



**MC.47xx**  
**multi channel 16 bit DAQ,**  
**A/D converter board**  
**for CompactPCI bus**

**Hardware Manual**  
**Software Driver Manual**

English version

May 24, 2018

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

SBench, digitizerNETBOX and generatorNETBOX are registered trademarks of Spectrum Instrumentation GmbH.  
Microsoft, Visual C++, Visual Basic, Windows, Windows 98, Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10 and Windows Server are trademarks/registered trademarks of Microsoft Corporation.  
LabVIEW, DASyLab, Diadem and LabWindows/CVI are trademarks/registered trademarks of National Instruments Corporation.  
MATLAB is a trademark/registered trademark of The Mathworks, Inc.  
Delphi and C++Builder are trademarks or registered trademarks of Embarcadero Technologies, Inc.  
Keysight VEE, VEE Pro and VEE OneLab are trademarks/registered trademarks of Keysight Technologies, Inc.  
FlexPro is a registered trademark of Weisang GmbH & Co. KG.  
PCIe, PCI Express, PCI-X and PCI-SIG are trademarks of PCI-SIG.  
PICMG and CompactPCI are trademarks of the PCI Industrial Computer Manufacturers Group.  
PXI is a trademark of the PXI Systems Alliance.  
LXI is a registered trademark of the LXI Consortium.  
IVI is a registered trademark of the IVI Foundation  
Oracle and Java are registered trademarks of Oracle and/or its affiliates.  
Intel and Intel Xeon are trademarks and/or registered trademarks of Intel Corporation.  
AMD and Opteron are trademarks and/or registered trademarks of Advanced Micro Devices.  
NVIDIA, CUDA, GeForce, Quadro and Tesla are trademarks and/or registered trademarks of NVIDIA Corporation.

<b>Introduction.....</b>	<b>6</b>
Preface .....	6
General Information .....	6
Different models of the MC.47xx series.....	6
Additional options .....	8
Extra I/O (Option -XMF).....	8
Starhub .....	8
Timestamp .....	9
The Spectrum type plate.....	10
Hardware information.....	11
Block diagram.....	11
Order Information.....	12
<b>Hardware Installation .....</b>	<b>13</b>
System Requirements .....	13
Warnings.....	13
ESD Precautions .....	13
Cooling Precautions.....	13
Sources of noise .....	13
Installing the board in the system.....	13
Installing a single board without any options.....	13
Installing a board with digital inputs/outputs.....	14
Installing a board with extra I/O (Option -XMF) .....	14
Installing multiple boards synchronized by starhub.....	15
Installing multiple synchronized boards .....	16
<b>Software Driver Installation .....</b>	<b>17</b>
Interrupt Sharing .....	17
Important Notes on Driver Version 4.00.....	17
Windows XP 32/64 Bit .....	18
Installation .....	18
Version control.....	18
Driver - Update.....	19
Windows Vista/7 32/64 Bit .....	20
Installation .....	20
Version control.....	21
Driver - Update.....	21
Windows NT / Windows 2000 32 Bit.....	22
Installation .....	22
Adding boards to the Windows NT / Windows 2000 driver.....	22
Driver - Update.....	22
Important Notes on Driver Version 4.00.....	22
Linux.....	23
Overview .....	23
Installation with Udev support.....	23
Installation without Udev support .....	24
<b>Software .....</b>	<b>26</b>
Software Overview.....	26
Accessing the hardware with SBench 6.....	27
C/C++ Driver Interface.....	27
Header files .....	27
Microsoft Visual C++ .....	27
Borland C++ Builder .....	28
Linux Gnu C.....	28
Other Windows C/C++ compilers .....	28
National Instruments LabWindows/CVI.....	28
Driver functions .....	29
Delphi (Pascal) Programming Interface .....	31
Type definition .....	31
Include Driver.....	31
Examples.....	31
Driver functions .....	31
Visual Basic Programming Interface .....	32
Include Driver.....	32
Visual Basic Examples .....	32
VBA for Excel Examples .....	32
Driver functions .....	32
Python Programming Interface and Examples.....	34
Driver interface .....	34
Examples.....	34

<b>Programming the Board .....</b>	<b>35</b>
Overview .....	35
Register tables .....	35
Programming examples.....	35
Error handling.....	35
Initialization.....	36
Starting the automatic initialization routine .....	36
PCI Register .....	36
Hardware version.....	37
Date of production.....	37
Serial number .....	37
Maximum possible sample rate .....	37
Installed memory .....	37
Installed features and options .....	37
Used interrupt line .....	38
Used type of driver .....	38
Powerdown and reset .....	39
<b>Analog Inputs.....</b>	<b>40</b>
Channel Selection .....	40
Important note on channels selection.....	40
Setting up the inputs .....	41
Input ranges.....	41
Automatic on-board calibration of the offset and gain settings.....	42
<b>Standard acquisition modes .....</b>	<b>43</b>
General Information .....	43
Programming .....	43
Memory, Pre- and Posttrigger .....	43
Starting without interrupt (classic mode).....	44
Starting with interrupt driven mode .....	45
Data organization .....	45
Sample format.....	46
Reading out the data with SpcGetData.....	46
<b>FIFO Mode.....</b>	<b>48</b>
Overview .....	48
General Information.....	48
Background FIFO Read .....	48
Speed Limitations.....	48
Programming .....	49
Software Buffers .....	49
Buffer processing .....	50
FIFO mode .....	51
Example FIFO acquisition mode .....	51
Data organization .....	52
Sample format.....	52
<b>Clock generation .....</b>	<b>53</b>
Overview .....	53
Internally generated sample rate .....	53
Standard internal sample rate .....	53
Using plain quartz without PLL.....	54
External reference clock .....	54
External clocking .....	55
Direct external clock .....	55
External clock with divider .....	56
<b>Trigger modes and appendant registers .....</b>	<b>57</b>
General Description.....	57
Software trigger .....	57
External TTL trigger .....	57
Edge triggers .....	58
Pulsewidth triggers.....	59
Channel Trigger .....	60
Overview of the channel trigger registers.....	60
Triggerlevel.....	61
Detailed description of the channel trigger modes.....	63

<b>Option Timestamp .....</b>	<b>69</b>
General information .....	69
Limits .....	69
Timestamp modes.....	69
Standard mode .....	69
StartReset mode.....	69
RefClock mode (optional) .....	70
Timestamp Status.....	70
Reading out timestamp data .....	70
Functions for accessing the data .....	70
Data format .....	71
Example programs .....	72
Standard acquisition mode .....	72
Acquisition with Multiple Recording .....	72
<b>Option Extra I/O .....</b>	<b>73</b>
Digital I/Os.....	73
Channel direction .....	73
Transfer Data .....	73
Analog Outputs.....	74
Programming example.....	74
<b>Synchronization (Option) .....</b>	<b>75</b>
The different synchronization options .....	75
Synchronization with option cascading .....	75
Synchronization with option starhub .....	75
The setup order for the different synchronization options .....	76
Setup Order for use with standard (non FIFO) mode and equally clocked boards .....	76
Setup synchronization for use with FIFO mode and equally clocked boards .....	80
Additions for synchronizing different boards .....	82
Additions for equal boards with different sample rates .....	83
Resulting delays using different boards or speeds .....	84
<b>Appendix .....</b>	<b>85</b>
Error Codes .....	85
Pin assignment of the multipin connector .....	86
Extra I/O with external connector(Option -XMF) .....	86
Pin assignment of the multipin cable .....	86
IDC footprints.....	87

# Introduction

## Preface

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

In addition, this guide takes you through the process of installing your board 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 board and the related driver. The reader of this manual will be able to integrate the board in any PC system with one of the supported bus and operating systems.

Please note that this manual provides no description for specific driver parts such as those for LabVIEW or MATLAB. These drivers are provided by special order.

For any new information on the board as well as new available options or memory upgrades please contact our website [www.spectrum-instrumentation.com](http://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 MC.47xx enables you for the first time to acquire 16 channels with full 16 bit resolution synchronously on one board.

Every channel has its own amplifier and A/D converter. This eliminates the problems known from multiplexed systems like phase error between the channels or high crosstalk. Every input channel could be completely calibrated using the software. The user will easily find a matching solution from the 6 offered models. These versions are working with sample rates of 100 kS/s, 250 kS/s or 500 kS/s. The boards have 8 or 16 channels and can also be updated to a multi channel system using the internal synchronization bus.

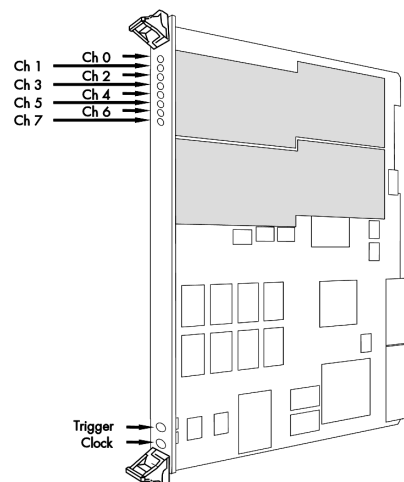
Data is written in the internal 8 MSample up to 256 MSample large memory. This memory can also be used as a FIFO buffer. In FIFO mode data will be transferred online into the PC RAM or to hard disk.

**Application examples: Automatic test systems, Supersonics, CCD imaging systems, Vibration analysis, Radar, Sonar.**

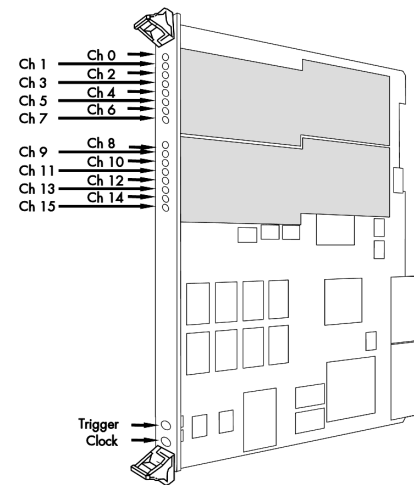
## Different models of the MC.47xx series

The following overview shows the different available models of the MC.47xx series. They differ in the number of mounted acquisition modules and the number of available channels. You can also see the model dependant allocation of the input connectors.

- **MC.4710**
- **MC.4720**
- **MC.4730**



- **MC.4711**
- **MC.4721**
- **MC.4731**



## Additional options

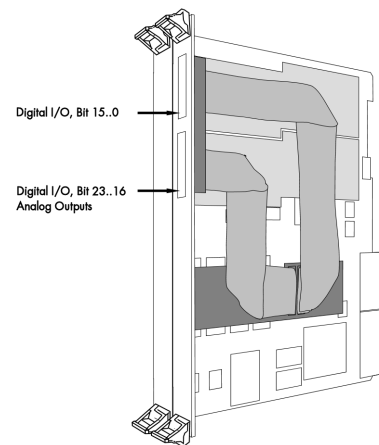
### Extra I/O (Option -XMF)

With this simple-to-use enhancement it is possible to control a wide range of external instruments or other equipment. Therefore you have 24 digital I/O and the 4 analog outputs available.

The extra I/O option is useful if an external amplifier should be controlled, any kind of signal source must be programmed, an antenna must be adjusted, a status information from external machine has to be obtained or different test signals have to be routed to the board.

The additional inputs and outputs are mounted on an extra bracket. The figure shows the allocation of the two connectors. The shown option is mounted exemplarily on a board with two modules. Of course you can also combine this option as well with a board that is equipped with only one module.

**It is not possible to use this option together with the star hub or timestamp option, because there is just space for one piggyback module on the on-board expansion slot.**



### Starhub

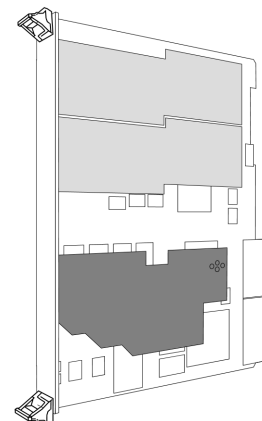
The star hub module allows the synchronisation of up to 16 MC boards. It is possible to synchronise boards of the same type with each other as well as different types.

The module acts as a star hub for clock and trigger signals. Each board is connected with a small cable of the same length, even the master board. That minimises the clock skew between the different boards. The figure shows the piggyback module mounted on the base board schematically without any cables to achieve a better visibility.

Any board could be the clock master and the same or any other board could be the trigger master. All trigger modes that are available on the master board are also available if the synchronisation star hub is used.

The cable connection of the boards is automatically recognised and checked by the driver at load time. So no care must be taken on how to cable the boards. The programming of the star hub is included in the standard board interface and consists of only 3 additional commands.

**It is not possible to use this option together with the timestamp or extra I/O option, because there is just space for one piggyback module on the on-board expansion slot.**





## Timestamp

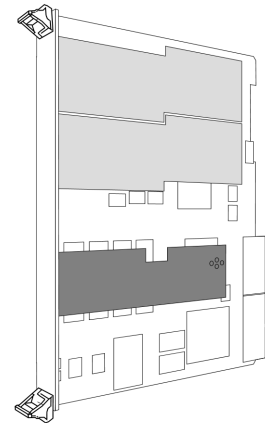
The timestamp module was designed to record the exact time information between trigger events.

The timestamp reset command sets an internal counter to zero. The counter is running with the same resolution as the sample rate. On each trigger event a timestamp is recorded in an extra FIFO. The recorded timestamps are read out asynchronously to the board sampling.

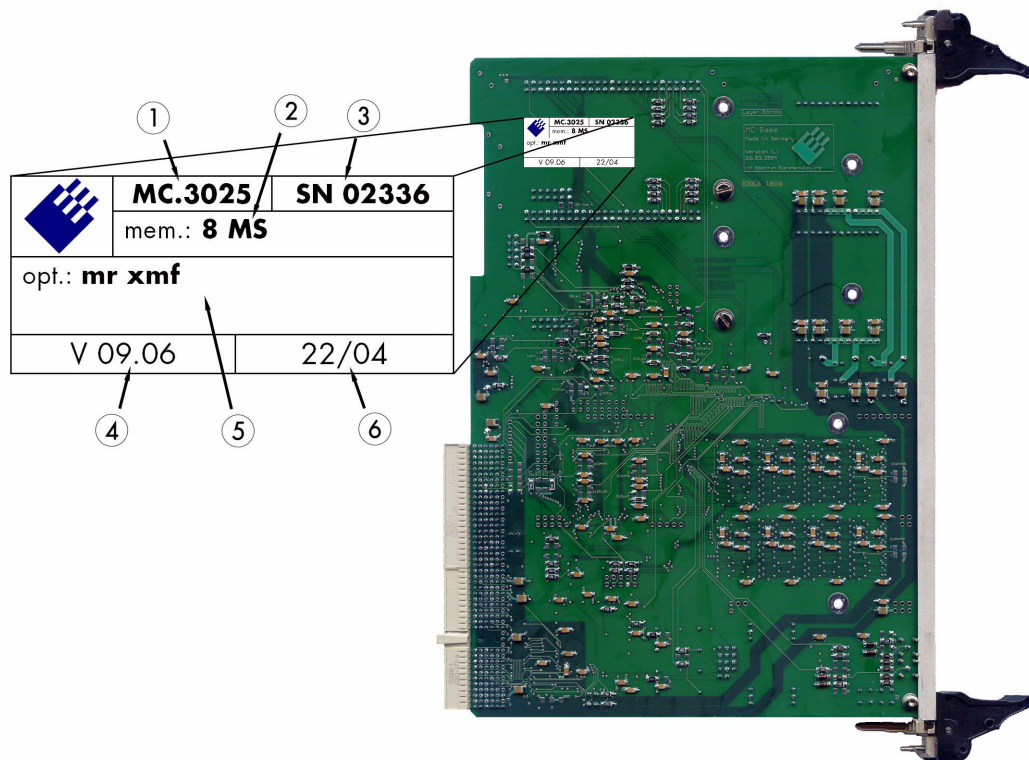
If the absolute time information is of interest it is possible to synchronise the timestamp counter with a 1 Hz "seconds" signal of a radio clock or a GPS receiver. In that case the 64 bit timestamp information is split up in two parts. The one part counts the number of seconds starting with the reset command, the other part is set to zero on every rising edge of the seconds signal and specifies the exact time position in relation to the seconds signal.

The figure shows the piggyback module installed on the on-board expansion slot. The shown option is mounted exemplarily on a board with two modules.

**It is not possible to use this option together with the star hub or extra I/O option, because there is just space for one piggyback module on the on-board expansion slot.**



## The Spectrum type plate



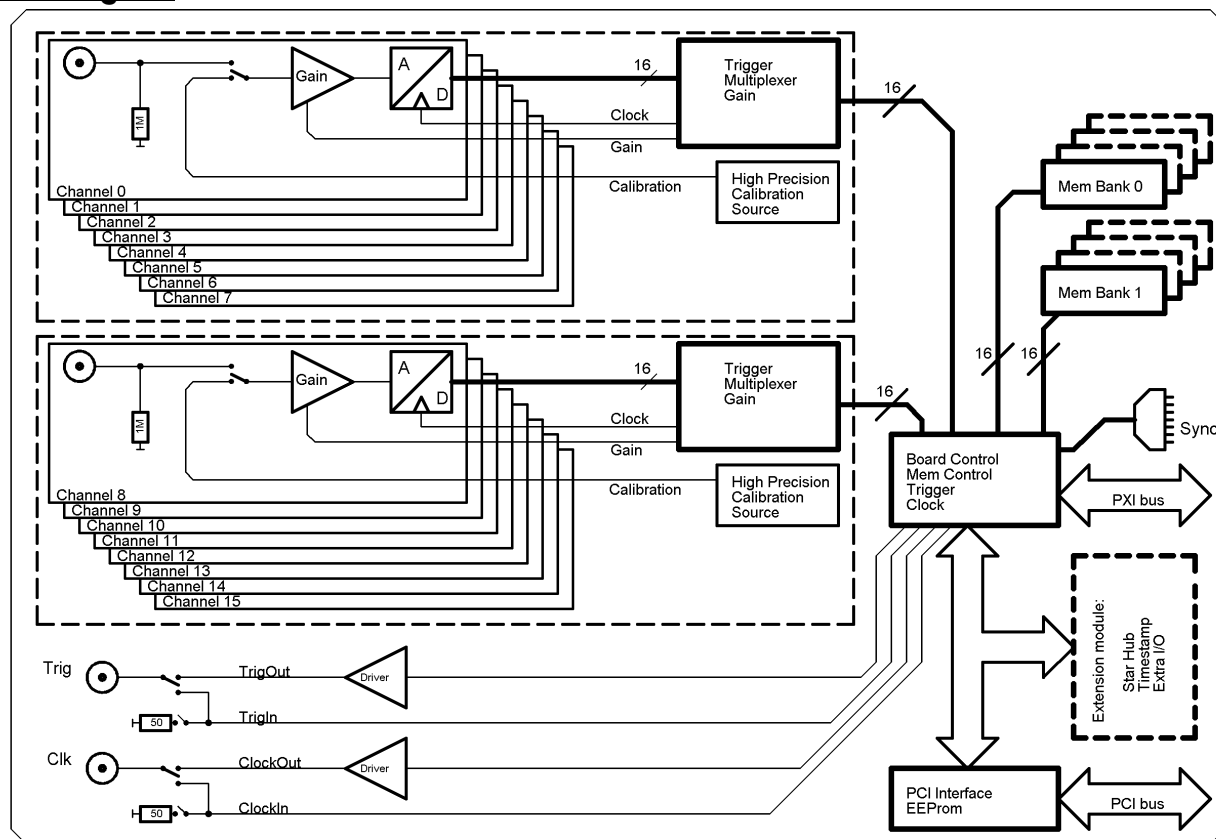
The Spectrum type plate, which consists of the following components, can be found on all of our boards.

- ① The board type, consisting of the two letters describing the bus (in this case MC for the CompactPCI bus) and the model number.
- ② The size of the on-board installed memory in MSamples. In this example there are 8 MS (16 MByte) installed.
- ③ The serial number of your Spectrum board. Every board has a unique serial number.
- ④ The board revision, consisting of the base version and the module version.
- ⑤ A list of the installed options. A complete list of all available options is shown in the order information. In this example the options 'Multiple recording' and 'Extra I/O with external outputs' are installed.
- ⑥ The date of production, 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



### Technical Data

#### Analog Inputs

Resolution	16 bit ( $\pm 32000$ values)
Differential non linearity (DNL)	$\pm 1$ LSB (ADC)
Integral non linearity (INL)	$\pm 3$ LSB (ADC)
Offset error (full speed)	$\leq 0.1\%$ of range (after warm-up and calibration)
Gain error (full speed)	$\leq 0.1\%$ (after warm-up and calibration)
Fixed input mode	bipolar
Crosstalk: all ranges 100 kHz signal	-100 dB
Analog Input impedance	1 MOhm against GND
Over voltage protection	$\pm 30$ V all ranges (activated card)
Aliasing filter	Butterworth filter 2nd order

Connector (analog)	MMCX female
Connector (trigger/clock)	3 mm SMB male

Power consumption (max speed)	3,3 V	5 V	-12 V	+12 V	Total
MC.47x0 (32 MS memory)	1.0 A	1.7 A	-	-	11.8 W
MC.47x1 (32 MS memory)	1.1 A	2.9 A	-	-	18.1 W
MC.4731 (256 MS memory), max power	2.7 A	2.9 A	-	-	23.4 W

Trigger input: Standard TTL level	Low: $-0.5 > \text{level} < 0.8$ V High: $2.0 \text{ V} > \text{level} < 5.5$ V Trigger pulse must be valid $\geq 2$ clock periods.
Trigger output	Standard TTL, capable of driving 50 Ohm. Low $< 0.4$ V (@ 20 mA, max 64 mA) High $> 2.4$ V (@ -20 mA, max -48 mA) One positive edge after the first internal trigger

Ext. clock: delay to internal clock	42 ns $\pm$ 2 ns
-------------------------------------	------------------

#### Trigger

Multi: Trigger to 1st sample delay	fixed
Multi: Recovery time	$< 20$ samples
ext. Trigger accuracy	1 Sample
int. Trigger accuracy	1 Sample
input signal with 50 ohm termination	max 5 V rms
Trigger output delay	1 Sample

#### Environmental and Physical details

Dimension	160 mm x 233 mm (Standard 6U)
Width (standard board)	1 slot
Width (with star hub)	2 slots

Warm up time	10 minutes
Operating temperature	0°C to 50°C
Storage temperature	-10°C to 70°C
Humidity	10% to 90%
MTBF	80000 hours

#### Certifications and Compliances

EMC Immunity	Compliant with CE Mark
EMC Emission	Compliant with CE Mark

Clock input: Standard TTL level	Low: $-0.5 \text{ V} > \text{level} < 0.8$ V High: $2.0 \text{ V} > \text{level} < 5.5$ V Rising edge. Duty cycle: 50% $\pm$ 5%
Clock output	Standard TTL, capable of driving 50 Ohm Low $< 0.4$ V (@ 20 mA, max 64 mA) High $> 2.4$ V (@ -20 mA, max -48 mA)

## Dynamic Parameters

	MC.4710	MC.4711	MC.4720	MC.4721	MC.4730	MC.4731
max internal or external clock	100 kS/s		250 kS/s		500 kS/s	
-3 dB bandwidth	>50 kHz		>125 kHz		>250 kHz	
RMS zero noise level ( $\geq \pm 500$ mV)	< 0.7 LSB	< 0.8 LSB	< 0.8 LSB	< 0.9 LSB	< 0.9 LSB	< 1.0 LSB
RMS zero noise level ( $< \pm 500$ mV)	< 6 $\mu$ V	< 7 $\mu$ V	< 7 $\mu$ V	< 8 $\mu$ V	< 10 $\mu$ V	< 13 $\mu$ V
Test - sampling rate	100 kS/s		250 kS/s		500 kS/s	
Test signal frequency	10 kHz		10 kHz		10 kHz	
SNR (typ)	91.5 dB	91.2 dB	90.6 dB	90.5 dB	88.7 dB	88.5 dB
THD (typ)	-101.3 dB	-101.2 dB	-100.5 dB	-100.5 dB	-92.5 dB	-92.5 dB
SFDR (typ), excl. harm.	108.8 dB	108.9 dB	106.7 dB	106.8 dB	104.5 dB	104.3 dB
ENOB (based on SNR)	14.9 bit	14.8 bit	14.7 bit	14.7 bit	14.4 bit	14.4 bit
ENOB (based on SINAD)	14.7 bit	14.6 bit	14.6 bit	14.6 bit	14.3 bit	14.2 bit

Dynamic parameters are measured at  $\pm 5$  V input range (if no other range is stated) and 1 MOhm termination with the sampling rate specified in the table. Measured parameters are averaged 20 times to get typical values. Test signal is a pure sine wave of the specified frequency with  $> 99\%$  amplitude. 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. For a detailed description please see application note 002.

## Order Information

The card is delivered with 32 MSample on-board memory and supports standard mode (Scope) and FIFO mode (streaming). Operating system drivers for Windows/Linux 32 bit and 64 bit, examples for C/C++, LabVIEW (Windows), MATLAB (Windows), LabWindows/CVI, Delphi, Visual Basic, Python and a Base license of the oscilloscope software SBench 6 are included. Drivers for other 3rd party products like VEE or DASyLab may be available on request.

**Adapter cables are not included. Please order separately!**

### Versions

Order no.	Standard mem	1 channel	2 channels	4 channels	8 channels	16 channels
MC.4710	32 MSample	100 kS/s	100 kS/s	100 kS/s	100 kS/s	
MC.4711	32 MSample	100 kS/s	100 kS/s	100 kS/s	100 kS/s	100 kS/s
MC.4720	32 MSample	250 kS/s	250 kS/s	250 kS/s	250 kS/s	
MC.4721	32 MSample	250 kS/s	250 kS/s	250 kS/s	250 kS/s	250 kS/s
MC.4730	32 MSample	500 kS/s	500 kS/s	500 kS/s	500 kS/s	
MC.4731	32 MSample	500 kS/s	500 kS/s	500 kS/s	500 kS/s	500 kS/s

### Memory

Order no.	Option
MC.4xxx-64M	Memory upgrade to 64 MSample (128 MB) of total memory
MC.4xxx-128M	Memory upgrade to 128 MSample (256 MB) of total memory
MC.4xxx-256M	Memory upgrade to 256 MSample (512 MB) of total memory
MC.4xxx-up	Additional fee for later memory upgrade

### Options

Order no.	Option
MC.4xxx-cs	Option Cascading: Synchronization of up to 4 cards (one option needed per system)
MC.4xxx-sm0d (1)	Option Star-Hub: Synchronization of up to 16 cards (one option needed per system)
MC.4xxx-time (1)	Option Timestamp: Recording of trigger timestamps in an extra memory
MC.4xxx-xmf (1)	Option Extra I/O with external connector, 24 digital I/O + 4 analog outputs. Including one cable Cab-d40-100.

### Cables

for Connections	Length	Order no. to BNC male	to BNC female	to SMA male	to SMA female	to SMB female
Analog Inputs	80 cm	Cab-1m-9m-80	Cab-1m-9f-80	Cab-1m-3mA-80	Cab-1m-3fA-80	Cab-1m-3f-80
Analog Inputs	200 cm	Cab-1m-9m-200	Cab-1m-9f-200	Cab-1m-3mA-200	Cab-1m-3fA-200	Cab-1m-3f-200
Probes (short)	5 cm		Cab-1m-9f-5			
Trigger/Clock I/O	80 cm	Cab-3f-9m-80	Cab-3f-9f-80	Cab-3f-3mA-80	Cab-3f-3fA-80	Cab-3f-3f-80
Trigger/Clock I/O	200 cm	Cab-3f-9m-200	Cab-3f-9f-200	Cab-3f-3mA-200	Cab-3f-3fA-200	Cab-3f-3f-200

### Software SBench6

Order no.	
SBench6	Base version included in delivery. Supports standard mode for one card.
SBench6-Pro	Professional version for one card: FIFO mode, export/import, calculation functions
SBench6-Multi	Option multiple cards: Needs SBench6-Pro. Handles multiple synchronized cards in one system.
Volume Licenses	Please ask Spectrum for details.

<sup>(1)</sup> : Just one of the options can be installed on a card at a time.

<sup>(2)</sup> : Third party product with warranty differing from our export conditions. No volume rebate possible.

# Hardware Installation

## System Requirements

All Spectrum MC.xxxx instrumentation boards are compliant to the CompactPCI 6U standard and require in general one free slot. Depending on the installed options additional free slots can be necessary.

## Warnings

### ESD Precautions

The boards of the MC.xxxx series contain electronic components that can be damaged by electrostatic discharge (ESD).

**Before installing the board in your system or even before touching it, it is absolutely necessary to bleed of any electrostatic electricity.**



### Cooling Precautions

The boards of the MC.xxxx series operate with components having very high power consumption at high speeds. For this reason it is absolutely required to cool this board sufficiently. It is strongly recommended to install an additional cooling fan producing a stream of air across the boards surface. In most cases CompactPCI systems are already equipped with sufficient cooling power. In that case please make sure that the air stream is not blocked.

During longer pauses between the single measurements the power down mode should be called to reduce the heat production.

### Sources of noise

The boards of the MC.xxxx series should be placed far away from any noise producing source (like e.g. the power supply). It should especially be avoided to place the board in the slot directly adjacent to another fast board (like the graphics controller).

## Installing the board in the system

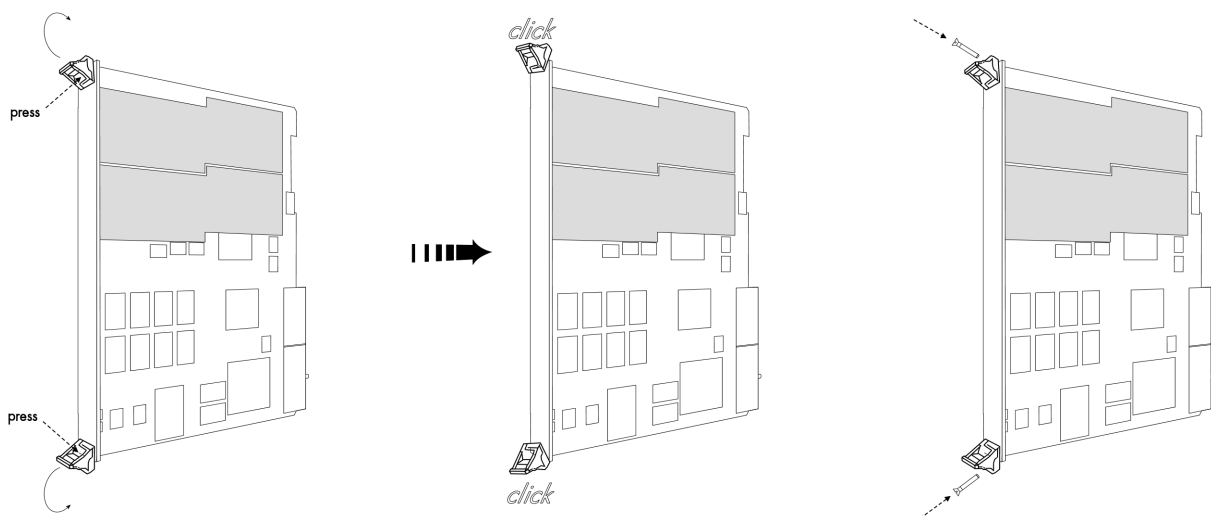
### Installing a single board without any options

The locks on the top and bottom side of CompactPCI boards need to be unlocked and opened before installing the board into a free slot of the system. Therefore you need to press the little buttons on the inside of the fasteners and move them outwards (see figure). Now slowly insert the card into the host system using the key ways until both locks snap in with a „click“.

**While inserting the board take care not to tilt it.**



After the board's insertion fasten the two screws carefully, without overdoing.



