

Becker & Hickl GmbH  
Nahmitzer Damm  
12277 Berlin  
Tel. +49 30 787 56 32  
Fax. +49 30 787 57 34  
email: [info@becker-hickl.de](mailto:info@becker-hickl.de)  
<http://www.becker-hickl.de>

spcm.dll.doc



# **SPCM- 32 Bits Dynamic Link Libraries**

## **User Manual for SPC modules**

Version 2.7

June 2008

## Introduction

The SPCM 32 bits Dynamic Link Library contains all functions to control the whole family of SPC and DPC modules which work on PCI bus. This manual is oriented to the functions which are used to control all SPC modules types. Up to eight SPC modules of the same type can be controlled using the SPCM DLL. The functions work under Windows NT/2000/XP/Vista. The program which calls the DLLs must be compiled with the compiler option 'Structure Alignment' set to '1 Byte'.

The distribution disks contain the following files:

SPCM32.DLL	dynamic link library main file
SPCM32.LIB	import library file for Microsoft Visual C/C++, Borland C/C++ compilers
SPCM_DEF.H	Include file containing Types definitions, Functions Prototypes and Pre-processor statements
SPCM.INI	DLL initialisation file for SPC modules
DPC230.INI	DLL initialisation file for DPC modules
DPCDLL.DOC	Description file for DPC modules
SPCMDLL.DOC	This description file
USE_DPC.XXX	Set of files to compile and run a simple example of using SPC DLL functions for DPC-230 module. Source file of the example is the file use_dpc.c.
USE_SPCM.XXX	Set of files to compile and run a simple example of using SPC DLL functions for SPC modules. Source file of the example is the file use_spcm.c.

There is no special installation procedure required. Simply execute the setup program from the 1st distribution diskette and follow its instructions.

## SPCM-DLL Functions list

The following functions implemented in the SPCM-DLL are used for SPC modules

### Initialisation functions:

- SPC\_init
- SPC\_get\_init\_status
- SPC\_get\_module\_info
- SPC\_test\_id
- SPC\_set\_mode
- SPC\_get\_mode

### Setup functions:

- SPC\_get\_parameters
- SPC\_set\_parameters
- SPC\_get\_parameter
- SPC\_set\_parameter
- SPC\_get\_eeprom\_data

SPC\_write\_eeprom\_data  
SPC\_get\_adjust\_parameters  
SPC\_set\_adjust\_parameters  
SPC\_read\_parameters\_from\_inifile  
SPC\_save\_parameters\_to\_inifile

**Status functions:**

SPC\_test\_state  
SPC\_get\_sync\_state  
SPC\_get\_time\_from\_start  
SPC\_get\_break\_time  
SPC\_get\_actual\_coltime  
SPC\_read\_rates  
SPC\_clear\_rates  
SPC\_get\_sequencer\_state  
SPC\_read\_gap\_time  
SPC\_get\_scan\_clk\_state  
SPC\_get\_fifo\_usage

**Measurement control functions:**

SPC\_start\_measurement  
SPC\_pause\_measurement  
SPC\_restart\_measurement  
SPC\_stop\_measurement  
SPC\_set\_page  
SPC\_enable\_sequencer

**SPC memory transfer functions:**

SPC\_configure\_memory  
SPC\_fill\_memory  
SPC\_read\_data\_block  
SPC\_write\_data\_block  
SPC\_read\_fifo  
SPC\_read\_data\_frame  
SPC\_read\_data\_page  
SPC\_read\_block  
SPC\_save\_data\_to\_sdtfile

**Other functions:**

SPC\_get\_error\_string  
SPC\_init\_phot\_stream  
SPC\_close\_phot\_stream  
SPC\_get\_phot\_stream\_info  
SPC\_get\_photon  
SPC\_get\_detector\_info  
SPC\_close

Functions listed above must be called with C calling convention which is default for C and C++ programs.

Identical set of functions is available for environments like Visual Basic which requires `_stdcall` calling convention. Names of these functions have 'std' letters after 'SPC', for example, `SPCstd_read_block` it is `_stdcall` version of `SPC_read_block`.

Description and behaviour of these functions are identical to the functions from the first (default) set – the only difference is calling convention.

## Application Guide

### Initialisation of the SPC Measurement Parameters

Before a measurement is started the measurement parameter values must be written into the internal structures of the DLL functions (not directly visible from the user program) and sent to the control registers of the SPC module(s). This is accomplished by the function `SPC_init`. This function

- checks whether DLL is correctly registered ( looks for a BH license number and verifies it)
- reads the parameter values from a specified file
- checks and recalculates the parameters depending on hardware restrictions and adjust parameters from the EEPROM on the SPC module(s)
- sends the parameter values to the SPC control registers
- performs a hardware test of the SPC module(s)

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file `spcm.ini` or to start with `spcm.ini` and to introduce the desired changes.

```
; SPCM DLL initialisation file for SPC modules
; SPC parameters have to be included in .ini file only when parameter
; value is different from default.
; for DPC230 module use file dpc230.ini instead of this one

[spc_base]
simulation = 0                ; 0 - hardware mode(default) ,
                             ; >0 - simulation mode (see spcm_def.h for possible values)
pci_bus_no= -1               ; PCI bus on which SPC modules will be looking for
                             ; 0 - 255, default -1 ( all PCI busses will be scanned)
pci_card_no= -1              ; number of the SPC module on PCI bus
                             ; 0 - 7, default -1 (all modules on PCI bus)

[spc_module]                 ; SPC hardware parameters
cfd_limit_low= 5.0           ; for SPCx3x (140,150) -500 .. 0mV ,for SPCx0x 5 .. 80mV
                             ; default 5mV
cfd_limit_high= 80.0         ; 5 ..80 mV, default 80 mV, not for SPC130,140,150,930
cfd_zc_level= 0.0           ;for SPCx3x(140,150) -96 .. 96mV ,for SPCx0x -10 .. 10mV
                             ; default 0mV
cfd_holdoff= 5.0            ; for SPCx0x 5 .. 20 ns , default 5ns
                             ; for other modules doesn't exist
sync_zc_level= 0.0          ; for SPCx3x(140,150) -96 .. 96mV ,for SPCx0x -10 .. 10mV
                             ; default 0mV
sync_freq_div= 4            ; for SPC130,140,150,930 1,2,4
                             ; for other SPC modules 1,2,4,8,16 , default 4
sync_holdoff= 4.0           ; 4 .. 16 ns , default 4 ns, for SPC130,140,150,930 doesn't exist
sync_threshold= -20.0       ; for SPCx3x(140,150) -500 .. -20mV ,default -20 mV
                             ; for SPCx0x doesn't exist

tac_range= 50.0              ; 50 .. 5000 ns , default 50 ns
tac_gain= 1                  ; 1 .. 15 ,default 1
tac_offset=0.0              ; 0 .. 100% ,default 0%
tac_limit_low= 10.0         ; 0 .. 100% ,default 10%
tac_limit_high= 80.0       ; 0 .. 100% ,default 80%

adc_resolution= 10          ; 6,8,10,12 bits, default 10
```

```

extLatchDelay=0 ; (additionally 0,2,4 bits for SPC830,140,150,930 )
; 0 ..255 ns, default 0, for SPC130 doesn't exist
; for SPC140,150,930 only values 0,10,20,30,40,50 ns are possible

collectTime=0.01 ; 0.0001 .. 100000s , default 0.01s
repeatTime=10.0 ; 0.0001 .. 100000s , default 10.0s
stopOnTime=1 ; 0,1 , default 1
stopOnOvf=1 ; 0,1 , default 1
ditherRange=0 ; possible values - 0, 32, 64, 128, 256
; have meaning: 0, 1/64, 1/32, 1/16, 1/8

countIncr=1 ; 1 .. 255 , default 1
memBank=0 ; for SPC130,600,630, 150 : 0 , 1 , default 0
; for other SPC modules always 0

deadTimeComp=1 ; 0 , 1 , default 1
mode=0 ; for SPC7x0 , default 0
; 0 - normal operation (routing in),
; 1 - block address out, 2 - Scan In, 3 - Scan Out
; for SPC6x0 , default 0
; 0 - normal operation (routing in),
; 2 - FIFO mode 48 bits, 3 - FIFO mode 32 bits
; for SPC130 , default 0
; 0 - normal operation (routing in),
; 2 - FIFO mode
; for SPC140 , default 0
; 0 - normal operation (routing in),
; 1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out
; 5 - FIFO_mode 32 bits with markers ( FIFO_32M )
; for SPC150 , default 0
; 0 - normal operation (routing in),
; 1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out
; 5 - FIFO_mode 32 bits with markers ( FIFO_32M ), with FPGA v. > B0
; for SPC830,930 , default 0
; 0 - normal operation (routing in),
; 1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out
; 4 - Camera mode ( only SPC930 )
; 5 - FIFO_mode 32 bits with markers ( FIFO_32M ),
; SPC830 with FPGA v. > C0
; for SPC7x0, 830, 140,150, 930 modules in scanning modes 1 .. 65536, default 1
; for SPC7x0, 830, 140,150, 930 modules in scanning modes 1 .. 65536, default 1
; number of X routing channels in Scan In & Scan Out modes
; for SPC7x0,830,140,150,930 modules
; 1 .. 128, ( SPC7x0,830 ), 1 .. 16 ( SPC140,150,930), default 1
; number of Y routing channels in Scan In & Scan Out modes
; for SPC7x0,830,140,150,930 modules
; 1 .. 128, ( SPC7x0,830 ), 1 .. 16 ( SPC140,150,930), default 1
; INT(log2(scan_size_x)) + INT(log2(scan_size_y)) +
; INT(log2(scan_rout_x)) + INT(log2(scan_rout_y)) <=
; max number of scanning bits
; max number of scanning bits depends on the current adc_resolution:
; 12 (10 for SPC7x0,140,150) - 12
; 14 (12 for SPC7x0,140,150) - 10
; 16 (14 for SPC7x0,140,150) - 8
; 18 (16 for SPC7x0,140,150) - 6
; 20 (18 for SPC140,150) - 4
; 22 (20 for SPC140,150) - 2
; 24 (22 for SPC140,150) - 0

scanPolarity=0 ; for SPC7x0, 830, 140,150, 930 modules in scanning modes, default 0
; bit 0 - polarity of HSYNC, bit 1 - polarity of VSYNC,
; bit 2 - pixel clock polarity
; bit = 0 - falling edge(active low)
; bit = 1 - rising edge(active high)
; for SPC140,150,830 in FIFO_32M mode
; bit = 8 - HSYNC (Line) marker disabled (1) or enabled (0, default )
; when disabled, line marker will not appear in FIFO photons stream

scanFlyback=0 ; for SPC7x0,830,140,150, 930 modules in Scan Out mode, default 0
; bits 15-0 Flyback X in number of pixels
; bits 31-16 Flyback Y in number of lines

scanBorders=0 ; for SPC7x0,830,140,150, 930 modules in Scan In mode, default 0
; bits 15-0 Upper boarder, bits 31-16 Left boarder

pixelTime=200e-9 ; pixel time in sec for SPC7x0,830,140,150, 930 modules in Scan In mode,
; 50e-9 .. 1.0 , default 200e-9

pixelClock=0 ; source of pixel clock for SPC7x0,830,140,150, 930 modules in Scan In mode
; 0 - internal, 1 - external, default 0
; for SPC140,150,830 in FIFO_32M mode it disables/enables pixel markers

```

```

;          in photons stream
line_compression= 1 ; line compression factor for SPC7x0,830,140,150, 930 modules in Scan In mode,
; 1,2,4,8,16,32,64,128, default 1
trigger = 0 ; external trigger condition
; bits 1 & 0 mean : 00 - ( value 0 ) none(default),
;                  01 - ( value 1 ) active low,
;                  10 - ( value 2 ) active high
; when sequencer is enabled on SPC130(6x0) modules additionally
; bits 9 & 8 of the value mean:
; 00 - trigger only at the start of the sequence,
; 01 ( 100 hex, 256 decimal ) - trigger on each bank
; 11 ( 300 hex, 768 decimal ) - trigger on each curve in the bank
; for SPC140,150 and SPC130 (FPGA v. > C0) multi-module configuration
; bits 13 & 12 of the value mean:
; x0 - module doesn't use trigger bus ( trigger defined via bits 0-1),
; 01 ( 1000 hex, 4096 decimal ) - module uses trigger bus as slave
; ( waits for the trigger on master),
; 11 ( 3000 hex, 12288 decimal ) - module uses trigger bus as master
; ( trigger defined via bits 0-1 ),
;
; ( only one module can be the master )

ext_pixclk_div= 1 ; divider of external pixel clock for SPC7x0,830,140, 930 modules
; in Scan In mode 1 .. 0x3ff, default 1
rate_count_time= 1.0 ; rate counting time in sec default 1.0 sec
; for SPC130,150, 830, 930 can be : 1.0s, 0.25s, 0.1s, 0.05s
; for SPC140 fixed to 50ms

macro_time_clk= 0 ; macro time clock definition for SPC130, 140,150, 830, 930 in FIFO mode
; for SPC130, 140:
; 0 - 50ns (default), 25ns for SPC150 & 140 with FPGA v. > B0 ,
; 1 - SYNC freq., 2 - 1/2 SYNC freq.,
; 3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.
; for SPC830: 0 - 50ns (default), 1 - SYNC freq.,
; for SPC930: 0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.

add_select= 0 ; selects ADD signal source for all modules except SPC930:
; 0 - internal (ADD only) (default), 1 - external
adc_zoom = 0 ; ADC zoom level for module SPC830, 140,150, 930 default 0
; bit 4 = 0(1) - zoom off(on) ,
; bits 0-3 zoom level,
; 0 - zoom of the 1st 1/16th of ADC range,
; 15 - zoom of the 16th 1/16th of ADC range

img_size_x = 1 ; image X size ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
; 1 .. 1024, default 1
img_size_y = 1 ; image Y size ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
; actually equal to img_size_x ( quadratic image )
img_rout_x = 1 ; no of X routing channels ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
; 1 .. 16, default 1
img_rout_y = 1 ; no of Y routing channels ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
; 1 .. 16, default 1
xy_gain = 1 ; selects gain for XY ADCs for module SPC930, 1,2,4 , default 1
master_clock = 0 ; use Master Clock( 1 ) or not ( 0 ), default 0,
; only for SPC140 ,150 multi-module configuration
; - value 2 (when read) means Master Clock state was set
; by other application and cannot be changed

adc_sample_delay = 0 ; ADC's sample delay, only for module SPC930
; 0,10,20,30,40,50 ns (default 0 )

detector_type = 1 ; detector type used in Camera mode, only for module SPC930,
; 1 .. 9899, default 1
; normally it is recognised automatically from the corresponding .bit file
; 1 - Hamamatsu Resistive Anode 4 channels detector
; 2 - Wedge & Strip 3 channels detector

x_axis_type = 0 ; X axis representation, only for module SPC930
; 0 - time (default ), 1 - ADC1 Voltage,
; 2 - ADC2 Voltage, 3 - ADC3 Voltage, 4 - ADC4 Voltage

```

After calling the `SPC_init` function the measurement parameters from the initialisation file are present in the module control registers and in the internal data structures of the DLLs. To give the user access to the parameters, the function `SPC_get_parameters` is provided. This function

transfers the parameter values from the internal structures of the DLLs into a structure of the type SPCdata (see spcm\_def.h) which has to be defined by the user. The parameter values in this structure are described below.

