPC_SYNC_READ_CABLECON0		read	Returns the index of the cable connection that is used for the logical connection 0. The cable connections can be seen printed on the PCB of the star-hub. Use these cable connection information in case that there are hardware failures with the star-hub cabeling.
...	49100	read	...
SPC_SYNC_READ_CABLECON15	49115	read	Returns the index of the cable connection that is used for the logical connection 15.

In standard systems where all cards are connected to one star-hub reading the star-hub logical index will simply return the index of the card again. This results in bit 0 of star-hub mask being 1 when doing the setup for card 0, bit 1 in star-hub mask being 1 when setting up card 1 and so on. On such systems it is sufficient to read out the SPC_SYNC_READ_SYNCCOUNT register to check whether the star-hub has found the expected number of cards to be connected.

```
spcm_dwGetParam_i32 (hSync, SPC_SYNC_READ_SYNCCOUNT, &lSyncCount);
for (i = 0; i < lSyncCount; i++)
    {
    spcm_dwGetParam_i32 (hSync, SPC_SYNC_READ_CARDIDX0 + i, &lCardIdx);
    printf ("star-hub logical index %d is connected with card %d\n", i, lCardIdx);
    }
```

In case of 4 cards in one system and all are connected with the star-hub this program excerpt will return:

```
star-hub logical index 0 is connected with card 0
star-hub logical index 1 is connected with card 1
star-hub logical index 2 is connected with card 2
star-hub logical index 3 is connected with card 3
```

Let's see a more complex example with two Star-Hubs and one independent card in one system. Star-Hub A connects card 2, card 4 and card 5. Star-Hub B connects card 0 and card 3. Card 1 is running completely independent and is not synchronized at all:

card	Star-Hub connection	card handle	star-hub handle	card index in star-hub	mask for this card in star-hub
card 0	-	/dev/spcm0		0 (of star-hub B)	0x0001
card 1	-	/dev/spcm1			-
card 2	star-hub A	/dev/spcm2	sync0	0 (of star-hub A)	0x0001
card 3	star-hub B	/dev/spcm3	sync1	1 (of star-hub B)	0x0002
card 4	-	/dev/spcm4		1 (of star-hub A)	0x0002
card 5	-	/dev/spcm5		2 (of star-hub A)	0x0004

Now the program has to check both star-hubs:

```
for (j = 0; j < lStarhubCount; j++)
    {
    spcm_dwGetParam_i32 (hSync[j], SPC_SYNC_READ_SYNCCOUNT, &lSyncCount);
    for (i = 0; i < lSyncCount; i++)
        {
        spcm_dwGetParam_i32 (hSync[j], SPC_SYNC_READ_CARDIDX0 + i, &lCardIdx);
        printf ("star-hub %c logical index %d is connected with card %d\n", (!j ? 'A' : 'B'), i, lCardIdx);
        }
    printf ("\n");
    }
```

In case of the above mentioned cabling this program excerpt will return:

```
star-hub A logical index 0 is connected with card 2
star-hub A logical index 1 is connected with card 4
star-hub A logical index 2 is connected with card 5

star-hub B logical index 0 is connected with card 0
star-hub B logical index 1 is connected with card 3
```

For the following examples we will assume that 4 cards in one system are all connected to one star-hub to keep things easier.

Setup of Synchronization

The synchronization setup only requires one additional register to enable the cards that are synchronized in the next run

Register	Value	Direction	Description
SPC_SYNC_ENABLEMASK	49200	read/write	Mask of all cards that are enabled for the synchronization

The enable mask is based on the logical index explained above. It is possible to just select a couple of cards for the synchronization. All other cards then will run independently. Please be sure to always enable the card on which the star-hub is located as this one is a must for the synchronization.