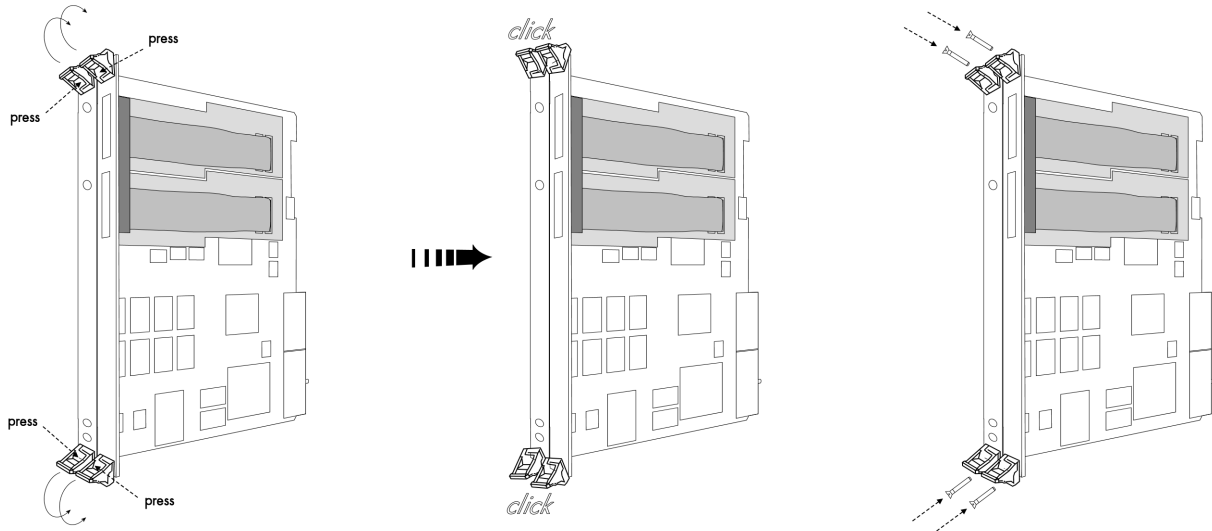
### Installing a board with digital inputs/outputs

The locks on the top and bottom side of both CompactPCI brackets need to be unlocked and opened before installing the board into a free slot of the system. Therefore you need to press the little buttons on the inside of the fasteners and move them outwards (see figure). Now slowly insert the card into the host system using the key ways until both locks snap in with a „click“.



**While inserting the board take care not to tilt it.**

After the board's insertion fasten the four screws of both brackets carefully, without overdoing. The figure shows an example of a board with two installed modules.



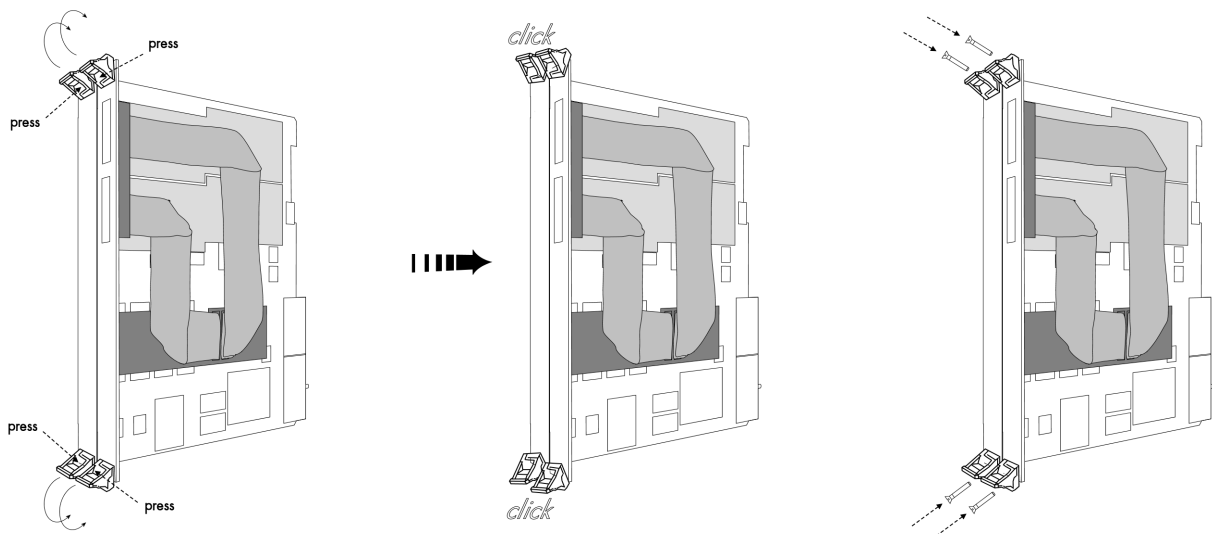
### Installing a board with extra I/O (Option -XMF)

The locks on the top and bottom side of both CompactPCI brackets need to be unlocked and opened before installing the board into a free slot of the system. Therefore you need to press the little buttons on the inside of the fasteners and move them outwards (see figure). Now slowly insert the card into the host system using the key ways until both locks snap in with a „click“.



**While inserting the board take care not to tilt it.**

After the board's insertion fasten the four screws of both brackets carefully, without overdoing. The figure shows exemplarily a board with two installed modules.



## Installing multiple boards synchronized by starhub

### Hooking up the boards

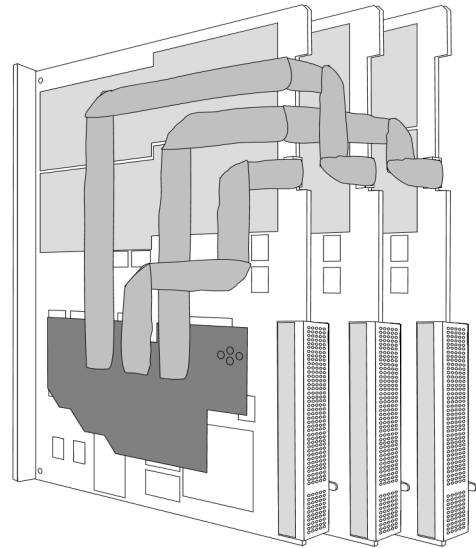
Before mounting several synchronized boards for a multi channel system into the chassis you have to hook up the boards with their synchronization cables first. Spectrum ships the boards together with the needed amount of synchronization cables. All of them are matched to the same length, to achieve a zero clock delay between the boards.

#### Only use the included flat ribbon cables.

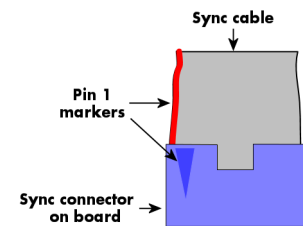
All of the boards, including the board that carries the starhub piggy-back module, must be wired to the starhub as the figure is showing exemplarily for three synchronized boards.

As you can see, all boards have a notch to get the cables to the other boards. Please only use these notches to lay the cables to avoid damage to the cables when inserting the boards into the host system.

It does not matter which of the 16 connectors on the starhub module you use for which board. The software driver will detect the types and order of the synchronized boards automatically. The right figure shows the three cables mounted next to each other only to achieve a better visibility.



As some of the synchronization cables are not secured against wrong plugging you should take care to have the pin 1 markers on the multiple connectors and the cable on the same side, as the figure on the right is showing.



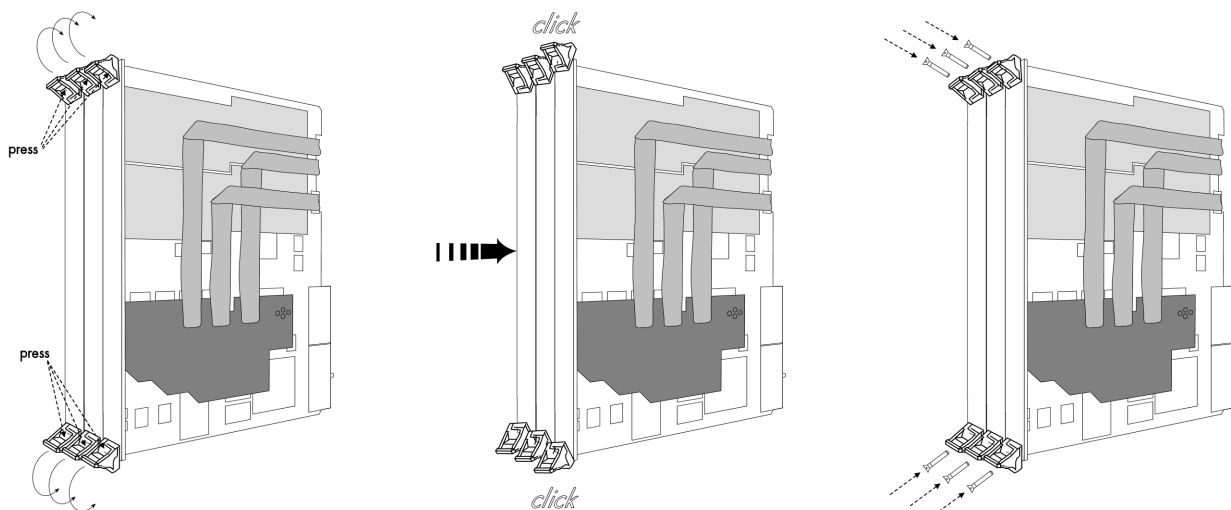
### Mounting the wired boards

The locks on the top and bottom side of all CompactPCI brackets need to be unlocked and opened before installing the boards into the slots of the system. Therefore you need to press the little buttons on the inside of the fasteners and move them outwards (see figure). Now slowly insert the boards into the host system using the key ways until both locks snap in with a „click“.

**While inserting the boards take care not to cant them and make sure that the cables are not squeezed by the backplane or any other components.**



After the board's insertion fasten the screws of all brackets carefully, without overdoing. The figure shows exemplarily a board with two installed modules.



## Installing multiple synchronized boards

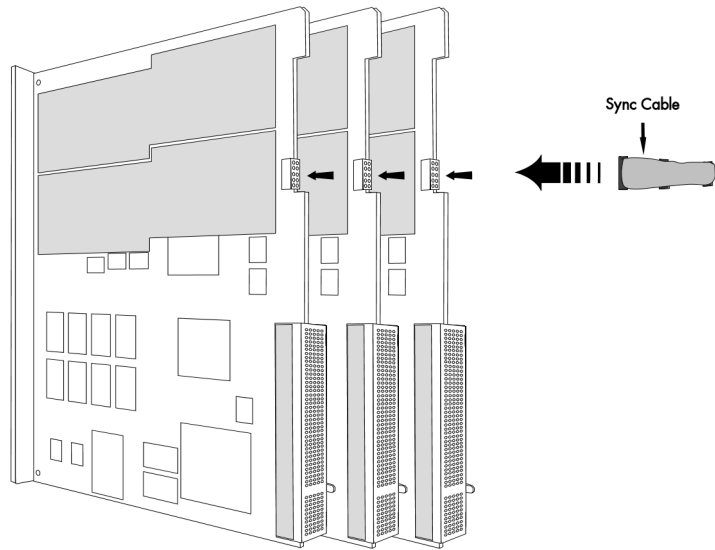
### Hooking up the boards

Before mounting several synchronized boards for a multi channel system into the chassis you have to hook up the boards with their synchronization cables first. Spectrum ships the boards together with the needed synchronization cable.

All of the possible four boards must be wired with delivered synchronization cable. The figure is showing that exemplarily for three synchronized boards. As you can see, all boards have a notch to get the cables from one board to the other. Please take care that the cable lays within these notches to avoid damages to the cable when inserting the boards into the host system.

The outer boards have a soldered termination for the sync bus. These boards are marked with an additional sticker.

**Only mount the cluster of synchronized boards in a row with the dedicated boards on the outer sides.**



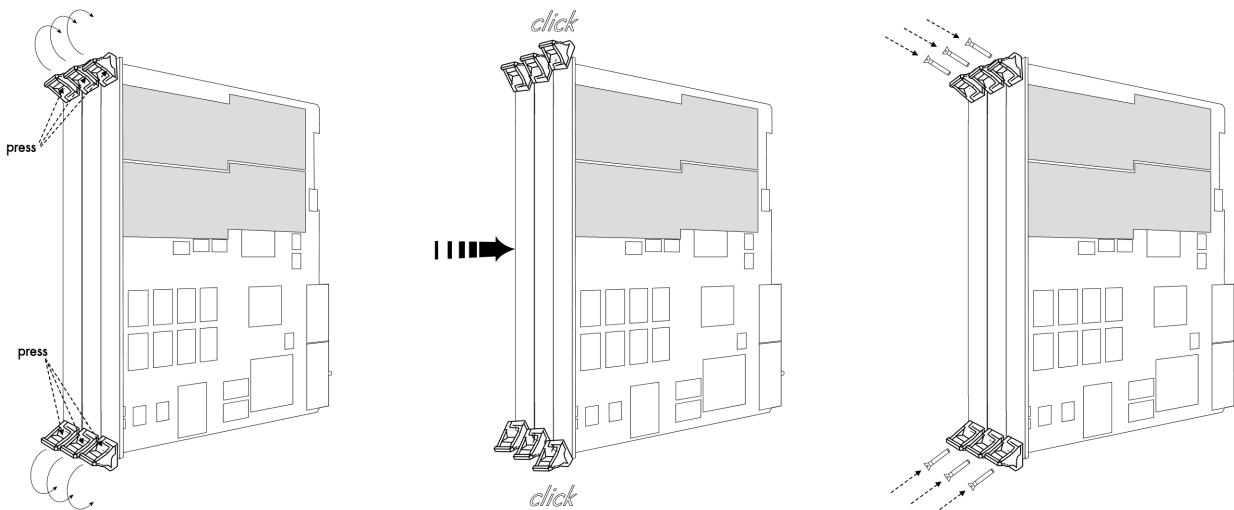
### Mounting the wired boards

The locks on the top and bottom side of all CompactPCI brackets need to be unlocked and opened before installing the boards into the slots of the system. Therefore you need to press the little buttons on the inside of the fasteners and move them outwards (see figure). Now slowly insert the boards into the host system using the key ways until both locks snap in with a „click“.



**While inserting the boards take care not to cant them and make sure that the cable is not squeezed by the backplane or any other components.**

After the board's insertion fasten the screws of all brackets carefully, without overdoing. The figure shows exemplarily a board with two installed modules.





## Software Driver Installation

Before using the board a driver 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 CD supports all boards of the MI, MC and MX series. That means that you can use the same driver for all boards of these families.

### Interrupt Sharing

This board uses a PCI interrupt for DMA data transfer and for controlling the FIFO mode. The used interrupt line is allocated by the PC BIOS at system start and is normally depending on the selected slot. Because there is only a limited number of interrupt lines available on the PCI bus it can happen that two or more boards must use the same interrupt line. This so called interrupt sharing must be supported by all drivers of the participating equipment.

Most available drivers and also the Spectrum driver for your board can manage interrupt sharing. But there are also some drivers on the market that can only use one interrupt exclusively. If this equipment shares an interrupt with the Spectrum board, the system will hang up if the second driver is loaded (the time is depending on the operating system).

If this happens it is necessary to reconfigure the system in that way that the critical equipment has an exclusive access to an interrupt.

On most systems the BIOS shows a list of all installed PCI boards with their allocated interrupt lines directly after system start. You have to check whether an interrupt line is shared between two boards. Some BIOS allow the manual allocation of interrupt lines. Have a look in your mainboard manual for further information on this topic.

Because normally the interrupt line is fixed for one PCI slot it is simply necessary to use another slot for the critical board to force a new interrupt allocation. You have to search a configuration where all critical boards have only exclusive access to one interrupt.

Depending on the system, using the Spectrum board with a shared interrupt may degrade performance a little. Each interrupt needs to be checked by two drivers. For this reason when using time critical FIFO mode even the Spectrum board should have an exclusively access to one interrupt line.

### Important Notes on Driver Version 4.00

With Windows driver version V4.00 and later the support for Windows 64 bit versions was added for MI, MC and MX series cards. This required an internal change such that Windows 98, Windows ME, and Windows 2000 versions are no longer compatible with the WDM driver version.

**Windows 98 and Windows ME should use the latest 3.39 driver version (delivered on CD revision 3.06), because with driver version V4.00 on these two operating systems are no longer supported.**



Windows 2000 users can alternatively change from the existing WDM driver to the Windows NT legacy driver, which is still supported by Spectrum.

**Because changing from one driver model (WDM) to another (NT legacy) might result in conflicts please contact Spectrum prior to the update.**



## Windows XP 32/64 Bit

### Installation

When installing the board in a Windows XP system the Spectrum board will be recognized automatically on the next start-up.

The system offers the direct installation of a driver for the board.

**Do not let Windows automatically search for the best driver, because sometimes the driver will not be found on the CD. Please take the option of choosing a manual installation path instead.**

Allow Windows XP to search for the most suitable driver in a specific directory. Select the CD that was delivered with the board as installation source. The driver files are located on CD in the directory  
 \Driver\win32\winxp\_vista\_7 for Windows Vista/7 (for 32 Bit)  
 or  
 \Driver\win64\winxp\_vista\_7 for Windows Vista/7 (for 64 Bit)

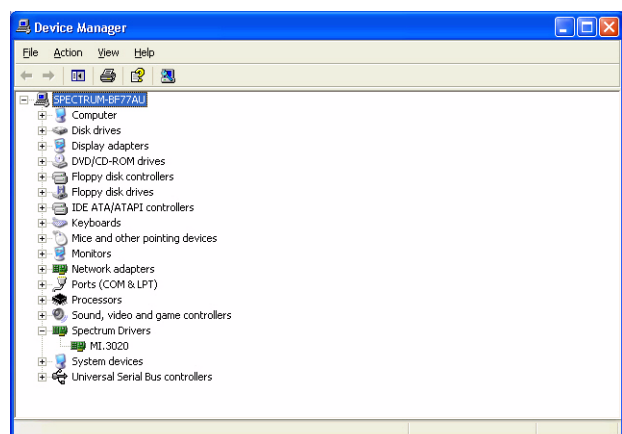
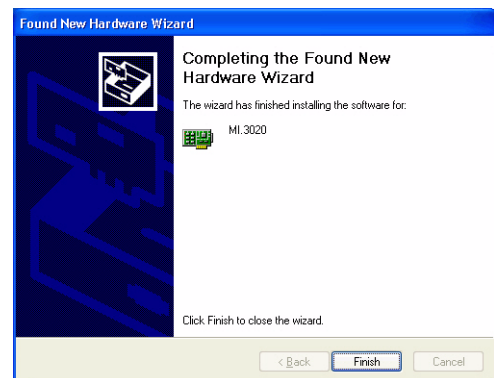
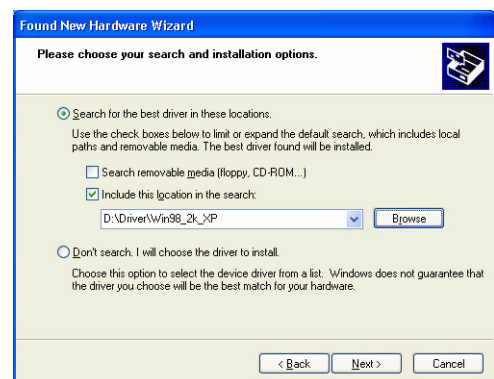
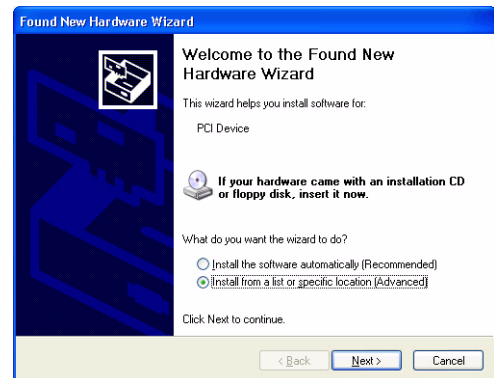
The hardware assistant shows you the exact board type that has been found like the MI.3020 in the example. Older boards (before June 2004) show „Spectrum Board“ instead.

The drivers can be used directly after installation. It is not necessary to restart the system. The installed drivers are linked in the device manager.

Below you'll see how to examine the driver version and how to update the driver with a newer version.

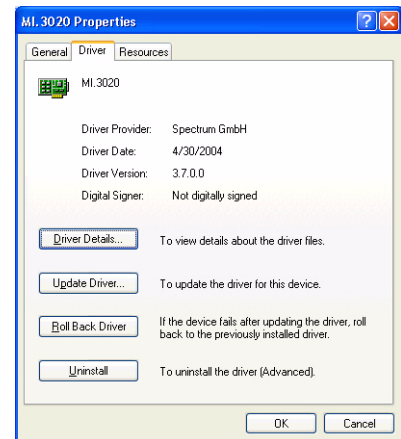
### Version control

If you want to check which driver version is installed in the system this can be easily done in the device manager. Therefore please start the device manager from the control panel and show the properties of the installed driver.



On the property page Windows XP shows the date and the version of the installed driver.

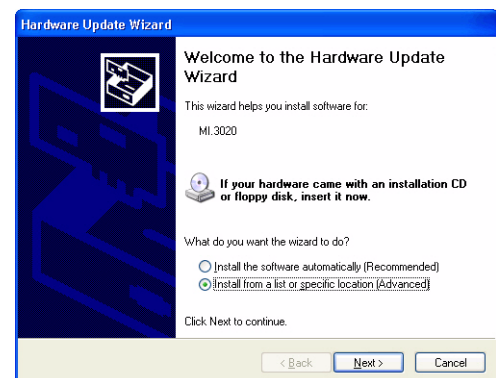
After clicking the driver details button the detailed version information of the driver is shown. In the case of a support question this information must be presented together with the board's serial number to the support team to help finding a fast solution.



## Driver - Update

If a new driver version should be installed no Spectrum board is allowed to be in use by any software. So please stop and exit all software that could access the boards.

A new driver version is directly installed from the device manager. Therefore please open the properties page of the driver as shown in the section before. As next step click on the update driver button and follow the steps of the driver installation in a similar way to the previous board and driver installation.

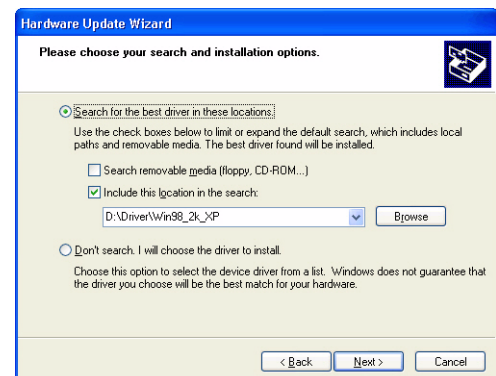


Please select the path where the new driver version was unzipped to. If you've got the new driver version on CD please select the proper path on the CD containing the new driver version:

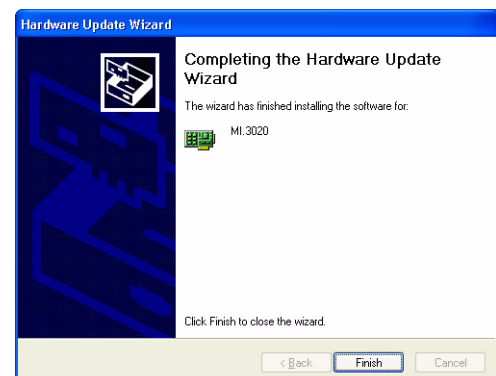
\Driver\win32\winxp\_vista\_7 for Windows Vista/7 (for 32 Bit)

or

\Driver\win64\winxp\_vista\_7 for Windows Vista/7 (for 64 Bit)



The new driver version can be used directly after installation without restarting the system. Please keep in mind to update the driver of all installed Spectrum boards.



## Windows Vista/7 32/64 Bit

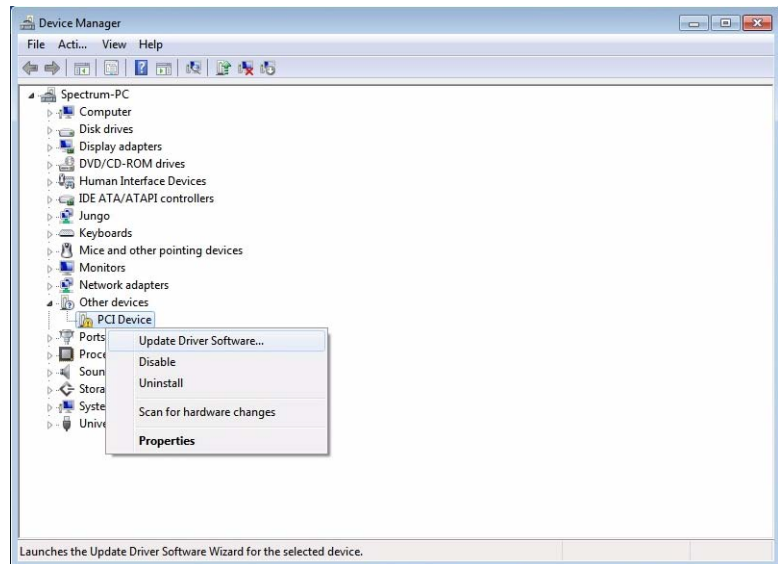
### Installation

When installing the card in a Windows Vista or Windows 7 system, it might be recognized automatically on the next start-up. The system tries at first to automatically search and install the drivers from the Microsoft homepage.

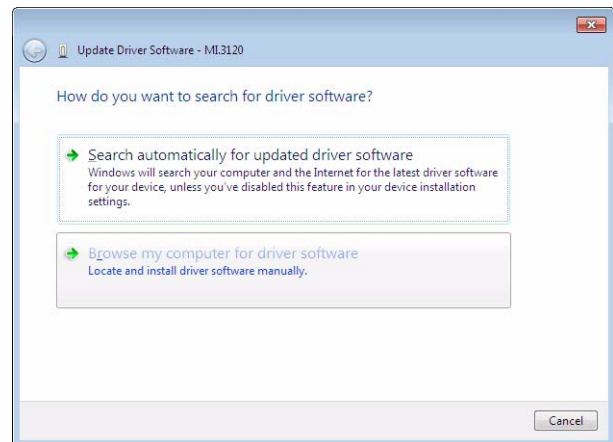
This mechanism will fail at first for the „PCI Device“ device, because the Spectrum drivers are not available via Microsoft, so simply close the dialog. This message can be safely ignored.

Afterwards open the device manager from the Windows control panel, as shown on the right.

Find the above mentioned „PCI Device“, right-click and select „Update Driver Software...“

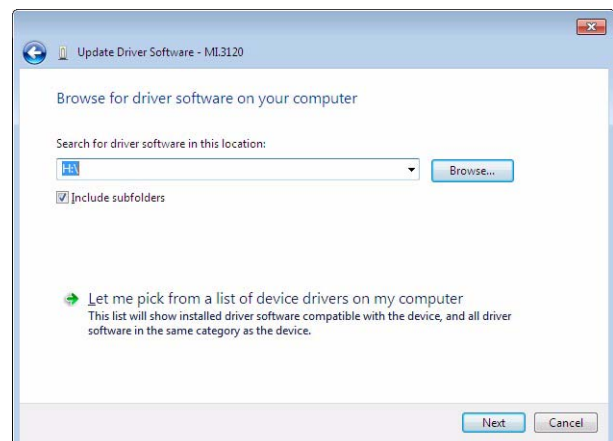


Do not let Windows Vista/7 automatically search for the best driver, because it will search the internet and not find a proper driver. Please take the option of browsing the computer manually for the driver software instead. Allow Windows Vista/7 to search for the most suitable driver in a specific directory.

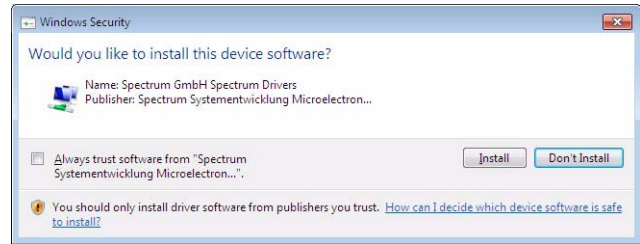


Now simply select the root folder of the CD that was delivered with the board as installation source and enable the „Include subfolders“ option.

Alternatively you can browse to the installations folders. The driver files are located on CD in the directory  
 \Driver\win32\winxp\_vista\_7 for Windows Vista/7 (for 32 Bit)  
 or  
 \Driver\win64\winxp\_vista\_7 for Windows Vista/7 (for 64 Bit)



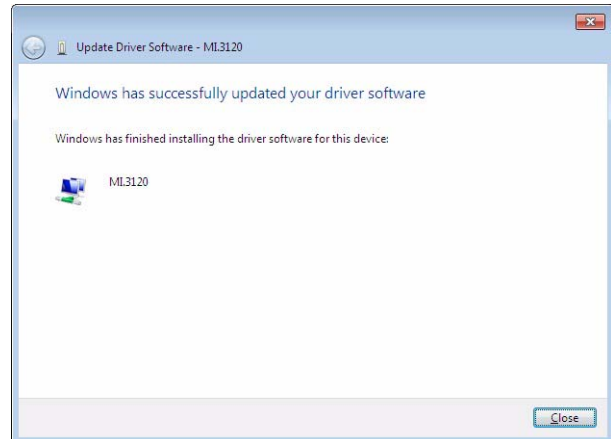
On the upcoming Windows security dialog select install. To prevent Windows Vista/7 to always ask this question for future updates, you can optionally select to always trust software from Spectrum.



The hardware assistant then shows you the exact board type that has been found like the MI.3120 in the example.

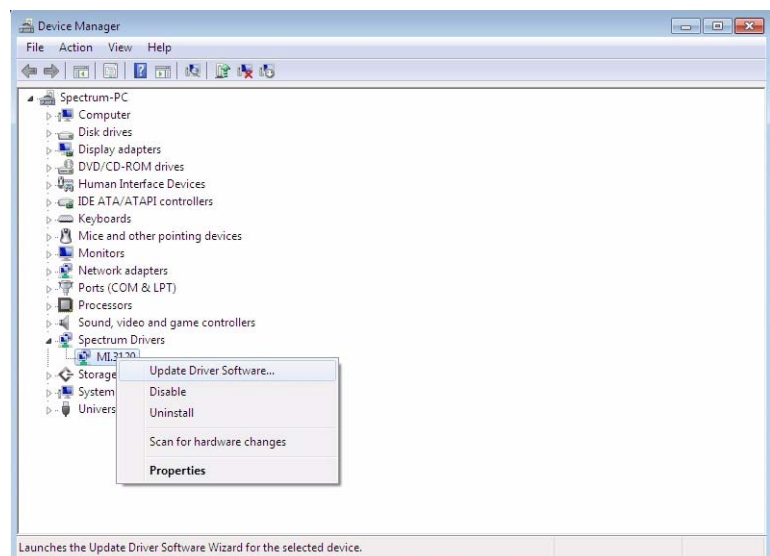
The drivers can be used directly after installation. It is not necessary to restart the system. The installed drivers are linked in the device manager.

Below you'll see how to examine the driver version and how to update the driver with a newer version.



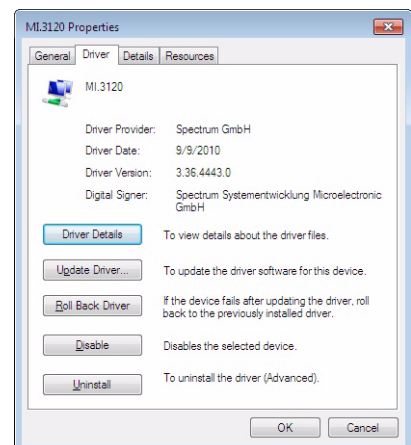
## Version control

If you want to check which driver version is installed in the system this can be easily done in the device manager. Therefore please start the device manager from the control panel and show the properties of the installed driver.