unsigned short base_adr	base I/O address on PCI bus
short init	set to initialisation result code
float cfd_limit_low	SPCx3x(140,150) -500 .. 0mV ,for SPCx0x 5 .. 80mV
float cfd_limit_high	5 ..80 mV, default 80 mV , not for SPC130,140,150,930
float cfd_zc_level	SPCx3x(140,150) -96 .. 96mV, SPCx0x -10 .. 10mV
float cfd_holdoff	SPCx0x: 5 .. 20 ns, other modules: no influence
float sync_zc_level	SPCx3x(140,150): -96 .. 96mV, SPCx0x: -10..10mV
float sync_holdoff	4 .. 16 ns ( SPC130,140,150,930 : no influence)
float sync_threshold	SPCx3x(140,150): -500 .. -20mV, SPCx0x: no influence
float tac_range	50 .. 5000 ns
short sync_freq_div	1,2,4,8,16 ( SPC130,140,150,930 : 1,2,4)
short tac_gain	1 .. 15
float tac_offset	0 .. 100%
float tac_limit_low	0 .. 100%
float tac_limit_high	0 .. 100%
short adc_resolution	6,8,10,12 bits, default 10 (additionally 0,2,4 bits for SPC830, 140,150, 930 )
short ext_latch_delay	0 ..255 ns, SPC130: no influence SPC140,150, 930 : only values 0,10,20,30,40,50 ns are possible
float collect_time	0.0001 .. 100000s , default 0.01s
float display_time	0.0001 .. 100000s , default 0.01s
float repeat_time	0.0001 .. 100000s , default 0.01s
short stop_on_time	1 (stop) or 0 (no stop)
short stop_on_ovfl	1 (stop) or 0 (no stop)
short dither_range	possible values - 0, 32, 64, 128, 256 have meaning: 0, 1/64, 1/32, 1/16, 1/8
short count_incr	1 .. 255
short mem_bank	for SPC130,600,630,150 : 0 , 1 , default 0 other SPC modules: always 0
short dead_time_comp	0 (off) or 1 (on)
unsigned short scan_control	SPC505(535,506,536) scanning(routing) control word other SPC modules always 0
short routing_mode	SPC150(140) - bits 8 - 11 - enable(1)/disable(0), default 0 of recording Markers 0-3 entries in FIFO mode other SPC modules not used
float tac_enable_hold	SPC230 10.0 .. 265.0 ns - duration of TAC enable pulse ,other SPC modules always 0
short pci_card_no	module no on PCI bus (0-7)
unsigned short mode;	for SPC7x0 , default 0 0 - normal operation (routing in), 1 - block address out, 2 - Scan In, 3 - Scan Out for SPC6x0 , default 0 0 - normal operation (routing in) 2 - FIFO mode 48 bits, 3 - FIFO mode 32 bits for SPC130 , default 0 0 - normal operation (routing in) 2 - FIFO mode for SPC140 , default 0 0 - normal operation (routing in) 1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out 5 - FIFO_mode 32 bits with markers ( FIFO_32M ), with FPGA v. > B0 for SPC150 , default 0 0 - normal operation (routing in) 1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out 5 - FIFO_mode 32 bits with markers ( FIFO_32M ) for SPC830,930 , default 0 0 - normal operation (routing in) 1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out 4 - Camera mode ( only SPC930 ) 5 - FIFO_mode 32 bits with markers ( FIFO_32M ), SPC830 with FPGA v. > B0
unsigned long scan_size_x;	for SPC7x0,830,140,150,930 modules in scanning modes 1 .. 65536, default 1
unsigned long scan_size_y;	for SPC7x0,830,140,150,930 modules in scanning modes 1 .. 65536, default 1
unsigned long scan_rout_x;	number of X routing channels in Scan In & Scan Out modes for SPC7x0,830,140,150,930 modules 1 .. 128, ( SPC7x0,830 ), 1 .. 16 (SPC140,150,930), default 1
unsigned long scan_rout_y;	number of Y routing channels in Scan In & Scan Out modes

for SPC7x0,830,140,150,930 modules  
1 .. 128, ( SPC7x0,830 ), 1 .. 16 (SPC140,150,930), default 1  
 $\text{INT}(\log_2(\text{scan\_size\_x})) + \text{INT}(\log_2(\text{scan\_size\_y})) +$   
 $\text{INT}(\log_2(\text{scan\_rout\_x})) + \text{INT}(\log_2(\text{scan\_rout\_y})) \leq$  max number of scanning bits  
max number of scanning bits depends on current adc\_resolution:

12 (10 for SPC7x0,140,150)	-	12
14 (12 for SPC7x0,140,150)	-	10
16 (14 for SPC7x0,140,150)	-	8
18 (16 for SPC7x0,140,150)	-	6
20 (18 for SPC140,150)	-	4
22 (20 for SPC140,150)	-	2
24 (22 for SPC140,150)	-	0

unsigned long scan\_flyback; for SPC7x0,830,140,150,930 modules in Scan Out or Rout Out mode, default 0  
bits 15-0 Flyback X in number of pixels  
bits 31-16 Flyback Y in number of lines

unsigned long scan\_borders; for SPC7x0,830,140,150,930 modules in Scan In mode, default 0  
bits 15-0 Upper boarder, bits 31-16 Left boarder

unsigned short scan\_polarity; for SPC7x0,830,140,150,930 modules in scanning modes, default 0  
bit 0 - polarity of HSYNC (Line), bit 1 - polarity of VSYNC (Frame),  
bit 2 - pixel clock polarity  
bit = 0 - falling edge(active low)  
bit = 1 - rising edge(active high)

for SPC140,150,830 in FIFO\_32M mode  
bit = 8 - HSYNC (Line) marker disabled (1) or enabled (0, default )  
when disabled, line marker will not appear in FIFO photons stream

unsigned short pixel\_clock; for SPC7x0,830,140,150,930 modules in Scan In mode,  
pixel clock source, 0 - internal,1 - external, default 0  
for SPC140,150,830 in FIFO\_32M mode it disables/enables pixel markers  
in photons stream

unsigned short line\_compression; line compression factor for SPC7x0,830,140,150,930 modules  
in Scan In mode, 1,2,4,8,16,32,64,128, default 1

unsigned short trigger; external trigger condition -  
bits 1 & 0 mean : 00 - ( value 0 ) none(default),  
01 - ( value 1 ) active low,  
10 - ( value 2 ) active high  
when sequencer is enabled on SPC130(6x0) modules additionally  
bits 9 & 8 of the value mean:  
00 - trigger only at the start of the sequence,  
01 ( 100 hex, 256 decimal ) - trigger on each bank  
11 ( 300 hex, 768 decimal ) - trigger on each curve in the bank  
for SPC140,150 and SPC130 (FPGA v. > C0) multi-module configuration  
bits 9 & 8 of the value mean:  
x0 - module does not use trigger bus ( trigger defined via bits 0-1),  
01 ( 1000 hex, 4096 decimal ) - module uses trigger bus as slave  
( waits for the trigger on master),  
11 ( 3000 hex, 12288 decimal ) - module uses trigger bus as master  
( trigger defined via bits 0-1),  
( only one module can be the master )

float pixel\_time; pixel time in sec for SPC7x0,830,140,150,930 modules in Scan In mode,  
50e-9 .. 1.0 , default 200e-9

unsigned long ext\_pixclk\_div; divider of external pixel clock for SPC7x0,830,140,150,930 modules  
in Scan In mode, 1 .. 0x3fe, default 1

float rate\_count\_time; rate counting time in sec default 1.0 sec  
for SPC130,830,930,150 can be : 1.0s, 0.25s, 0.1s, 0.05s  
for SPC140 fixed to 50ms

short macro\_time\_clk; macro time clock definition for SPC130,140,150,830,930 in FIFO mode  
for SPC130, 140,150:  
0 - 50ns (default), 25ns for SPC150 & 140 with FPGA v. > B0 ,  
1 - SYNC freq., 2 - 1/2 SYNC freq.,  
3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.  
for SPC830: 0 - 50ns (default), 1 - SYNC freq.,  
for SPC930: 0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.,

short add\_select; selects ADD signal source for all modules except SPC930 :  
0 - internal (ADD only) (default), 1 - external

short test\_eep 0: EEPROM is not read and not tested, default adjust parameters are used  
1: EEPROM is read and tested for correct checksum

short adc\_zoom; selects ADC zoom level for module SPC830,140,150,930 default 0  
bit 4 = 0(1) - zoom off(on),  
bits 0-3 zoom level =  
0 - zoom of the 1st 1/16th of ADC range,  
15 - zoom of the 16th 1/16th of ADC range

unsigned long img\_size\_x; image X size ( SPC140,150,830 in FIFO\_32M mode, SPC930 in Camera mode ),  
1 .. 1024, default 1

unsigned long img\_size\_y; image Y size ( SPC140,150,830 in FIFO\_32M mode, SPC930 in Camera mode ),

unsigned long img_rout_x;	actually equal to img_size_x ( quadratic image ) no of X routing channels ( SPC140,150,830 in FIFO_32M mode, SPC930 in Camera mode ), 1 .. 16, default 1
unsigned long img_rout_y;	no of Y routing channels ( SPC140,150,830 in FIFO_32M mode, SPC930 in Camera mode ), 1 .. 16, default 1
short xy_gain;	selects gain for XY ADCs for module SPC930, 1,2,4, default 1
short master_clock;	use Master Clock( 1 ) or not ( 0 ), default 0, only for SPC140, 150 multi-module configuration - value 2 (when read) means Master Clock state was set by other application and cannot be changed
short adc_sample_delay;	ADC's sample delay, only for module SPC930 0,10,20,30,40,50 ns (default 0 )
short detector_type;	detector type used in Camera mode, only for module SPC930 1 .. 9899, default 1, normally recognised automatically from the corresponding .bit file 1 - Hamamatsu Resistive Anode 4 channels detector 2 - Wedge & Strip 3 channels detector
short x_axis_type;	X axis representation, only for module SPC930 0 - time (default ) , 1 - ADC1 Voltage, 2 - ADC2 Voltage, 3 - ADC3 Voltage, 4 - ADC4 Voltage
unsigned short chan_enable;	for module DPC230 - enable(1)/disable(0) input channels bits 0-7 - en/disable TTL channel 0-7 in TDC1 bits 8-9 - en/disable CFD channel 0-1 in TDC1 bits 12-19 - en/disable TTL channel 0-7 in TDC2 bits 20-21 - en/disable CFD channel 0-1 in TDC2
unsigned short chan_slope;	for module DPC230 - active slope of input channels 1 - rising, 0 - falling edge active bits 0-7 - slope of TTL channel 0-7 in TDC1 bits 8-9 - slope of CFD channel 0-1 in TDC1 bits 12-19 - slope of TTL channel 0-7 in TDC2 bits 20-21 - slope of CFD channel 0-1 in TDC2
unsigned long chan_spec_no;	for module DPC230 - channel numbers of special inputs, default 0x8813 bits 0-4 - reference chan. no ( TCSPC and Multiscaler modes ) default = 19, value: 0-1 CFD chan. 0-1 of TDC1, 2-9 TTL chan. 0-7 of TDC1 10-11 CFD chan. 0-1 of TDC2, 12-19 TTL chan. 0-7 of TDC2 bits 8-10 - frame clock TTL chan. no ( imaging modes ) 0-7, default 0 bits 11-13 - line clock TTL chan. no ( imaging modes ) 0-7, default 1 bits 14-16 - pixel clock TTL chan. no ( imaging modes ) 0-7, default 2 bit 17 - TDC no for pixel, line, frame clocks ( imaging modes ) 0 = TDC1, 1 = TDC2, default 0 bits 18-31 - not used

To send the complete parameter set back to the DLLs and to the SPC module (e.g. after changing parameter values) the function **SPC\_set\_parameters** is used. This function checks and - if required - recalculates all parameter values due to cross dependencies and hardware restrictions. Therefore, it is recommended to read the parameter values after calling **SPC\_set\_parameters** by **SPC\_get\_parameters**.

Parameters set can be saved to ini\_file using the function **SPC\_save\_parameters\_to\_inifile**. **SPC\_read\_parameters\_from\_inifile** enables reading back saved parameters from ini\_file and then with **SPC\_set\_parameters** send it to the SPC module.

Single parameter values can be transferred to or from the DLL and module level by the functions **SPC\_set\_parameter** and **SPC\_get\_parameter**. To identify the desired parameter, the parameter identification par\_id is used. For the parameter identification keywords are defined in spcm\_def.h.

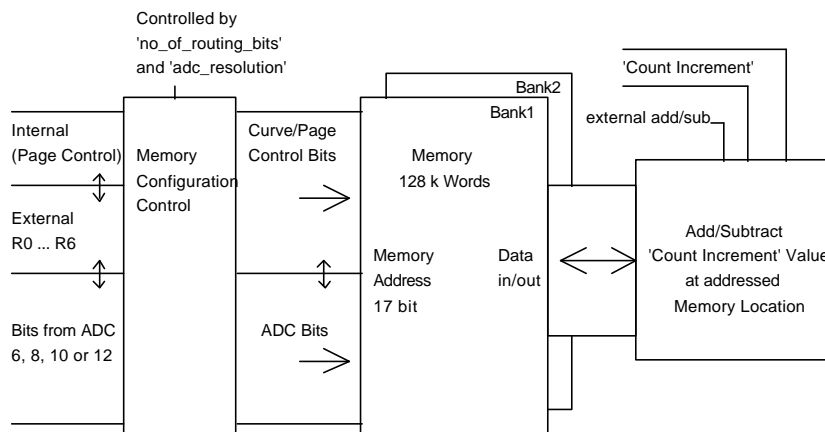
## Memory Configuration

The SPC memory is interpreted as a set of 'pages'. One page contains a number of 'blocks'. One block contains one decay curve. The number of points per block (per curve) is defined by

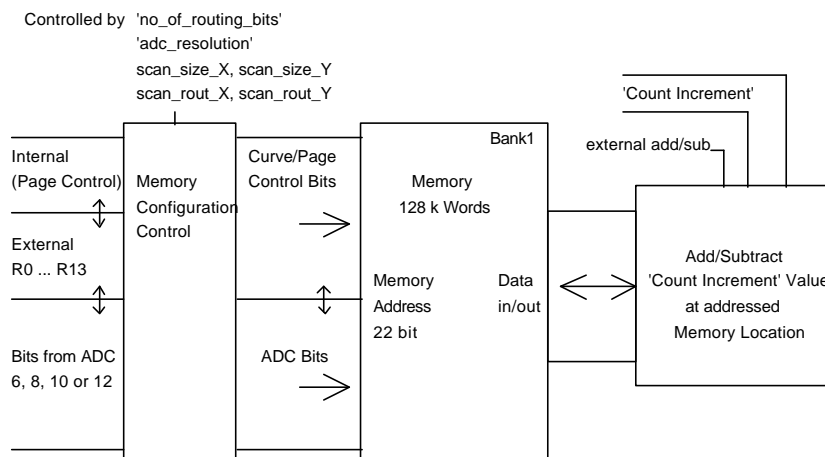
the ADC resolution. The number of blocks per page depends on the number of points per block and on the number of detector channels (PointsX and PointsY)

In the scanning modes of the SPC-7(8)(9)(140,150), a page contains a number of ‘frames’ (normally 1). Each frame contains a number of blocks. The number of blocks per page depends on the number of points per block and on the number of detector channels (PointsX and PointsY) and on the scanning parameters (pixels per line and lines per frame).

The figures below show the influence of these parameters on the memory configuration.



Memory Control (SPC-400/430/600/630/130)



Memory Control (SPC-500/530/700/730)

To configure the SPC memory depending on the module type, the ADC resolution, the number of detector channels in normal operation modes the function ‘SPC\_configure\_memory’ is provided. This procedure has to be called before the first access to the SPC memory or before a measurement is started and always after setting ADC resolution parameter value. The memory configuration determines in which part of the SPC memory the measurement data is recorded (see also SPC\_set\_page).

SPC-130/600/630 modules:

The length of the recorded curves is determined by the ADC resolution and can range from 64 to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

#### SPC-130/600/630/150 modules in the Histogram modes:

The length of the recorded curves is determined by the ADC resolution and can range from 64 (0 for SPC-150) to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

#### SPC-130/600/630/830/140/150/930 modules in the Fifo modes:

The module memory is configured as a FIFO memory – there are no curves and pages. Instead, a stream of collected photons is written to the fifo. A SPC\_configure\_memory function call is not required.