In our example we synchronize all four cards. The star-hub is located on card #2 and is therefor the clock master

```
spcm_dwSetParam_i32 (hSync, SPC_SYNC_ENABLEMASK, 0x000F); // all 4 cards are masked

// set the clock master to 100 MS/s internal clock
spcm_dwSetParam_i32 (hCard[2], SPC_CLOCKMODE, SPC_CM_INTPLL);
spcm_dwSetParam_i32 (hCard[2], SPC_SAMPLEATE, MEGA(100));

// set all the slaves to run synchronously with 100 MS/s
spcm_dwSetParam_i32 (hCard[0], SPC_SAMPLEATE, MEGA(100));
spcm_dwSetParam_i32 (hCard[1], SPC_SAMPLEATE, MEGA(100));
spcm_dwSetParam_i32 (hCard[3], SPC_SAMPLEATE, MEGA(100));
```

Setup of Trigger

Setting up the trigger does not need any further steps of synchronization setup. Simply all trigger settings of all cards that have been enabled for synchronization are connected together. All trigger sources and all trigger modes can be used on synchronization as well.

Having positive edge of external trigger on card 0 to be the trigger source for the complete system needs the following setup:

```
spcm_dwSetParam_i32 (hCard[0], SPC_TRIG_ORMASK, SPC_TMASK_EXT0);
spcm_dwSetParam_i32 (hCard[0], SPC_TRIG_EXT0_MODE, SPC_TM_POS);

spcm_dwSetParam_i32 (hCard[1], SPC_TRIG_ORMASK, SPC_TM_NONE);
spcm_dwSetParam_i32 (hCard[2], SPC_TRIG_ORMASK, SPC_TM_NONE);
spcm_dwSetParam_i32 (hCard[3], SPC_TRIG_ORMASK, SPC_TM_NONE);
```

Assuming that the 4 cards are analog data acquisition cards with 4 channels each we can simply setup a synchronous system with all channels of all cards being trigger source. The following setup will show how to set up all trigger events of all channels to be OR connected. If any of the channels will now have a signal above the programmed trigger level the complete system will do an acquisition:

```
for (i = 0; i < lSyncCount; i++)
    {
    int32 lAllChannels = (SPC_TMASK0_CH0 | SPC_TMASK0_CH1 | SPC_TMASK_CH2 | SPC_TMASK_CH3);
    spcm_dwSetParam_i32 (hCard[i], SPC_TRIG_CH_ORMASK0, lAllChannels);
    for (j = 0; j < 2; j++)
        {

        // set all channels to trigger on positive edge crossing trigger level 100
        spcm_dwSetParam_i32 (hCard[i], SPC_TRIG_CH0_MODE + j, SPC_TM_POS);
        spcm_dwSetParam_i32 (hCard[i], SPC_TRIG_CH0_LEVEL0 + j, 100);
        }
    }
```

Run the synchronized cards

Running of the cards is very simple. The star-hub acts as one big card containing all synchronized cards. All card commands have to be omitted directly to the star-hub which will check the setup, do the synchronization and distribute the commands in the correct order to all synchronized cards. The same card commands can be used that are also possible for single cards:

Register	Value	Direction	Description
SPC_M2CMD	100	write only	Executes a command for the card or data transfer
M2CMD_CARD_RESET	1h	Performs a hard and software reset of the card as explained further above	
M2CMD_CARD_WRITESETUP	2h	Writes the current setup to the card without starting the hardware. This command may be useful if changing some internal settings like clock frequency and enabling outputs.	
M2CMD_CARD_START	4h	Starts the card with all selected settings. This command automatically writes all settings to the card if any of the settings has been changed since the last one was written. After card has been started none of the settings can be changed while the card is running.	
M2CMD_CARD_ENABLETRIGGER	8h	The trigger detection is enabled. This command can be either send together with the start command to enable trigger immediately or in a second call after some external hardware has been started.	
M2CMD_CARD_FORCETRIGGER	10h	This command forces a trigger even if none has been detected so far. Sending this command together with the start command is similar to using the software trigger.	
M2CMD_CARD_DISABLETRIGGER	20h	The trigger detection is disabled. All further trigger events are ignored until the trigger detection is again enabled. When starting the card the trigger detection is started disabled.	
M2CMD_CARD_STOP	40h	Stops the current run of the card. If the card is not running this command has no effect.	

All other commands and settings need to be send directly to the card that it refers to.