On the property page Windows Vista/7 shows the date and the version of the installed driver.

After clicking the driver details button the detailed version information of the driver is shown. In the case of a support question this information must be presented together with the board's serial number to the support team to help finding a fast solution.



## Driver - Update

The driver update under Windows Vista/7 is exact the same procedure as the initial installation. Please follow the steps above, starting from the device manager, select the Spectrum card to be updated, right-click and select „Update Driver Software...” and follow the steps above.

## Windows NT / Windows 2000 32 Bit

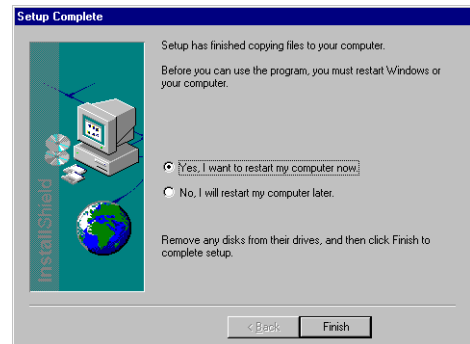
### Installation



Under Windows NT and Windows 2000 the Spectrum driver must be installed manually. The driver is found on CD in the directory \Driver\win32\winnt.

Please start the „winNTDrv\_Install.exe“ program. The installation is performed totally automatically, simply click on the „Next“ button. After installation the system must be rebooted once (see picture on the right

side). The driver is install to support one PCI/PXI or CompactPCI device. If more boards are installed in the system the configuration of the driver has to be changed. Please see the following chapter for this topic.



### Adding boards to the Windows NT / Windows 2000 driver



The Windows NT legacy driver must be configured by the Driver Configuration utility to support more than one board. The Driver Configuration utility is automatically installed with the driver. The Utility can be found in the start menu as „DrvConfig“.



To add a new card please follow these steps:

- Increase the board number on top of the screen by pressing the right button
- Change the board type from „Not Installed“ to „PCI Board“
- Press the „Apply changes“ button
- Press the „OK“ button
- Restart the system

### Driver - Update

If a new driver version should be installed no Spectrum board is allowed to be in use by any software. So please stop and exit all software that could access the boards.

When updating a system please simply execute the setup file of the new driver version. Afterwards the system has to be rebooted. The driver configuration is not changed.

### Important Notes on Driver Version 4.00

With Windows driver version V4.00 and later the support for Windows 64 bit versions was added for MI, MC and MX series cards. This required an internal change such that Windows 98, Windows ME, and Windows 2000 versions are no longer compatible with the WDM driver version.



**Because changing from one driver model (WDM) to another (NT legacy) might result in conflicts please contact Spectrum prior to the update.**

## Linux

### Overview

The Spectrum boards are delivered with drivers for linux. It is necessary to install them manually following the steps explained afterwards. The linux drivers can be found on CD in the directory /Driver/linux. As linux is an open source operating system there are several distributions in use world-wide that are compiled with different kernel settings. As we are not able to install and maintain hundreds of different distributions and versions we had to focus on some common used linux distributions.

However if your distribution does not work with one of these pre-compiled kernel modules or you have a specialized kernel installed (like a SMP kernel) you can get the linux driver sources directly from us. With this sources it's no problem to compile and use the linux driver on your system. Please contact your local distributor to get the sources. The Spectrum linux drivers are compatible with kernel versions 2.4, 2.6, 3.x and 4.x.

On this CD you'll find pre-compiled linux kernel modules for the following versions

Distribution	Kernel Version	Processor	Width	Distribution	Kernel Version	Processor	Width
Suse 9.3	2.6.11	single and smp	32 bit	Fedora Core 3	2.6.9	single and smp	32 bit
Suse 10.0	2.6.13	single only	32 bit and 64 bit	Fedora Core 4	2.6.11	single and smp	32 bit
Suse 10.1	2.6.16	single only	32 bit and 64 bit	Fedora Core 5	2.6.15	single and smp	32 bit and 64 bit
Suse 10.2	2.6.18	single and smp	32 bit and 64 bit	Fedora Core 6	2.6.18	single and smp	32 bit and 64 bit
Suse 10.3	2.6.22	single and smp	32 bit and 64 bit	Fedora Core 7	2.6.21	single and smp	32 bit and 64 bit
Suse 11.0	2.6.25	single and smp	32 bit and 64 bit	Fedora 8	2.6.23	single and smp	32 bit and 64 bit
Suse 11.1	2.6.27	single and smp	32 bit and 64 bit	Fedora 9	2.6.25	single and smp	32 bit and 64 bit
Suse 11.2	2.6.31	single and smp	32 bit and 64 bit	Fedora 10	2.6.27	single and smp	32 bit and 64 bit
Suse 11.3	2.6.34	single and smp	32 bit and 64 bit	Fedora 11	2.6.29	single and smp	32 bit and 64 bit
Suse 11.4	2.6.38	single and smp	32 bit and 64 bit	Fedora 12	2.6.31	single and smp	32 bit and 64 bit
Suse 12.1	3.1	single and smp	32 bit and 64 bit	Fedora 13	2.6.33.3	single and smp	32 bit and 64 bit
Suse 12.2	3.4.6	single and smp	32 bit and 64 bit	Fedora 14	2.6.35.6	single and smp	32 bit and 64 bit
Suse 12.3	3.7.0	single and smp	32 bit and 64 bit	Fedora 15	2.6.38.6	single and smp	32 bit and 64 bit
Suse 13.1	3.11.6	single and smp	32 bit and 64 bit	Fedora 16	3.1	single and smp	32 bit and 64 bit
Suse 13.2	3.16.6	single and smp	32 bit and 64 bit	Fedora 17	3.3.4	single and smp	32 bit and 64 bit
Suse 42.1	4.1.12	single and smp	64 bit	Fedora 18	3.6.10	single and smp	32 bit and 64 bit
Debian Sarge	2.4.27	single	32 bit	Fedora 19	3.9.5	single and smp	32 bit and 64 bit
Debian Sarge	2.6.8	single	32 bit	Fedora 20	3.11.10	single and smp	32 bit and 64 bit
Debian Etch	2.6.18	single and smp	32 bit and 64 bit	Fedora 21	3.17.4	single and smp	32 bit and 64 bit
Debian Lenny	2.6.26	single and smp	32 bit and 64 bit	Fedora 22	4.0.4	single and smp	32 bit and 64 bit
Debian Squeeze	2.6.32	single and smp	32 bit and 64 bit	Fedora 23	4.2.3	single and smp	32 bit and 64 bit
Debian Wheezy	3.2.41	single and smp	32 bit and 64 bit	Fedora 24	4.5.5	single and smp	32 bit and 64 bit
Debian Jessie	3.16.7	single and smp	32 bit and 64 bit	Ubuntu 12.04 LTS	3.2	single and smp	32 bit and 64 bit
				Ubuntu 14.04 LTS	3.15.0	single and smp	32 bit and 64 bit
				Ubuntu 16.04 LTS	4.4.0	single and smp	32 bit and 64 bit

### 64 bit

The Spectrum Linux Drivers also run under 64 bit systems based on the AMD 64 bit architecture (AMD64). The Intel architecture (IA64) is not supported and has not been tested. All drivers, examples and programs need to be recompiled to run under 64 bit Linux. The 64 bit support is available starting with driver version 3.18. Due to the different pointer size two additional functions have been implemented that are described later on. All special functionality concerning 64 bit Linux support is marked with the logo seen on the right.



### Installation with Udev support

Starting with driver version 3.21 build 1548 the driver natively supports udev. Once the driver is loaded it automatically generates the device nodes under /dev. The cards are automatically named to /dev/spc0, /dev/spc1, ... If udev is installed on your system the following two installation steps are not necessary to be made manually. You may use all the standard naming and rules that are available with udev.

#### Login as root.

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

#### Select the right driver from the CD.

Refer to the list shown above. If your distribution is not listed there please select the module that most closely matches your installed kernel version. Copy the driver kernel module spc.o from the CD directory to your hard disk. Be sure to use a hard disk directory that is accessible by all users who should work with the board.

#### First time load of the driver

The linux driver is shipped as the loadable module spc.o. The driver includes all Spectrum PCI, PXI and CompactPCI boards. The boards are recognized automatically after driver loading. Load the driver with the insmod command:

```
linux:~ # insmod spc.o
```



The insmod command may generate a warning that the driver module was compiled for another kernel version. In that case you may try to load the driver module with the force parameter and test the board very carefully.

```
linux:~ # insmod -f spc.o
```

If the kernel module could not be loaded in your linux installation it is necessary to compile the driver directly on your system. Please contactSpectrum to get the needed source files including the compilation description.

Depending on the used linux distribution the insmod command generates a message telling the driver version and the board types and serial numbers that have been found. If your distribution does not show this message it is possible to view them with the dmesg command:

```
linux:~ # dmesg
... some other stuff
spc driver version: 3.07 build 0
sp0: MI.3020 sn 01234
```

In the example we show you the output generated by a MI.3020. All other board types are similar to this output but showing the correct board type.

### **Driver info**

Information about the installed boards could be found in the /proc/spectrum file. All PCI, PXI and CompactPCI boards show the basic information found in the EEPROM there. This is an example output generated by a MI.3020:

```
linux:~ # cat /proc/spectrum

Spectrum driver information
-----
Driver Version: 3.07 build 0

Board#0: MI.3020
  serial number:    01234
  production month: 05/2004
  version:          9.6
  samplerate:       100 MHz
  installed memory: 16 MBytes
```

### **Automatic load of the driver**

It is necessary to load the kernel driver module after each start of the system before using the boards. Therefore you may add the „insmod spc.o“ command in one of the start-up files. Or you may load the kernel driver module manually whenever you need access to the board.

## **Installation without Udev support**

### **Login as root.**

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

### **Select the right driver from the CD.**

Refer to the list shown above. If your distribution is not listed there please select the module that most closely matches your installed kernel version. Copy the driver kernel module spc.o from the CD directory to your hard disk. Be sure to use a hard disk directory that is accessible by all users who should work with the board.

### **First time load of the driver**

The linux driver is shipped as the loadable module spc.o. The driver includes all Spectrum PCI, PXI and CompactPCI boards. The boards are recognized automatically after driver loading. Load the driver with the insmod command:

```
linux:~ # insmod spc.o
```

The insmod command may generate a warning that the driver module was compiled for another kernel version. In that case you may try to load the driver module with the force parameter and test the board very carefully.

```
linux:~ # insmod -f spc.o
```

If the kernel module could not be loaded in your linux installation it is necessary to compile the driver directly on your system. Please contactSpectrum to get the needed source files including the compilation description.



Depending on the used linux distribution the insmod command generates a message telling the driver version and the board types and serial numbers that have been found. If your distribution does not show this message it is possible to view them with the dmesg command:

```
linux:~ # dmesg
... some other stuff
spc driver version: 3.07 build 0
sp0: MI.3020 sn 01234
```

In the example we show you the output generated by a MI.3020. All other board types are similar to this output but showing the correct board type.

### **Examine the major number of the driver**

For accessing the device driver it is necessary to know the major number of the device. This number is listed in the /proc/devices list. The device driver is called "spec" in this list. Normally this number is 254 but this depends on the device drivers that have been installed before.

```
linux:~ # cat /proc/devices
Character devices:
...
171 ieee1394
180 usb
188 ttyUSB
254 spec

Block devices:
 1 ramdisk
 2 fd
...
```

### **Installing the device**

You connect a device to the driver with the mknod command. The major number is the number of the driver as shown in the last step, the minor number is the index of the board starting with 0. This step must only be done once for the system where the boards are installed in. The device will remain in the file structure even if the board is de-installed from the system.

The following command makes a device for the first Spectrum board the driver has found:

```
linux:~ # mknod /dev/spc0 c 254 0
```

Make sure that the users who work with the driver have full rights access for the device. Therefore you should give all persons all rights to the device:

```
linux:~ # chmod a+w /dev/spc0
```

Now it is possible to access the board using this device.

### **Driver info**

Information about the installed boards could be found in the /proc/spectrum file. All PCI, PXI and CompactPCI boards show the basic information found in the EEPROM there. This is an example output generated by a MI.3020:

```
linux:~ # cat /proc/spectrum

Spectrum driver information
-----
Driver Version: 3.07 build 0

Board#0: MI.3020
  serial number:    01234
  production month: 05/2004
  version:         9.6
  samplerate:      100 MHz
  installed memory: 16 MBytes
```

### **Automatic load of the driver**

It is necessary to load the kernel driver module after each start of the system before using the boards. Therefore you may add the „insmod spc.o“ command in one of the start-up files. Or you may load the kernel driver module manually whenever you need access to the board.

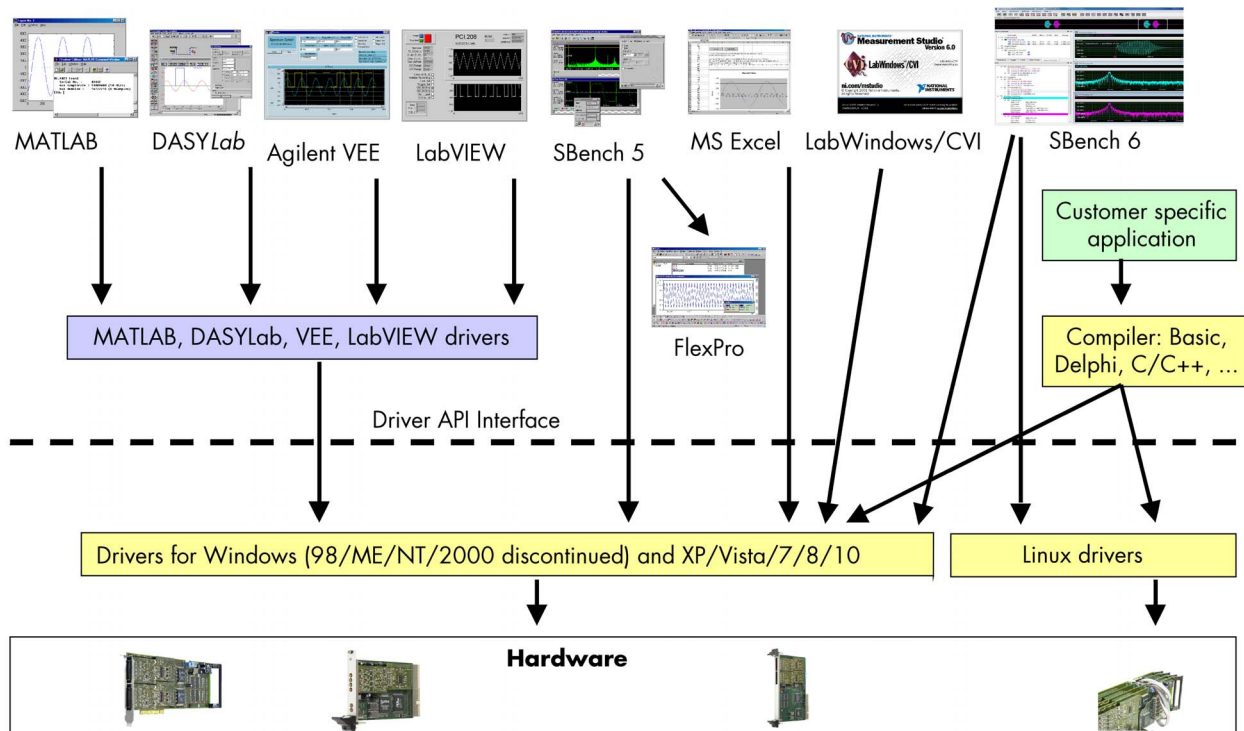
## Software

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 detailed shows how the drivers are included under different programming languages and where the differences are when calling the driver functions from different programming languages.



**This manual only shows the use of the standard driver API. For further information on programming drivers for third-party software like LabVIEW, MATLAB (and on request DASYLab or VEE) an additional manual can be found on the CD delivered with the card.**

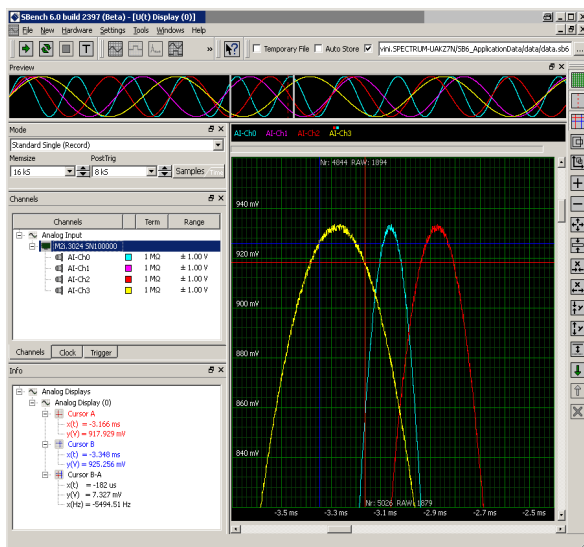
## Software Overview



The Spectrum drivers offer you a common and fast API for using all of the board hardware features. This API is nearly the same on all operating systems. Based on this API one can write your own programs using any programming language that can access the driver API. This manual detailed describes the driver API allowing you to write your own programs.

The optional drivers for third-party products like LabVIEW or DASYLab are also based on this API. The special functionality of these drivers is not subject of this manual and is described on separate manuals delivered with the driver option.

## 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 CD 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.

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 6 as a digital storage oscilloscope, a spectrum analyzer, 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 CD 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](http://www.spectrum-instrumentation.com).

Please go to the download section and get the latest version there. If using

the digitizerNETBOX/generatorNETBOX, a SBench 6 version is also available on the webpages of the digitizerNETBOX/generatorNETBOX.

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 build up. 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.

### Header files

The basic task before using the driver is to include the header files that are delivered on CD together with the board. The header files are found in the directory /Driver/header\_c. 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 this definitions. The use of this typ definition file allows the use of examples and programs on different platforms without changes to the program source.
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.
spectrum.h	Defines the functions of the driver. All definitions are taken from the file dlltyp.h. The functions itself are described below.
spcerr.h	Lists all and describes all error codes that can be given back by any of the driver functions. The error codes and their meaning are described in detail in the appendix of this manual.
errors.h	Only there for backward compatibility with older program versions. Please use spcerr.h instead.

Example for including the header files:

```
// ----- driver includes -----
#include "../c_header/dlltyp.h"
#include "../c_header/spectrum.h"
#include "../c_header/spcerr.h"
#include "../c_header/regs.h"
```

## Microsoft Visual C++

### Include Driver

The driver files can be easily included in Microsoft C++ by simply using the library file that is delivered together with the drivers. The library file can be found on the CD in the path /Examples/vc/c\_header. Please include the library file Spectrum.lib in your Visual C++ project. All functions described below are now available in your program.

### Examples

Examples can be found on CD in the path /Examples/vc. There is one subdirectory for each board family. You'll find board specific examples for that family there. The examples are bus type independent. As a result that means that the MI30xx directory contains examples for the MI.30xx, the MC.30xx and the MX.30xx families. The example directories contain a running project file for Microsoft Visual C++ that can be directly loaded and compiled.

There are also some more board independent examples in the directory Mlxxx. These examples show different aspects of the boards like programming options or synchronization and have to be combined with one of the board specific example.

## **Borland C++ Builder**

### **Include Driver**

The driver files can be easily included in Borland C++ Builder by simply using the library file that is delivered together with the drivers. The library file can be found on the CD in the path /Examples/vc/c\_header. Please include the library file splib\_bcc.lib in your Borland C++ Builder project. All functions described below are now available in your program.

### **Examples**

The Borland 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 Borland C++ Builder.

## **Linux Gnu C**

### **Include Driver**

The interface of the linux drivers is a little bit different from the windows interface. To make the access easier and to have more similar examples we added an include file that re maps the standard driver functions to the linux specific functions. This include file is found in the path /Examples/linux/spciocli.inc. All examples are based on this file.

Example for including Linux driver:

```
// ----- driver includes -----
#include "../c_header/dlltyp.h"
#include "../c_header/regs.h"
#include "../c_header/spcerr.h"

// ----- include the easy ioctl commands from the driver -----
#include "../c_header/spciocli.inc"
```

### **Examples**

Examples can be found on CD in the path /Examples/linux. There is one subdirectory for each board family. You'll find board specific examples for that family there. The examples are bus type independent. As a result that means that the MI30xx directory contains examples for the MI.30xx, the MC.30xx and the MX.30xx families. The examples are simple one file programs and can be compiled using the Gnu C compiler gcc. It's not necessary to use a makefile for them.

## **Other Windows C/C++ compilers**

### **Include Driver**

To access the driver, the driver functions must be loaded from the driver dll. This can be easily done by standard windows functions. There is one example in the directory /Examples/other that shows the process. After loading the functions from the dll one can proceed with the examples that are given for Microsoft Visual C++.

Example of function loading:

```
// definition of external function that has to be loaded from DLL
typedef int16 (SPCINITPCIBOARDS) (int16* pnCount, int16* pnPCIVersion);
typedef int16 (SPCSETPARAM) (int16 nNr, int32 lReg, int32 lValue);
typedef int16 (SPCGETPARAM) (int16 nNr, int32 lReg, int32* plValue);
...
SPCINITPCIBOARDS* pfnSpcInitPCIBoards;
SPCSETPARAM* pfnSpcSetParam;
SPCGETPARAM* pfnSpcGetParam;
...
// ----- Search for dll -----
hDLL = LoadLibrary ("spectrum.dll");

// ----- Load functions from DLL -----
pfnSpcInitPCIBoards = (SPCINITPCIBOARDS*) GetProcAddress (hDLL, "SpcInitPCIBoards");
pfnSpcSetParam = (SPCSETPARAM*) GetProcAddress (hDLL, "SpcSetParam");
pfnSpcGetParam = (SPCGETPARAM*) GetProcAddress (hDLL, "SpcGetParam");
```

## **National Instruments LabWindows/CVI**

### **Include Drivers**

To use the Spectrum driver under LabWindows/CVI it is necessary to first load the functions from the driver dll. This is more or less similar to the above shown process with the only difference that LabWindows/CVI uses it's own library handling functions instead of the windows standard functions.

Example of function loading under LabWindows/CVI:

```
// ----- load the driver entries from the DLL -----
DriverId = LoadExternalModule ("spectrum.lib");

// ----- Load functions from DLL -----
SpcInitPCIBoards = (SPCINITPCIBOARDS*) GetExternalModuleAddr (DriverId, "SpcInitPCIBoards", &Status);
SpcSetParam = (SPCSETPARAM*) GetExternalModuleAddr (DriverId, "SpcSetParam", &Status);
SpcGetParam = (SPCGETPARAM*) GetExternalModuleAddr (DriverId, "SpcGetParam", &Status);
```

### Examples

Examples for LabWindows/CVI can be found on CD in the directory /Examples/cvi. These examples show mainly how to include the driver in a LabWindows/CVI environment and don't use any special functions of the boards. The examples have to be merged with the standard windows examples described under Visual C++.

### Driver functions

The driver contains five functions to access the hardware.

#### Function SpcInitPCIBoard

This function initializes all installed PCI, PXI and CompactPCI boards. The boards are recognized automatically. All installation parameters are read out from the hardware and stored in the driver. The number of PCI boards will be given back in the value Count and the version of the PCI bus itself will be given back in the value PCIVersion.

Function SpcInitPCIBoards:

```
int16 SpcInitPCIBoards (int16* count, int16* PCIVersion);
```

**Under Linux this function is not available. Instead one must open and close the driver with the standard file functions open and close. The functionality behind this function is the same as the SpcInitPCIBoards function.**



Using the Driver under Linux:

```
hDrv = open ("/dev/spc0", O_RDWR);
...
close (hDrv);
```

#### Function SpcSetParam

All hardware settings are based on software registers that can be set by the function SpcSetParam. This function sets a register to a defined value or executes a command. The board must first be initialized. The available software registers for the driver are listed in the board specific part of the documentation below.

The value „nr“ contains the index of the board that you want to access, the value „reg“ is the register that has to be changed and the value „value“ is the new value that should be set to this software register. The function will return an error value in case of malfunction.

Function SpcSetParam

```
int16 SpcSetParam (int16 nr, int32 reg, int32 value);
```

**Under Linux the value „nr“ must contain the handle that was retrieved by the open function for that specific board. The value is then not of the type „int16“ but of the type „handle“.**



#### Function SpcGetParam

The function SpcGetParam reads out software registers or status information. The board must first be initialized. The available software registers for the driver are listed in the board specific part of the documentation below.

The value „nr“ contains the index of the board that you want to access, the value „reg“ is the register that has to be read out and the value „value“ is a pointer to a value that should contain the read parameter after function call. The function will return an error value in case of malfunction.

Function SpcGetParam

```
int16 SpcGetParam (int16 nr, int32 reg, int32* value);
```

**Under Linux the value „nr“ must contain the handle that was given back by the open function of that specific board. The value is then not of the type „int16“ but of the type „handle“.**



**Function SpcSetAdr**

This function is only available under Linux. It is intended to program one of the FIFO buffer addresses to the driver. Depending on the platform (32 bit or 64 bit) the address parameter has a matching pointer size of 32 bit or 64 bit. This function can be used with Linux 32 bit as well as Linux 64 bit installations. The function was implemented with driver version 3.18 and is not available with prior driver versions. Please be sure to use the matching spcioctl.inc file including this function declaration.

Linux  
64 Bit

Function SpcSetAdr

```
int16 SpcSetAdr (drv_handle hDrv, int32 lReg, void* pvAdr);
```

**Function SpcGetAdr**

This function is only available under Linux. It is intended to read out one of the FIFO buffer addresses from the driver. Depending on the platform (32 bit or 64 bit) the address parameter has a matching pointer size of 32 bit or 64 bit. This function can be used with Linux 32 bit as well as Linux 64 bit installations. The function was implemented with driver version 3.18 and is not available with prior driver versions. Please be sure to use the matching spcioctl.inc file including this function declaration.

Linux  
64 Bit

Function SpcGetAdr

```
int16 SpcGetAdr (drv_handle hDrv, int32 lReg, void** ppvAdr);
```

**Function SpcSetData**

Writes data to the board for a specific memory channel. The board must first be initialized. The value „nr“ contains the index of the board that you want to access, the „ch“ parameter contains the memory channel. „start“ and „len“ define the position of data to be written. „data“ is a pointer to the array holding the data. The function will return an error value in case of malfunction.



**This function is only available on generator or I/O boards. The function is not available on acquisition boards.**

Function SpcSetData (Windows)

```
int16 SpcSetData (int16 nr, int16 ch, int32 start, int32 len, dataptr data);
```

Under Linux the additional parameter nBytesPerSample must be used for this function. For all boards with 8 bit resolution the parameter is „1“, for all boards with 12, 14 or 16 bit resolution this parameter has to be „2“. Under Linux the value „hDrv“ must contain the handle that was given back by the open function of that specific board. Under Linux the return value is not an error code but the number of bytes that has been written.

Function SpcSetData (Linux)

```
int32 SpcSetData (int hDrv, int32 lCh, int32 lStart, int32 lLen, int16 nBytesPerSample, dataptr pvData)
```

**Function SpcGetData**

Reads data from the board from a specific memory channel. The board must first be initialized. The value „nr“ contains the index of the board that you want to access, the „ch“ parameter contains the memory channel. „start“ and „len“ define the position of data to be read. „data“ is a pointer to the array that should hold the data. The function will return an error value in case of malfunction.



**This function is only available on acquisition or I/O boards. The function is not available on generator boards.**

Function SpcGetData

```
int16 SpcGetData (int16 nr, int16 ch, int32 start, int32 len, dataptr data);
```

Under Linux the additional parameter nBytesPerSample must be used for this function. For all boards with 8 bit resolution the parameter is „1“, for all boards with 12, 14 or 16 bit resolution this parameter has to be „2“, when reading timestamps this parameter has to be „8“. Under Linux the value „hDrv“ must contain the handle that was given back by the open function of that specific board. Under Linux the return value is not an error code but is the number of bytes that has been read.

Function SpcGetData (Linux)

```
int32 SpcGetData (int hDrv, int32 lCh, int32 lStart, int32 lLen, int16 nBytesPerSample, dataptr pvData)
```

## **Delphi (Pascal) Programming Interface**

### **Type definition**

All Spectrum driver functions are using pre-defined variable types to cover different operating systems and to use the same driver interface for all programming languages. Under Delphi it is necessary to define these types once. This is also shown in the examples delivered on CD.

Delphi type definition:

```
type
  int8   = shortint;
  pint8  = ^shortint;
  int16  = smallint;
  pint16 = ^smallint;
  int32  = longint;
  pint32 = ^longint;
  data   = array[1..MEMSIZE] of smallint;
  dataptr = ^data;
```

**In the example shown above the size of data is defined to „smallint“. This definition is only valid for boards that have a sample resolution of 12, 14 or 16 bit. On 8 bit boards this has to be a „shortint“ type.**



### **Include Driver**

To include the driver functions into delphi it is necessary to first add them to the implementation section of the program file. There the name of the function and the location in the dll is defined:

Driver implementation:

```
function SpcSetData (nr,ch:int16; start,len:int32; data:dataptr): int16; cdecl; external 'SPECTRUM.DLL';
function SpcGetData (nr,ch:int16; start,len:int32; data:dataptr): int16; cdecl; external 'SPECTRUM.DLL';
function SpcSetParam (nr:int16; reg,value: int32): int16; cdecl; external 'SPECTRUM.DLL';
function SpcGetParam (nr:int16; reg:int32; value:pint32): int16; cdecl; external 'SPECTRUM.DLL';
function SpcInitPCIBoards (count,PCIVersion: pint16): int16; cdecl; external 'SPECTRUM.DLL';
```

### **Examples**

Examples for Delphi can be found on CD in the directory /Examples/delphi. There is one subdirectory for each board family. You'll find board specific examples for that family there. The examples are bus type independent. As a result that means that the MI30xx directory contains examples for the MI.30xx, the MC.30xx and the MX.30xx families. The example directories contain a running project file for Borland Delphi that can be directly loaded and compiled.

### **Driver functions**

The driver contains five functions to access the hardware.

#### **Function SpcInitPCIBoard**

This function initializes all installed PCI, PXI and CompactPCI boards. The boards are recognized automatically. All installation parameters are read out from the hardware and stored in the driver. The number of PCI boards will be given back in the value Count and the version of the PCI bus itself will be given back in the value PCIVersion.

#### **Function SpcSetParam**

All hardware settings are based on software registers that can be set by the function SpcSetParam. This function sets a register to a defined value or executes a command. The board must first be initialized. The available software registers for the driver are listed in the board specific part of the documentation below.

The value „nr“ contains the index of the board that you want to access, the value „reg“ is the register that has to be changed and the value „value“ is the new value that should be set to this software register. The function will return an error value in case of malfunction.

#### **Function SpcGetParam**

The function SpcGetParam reads out software registers or status information. The board must first be initialized. The available software registers for the driver are listed in the board specific part of the documentation below.

The value „nr“ contains the index of the board that you want to access, the value „reg“ is the register that has to be read out and the value „value“ is a pointer to a value that should contain the read parameter after function call. The function will return an error value in case of malfunction.

#### **Function SpcSetData**

Writes data to the board for a specific memory channel. The board must first be initialized. The value „nr“ contains the index of the board that you want to access, the „ch“ parameter contains the memory channel. „start“ and „len“ define the position of data to be written. „data“ is a pointer to the array holding the data. The function will return an error value in case of malfunction.



**This function is only available on generator or i/o boards. The function is not available on acquisition boards.**

### **Function SpcGetData**

Reads data from the board from a specific memory channel. The board must first be initialized. The value „nr“ contains the index of the board that you want to access, the „ch“ parameter contains the memory channel. „start“ and „len“ define the position of data to be read. „data“ is a pointer to the array that should hold the data. The function will return an error value in case of malfunction.