#### SPC-700/730/830/140/150/930 modules, Normal operation modes:

The length of the recorded curves is determined by the ADC resolution and can range from 0(SPC-830/140/150/930) or 64(SPC-7x0) to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

#### SPC-700/730/830/140/150/930 modules, Scanning modes:

The Memory configuration is not done by SPC\_configure\_memory. Instead, the memory is configured by but by setting the parameters:

ADC\_RESOLUTION – defines block\_length,  
SCAN\_SIZE\_X, SCAN\_SIZE\_Y – defines blocks\_per\_frame  
SCAN\_ROUT\_X, SCAN\_ROUT\_Y – defines frames\_per\_page

However, after setting these parameters SPC\_configure\_memory should be called with 'adc\_resolution' = -1 to get the current state of the DLL SPCMemConfig structure.

To ensure correct access to the curves in the memory by the memory read/write functions, the SPC\_configure\_memory function loads a structure of the type SPCMemConfig with the values listed below:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0,130, 150)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page (Normally 1)
long maxpage	max number of pages to use in a measurement
short block_length	Number of curve points 16-bits words per block (curve)

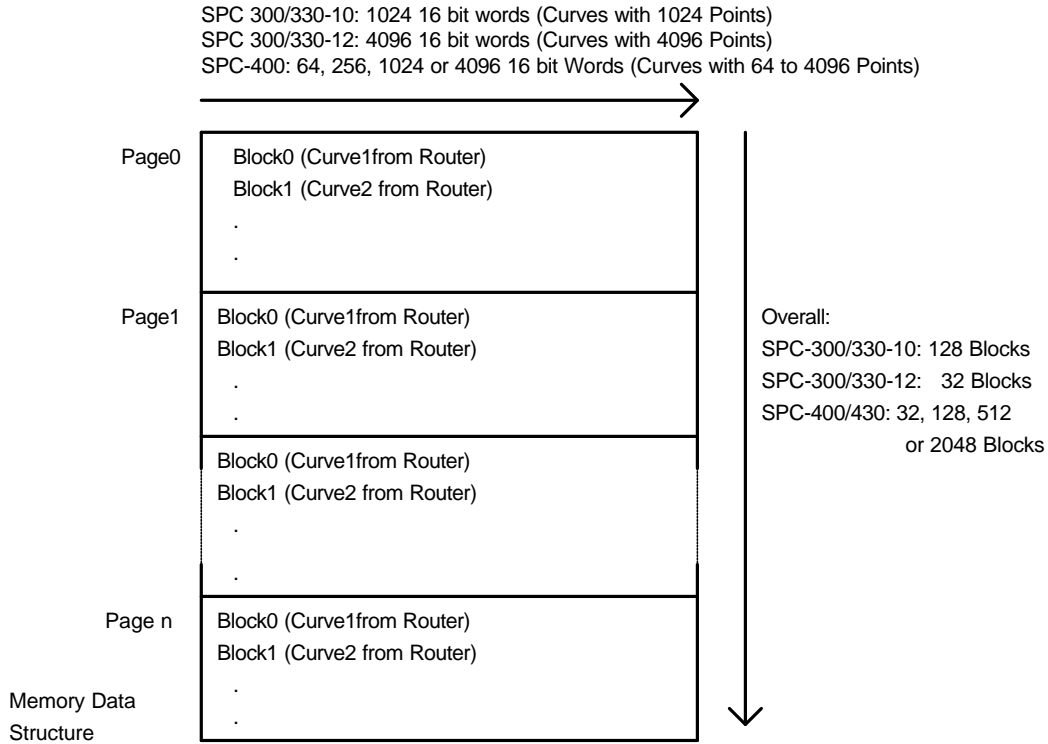
### **Memory Read/Write Functions**

Reading and writing the memory of the SPC module is accomplished by the functions **SPC\_read\_data\_block** and **SPC\_write\_data\_block**. To fill the memory with a constant value (or to clear the memory) the function **SPC\_fill\_memory** is available.

For reading whole pages or frames from the memory **SPC\_read\_data\_page** and **SPC\_read\_data\_frame** functions are available.

To read one data block from SPC-7x0/830/140/150/930 modules in scanning modes **SPC\_read\_block** function is available.

For all memory read/write functions the desired curve within the desired memory page is specified by the parameters 'block' and 'page'. The meaning of these parameters is shown in the table below.



The memory is divided in a number of 'pages' which contain a number of 'frames' each. Normally number of frames is equal 1, so page = frame. Each frame(page) contains a number of 'blocks'. Each block contains one curve. The number of blocks per page depends on the number of the detector channels used or number of scanning bits for SPC7x0/830/140/150/930 modules in scanning modes. Therefore, the memory structure is determined by the number of routing bits and the ADC resolution. The memory is configured by the function **SPC\_configure\_memory** (see 'Memory Configuration').

Using **SPC\_save\_data\_to\_sdtfile** function measurements results (read from SPC memory) can be saved in a .sdt file. Such file can then be loaded into the SPC standard measurement software.

**Standard Measurements**

The most important measurement functions are listed below.

**SPC\_set\_page** sets the memory page into which the measurement data is to be stored.

**SPC\_test\_state** sets a state variable according to the current state of the measurement. The function is used to control the measurement loop. The status bits delivered by the function are listed below (see also SPCM\_DEF.H).

SPC_OVERFLOW	0x1	stopped on overflow
SPC_OVERFLOW	0x2	overflow occurred
SPC_TIME_OVER	0x4	stopped on expiration of collection timer
SPC_COLTIM_OVER	0x8	collection timer expired

SPC_CMD_STOP	0x10	stopped on user command
SPC_ARMED	0x80	measurement in progress (current bank)
SPC_REPTIM_OVER	0x20	repeat timer expired
SPC_COLTIM_2OVER	0x100	second overflow of collection timer
SPC_REPTIM_2OVER	0x200	second overflow of repeat timer

For SPC600(630) and SPC130 modules only :

SPC_SEQ_GAP	0x40	Sequencer is waiting for other bank to be armed
-------------	------	---

For SPC600(630)(130)(140)(830)(150) and SPC930 modules only :

SPC_FOVFL	0x400	Fifo overflow, data lost
SPC_FEMPTY	0x800	Fifo empty

For SPC700(730)(140) (830) (150) and SPC930 modules only:

SPC_SCRDY	0x400	Scan ready (data can be read )
SPC_FBRDY	0x800	Flow back of scan finished
SPC_MEASURE	0x40	Measurement active = no margin, no wait for trigger, armed

SPC_WAIT_TRG	0x1000	Wait for external trigger
SPC_HFILL_NRDY	0x8000	hardware fill not finished

For SPC140 (150) and SPC930 modules only:

SPC_SEQ_STOP	0x4000	disarmed (measurement stopped) by sequencer
--------------	--------	---

For SPC150 modules only:

SPC_SEQ_GAP150	0x2000	Sequencer is waiting for other bank to be armed
----------------	--------	---

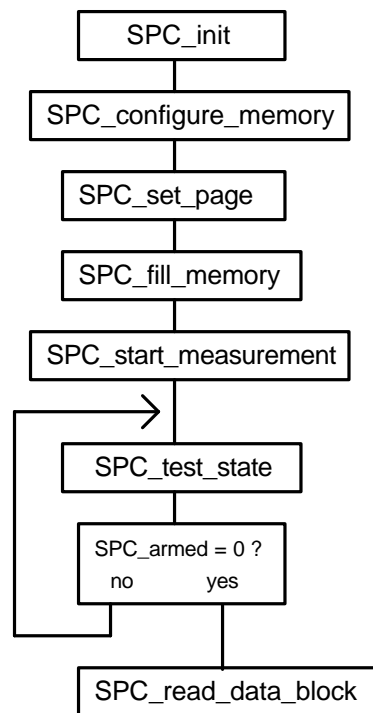
For SPC140 (150) and SPC830 modules in FIFO\_32M mode

SPC_WAIT_FR	0x2000	FIFO IMAGE measurement waits for the frame signal to stop
-------------	--------	---

**SPC\_start\_measurement** starts the measurement with the parameters set before by the SPC\_init, SPC\_set\_parameters or SPC\_set\_parameter functions. When the measurement is started by SPC\_start\_measurement, also the collection timer and the repeat timer are started. In the standard mode, i.e. when intensity-versus-time functions are recorded in one or more detector channels, the measurement stops when the specified stop condition appears (collection time expired, overflow or stop by SPC\_stop\_measurement).

**SPC\_stop\_measurement** is used to stop the measurement by a software command.

A simple measurement sequence is shown in the block diagram below.

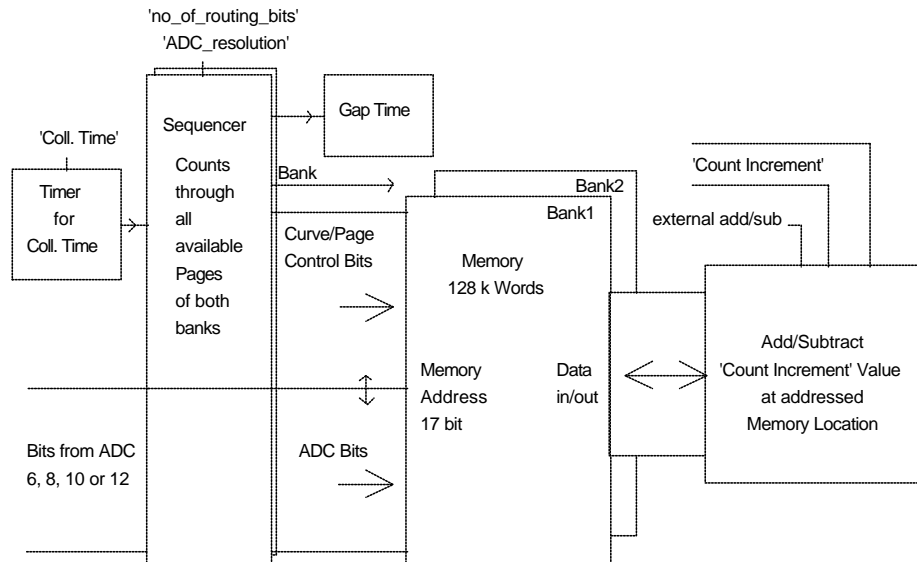


At the beginning, the measurement parameters are read from an initialisation file and send to the SPC module by `SPC_init`, and the memory is configured by `SPC_configure_memory`. The memory page in which the data is to be recorded is set by `SPC_set_page`. `SPC_fill_memory` is used to clear the data blocks (normally the current page) into which the data will be measured.

When the measurement has been started by `SPC_start_measurement` a decay curve (or several decay curves if a router is used and `no_of_routing_bits` was  $>0$  in the call of `SPC_configure_memory`). The measurement runs until a stop condition (specified in the measurement parameters) is reached. In this case the call of `SPC_test_state` returns `SPC_armed = 0` indicating that the measurement has been stopped and the data can be read from the module memory.

### Measurements with the SPC-6x0, SPC-130 and SPC-150 Sequencer

In the SPC-6x0, SPC-130 and SPC-150 modules a sequencer is available which automatically repeats the measurement with the specified collection time interval while switching through all available memory pages of both memory banks. Number of memory pages depends on the module's bank size ( e.g SPC-150 bank is 16 times bigger than SPC-130 bank) and on ADC resolution ( for SPC-150 additionally 0, 2 and 4 ADC bits can be used). The figure below shows the structure of the measurement system in the case that the sequencer is enabled.



The sequencer is controlled by the 'collection time' timer. The signals of several detector channels (specified by the parameter `no_of_routing_bits` via the function `SPC_configure_memory`) can be recorded simultaneously. When the collection time is over, the measurement is continued in the next memory page. When all pages of the current memory bank are filled with data, the measurement is continued in the other memory bank. A typical program sequence is shown in the block diagram below.

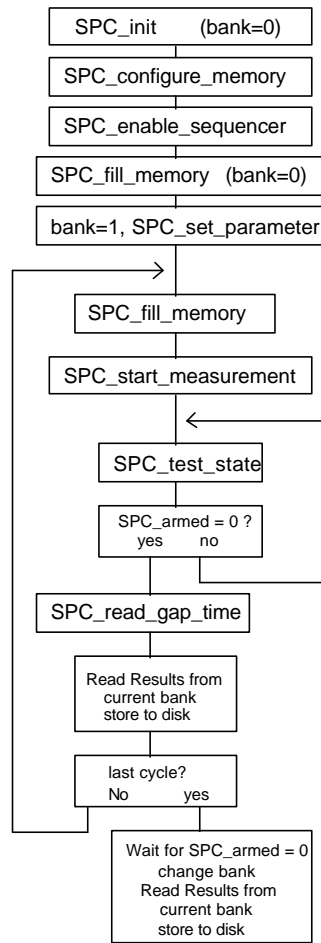
At the beginning, the measurement parameters are read from an initialisation file and sent to the SPC module by `SPC_init`. The memory is configured by `SPC_configure_memory` and the sequencer is enabled by **`SPC_enable_sequencer`**.

A simplified program for a sequencer measurement is shown in the block diagram below.

For this example it is assumed that the actual memory bank was set to 0 by `SPC_init` (`mem_bank = 0` in the initialisation data). In the next call of `SPC_fill_memory` all blocks and pages of this bank are cleared. Then the bank is switched to 1, and bank 1 is cleared.

After starting the measurement by `SPC_start_measurement`, the module records decay curves (or sets of decay curves if a router is used and `no_of_routing_bits` is greater than 0). Each recording lasts for the programmed collection time and is stored into the next page of the memory of the current memory bank (1). A call of `SPC_test_state` returns the `SPC_armed` bit = 1 in this situation.

When the 'current' memory bank (1) is full, the measurement proceeds in the other ('alternate') bank (0). A call of `SPC_test_state` returns `SPC_armed = 0` now, indicating that the current bank (1) is not longer used by the measurement. The software now reads the data from the current bank (1) while the measurement proceeds in the alternate bank (0).



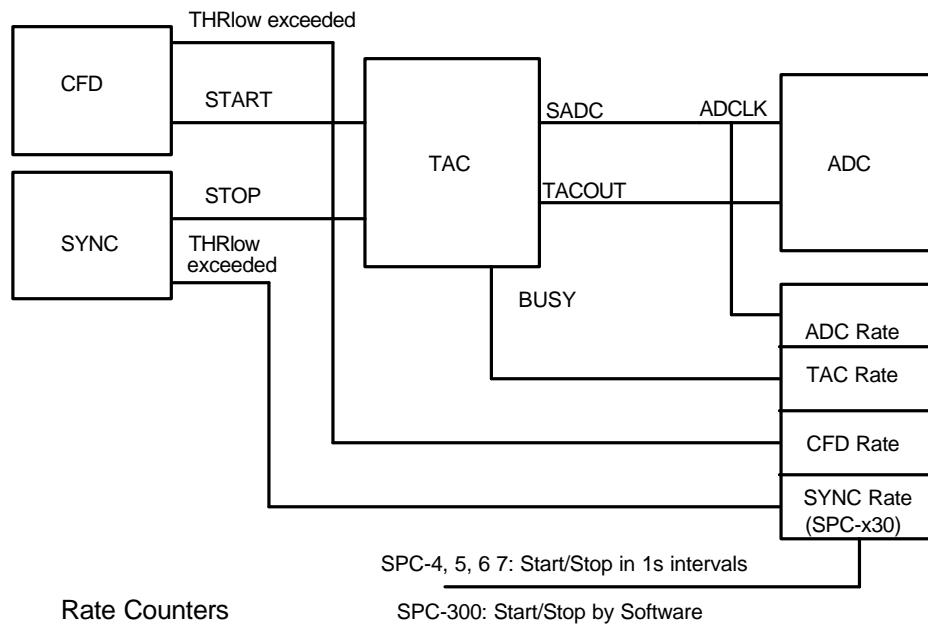
To measure an unlimited number of data blocks, the sequence is repeated in a loop. Thus, after reading the data from the current bank, this bank is cleared (SPC\_fill\_memory), while the measurement is still running in the alternate bank. The subsequent restarting of the measurement (SPC\_start\_measurement) sets SPC\_armed to 1 again. Because the sequencer is running, the function SPC\_start\_measurement reverses the memory banks, i.e. subsequent read operations will deliver data from the bank in which the measurement is still running. Because the measurement is already restarted the measurement immediately proceeds in the alternate (previously cleared) bank. SPC\_armed is reset to indicate that the current bank (with the measured data) is not longer needed by the measurement system and can be read by the software. The sequence continues until a specified number of cycles is reached.