This example shows the complete setup and synchronization start for our four cards:

```
spcm_dwSetParam_i32 (hSync, SPC_SYNC_ENABLEMASK, 0x000F); // all 4 cards are masked

// to keep it easy we set all card to the same clock and disable trigger
for (i = 0; i < 4; i++)
    {
    spcm_dwSetParam_i32 (hCard[i], SPC_CLOCKMODE, SPC_CM_INTPLL);
    spcm_dwSetParam_i32 (hCard[i], SPC_SAMPLERATE, MEGA(100));
    spcm_dwSetParam_i32 (hCard[i], SPC_TRIG_ORMASK, SPC_TM_NONE);
    }

// card 0 is trigger master and waits for external positive edge
spcm_dwSetParam_i32 (hCard[0], SPC_TRIG_ORMASK, SPC_TMASK_EXT0);
spcm_dwSetParam_i32 (hCard[0], SPC_TRIG_EXT0_MODE, SPC_TM_POS);

// start the cards and wait for them a maximum of 1 second to be ready
spcm_dwSetParam_i32 (hSync, SPC_TIMEOUT, 1000);
spcm_dwSetParam_i32 (hSync, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER);
if (spcm_dwSetParam_i32 (hSync, SPC_M2CMD, M2CMD_CARD_WAITREADY) == ERR_TIMEOUT)
    printf ("Timeout occured - no trigger received within time\n")
```

Using one of the wait commands for the Star-Hub will return as soon as the card holding the Star-Hub has reached this state. However when synchronizing cards with different memory sizes there may be other cards that still haven't reached this level.

SH-Direct: using the Star-Hub clock directly without synchronization

Starting with driver version 1.26 build 1754 it is possible to use the clock from the Star-Hub just like an external clock and running one or more cards totally independent of the synchronized card. The mode is by example useful if one has one or more output cards that run continuously in a loop and are synchronized with Star-Hub and in addition to this one or more acquisition cards should make multiple acquisitions but using the same clock.

For all M2i cards is is also possible to run the „slave" cards with a divided clock. Therefore please program a desired divided sampling rate in the SPC_SAMPLERATE register (example: running the Star-Hub card with 10 MS/s and the independent cards with 1 MS/s). The sampling rate is automatically adjusted by the driver to the next matching value.

What is necessary?

- All cards need to be connected to the Star-Hub
- The card(s) that should run independently can not hold the Star-Hub
- The card(s) with the Star-Hub must be setup to synchronization even if it's only one card
- The synchronized card(s) have to be started prior to the card(s) that run with the direct Star-Hub clock

Setup

At first all cards that should run synchronized with the Star-Hub are set-up exactly as explained before. The card(s) that should run independently and use the Star-Hub clock need to use the following clock mode:

Register	Value	Direction	Description
SPC_CLOCKMODE	20200	read/write	Defines the used clock mode
SPC_CM_SHDIRECT	128	Uses the clock from the Star-Hub as if this was an external clock	

When using SH_Direct mode, the register call to SPC_CLOCKMODE enabling this mode must be written before initiating a card start command to any of the connected cards. Also it is not allowed to be modified later in the programming sequence to prevent the driver from calculating wrong sample rates.

Example

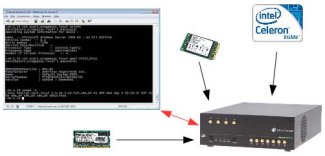
In this example we have one generator card with the Star-Hub mounted running in a continuous loop and one acquisition card running independently using the SH-Direct clock.

```
// setup of the generator card
spcm_dwSetParam_i32 (hCard[0], SPC_CARDMODE, SPC_REP_STD_SINGLE);
spcm_dwSetParam_i32 (hCard[0], SPC_LOOPS, 0);                    // infinite data replay
spcm_dwSetParam_i32 (hCard[0], SPC_CLOCKMODE, SPC_CM_INTPLL);
spcm_dwSetParam_i32 (hCard[0], SPC_SAMPLERATE, MEGA(1));
spcm_dwSetParam_i32 (hCard[0], SPC_TRIG_ORMASK, SPC_TM_SOFTWARE);

spcm_dwSetParam_i32 (hSync, SPC_SYNC_ENABLEMASK, 0x0001); // card 0 is the generator card
spcm_dwSetParam_i32 (hSync, SPC_SYNC_CLKMASK,    0x0001); // only for M2i/M3i cards: set ClkMask

// Setup of the acquisition card (waiting for external trigger)
spcm_dwSetParam_i32 (hCard[1], SPC_CARDMODE, SPC_REC_STD_SINGLE);
spcm_dwSetParam_i32 (hCard[1], SPC_CLOCKMODE, SPC_CM_SHDIRECT);
spcm_dwSetParam_i32 (hCard[1], SPC_SAMPLERATE, MEGA(1));
spcm_dwSetParam_i32 (hCard[1], SPC_TRIG_ORMASK, SPC_TMASK_EXT0);
spcm_dwSetParam_i32 (hCard[1], SPC_TRIG_EXT0_MODE, SPC_TM_POS);

// now start the generator card (sync!) first and then the acquisition card
spcm_dwSetParam_i32 (hSync, SPC_TIMEOUT, 1000);
spcm_dwSetParam_i32 (hSync, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER);

// start first acquisition
spcm_dwSetParam_i32 (hCard[1], SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER | M2CMD_CARD_WAITREADY);

// process data

// start next acquistion
spcm_dwSetParam_i32 (hCard[1], SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER | M2CMD_CARD_WAITREADY);

// process data
```