**This function is only available on acquisition or i/o boards. The function is not available on generator boards.**

## **Visual Basic Programming Interface**

The Spectrum boards can be used together with Microsoft Visual Basic as well as with Microsoft Visual Basic for Applications. This allows per example the direct access of the hardware from within Microsoft Excel. The interface between the programming language and the driver is the same for both.

### **Include Driver**

To include the driver functions into Basic it is necessary to first add them to the module definition section of the program file. There the name of the function and the location in the dll is defined:

Module definition:

```
Public Declare Function SpcInitPCIBoards Lib "SpcStdNT.dll" Alias "_SpcInitPCIBoards@8" (ByRef Count As Integer,
ByRef PCIVersion As Integer) As Integer
Public Declare Function SpcInitBoard Lib "SpcStdNT.dll" Alias "_SpcInitBoard@8" (ByVal Nr As Integer, ByVal Typ
As Integer) As Integer
Public Declare Function SpcGetParam Lib "SpcStdNT.dll" Alias "_SpcGetParam@12" (ByVal BrdNr As Integer, ByVal
RegNr As Long, ByRef Value As Long) As Integer
Public Declare Function SpcSetParam Lib "SpcStdNT.dll" Alias "_SpcSetParam@12" (ByVal BrdNr As Integer, ByVal
RegNr As Long, ByVal Value As Long) As Integer
Public Declare Function SpcGetData8 Lib "SpcStdNT.dll" Alias "_SpcGetData@20" (ByVal BrdNr As Integer, ByVal
Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Byte) As Integer
Public Declare Function SpcSetData8 Lib "SpcStdNT.dll" Alias "_SpcSetData@20" (ByVal BrdNr As Integer, ByVal
Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Byte) As Integer
Public Declare Function SpcGetData16 Lib "SpcStdNT.dll" Alias "_SpcGetData@20" (ByVal BrdNr As Integer, ByVal
Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Integer) As Integer
Public Declare Function SpcSetData16 Lib "SpcStdNT.dll" Alias "_SpcSetData@20" (ByVal BrdNr As Integer, ByVal
Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Integer) As Integer
```

The module definition is already done for the examples and can be found in the Visual Basic examples directory. Please simply use the file declnt.bas.

### **Visual Basic Examples**

Examples for Visual Basic can be found on CD in the directory /Examples/vb. There is one subdirectory for each board family. You'll find board specific examples for that family there. The examples are bus type independent. As a result that means that the MI30xx directory contains examples for the MI.30xx, the MC.30xx and the MX.30xx families. The example directories contain a running project file for Visual Basic that can be directly loaded.

### **VBA for Excel Examples**

Examples for VBA for Excel can be found on CD in the directory /Examples/excel. The example here simply show the access of the driver and make a very small demo acquisition. It is necessary to combine these examples with the Visual Basic examples to have full board functionality.

### **Driver functions**

The driver contains five functions to access the hardware.

#### **Function SpcInitPCIBoard**

This function initializes all installed PCI, PXI and CompactPCI boards. The boards are recognized automatically. All installation parameters are read out from the hardware and stored in the driver. The number of PCI boards will be given back in the value Count and the version of the PCI bus itself will be given back in the value PCIVersion.

Function SpcInitPCIBoard:

```
Function SpcInitPCIBoards (ByRef Count As Integer, ByRef PCIVersion As Integer) As Integer
```



**Function SpcSetParam**

All hardware settings are based on software registers that can be set by the function SpcSetParam. This function sets a register to a defined value or executes a command. The board must first be initialized. The available software registers for the driver are listed in the board specific part of the documentation below.

The value „nr“ contains the index of the board that you want to access, the value „reg“ is the register that has to be changed and the value „value“ is the new value that should be set to this software register. The function will return an error value in case of malfunction.

Function SpcSetParam:

```
Function SpcSetParam (ByVal BrdNr As Integer, ByVal RegNr As Long, ByVal Value As Long) As Integer
```

**Function SpcGetParam**

The function SpcGetParam reads out software registers or status information. The board must first be initialized. The available software registers for the driver are listed in the board specific part of the documentation below.

The value „nr“ contains the index of the board that you want to access, the value „reg“ is the register that has to be read out and the value „value“ is a pointer to a value that should contain the read parameter after function call. The function will return an error value in case of malfunction.

Function SpcGetParam:

```
Function SpcGetParam (ByVal BrdNr As Integer, ByVal RegNr As Long, ByRef Value As Long) As Integer
```

**Function SpcSetData**

Writes data to the board for a specific memory channel. The board must first be initialized. The value „nr“ contains the index of the board that you want to access, the „ch“ parameter contains the memory channel. „start“ and „len“ define the position of data to be written. „data“ is a pointer to the array holding the data. The function will return an error value in case of malfunction.

Function SpcSetData:

```
Function SpcSetData8 (ByVal BrdNr As Integer, ByVal Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Byte) As Integer

Function SpcSetData16 (ByVal BrdNr As Integer, ByVal Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Integer) As Integer
```

**It is necessary to select the function with the matching data width from the above mentioned data write functions. Use the SpcSetData8 function for boards with 8 bit resolution and use the SpcSetData16 function for boards with 12, 14 and 16 bit resolution.**



**This function is only available on generator or i/o boards. The function is not available on acquisition boards.**

**Function SpcGetData**

Reads data from the board from a specific memory channel. The board must first be initialized. The value „nr“ contains the index of the board that you want to access, the „ch“ parameter contains the memory channel. „start“ and „len“ define the position of data to be read. „data“ is a pointer to the array that should hold the data. The function will return an error value in case of malfunction.

Function SpcGetData:

```
Function SpcGetData8 (ByVal BrdNr As Integer, ByVal Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Byte) As Integer

Function SpcGetData16 (ByVal BrdNr As Integer, ByVal Channel As Integer, ByVal Start As Long, ByVal Length As Long, ByRef data As Integer) As Integer
```

**It is necessary to select the function with the matching data width from the above mentioned data read functions. Use the SpcGetData8 function for boards with 8 bit resolution and use the SpcGetData16 function for boards with 12, 14 and 16 bit resolution.**



**This function is only available on acquisition or i/o boards. The function is not available on generator boards.**



## 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 pypscm you need either python 2 (2.4, 2.6 or 2.7) or python 3 (3.x) and ctypes, which is included in python 2.6 and newer and needs to be installed separately for Python 2.4.

#### file pymicx.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 -----
micxDll = windll.LoadLibrary ("c:\\windows\\system32\\spectrum.dll")

# load SpcInitPCIBoards
SpcInitPCIBoards = getattr (micxDll, "SpcInitPCIBoards")
SpcInitPCIBoards.argtype = [ptr16, ptr16]
SpcInitPCIBoards.restype = int16

# load SpcInitBoard
SpcInitBoard = getattr (micxDll, "SpcInitBoard")
SpcInitBoard.argtype = [int16, int16]
SpcInitBoard.restype = int16

# load SpcGetParam
SpcGetParam = getattr (micxDll, "SpcGetParam")
SpcGetParam.argtype = [int16, int32, ptr32]
SpcGetParam.restype = int16

# load SpcSetParam
SpcSetParam = getattr (micxDll, "SpcSetParam")
SpcSetParam.argtype = [int16, int32, int32]
SpcSetParam.restype = int16

# load SpcGetData
SpcGetData = getattr (micxDll, "SpcGetData")
SpcGetData.argtype = [int16, int16, int32, int32, dataptr]
SpcGetData.restype = int16

# load SpcSetData
SpcSetData = getattr (micxDll, "SpcSetData")
SpcSetData.argtype = [int16, int16, int32, int32, dataptr]
SpcSetData.restype = int16

# ----- Linux -----
# Python for Linux for MI/MC/MX cards is not yet implemented (as of August 2014)
# ... please contact Spectrum
```

#### 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_COMMAND = 01          # write a command
SPC_RESET = 01            # hardware reset
SPC_START = 101           # start of card (standard mode)
SPC_FIFOSTART = 121       # start of card (FIFO mode)
...
```

#### 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 CD in the directory /examples/python. The directory contains the above mentioned header files and a 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])**

# 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 focussed 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:

The name of the software register as found in the regs.h file. Could directly be used by C and C++ compiler	The decimal value of the software register. Also found in the regs.h file. This value must be used with all programs or compilers that cannot use the header file directly.	Describes whether the register can be read (r) and/or written (w).	Short description of the functionality of the register. A more detailed description is found above or below this register.
Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board.
SPC_START	10		Starts the board with the current register settings.
SPC_STOP	20		Stops the board manually.
Any constants that can be used to program the register directly are shown inserted beneath the register table.	The decimal value of the constant. Also found in the regs.h file. This value must be used with all programs or compilers that cannot use the header file directly.		Short description of the use of this constant.

**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 basically written using the Visual C++ compiler for Windows. If you use Linux there are some changes in the function's parameter lists as mentioned in the relating software chapter.

To keep the examples as compatible as possible for users of both operational systems (Windows and Linux) all the functions that contain either a board number (Windows) or a handle (Linux) use the common parameter name 'hDrv'. Windows users simply have to set the parameter to the according board number (as the example below is showing), while Linux users can easily use the handle that is given back for the according board by the initialization function.



```
// Windows users must set hDrv to the according board number before.
// Assuming that there is only one Spectrum board installed you'll
// have to set hDrv like this:

hDrv = 0;

SpcGetParam (hDrv, SPC_LASTERRORCODE, &ErrorCode); // Any command just to show the hDrv usage
```

## 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 SPC\_LASTERRORCODE function. All other functions will lead to the same errorcode unless the error is cleared by reading SPC\_LASTERRORCODE.**



This means as a result that it is not necessary to check each driver call for an error but to check for an error before the board is started to see whether all settings have been valid.

By reading all the error information one can easily examine where the error occurred. The following table shows all the error related registers that can be read out.

Register	Value	Direction	Description
SPC_LASTERRORCODE	999999	r	Error code of the last error that occurred. The errorcodes are found in spcerr.h. If this register is read, the driver will be unlocked.
SPC_LASTERRORREG	999998	r	Software register that causes the error.
SPC_LASTERRORVALUE	999997	r	The value that has been written to the faulty software register.



**The 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.**

Example for error checking:

```

SpcSetParam (hDrv, SPC_MEMSIZE, -345);                // faulty command
if (SpcSetParam (hDrv, SPC_COMMAND, SPC_START) != ERR_OK) // try to start and check for an error
{
    SpcGetParam (hDrv, SPC_LASTERRORCODE, &lErrorCode);    // read out the error information
    SpcGetParam (hDrv, SPC_LASTERRORREG, &lErrorReg);
    SpcGetParam (hDrv, SPC_LASTERRORVALUE, &lErrorValue);
    printf („Error %d when writing Register %d with Value %d !\n“, lErrorCode, lErrorReg, &lErrorValue);
}

```

This short program then would generate a printout as:

```
Error 101 when writing Register 10000 with Value -345 !
```

## Initialization

### Starting the automatic initialization routine

Before you can access the boards in your program, you have to initialize them first. Therefore the Spectrum function SpcInitPCIBoards is used. If it is called, all Spectrum boards in the host system are initialized automatically. If no errors occurred during the initialization, the returned value is 0 (ERR\_OK). In any other cases something has gone wrong. Please see appendix for explanations of the different error codes.

If the process of initializing the boards was successful, the function returns the total number of Spectrum boards that have been found in your system. The third return value is the revision of the PCI Bus, the Spectrum boards are installed in.

The following example shows how to start the initialization of the board and check for errors.

```

// ----- Initialization of PCI Bus Boards-----
if (SpcInitPCIBoards (&nCount, &nPCIBusVersion) != ERR_OK)
    return;
if (nCount == 0)
{
    printf ("No Spectrum board found\n");
    return;
}

```

## PCI Register

These registers are set by the driver after the PCI initialization. The information is found in the on-board EEPROM, and can easily be read out by your own application software. All of the following PCI registers are read only. You get access to all registers by using the Spectrum function SpcGetParam with one of the following registers.

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

One of the following values is returned, when reading this register.

Boardtype	Value hexadecimal	Value decimal	Boardtype	Value hexadecimal	Value decimal
TYP_MC4710	14710h	83728	TYP_MC4711	14711h	83729
TYP_MC4720	14720h	83744	TYP_MC4721	14721h	83745
TYP_MC4730	14730h	83760	TYP_MC4731	14731h	83761

## Hardware version

Since all of the MI, MC and MX boards from Spectrum are modular boards, they consist of one base board and one or two (only PCI and CompactPCI) piggy-back modules. This register SPC\_PCIVERSION gives information about the revision of either the base board and the modules. 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	r	Board revision: bit 15..8 show revision of the base card, bit 7..0 the revision of the modules

If your board has a piggy-back expansion module mounted (MC und MI series boards only) you can get the hardwareversion with the following register.

Register	Value	Direction	Description
SPC_PCIEXTVERSION	2011	r	Board's expansion module hardware revision as integer value.

## Date of production

This register informs you about the production date, which is returned as one 32 bit longword. The upper word is holding the information about the year, while the lower byte informs about the month. The second byte (counting from below) is not used. If you only need to know the production year of your board you have to mask the value accordingly. Normally you do not need this information, but if you have a support question, please provide the revision within.

Register	Value	Direction	Description
SPC_PCIDATE	2020	r	Production date: year in bit 31..16, month in bit 7..0, bit 15..8 are not used

## 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	r	Serial number of the board

## Maximum possible sample rate

This register gives you the maximum possible samplerate the board can run however. 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 samplerate and the number of activated channels please refer to the according chapter.

Register	Value	Direction	Description
SPC_PCISAMPLERATE	2100	r	Maximum samplerate in Hz as a 32 bit integer value

## Installed memory

This register returns the size of the installed on-board memory in bytes as a 32 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 Spectrum board. All 8 bit boards can store only sample per byte, while all other boards with 12, 14 and 16 bit use two bytes to store one sample.

Register	Value	Direction	Description
SPC_PCIMEMSIZE	2110	r	Installed memory in bytes as a 32 bit integer value

The following example is written for a „two bytes“ per sample board (12, 14 or 16 bit board).

```
SpcGetParam (hDrv, SPC_PCIMEMSIZE, &lInstMemsize);
printf ("Memory on board: %ld MBytes (%ld MSamples)\n", lInstMemsize / 1024 / 1024, lInstMemsize / 1024 / 1024 / 2);
```

## Installed features and options

The SPC\_PCIFEATURES register informs you about the options, 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.

Register	Value	Direction	Description
SPC_PCIFEATURES	2120	r	PCI feature register. Holds the installed features and options as a bitfield, so the return value must be masked with one of the masks below to get information about one certain feature.
PCIBIT_MULTI	1		Is set if the Option Multiple Recording / Multiple Replay is installed.
PCIBIT_DIGITAL	2		Is set if the Option Digital Inputs / Digital Outputs is installed.
PCIBIT_GATE	32		Is set if the Option Gated Sampling / Gated Replay is installed.
PCIBIT_SYNC	512		Is set if the Option Synchronization is installed for that certain board, regardless what kind of synchronization you use. Boards without this option cannot be synchronized with other boards.
PCIBIT_TIMESTAMP	1024		Is set if the Option Timestamp is installed.

PCIBIT_STARHUB	2048	Is set on the board, that carries the starhub piggy-back module. This flag is set in addition to the PCIBIT_SYNC flag mentioned above. If on no synchronized board the starhub option is installed, the boards are synchronized with the cascading option.
PCIBIT_XIO	8192	Is set if the Option Extra I/O is installed.
PCIBIT_AMPLIFIER	16384	Arbitrary Waveform Generators only: card has additional set of calibration values for amplifier card

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

```
SpcGetParam (hDrv, SPC_PCIFEATURES, &lFeatures);

if (lFeatures & PCIBIT_DIGITAL)
    printf("Option digital inputs is installed on your board");
```

## Used interrupt line

This register holds the information of the actual used interrupt line for the board. This information is sometimes more easy in getting the interrupt line of one specific board then using the hardware setups of your operating system.

Register	Value	Direction	Description
SPC_PCII_INTERRUPT	2300	r	The used interrupt line of the board.

## Used type of driver

This register holds the information about the driver that is actually used to access the board. Although most users will use the boards within a Windows system and most Windows users will use the WDM driver, it can be sometimes necessary of knowing the type of driver.

Register	Value	Direction	Description
SPC_GETDRVTYPE	1220	r	Gives information about what type of driver is actually used
DRVTYP_DOS	0		DOS driver is used (discontinued)
DRVTYP_LINUX32	1		Linux 32bit driver is used
DRVTYP_VXD	2		Windows VXD driver is used (only Windows 95) (discontinued)
DRVTYP_NTLEGACY	3		Windows NT Legacy driver is used (only Windows NT) (discontinued)
DRVTYP_WDM32	4		Windows WDM 32bit driver is used (Windows 98, Windows 2000). (discontinued)
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 informs Windows users about the actual used driver DLL. This information can also be obtained from the device manager. Please refer to the „Driver Installation“ chapter. Linux users will get the revision of their kernel driver instead, because linux does not use any DLL.

Register	Value	Direction	Description
SPC_GETDRVVERSION	1200	r	Gives information about the driver DLL version

## Kernel Driver version

This register informs OS independent about the actual used kernel driver. Windows users can also get this information from the device manager. Please refer to the „Driver Installation“ chapter. Linux users can get the driver version by simply accessing the following register for the kernel driver.

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

### Example program for the board initialization

The following example is only an excerpt to give you an idea on how easy it is to initialize a Spectrum board.

```
// ----- Initialization of PCI Bus Boards -----
if (SpcInitPCIBoards (&nCount, &nPCIBusVersion) != ERR_OK)
    return;

if (nCount == 0)
{
    printf ("No Spectrum board found\n");
    return;
}

// ----- request and print Board type and some information -----
SpcGetParam (hDrv, SPC_PCITYP,          &lBrdType);
SpcGetParam (hDrv, SPC_PCIMEMSIZE,      &lInstMemsize);
SpcGetParam (hDrv, SPC_PCISERIALNO,     &lSerialNumber);

// ----- print the board type depending on bus. Board number is always the lower 16 bit of type -----
switch (lBrdType & TYP_SERIESMASK)
{
    case TYP_MISERIES:
        printf ("Board found:      MI.%x sn: %05d\n", lBrdType & 0xffff, lSerialNumber);
        break;

    case TYP_MC SERIES:
        printf ("Board found:      MC.%x sn: %05d\n", lBrdType & 0xffff, lSerialNumber);
        break;

    case TYP_MX SERIES:
        printf ("Board found:      MX.%x sn: %05d\n", lBrdType & 0xffff, lSerialNumber);
        break;
}

printf ("Memory on board: %ld MBytes (%ld MSamples)\n", lInstMemsize /1024/1024, lInstMemsize /1024/1024 /2);
printf ("Serial Number:  %05ld\n", lSerialNumber);
```

## Powerdown and reset

Every Spectrum board can be set to powerdown mode by software. In this mode the board is therefore consuming less power than in normal operation mode. The amount of saved power is board dependant. Please refer to the technical data section for details. The board can be set to normal mode again either by performing a reset as mentioned below or by starting the board as described in the according chapters later in this manual.

**If the board is set to powerdown mode or a reset is performed the data in the on-board memory will be no longer valid and therefore cannot be read out or replayed again.**



Performing a board reset or powering down the board can be easily done by the related board commands mentioned in the following table.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board.
SPC_POWERDOWN	30		Sets the board to powerdown mode. The data in the on-board memory is no longer valid and cannot be read out or replayed again. The board can be set to normal mode again by the reset command or by starting the boards.
SPC_RESET	0		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.

## Analog Inputs

### Channel Selection

One key setting that influences all other possible settings is the channel enable register. A unique feature of the Spectrum boards 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 board family. However your specific board may have less channels depending on the board type you purchased and did 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	
CHANNEL2	4	Activates channel 2	
CHANNEL3	8	Activates channel 3	
CHANNEL4	16	Activates channel 4	
CHANNEL5	32	Activates channel 5	
CHANNEL6	64	Activates channel 6	
CHANNEL7	128	Activates channel 7	
CHANNEL8	256	Activates channel 8	
CHANNEL9	512	Activates channel 9	
CHANNEL10	1024	Activates channel 10	
CHANNEL11	2048	Activates channel 11	
CHANNEL12	4096	Activates channel 12	
CHANNEL13	8192	Activates channel 13	
CHANNEL14	16384	Activates channel 14	
CHANNEL15	32768	Activates channel 15	

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 4 channels:

```
SpcSetParam (hDrv, SPC_CHENABLE, CHANNEL0 | CHANNEL1 | CHANNEL2 | CHANNEL3);
```

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

Channels to activate																Values to program	Value as hex	Value as decimal
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	CHANNEL0	0001h	1
X	X															CHANNEL0   CHANNEL1	0003h	3
X	X	X	X													CHANNEL0   CHANNEL1   CHANNEL2   CHANNEL3	000Fh	15
X	X	X	X	X	X	X	X									CHANNEL0   CHANNEL1   ...   CHANNEL6   CHANNEL7	00FFh	255
X								X								CHANNEL0   CHANNEL8	0101h	257
X	X							X	X							CHANNEL0   CHANNEL1   CHANNEL8   CHANNEL9	0303h	771
X	X	X	X					X	X	X	X					CHANNEL0   ...   CHANNEL3   CHANNEL8   ...   CHANNEL11	0F0Fh	3855
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	CHANNEL0   CHANNEL1   ...   CHANNEL14   CHANNEL15	FFFFh	65535



**Any channel activation mask that is not shown here is not valid. If programming another channel activation the driver automatically remaps this to the best matching activation mask. You can read out the channel enable register to see what channel activation mask the driver has set.**

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

```
SpcGetParam (hDrv, SPC_CHENABLE, &lActivatedChannels);

printf ("Activated channels bitmask is: %x\n", lActivatedChannels);
```

### Important note on channels 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 table(s) above to get the actual number of available channels.**



## Setting up the inputs

### Input ranges

This analog acquisition board uses separate input amplifiers 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. 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 version available with different input ranges it is recommended to read out the currently available input ranges as shown later in this chapter.

Register	Value	Direction	Description
SPC_AMP0	30010	read/write	Defines the input range of channel 0.
SPC_AMP1	30110	read/write	Defines the input range of channel 1.
SPC_AMP2	30210	read/write	Defines the input range of channel 2.
SPC_AMP3	30310	read/write	Defines the input range of channel 3.
SPC_AMP4	30410	read/write	Defines the input range of channel 4.
SPC_AMP5	30510	read/write	Defines the input range of channel 5.
SPC_AMP6	30610	read/write	Defines the input range of channel 6.
SPC_AMP7	30710	read/write	Defines the input range of channel 7.
SPC_AMP8	30810	read/write	Defines the input range of channel 8.
SPC_AMP9	30910	read/write	Defines the input range of channel 9.
SPC_AMP10	31010	read/write	Defines the input range of channel 10.
SPC_AMP11	31110	read/write	Defines the input range of channel 11.
SPC_AMP12	31210	read/write	Defines the input range of channel 12.
SPC_AMP13	31310	read/write	Defines the input range of channel 13.
SPC_AMP14	31410	read/write	Defines the input range of channel 14.
SPC_AMP15	31510	read/write	Defines the input range of channel 15.
	50		± 50 mV calibrated input range for the appropriate channel.
	100		± 100 mV calibrated input range for the appropriate channel.
	250		± 250 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.

The different input ranges are set with the help of relais. These relais need a settling time if they are changed, so that the relais are fully set and didn't influence the signal when the board is started. The following table shows the related register to adjust the wait time. Any changes of the wait time below the default value should only be done after detailed tests of the boards behaviour. Setting lower values may be possible or may not be possible depending on the application that is done.

Register	Value	Direction	Description
SPC_RELAIWAITTIME	200700	read/write	Wait time in ms for relais settling before the start of the board.

If you want to know, how many different input ranges are available on the actual board per channel, you can easily read that information by using the read-only register shown in the table below.

Register	Value	Direction	Description
SPC_READIRCOUNT	3000	read	Inform about the number of the board's calibrated input ranges.

Additionally you can read out the minimum and the maximum value of each input range as shown in the table below. The number of input ranges is read out with the above shown register.

Register	Value	Direction	Description
SPC_READRANGEMIN0	4000	read	Gives back the minimum value of input range 0 in mV.
SPC_READRANGEMIN1	4001	read	Gives back the minimum value of input range 1 in mV.
SPC_READRANGEMIN2	4002	read	Gives back the minimum value of input range 2 in mV.
...	...	read	...
SPC_READRANGEMAX0	4100	read	Gives back the maximum value of input range 0 in mV.
SPC_READRANGEMAX1	4101	read	Gives back the maximum value of input range 1 in mV.
SPC_READRANGEMAX2	4102	read	Gives back the maximum value of input range 2 in mV.
...	...	read	...

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

```
SpcGetParam (hDrv, READIRCOUNT,      &lNumberOfRanges);
for (i = 0; i < lNumberOfRanges; i++)
{
    SpcGetParam (hDrv, SPC_READRANGEMIN0 + i, &lMinimumInputRange);
    SpcGetParam (hDrv, SPC_READRANGEMAX0 + i, &lMaximumInputRange);
    printf („Range %d: %d mV to %d mV\n“, i, lMinimumInputRange, lMaximumInputRange);
}
```

## 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.

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

When working with a modified calibration setting instead of the default ones, you may need to restore your calibration settings with the help of the SPC\_ADJ\_LOAD register as the following example shows.

```
SpcSetParam (hDrv, SPC_ADJ_LOAD,      ADJ_DEFAULT); // and load values to DEFAULT set in the EEPROM
```

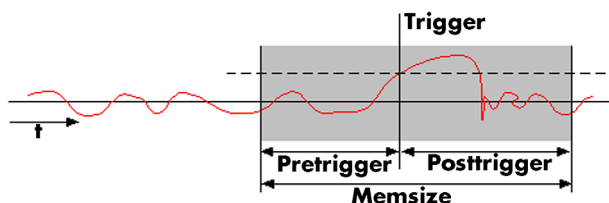
## Standard acquisition modes

### General Information

The standard mode is the easiest and mostly used mode to acquire analog data with a Spectrum A/D board. In standard recording mode the board is working totally independent from the host system (in most cases a standard PC), after the board setup is done. The advantage of the Spectrum boards is that regardless to the system usage the board will sample with equidistant time intervals.

The sampled and converted data is stored in the onboard 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 PCI bus into PC memory.

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



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



### Programming

#### 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 after the trigger event has been detected.

You can access these settings by the registers 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:

$$\text{pretrigger} = \text{memsize} - \text{posttrigger}$$

The maximum memsize that can be used for recording is of course limited by the installed amount of memory and by the number of channels to be recorded. The following table gives you an overview on the maximum memsize in relation to the installed memory.

#### Maximum memsize

ch0	ch1	ch2	ch3	ch4	ch5	ch6	ch7	ch8	ch9	ch10	ch11	ch12	ch13	ch14	ch15	4710 4720 4730	4711 4721 4731
x																1/1	1/1
x	x															1/2	1/2
x	x	x	x													1/4	1/4
x	x	x	x	x	x	x	x									1/8	1/8
x	x							x								n.a.	1/2
x	x							x	x							n.a.	1/4
x	x	x	x					x	x	x	x					n.a.	1/8
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	n.a.	1/16

How to read this table: If you have installed the standard amount of 32 MSample on your 4721 board and you want to record all four channels, you have a total maximum memory of 32 MSample \* 1/4 = 8 MSample per channel for your data.

The maximum settings for the post counter are limited by the hardware, because the post counter has a limited range for counting. The settings depend on the number of activated channels, as the table below is showing.

**Maximum posttrigger in MSamples**

ch0	ch1	ch2	ch3	ch4	ch5	ch6	ch7	ch8	ch9	ch10	ch11	ch12	ch13	ch14	ch15	4710 4720 4730	4711 4721 4731
x																128 MS	128 MS
x	x															64 MS	64 MS
x	x	x	x													32 MS	32 MS
x	x	x	x	x	x	x	x									16 MS	16 MS
x								x								n.a.	128 MS
x	x							x	x							n.a.	64 MS
x	x	x	x					x	x	x	x					n.a.	32 MS
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	n.a.	16 MS

The amount of memory that can be used either for the memsize and the postcounter values can only be set by certain steps. These steps are results of the internal memory organization. For this reason these steps also define the minimum size for the data memory and the postcounter. The values depend on the number of activated channels and on the type of board being used. The minimum stepsizes for setting up the memsize and the postcounter are shown in the table below.

**Minimum and stepsize of memsize and posttrigger in samples**

ch0	ch1	ch2	ch3	ch4	ch5	ch6	ch7	ch8	ch9	ch10	ch11	ch12	ch13	ch14	ch15	4710 4720 4730	4711 4721 4731
x																32	32
x	x															16	16
x	x	x	x													8	8
x	x	x	x	x	x	x	x									4	4
x								x								n.a.	32
x	x							x	x							n.a.	16
x	x	x	x					x	x	x	x					n.a.	8
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	n.a.	4

**Starting without interrupt (classic mode)****Command register**

Register	Value	Direction	Description
SPC_COMMAND	0	read/write	Command register of the board.
SPC_START	10		Starts the board with the current register settings.
SPC_STOP	20		Stops the board manually.

In this mode the board is started by writing the SPC\_START value to the command register. All settings like for example the size of memory and postcounter, the number of activated channels and the trigger settings must have been programmed before. If the start command has been given, the setup data is transferred to the board and the board will start.

If your board has relays to switch between different settings a programmed time will be waited to prevent having the influences of the relays settling time in the signal. For additional information please first see the chapter about the relay settling time. You can stop the board at any time with the command SPC\_STOP. This command will stop immediately.

Once the board has been started, it is running totally independent from the host system. Your program has full CPU time to do any calculations or display. The status register shown in the table below shows the current status of the board. The most simple programming loop is simply waiting for the status SPC\_READY. This status shows that the board has stopped automatically.

The read only status register can be read out at any time, but it is mostly used for polling on the board's status after the board has been started. However polling the status will need CPU time.

### Status register

Register	Value	Direction	Description
SPC_STATUS	10	read	Status register, of the board.
SPC_RUN	0		Indicates that the board has been started and is waiting for a trigger event.
SPC_TRIGGER	10		Indicates that the board is running and a trigger event has been detected.
SPC_READY	20		Indicates that the board has stopped.

The following shortened excerpt of a sample program gives you an example of how to start the board in classic mode and how to poll for the SPC\_READY flag. It is assumed that all board setup has been done before.

```
// ----- start the board -----
nErr = SpcSetParam (hDrv, SPC_COMMAND, SPC_START);

// Here you can check for driver errors as mentioned in the relating chapter

// ----- Wait for Status Ready (polling for SPC_READY in a loop) -----
do
{
    SpcGetParam (hDrv, SPC_STATUS, &lStatus);
}
while (lStatus != SPC_READY);

printf ("Board has stopped\n");
```

### Starting with interrupt driven mode

In contrast to the classic mode, the interrupt mode has no need for polling for the board's status. Starting your board in the interrupt driven mode does in the main not differ from the classic mode. But there has to be done some additional programming to prevent the program from hanging. The SPC\_STARTANDWAIT command doesn't return until the board has stopped. Big advantage of this mode is that it doesn't waste any CPU time for polling. The driver is just waiting for an interrupt and the System has full CPU time for other jobs. To benefit from this mode it is necessary to set up a program with at least two different tasks: One for starting the board and to be blocked waiting for an interrupt. The other one to make any kind of calculations or display activities.

### Command register

Register	Value	Direction	Description
SPC_COMMAND	0	read/write	Command register, of the board.
SPC_STARTANDWAIT	11		Starts the board with the current register settings in the interrupt driven mode.
SPC_STOP	20		Stops the board manually.