After the last cycle the measurement is still running in the alternate bank. Therefore, the program waits until the measurement is finished (SPC\_test\_state returns SPC\_armed=0). The measurement now stops because no restart command has been issued, the banks are reversed and the measured data is read by the software.

Normally, the data readout and the bank clearing should be accomplished in a time shorter than the overall measurement time for one memory bank. If the end of the alternate bank is reached by the measurement before a new start command has been issued, the measurement stops until the start command is received. In the latter case a time gap occurs in the sequence of the measured decay curves. The gap time can (but need not) be determined by SPC\_read\_gap\_time).

## Rate Counters

The operation of the rate counters in the SPC modules is illustrated in the figure below.



The CFD rate counter counts all pulses that exceed the lower discriminator threshold of the CFD.

The SYNC rate counter counts all pulses that exceed the lower discriminator threshold of the SYNC input. The sync rate counter is present only in the SPC-130, -140, -150, -630, -730, -830 and -930 modules. To check whether the SYNC input triggers, the function **SPC\_get\_sync\_state** can be used. This function is available for all module types.

The TAC rate is the conversion rate of the TAC. Because the TAC does not accept start pulses during the conversion of a previous pulse, the TAC rate is lower than the CFD rate.

The ADC rate is the conversion rate of the ADC. Because the ADC is not started for events outside the selected TAC window the ADC rate is usually smaller than the TAC rate.

Integration time of rate values is equal 1sec, but for SPC-130/830/930/150 modules can have also other values according to the parameter **RATE\_COUNT\_TIME** (1.0s, 250ms, 100ms, 50ms are possible). For SPC-140 module integration time is equal 50 ms.

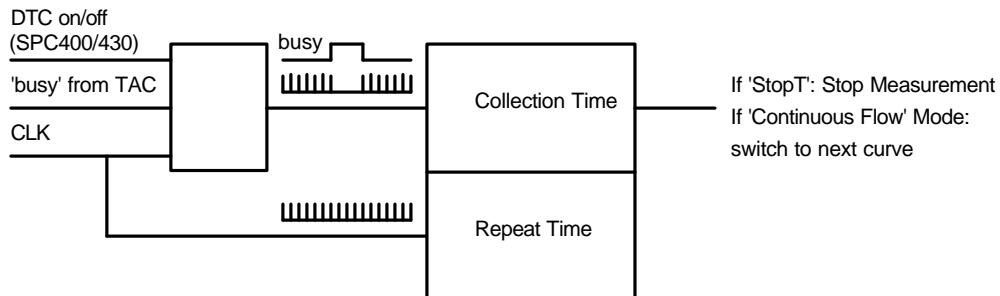
The rates are read by the **SPC\_read\_rates** function. The results are stored into a structure of the following type:

float sync_rate	SYNC rate in counts/s
float cfd_rate	CDF rate in counts/s
float tac_rate	TAC rate in counts/s
float adc_rate	ADC rate in counts/s

To get correct results the **SPC\_clear\_rates** function must be called before the first call of **SPC\_read\_rates**

## On-Board Timers

The structure of the timers of the SPC modules is shown in the figure below.



The timer for the collection time interval can be operated with or without 'dead time compensation'. If the dead time compensation is enabled (via the system parameters) the timer is stopped as long as the TAC is busy to convert a start/stop event. Thus the timer runs for the same time in which the system is able to accept a new photon. With increasing count rate, the collection time interval increases by an amount which compensates the loss of count rate due to the TAC dead time.

If the dead time compensation is disabled the collection time interval is independent of the count rate. If the sequencer is enabled the dead time compensation is switched off automatically.

The desired collection time interval is loaded with `SPC_init` or with a call of the functions `SPC_set_parameters` or `SPC_set_parameter`. The control of the timer is managed by the `SPC_start_measurement` function so that no explicit load/start/stop operations are required.

When the programmed collection time has expired, the measurement is stopped automatically if `stop_on_time` has been set (system parameters). The status can be read by the function `SPC_test_state` (see also 'Measurement Functions' and `spcm_def.h`).

The residual collection time to the end of the measurement can be determined by the function **`SPC_get_actual_coltime`**.

The repeat timer is independent of the system count rate. It is used to control measurement sequences such as the  $f(t,T)$  mode in the standard software. The time from the start of a measurement is returned by the function **`SPC_get_time_from_start`**. The function `SPC_test_state` returns a status bit which indicates the expiration of the repeat counter.

For collection times longer than 80 seconds `SPC_test_state` updates also DLL software counters (hardware timers count up to 80 sec). Therefore during the measurement `SPC_get_actual_coltime` or `SPC_get_time_from_start` calls must be done after `SPC_test_state` call.

When the sequencer is enabled the repeat timer is not available.

## Error Handling

Each SPC DLL function returns an error status. Return values  $\geq 0$  indicate error free execution. A value  $< 0$  indicates that an error occurred during execution. The meaning of a particular error code can be found in `spcm_def.h` file and can be read using `SPC_get_error_string`. We recommend to check the return value after each function call.

## Using DLL functions in LabView environment

Each DLL function can be called in LabView program by using 'Call Library' function node. If you select Configure from the shortcut menu of the node, you see a Call Library Function dialog box from which you can specify the library name or path, function name, calling conventions, parameters, and return value for the node.

You should pay special attention to choosing correct parameter types using following conversion rules:

Type in C programs	Type in LabView
char	signed 8-bit integer, byte ( I8)
unsigned char	unsigned 8-bit integer, unsigned byte ( U8)
short	signed 16-bit integer, word ( I16)
unsigned short	unsigned 16-bit integer, unsigned word ( U16)
long, int	signed 32-bit integer, long ( I32)
unsigned long, int	unsigned 32-bit integer, unsigned long ( U32)
float	4-byte single, single precision ( SGL)
double	8-byte double, double precision ( DBL)
char *	C string pointer
float *	Pointer to Value ( Numeric, 4-byte single)

For structures defined in include file `spcm_def.h` user should build in LabView a proper cluster. The cluster must contain the same fields in the same order as the C structure.

If a pointer to a structure is a function parameter, you connect to the node the proper cluster and define parameter type as 'Adapt to Type' (with data format = 'Handles by Value').

Connecting clusters with the contents which do not exactly correspond to the C structure fields can cause the program crash.

Problems appear if the **structure and the corresponding cluster contain string fields** - due to the fact that LabView sends to the DLL handles to LabView string instead of the C string pointers for strings inside the cluster.

In such case special version of the DLL function must be used which is prepared especially for use in LabView. Such functions have '\_LV' letters after 'SPC' ( for example `SPC_LV_get_eeprom_data` ), and if found in `spcm_def.h` file they should be used in 'Call Library' function node instead of the standard function.

Another solution is to write extra C code to transform these data types, create `.lsb` file and use it in 'Code Interface' node (CIN) instead of 'Call Library'.

Experienced LabView and C users can prepare such CINs for every external code.



## Description of the SPC DLL Functions

### Initialisation functions:

```
-----  
short CVICDECL SPC_init (char * ini_file);  
-----
```

#### Input parameters:

- \* ini\_file: pointer to a string containing the name of the initialisation file in use (including file name and extension)

#### Return value:

0 no errors, <0 error code

#### Description:

Before a measurement is started the measurement parameter values must be written into the internal structures of the DLL functions (not directly visible from the user program) and sent to the control registers of the SPC module(s). This is accomplished by the function **SPC\_init**. The function

- checks whether DLL is correctly registered ( looks for a BH license number and verifies it)
- reads the parameter values from the specified file ini\_file
- checks and recalculates the parameters depending on hardware restrictions and adjust parameters from the EEPROM on the SPC module(s)
- sends the parameter values to the internal structures of the DLL
- sends the parameter values to the SPC control registers
- performs a hardware test of the SPC module(s)

The SPC module, which will be initialised, is defined by 'pci\_bus\_no' and 'pci\_card\_no' parameter from ini\_file.

'pci\_bus\_no' defines which PCI bus with SPC modules will be initialised:

- value 0 – 255 defines specific bus number (from the range: 0 to number of PCI busses with SPC modules) ( if 'pci\_bus\_no' is greater than number of PCI busses with SPC modules, it is rounded to the number of busses –1 )
- value –1 means that that the function will try to initialise SPC modules on all PCI busses

'pci\_card\_no' defines the number of SPC module on PCI bus to be initialised:

- value 0 – 7 defines one specific module ( if 'pci\_card\_no' is greater than number of SPC modules on PCI bus, it is rounded to the number of modules –1 )
- value –1 means that that the function will try to initialise all SPC modules on PCI bus

The module will be initialised, but only when it is not in use (locked) by other application.

After successful initialisation the module is locked to prevent that other application can access it. The user should check initialisation status of all modules he wants to use by calling SPC\_get\_init\_status function.

If, for some reasons, the module which was locked must be initialised, it can be done using the function SPC\_set\_mode with the parameter 'force\_use' = 1.

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file spcm.ini or to start with spcm.ini and introduce the desired changes.

```
; SPCM DLL initialisation file for SPC modules
; SPC parameters have to be included in .ini file only when parameter
; value is different from default.
; for DPC230 module use file dpc230.ini instead of this one

[spc_base]
simulation = 0                ; 0 - hardware mode(default) ,
                             ; >0 - simulation mode (see spcm_def.h for possible values)
pci_bus_no= -1               ; PCI bus on which SPC modules will be looking for
                             ; 0 - 255, default -1 ( all PCI busses will be scanned)
pci_card_no= -1              ; number of the SPC module on PCI bus
                             ; 0 - 7, default -1 (all modules on PCI bus)

[spc_module]
cfd_limit_low= 5.0           ; SPC hardware parameters
                             ; for SPCx3x (140,150) -500 .. 0mV ,for SPCx0x 5 .. 80mV
                             ; default 5mV
cfd_limit_high= 80.0         ; 5 ..80 mV, default 80 mV, not for SPC130,140,150,930
cfd_zc_level= 0.0            ;for SPCx3x(140,150) -96 .. 96mV ,for SPCx0x -10 .. 10mV
                             ; default 0mV
cfd_holdoff= 5.0            ; for SPCx0x 5 .. 20 ns , default 5ns
                             ; for other modules doesn't exist
sync_zc_level= 0.0           ; for SPCx3x(140,150) -96 .. 96mV ,for SPCx0x -10 .. 10mV
                             ; default 0mV
sync_freq_div= 4             ; for SPC130,140,150,930 1,2,4
                             ; for other SPC modules 1,2,4,8,16 , default 4
sync_holdoff= 4.0           ; 4 .. 16 ns , default 4 ns, for SPC130,140,150,930 doesn't exist
sync_threshold= -20.0        ; for SPCx3x(140,150) -500 .. -20mV ,default -20 mV
                             ; for SPCx0x doesn't exist

tac_range= 50.0              ; 50 .. 5000 ns , default 50 ns
tac_gain= 1                  ; 1 .. 15 ,default 1
tac_offset=0.0               ; 0 .. 100% ,default 0%
tac_limit_low= 10.0          ; 0 .. 100% ,default 10%
tac_limit_high= 80.0         ; 0 .. 100% ,default 80%

adc_resolution= 10           ; 6,8,10,12 bits, default 10
                             ; (additionally 0,2,4 bits for SPC830,140,150,930 )
extLatchDelay= 0             ; 0 ..255 ns, default 0, for SPC130 doesn't exist
                             ; for SPC140,150,930 only values 0,10,20,30,40,50 ns are possible

collect_time= 0.01           ; 0.0001 .. 100000s , default 0.01s
repeat_time= 10.0            ; 0.0001 .. 100000s , default 10.0s
stop_on_time= 1              ; 0,1 , default 1
stop_on_ovfl= 1              ; 0,1 , default 1
dither_range= 0              ; possible values - 0, 32, 64, 128, 256
                             ; have meaning: 0, 1/64, 1/32, 1/16, 1/8

count_incr= 1                ; 1 .. 255 , default 1
mem_bank= 0                  ; for SPC130,600,630, 150 : 0 , 1 , default 0
                             ; for other SPC modules always 0

dead_time_comp= 1            ; 0 , 1 , default 1
mode= 0                       ; for SPC7x0 , default 0
                             ; 0 - normal operation (routing in),
                             ; 1 - block address out, 2 - Scan In, 3 - Scan Out
                             ; for SPC6x0 , default 0
                             ; 0 - normal operation (routing in),
                             ; 2 - FIFO mode 48 bits, 3 - FIFO mode 32 bits
                             ; for SPC130 , default 0
                             ; 0 - normal operation (routing in),
```



```

rate_count_time= 1.0          ; in Scan In mode 1 .. 0x3ff, default 1
                              ; rate counting time in sec default 1.0 sec
                              ;   for SPC130,150, 830, 930 can be : 1.0s, 0.25s, 0.1s, 0.05s
                              ;   for SPC140 fixed to 50ms

macro_time_clk= 0             ; macro time clock definition for SPC130, 140,150, 830, 930 in FIFO mode
                              ; for SPC130, 140:
                              ; 0 - 50ns (default), 25ns for SPC150 & 140 with FPGA v. > B0 ,
                              ;   1 - SYNC freq., 2 - 1/2 SYNC freq.,
                              ;   3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.
                              ; for SPC830: 0 - 50ns (default), 1 - SYNC freq.,
                              ; for SPC930: 0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.

add_select= 0                 ; selects ADD signal source for all modules except SPC930:
                              ;   0 - internal (ADD only) (default), 1 - external

adc_zoom = 0                  ; ADC zoom level for module SPC830, 140,150, 930 default 0
                              ; bit 4 = 0(1) - zoom off(on ),
                              ; bits 0-3 zoom level,
                              ; 0 - zoom of the 1st 1/16th of ADC range,
                              ; 15 - zoom of the 16th 1/16th of ADC range

img_size_x = 1                ; image X size ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
                              ; 1 .. 1024, default 1

img_size_y = 1                ; image Y size ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
                              ; actually equal to img_size_x ( quadratic image )

img_rout_x = 1                ; no of X routing channels ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
                              ; 1 .. 16, default 1

img_rout_y = 1                ; no of Y routing channels ( SPC140,150, 830 in FIFO_32M mode, SPC930 in Camera mode ),
                              ; 1 .. 16, default 1

xy_gain = 1                   ; selects gain for XY ADCs for module SPC930, 1,2,4 , default 1
master_clock = 0              ; use Master Clock( 1 ) or not ( 0 ), default 0,
                              ; only for SPC140 ,150 multi-module configuration
                              ; - value 2 (when read) means Master Clock state was set
                              ; by other application and cannot be changed

adc_sample_delay = 0          ; ADC's sample delay, only for module SPC930
                              ; 0,10,20,30,40,50 ns (default 0 )

detector_type = 1             ; detector type used in Camera mode, only for module SPC930,
                              ; 1 .. 9899, default 1
                              ; normally it is recognised automatically from the corresponding .bit file
                              ; 1 - Hamamatsu Resistive Anode 4 channels detector
                              ; 2 - Wedge & Strip 3 channels detector

x_axis_type = 0               ; X axis representation, only for module SPC930
                              ; 0 - time (default), 1 - ADC1 Voltage,
                              ; 2 - ADC2 Voltage, 3 - ADC3 Voltage, 4 - ADC4 Voltage

```

---

```
short CVICDECL SPC_get_init_status(short mod_no);
```

---

Input parameters:

mod\_no                    0 .. 7, SPC module number

Return value: initialisation result code of the SPC module 'mod\_no'

Description:

The procedure returns the initialisation result code set by the function SPC\_init. The possible values are shown below (see also spcm\_def.h):

INIT_OK	0	no error
INIT_NOT_DONE	-1	init not done
INIT_WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_WRONG_MOD_ID	-3	wrong module identification code
INIT_HARD_TEST_ERR	-4	hardware test failed
INIT_CANT_OPEN_PCI_CARD	-5	cannot open PCI card
INIT_MOD_IN_USE	-6	module in use (locked) - cannot initialise
INIT_WINDRVR_VER	-7	incorrect WinDriver version

INIT_WRONG_LICENSE	-8	wrong or missing license key
INIT_XILINX_ERR	-1xx	Xilinx chip configuration error - where xx = Xilinx error code

---

```
short CVICDECL SPC_get_module_info(short mod_no, SPCModInfo * mod_info);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
*mod_info	pointer to result structure (type SPCModInfo)

Return value: 0 no errors, <0 error code (see spcm\_def.h)

Description:

After calling the SPC\_init function (see above) the SPCModInfo internal structures for all 8 modules are filled. This function transfers the contents of the internal structure of the DLL into a structure of the type SPCModInfo (see spcm\_def.h) which has to be defined by the user. The parameters included in this structure are described below.

short module_type	SPC module type (see spcm_def.h)
short bus_number	PCI bus number of the module
short slot_number	slot number on PCI bus 'bus_number' occupied by the module
short in_use	-1 used and locked by other application, 0 - not used, 1 - in use
short init	set to initialisation result code
unsigned short base_adr	base I/O address on PCI bus

---

```
short CVICDECL SPC_test_id(short mod_no) ;
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
--------	---------------------------

Return value: on success - module type, on error <0 (error code)

The procedure can be used to check whether an SPC module is present and which type of the module it is. It is a low level procedure that is called also during the initialisation in SPC\_init. The procedure returns a module type value of module 'mod\_no'. Possible module type values are defined in spcm\_def.h file.