Error Handling

The Star-Hub error handling is similar to the card error handling and uses the function spcm_dwGetErrorInfo_i32. Please see the example in the card error handling chapter to see how the error handling is done.

Option Embedded Server



The option turns the digitizerNETBOX/generatorNETBOX in a powerful PC that allows to run own programs on a small and remote data acquisition system. The digitizerNETBOX/generatorNETBOX is enhanced by more memory, a powerful CPU, a freely accessable internal SSD and a remote software development access method.

The digitizerNETBOX/generatorNETBOX can either run connected to LAN or it can run totally independent, storing/replaying data to/from the internal SSD. The original digitizerNETBOX/generatorNETBOX remote instrument functionality is still 100% available. Running the embedded server option it is possible to pre-calculate results based on the acquired data, pre-calculate generator data, store acquisitions locally and to transfer just the required data or results parts in a client-server based software structure. A different example for the digitizerNETBOX embedded server is surveillance/logger application which can run totally independent for days and send notification emails only over LAN or offloads stored data as soon as it's connected again.

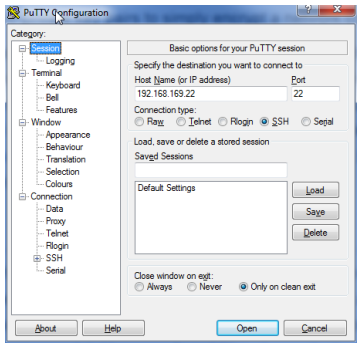
Access to the embedded server is done through a standard text based Linux shell based on the ssh secure shell.

Acessing the Embedded Server

Access to the Embedded Server is only available if that particular option is installed. As this option is a combination of hardware features and software access a later update with that options needs some factory work. As long as no one uses the embedded server connection and no programs are placed in the autostart folder, the digitizerNETBOX/generatorNETBOX will behave just like a standard digitizerNETBOX or generatorNETBOX and can be used as a remote LXI device.

SSH Connection

The embedded server is accessed using a standard SSH (secure shell) connection. Please install a SSH client on your working system and connect to the digitizerNETBOX/generatorNETBOX IP address (found in the control center) using port 22. Any SSH compatible client will do the job.



An example for a Windows based SSH client is PuTTY which shown on the right.

You may enter the login parameters here also and save a session for faster access.

Login

Login is done using a separate user space with some restricted access to the system. A login as root isn't possible due to security and system stability reasons. Please use the following default user settings:

Username	embedded
Password	embedded

After first login you should immediately change the password to a personal one using the command „passwd". Please keep in mind that it is possible to reset the password using the web interface of the digitizerNETBOX/generatorNETBOX. To fully secure access to the digitizerNETBOX/generatorNETBOX it is necessary to give a password to the web interface setup.

Mounting network folders

Network folders can be mounted and unmounted using the standard Linux mount/unmount command. Please note that you need root rights to do a mounting/unmounting of a network folder. You get root rights for this command by using the „sudo" command which gives you root rights for some dedicated commands.