**If the board is started in the interrupt mode the task calling the start function will not return until the board has finished. If no trigger event is found or the external clock is not present, this function will wait until the program is terminated from the taskmanager (Windows) or from another console (Linux).**



To prevent the program from this deadlock, a second task must be used which can send the SPC\_STOP signal to stop the board. Another possibility, that does not require the need of a second task is to define a timeout value.

Register	Value	Direction	Description
SPC_TIMEOUT	295130	read/write	Defines a time in ms after which the function SPC_STARTANDWAIT terminates itself. Writing a zero defines infinite wait

This is the easiest and safest way to use the interrupt driven mode. If the board started in the interrupts mode it definitely will not return until either the recording has finished or the timeout time has expired. In that case the function will return with an error code. See the appendix for details.

The following excerpt of a sample program gives you an example of how to start the board in the interrupt driven mode. It is assumed that all board setup has been done before.

```
SpcSetParam (hDrv, SPC_TIMEOUT, 1000); // Define the timeout to 1000 ms = 1 second
nErr = SpcSetParam (hDrv, SPC_COMMAND, SPC_STARTANDWAIT); // Starts the board in the interrupt driven mode

if (nErr == ERR_TIMEOUT) // Checks for the timeout
    printf ("No trigger found. Timeout has expired.\n");
```

An example on how to get a second task that can do some monitoring on the running task and eventually send the SPC\_STOP command can be found on the Spectrum driver CD that has been shipped with your board. The latest examples can also be down loaded via our website at [www.spectrum-instrumentation.com](http://www.spectrum-instrumentation.com).

### Data organization

This chapter shows the data organization when using the board in standard mode.

In standard mode the data is organized on the board in two memory channels, named memory channel 0 and memory channel 1. The data in memory is organized depending on the used channels and the type of board. This is a result of the internal hardware structure of the board.

Channels																Sample ordering in standard mode on memory channel 0										Sample ordering in standard mode on memory channel 1										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																					
x																A0	A1	A2	A3	A4	A5	A6	A7	A8	A9											
x	x															A0	B0	A1	B1	A2	B2	A3	B3	A4	B4											
x	x	x	x													A0	B0	C0	D0	A1	B1	C1	D1	A2	B2											
x	x	x	x	x	x											A0	B0	C0	D0	E0	F0	G0	H0	A1	B1											
x																A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	
x	x															A0	B0	A1	B1	A2	B2	A3	B3	A4	B4	I0	J0	I1	J1	I2	J2	I3	J3	I4	J4	
x	x	x	x													A0	B0	C0	D0	A1	B1	C1	D1	A2	B2	I0	J0	K0	L0	I1	J1	K1	L1	I2	J2	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	A0	B0	C0	D0	E0	F0	G0	H0	A1	B1	I0	J0	K0	L0	M0	N0	O0	P0	I1	J1	

The samples are re-named for better readability. A0 is sample 0 of channel 0, C4 is sample 4 of channel 2, ...

## Sample format

The 16 bit samples in two's complement are stored in memory as sign extended 16 bit integer values. This would lead to a theoretical range of possible integer values from -32768...0...+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)



**Because the offset errors within the signal path are corrected on the digital side by adding or subtracting a constant value, the resulting range of values that is available for acquired data is reduced.**

The maximum decimal digital value range spans from -32000 up to +32000. The range of value therefore consists of 64001 different possible values, as the following table shows by example for an input range of  $\pm 5$  V:

Input voltage	decimal value	Hexadecimal value
+ 5V	+32000	7D00h
...		
0 V	0	0000h
...		
- 5V	-32000	8300h



**Depending on the existing offset error in the signal path not all of the reserved 768/767 values are effectively used for the offset error compensation. Therefore digital values above or below the  $\pm 32000$  boundaries might occur and indicate voltages above/below the selected range.**

## Reading out the data with SpcGetData

The function SpcGetData enables you to read out the data that is stored in the on-board memory during any of the standard recording modes easily after the acquisition has finished. Depending on your operating system, the function is called with a different amount of parameters. Please refer to the relating chapter earlier in this manual. The examples in this section are written in Visual C++ for Windows, so the examples differ a little bit for the use with linux.

As the data is read out individually for every memory channel, it is important to know where the data has been stored. Please refer to the data organization section, to get the information you need first.

Assuming that you know the memory channel or channels that contain the acquired data, you now have to decide whether you want to read out the whole memory or just one part of it. To select the area to be read out two values are needed by the function SpcGetData.

**The value 'start' as a 32 bit integer value**

This value defines the start of the memory area to be read out in samples. This result is, that you do not need to care for the number of bytes a single sample contains. If you want to read out the whole memory this value must be set to 0.

**The value 'len' as a 32 bit integer value**

This value defines the number of samples that are read out, beginning with the first sample defined by the 'start' value mentioned above. If you want to read out the whole on-board memory you need to program the „len“ parameter to the before programmed memory size. At this point please keep in mind that depending on the activated channels there may be more than one board channel in one memory channel. This „len“ value must be a total memsize for all channels that are acquired in that memory channel. As a result that means if acquiring two channels to memory channel 0 the „len“ value must be set to „2 \* memsize“.

**Multiplexed data**

Depending on the activated channels and the board type several channels could be stored in one memory channel. As a result that means that „start“ and „len“ parameter have to be multiplied by the number of channels per memory channel (module). If for example two channels have been acquired into one memory channel a call like:

```
SpcGetData (hDrv, 0, 2 * 4096, 2 * 2048, Data);
```

reads out data of both channels from memory channel 0 starting at sample position 4k and a length of 2k. The Data array must be of course large enough to hold data of both channels (in that case  $2 * 2k = 4k$  of data).

**Standard mode**

Reading out the data is really easy, if a recording modes is used that stores non multiplexed data in the dedicated memory channels. The next example shows, how to read out the data after having recorded two channels that have been written without multiplexing to both memory channels.

Example for SpcGetData, no memory allocation error checking performed:

```
for (i = 0; i < 2; i++) // both memory channels have been used
    pData[i] = (ptr16) malloc (lMemsize * lBytesPerSample); // allocate memory for the data pointers
// with the maximum size (lMemsize)

SpcGetData (hDrv, 0, 0, lMemsize, (dataptr) pData[0]); // no demultiplexing is necessary on channel 0
SpcGetData (hDrv, 1, 0, lMemsize, (dataptr) pData[1]); // neither it is on channel 1
```

If you use two channels for recording using only one memory channel or four channels, the data in the memory channel(s) is multiplexed and needs to be unsorted by the user. The following example shows how to unsort the data for the recording of two channels using memory channel 0.

```
for (i = 0; i < 2; i++) // 2 channels to read out from 1 memory channel
    pData[i] = (ptr16) malloc (lMemsize * lBytesPerSample); // allocate memory for the data pointers
// with the maximum size (lMemsize) per channel

pTmp = (ptr16) malloc (lMemsize * 2 * lBytesPerSample); // allocate temporary buffer for copy

SpcGetData (hDrv, 0, 0, 2 * lMemsize, (dataptr) pTmp); // get both channels together
// from memory channel 0

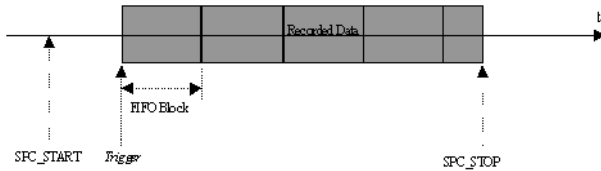
for (i = 0; i < lMemsize; i++) // split data in the two channels
{
    pData[0][i] = pTmp[(2 * i)];
    pData[1][i] = pTmp[(2 * i) + 1];
}

free (pTmp); // free the temporary buffer
```

## FIFO Mode

### Overview

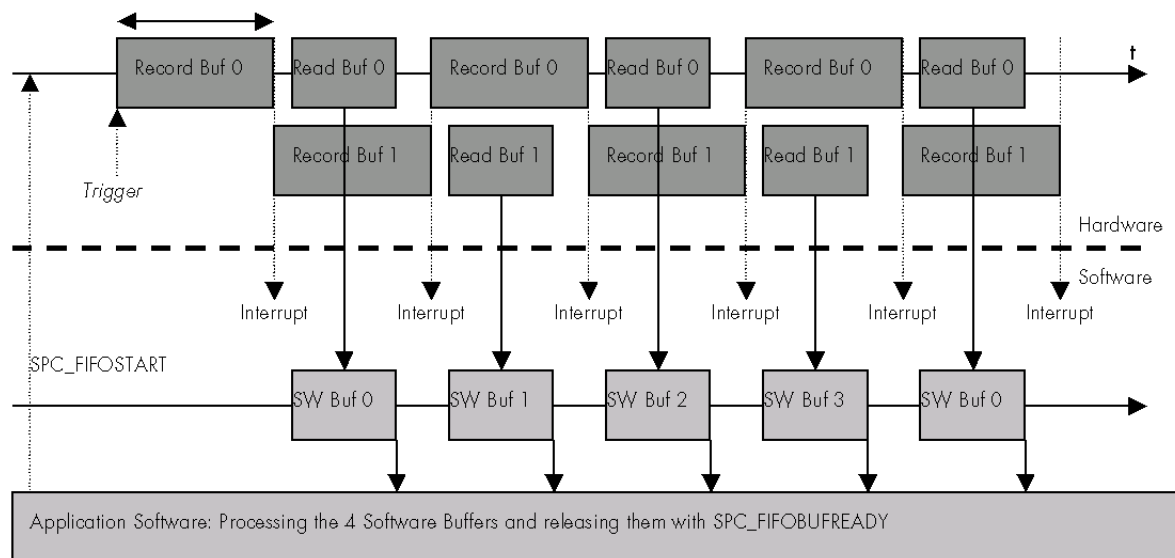
#### General Information



The FIFO mode allows to record data continuously and transfer it online to the PC (acquisition boards) or allows to write data continuously from the PC to the board (generation boards). Therefore the on-board memory of the board is used as a continuous buffer. On the PC the data can be used for any calculation or can be written to hard disk while recording is running (acquisition boards) or the data can be read from hard disk and calculated online before writing it to the board.

FIFO mode uses interrupts and is supported by the drivers on 32 bit and 64 bit operating systems. Start of FIFO mode waits for a trigger event. If you wish to start FIFO mode immediately, you may use the software trigger. FIFO mode can be used together with the options Multiple Recording/Replay and Gated Sampling/Replay. Details on this can be found in the appropriate chapters about the options.

#### Background FIFO Read



On the hardware side the board memory is split in two buffers of the same length. These buffers can be up to half of the on-board memory in size. In addition to the hardware buffers the driver holds up to 256 software buffers of the same length as the hardware buffers are. Whenever a hardware buffer is full with data the hardware generates an interrupt and the driver transfers this hardware buffer to the next software buffer that is available. While transferring one buffer to the PC, the other one is filled up with data. The driver is doing this job automatically in the background.

After the driver has finished transferring the data, the application software gets a signal and can process data (e.g. stores data to hard disk or makes some calculations). After processing the data the application software tells the driver that he can again use the software buffer for acquisition data.

This two stages buffering has big advantages when running FIFO mode at the speed limit. The software buffers extremely expand the acquisition time that can be buffered and protects the whole system against buffer overruns.

#### Speed Limitations

The FIFO mode is running continuously all the time. Therefore the data must be read out from the board (data acquisition) or written to the board (data generation) at least with the same speed that it is recorded/replayed. If data is read out from the board or written to the board more slowly, the hardware buffers will overrun at a certain point and FIFO mode is stopped.

One bottleneck with the FIFO mode is the PCI bus. The standard PCI bus is theoretically capable of transferring data with 33 MHz and 32 Bit. As a result a maximum burst transfer rate of 132 MByte per second can be achieved. As several devices can share the PCI bus this maximum transfer rate is only available to a short transfer burst until a new bus arbitration is necessary. In real life the continuous transfer rate is limited to approximately 100-110 MBytes per second. The maximum FIFO speed one can achieve heavily depends on the PC system and the operating system and varies from system to system.

The maximum sample rate one can run in continuous FIFO mode depends on the number of activated channels:



	Theoretical maximum sample rate	PCI Bus Throughput
1 Channel	50 MS/s	[1 Channel] x [2 Bytes per sample] * 50 MS/s = 100 MB/s
2 Channels	25 MS/s	[2 Channels] x [2 Bytes per sample] * 25 MS/s = 100 MB/s
4 Channels	12.5 MS/s	[4 Channels] x [2 Bytes per sample] * 12.5 MS/s = 100 MB/s
8 Channels	6.25 MS/s	[8 Channels] x [2 Bytes per sample] * 6.25 MS/s = 100 MB/s

When using FIFO mode together with one of the options that allow to have gaps in the acquisition like Multiple Recording or Gated Sampling one can even run the board with higher sample rates. It just has to be sure that the average sample rate (calculated with acquisition time and gap) does not exceed the above mentioned sample rate limitations.

The sample rate that can be run in one of these mode is depending on the number of channels that have been activated. Due to the internal structure of the board this is limited to a internal throughput of 250 MB/s (125 MS/s):

	Maximum sample rate that can be programmed	Internal throughput
1 Channel	125 MS/s	[1 Channel] x [2 Bytes per sample] x 125 MS/s = 250 MB/s
2 Channels	62.5 MS/s	[2 Channels] x [2 Bytes per sample] x 62.5 MS/s = 250 MB/s
4 Channels	31.25 MS/s	[4 Channels] x [2 Bytes per sample] x 31.25 MS/s = 250 MB/s
8 Channels	15.625 MS/s	[8 Channels] x [2 Bytes per sample] x 15.625 MS/s = 250 MB/s

## Programming

The setup of FIFO mode is done with a few additional software registers described in this chapter. All the other settings can be used as described before. In FIFO mode the register SPC\_MEMSIZE and SPC\_POSTTRIGGER are not used.

### Software Buffers

This register defines the number of software buffers that should be used for FIFO mode. The number of hardware buffers is always two and can not be changed by software.

Register	Value	Direction	Description
SPC_FIFO_BUFFERS	60000	r/w	Number of software buffers to be used for FIFO mode. Value has to be between 2 and 256

When this manual was printed there are a total of 256 buffers possible. However if there are changes and enhancements to the driver in the future it will be informative to read out the number of buffers the new driver version can hold.

Register	Value	Direction	Description
SPC_FIFO_BUFADRCNT	60040	r	Read out the number of available FIFO buffers

The length of each buffer is defined in bytes. This length is used for hardware and software buffers as well. Both have the same length. The maximum length that can be used is depending on the installed on-board memory.

Register	Value	Direction	Description
SPC_FIFO_BUFLN	60010	r/w	Length of each buffer in bytes. Must be a multiple of 1024 bytes.

Each FIFO buffer can be a maximum of half the memory. Be aware that the buffer length is given in overall bytes not in samples. Therefore the value has to be calculated depending on the activated channels and the resolution of the board:

### Analog acquisition or generation boards

	Buffer length to be programmed in Bytes			
	8 bit resolution	12 bit resolution	14 bit resolution	16 bit resolution
1 Channel	1 x [Samples in Buffer]	1 x 2 x [Samples in Buffer]	1 x 2 x [Samples in Buffer]	1 x 2 x [Samples in Buffer]
2 Channels	2 x [Samples in Buffer]	2 x 2 x [Samples in Buffer]	2 x 2 x [Samples in Buffer]	2 x 2 x [Samples in Buffer]
4 Channels	4 x [Samples in Buffer]	4 x 2 x [Samples in Buffer]	4 x 2 x [Samples in Buffer]	4 x 2 x [Samples in Buffer]
8 Channels	8 x [Samples in Buffer]	8 x 2 x [Samples in Buffer]	8 x 2 x [Samples in Buffer]	8 x 2 x [Samples in Buffer]

### Digital I/O (701x or 702x) or pattern generator boards (72xx)

	Buffer length to be programmed in Bytes			
	8 bit mode	16 bit mode	32 bit mode	64 bit mode
	[Samples in Buffer]	2 x [Samples in Buffer]	4 x [Samples in Buffer]	8 x [Samples in Buffer]

### Digital I/O board 7005 only

	Buffer length to be programmed in Bytes				
	1 bit mode	2 bit mode	4 bit mode	8 bit mode	16 bit mode
1 Channel	1/8 x [Samples in Buffer]	1/4 x [Samples in Buffer]	1/2 x [Samples in Buffer]	[Samples in Buffer]	2 x [Samples in Buffer]

We at Spectrum achieved best results when programming the buffer length to a number of samples that can hold approximately 100 ms of data. However if going to the limit of the PCI bus with the FIFO mode or when having buffer overruns it can be useful to have larger FIFO buffers to buffer more data in it.

When the goal is a fast update in FIFO mode smaller buffers and a larger number of buffers can be a better setup.

Register	Value	Direction	Description
SPC_FIFO_BUFADRO	60100	r/w	address of FIFO buffer 0. Must be allocated by application program
SPC_FIFO_BUFADR1	60101	r/w	address of FIFO buffer 1. Must be allocated by application program
...			...
SPC_FIFO_BUFADR255	60355	r/w	address of FIFO buffer 255. Must be allocated by application program

The driver handles the programmed number of buffers. To speed up FIFO transfer the driver uses buffers that are allocated and maintained by the application program. Before starting the FIFO mode the addresses of the allocated buffers must be set to the driver.

Example of FIFO buffer setup. Neither memory allocation nor error checking is done in the example to improve readability:

```
// ----- setup FIFO buffers -----
SpcSetParam (hDrv, SPC_FIFO_BUFFERS, 64); // 64 FIFO buffers used in the example
SpcSetParam (hDrv, SPC_FIFO_BUFLEN, 8192); // Each FIFO buffer is 8 kBytes long

// ----- allocate memory for data -----
for (i = 0; i < 64; i++)
    pData[i] = (ptr16) malloc (8192); // memory allocation for 12, 14, 16 bit analog boards
// and digital boards
// pbyData[i] = (ptr8) malloc (8192); // memory allocation for 8 bit analog boards

// ----- tell the used buffer addresses to the driver -----
for (i = 0; i < 64; i++)
    nErr = SpcSetParam (hDrv, SPC_FIFO_BUFADRO + i, (int32) pData[i]); // for 12, 14, 16 bit analog boards
// and digital boards only
// nErr = SpcSetParam (hDrv, SPC_FIFO_BUFADRO + i, (int32) pbyData[i]); // for 8 bit analog boards only
```

When using 64 bit Linux systems it is necessary to program the buffer addresses using a special function as the SpcSetParam function is limited to 32 bit as a parameter. Under 64 bit Linux systems all addresses are 64 bit wide. Please use the function SpcSetAdr as described in the introduction and shown in the example below:



```
// ----- tell the used buffer addresses to the driver (Linux 32 and 64 bit systems) -----
for (i = 0; i < 64; i++)
    nErr = SpcSetAdr (hDrv, SPC_FIFO_BUFADRO + i, (void*) pData[i]);
```

### Buffer processing

The driver counts all the software buffers that have been transferred. This number can be read out from the driver to know the exact amount of data that has been transferred.

Register	Value	Direction	Description
SPC_FIFO_BUFCOUNT	60020	r	Number of transferred buffers until now

If one knows before starting FIFO mode how long this should run it is possible to program the number of buffers that the driver should process. After transferring this number of buffer the driver will automatically stop. If FIFO mode should run endless a zero must be programmed to this register. Then the FIFO mode must be stopped by the user.

Register	Value	Direction	Description
SPC_FIFO_BUFMAXCNT	60030	r/w	Number of buffers to be transferred until automatic stop. Zero runs endless

## FIFO mode

In normal applications the FIFO mode will run in a loop and process one buffer after the other. There are a few special commands and registers for the FIFO mode:

Register	Value	Direction	Description
SPC_COMMAND	0	w	Command register. Allowed values for FIFO mode are listed below
SPC_FIFOSTART	12		Starts the FIFO mode and waits for the first data interrupt
SPC_FIFOWAIT	13		Waits for the next buffer interrupt
SPC_FIFOSTARTNOWAIT	14		Start the card and return immediately without waiting for the first data interrupt
SPC_STOP	20		Stops the FIFO mode

The start command and the wait command both wait for the signal from the driver that the next buffer has to be processed. This signal is generated by the driver on receiving an interrupt from the hardware. While waiting none of these commands waste cpu power (no polling mode). If for any reason the signal is not coming from the hardware (e.g. trigger is not found) the FIFO mode must be stopped from a second task with a stop command.

This handshake command tells the driver that the application has finished it's work with the software buffer. The both commands SPC\_FIFOWAIT (SPC\_FIFOSTART) and SPC\_FIFO\_BUFFERS form a simple but powerful handshake protocol between application software and board driver.

Register	Value	Direction	Description
SPC_FIFO_BUFREADY	60050	w	FIFO mode handshake. Application has finished with that buffer. Value is index of buffer

**Backward compatibility: This register replaces the formerly known SPC\_FIFO\_BUFREADY0... SPC\_FIFO\_BUFREADY15 commands. It has the same functionality but can handle more FIFO buffers. For backward compatibility the older commands still work but are still limited to 16 buffers.**



## Example FIFO acquisition mode

This example shows the main loop of a FIFO acquisition. The example is a part of the FIFO examples that are available for each board on CD. The example simply counts the buffers when it receives a new buffer from the driver and returns control immediately back to the driver.

FIFO acquisition example:

```
nBufIdx = 0;
lBufCount = 0;
lCommand = SPC_FIFOSTART;

printf ("Start\n");
do
{
    nErr = SpcSetParam (hDrv, SPC_COMMAND, lCommand);
    lCommand = SPC_FIFOWAIT;

    // ----- perform any data calculation or hard disk recording (in example only counting buffers)-----
    printf ("FIFO Buffer %ld\n", lBufCount++);

    // ----- buffer is ready -----
    SpcSetParam (hDrv, SPC_FIFO_BUFREADY, nBufIdx);

    // ----- next Buffer -----
    nBufIdx++;
    if (nBufIdx == MAX_BUF)
        nBufIdx = 0;
}
while (nErr == ERR_OK);
```

## Data organization

When using FIFO mode data in memory is organized in some cases a little bit different then in standard mode. This is a result of the internal hardware structure of the board. The organization of data is depending on the activated channels:

Channels																Sample ordering in standard mode on memory channel 0													
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15														
x	x															A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13
x	x															A0	B0	A1	B1	A2	B2	A3	B3	A4	B4	A5	B5	A6	B6
x	x	x	x													A0	B0	C0	D0	A1	B1	C1	D1	A2	B2	C2	D2	A3	B3
x	x	x	x	x	x											A0	B0	C0	D0	E0	F0	G0	H0	A1	B1	C1	D1	E1	F1
x																A0	I0	A1	I1	A2	I2	A3	I3	A4	I4	A5	I5	A6	I6
x	x															A0	I0	B0	J0	A1	I1	B1	J1	A2	I2	B2	J2	A3	I3
x	x	x	x													A0	I0	B0	J0	C0	K0	D0	L0	A1	I1	B1	J1	C1	K1
x	x	x	x	x	x	x	x									A0	I0	B0	J0	C0	K0	D0	L0	E0	M0	F0	N0	G0	O0

The samples are re-named for better readability. A0 is sample 0 of channel 0, C4 is sample 4 of channel 2, ...

The following example shows how to sort the channel data when using 4 channels in FIFO mode:

```
for (i = 0; i < (lBufferSizeInSamples / 4); i++)
{
    Data[0][i] = FIFOBuffer[i * 4 + 0];
    Data[1][i] = FIFOBuffer[i * 4 + 2];
    Data[2][i] = FIFOBuffer[i * 4 + 1];
    Data[3][i] = FIFOBuffer[i * 4 + 3];
}
```

## Sample format

The sample format in FIFO mode does not differ from the one of the standard (non FIFO) mode. Please refer to the relating passage concerning the sample format in the standard acquisition chapter.

# Clock generation

## Overview

The Spectrum boards offer a wide variety of different clock modes to match all the customers needs. All the clock modes are described in detail with programming examples below. This chapter simply gives you an overview which clock mode to select:

### Standard internal sample rate

PLL with internal 40 MHz reference. This is the easiest way to generate a sample rate with no need for additional external clock signals. The sample rate has a fine resolution.

### Quartz and divider

Internal quartz clock with divider. For applications that need a lower clock jitter than the PLL produces. The possible sample rates are restricted to the values of the divider.

### External reference clock

PLL with external 1 MHz to 125 MHz reference clock. This provides a very good clock accuracy if a stable external reference clock is used. It also allows the easy synchronization with an external source.

### Direct external 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.

**Direct external clock is not available for MC.49xx/MX.49xx cards. Please use external reference clock mode instead.**



### External clock with divider

The externally fed in clock can be divided to generate a low-jitter sample rate of a slower speed than the external clock available.

**Direct external clock with divider is not available for MC.49xx/MX.49xx cards. Please use external reference clock mode instead.**



**There is a more detailed description of the clock generation part available as an application note. There some more background information and details of the internal structure are explained.**



## Internally generated sample rate

### Standard internal sample rate

The internal sample rate is generated in default mode by a PLL and dividers out of an internal 40 MHz frequency reference. In most cases the user does not need to care on how the desired sample rate is generated by multiplying and dividing internally. You simply write the desired sample rate to the according register shown in the table below. 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 sample rate that is matching your desired one best.

Register	Value	Direction	Description
SPC_SAMPLERATE	20000	w	Defines the sample rate in Hz for internal sample rate generation.
		r	Read out the internal sample rate that is nearest matching to the desired one.

If a sample rate is generated internally, you can additionally enable the clock output. The clock will be available on the external clock connector and can be used to synchronize external equipment with the board.

Register	Value	Direction	Description
SPC_EXTERNOUT	20110	r/w	Enables clock output on external clock connector. Only possible with internal clocking. (old name)
SPC_CLOCKOUT	20110	r/w	Enables clock output on external clock connector. Only possible with internal clocking. (new name)

Example on writing and reading internal sample rate

```
SpcSetParam (hDrv, SPC_SAMPLERATE, 1000000);           // Set internal sample rate to 1 MHz
SpcSetParam (hDrv, SPC_CLOCKOUT, 1);                  // enable the clock output of that 1 MHz
SpcGetParam (hDrv, SPC_SAMPLERATE, &lSamplerate);      // Read back the sample rate that has been programmed
printf („Samplerate = %d\\n“, lSamplerate);            // print it. Output should be „Samplerate = 1000000“
```

### Minimum internal sample rate

The minimum internal sample rate is limited on all boards to 1 kHz and the maximum sample rate depends on the specific type of board. The maximum sample rates for your type of board are shown in the tables below.

### Maximum internal sampling rate in normal mode

Version	Sampling rate	Version	Sampling rate
4710	100 kS/s	4711	100 kS/s
4720	250 kS/s	4721	250 kS/s
4730	500 kS/s	4731	500 kS/s

### Using plain quartz without PLL

In some cases it is useful for the application not to have the on-board PLL activated. Although the PLL used on the Spectrum boards is a low-jitter version it still produces more clock jitter than a plain quartz oscillator. For these cases the Spectrum boards have the opportunity to switch off the PLL by software and use a simple clock divider.

Register	Value	Direction	Description
SPC_PLL_ENABLE	20030	r/w	A „1“ enables the PLL mode (default) or disables it by writing a 0 to this register

The sample rates that could be set are then limited to the quartz speed divided by one of the below mentioned dividers. The quartz used on the board is similar to the maximum sample rate the board can achieve. As with PLL mode it's also possible to set a desired sample rate and read it back. The result will then again be the best matching sample rate.

Available divider values

1	2	4	8	10	16	20	40	50	80	100	200
400	500	800	1000	2000							

### External reference clock

If you have an external clock generator with a extremely stable frequency, you can use it as a reference clock. You can connect it to the external clock connector and the PLL will be fed with this clock instead of the internal reference. Due to the fact that the driver needs to know the external fed in frequency for an exact calculation of the sample rate you must set the the SPC\_REFERENCECLOCK register accordingly as shown in the table below:

Register	Value	Direction	Description
SPC_REFERENCECLOCK	20140	r/w	Programs the external reference clock in the range from 1 MHz to 125 MHz.
0			Internal reference is used for internal sampling rate generation
External sample rate in Hz as an integer value			External reference is used. You need to set up this register exactly to the frequency of the external fed in clock.

The driver automatically sets the PLL to achieve the desired sample rate. Therefore it examines the reference clock and the sample rate registers.

Example of reference clock:

```
SpcSetParam (hDrv, SPC_EXTERNALCLOCK, 0);           // Set to internal clock
SpcSetParam (hDrv, SPC_REFERENCECLOCK, 10000000);  // Reference clock that is fed in is 10 MHz
SpcSetParam (hDrv, SPC_SAMPLERATE, 25000000);      // We want to have 25 MHz as sample rate
```

### Termination of the clock input

If the external connector is used as an input, either for feeding in an external reference clock or for external clocking you can enable a 50 Ohm termination on the board. If the termination is disabled, the impedance is high. Please make sure that your source is capable of driving that current and that it still fulfills the clock input specification as given in the technical data section.

Register	Value	Direction	Description
SPC_CLOCK50OHM	20120	read/write	A „1“ enables the 50 Ohm termination at the external clock connector. Only possible, when using the external connector as an input.

## External clocking

### Direct external clock

An external clock can be fed in on the external clock connector of the board. This can be any clock, that matches the specification of the card. The external clock signal can be used to synchronize the card on a system clock or to feed in an exact matching sample rate.

Register	Value	Direction	Description
SPC_EXTERNALOCK	20100	read/write	A „1“ enables the external clock input. If external clock input is disabled, internal clock will be used.

The maximum values for the external clock is board dependant and shown in the table below.

### Termination of the clock input

If the external connector is used as an input, either for feeding in an external reference clock or for external clocking you can enable a 50 Ohm termination on the board. If the termination is disabled, the impedance is high. Please make sure that your source is capable of driving that current and that it still fulfills the clock input specification as given in the technical data section.

Register	Value	Direction	Description
SPC_CLOCK50OHM	20120	read/write	A „1“ enables the 50 Ohm termination at the external clock connector. Only possible, when using the external connector as an input.

### External sampling rate

Version	Minimum	Maximum	Version	Minimum	Maximum
4710	DC	100 kS/s	4711	DC	100 kS/s
4720	DC	250 kS/s	4721	DC	250 kS/s
4730	DC	500 kS/s	4731	DC	500 kS/s

**An external sample rate above the mentioned maximum can cause damage to the board.**



### Ranges for external sample rate

Due to the internal structure of the board it is essential to know for the driver in which clock range the external clock is operating. The external range register must be set according to the clock that is fed in externally.

Register	Value	Direction	Description
SPC_EXTERNRANGE	20130	read/write	Defines the range of the actual fed in external clock. Use one of the below mentioned ranges
EXRANGE_SINGLE	2		External Range Single
EXRANGE_BURST_S	4		External Range Burst S
EXRANGE_BURST_M	8		External Range Burst M
EXRANGE_BURST_L	16		External Range Burst X
EXRANGE_BURST_XL	32		External Range Burst XL

**The range must not be left by more than 5 % when the board is running. Remember that the ranges depend on the activated channels as well, so a different board setup for external clocking must always include the related clock ranges.**



This table below shows the ranges that are defined by the different range registers mentioned above. The range depends on the activated channels and the mode the board is used in. Please be sure to select the correct range. Otherwise it is possible that the board will not run properly.

Activated channels																Mode	EXRANGE_SINGLE
0	1	2	3	4	5	6	7	7	8	10	11	12	13	14	15		
x																Standard/FIFO	≤ 500 kS/s
x	x															Standard/FIFO	≤ 500 kS/s
x	x	x	x													Standard/FIFO	≤ 500 kS/s
x	x	x	x	x	x	x	x									Standard/FIFO	≤ 500 kS/s
x								x								Standard/FIFO	≤ 500 kS/s
x	x							x	x							Standard/FIFO	≤ 500 kS/s
x	x	x	x					x	x	x	x					Standard/FIFO	≤ 500 kS/s
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Standard/FIFO	≤ 500 kS/s

How to read this table? No matter what sampling rate you activate, the external clock range always has to be set to EXRANGE\_SINGLE.