The x0x and x3x versions are not distinguished by the id but by the EEPROM data and the function SPC\_init. SPC\_test\_id will return the correct values only if SPC\_init has been called.

```

/* supported SPC module types - returned value from SPC_test_id */
#define M_SPC600      600      PCI version of 400
#define M_SPC630      630      PCI version of 430
#define M_SPC700      700      PCI version of 500
#define M_SPC730      730      PCI version of 530
#define M_SPC130      130      PCI special version of 630
#define M_SPC830      830      version of 730 with large memory, Fifo mode
#define M_SPC140      140      130 with large memory, Fifo mode, Scan modes
#define M_SPC930      930      830 with Camera mode
#define M_SPC150      150      140 with Fifo mode, Scan modes, Fifo Imaging, Cont. Flow

```

---

```
short CVICDECL SPC_set_mode(short mode, short force_use, int *in_use);
```

---

Input parameters:

mode	mode of DLL operation
force_use	force using the module if they are locked ( in use)
*in_use	pointer to the table with information which module must be used

Return value: on success - DLL mode, on error <0 (error code)

The procedure is used to change the mode of the DLL operation between the hardware mode and the simulation mode. It is also used to switch the DLL to the simulation mode if hardware errors occur during the initialisation.

Table 'in\_use' should contain entries for all 8 modules:

0 – means that the module will be unlocked and not used longer

1 – means that the module will be initialised and locked

When the Hardware Mode is requested for each of 8 possible modules:

-if 'in\_use' entry = 1 : the proper module is locked and initialised (if it wasn't) with the initial parameters set (from ini\_file) but only when it was not locked by another application or when 'force\_use' = 1.

-if 'in\_use' entry = 0 : the proper module is unlocked and cannot be used further.

When one of the simulation modes is requested for each of 8 possible modules:

-if 'in\_use' entry = 1 : the proper module is initialised (if it wasn't) with the initial parameters set (from ini\_file).

-if 'in\_use' entry = 0 : the proper module is unlocked and cannot be used further.

Errors during the module initialisation can cause that the module is excluded from use.

Use the function SPC\_get\_init\_status and/or SPC\_get\_module\_info to check which modules are correctly initialised and can be use further.

Use the function SPC\_get\_mode to check which mode is actually set. Possible 'mode' values are defined in the spcm\_def.h file.

---

```
short CVICDECL SPC_get_mode(void);
```

---

Input parameters: none

Return value: DLL operation mode

The procedure returns the current DLL operation mode. Possible 'mode' values are defined in the spcm\_def.h file:

```
#define SPC_HARD 0 /* hardware mode */
```

```

#define SPC_SIMUL600      600      /* simulation mode of SPC600 module */
#define SPC_SIMUL630      630      /* simulation mode of SPC630 module */
#define SPC_SIMUL700      700      /* simulation mode of SPC700 module */
#define SPC_SIMUL730      730      /* simulation mode of SPC730 module */
#define SPC_SIMUL130     130      /* simulation mode of SPC130 module */
#define SPC_SIMUL830      830      /* simulation mode of SPC830 module */
#define SPC_SIMUL140     140      /* simulation mode of SPC140 module */
#define SPC_SIMUL930     930      /* simulation mode of SPC930 module */
#define SPC_SIMUL150     150      /* simulation mode of SPC150 module */
#define DPC_SIMUL230     230      /* simulation mode of DPC230 module */

```

## Setup functions:

```

-----
short CVICDECL SPC_get_parameters(short mod_no, SPCdata * data);
-----

```

### Input parameters:

**mod\_no**                    0 .. 7, SPC module number  
**\*data**                    pointer to result structure (type SPCdata)

Return value: 0 no errors, <0 error code (see spcm\_def.h)

### Description:

After calling the `SPC_init` function (see above) the measurement parameters from the initialisation file are present in the module and in the internal data structures of the DLLs. To give the user access to the parameters, the function **SPC\_get\_parameters** is provided. This function transfers the parameter values from the internal DLL structures of the module 'mod\_no' into a structure of the type `SPCdata` (see `spcm_def.h`) which has to be defined by the user. The parameter values in this structure are described below.

unsigned short base_adr	base I/O address on PCI bus
short init	set to initialisation result code
float cfd_limit_low	SPCx3x(140,150) -500 .. 0mV ,for SPCx0x 5 .. 80mV
float cfd_limit_high	5 ..80 mV, default 80 mV , not for SPC130,140,150,930
float cfd_zc_level	SPCx3x(140,150) -96 .. 96mV, SPCx0x -10 .. 10mV
float cfd_holdoff	SPCx0x: 5 .. 20 ns, other modules: no influence
float sync_zc_level	SPCx3x(140,150): -96 .. 96mV, SPCx0x: -10..10mV
float sync_holdoff	4 .. 16 ns ( SPC130,140,150,930 : no influence)
float sync_threshold	SPCx3x(140,150): -500 .. -20mV, SPCx0x: no influence
float tac_range	50 .. 5000 ns
short sync_freq_div	1,2,4,8,16 ( SPC130,140,150,930 : 1,2,4)
short tac_gain	1 .. 15
float tac_offset	0 .. 100%
float tac_limit_low	0 .. 100%
float tac_limit_high	0 .. 100%
short adc_resolution	6,8,10,12 bits, default 10 (additionally 0,2,4 bits for SPC830, 140,150, 930 )
short ext_latch_delay	0 ..255 ns, SPC130: no influence SPC140,150, 930 : only values 0,10,20,30,40,50 ns are possible
float collect_time	0.0001 .. 100000s , default 0.01s
float display_time	0.0001 .. 100000s , default 0.01s
float repeat_time	0.0001 .. 100000s , default 0.01s

short stop\_on\_time 1 (stop) or 0 (no stop)  
short stop\_on\_ovfl 1 (stop) or 0 (no stop)  
short dither\_range possible values - 0, 32, 64, 128, 256  
have meaning: 0, 1/64, 1/32, 1/16, 1/8  
short count\_incr 1 .. 255  
short mem\_bank for SPC130,600,630,150 : 0, 1, default 0  
other SPC modules: always 0  
short dead\_time\_comp 0 (off) or 1 (on)  
unsigned short scan\_control SPC505(535,506,536) scanning(routing) control word  
other SPC modules always 0  
short routing\_mode SPC150(140) - bits 8 - 11 - enable(1)/disable(0), default 0  
of recording Markers 0-3 entries in FIFO mode  
other SPC modules not used  
float tac\_enable\_hold SPC230 10.0 .. 265.0 ns - duration of  
TAC enable pulse ,other SPC modules always 0  
short pci\_card\_no module no on PCI bus (0-7)  
unsigned short mode; for SPC7x0 , default 0  
0 - normal operation (routing in),  
1 - block address out, 2 - Scan In, 3 - Scan Out  
for SPC6x0 , default 0  
0 - normal operation (routing in)  
2 - FIFO mode 48 bits, 3 - FIFO mode 32 bits  
for SPC130 , default 0  
0 - normal operation (routing in)  
2 - FIFO mode  
for SPC140 , default 0  
0 - normal operation (routing in)  
1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out  
5 - FIFO\_mode 32 bits with markers ( FIFO\_32M ), with FPGA v. > B0  
for SPC150 , default 0  
0 - normal operation (routing in)  
1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out  
5 - FIFO\_mode 32 bits with markers ( FIFO\_32M )  
for SPC830,930 , default 0  
0 - normal operation (routing in)  
1 - FIFO mode 32 bits, 2 - Scan In, 3 - Scan Out  
4 - Camera mode ( only SPC930 )  
5 - FIFO\_mode 32 bits with markers ( FIFO\_32M ), SPC830 with FPGA v. > B0  
unsigned long scan\_size\_x; for SPC7x0,830,140,150,930 modules in scanning modes 1 .. 65536, default 1  
unsigned long scan\_size\_y; for SPC7x0,830,140,150,930 modules in scanning modes 1 .. 65536, default 1  
unsigned long scan\_rout\_x; number of X routing channels in Scan In & Scan Out modes  
for SPC7x0,830,140,150,930 modules  
1 .. 128, ( SPC7x0,830 ), 1 .. 16 (SPC140,150,930), default 1  
unsigned long scan\_rout\_y; number of Y routing channels in Scan In & Scan Out modes  
for SPC7x0,830,140,150,930 modules  
1 .. 128, ( SPC7x0,830 ), 1 .. 16 (SPC140,150,930), default 1  
 $\text{INT}(\log_2(\text{scan\_size\_x})) + \text{INT}(\log_2(\text{scan\_size\_y})) +$   
 $\text{INT}(\log_2(\text{scan\_rout\_x})) + \text{INT}(\log_2(\text{scan\_rout\_y})) \leq$  max number of scanning bits  
max number of scanning bits depends on current adc\_resolution:  
12 (10 for SPC7x0,140,150) - 12  
14 (12 for SPC7x0,140,150) - 10  
16 (14 for SPC7x0,140,150) - 8  
18 (16 for SPC7x0,140,150) - 6  
20 (18 for SPC140,150) - 4  
22 (20 for SPC140,150) - 2  
24 (22 for SPC140,150) - 0  
unsigned long scan\_flyback; for SPC7x0,830,140,150,930 modules in Scan Out or Rout Out mode, default 0  
bits 15-0 Flyback X in number of pixels  
bits 31-16 Flyback Y in number of lines  
unsigned long scan\_borders; for SPC7x0,830,140,150,930 modules in Scan In mode, default 0  
bits 15-0 Upper boarder, bits 31-16 Left boarder  
unsigned short scan\_polarity; for SPC7x0,830,140,150,930 modules in scanning modes, default 0  
bit 0 - polarity of HSYNC (Line), bit 1 - polarity of VSYNC (Frame),  
bit 2 - pixel clock polarity  
bit = 0 - falling edge(active low)  
bit = 1 - rising edge(active high)  
for SPC140,150,830 in FIFO\_32M mode  
bit = 8 - HSYNC (Line) marker disabled (1) or enabled (0, default )  
when disabled, line marker will not appear in FIFO photons stream  
unsigned short pixel\_clock; for SPC7x0,830,140,150,930 modules in Scan In mode,  
pixel clock source, 0 - internal, 1 - external, default 0  
for SPC140,150,830 in FIFO\_32M mode it disables/enables pixel markers  
in photons stream

unsigned short line\_compression; line compression factor for SPC7x0,830,140,150,930 modules in Scan In mode, 1,2,4,8,16,32,64,128, default 1

unsigned short trigger; external trigger condition -  
bits 1 & 0 mean : 00 - ( value 0 ) none(default),  
01 - ( value 1 ) active low,  
10 - ( value 2 ) active high  
when sequencer is enabled on SPC130(6x0) modules additionally  
bits 9 & 8 of the value mean:  
00 - trigger only at the start of the sequence,  
01 ( 100 hex, 256 decimal ) - trigger on each bank  
11 ( 300 hex, 768 decimal ) - trigger on each curve in the bank  
for SPC140,150 and SPC130 (FPGA v. > C0) multi-module configuration  
bits 9 & 8 of the value mean:  
x0 - module does not use trigger bus ( trigger defined via bits 0-1),  
01 ( 1000 hex, 4096 decimal ) - module uses trigger bus as slave  
( waits for the trigger on master),  
11 ( 3000 hex, 12288 decimal ) - module uses trigger bus as master  
( trigger defined via bits 0-1),  
( only one module can be the master )

float pixel\_time; pixel time in sec for SPC7x0,830,140,150,930 modules in Scan In mode,  
50e-9 .. 1.0 , default 200e-9

unsigned long ext\_pixclk\_div; divider of external pixel clock for SPC7x0,830,140,150,930 modules  
in Scan In mode, 1 .. 0x3fe, default 1

float rate\_count\_time; rate counting time in sec default 1.0 sec  
for SPC130,830,930,150 can be : 1.0s, 0.25s, 0.1s, 0.05s  
for SPC140 fixed to 50ms

short macro\_time\_clk; macro time clock definition for SPC130,140,150,830,930 in FIFO mode  
for SPC130, 140,150:  
0 - 50ns (default), 25ns for SPC150 & 140 with FPGA v. > B0 ,  
1 - SYNC freq., 2 - 1/2 SYNC freq.,  
3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.  
for SPC830: 0 - 50ns (default), 1 - SYNC freq.,  
for SPC930: 0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.,

short add\_select; selects ADD signal source for all modules except SPC930 :  
0 - internal (ADD only) (default), 1 - external

short test\_eeep 0: EEPROM is not read and not tested, default adjust parameters are used  
1: EEPROM is read and tested for correct checksum

short adc\_zoom; selects ADC zoom level for module SPC830,140,150,930 default 0  
bit 4 = 0(1) - zoom off(on),  
bits 0-3 zoom level =  
0 - zoom of the 1st 1/16th of ADC range,  
15 - zoom of the 16th 1/16th of ADC range

unsigned long img\_size\_x; image X size ( SPC140,150,830 in FIFO\_32M mode, SPC930 in Camera mode ),  
1 .. 1024, default 1

unsigned long img\_size\_y; image Y size ( SPC140,150,830 in FIFO\_32M mode, SPC930 in Camera mode ),  
actually equal to img\_size\_x ( quadratic image )

unsigned long img\_rout\_x; no of X routing channels ( SPC140,150,830 in FIFO\_32M mode, SPC930 in Camera mode ),  
1 .. 16, default 1

unsigned long img\_rout\_y; no of Y routing channels ( SPC140,150,830 in FIFO\_32M mode, SPC930 in Camera mode ),  
1 .. 16, default 1

short xy\_gain; selects gain for XY ADCs for module SPC930, 1,2,4, default 1

short master\_clock; use Master Clock( 1 ) or not ( 0 ), default 0,  
only for SPC140, 150 multi-module configuration  
- value 2 (when read) means Master Clock state was set  
by other application and cannot be changed

short adc\_sample\_delay; ADC's sample delay, only for module SPC930  
0,10,20,30,40,50 ns (default 0)

short detector\_type; detector type used in Camera mode, only for module SPC930  
1 .. 9899, default 1,  
normally recognised automatically from the corresponding .bit file  
1 - Hamamatsu Resistive Anode 4 channels detector  
2 - Wedge & Strip 3 channels detector

short x\_axis\_type; X axis representation, only for module SPC930  
0 - time (default ), 1 - ADC1 Voltage,  
2 - ADC2 Voltage, 3 - ADC3 Voltage, 4 - ADC4 Voltage

unsigned short chan\_enable; for module DPC230 - enable(1)/disable(0) input channels  
bits 0-7 - en/disable TTL channel 0-7 in TDC1  
bits 8-9 - en/disable CFD channel 0-1 in TDC1  
bits 12-19 - en/disable TTL channel 0-7 in TDC2  
bits 20-21 - en/disable CFD channel 0-1 in TDC2

unsigned short chan\_slope; for module DPC230 - active slope of input channels  
1 - rising, 0 - falling edge active  
bits 0-7 - slope of TTL channel 0-7 in TDC1  
bits 8-9 - slope of CFD channel 0-1 in TDC1

bits 12-19 - slope of TTL channel 0-7 in TDC2  
 bits 20-21 - slope of CFD channel 0-1 in TDC2  
 unsigned long chan\_spec\_no; for module DPC230 - channel numbers of special inputs, default 0x8813  
 bits 0-4 - reference chan. no ( TCSPC and Multiscaler modes)  
 default = 19, value:  
 0-1 CFD chan. 0-1 of TDC1, 2-9 TTL chan. 0-7 of TDC1  
 10-11 CFD chan. 0-1 of TDC2, 12-19 TTL chan. 0-7 of TDC2  
 bits 8-10 - frame clock TTL chan. no ( imaging modes ) 0-7, default 0  
 bits 11-13 - line clock TTL chan. no ( imaging modes ) 0-7, default 1  
 bits 14-16 - pixel clock TTL chan. no ( imaging modes ) 0-7, default 2  
 bit 17 - TDC no for pixel, line, frame clocks ( imaging modes )  
 0 = TDC1, 1 = TDC2, default 0  
 bits 18-31 - not used

---

```
short CVICDECL SPC_set_parameters(short mod_no, SPCdata *data);
```

---

Input parameters:

mod\_no            0 .. 7, SPC module number  
 \*data            pointer to result structure (type SPCdata)

Return value: 0 no errors, <0 error code (see spcm\_def.h)

Description:

The procedure sends all parameters from the 'SPCdata' structure to the internal DLL structures of the module 'mod\_no' and to the control registers of the SPC module 'mod\_no'.