Mounting a test folder from a Windows server with active directory may look like this:

```
cd
mkdir tmp
sudo mount -t cifs //192.168.169.123/tmp tmp -o user=YourUsername,domain=YourDomain,password=YourPassword
```

You may unmount the folder again with:

```
sudo umount tmp
```

Access to the /etc/fstab table is not available.

Access to NTP (Network Time Protocol)

You access NTP with (requires firmware version V34 or newer):

```
sudo /usr/sbin/sntp -s de.pool.ntp.org
```

Editors

As a default there are two standard editors installed on the system:

- GNU nano
- vim

Installing packages

Any matching RPM modules can be installed to the system using root rights and the rpm packet manager:

```
sudo rpm -ihv mypackage.rpm
```

Programming

For general information on programming of the internal Spectrum cards please have a look through the complete manual. Programming the cards inside the Embedded Server is 100% similar to programming of the cards of any other host system.

Accessing the cards

Depending on the type of digitizerNETBOX/generatorNETBOX that you have there might be one or two cards installed in the system. If two cards are installed then there is also a Star-Hub installed. Please refer to the chapter „Introduction - Internal Digitizer Modules" or „Introduction - Internal Generator Modules" respectively to see how many digitizers are installed in your digitizerNETBOX/generatorNETBOX and whether a starhub is present or not.

As an example, for a DN2.491-16 you will find the information that you have 2 cards M2i.4912-exp and one Star-Hub installed. Accessing these components is done with the following handles:

```
1st card: „/dev/spcm0"
2nd card: „/dev/spcm1"
Star-Hub: „sync0"
```

Examples

The home folder „examples-cpp" contains all Linux based examples that are currently available. Please use and modify these examples for your own programs as you like.

The sub-folder „netbox_embedded_server" contains some additional examples for using the embedded server features. The following examples are available:

Client/Server

A simple example showing the communication over TCP/IP between the digitizerNETBOX/generatorNETBOX (server) and the host PC (client). The server is running an acquisition in FIFO Multiple Recording mode and calculates minimum and maximum value from every block. These results are then sent to the client program for further processing. In our example the results are simply printed to console.

Please change the TCP/IP settings inside the client program to your local settings to get it running.

simple_rec_fifo_mail

This example will run a FIFO multi acquisition and send a mail for each acquired segment as a SBench6 - compatible binary file and text header for that file. The example can easily be modified and used as a base for a monitoring application.

Please be sure to change the email settings to a server and port settings that is available on your system.

Please keep in mind that a high trigger frequency will flood your mailserver with emails which might trigger some spam detection mechanisms. You should therefore use this example only with single trigger events.

dbus

This is an example on how to connect to the digitizerNETBOX/generatorNETBOX internal signals (currently only LAN state).

Autostart

All executable files in the autostart folder will automatically be executed on system start-up. Please place any program in here that should run automatically after powering the system. It is requested to use the „fork()" command to continue a program or a service in the background if multiple commands should be running.

The autoastart feature can be turned off using the web interface in case that some failing program prevents the machine from starting.

LEDs

The digitizerNETBOX/generatorNETBOX LEDs can be accessed using the special system command „netbox_led_client". Calling this system command from inside a C++ program is shown in the client-server example.

The following commands will manipulate the Arm/Trig and Connected LEDs on the frontplate:

```
system ("netbox_led_client armgreen=1");
system ("netbox_led_client armgreen=0");
system ("netbox_led_client conngreen=1");
system ("netbox_led_client conngreen=0");
```

Appendix

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.

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_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 timeout 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 timeout 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.P ower 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 4k in the range of 16, 32, 64, 128, 256, 512, 1k or 2k. For ABA and timestamp the notify size can be 2k as a minimum.
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.

error name	value (hex)	value (dec.)	error description
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_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	Timeout 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 occured during config writes or reads. Please contact Spectrum support for further assistance.
ERR_FIFOHWOVERRUN	301h	769	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. If acquisition speed is smaller than the theoretical bus transfer speed please check the application buffer and try to improve the handling of this one.
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 Knowledge Base

You will also find additional help and information in our knowledge base available on our website:

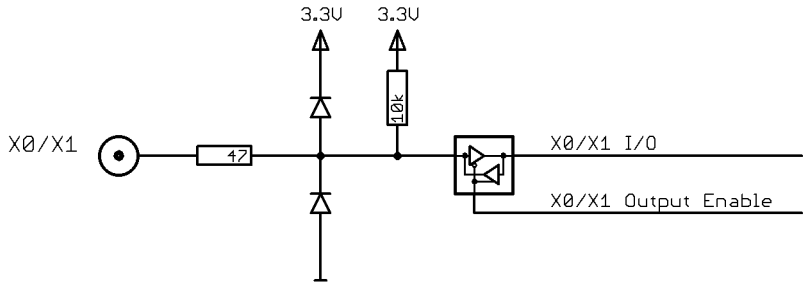
https://spectrum-instrumentation.com/en/knowledge-base-overview

Details on M3i cards I/O lines

Multi Purpose I/O Lines

The MMCX Multi Purpose I/O connectors (X0 and X1) of the M3i cards from Spectrum are protected against over voltage conditions.

For this purpose clamping diodes of the types 1N4148 are used in conjunction with a series resistor. Both I/O lines are internally clamped to signal ground and to 3.3V clamping voltage. So when connecting sources with a higher level than the clamping voltage plus the forward voltage of typically 0.6..0.7 V will be the resulting maximum high-level level.



The maximum forward current limit for the used 1N4148 diodes is 100 mA, which is effectively limited by the used series resistor for logic levels up to 5.0V. To avoid floating levels with unconnected inputs, a pull up resistor of 10 kOhm to 3.3V is used on each line.

Interfacing with clock input

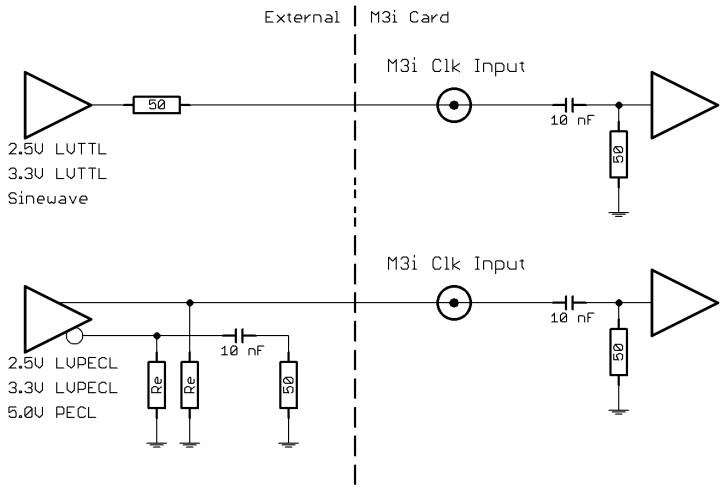
The clock input of the M3i cards is AC-coupled, single-ended PECL type. Due to the internal biasing and a relatively high maximum input voltage swing, it can be directlly connected to various logic standards, without the need for external level converters.

Single-ended LVTTL sources

All LVTTL sources, be it 2.5V LVTTL or 3.3V LVTTL must be terminated with a 50 Ohm series resistor to avoid reflections and limit the maximum swing for the M3i card.

Differential (LV)PECL sources

Differential drivers require equal load on both the true and the inverting outputs. Therefore the inverting output should be loaded as shown in the drawing. All PECL drivers require a proper DC path to ground, therefore emitter resistors R_E must be used, whose value depends on the supply voltage of the driving PECL buffer:



V_{CC} - V_{EE}	2.5 V	3.3 V	5.0 V
R_E	~50 Ohm	~100 Ohm	~200 Ohm

Interfacing with clock output

The clock output of the M3i cards is AC-coupled, single-ended PECL type. The output swing of the M3i clock output is approximately 800 mV_{PP}.

Internal biased single-ended receivers

Because of the AC coupling of the M3i clock output, the signal must be properly re-biased for the receiver. Receivers that provide an internal re-bias only require the signal to be terminated to ground by a 50 Ohm resistor.

Differential (LV)PECL receivers

Differential receivers require proper re-biasing and likely a small minimum difference between the true and the inverting input to avoid ringing with open receiver inputs. Therefore a Thevenin-equivalent can be used, with receiver-type dependant values for R1, R2, R1' and R2'.