Example:

```
SpcSetParam (hDrv, SPC_CHENABLE, CHANNEL0 | CHANNEL1 | CHANNEL2 | CHANNEL3); // activate 4 channels
SpcSetParam (hDrv, SPC_EXTERNALCLOCK, 1); // activate external clock
SpcSetParam (hDrv, SPC_EXTERNRANGE, EXRANGE_SINGLE); // set external range to Single
```

**External clock with divider**

The extra clock divider can be used to divide an external fed in clock by a fixed value. The external clock must be > 1 MS/s. This divided clock is used as a sample clock for the board.

Register	Value	Direction	Description
SPC_CLOCKDIV	20040	read/write	Extra clock divider for external samplerate. Allowed values are listed below

Available divider values

1	2	4	8	10	16	20	40	50	80	100	200
400	500	800	1000	2000							



## Trigger modes and appendant registers

### General Description

The trigger modes of the Spectrum MI, MC and MX A/D boards are very complex and give you the possibility to detect nearly any trigger event, you can think of.

You can choose between seven external TTL trigger modes and up to 18 internal trigger modes including software and channel trigger, depending on your type of board. Five of the internal 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 board's registers for the desired mode.

Every analog Spectrum board has one dedicated SMB connector mounted in it's bracket for feeding in an external trigger signal or outputting a trigger signal of an internal trigger event. Due to the fact that only one connector is available for external trigger I/O, it is not possible to forward the fed in external trigger signal to another board. If this is however necessary, you need to split up the external trigger signal before.

### Software trigger

The software trigger is the easiest way of triggering any Spectrum board. The acquisition or replay of data will start immediately after starting the board. The only delay results from the time the board needs for its setup.



Register	Value	Direction	Description
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_SOFTWARE	0		Sets the trigger mode to software, so that the recording/replay starts immediately.

In addition to the softwaretrigger (free run) it is also possible to force a triggerevent by software while the board is waiting for an internal or external trigger event. Therefore you can use the board command shown in the following table.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board.
SPC_FORCETRIGGER	16		Forces a trigger event if the hardware is still waiting for a trigger event. Needs a base board hardware version $\geq 7.x$ .

Due to the fact that the software trigger is an internal trigger mode, you can optionally enable the external trigger output to generate a high active trigger signal, which indicates when the data acquisition or replay begins. This can be useful to synchronize external equipment with your Spectrum board.

Register	Value	Direction	Description
SPC_TRIGGEROUT	40100	r/w	Defines the data direction of the external trigger connector.
	0		The trigger connector is not used and the line driver is disabled.
	1		The trigger connector is used as an output that indicates a detected internal trigger event.

Example for setting up the software trigger:

```
SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_SOFTWARE); // Internal software trigger mode is used
SpcSetParam (hDrv, SPC_TRIGGEROUT, 1); // And the trigger output is enabled
```

### External TTL trigger

Enabling the external trigger input is done, if you choose one of the following external trigger modes. The dedicated register for that operation is shown below.

Register	Value	Direction	Description
SPC_TRIGGERMODE	40000	r/w	
TM_TTLPOS	20000		Sets the trigger mode for external TTL trigger to detect positive edges.
TM_TTLNEG	20010		Sets the trigger mode for external TTL trigger to detect negative edges
TM_TTLBOTH	20030		Sets the trigger mode for external TTL trigger to detect positive and negative edges
TM_TTLHIGH_LP	20001		Sets the trigger mode for external TTL trigger to detect HIGH pulses that are longer than a programmed pulsewidth.
TM_TTLHIGH_SP	20002		Sets the trigger mode for external TTL trigger to detect HIGH pulses that are shorter than a programmed pulsewidth.
TM_TTLOW_LP	20011		Sets the trigger mode for external TTL trigger to detect LOW pulses that are longer than a programmed pulsewidth.
TM_TTLOW_SP	20012		Sets the trigger mode for external TTL trigger to detect LOW pulses that are shorter than a programmed pulsewidth.

If you choose an external trigger mode the SPC\_TRIGGEROUT register will be overwritten and the trigger connector will be used as an input anyways.

Register	Value	Direction	Description
SPC_TRIGGEROUT	40100	r/w	Defines the data direction of the external trigger connector.
	X		If external triggermodes are used, this register will have no effect.

As the trigger connector is used as an input, you can decide whether the input is 50 Ohm terminated or not. 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 fulfill the trigger input specification given in the technical data section. If termination is disabled, the input is at high impedance.

Register	Value	Direction	Description
SPC_TRIGGER50OHM	40110	r/w	A „1“ sets the 50 Ohm termination, if the trigger connector is used as an input for external trigger signals. A „0“ sets the 1 MOhm termination

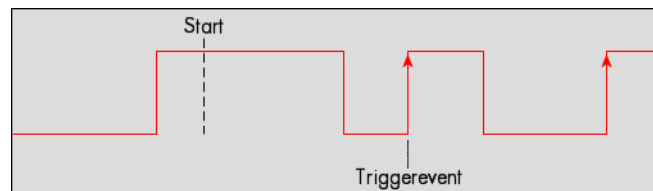
The following short example shows how to set up the board for external positive edge TTL trigger. The trigger input is 50 Ohm terminated. The different modes for external TTL trigger will be described in detail on the next few passages.

```
SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_TTLPOS); // External positive TTL edge trigger
SpcSetParam (hDrv, SPC_TRIGGER50OHM, 1); // and the 50 Ohm termination of the trigger input are used
```

## Edge triggers

### Positive 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 trigger event will then be detected, if the actual recording/replay has finished and the board is armed and waiting for a trigger again.



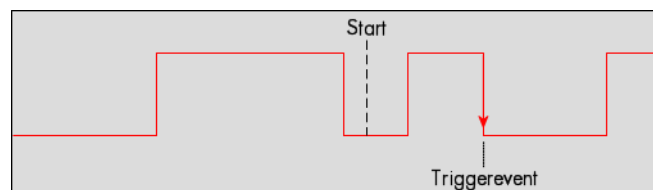
Register	Value	Direction	Description
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board
TM_TTLPOS	20000		Sets the trigger mode for external TTL trigger to detect positive edges

Example on how to set up the board for positive TTL trigger:

```
SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_TTLPOS); // Setting up external TTL trigger to detect positive edges
```

### Negative TTL trigger

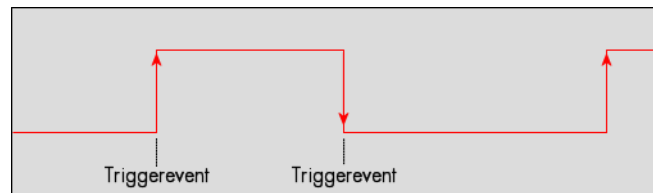
This mode is for detecting the falling edges of an external TTL signal. The board will trigger on the first falling edge that is detected after starting the board. The next trigger event will then be detected, if the actual recording/replay has finished and the board is armed and waiting for a trigger again.



Register	Value	Direction	Description
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_TTLNEG	20010		Sets the trigger mode for external TTL trigger to detect negative edges.

### Positive and negative TTL trigger

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.

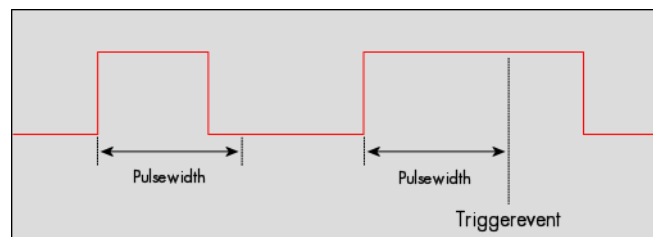


Register	Value	Direction	Description
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_TTLBOTH	20030		Sets the trigger mode for external TTL trigger to detect positive and negative edges.

### Pulsewidth triggers

#### TTL pulsewidth trigger for long HIGH pulses

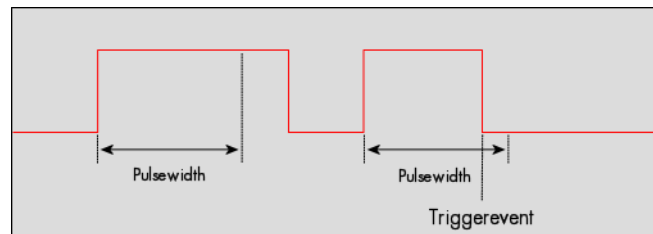
This mode is for detecting HIGH pulses of an external TTL signal that are longer than a programmed pulsewidth. If the pulse is shorter than the programmed pulsewidth, no trigger will be detected. The board will trigger on the first pulse matching the trigger condition after starting the board. The next 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_PULSEWIDTH	44000	r/w	Sets the pulsewidth in samples. Values from 2 to 255 are allowed.
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_TTLHIGH_LP	20001		Sets the trigger mode for external TTL trigger to detect HIGH pulses that are longer than a programmed pulsewidth.

#### TTL pulsewidth trigger for short HIGH pulses

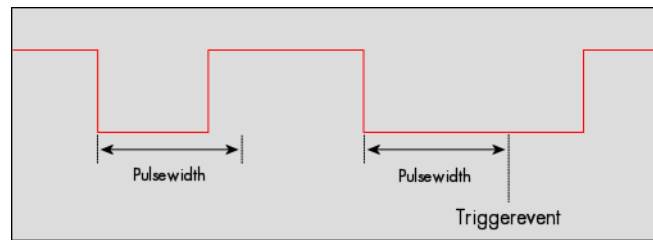
This mode is for detecting HIGH pulses of an external TTL signal that are shorter than a programmed pulsewidth. If the pulse is longer than the programmed pulsewidth, no trigger will be detected. The board will trigger on the first pulse matching the trigger condition after starting the board. The next 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_PULSEWIDTH	44000	r/w	Sets the pulsewidth in samples. Values from 2 to 255 are allowed.
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_TTLHIGH_SP	20002		Sets the trigger mode for external TTL trigger to detect HIGH pulses that are shorter than a programmed pulsewidth.

### TTL pulsewidth trigger for long LOW pulses

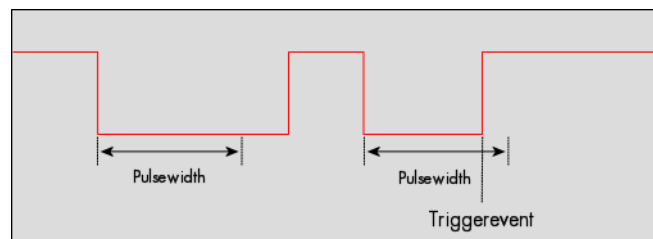
This mode is for detecting LOW pulses of an external TTL signal that are longer than a programmed pulsewidth. If the pulse is shorter than the programmed pulsewidth, no trigger will be detected. The board will trigger on the first pulse matching the trigger condition after starting the board. The next 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_PULSEWIDTH	44000	r/w	Sets the pulsewidth in samples. Values from 2 to 255 are allowed.
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_TTLLOW_LP	20011		Sets the trigger mode for external TTL trigger to detect LOW pulses that are longer than a programmed pulsewidth.

### TTL pulsewidth trigger for short LOW pulses

This mode is for detecting LOW pulses of an external TTL signal that are shorter than a programmed pulsewidth. If the pulse is longer than the programmed pulsewidth, no trigger will be detected. The board will trigger on the first pulse matching the trigger condition 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_PULSEWIDTH	44000	r/w	Sets the pulsewidth in samples. Values from 2 to 255 are allowed.
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_TTLLOW_SP	20012		Sets the trigger mode for external TTL trigger to detect LOW pulses that are shorter than a programmed pulsewidth.

```
SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_TTLHIGH_LP); // Setting up external TTL trigger to detect high pulses
SpcSetParam (hDrv, SPC_PULSEWIDTH, 50); // that are longer than 50 samples.
```

## Channel Trigger

### Overview of the channel trigger registers

The channel trigger modes are the most common modes, compared to external equipment like oscilloscopes. The 17 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 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 trigger. The following table shows the according registers for the two general channel trigger modes.

Register	Value	Direction	Description
SPC_TRIGGERMODE	40000	r/w	Sets the triggermode for the board.
TM_CHANNEL	20040		Enables the channel trigger mode so that only one channel can be a trigger source.
TM_CHOR	35000		Enables the channel trigger mode so that more than one channel can be a trigger source.

If you have set the general triggermode to channel trigger you must set the all of the channels to their modes according to the following table.



**So even if you use TM\_CHANNEL and only want to observe one channel, you need to deactivate all other channels. You can do this by setting the channel specific register to the value TM\_CHXOFF.**

The tables lists the maximum of the available channel mode registers for your card's series. So it is possible 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, an error message will be returned.

Register	Value	Direction	Description
SPC_TRIGGERMODE0	40200	read/write	Sets the channel trigger for channel 0. Channeltrigger must be activated with SPC_TRIGGERMODE.
SPC_TRIGGERMODE1	40201	read/write	Sets the channel trigger for channel 1. Channeltrigger must be activated with SPC_TRIGGERMODE.
...	...	...	...
SPC_TRIGGERMODE14	40214	read/write	Sets the channel trigger for channel 14. Channeltrigger must be activated with SPC_TRIGGERMODE.
SPC_TRIGGERMODE15	40215	read/write	Sets the channel trigger for channel 15. Channeltrigger must be activated with SPC_TRIGGERMODE.
TM_CHXOFF	10020		Channel is not used for trigger detection.
TM_CHXPOS	10000		Enables the trigger detection for positive edges
TMCHXNEG	10010		Enables the trigger detection for negative edges
TMCHXBOTH	10030		Enables the trigger detection for positive and negative edges
TM_CHXPOS_LP	10001		Enables the pulsewidth trigger detection for long positive pulses
TMCHXNEG_LP	10011		Enables the pulsewidth trigger detection for long negative pulses
TM_CHXPOS_SP	10002		Enables the pulsewidth trigger detection for short positive pulses
TMCHXNEG_SP	10012		Enables the pulsewidth trigger detection for short negative pulses
TM_CHXPOS_GS	10003		Enables the steepness trigger detection for flat positive pulses
TMCHXNEG_GS	10013		Enables the steepness trigger detection for flat negative pulses
TM_CHXPOS_SS	10004		Enables the steepness trigger detection for steep positive pulses
TMCHXNEG_SS	10014		Enables the steepness trigger detection for steep negative pulses
TM_CHXWINENTER	10040		Enables the window trigger for entering signals
TM_CHXWINLEAVE	10050		Enables the window trigger for leaving signals
TM_CHXWINENTER_LP	10041		Enables the window trigger for long inner signals
TM_CHXWINLEAVE_LP	10051		Enables the window trigger for long outer signals
TM_CHXWINENTER_SP	10042		Enables the window trigger for short inner signals
TM_CHXWINLEAVE_SP	10052		Enables the window trigger for short outer signals

So if you want to set up a four channel board to detect only a positive edge on channel0, you would have to setup the board like the following example. Both of the examples either for the TM\_CHANNEL and the TM\_CHOR triggermode do not include the necessary settings for the triggerlevels. These settings are detailed described in the following paragraphs.

```

SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_CHANNEL); // Enable channel trigger mode
SpcSetParam (hDrv, SPC_TRIGGERMODE0, TM_CHXPOS ); // Set triggermode of channel0 to positive edge trigger
for (i=1; i<16; i++) // Disable trigger detection of channel 1 to 15
    SpcSetParam (hDrv, SPC_TRIGGERMODE0+i, TM_CHXOFF );

```

If you want to set up a four channel board to detect a trigger event on either a positive edge on channel1 or a negative edge on channel3 you would have to set up your board as the following example shows.

```

SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_CHOR ); // Enable channel OR trigger mode
SpcSetParam (hDrv, SPC_TRIGGERMODE0, TM_CHXOFF); // Disable channel0 concerning trigger detection
SpcSetParam (hDrv, SPC_TRIGGERMODE1, TM_CHXPOS); // Set triggermode of channel1 to positive edge trigger
SpcSetParam (hDrv, SPC_TRIGGERMODE2, TM_CHXOFF); // Disable channel2 concerning trigger detection
SpcSetParam (hDrv, SPC_TRIGGERMODE3, TM_CHXNEG); // Set triggermode of channel3 to positive edge trigger
for (i=4; i<16; i++) // (( Disable trigger detection of channel 4 to 15
    SpcSetParam (hDrv, SPC_TRIGGERMODE0+i, TM_CHXOFF );

```

## Triggerlevel

All of the channel trigger modes listed above require at least one triggerlevel to be set (except TM\_CHXOFF of course). Some like the window trigger require even two levels (upper and lower level) to be set. Before explaining the different channel trigger modes, it is necessary to explain the board's series specific range of triggerlevels.

After the data has been sampled, the upper N data bits are compared with the N bits of the trigger levels. The amount of bits, the trigger levels are represented with depends on the board's series. The following table shows the level registers and the possible values they can be set to for your specific board.

Register	Value	Direction	Description	Range
SPC_HIGHLEVEL0	42000	read/write	Defines the upper level (triggerlevel) for channel 0	-8000 to +8000
SPC_HIGHLEVEL1	42001	read/write	Defines the upper level (triggerlevel) for channel 1	-8000 to +8000
...	...	...	...	...
SPC_HIGHLEVEL14	42014	read/write	Defines the upper level (triggerlevel) for channel 14	-8000 to +8000
SPC_HIGHLEVEL15	42015	read/write	Defines the upper level (triggerlevel) for channel 15	-8000 to +8000
SPC_LOWLEVEL0	42100	read/write	Defines the lower level (triggerlevel) for channel 0	-8000 to +8000
SPC_LOWLEVEL1	42101	read/write	Defines the lower level (triggerlevel) for channel 1	-8000 to +8000
...	...	...	...	...

Register	Value	Direction	Description	Range
SPC_LOWLEVEL14	42114	read/write	Defines the lower level (triggerlevel) for channel 14	-8000 to +8000
SPC_LOWLEVEL15	42115	read/write	Defines the lower level (triggerlevel) for channel 15	-8000 to +8000

In the above table the values for the triggerlevels represent the digital values for the corresponding data width N of the triggerlevels. If for example the triggerlevels are represented by 8 bit, the bipolar range would be -128 ... 127. To achieve symmetric triggerlevels the most negative value is not used and the so resulting range would be -127 ... +127.

As the triggerlevels are compared to the digitized data, the triggerlevels depend on the channels input range. For every input range available to your board there is a corresponding range of triggerlevels. On the different input ranges the possible stepsize for the triggerlevels differs as well as the maximum and minimum values. The following table, gives you the absolute triggerlevels for your specific board's series.

Triggerlevel	Input ranges							
	±50 mV	±100 mV	±250 mV	±500 mV	±1 V	±2 V	±5 V	±10 V
8000	+50.000 mV	+100.000 mV	+250.000 mV	+500.000 mV	+1.000 V	+2.000 V	+5.000 V	+10.000 V
7999	+49.997 mV	+99.987 mV	+249.969 mV	+499.938 mV	+0.999 V	+1.999 V	+4.999 V	+9.999 V
...	...	...	...	...	...	...	...	...
4000	+25.000 mV	+50.000 mV	+125.000 mV	+250.000 mV	+0.500 V	+1.000 V	+2.500 V	+5.000 V
...	...	...	...	...	...	...	...	...
2	+0.013 mV	+0.025 mV	+0.062 mV	+0.125 mV	+0.250 mV	+0.500 mV	+1.000 mV	+2.500 mV
1	+0.006 mV	+0.013 mV	+0.031 mV	+0.063 mV	+0.125 mV	+0.250 mV	+0.500 mV	+1.250 mV
0	0.000 mV	0.000 mV	0.000 mV	0.000 mV	0.000 mV	0.000 mV	0.000 mV	0.000 mV
-1	-0.006 mV	-0.013 mV	-0.031 mV	-0.063 mV	-0.125 mV	-0.250 mV	-0.500 mV	-1.250 mV
-2	-0.013 mV	-0.025 mV	-0.062 mV	-0.125 mV	-0.250 mV	-0.500 mV	-1.000 mV	-2.500 mV
...	...	...	...	...	...	...	...	...
-4000	-25.000 mV	-50.000 mV	-125.000 mV	-250.000 mV	-0.500 V	-1.000 V	-2.500 V	-5.000 V
...	...	...	...	...	...	...	...	...
-7999	-49.997 mV	-99.987 mV	-249.969 mV	-499.938 mV	-0.999 V	-1.999 V	-4.999 V	-9.999 V
-8000	-50.000 mV	-100.000 mV	-250.000 mV	-500.000 mV	-1.000 V	-2.000 V	-5.000 V	-10.000 V
Stepsize	6.25 µV	12.5 µV	31.3 µV	62.5 µV	125.0 µV	250.0 µV	500 µV	1.25 mV

The following example shows, how to set up a one channel board to trigger on channel0 rising edge. It is assumed, that the input range of channel0 is set to the ±2 V range. The decimal value for SPC\_HIGHLEVEL0 corresponds then with 10.0 mV, which is the resulting triggerlevel.

```
SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_CHANNEL); // Enable channel trigger mode
SpcSetParam (hDrv, SPC_TRIGGERMODE0, TM_CHXPOS); // Enable channel trigger mode
SpcSetParam (hDrv, SPC_HIGHLEVEL0, 40); // Sets triggerlevel to 10.0 mV
```

### 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 runtime. The register is shown in the following table:

Register	Value	Direction	Description
SPC_READTRIGLVCOUNT	2500	r	Contains the number of different possible trigger levels.

In case of a board that uses 8 bits for trigger detection the returned value would be 255, 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.

$$\text{Trigger step width} = \frac{\text{Input Range}_{\max} - \text{Input Range}_{\min}}{\text{Number of trigger levels} + 1}$$

To give you an example on how to use this formular we assume, that the ±1.0 V input range is selected and the board uses 8 bits for trigger detection.

$$\text{Trigger step width} = \frac{+1000 \text{ mV} - (-1000 \text{ mV})}{255 + 1}$$

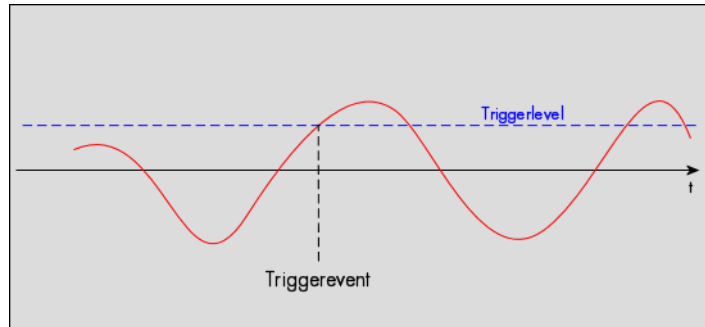
The result would be 7.81 mV, which is the step width for your type of board within the actually chosen input range.

## Detailed description of the channel trigger modes

### Channel trigger on positive edge

The analog input is continuously sampled with the selected sample rate. If the programmed triggerlevel is crossed by the channel's signal from lower values to higher values (rising edge) then the triggerevent will be detected.

These edge triggered channel trigger modes correspond to the trigger possibilities of usual oscilloscopes.

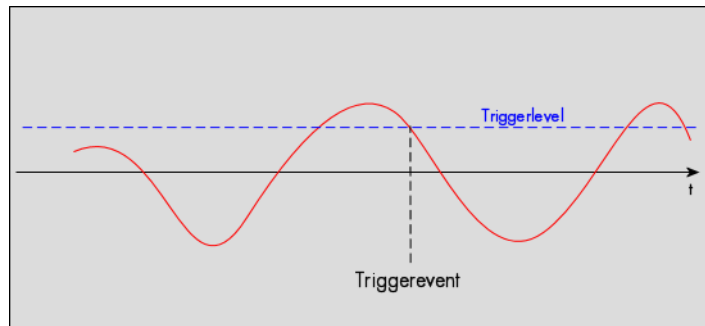


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXPOS	10000
SPC_HIGHLEVEL0	42000	read/write	Set it to the desired triggerlevel relatively to the channel's input range.	board dependant

### Channel trigger on negative edge

The analog input is continuously sampled with the selected sample rate. If the programmed triggerlevel is crossed by the channel's signal from higher values to lower values (falling edge) then the triggerevent will be detected.

These edge triggered channel trigger modes correspond to the trigger possibilities of usual oscilloscopes.

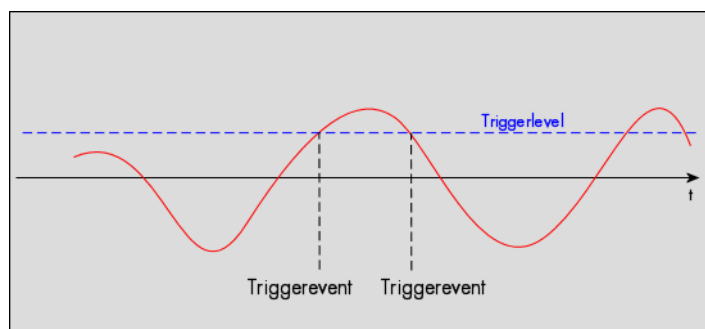


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXNEG	10010
SPC_HIGHLEVEL0	42000	read/write	Set it to the desired triggerlevel relatively to the channel's input range.	board dependant

### Channel trigger on positive and negative edge

The analog input is continuously sampled with the selected sample rate. If the programmed triggerlevel is crossed by the channel's signal (either rising or falling edge) the triggerevent will be detected.

These edge triggered channel trigger modes correspond to the trigger possibilities of usual oscilloscopes.

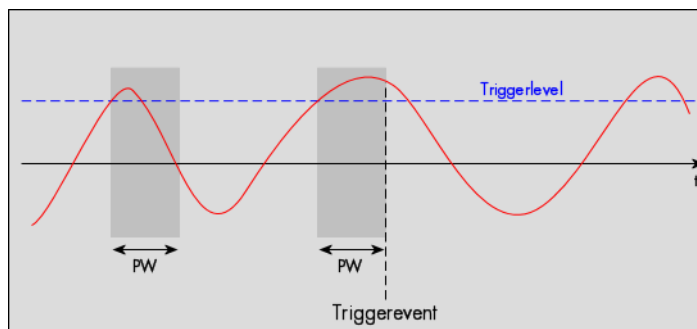


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXBOTH	10030
SPC_HIGHLEVEL0	42000	read/write	Set it to the desired triggerlevel relatively to the channel's input range.	board dependant

### Channel pulsewidth trigger for long positive pulses

The analog input is continuously sampled with the selected sample rate. If the programmed triggerlevel is crossed by the channel's signal from lower to higher values (rising edge) the pulsewidth counter is started. If the signal crosses the triggerlevel again in the opposite direction within the programmed pulsewidth time, no trigger will be detected. If the pulsewidth counter reaches the programmed amount of samples, without the signal crossing the triggerlevel in the opposite direction, the trigger event will be detected.

The pulsewidth trigger modes for long pulses can be used to prevent the board from triggering on wrong (short) edges in noisy signals.

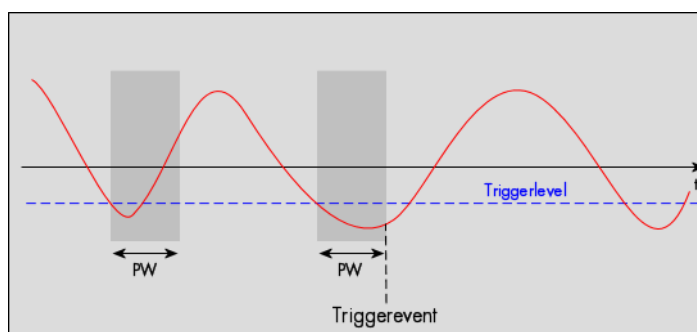


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXPOS_LP	10001
SPC_HIGHLEVEL0	42000	read/write	Set it to the desired triggerlevel relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255

### Channel pulsewidth trigger for long negative pulses

The analog input is continuously sampled with the selected sample rate. If the programmed triggerlevel is crossed by the channel's signal from higher to lower values (falling edge) the pulsewidth counter is started. If the signal crosses the triggerlevel again in the opposite direction within the programmed pulsewidth time, no trigger will be detected. If the pulsewidth counter reaches the programmed amount of samples, without the signal crossing the triggerlevel in the opposite direction, the trigger event will be detected.

The pulsewidth trigger modes for long pulses can be used to prevent the board from triggering on wrong (short) edges in noisy signals.



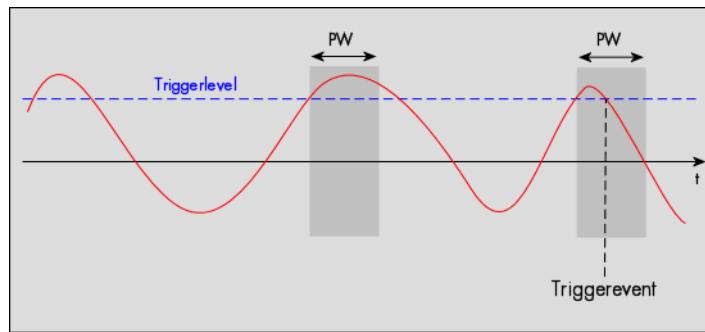
Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXNEG_LP	10011
SPC_HIGHLEVEL0	42000	read/write	Set it to the desired triggerlevel relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255



**Channel pulsewidth trigger for short positive pulses**

The analog input is continuously sampled with the selected sample rate. If the programmed triggerlevel is crossed by the channel's signal from lower to higher values (rising edge) the pulsewidth counter is started. If the pulsewidth counter reaches the programmed amount of samples, no trigger will be detected.

If the signal does cross the triggerlevel again within the the programmed pulsewidth time, a triggerevent will be detected.

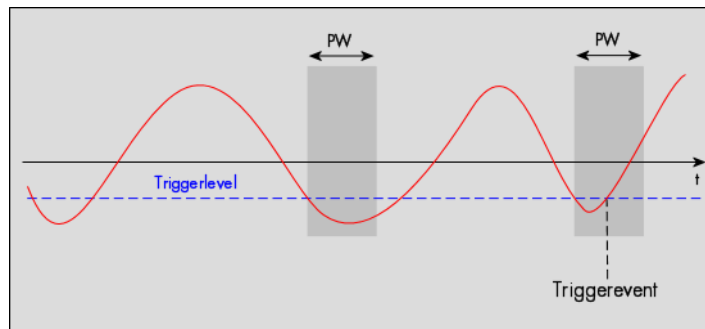


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXPOS_SP	10002
SPC_HIGHLEVEL0	42000	read/write	Set it to the desired triggerlevel relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255

**Channel pulsewidth trigger for short negative pulses**

The analog input is continuously sampled with the selected sample rate. If the programmed triggerlevel is crossed by the channel's signal from higher to lower values (falling edge) the pulsewidth counter is started. If the pulsewidth counter reaches the programmed amount of samples, no trigger will be detected.

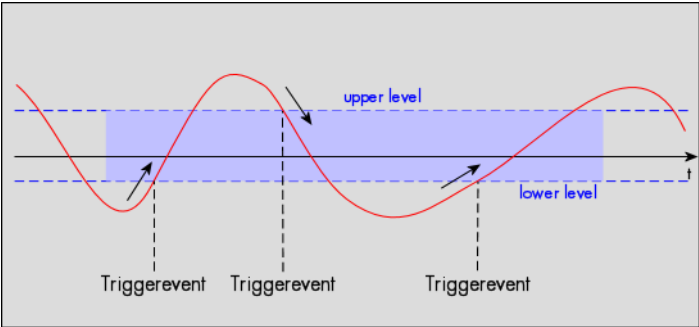
If the signal does cross the triggerlevel again within the the programmed pulsewidth time, a triggerevent will be detected.



Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXNEG_SP	10012
SPC_HIGHLEVEL0	42000	read/write	Set it to the desired triggerlevel relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255

**Channel window trigger for entering signals**

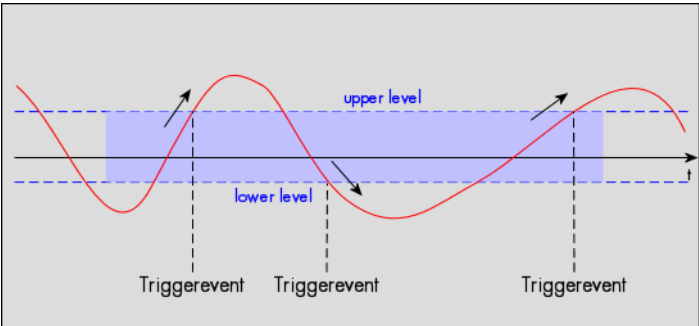
The analog input is continuously sampled with the selected sample rate. The upper and the lower level define a window. Every time the signal enters the the window from the outside, a trigger event will be detected.



Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXWINENTER	10040
SPC_HIGHLEVEL0	42000	read/write	Sets the window's upper level relatively to the channel's input range.	board dependant
SPC_LOWLEVEL0	42100	read/write	Sets the window's lower level relatively to the channel's input range.	board dependant

**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 the window from the inside, a trigger event will be detected.

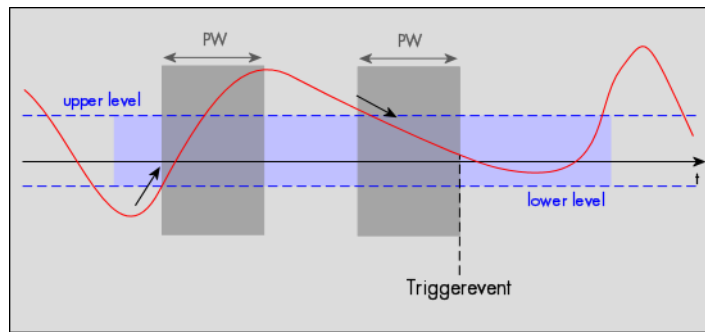


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXWINLEAVE	10050
SPC_HIGHLEVEL0	42000	read/write	Sets the window's upper level relatively to the channel's input range.	board dependant
SPC_LOWLEVEL0	42100	read/write	Sets the window's lower level relatively to the channel's input range.	board dependant

### Channel window trigger for long inner signals

The analog input is continuously sampled with the selected sample rate. The upper and the lower levels define a window. Every time the signal enters the window from the outside, the pulsewidth counter is started. If the signal leaves the window before the pulsewidth counter has stopped, no trigger will be detected.

If the pulsewidth counter stops and the signal is still inside the window, the trigger event will be detected.

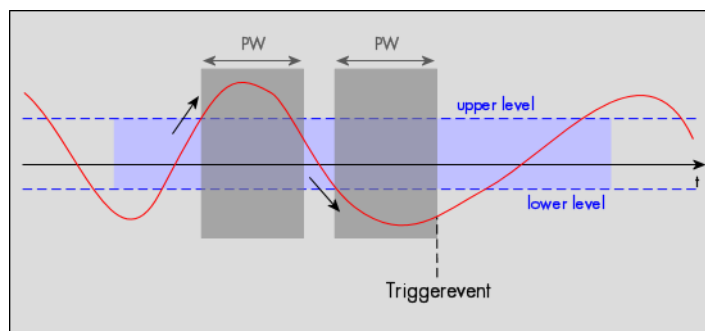


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXWINENTER_LP	10041
SPC_HIGHLEVEL0	42000	read/write	Sets the window's upper level relatively to the channel's input range.	board dependant
SPC_LOWLEVEL0	42100	read/write	Sets the window's lower level relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255

### Channel window trigger for long outer signals

The analog input is continuously sampled with the selected sample rate. The upper and the lower levels define a window. Every time the signal leaves the window from the inside, the pulsewidth counter is started. If the signal enters the window before the pulsewidth counter has stopped, no trigger will be detected.

If the pulsewidth counter stops and the signal is still outside the window, the trigger event will be detected.

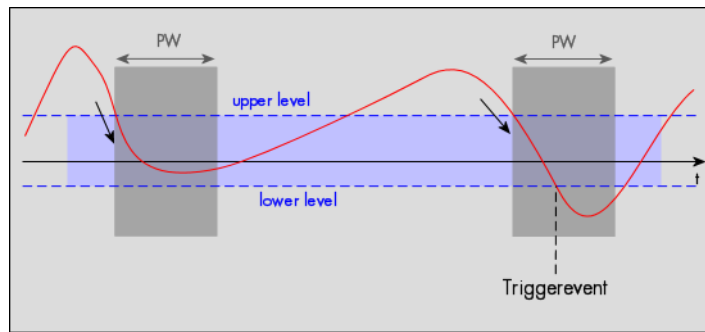


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXWINLEAVE_LP	10051
SPC_HIGHLEVEL0	42000	read/write	Sets the window's upper level relatively to the channel's input range.	board dependant
SPC_LOWLEVEL0	42100	read/write	Sets the window's lower level relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255

### Channel window trigger for short inner signals

The analog input is continuously sampled with the selected sample rate. The upper and the lower levels define a window. Every time the signal enters the window from the outside, the pulsewidth counter is started. If the pulsewidth counter stops and the signal is still inside the window, no trigger will be detected.

If the signal leaves the window before the pulsewidth counter has stopped, the trigger event will be detected.

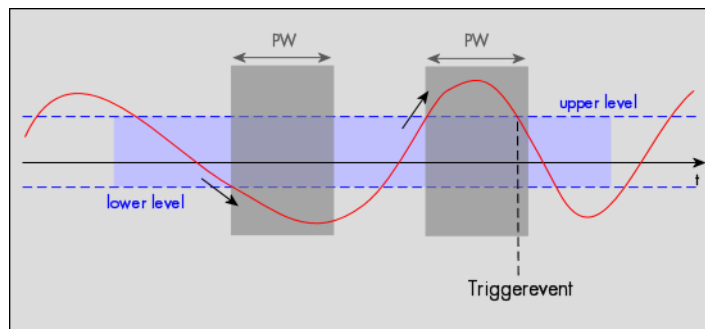


Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXWINENTER_SP	10042
SPC_HIGHLEVEL0	42000	read/write	Sets the window's upper level relatively to the channel's input range.	board dependant
SPC_LOWLEVEL0	42100	read/write	Sets the window's lower level relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255

### Channel window trigger for short outer signals

The analog input is continuously sampled with the selected sample rate. The upper and the lower levels define a window. Every time the signal leaves the window from the inside, the pulsewidth counter is started. If the pulsewidth counter stops and the signal is still outside the window, no trigger will be detected.

If the signal enters the window before the pulsewidth counter has stopped, the trigger event will be detected.



Register	Value	Direction	set to	Value
SPC_TRIGGERMODE	40000	read/write	TM_CHANNEL	20040
SPC_TRIGGERMODE0	40200	read/write	TM_CHXWINLEAVE_SP	10052
SPC_HIGHLEVEL0	42000	read/write	Sets the window's upper level relatively to the channel's input range.	board dependant
SPC_LOWLEVEL0	42100	read/write	Sets the window's lower level relatively to the channel's input range.	board dependant
SPC_PULSEWIDTH	44000	read/write	Set to the desired pulsewidth in samples.	2 to 255

## Option Timestamp

### 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 radio clock. This is done by a wide resetable counter that is incremented with every sample rate. With every detected trigger event the actual counter value is stored in a separate timestamp memory.

This function is designed as an enhancement to the Multiple Recording and the Gated Sampling mode but can also be used without these options. If Gated Sampling mode is used, then both the start and end of a recorded segment are timestamped.

The timestamp memory is designed as a FIFO buffer so that it can be read out even while the Spectrum board is recording data continuously to the PC in the FIFO mode. This extra memory is 64 K Timestamps in size.

Each recorded timestamp consists of the number of samples that has been counted since the last counter reset has been done. The actual time from the point since the last reset has been done so can easily be calculated by the formular besides.

$$t = \frac{\text{Timestamp}}{\text{Sample rate}}$$

If you want to know the time between two timestamps, you can simply calculate this by the formular besides.

$$\Delta t = \frac{\text{Timestamp}_{n+1} - \text{Timestamp}_n}{\text{Sample rate}}$$

### Limits

The timestamp counter is running with the sampling clock on the base card. Some card types (like 2030 and 3025) use an interlace mode to double the sampling speed. In this case the timestamp counter is only running with the non-interlaced sampling rate. Therefore the maximum counting frequency of the timestamp option is limited to 125 MS/s.

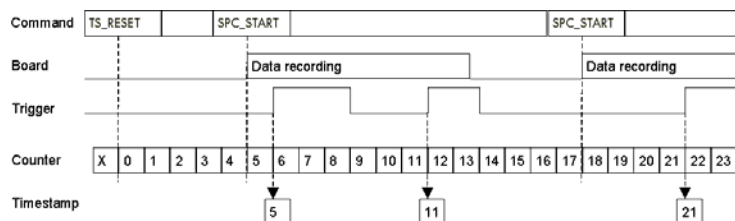
## Timestamp modes

### Standard mode

In standard mode the timestamp counter is set to zero once by writing the TS\_RESET command to the command register. After that command the counter counts continuously.

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.

The following table shows the valid values that can be written to the timestamp command register.



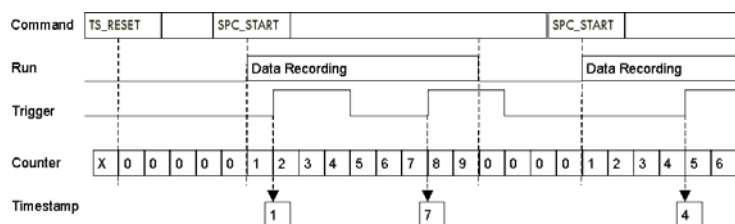
Register	Value	Direction	Description
SPC_TIMESTAMP_CMD	47000	w	Writes a command to the timestamp command register.
SPC_TIMESTAMP_CMD	47000	r	Reads out the actual timestamp mode.
TS_RESET	0		Resets the counter of the timestamp module to zero.
TS_MODE_DISABLE	10		Disables the timestamp module. No timestamps are recorded.
TS_MODE_STANDARD	12		Must be written to enable the Standard timestamp mode. The counter must be manually reset by writing the command TS_RESET to the command register. The timestamps values will be relative to this reset time.

### StartReset mode

In StartReset mode the timestamp counter is set to zero on every start of the board. After starting the board 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	w	Writes a command to the timestamp command register.
SPC_TIMESTAMP_CMD	47000	r	Reads out the actual timestamp mode.
TS_RESET	0		Resets the counter of the timestamp module to zero.

TS_MODE_DISABLE	10	Disables the timestamp module. No timestamps are recorded.
TS_MODE_STARTRESET	11	Must be written to enable the StartReset timestamp mode. The counter is reset on each start of the board. The timestamps values are relative to the board start.

### RefClock mode (optional)

The counter is split in a HIGH and a LOW part and an additional seconds signal, that affects both parts of the counter (TTL pulse with  $f = 1$  Hz) must be fed in externally.

The HIGH part counts the seconds that have elapsed since the last counter reset with the reset command TS\_RESET. The LOW part is reset to zero on every seconds signal and is clocked with the actual sample rate. The edge of the external secondssignal must be set separately as described below.

This mode allows the recording of an absolute time of a trigger event. This even allows the synchronization of data that has been recorded with different boards.

Register	Value	Direction	Description
SPC_TIMESTAMP_CMD	47000	w	Writes a command to the timestamp command register.
SPC_TIMESTAMP_CMD	47000	r	Reads out the actual timestamp mode.
TS_RESET	0		Resets the whole counter of the timestamp module to zero. Waits for synchronization to an external seconds signal. This may last up to 1 second. The lower part of the counter can be reset with the external fed in second signal. The edge of the reset signal can be programmed with the SPC_TIMESTAMP_RESETMODE register as shown in the table below.
TS_MODE_DISABLE	10		Disables the timestamp module. No timestamps are recorded.
TS_MODE_REFCLOCK	13		Must be written to enable the RefClock timestamp mode. The counter must be manually reset by writing the command TS_RESET to the command register. The counter is splitted into two parts. The upper part counts the seconds of an external reference clock. The lower part is reset on each second signal and counts the samples.

The edge of the external TTL seconds signal can be programmed by the following register either to detect the rising or falling edge.

Register	Value	Direction	Description
SPC_TIMESTAMP_RESETMODE	47050	r/w	Defines the active edge of the external fed in seconds signal to reset the lower part of the counter. The values written here do not affect the timestamp command TS_RESET.
TS_RESET_POS	10		The lower part of the counter will be reset on every rising edge of the external reset signal (seconds signal).
TS_RESET_NEG	20		The lower part of the counter will be reset on every falling edge of the external reset signal (seconds signal).

To get recordings in relation to each other it is important to know the absolute start time. This time can be easily read out by the following register. The time is given back in seconds since midnight (00:00:00), January 1, 1970, which is the standard 'time\_t' in C/C++.

Register	Value	Direction	Description
SPC_TIMESTAMP_STARTTIME	47030	r	Reads out the start time of the RefClock mode. Return value is the number of seconds since midnight (00:00:00), January 1, 1970, which is the standard 'time_t' in C/C++.

## Timestamp Status

The timestamp module has its own status register for the timestamp FIFO. You can easily read out the FIFO status with the help of the timestamp status register shown in the table below.

Register	Value	Direction	Description
SPC_TIMESTAMP_STATUS	47010	r	Reads the status of the timestamp FIFO.
TS_FIFO_EMPTY	0		The timestamp FIFO is still empty.
TS_FIFO_LESSHAF	1		There are values in the timestamp FIFO but less than half of the FIFO is filled.
TS_FIFO_MOREHALF	2		More than half of the FIFO is filled with timestamps.
TS_FIFO_OVERFLOW	3		The timestamp FIFO is full and possibly data has been lost.

## Reading out timestamp data

### Functions for accessing the data

There are two possibilities to access the timestamps that have been stored in the timestamp FIFO.

#### Reading out a single timestamp

You can read out one 32 bit value from the timestamp FIFO by using the register shown in the table below.

Register	Value	Direction	Description
SPC_TIMESTAMP_FIFO	47040	r	Get one 32 bit value from the timestamp FIFO. If the FIFO is empty a zero will be returned.

**Because accessing the timestamp with this function will be done with single accesses, getting the value(s) this way is much slower than using the SpcGetData function as described below.**



**Using this function will not give you back a whole timestamp, as the timestamp values are wider than 32 bit. Please also refer to the section on the timestamp data format below.**



### **Reading out all the timestamps with SpcGetData**

When using the function SpcGetData the data stored in the timestamp FIFO will be read out in one block by the driver. The usage of the function SpcGetData is described in the relating section earlier in this manual. The following list does only show the different parameters in a very short way:

#### **SpcGetData (nr, ch, start, len, data)**

- nr: Number of the board (Windows). Linux users please refer to the Driver section for differences using linux.
- ch: Channel to be read out. Must be set to CH\_TIMESTAMP (9999) to access timestamp FIFO.
- start: [Windows only:] Differing from the standard use, this parameter gives back the number of actually read timestamps and therefore needs to be a pointer. Please refer to the example at the end of this chapter. Under linux please use the SPC\_TIMESTAMP\_COUNT register instead and program the „start“ parameter to zero.
- len: Number of timestamps that fit in the data buffer and so defines the number of timestamps to be read out.
- data: Huge buffer for the read out timestamps, that must have at least enough space for 8\*len bytes.

It might be that you try to read out more timestamps than there actually are in the timestamp FIFO because you don't know how many trigger events have been detected. Please make use of the value given back by the parameter start to get to know what parts of your buffer contain valid timestamps.

Register	Value	Direction	Description
SPC_TIMESTAMP_COUNT	47020	r	Return the number of timestamps that have been read by the prior SpcGetData call. Needs only to be used under Linux.

### **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 sample 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. The delay is depending on the actual sample rate, the number of activated channels and the used trigger mode. This delay can be ignored, as it will be identically for all recordings with the same setup.

Timestamp Mode	Recording Mode	1 <sup>st</sup> 4 bytes	2 <sup>nd</sup> 4 bytes	3 <sup>rd</sup> 4 bytes	4 <sup>th</sup> 4 bytes	5 <sup>th</sup> 4 bytes	6 <sup>th</sup> 4 bytes
Standard/StartReset	Normal / Multiple Recording	Trigger 0 LOW part	Trigger 0 HIGH part	Trigger 1 LOW part	Trigger 1 HIGH part	Trigger 2 LOW part	Trigger 2 HIGH part
Standard/StartReset	Gated Sampling	Gate Start 0 LOW part	Gate Start 0 HIGH part	Gate End 0 LOW part	Gate End 0 HIGH part	Gate Start 1 LOW part	Gate Start 1 HIGH part
RefClock	Normal / Multiple Recording	Trigger 0 Counter value	Trigger 0 Seconds	Trigger 1 Counter value	Trigger 1 Seconds	Trigger 2 Counter value	Trigger 2 Seconds
RefClock	Gated Sampling	Gate Start 0 Counter value	Gate Start 0 Seconds	Gate End 0 Counter value	Gate End 0 Seconds	Gate Start 1 Counter value	Gate Start 1 Seconds

## Example programs

### Standard acquisition mode

```
// ----- Allocate memory for the timestamp data buffer -----
plTimeStamps = (ptr32) malloc (MAX_TIMESTAMPS * 8);

// ----- Reset the board and flush the FIFO -----
SpcSetParam (hDrv, SPC_COMMAND,          SPC_RESET);

// ----- Setup and start timestamp module -----
SpcSetParam (hDrv, SPC_TIMESTAMP_CMD, TS_MODE_STANDARD); // Standard mode set
SpcSetParam (hDrv, SPC_TIMESTAMP_CMD, TS_RESET);          // Counter is set to Zero

// ----- Start the board 4 times to generate 4 timestamps -----
for (i=0; i<4; i++)
{
    SpcSetParam (hDrv, SPC_COMMAND,          SPC_START); // Start recording
    do
    {
        SpcGetParam (hDrv, SPC_STATUS, &lStatus); // Wait for Status Ready
    }
    while (lStatus != SPC_READY);
}

// ----- Read out and display the timestamps -----
SpcGetData (hDrv, CH_TIMESTAMP, (int32) &lCount, MAX_TIMESTAMPS, (dataptr) plTimeStamps);
for (i=0; i<lCount; i++)
    printf ("Timestamp: %d\tHIGH: %08lx\tLOW: %08lx\n", i, plTimeStamps[2*i+1], plTimeStamps[2*i]);

// ----- Free the allocated memory for the timestamp data buffer -----
free (plTimeStamps);
}
```

### Acquisition with Multiple Recording

```
// ----- Reset the board and flush the FIFO -----
SpcSetParam (hDrv, SPC_COMMAND,          SPC_RESET);

// ----- Simple setup for recording -----
SpcSetParam (hDrv, SPC_CHENABLE, 1); // 1 channel for recording
SpcSetParam (hDrv, SPC_SAMPLERATE, 1000000); // Samplerate 1 MHz.
SpcSetParam (hDrv, SPC_TRIGGERMODE, TM_TTLPOS); // External positive Edge
SpcSetParam (hDrv, SPC_MULTI, 1); // Enable Multiple Recording
SpcSetParam (hDrv, SPC_MEMSIZE, 8192); // 8k Memsize
SpcSetParam (hDrv, SPC_POSTTRIGGER, 1024); // Each segment 1k = 8 segments
SpcSetParam (hDrv, SPC_MULTI, 1); // Enable Multiple Recording

// ----- Setup and start timestamp module -----
SpcSetParam (hDrv, SPC_TIMESTAMP_CMD, TS_MODE_STANDARD); // Standard Timestamp mode set
SpcSetParam (hDrv, SPC_TIMESTAMP_CMD, TS_RESET);          // Counter is set to Zero

// ----- Start the board -----
SpcSetParam (hDrv, SPC_COMMAND,          SPC_START); // Start recording
do
{
    SpcGetParam (hDrv, SPC_STATUS, &lStatus); // Wait for Status Ready
}
while (lStatus != SPC_READY);

// ----- Read out the timestamps -----
SpcGetData (hDrv, CH_TIMESTAMP, (int32) &lCount, 8, (dataptr) plTimeStamps);

// ----- display the timestamps (There should be 8 stamps, 1 for each segment) -----
for (i=0; i<lCount; i++)
    printf ("Segment: %d Counter: %08lx %08lx\n", i, plTimeStamps[2*i+1], plTimeStamps[2*i]);
```



## Option Extra I/O

### Digital I/Os

With this simple-to-use enhancement it is possible to control a wide range of external instruments or other equipment. Therefore you have several digital I/Os and the 4 analog outputs available. All extra I/O lines are completely independent from the board's function, data direction or sample rate and directly controlled by software (asynchronous I/Os).

The extra I/O option is useful if an external amplifier should be controlled, any kind of signal source must be programmed, an antenna must be adjusted, a status information from external machine has to be obtained or different test signals have to be routed to the board.

**It is not possible to use this option together with the star hub or timestamp option, because there is just space for one piggyback module on the on-board expansion slot.**



### Channel direction

#### Option -XMF (external connector)

The additional inputs and outputs are mounted on an extra bracket.

The direction of the 24 available digital lines can be programmed for every group of eight lines. The table below shows the direction register and the possible values. To combine the values simply OR them bitwise.

Register	Value	Direction	Description
SPC_XIO_DIRECTION	47100	r/w	Defines bitwise the direction of the digital I/O lines. The values can be combined by a bitwise OR.
XD_CH0_INPUT	0		Sets the direction of channel 0 (bit D7...D0) to input.
XD_CH1_INPUT	0		Sets the direction of channel 1 (bit D15...D8) to input.
XD_CH2_INPUT	0		Sets the direction of channel 2 (bit D23...D16) to input.
XD_CH0_OUTPUT	1		Sets the direction of channel 0 (bit D7...D0) to output.
XD_CH1_OUTPUT	2		Sets the direction of channel 1 (bit D15...D8) to output.
XD_CH2_OUTPUT	4		Sets the direction of channel 2 (bit D23...D16) to output.

#### Option -XIO (internal connector)

The additional inputs and outputs are available through an internal connector directly on the extra I/O piggyback module.

The direction of the 16 available digital lines can be programmed for every group of eight lines. The table below shows the direction register and the possible values. To combine the values so simply have to OR them bitwise.

Register	Value	Direction	Description
SPC_XIO_DIRECTION	47100	r/w	Defines bitwise the direction of the digital I/O lines. The values can be combined by a bitwise OR.
XD_CH0_INPUT	0		Sets the direction of channel 0 (bit D7...D0) to input.
XD_CH1_INPUT	0		Sets the direction of channel 1 (bit D15...D8) to input.
XD_CH0_OUTPUT	1		Sets the direction of channel 0 (bit D7...D0) to output.
XD_CH1_OUTPUT	2		Sets the direction of channel 1 (bit D15...D8) to output.

### Transfer Data

The outputs can be written or read by a single 32 bit register. If the register is read, the actual pin data will be taken. Therefore reading the data of outputs gives back the generated pattern. The single bits of the digital I/O lines correspond with the bitnumber of the 32 bit register. Values written to the most significant byte will be ignored.

Register	Value	Direction	Description
SPC_XIO_DIGITALIO	47110	r	Reads the data directly from the pins of all digital I/O lines either if they are declared as inputs or outputs.
SPC_XIO_DIGITALIO	47110	w	Writes the data to all digital I/O lines that are declared as outputs. Bytes that are declared as inputs will ignore the written data.

## Analog Outputs

In addition to the digital I/Os there are four analog outputs available. These outputs are directly programmed with the voltage values in mV. As the analog outputs are driven by a 12 bit DAC, the output voltage can be set in a stepsize of 5 mV. The table below shows the registers, you must write the desired levels too. If you read these outputs, the actual output level is given back from an internal software register.

Register	Value	Direction	Description	Offset range
SPC_XIO_ANALOGOUT0	47120	r/w	Defines the output value for the analog output A0.	± 10000 mV in steps of 5 mV
SPC_XIO_ANALOGOUT1	47121	r/w	Defines the output value for the analog output A1.	± 10000 mV in steps of 5 mV
SPC_XIO_ANALOGOUT2	47122	r/w	Defines the output value for the analog output A2.	± 10000 mV in steps of 5 mV
SPC_XIO_ANALOGOUT3	47123	r/w	Defines the output value for the analog output A3.	± 10000 mV in steps of 5 mV

After programming the levels of all analog outputs by the registers above, you have to update the analog outputs. This is done by the register shown in the table below. To update all of the outputs all you need to do is write a "1" to the dedicated register.

Register	Value	Direction	Description
SPC_XIO_WRITEDACS	47130	w	All the analog outputs are simultaneously updated by the programmed levels if a "1" is written.

## Programming example

The following example shows how to use either the digital I/O#s and the analog outputs.

```
// ----- output 8 bit on D7 to D0 and read 8 bit on D15 to D8 -----
SpcSetParam (hDrv, SPC_XIO_DIRECTION, XD_CH0_OUTPUT | XD_CH1_INPUT); // set directions of digital I/O transfer

SpcSetParam (hDrv, SPC_XIO_DIGITALIO, 0x00005A); // write data to D7-D0
SpcGetParam (hDrv, SPC_XIO_DIGITALIO, &lData); // read data and write values to lData

// ----- write some values to the analog channels. -----
SpcSetParam (hDrv, SPC_XIO_ANALOGOUT0, -2000); // -2000 mV = -2.0 V
SpcSetParam (hDrv, SPC_XIO_ANALOGOUT1, 0); // 0 mV = 0.0 V
SpcSetParam (hDrv, SPC_XIO_ANALOGOUT2, +3500); // 3500 mV = 3.5 V
SpcSetParam (hDrv, SPC_XIO_ANALOGOUT3, +10000); // 10000 mV = 10.0 V
SpcSetParam (hDrv, SPC_XIO_WRITEDACS, 1); // Write data simultaneously to DAC
```

## Synchronization (Option)

This option allows the connection of multiple boards to generate a multi-channel system. It is possible to synchronize multiple Spectrum boards of the same type as well as different board types. Therefore the synchronized boards must be linked concerning the board's system clock and the trigger signals.

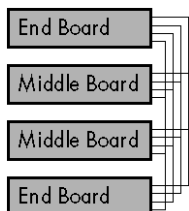
If no synchronization is desired for a certain board you can exclude it by setting the register shown in the following table. This must be done separately for every board that should not work synchronized.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_NOSYNC	120		Disables the synchronization globally.

## The different synchronization options

### Synchronization with option cascading

With the option cascading up to four Spectrum boards can be synchronized. All boards are connected with one synchronization cable on their sync-connectors (for details please refer to the chapter about installing the hardware).



As the synchronization lines are organized as a bus topology, there is a need for termination at both ends of the bus. This is done in factory for the both end-boards. The maximum possible two middle-boards have no termination on board.

When synchronizing multiple boards, one is set to be the clock master for all the connected boards. All the other boards are working as clock slaves. It's also possible to temporarily disable boards from the synchronization.

The same board or another one of the connected boards can be defined as a trigger master for all boards. All trigger modes of the trigger master board can be used. It is also possible to synchronize the connected boards only for the samplerate and not for trigger. This can be useful if one generator board is continuously generating a testpattern, while the connected acquisition board is triggering for test results or error conditions of the device under test.

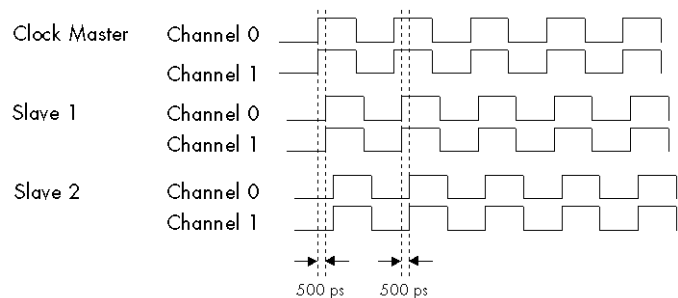
**For the fact that the termination is set in factory the order of the synchronized boards cannot be changed by the user. Please refer to the boards type plate for details on the board's termination. End boards are marked with the option „cs-end“ while middle boards are marked with the option „cs-mid“**



When the boards are synchronized by the option cascading there will be a delay of about 500 ps between two adjacent boards.

The figure on the right shows the clocks of three cascaded boards with two channels each, where one end-board is defined as a clock master. Slave 1 is therefore a middle-board and Slave 2 is the other end-board. The resulting delay between data of the two end-boards is therefore about 1 ns.

Please keep in mind that the delay between the channels of two boards is depending on which board is actually set up as the clock master and what boards are directly adjacent to the master.



### Synchronization with option starhub

With the option starhub up to 16 Spectrum boards can be synchronized. All boards are connected with a separate synchronization cable from their sync-connectors to the starhub module, which is a piggy-back module on one Spectrum board (for details please refer to the chapter about installing the hardware).

When synchronizing multiple boards, one is set to be the clock master for all the connected boards. All the other boards are working as clock slaves. It's also possible to temporarily disable the synchronization of one board. This board then runs individually while the other boards still are synchronized.

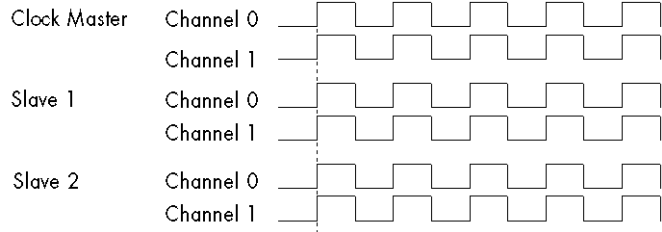
The same board or another one of the connected boards can be defined as a trigger master for all boards. All trigger modes of the board defined as the trigger master can be used. It is also possible to synchronize the connected boards only for the samplerate and not for trigger. This can be useful, if one generator board is continuously generating a testpattern, while the connected acquisition board is triggering for test results or error conditions of the device under test.

Additionally you can even define more than one board as a trigger master. The trigger events of all boards are combined by a logical OR, so that the first board that detects a trigger will start the boards. This OR connection is available starting with starhub hardware version V4.



When the boards are synchronized by the option starhub there will be no delay between the connected boards. This is achieved as all boards, including the one the starhub module is mounted on, are connected to the starhub with cables of the same length.

The figure on the right shows the clock of three boards with two channels each that are synchronized by starhub.



## The setup order for the different synchronization options



**If you setup the boards for the use with synchronization it is important to keep the order within the software commands as mentioned below to get the boards working correctly.**

Depending on if you use the board either in standard or in FIFO mode there are slightly different orders in the setup for the synchronization option. The following steps are showing the setups either for standard or FIFO mode.

### Setup Order for use with standard (non FIFO) mode and equally clocked boards

#### (1) Set up the board parameters

Set all parameters like for example sample rate, memsize and trigger modes for all the synchronized boards, except the dedicated registers for the synchronization itself that are shown in the tables below.

All boards must be set to the same settings for the entire clocking registers (see the according chapter for sample rate generation), for the trigger mode and memory and should be set to the same postcounter size to get the same pretrigger sizes as well.



**If you use acquisition boards with different pretrigger sizes, please keep in mind that after starting the board the pretrigger memory of all boards will be recorded first, before the boards trigger detection is armed. Take care to prevent boards with a long pretrigger setup time from hangup by adequately checking the board's status. Long setup times are needed if either you use a huge pretrigger size and/or a slow sample rate.**

If you don't care it might happen that boards with a small pretrigger are armed first and detect a trigger event, while one or more boards with a huge pretrigger are still not armed. This might lead to an endless waiting-state on these boards, which should be avoided.