The new parameter values are recalculated according to the parameter limits, hardware restrictions (e.g. DAC resolution) and the SPC module type. Furthermore, cross dependencies between different parameters are taken into account to ensure the correct hardware operation. It is recommended to read back the parameters after setting it to get their real values after recalculation.

If an error occurs at a particular parameter, the procedure does not set the rest of the parameters and returns with an error code.

---

```
short CVICDECL SPC_get_parameter(short mod_no, short par_id, float * value);
```

---

Input parameters:

mod\_no            0 .. 7, SPC module number  
 par\_id            parameter identification number (see spcm\_def.h)  
 \*value            pointer to the parameter value

Return value:

0 no errors, <0 error code

The procedure loads 'value' with the actual value of the requested parameter from the internal DLL structures of the module 'mod\_no'. The par\_id values are defined in spcm\_def.h file as SPC\_PARAMETERS\_KEYWORDS.

---

```
short CVICDECL SPC_set_parameter(short mod_no, short par_id, float value);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number, -1 all used modules
par_id	parameter identification number
value	new parameter value

Return value:

0 no errors, <0 error code

The procedure sets the specified hardware parameter. The value of the specified parameter is transferred to the internal DLL structures of the module 'mod\_no' and to the SPC module 'mod\_no'.

If 'mod\_no' = -1, parameter will be changed for all SPC modules which are actually in use.

The new parameter value is recalculated according to the parameter limits, hardware restrictions (e.g. DAC resolution) and SPC module type. Furthermore, cross dependencies between different parameters are taken into account to ensure the correct hardware operation. It is recommended to read back the parameters after setting it to get their real values after recalculation.

The par\_id values are defined in spcm\_def.h file as SPC\_PARAMETERS\_KEYWORDS.

---

```
short CVICDECL SPC_get_eeeprom_data( short mod_no, SPC_EEP_Data *eep_data);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
*eep_data	pointer to result structure

Return value: 0: no errors, <0: error code

The structure "eep\_data" is filled with the contents of EEPROM of SPC module 'mod\_no'. The EEPROM contains production data and adjust parameters of the module. The structure "SPC\_EEP\_Data" is defined in the file spcm\_def.h.

Normally, the EEPROM data need not be read explicitly because the EEPROM is read during SPC\_init and the module type information and the adjust values are taken into account when the SPC module registers are loaded.

```
short CVICDECL SPC_write_eeprom_data (short mod_no , unsigned short write_enable,  
                                     SPC_EEP_Data *eep_data);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
write_enable	write enable value (known by B&H)
*eep_data	pointer to a structure which will be sent to EEPROM

Return value: 0: no errors, <0: error code

The function is used to write data to the EEPROM of an SPC module 'mod\_no' by the manufacturer. To prevent corruption of the adjust data writing to the EEPROM can be executed only when correct 'write\_enable' value is used.

---

```
short CVICDECL SPC_get_adjust_parameters (short mod_no, SPC_Adjust_Para * adjpara);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
*adjpara	pointer to result structure

Return value: 0: no errors, <0: error code

The structure 'adjpara' is filled with adjust parameters of the SPC module 'mod\_no' that are currently in use. The parameters can either be previously loaded from the EEPROM by SPC\_init or SPC\_get\_eeprom\_data or - not recommended - set by SPC\_set\_adust\_parameters.

The structure "SPC\_Adjust\_Para" is defined in the file spcm\_def.h.

Normally, the adjust parameters need not be read explicitly because the EEPROM is read during SPC\_init and the adjust values are taken into account when the SPC module registers are loaded.

---

```
short CVICDECL SPC_set_adjust_parameters (short mod_no, SPC_Adjust_Para *adjpara);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
*adjpara	pointer to a structure which contains new adjust parameters

Return value: 0: no errors, <0: error code

The adjust parameters in the internal DLL structures (not in the EEPROM) of the module 'mod\_no' are set to values from the structure "adjpara". The function is used to set the module adjust parameters to values other than the values from the EEPROM. The new adjust values will be used until the next call of SPC\_init. The next call to SPC\_init replaces the adjust parameters by the values from the EEPROM. We strongly discourage to use modified adjust parameters, because the module function can be seriously corrupted by wrong adjust values.

The structure "SPC\_Adjust\_Para" is defined in the file spcm\_def.h.

```
-----  
short CVICDECL SPC_read_parameters_from_inifile ( SPCdata *data, char *inifile);  
-----
```

Input parameters:

\*data            pointer to result structure (type SPCdata)  
\*inifile:        pointer to a string containing the name of the initialisation file (including  
                 file name and extension)

Return value:    0 no errors, <0 error code (see spcm\_def.h)

Description:

The procedure reads parameters from the file 'inifile' and transfers them to the 'SPCdata' structure 'data'.

The 'inifile' file is an ASCII file with a structure shown in SPC\_init description. We recommend to use either the original .ini files or the files created using function SPC\_save\_parameters\_to\_inifile.

If a particular parameter is not present in .ini file or cannot be read, the appropriate field in SPCdata 'data' structure is set to the parameter's default value.

Use SPC\_set\_parameters to send result parameters set to the SPC module.

```
-----  
short CVICDECL SPC_save_parameters_to_inifile ( SPCdata *data, char *dest_inifile,  
                                                 char *source_inifile, int with_comments);  
-----
```

Input parameters:

\*data            pointer to result structure (type SPCdata)  
\*dest\_inifile:   pointer to a string containing the name of the destination file  
\*source\_inifile: pointer to a string containing the name of the source ini file, can be NULL  
with\_comments    0 or 1 says whether comments from source\_inifile will be copied to  
                 dest\_inifile

Return value:    0 no errors, <0 error code (see spcm\_def.h)

Description:

The parameters set from the 'SPCdata' structure 'data' is saved to the section [spc\_module] in dest\_inifile file. [spc\_base] section and initial comment lines are copied to dest\_inifile from the source\_inifile file.

If the parameter 'source\_inifile' is equal NULL, ini\_file used in SPC\_init function call is used as the source file for dest\_inifile.

Additionally when 'with\_comments' parameter is equal 1, comment lines for the particular parameters are taken from the source file and saved to dest\_inifile together with the parameter value.

The 'dest\_inifile' and 'source\_inifile' files are ASCII files with a structure shown in SPC\_init description.

Use SPC\_read\_parameters\_from\_inifile to read back the parameters set from the file and then SPC\_set\_parameters to send it to the SPC module.

### Status functions:

-----  
short CVICDECL SPC\_test\_state(short mod\_no, short \*state);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
*state	pointer to result variable

Return value: 0: no errors, <0: error code

SPC\_test\_state sets a state variable according to the current state of the measurement on SPC module 'mod\_no'. The function is used to control the measurement loop. The status bits delivered by the function are listed below (see also SPCM\_DEF.H).

SPC_OVERFL	0x1	stopped on overflow
SPC_OVERFLOW	0x2	overflow occurred
SPC_TIME_OVER	0x4	stopped on expiration of collection timer
SPC_COLTIM_OVER	0x8	collection timer expired
SPC_CMD_STOP	0x10	stopped on user command
SPC_ARMED	0x80	measurement in progress (current bank)
SPC_REPTIM_OVER	0x20	repeat timer expired
SPC_COLTIM_2OVER	0x100	second overflow of collection timer
SPC_REPTIM_2OVER	0x200	second overflow of repeat timer

For SPC600(630) and SPC130 modules only :

SPC_SEQ_GAP	0x40	Sequencer is waiting for other bank to be armed
-------------	------	---

For SPC600(630)(130)(140)(830)(150) and SPC930 modules only :

SPC_FOVFL	0x400	Fifo overflow, data lost
SPC_FEMPTY	0x800	Fifo empty

For SPC700(730)(140)(830)(150) and SPC930 modules only:

SPC_SCRDY	0x400	Scan ready (data can be read )
SPC_FBRDY	0x800	Flow back of scan finished
SPC_MEASURE	0x40	Measurement active : no margin, no wait for trigger, armed

SPC_WAIT_TRG	0x1000	Wait for external trigger
SPC_HFILL_NRDY	0x8000	hardware fill not finished

For SPC140 (150) and SPC930 modules only:

SPC_SEQ_STOP	0x4000	disarmed (measurement stopped) by sequencer
--------------	--------	---

For SPC150 modules only:

SPC_SEQ_GAP150	0x2000	Sequencer is waiting for other bank to be armed
----------------	--------	---

For SPC140 (150) and SPC830 modules in FIFO\_32M mode

SPC_WAIT_FR	0x2000	FIFO IMAGE measurement waits for the frame signal to stop
-------------	--------	---

For collection times longer than 80 seconds SPC\_test\_state updates also DLL software counters (hardware timers count up to 80 sec). Therefore during the measurement SPC\_get\_actual\_coltime or SPC\_get\_time\_from\_start calls must be done after SPC\_test\_state call.

-----  
short CVICDECL SPC\_get\_sync\_state(short mod\_no, short \*sync\_state);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
*sync_state	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure sets "sync\_state" according to the actual sync state on the SPC module 'mod\_no'.

For SPC-130(140)(930) module possible values are:

0:	SYNC NOT OK, sync input not triggered
1:	SYNC OK, sync input triggers

For other SPC modules possible values are:

0:	NO SYNC, sync input not triggered
1:	SYNC OK, sync input triggers
2, 3:	SYNC OVERLOAD.

-----  
short CVICDECL SPC\_get\_time\_from\_start(short mod\_no, float \*time);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
*time	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure reads the SPC repeat timer and calculates the time from the start of the measurement for the SPC module 'mod\_no'. It should be called during the measurement, because the timer starts to run after (re)starting the measurement.

For collection times longer than 80 seconds be sure that SPC\_test\_state is called in the loop during the measurement before the SPC\_get\_time\_from\_start call. SPC\_test\_state updates software counter which is needed for times longer than 80 sec.

The procedure can be used to test the progress of the measurement or to the start next measurement step in a multi-step measurements (such as f(t,T) in the standard software).

When the sequencer is running the repeat timer is not available. In this case SPC\_get\_time\_from\_start uses a software timer to measure the time from the start of the measurement.

-----  
short CVICDECL SPC\_get\_break\_time(short mod\_no, float \*time);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
*time	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure calculates for the SPC module 'mod\_no' the time from the start of the measurement to the moment of a measurement interruption by a user break (SPC\_stop\_measurement or SPC\_pause\_measurement) or by a stop on overflow. The procedure can be used to find out the moment of measurement interrupt.

-----  
short CVICDECL SPC\_get\_actual\_coltime(short mod\_no, float \*time);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
*time	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure reads the timer for the collection time and calculates the actual collection time value for the SPC module 'mod\_no'. During the measurement this value decreases from the specified collection time to 0.

For collection times longer than 80 seconds be sure that SPC\_test\_state is called in the loop during the measurement before the SPC\_get\_actual\_coltime call. SPC\_test\_state updates software counter which is needed for times longer than 80 sec.

In comparison to the procedure SPC\_get\_time\_from\_start, which delivers the real time from start, the procedure returns the actual state of the dead time compensated collection time. At

high count rates the real time of collection can be considerably longer than the specified collection time value.

For SPC6x0,130 modules only:

- If the sequencer is running, the collection timer cannot be accessed.
- The dead time compensation can be switched off. In this case the collection timer runs with the same speed as the repeat timer, and the result is the same as that of the procedure `SPC_get_time_from_start`.

-----  
short CVICDECL SPC\_read\_rates(short mod\_no, rate\_values \*rates);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
* rates	pointer to result rates structure

Return value:

0 - OK, -SPC\_RATES\_NOT\_RDY - rate values not ready yet, < 0: error code

The procedure reads the rate counters for the SPC module 'mod\_no', calculates the rate values and writes the results to the 'rates' structure.

The procedure can be called at any time after an initial call to the `SPC_clear_rates` function. If the rate values are ready (after 1sec of integration time), the procedure fills 'rates', starts a new integration cycle and returns 0, otherwise it returns -SPC\_RATES\_NOT\_RDY.

Integration time of rate values is equal 1sec, but for SPC-130/830/930/150 modules can have also other values according to the parameter `RATE_COUNT_TIME` (1.0s, 250ms, 100ms, 50ms are possible).

For SPC-140 module integration time is equal 50 ms. During this time only one rate value is collected. When one value is ready the procedure switches the hardware to collect the next one. Therefore for SPC-140 module procedure must be called in the loop (minimum 4 times) until it returns 0 – it means that all rate values are ready.

-----  
short CVICDECL SPC\_clear\_rates(short mod\_no);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
--------	---------------------------

Return value: 0: no errors, <0: error code

Description:

The procedure clears all rate counters for the SPC module 'mod\_no'.

To get correct rate values the procedure must be called once before the first call of the `SPC_read_rates` function. `SPC_clear_rates` starts a new rate integration cycle.

---

```
short CVICDECL SPC_get_sequencer_state(short mod_no, short *state);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
*state	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure is used to get the current state of the sequencer status bits on SPC module 'mod\_no'. The sequencer status bits are defined in the spcm\_def.h file:

SPC_SEQ_ENABLE	0x1	sequencer is enabled
SPC_SEQ_RUNNING	0x2	sequencer is running
only for SPC6x0/130/150 modules		
SPC_SEQ_GAP_BANK	0x4	sequencer is waiting for other bank to be armed

---

```
short CVICDECL SPC_read_gap_time(short mod_no, float *time);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
* time	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure is used to read the gap time that can occur during a measurement with the sequencer of SPC6x0/130/150 modules. 'time' is set to the last gap time in ms on the SPC module 'mod\_no'.

---

```
short CVICDECL SPC_get_scan_clk_state (short mod_no, short *scan_state);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number
*scan_state	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure sets "scan\_state" according to the actual state of the scanning clocks on the SPC module 'mod\_no'.