Example of board setup for three boards

```
// ----- Set the Handles to fit for Windows driver -----
hDrv[0] = 0;
hDrv[1] = 1;
hDrv[2] = 2;

// (1) ----- Setup all boards, shortened here !!!-----
for (i = 0; i < 3; i++)
{
    SpcSetParam (hDrv[i], SPC_MEMSIZE,      1024);           // memory in samples per channel
    SpcSetParam (hDrv[i], SPC_POSTTRIGGER, 512);           // posttrigger in samples
    // ...
    SpcSetParam (hDrv[i], SPC_SAMPLERATE,  10000000);       // set sample rate to all boards
    SpcSetParam (hDrv[i], SPC_TRIGGERMODE, TM_SOFTWARE);    // set trigger mode to all boards
}
```

#### (2) Let the master calculate it's clocking

To obtain proper clock initialization when doing the first start it is necessary to let the clock master do all clock related calculations prior to setting all the synchronization configuration for the slave boards.

Example of board #0 set as clock master and forced to do the appropriate clock calculation

```
SpcSetParam (hDrv[0], SPC_COMMAND,      SPC_SYNCCALCMaster); // Calculate clock settings on master
```

#### (3) Write Data to on-board memory (output boards only)

If one or more of the synchronized boards are used for generating data (arbitrary waveform generator boards or digital I/O boards with one or more channels set to output direction) you have to transfer the data to the board's on-board memory before starting the synchronization. Please refer to the related chapter for the standard mode in this manual. If none of your synchronized boards is used for generation purposes you can ignore this step.

Example for data writing

```
SpcSetData (hDrv[0], 0, 0, 1024, pData[0]);
SpcSetData (hDrv[1], 0, 0, 1024, pData[1]);
SpcSetData (hDrv[2], 0, 0, 1024, pData[2]);
```

#### (4) Define the board(s) for trigger master

At least one board must be set as the trigger master to get synchronization running. Every one of the synchronized boards can be programmed for being the trigger master device.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_SYNCTRIGGERMASTER	101		Defines the according board as the triggermaster.

Example of board #2 set as trigger master

```
SpcSetParam (hDrv[2], SPC_COMMAND, SPC_SYNCTRIGGERMASTER); // Set board 2 to trigger master
```

#### (4a) Define synchronization OR trigger

If you use synchronization with the starhub option you can even set up more than one board as the trigger master. The boards will be combined by a logical OR and therefore the boards will be started if any of the trigger masters has detected a trigger event.

**The synchronization OR-trigger is not available when using the cascading option. It is also not available with starhub option prior to hardware version V4. See the initialization section of this manual to find out how to determint the hardware version of the starhub.**



If you set up the boards for the synchronization OR trigger all boards that are set as trigger master must be programmed to the same trigger-mode. If the boards are using different trigger modes this will result in a time shift between the boards. It is of course possible to set different edges or different trigger levels on the channels.

**It is only possible to use the synchronization OR trigger if the board carrying the starhub piggy-back module is one of the boards that is programmed as a trigger master.**



To find out what board is carrying the starhub piggy-back module you make use of the board's feature registers as described in the chapter about initialising the board.



Example of setting up three boards to be trigger master

```
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCTRIGGERMASTER); // Set board 0 to trigger master
SpcSetParam (hDrv[1], SPC_COMMAND, SPC_SYNCTRIGGERMASTER); // Set board 1 to trigger master
SpcSetParam (hDrv[2], SPC_COMMAND, SPC_SYNCTRIGGERMASTER); // Set board 2 to trigger master
```

#### (5) Define the remaining boards as trigger slaves

As you can set more than one board as the trigger master (starhub option only) you have to tell the driver additionally which of the boards are working as trigger slaves.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_SYNCTRIGGERSLAVE	111		Defines the according board as the trigger slave.

**Each of the synchronized boards must be set up either as a trigger master or as a trigger slave to get the synchronization option working correctly. Therefore it does not matter if you use the cascading or starhub option.**



It is assumed that only one of the three boards (board 2 in this case) is set up as trigger master, as described in (3)

```
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCTRIGGERSLAVE); // Setting all the other boards to
SpcSetParam (hDrv[1], SPC_COMMAND, SPC_SYNCTRIGGERSLAVE); // trigger slave is a must !
```

It sometimes might be necessary to exclude one or more boards from the synchronization trigger. An example for this solution is that one or more output boards are used for continuously generating test patterns, while one or more acquisition boards are triggering for test results or error conditions. Therefore it is possible to exclude a board from the triggerbus so that only a synchronization for clock is done and the according boards are just using the trigger events they have detected on their own.

Register	Value	Direction	Description
SPC_NOTRIGSYNC	200040	r/w	If activated the dedicated board will use its own trigger modes instead of the synchronization trigger.



**Even if a board is not using the synchronization trigger, it before must be set as a triggerslave with the SPC\_SYNCTRIGGERSLAVE command.**



After you have excluded one or more of the installed boards from the synchronization trigger it is possible to change the triggermodes of these boards. So only all the boards that should work synchronously must be set up for the same trigger modes to get the synchronization mode working correctly.

#### **(6) Define the board for clock master**

Using the synchronization option requires one board to be set up as the clock master for all the synchronized boards. It is not allowed to set more than one board to clock master.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_SYNCMAS	100		Defines the according board as the clock master for operating in standard (non FIFO) mode only.

Example: board number 0 is clock master

```
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCMAS); // Set board 0 to clock master
```

#### **(7) Define the remaining boards as clock slaves**

It is necessary to set all the remaining boards to clock slaves to obtain correct internal driver settings.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_SYNCSLAVE	110		Defines the according board as a clock slave for operating in standard (non FIFO) mode only.

Setting the remaining boards to clock slaves. Board number 0 is clock master in the example

```
SpcSetParam (hDrv[1], SPC_COMMAND, SPC_SYNCSLAVE); // Setting all the other boards to
SpcSetParam (hDrv[2], SPC_COMMAND, SPC_SYNCSLAVE); // clock slave is a must !
```

#### **(8) Arm the boards for synchronization**

Before you can start every single one of the synchronized boards on their own you have to arm all the synchronized boards before for the use with synchronization. The synchronization has to be started on the clock master board.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_SYNCSTART	130		Arms all boards for the use with synchronization.

Example of starting the synchronization. Board number 0 is clock master.

```
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCSTART);
```

#### **(9) Start all of the trigger slave boards**

After having armed the synchronized boards, you must start all of the boards that are defined as trigger slaves first.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_START	10		Starts the board with the current register settings.
SPC_STARTANDWAIT	11		Starts the board with the current register settings in the interrupt driven mode.

For details on how to start the board in the different modes in standard mode (non FIFO) please refer to the according chapter earlier in this manual.



**If using the interrupt driven mode SPC\_STARTANDWAIT it is necessary to start each board in it's own software thread. This is necessary because the function does not return until the board has stopped again. If not using different threads this will result in a program deadlock.**

Example of starting trigger slave boards. Board number 2 is trigger master.

```
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_START);
SpcSetParam (hDrv[1], SPC_COMMAND, SPC_START);
```

**(10) Start all of the trigger master boards**

After having armed the synchronized boards, you must start all of the boards, that are defined as trigger masters.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_START	10		Starts the board with the current register settings.
SPC_STARTANDWAIT	11		Starts the board with the current register settings in the interrupt driven mode.

For details on how to start the board in the different modes in standard mode (non FIFO) please refer to the according chapter earlier in this manual.

**If you use the synchronization OR with the starhub option it is important to start the board carrying the starhub piggy-back module as last. Otherwise the trigger masters that are started first might detect trigger events while other trigger masters haven't even been started. Be sure that the pretrigger area of all other trigger masters is filled at the moment when the pretrigger area of the star-hub board has been filled.**



To find out which board is carrying the starhub piggy-back module you make use of the board's feature registers as described in the chapter about programming the board.

Example of starting the trigger master board

```
SpcSetParam (hDrv[2], SPC_COMMAND, SPC_START);
```

**(11) Wait for the end of the measurement**

After having started the last board, you will have to wait until the measurement is done. Depending if you use the board in standard (non FIFO) mode interrupt driven or not, you can poll for the board's status. Please refer to the relating chapter in this manual. It is necessary to wait until each board returns the status SPC\_READY before proceeding.

Example for polling for three synchronized boards

```
for (i = 0; i < 3; i++)           // For all synchronized boards
do                               // The status is read out
{
    SpcGetParam (hDrv[i], SPC_STATUS, &lStatus);    // by polling for SPC_READY
}
while (lStatus != SPC_READY);
}
printf ("All boards have stopped");
```

**(12) Read data from the on-board memory (acquisition boards only)**

If one or more of the synchronized boards are used for recording data (transient recorder boards or digital I/O boards with one or more channels set to input direction) you have to read out the data from the board's on-board memory now. Please refer to the related chapter for the standard (non FIFO) mode in this manual. If none of your synchronized boards is used for recording purposes you can ignore this step.

Example for data reading

```
SpcGetData (hDrv[0], 0, 0, 1024, pData[0]);
SpcGetData (hDrv[1], 0, 0, 1024, pData[1]);
SpcGetData (hDrv[2], 0, 0, 1024, pData[2]);
```

**(13) Restarting the board for another synchronized run**

If you want to restart the synchronized boards with the same settings as before it is sufficient to repeat only the steps starting with (8). This assumes that on generation boards the output data is not changed as well.

If you want to change the output data of generation boards you'll have to restart the setup procedure starting with step (2).

If you even want to change any of the boards parameters you'll have to restart the setup procedure from the first step on.

## Setup synchronization for use with FIFO mode and equally clocked boards

Most of the steps are similar to the setup routine for standard synchronization mentioned before. In this passage only the differences between the two modes are shown. Please have a look at the passage before to see the complete setup procedure. The following steps differ from standard mode to FIFO mode. All steps that are not mentioned here are similar as described before.

### (2) Allocate the FIFO software buffers

If you use the board in FIFO mode additional memory in the PC RAM is needed for software FIFO buffers. For details please refer to the according chapter for the FIFO mode.

Example of FIFO buffer allocation:

```
for (i = 0; i < FIFO_BUFFERS; i++)
    for (b = 0; b < 3; b++)
    {
        pData[b][i] = (ptr16) GlobalAlloc (GMEM_FIXED, FIFO_BUFLen); // allocate memory
        SPCSetParam (b, SPC_FIFO_BUFADR0 + i, (int32) pData[b][i]); // send the address to the driver
    }
```

### (2a) Write first data for output boards

When using the synchronization FIFO mode with output boards this is the right position to fill the first software buffers with data. As you can read in the FIFO chapter, output boards need some data to be written to the software FIFO buffers before starting the board.

Example of calculating and writing output data to software FIFO buffers:

```
// ----- data calculation routine -----
int g_nPos = 0; // some global variables

void vCalcOutputData (ptr16 pData, int32 lBufsize) // function to calculate the
{ // output data. In this case
    int i; // a sine function is used.

    for (i = 0; i < (lBufsize/2); i++)
        pData[b][i] = (int16) (8191.0 * sin (2 * PI / 500000 * (g_nPos+i)));
    g_nPos += lBufsize/2;
}

// ----- main task -----
int main(int argc, char **argv)
{
    ...
    for (i = 0; i < MAX_BUF; i++) // fill the first buffers with data
        for (b = 0; b < 3; b++) // for all installed boards
            vCalcOutputData (pData[b][i], BUFSIZE);
    ...
}
```

### (6) Define the board for clock master

Using the synchronization option requires one board to be set up as the clock master for all the synchronized board. It is not allowed to set more than one board to clock master.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_SYNCMASTERFIFO	102		Defines the according board as the clock master for operating in FIFO mode only.

Example: board number 0 is clock master

```
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCMASTERFIFO); // Set board 0 to clock master
```

### (7) Define the remaining boards as clock slaves

It is necessary to set all the remaining boards to clock slaves to obtain correct internal driver settings.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_SYNCSLAVEFIFO	112		Defines the according board as a clock slave for operating in FIFO mode only.

Settings the remaining boards to clock slaves. Board number 0 is clock master in the example

```
SpcSetParam (hDrv[1], SPC_COMMAND, SPC_SYNCSLAVEFIFO); // Setting all the other boards to
SpcSetParam (hDrv[2], SPC_COMMAND, SPC_SYNCSLAVEFIFO); // clock slave is a must !
```



**(9) Start all of the trigger slave boards**

After having armed the synchronized boards, you must start all of the boards, that are defined as trigger slaves first. This is done with the FIFOSTART command.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_FIFOSTART	12		Starts the board with the current register settings in FIFO mode and waits for the first interrupt.

**Remember that the FIFO mode is allways interrupt driven. As a result the FIFOSTART function will not return until the first software buffer is transferred. For that reason it is absolutely necessary to start different threads for each board that runs synchronously in FIFO mode. If this is not done a deadlock will occur and the program will not start properly.**

**(10) Start all of the trigger master boards**

After having armed the synchronized boards, you must start all of the boards, that are defined as trigger masters.

Register	Value	Direction	Description
SPC_COMMAND	0	r/w	Command register of the board
SPC_FIFOSTART	12		Starts the board with the current register settings in FIFO mode and waits for the first interrupt.

This example shows how to set up three boards for synchronization in FIFO mode. Board 0 is clock master and board 2 is trigger master.

```
// (3) ----- trigger synchronization of trigger master board(s) -----
SpcSetParam (hDrv[2], SPC_COMMAND, SPC_SYNCMASTER);           // board 2 set as trigger master

// (4) ----- trigger synchronization of trigger slave boards -----
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCTRIGGERSLAVE);      // as trigger slaves
SpcSetParam (hDrv[1], SPC_COMMAND, SPC_SYNCTRIGGERSLAVE);      // as trigger slaves

// (5) ----- synchronization information for clock master board -----
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCMASTERFIFO);

// (6) ----- synchronization information for clock slave boards -----
SpcSetParam (hDrv[1], SPC_COMMAND, SPC_SYNCSLAVEFIFO);
SpcSetParam (hDrv[2], SPC_COMMAND, SPC_SYNCSLAVEFIFO);

// (7) ----- start the synchronization -----
SpcSetParam (hDrv[0], SPC_COMMAND, SPC_SYNCSTART);

// (8) ----- start the FIFO tasks. Trigger slaves are started first -----
CreateThread (NULL, 0, &dwFIFOTask, (void*) hDrv[0], 0, &dwThreadId[b]);
CreateThread (NULL, 0, &dwFIFOTask, (void*) hDrv[1], 0, &dwThreadId[b]);

// (9) ----- start the trigger master FIFO task -----
CreateThread (NULL, 0, &dwFIFOTask, (void*) hDrv[2], 0, &dwThreadId[hDrv[2]]);
```

It is assumed, that the created threads start in the same order as they are called from within the program. As described before, starting of the FIFO mode in synchronization has to be done in different threads to avoid a deadlock. A simple example for a FIFO thread can be found below.

Example of FIFO task. It simply starts the boards and counts the buffers that have been transferred:

```
unsigned long __stdcall dwFIFOTask (void* phDrv)
{
    int16  hDrv = (int16) phDrv;
    int32  lCmd = SPC_FIFOSTART;
    int16  nBufIdx = 0, nErr;
    int32  lTotalBuf;

    lTotalBuf = 0;
    do
    {
        nErr = SpcSetParam (hDrv, SPC_COMMAND, lCmd);           // wait for buffer
        lCmd = SPC_FIFOWAIT;                                     // here you can do
        printf ("Board %d  Buffer %d  total buffers: %d\n", nIdx, nBufIdx, lTotalBuf); // e.g. calculations
                                                    // just a printf here
        SpcSetParam (hDrv, SPC_COMMAND, SPC_FIFO_BUFREADY0 + nBufIdx); // release buffer

        nBufIdx++;
        lTotalBuf++;
        if (nBufIdx == FIFO_BUFFERS)
            nBufIdx = 0;
    }
    while (nErr == ERR_OK);

    return 0;
}
```

## Additions for synchronizing different boards

### General information

Spectrum boards with different speed grades, different number of channels or even just different clock settings for the same types of boards can be synchronized as well. To get the boards working together synchronously some extra setups have to be done, which are described in the following passages.

All clock rates of all synchronized boards are derived from the clock signal that is distributed via the sync bus. This clock is the sum samplerate of one module of the clock master board. Based on this speed the clock rates of the slave boards can be set. As these clock rates are divided from the sync clock, the board with the maximum sum sample rate should be set up as clock master.

### Calculating the clock dividers

The sum sample rate can easily be calculated by the formula on the right. The value for the sample rate of board N must contain the actual desired conversion rate for one channel of board N. Please refer to the dedicated chapter in the board's manual to get informed about the relation between the board model and the number of actually activated channels per module for the different channel setups.

$$\text{SumSampleRate}_N = \text{SampleRate}_N \cdot \text{ActChPerModule}_N$$

As mentioned above the board with the highest sum sample rate must be set up as the clock master. This maximum sum sample rate is used as the overall sync speed, which is distributed via the sync bus. If you have calculated the sync speed you can calculate the clock dividers for the different boards with the formula on the right.

$$\text{ClockDivider}_N = \frac{\text{SyncSpeed}}{\text{SampleRate}_N \cdot \text{ActChPerModule}_N}$$

The maximum possible channels per module for all Spectrum boards are given in the table below.

	20xx	x	30xx	x	31xx	x	40xx	x	46xx	x	47xx	x	60xx	x	61xx	x	70xx	x	72xx	x
xx0x																	7005	1		
xx1x			3010	1	3110	2					4710	8			6110	2	7010	1	7210	1
			3011	2	3111	4					4711	8	6011	2	6111	2	7011	2	7211	1
			3012	2	3112	4							6012	2						
			3013	2																
			3014	2																
			3015	1																
			3016	2																
xx2x	2020	2	3020	1	3120	2	4020	1	4620	2	4720	8					7020	1	7220	1
	2021	2	3021	2	3121	4	4021	2	4621	4	4721	8	6021	2			7021	2	7221	1
			3022	2	3122	4	4022	2	4622	4			6022	2						
			3023	2																
			3024	2																
			3025	1																
			3026	2																
			3027	1																
xx3x					3130	2	4030	1	4630	2	4730	8	6030	1						
	2031	2	3031	2	3131	4	4031	2	4631	4	4731	8	6031	1						
					3132	4	4032	2	4632	4										
	2033	2	3033	2									6033	2						
													6034	2						
xx4x									4640	2										
									4641	4										
									4642	4										
xx5x									4650	2										
									4651	4										
									4652	4										

### Setting up the clock divider

The clock divider can easily be set by the following register. Please keep in mind that the divider must be set for every synchronized board to have synchronization working correctly. For more details on the board's clocking modes please refer to the according chapter in this manual.

Register	Value	Direction	Description
SPC_CLOCKDIV	20040	r/w	Extra clock divider for synchronizing different boards.

Available divider values

1	2	4	8	10	16	20	40	50	80	100	200
400	500	800	1000	2000							

**The clock divider is also used by internal clock generation for all clock rates that are below 1 MS/s sum sample rate per module. If internal clock divider and extra clock divider are used together the resulting clock divider is one value of the above listed. The driver searches for the best matching divider. Read out the register after all sample rate registers are set to receive the resulting extra clock divider. For correct setting of the clock divider the sample rate and channel enable information must be set before the clock divider is programmed.**



Although this setup is looking very complicated at first glance, it is not really difficult to set up different boards to work synchronously with the same speed. To give you an idea on how to setup the boards the calculations are shown in the following two examples.

Each example contains of a simple setup of two synchronized boards. It is assumed that all of the available channels on the dedicated boards have been activated.

Example calculation with synchronous speed where slave clock is divided

<b>Board type</b>	<b>3122</b>	<b>3120</b>
Channels available	8 x 12 bit A/D	2 x 12 bit A/D
Desired sample rate	10 MS/s	10 MS/s
Enabled channels per module	4	2
Sum sample rate	<b>40 MS/s</b>	20 MS/s
	Therefore this board is set up to be the clockmaster.	
Sync speed	40 MS/s	40 MS/s
Clock divider	1	2
Divided sum clock	40 MS/s	20 MS/s
Enabled channels per module	4	2
Conversion speed	10 MS/s	10 MS/s

### Example calculation with synchronous speed where master clock is divided

<b>Board type</b>	<b>3025</b>	<b>3131</b>
Channels available	2 x 12 bit A/D	4 x 12 bit A/D
Desired sample rate	20 MS/s	20 MS/s
Enabled channels per module	1	2
Sum sample rate	20 MS/s	<b>40 MS/s</b>
	Therefore this board is set up to be the clockmaster.	
Sync speed	40 MS/s	40 MS/s
Clock divider	2	1
Divided sum clock	20 MS/s	40 MS/s
Enabled channels per module	1	2
Conversion speed	20 MS/s	20 MS/s

### Additions for equal boards with different sample rates

In addition to the possibility of synchronizing different types of boards to one synchronous sample rate it can be also useful in some cases to synchronize boards of the same type, with one working at a divided speed.

In this case you simply set up the fastest board as the clock master and set its clock divider to one. Now you can easily generate divided clock rates on the slave boards by setting their dividers to according values of the divider list.

**Please keep in mind that only the dedicated divider values mentioned in the list above can be used to derive the sample rates of the slave boards.**



The following example calculation is explaining that case by using two acquisition boards. One of the boards is running with only a hundredth of the other sample rate.

Example with equal boards but asynchronous speeds

Board type	3121	3121
Channels available	4 x 12 bit A/D	4 x 12 bit A/D
Desired sample rate	10 MS/s	
Enabled channels per module	4	4
Sum sample rate	<b>40 MS/s</b>	
This board is set up to be the clockmaster now.		
Sync speed	40 MS/s	40 MS/s
Clock divider (is set to)	1	100
Divided sum clock	40 MS/s	400 kS/s
Enabled channels per module	4	4
Conversion speed	10 MS/s	100 kS/s

## Resulting delays using different boards or speeds

### Delay in standard (non FIFO) modes

There is a fixed delay between the samples of the different boards depending on the type of board, the selected clock divider and the activated channels. This delay is fixed for data acquisition or generation with the same setup.



If you use generation boards in the single shot mode this delay will be compensated within the software driver automatically.

### Delay in FIFO mode

When the FIFO mode is used a delay is occurring between the data of the different boards. This delay is depending on the type of board, the selected clock divider and the activated channel. You can read out the actual resulting delay from every board with the following register.

Register	Value	Direction	Description
SPC_STARTDELAY	295110	r	Start delay in samples for FIFO synchronization only.

The resulting delay between the clock master board and the single clock slave boards can be easily calculated with the formular mentioned on the right.

$$\text{ResultingDelay} = \text{ClockMasterDelay} - \text{ClockSlaveDelay}_N$$

# 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.

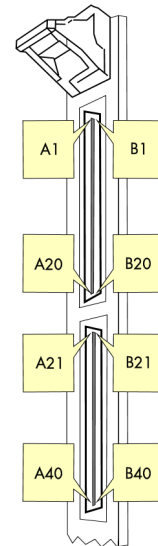
error name	value (hex)	value (dec.)	error description
ERR_OK	0h	0	Execution OK, no error.
ERR_INIT	1h	1	The board number is not in the range of 0 to 15. When initialisation is executed: the board number is yet initialised, the old definition will be used.
ERR_NR	2h	2	The board is not initialised yet. Use the function SpclnitPCIBoards first. If using ISA boards the function SpclnitBoard must be called first.
ERR_TYP	3h	3	Initialisation only: The type of board is unknown. This is a critical error. Please check whether the board is correctly plug 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 remap 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 reinstallation of the hardware driver. This error normally only occurs if someone copies the dll manually to the system directory.
ERR_HWRDRVVERSION	7h	7	The hardware needs a newer driver version to run properly. Please install the driver that was delivered together with the board.
ERR_ADDRANGE	8h	8	The address range is disabled (fatal error)
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_ABORT	20h	32	Abort of wait function. This return value just tells that the function has been aborted from another thread.
ERR_BOARDLOCKED	30h	48	Access to the driver already locked by another program. Stop the other program before starting this one. Only one program can access the driver at the time.
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	Channel sequence is not allowed.
ERR_READABORT	104h	260	Data read is not allowed after aborting the data acquisition.
ERR_NOACCESS	105h	261	Access to this register denied. No access for user allowed.
ERR_POWERDOWN	106h	262	Not allowed if powerdown mode is activated.
ERR_TIMEOUT	107h	263	A timeout occurred while waiting for an interrupt. Why this happens depends on the application. Please check whether the timeout value is programmed too small.
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 actual setup (e.g. Only channel 0 is accessible in interlace mode)
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 adjustment has reported an error. Please check the boards inputs.
ERR_NOPCI	200h	512	No PCI BIOS is found on the system.
ERR_PCIVERSION	201h	513	The PCI bus has the wrong version. SPECTRUM PCI boards require PCI revision 2.1 or higher.
ERR_PCINOBOARDS	202h	514	No SPECTRUM PCI boards found. If you have a PCI board in your system please check whether it is correctly plug into the slot connector and whether you have the latest driver version.
ERR_PCICHECKSUM	203h	515	The checksum of the board information has failed. This could be a critical hardware failure. Restart the system and check the connection of the board in the slot.
ERR_DMALOCKED	204h	516	DMA buffer not available now.
ERR_MEMALLOC	205h	517	Internal memory allocation failed. Please restart the system and be sure that there is enough free memory.
ERR_FIFOBUFOVERRUN	300h	768	Driver buffer overrun in FIFO mode. The hardware and the driver have been fast enough but the application software didn't manage to transfer the buffers in time.
ERR_FIFOWOVERRUN	301h	769	Hardware buffer overrun in FIFO mode. The hardware transfer and the driver has not been fast enough. Please check the system for bottlenecks and make sure that the driver thread has enough time to transfer data.
ERR_FIFOFINISHED	302h	770	FIFO transfer has been finished, programmed number of buffers has been transferred.
ERR_FIFOSETUP	309h	777	FIFO setup not possible, transfer rate too high (max 250 MB/s).
ERR_TIMESTAMP_SYNC	310h	784	Synchronisation to external timestamp reference clock failed. At initialisation is checked whether there is a clock edge present at the input.
ERR_STARHUB	320h	800	The autorouting function of the star-hub initialisation has failed. Please check whether all cables are mounted correctly.

### Pin assignment of the multipin connector

The 40 lead multipin connector is used for different options, like "Extra I/O" or the additional digital inputs (on analog acquisition boards only) or additional digital outputs (on analog generation boards only).

The connectors mentioned here are mounted on an extra bracket.

The pin assignment depends on which of the below mentioned options are installed.



### Extra I/O with external connector(Option -XMF)

B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17	B18	B19	B20
D0	GND	D1	GND	D2	GND	D3	GND	D4	GND	D5	GND	D6	GND	D7	GND	n.c.	n.c.	n.c.	n.c.

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
D8	GND	D9	GND	D10	GND	D11	GND	D12	GND	D13	GND	D14	GND	D15	GND	n.c.	n.c.	n.c.	n.c.

B21	B22	B23	B24	B25	B26	B27	B28	B29	B30	B31	B32	B33	B34	B35	B36	B37	B38	B39	B40
D16	GND	D17	GND	D18	GND	D19	GND	D20	GND	D21	GND	D22	GND	D23	GND	n.c.	n.c.	n.c.	n.c.

A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31	A32	A33	A34	A35	A36	A37	A38	A39	A40
A0	GND	GND	GND	A1	GND	GND	GND	A2	GND	GND	GND	A3	GND	GND	GND	n.c.	n.c.	n.c.	n.c.

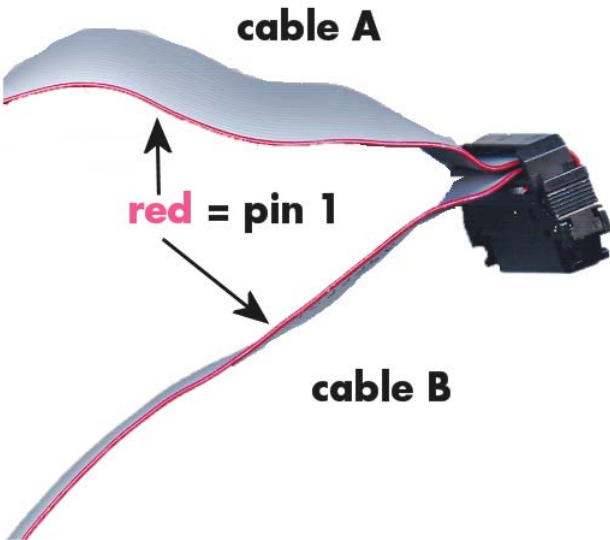
A3...A0 are the pins for the analog outputs, while D23...D0 are the 24 digital I/Os.

### Pin assignment of the multipin cable

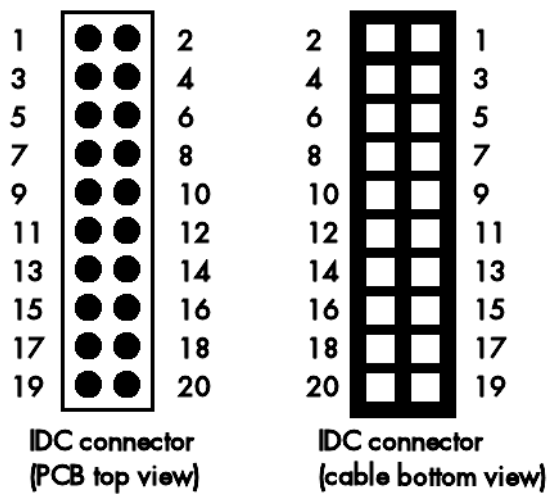
The 40 lead multipin cable is used for the additional digital inputs (on analog acquisition boards only) or additional digital outputs (on analog generation boards only) as well as for the digital I/O or pattern generator boards.

The flat ribbon cable is shipped with the boards that are equipped with one or more of the above mentioned options. The cable ends are assembled with two standard 20 pole IDC socket connector so you can easily make connections to your type of equipment or DUT (device under test).

The pin assignment is given in the table in the according chapter of the appendix.



## IDC footprints



The 20 pole IDC connectors have the following footprints. For easy usage in your PCB the cable footprint as well as the PCB top footprint are shown here. Please note that the PCB footprint is given as top view.



The following table shows the relation between the card connector pin and the IDC pin:

IDC footprint pin	Card connector pin
1	A1, A21, A41, A61, B1, B21, B41 or B61
3	A3, A23, A43, A63, B3, B23, B43 or B63
5	A5, A25, A45, A65, B5, B25, B45 or B65
7	A7, A27, A47, A67, B7, B27, B47 or B67
9	A9, A29, A49, A69, B9, B29, B49 or B69
11	A9, A29, A49, A69, B9, B29, B49 or B69
13	A13, A33, A53, A73, B13, B33, B53 or B73
15	A15, A35, A55, A75, B15, B35, B55 or B75
17	A17, A37, A57, A77, B17, B37, B57 or B77
19	A19, A39, A59, A79, B19, B39, B59 or B79

Card connector pin	IDC footprint pin
A2, A22, A42, A62, B2, B22, B42 or B62	2
A4, A24, A44, A64, B4, B24, B44 or B64	4
A6, A26, A46, A66, B6, B26, B46 or B66	6
A8, A28, A48, A68, B8, B28, B48 or B68	8
A10, A30, A50, A70, B10, B30, B50 or B70	10
A12, A32, A52, A72, B12, B32, B52 or B72	12
A14, A34, A54, A74, B14, B34, B54 or B74	14
A16, A36, A56, A76, B16, B36, B56 or B76	16
A18, A38, A58, A78, B18, B38, B58 or B78	18
A20, A40, A60, A80, B20, B40, B60 or B80	20