Scan\_state value is valid only when the module's measurement mode is set to 'Scan In' ( by setting parameter MODE using SPC\_set\_parameter procedure). Otherwise the procedure returns error code -SPC\_BAD\_FUNC.

The procedure works only for SPC-830/140/150/930 modules and SPC-7x0 modules with FPGA version greater than 300(hex). For other modules it returns error code - SPC\_BAD\_FUNC (FPGA version can be checked using SPC\_get\_version procedure).

Scan\_state bits should be interpreted as follows:

Bit mask value (hex):	1 – External Pixel Clock present
	2 – Line Clock present
	4 – Frame Clock present

-----  
short CVICDECL SPC\_get\_fifo\_usage (short mod\_no, float \*usage\_degree);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
* usage_degree	pointer to result variable

Return value: 0: no errors, <0: error code

The procedure works only for SPC-130/140/830/6x0/930/150 modules in fifo mode.

The procedure sets "usage\_degree" with the value in the range from 0 to 1, which tells how occupied is FIFO memory on the SPC module 'mod\_no'.

FIFO memory usage is set to 0 at the start of FIFO measurement ( in SPC\_start\_measurement) and after reading data from FIFO ( SPC\_read\_fifo).

## Measurement control functions:

---

```
short CVICDECL SPC_start_measurement(short mod_no);
```

---

Input parameters:

mod\_no            0 .. 7, SPC module number

Return value: 0: no errors, <0: error code

The procedure is used to start the measurement on the SPC module 'mod\_no'.

Before a measurement is started by SPC\_start\_measurement

- the SPC parameters must be set (SPC\_init or SPC\_set\_parameter(s) ),
- the SPC memory must be configured ( SPC\_configure\_memory in normal modes),
- the measured blocks in SPC memory must be filled (cleared) (SPC\_fill\_memory),
- the measurement page must be set (SPC\_set\_page)

Because of hardware differences the procedure action is different for different SPC module types.

If the sequencer is not enabled (normal measurement):

- The repeat and collection timers are started with the specified collect\_time
- The SPC is armed i.e. the photon collection is started.

If the sequencer is enabled ('Continuous Flow Mode'):

If the sequencer is not running:

- The sequencer is started
- The SPC is armed for next memory bank. The photon collection is not yet started! This action is not done when sequencer was enabled with 'enable' = 2 ( see also SPC\_enable\_sequencer function ) !
- SPC is armed for the current memory bank and the photon collection is started.

If the sequencer is already running:

- SPC is armed for the current memory bank
- The memory bank is reversed.

For SPC-6x0/130/830/140/930/150 in FIFO mode:

- Macro time and FIFO are cleared
- The SPC is armed i.e. the photon collection is started.

---

```
short CVICDECL SPC_pause_measurement(short mod_no);
```

---

Input parameters:

mod\_no            0 .. 7, SPC module number

Return value: 0: not paused, because already finished,  
> 0 paused, <0: error code

The procedure is used to pause a running measurement on the SPC module 'mod\_no'.

Because of hardware differences the procedure action is different for different SPC module types.

For all SPC module types (except SPC-6x0/130/830/140/930/150 in FIFO modes) :

When the sequencer is not enabled (normal measurement):

- the repeat and collection timers are stopped,
- the SPC is disarmed (photon collection is stopped).

When the sequencer is enabled:

- an error is returned - this measurement can't be paused

For SPC-6x0/130/830/140/930/150 in FIFO mode:

The procedure should not be used for FIFO modules.

The measurement can be restarted by the procedure 'SPC\_restart\_measurement'.

---

```
short CVICDECL SPC_restart_measurement(short mod_no);
```

---

Input parameters:

mod\_no            0 .. 7, SPC module number

Return value: 0: no errors, <0: error code

The procedure is used to restart a measurement that was paused by SPC\_pause\_measurement on the SPC module 'mod\_no'.

Because of hardware differences the procedure action is different for different SPC module types.

For all SPC module types (except SPC-6x0/130/830/140/930/150 in FIFO modes):

When the sequencer is not enabled (normal measurement):

- the repeat and collection timers are started,
- the SPC is armed (photon collection is started).

When the sequencer is enabled:

- an error is returned, this measurement can't be restarted

For SPC-6x0/130/830/140/930/150 in FIFO mode:

The procedure should not be used for FIFO modules.

---

```
short CVICDECL SPC_stop_measurement(short mod_no);
```

---

---

Input parameters:

mod\_no            0 .. 7, SPC module number

Return value: 0: no errors, <0: error code

The procedure is used to terminate a running measurement on the SPC module 'mod\_no'. Because of hardware differences the procedure action is different for different SPC module types.

**For all SPC module types (except SPC-6x0/130/830/140/930/150 in FIFO mode):**

If the sequencer is not enabled (normal measurement):

- The SPC is disarmed (i.e. the photon collection is stopped)
- The repeat and collection timers are read to get the break times

When the sequencer is enabled:

- The sequencer is stopped
- The SPC is disarmed (photon collection is stopped)

The procedure should be called after finished scan mode measurement to stop the sequencer and clear scan flags (SPC\_FBRDY).

**For all SPC module types (except SPC-6x0/130) in SCAN\_IN mode:**

If the measurement was started in SCAN\_IN mode (and sequencer is enabled), 1<sup>st</sup> call to the function forces very short collection time to finish the current frame and returns error -21. The measurement will stop automatically after finishing current frame. 2<sup>nd</sup> call will stop the measurement without waiting for the end of frame.

**For SPC-6x0/130/830/140/930/150 in FIFO mode:**

- The SPC is disarmed (photon collection is stopped)
- The FIFO pipeline is cleared

---

short CVICDECL SPC\_set\_page ( short mod\_no, long page);

---

Input parameters:

mod\_no            0 .. 7, SPC module number, -1 all used modules  
page                page number, 0 to maxpage - 1

Return value:        0: no errors, <0: error code

The procedure defines the page of memory (on SPC module 'mod\_no') in which the data of a subsequent measurement will be recorded. SPC\_set\_page must be called before a measurement is started.

If 'mod\_no' = -1, page will be changed for all SPC modules which are actually in use. Be sure that in such case all modules are configured in the same way (the best is to use SPC\_configure\_memory with 'mod\_no' = -1).

The range of the parameters 'block' and 'page' depends on the actual configuration of the SPC memory (see SPC\_configure\_memory). To provide correct access to the SPC memory it is required that the function SPC\_configure\_memory be used before the first call of SPC\_set\_page ( in normal modes). This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0/130)
short block_length	Number of 16-bits words per one block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement

-----  
short CVICDECL SPC\_enable\_sequencer(short mod\_no, short enable);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number, -1 all used modules
enable	0 or 1(2) to disable or enable

Return value: 0: no errors, <0: error code

The procedure is used to enable or disable the sequencer of the SPC module 'mod\_no'.

If 'mod\_no' = -1, sequencer will be enabled/disabled for all SPC modules which are actually in use.

If enable = 0:

If the sequencer is running:

- The sequencer is stopped and disabled,
- The SPC is disarmed (photon collection is stopped),

If the sequencer was enabled:

- The sequencer is disabled
- The dead time compensation of collection timer is switched to the state specified in the system parameters

When enable = 1 or 2:

If the sequencer was not enabled:

- The sequencer is enabled
- The dead time compensation of the collection timer is switched off

The way of enabling sequencer ( 'enable' = 1 or 2 ) changes slightly the action of SPC\_start\_measurement function for sequencer operation on SPC-130/6x0/150. When 'enable' =1, SPC\_start\_measurement arms SPC for both memory banks, while for 'enable' = 2, the function arms SPC only for current memory bank.

The 2<sup>nd</sup> case is used by the main software to program Continuous Flow measurements with accumulation.

The sequencer must be enabled before starting the measurement in the following cases:

- Continuous flow measurements for SPC modules 130/150 and 6x0 - normal operation (routing in ) with sequencer
- scanning modes ( Scan In , Scan Out ) for SPC modules 7x0, 830, 140, 150, 930 )
- block address out mode for SPC modules 7x0

### SPC memory transfer functions:

---

```
short CVICDECL SPC_configure_memory (short mod_no, short adc_resolution,  
                                     short no_of_routing_bits,  
                                     SPCMemConfig * mem_info);
```

---

Input parameters:

mod_no	0 .. 7, SPC module number, -1 all used modules
adc_resolution	ADC resolution (-1*,6,8,10,12 additionally 0,2,4 for SPC-830/140/930/150) * With adc_resolution = -1 the procedure 'mem_info' is filled with the current values disregarding 'no_of_routing_bits'.
no_of_routing_bits	number of routing bits (0 - 3 for SPC-130) (0 - 7 for SPC-6x0/140/150) (0 - 14 for SPC-7xx(830) modules),
* mem_info	pointer to result memory info structure

Return value:

0 no errors, <0 error code

The procedure configures the memory of SPC module 'mod\_no' depending on the specified ADC resolution, the module type and the number of detector channels (if a router is used). The action is done for normal operation modes. In FIFO modes the procedure sets hardware defined fixed values to ADC and no\_of\_routing\_bits. In Scan modes the procedure does not configure the memory and should be called with 'adc\_resolution' = -1 to get the current state of the DLL SPCMemConfig structure ( after setting scan parameters).

If 'mod\_no' = -1, memory will be configured on all SPC modules which are actually in use.

The procedure has to be called before the first access to the SPC memory or before a measurement is started and always after setting ADC resolution parameter value. The memory configuration determines in which part of SPC memory the measurement data is recorded (see also SPC\_set\_page).

The SPC memory is interpreted as a set of 'pages'. One page contains a number of 'blocks'. One block contains one decay curve. The number of points per block (per curve) is defined by the ADC resolution. The number of blocks per page depends on the number of points per block and on the number of detector channels (PointsX and PointsY)

In the scanning modes of the SPC-7/8/9/140/150, a page contains a number of 'frames' (normally 1). Each frame contains a number of blocks. The number of blocks per page depends on the number of points per block and on the number of detector channels (PointsX and PointsY) and on the scanning parameters (pixels per line and lines per frame).

The differences between the modules are listed below.

#### SPC-130/600/630/150 modules:

The length of the recorded curves is determined by the ADC resolution and can range from 64(0 for SPC-150) to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

#### SPC-130/600/630/150 modules in the Histogram modes:

The length of the recorded curves is determined by the ADC resolution and can range from 64 (0 for SPC-150) to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

#### SPC-130/600/630/830/140/150/930 modules in the Fifo modes:

The module memory is configured as a FIFO memory – there are no curves and pages. Instead, a stream of collected photons is written to the fifo. A SPC\_configure\_memory function call is not required.

#### SPC-700/730/830/140/150/930 modules, Normal operation modes:

The length of the recorded curves is determined by the ADC resolution and can range from 0( SPC-830 and other) or 64( only SPC-7x0) to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

#### SPC-700/730/830/140/150/930 modules, Scanning modes:

The Memory configuration is not done not by SPC\_configure\_memory. Instead, the memory is configured by but by setting the parameters:

ADC\_RESOLUTION – defines block\_length,  
SCAN\_SIZE\_X, SCAN\_SIZE\_Y – defines blocks\_per\_frame  
SCAN\_ROUT\_X, SCAN\_ROUT\_Y – defines frames\_per\_page

However, after setting these parameters SPC\_configure\_memory should be called with 'adc\_resolution' = -1 to get the current state of the DLL SPCMemConfig structure.

To ensure correct access to the curves in the memory by the memory read/write functions, the SPC\_configure\_memory function loads a structure of the type SPCMemConfig with the values listed below:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0(130/150))
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement
short block_length	Number of curve points 16-bits words per block (curve)

Possible operation modes for the SPC modules are defined in the spcm\_def.h file. The operation mode can be changed by setting the parameter MODE.

-----  
short CVICDECL SPC\_fill\_memory(short mod\_no, long block, long page, unsigned short  
fill\_value);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number, -1 all used modules
block	block number to be filled
page	page number
fill_value	value written to SPC memory

Return value:

0: no errors, fill is finished, <0: error code, >0: number of modules on which filling the memory is started but not finished

The procedure is used to clear the measurement memory before a new measurement is started.

If 'mod\_no' = -1 memory on all used SPC modules will be cleared, otherwise only on the module 'mod\_no'.

The procedure fills a specified part of the SPC memory with the value 'fill\_value'. To provide correct memory access it is required that the function SPC\_configure\_memory be used ( normal operation modes) before the first call of SPC\_fill\_memory and always after setting ADC resolution parameter value.

The parameter 'block' can range from 0 to blocks\_per\_page - 1. If the value '-1' is used all blocks on the specified page(s) are filled. The parameter 'page' can vary from 0 to maxpage - 1. If the value '-1' is used all pages in current memory bank are filled.

The procedure returns on success the number of the modules on which filling the memory was started but is still not finished. If this value is > 0, the function SPC\_test\_state must be next called to check whether the started filling process is already finished ( if the bit SPC\_HFILL\_NRDY is set in state, filling is not finished, see spcm\_def.h for bit definition).

-----  
short CVICDECL SPC\_read\_data\_block( short mod\_no, long block, long page,  
short reduction\_factor, short from, short to,  
unsigned short \*data);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
block	block number to read, 0 to blocks_per_page - 1
page	page number, 0 to maxpage - 1
reduction_factor	data reduction factor
from	first point number
to	last point number
*data	pointer to data buffer which will be filled

Return value:

0 no errors, <0 error code

The procedure reads data from a block of the SPC memory (on module 'mod\_no') defined by the parameters 'block' and 'page' to the buffer 'data'. The procedure is used to read measurement results from the SPC memory.

The function performs a data reduction by averaging a specified number of data points into one result value. The parameter 'reduction\_factor' defines the number of points that are averaged. The value of 'reduction\_factor' must be a power of 2. The number of values stored in 'data' (named below no\_of\_points) is equal to block length divided by reduction\_factor.

The parameters 'from' and 'to' define the address range inside the buffer 'data' i.e. refer to the (compressed) destination data. 'from' and 'to' must be in the range from 0 to no\_of\_points-1. The parameter 'to' must be greater than or equal to the parameter 'from'.

The range of the parameters 'block' and 'page' depends on the actual configuration of the SPC memory (see SPC\_configure\_memory).

The assumption is done that frames\_per\_page is equal 1 ( page = frame ) (see 'Memory Configuration').

To provide correct access to the SPC memory it is required that the function SPC\_configure\_memory be used ( in normal operation modes) before the first call of SPC\_read\_data\_block. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0/130/150)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement
short block_length	Number of 16-bits words per one block (curve)

Please make sure that the buffer 'data' be allocated with enough memory for no\_of\_points.

-----  
short CVICDECL SPC\_write\_data\_block(short mod\_no, long block, long page, short from,  
short to, unsigned short \*data);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
block	block number to read, 0 to blocks_per_page - 1
page	page number, 0 to maxpage - 1
from	first point number, 0 to block length - 1
to	last point number, 'from' to block length - 1
*data	pointer to data buffer

Return value: 0: no errors, <0: error code

The procedure reads data from the buffer 'data' in the PC and writes it to a block of the memory defined by the parameters 'block' and 'page' on the SPC module 'mod\_no'. The procedure is used to write data from the from PC memory to the memory of the SPC module.

Parameters 'from' and 'to' define the address range inside the buffer 'data' and the address range inside the SPC memory block to which the data will be written.

The range of the parameters 'block' and 'page' depends on the actual configuration of the SPC memory (see SPC\_configure\_memory).

The assumption is done that frames\_per\_page is equal 1 ( page = frame ) (see 'Memory Configuration').

To provide correct access to the SPC memory it is required that the function SPC\_configure\_memory be called ( in normal operation modes) before the first call of SPC\_write\_data\_block. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0/130/150)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement
short block_length	Number of 16-bits words per one block (curve)

-----  
short CVICDECL SPC\_read\_fifo (short mod\_no, unsigned long \* count, unsigned short \*data);  
-----

Input parameters:

mod_no	0 .. 7, SPC module number
*count	pointer to variable which: - on input contains required number of 16-bit words - on output will be filled with number of 16-bit words written to the buffer 'data'
*data	pointer to data buffer which will be filled

Return value:

0 no errors, <0 error code

The procedure reads data from the FIFO memory of SPC modules SPC-6x0/130/830/140/150 and has no effect for other SPC module types. Because of hardware differences the procedure action is different for different SPC module types.

For SPC600(630) modules:

Before calling the function FIFO mode must be set by calling function SPC\_set\_parameter to change parameter MODE to one of two possible FIFO modes: FIFO\_48 (48 bits frame) or FIFO\_32 (32 bits frame) (fifo mode values are defined in spcm\_def.h file).

For FIFO\_48 mode the function reads 48-bits frames from the FIFO memory and writes them to the buffer 'data' until the FIFO is empty or 'Count' number of 16-bit words was already written. The 'Count' variable is filled on exit with the number of 16-bit words written to the buffer.

For FIFO\_32 mode the function reads 32-bits frames from the FIFO memory and writes them to the buffer 'data' until the FIFO is empty or 'Count' number of 16-bit words was already written. The 'Count' variable is filled on exit with the number of 16-bit words written to the buffer. Subsequent frames which don't contain valid data but only macro time overflow information are compressed to one frame which contains the number of macro time overflows in the bits 29:0. It enables a correct macro time calculation and eliminates invalid data frames from the buffer.

For SPC130/830/140/150/930 modules:

Before calling the function FIFO mode must be set by calling function SPC\_set\_parameter to change parameter MODE to FIFO mode (32 bits frame different than for SPC6x0 modules) (fifo mode values are defined in spcm\_def.h file). After setting FIFO mode SPC module memory has FIFO structure. SPC\_read\_fifo function reads 32-bits frames from the FIFO memory and writes them to the buffer 'data' until the FIFO is empty or 'Count' number of 16-bit words was already written.

The 'Count' variable is filled on exit with the number of 16-bit words written to the buffer. Subsequent frames which don't contain valid data ( photons or markers ) but only macro time overflow information are compressed to one frame which contains the number of macro time overflows in the bits 29:0. It enables a correct macro time calculation and eliminates invalid data frames from the buffer.

Please make sure that the buffer 'data' be allocated with enough memory for the expected number of frames (at least 'Count' 16-bit words).

```
-----
short  CVICDECL  SPC_read_data_frame (short mod_no, long frame, long page,
                                     unsigned short *data);
-----
```

Input parameters:

mod_no	0 .. 7, SPC module number
frame	frame number to read, 0 to frames_per_page - 1, or -1
page	page number, 0 to maxpage - 1
*data	pointer to data buffer which will be filled

Return value:

0 no errors, <0 error code

The procedure reads data from a frame of the SPC memory on module 'mod\_no' defined by the parameters 'frame' and 'page' to the buffer 'data'. The procedure is used to read measurement results from the SPC memory when frames\_per\_page is greater than 1 (this can be the case for SPC7x0/830/140 modules in scanning modes).

The range of the parameters 'frame' and 'page' depends on the actual configuration of the SPC memory (see SPC\_configure\_memory).

If 'frame' is equal -1, all frames(frames\_per\_page) from the page 'page' are read.

To provide correct access to the SPC memory it is required that the function SPC\_configure\_memory be used( in normal operation modes) before the first call of

SPC\_read\_data\_frame. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0/130/150)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement
short block_length	Number of 16-bits words per one block (curve)

Please make sure that the buffer 'data' be allocated with enough memory for  
 block\_length \* blocks\_per\_frame 16-bit values, when one frame is read, or  
 block\_length \* blocks\_per\_frame \* frames\_per\_page 16-bit values, when 'frame' = -1.

-----  
 short CVICDECL SPC\_read\_data\_page (short mod\_no, long first\_page, long last\_page,  
 unsigned short \*data);  
 -----

Input parameters:

mod_no	0 .. 7, SPC module number
first_page	number of the first page to read, 0 to maxpage - 1
last_page	number of the last page to read, first_page to maxpage - 1
*data	pointer to data buffer which will be filled

Return value:

0 no errors, <0 error code

The procedure reads data from the pages of the SPC memory on module 'mod\_no' defined by the parameters 'first\_page' and 'last\_page' to the buffer 'data'. The procedure is used to read measurement results from the SPC memory.

The procedure is recommended when big amounts of SPC memory must be read as fast as possible, because it works much faster than calling in the loop the function SPC\_read\_data\_block. Even the whole memory bank can be read in one call, when 'first\_page' = 0 and 'last\_page' = maxpage - 1.

The range of the parameters 'first\_page' and 'last\_page' depends on the actual configuration of the SPC memory (see SPC\_configure\_memory).

To provide correct access to the SPC memory it is required that the function SPC\_configure\_memory be used ( in normal operation modes) before the first call of SPC\_read\_data\_page. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0/130/150)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement
short block_length	Number of 16-bits words per one block (curve)

Please make sure that the buffer 'data' be allocated with enough memory for  $\text{block\_length} * \text{blocks\_per\_frame} * \text{frames\_per\_page} * (\text{last\_page} - \text{first\_page} + 1)$  16-bit values.

```
-----  
short CVICDECL SPC_read_block(short mod_no, long block, long frame, long page,  
                             short from, short to, unsigned short *data);  
-----
```

Input parameters:

mod_no	0 .. 7, SPC module number
block	block number to read, 0 to blocks_per_page - 1
frame	frame number to read, 0 to frames_per_page - 1, or -1
page	page number, 0 to maxpage - 1
from	first point number
to	last point number
*data	pointer to data buffer which will be filled

Return value:

0 no errors, <0 error code

The procedure reads data from a block of the SPC memory (on module 'mod\_no') defined by the parameters 'block', 'frame' and 'page' to the buffer 'data'. The procedure is used to read measurement results from the SPC memory especially for SPC7x0/830/140/150 modules in scanning modes ( when frames\_per\_page is greater than 1).

The parameters 'from' and 'to' define the address range inside the buffer 'data' i.e. refer to the destination data. 'from' and 'to' must be in the range from 0 to block\_length -1. The parameter 'to' must be greater than or equal to the parameter 'from'.

The range of the parameters 'block', 'frame' and 'page' depends on the actual configuration of the SPC memory (see SPC\_configure\_memory).

To provide correct access to the SPC memory it is required that the function SPC\_configure\_memory be used ( in normal operation modes) before the first call of SPC\_read\_block. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-6x0/130/150)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement
short block_length	Number of 16-bits words per one block (curve)

Please make sure that the buffer 'data' be allocated with enough memory for block\_length.

-----  
short CVICDECL SPC\_save\_data\_to\_sdtfile( short mod\_no, unsigned short \*data\_buf,  
unsigned long bytes\_no, char \* sdt\_file);  
-----

Input parameters:

mod_no	-1, 0 .. 7, SPC module number
data_buf	pointer to data buffer which will be saved
bytes_no	number of bytes to save from the data_buf
sdt_file	file path of the result sdt file

Return value:

0 no errors, <0 error code

This procedure saves measurement data from the buffer 'data\_buf' into the 'sdt\_file' file in the .sdt format using current parameters of the SPC module 'mod\_no'.

**The assumption is done, that before using this function user made a measurement, read the results from SPC memory and that no DLL parameters were changed in between.** Because during the measurements always whole page(s) is affected, the best choice to read results is SPC\_read\_data\_page function.

The created .sdt file can then be loaded into the SPC standard measurement software.

It contains a file header, an INFO section, measurement description block(s) and data set(s). It does not contain SETUP section, therefore Display, Trace, Window, Print settings will not change when the file will be loaded to SPC software.

( for .sdt format details see SPC hardware manual and spc\_mininfo.h file)

One measurement description block and data set is created per one module (measurement description block fields are created from current 'mod\_no' module's parameters).

In case of multi-module configuration - if 'mod\_no' = -1 (all used modules), measurement description blocks and data sets are created for all modules of the same type which are 'in\_use' ( call SPC\_get\_module\_info to know which modules are 'in\_use', SPC\_set\_mode to change used modules configuration)

The 'data\_buf' buffer should contain measurement data which were read earlier from SPC memory using one of memory transfer functions (SPC\_read\_data\_page function is recommended).

When 'mod\_no' = -1 ( all used modules), the buffer should contain the data of all used modules in contiguous way (one after another – after last byte of 1<sup>st</sup> module's data the first byte of 2<sup>nd</sup> module should appear).

Please make sure that the buffer is allocated with minimum 'bytes\_no' bytes, otherwise it can cause a crash.

The procedure checks at the beginning whether 'bytes\_no' parameter fits to the current memory configuration ( use SPC\_configure\_memory with parameter 'adc\_resolution' = -1 to get it).

'bytes\_no' must be equal to  $2 * page\_size * no\_of\_pages * no\_of\_modules$ , where

$page\_size = blocks\_per\_frame * frames\_per\_page * block\_length$ ,

$no\_of\_pages = always\ 1$ ,

except 'Continuos Flow' mode ( for modules SPC6x0/130/150 with mode = Normal and with sequencer enabled), where no\_of\_pages = maxpage,

no\_of\_modules = 1, when 'mod\_no'parameter >= 0,

= number of active modules, when 'mod\_no'parameter = -1

Different measurement modes are used in .sdt file depending on the module type and current DLL parameters :

- mode SINGLE : all module types, when DLL parameter MODE = NORMAL and sequencer is disabled
- mode 'Continuos Flow' : SPC-6x0 & SPC-130/150 module types, when DLL parameter MODE = NORMAL and sequencer is enabled
- mode 'Scan Sync In' : SPC-7x0/830/140/930/150 module types, when DLL parameter MODE = SCAN\_IN and sequencer is enabled
- mode 'Scan Sync Out' : SPC-7x0/830/140/930/150 module types, when DLL parameter MODE = SCAN\_OUT and sequencer is enabled
- mode 'Scan XY Out' : SPC-7x0 module type, when DLL parameter MODE = ROUT\_OUT and sequencer is enabled
- mode 'Camera' : SPC-930 module type, when DLL parameter MODE = CAMERA

Other combinations of MODE value and module type are not supported and will return error. Especially the procedure does not create the file ( returns error) when DLL parameter MODE is set to one of FIFO modes ( for modules SPC-6x0/830/130/140/930/150).



The procedure is needed to initiate the process of extracting photons from a stream of .spc files created during FIFO measurement.

If the files were created using BH measurement software, 1<sup>st</sup> entry in each file contains Macro Time clock resolution and used routing channels information . In such case bit 0 of 'stream\_type' parameter should be set to 1, otherwise if the 1<sup>st</sup> entry have no special meaning ( just photon frame) set it to 0. Set bit 9 of 'stream\_type' if the file contains markers (was created in FIFO\_IMG mode or FIFO mode with enabled markers).

Subsequent files created in BH software during one measurement have 3 digits file number in file name part of the path ( for example xxx000.spc, xxx001.spc and so on). Such files can be treated together during extracting photons ( they contain the same measurement) as a stream of files. When all photons from the 1<sup>st</sup> file will be extracted, the 2<sup>nd</sup> one will be opened during extraction and so on. The first file in the stream is given by 'spc\_file' parameter and 'files\_to\_use' parameter tells how many files belong to the stream ( -1 means the procedure will evaluate number of files in the stream and use it as 'files\_to\_use' value).

'fifo\_type' parameter defines the format of the photons data in the stream. It depends mainly on the SPC module type which was used during the measurement. Possible 'fifo\_type' values are defined in the spcm\_def.h file.

'what\_to\_read' parameter defines which entries will be extracted from the stream.

In most cases only bit 0 will be set ( valid photons ). For files containing also markers – markers 0-3 (pixel, line, frame) can also be extracted ( bits 2-5).

If the stream is successfully initialised, the procedure creates internal DLL PhotStreamInfo structure and returns the handle to the stream ( positive value). Use this handle as an input parameter to the other extraction functions ( SPC\_close\_phot\_stream, SPC\_get\_phot\_stream\_info, SPC\_get\_photon).

Max 8 streams can be initialised by the SPCM DLL.

Use SPC\_get\_phot\_stream\_info to get the current state of the stream and SPC\_get\_photon to extract subsequent photons from the stream.

After extracting photons stream should be closed using SPC\_close\_phot\_stream function.

See use\_spcm.c file for the extraction example.

```
-----  
short          CVICDECL          SPC_get_phot_stream_info          (short          stream_hndl,  
                                     PhotStreamInfo * stream_info );  
-----
```

Input parameters:

stream\_hndl handle of the initialised photons stream

stream\_info pointer to the stream info structure

Return value: 0: no errors, <0: error code

The procedure fills 'stream\_info' structure with the contents of DLL internal structure of the stream defined by handle 'stream\_hndl'. Procedure returns error, if 'stream\_hndl' is not the handle of the stream opened using SPC\_init\_phot\_stream function.

PhotStreamInfo structure is defined in the spcm\_def.h file.

-----  
short CVICDECL SPC\_get\_photon (short stream\_hndl, PhotInfo \* phot\_info );  
-----

Input parameters:

stream\_hndl handle of the initialised photons stream  
phot\_info pointer to the photon info structure

Return value: 0: no errors, <0: error code

The procedure can be used in a loop to extract subsequent photons from the opened photons stream defined by handle 'stream\_hndl'.

The procedure fills 'phot\_info' structure with the information of the photon extracted from the current stream position. After extracting procedure updates internal stream structures. If needed, it opens and read data from the next stream file. Use SPC\_get\_phot\_stream\_info function to get current stream state.

Procedure returns error, if 'stream\_hndl' is not the handle of the stream opened using SPC\_init\_phot\_stream function.

PhotInfo structure is defined in the spcm\_def.h file.

-----  
short CVICDECL SPC\_close\_phot\_stream (short stream\_hndl );  
-----

Input parameters:

stream\_hndl handle of the initialised photons stream  
phot\_info pointer to the photon info structure

Return value: 0: no errors, <0: error code

The procedure is used to close the opened photons stream defined by handle 'stream\_hndl' after extraction of the photons.

The procedure frees all stream's memory and finally invalidates the handle 'stream\_hndl'.

Procedure returns error, if 'stream\_hndl' is not the handle of the stream opened using SPC\_init\_phot\_stream function.

```
-----  
short CVICDECL SPC_get_detector_info (short previous_type, short * det_type,  
                                     char * fname );  
-----
```

Input parameters:

previous_type	0 .. 9999, type of the detector from which list of detectors will be searched
det_type	pointer filled with the next detector type found on the list of detectors
fname	string filled with filename of the corresponding .bit file of det_type detector

Return value: 0: no errors, <0: error code

The procedure is used to look for specific detector type on the detectors list for SPC-930 module.

When next detector type is found, procedure sets 'det\_type' to the detector type and fills 'fname' with the proper .bit filename.

The procedure sets 'det\_type' to -1, when next detector type is not found, or when module type is not SPC-930.

To use specific detector in SPC-930 Camera mode proper .bit file must be loaded to Xilinx chip on the module to define hardware behaviour. During initialisation ( SPC\_init ) a list of possible detectors is created (only when SPC-930 module is in use ) by looking for .bit files in the directory where ini file is located.

Detectors .bit filenames must conform following naming convention "detector\_name#xxyy.bit", where xx – detector's type (two decimal digits), yy – detector's file version ( two decimal digits ). For example "Resistive Anode 4chan#0101.bit" it is Hamamatsu Resistive Anode 4 channels detector , which has type 01 and version 01.

To find 1<sup>st</sup> available detector's type call the procedure with 'previous\_type' parameter = 0.

Detector type value 99 is reserved for .bit files which defines Xilinx configuration for modes other than Camera mode.

SPC-930 hardware will be prepared for using required detector by setting proper detector\_type in spcm.ini file or by setting parameter DETECTOR\_TYPE using SPC\_set\_parameter function.

```
-----  
short CVICDECL SPC_close(void);
```

---

Input parameters: none

Return value: 0: no errors, <0: error code

It is a low level procedure and not intended to normal use.

The procedure frees buffers allocated via DLL and set the DLL state as before SPC\_init call.

SPC\_init is the only procedure which can be called after SPC\_close.

---