SPC
SPCM Dynamic Link Libraries

USER MANUAL

Version 4.0, March 2023

Content

Introduction

The SPCM 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 32 or 64 bit Windows 10/11. Both 32 and 64-bit DLL versions are available. 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	32-bit dynamic link library main file for use on 32-bit systems
SPCM32x64.DLL	32-bit dynamic link library main file for use on 64-bit systems
SPCM32.LIB	import library file for Microsoft Visual C/C++ for use on 32-bit systems
SPCM32x64.LIB	import library file for Microsoft Visual C/C++ for use on 64-bit systems
SPCM64.DLL	64-bit dynamic link library main file for use on 64-bit systems
SPCM64.LIB	import library file for Microsoft Visual C/C++ for use on 64-bit systems
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.C	Simple example of using SPC DLL functions for DPC-230 module. Source file of the example is the file use_dpc.c.
USE_SPCM.C	Simple example of using SPC DLL functions for SPC modules. Source file of the example is the file use_spcm.c.

To install DLLs execute installation package (tcspc_setup_(32/64).exe) 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

Functions to Manage Photons Streams:

SPC_init_phot_stream
SPC_close_phot_stream
SPC_get_phot_stream_info
SPC_get_photon
SPC_get_fifo_init_vars
SPC_init_buf_stream
SPC_add_data_to_stream
SPC_read_fifo_to_stream
SPC_get_photons_from_stream
SPC_stream_start_condition
SPC_stream_stop_condition
SPC_stream_reset
SPC_get_stream_buffer_size
SPC_get_buffer_from_stream

Other Functions:

SPC_get_error_string
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(140,15x,131,160) -500 ... 0mV, for SPCx0x 5 ... 80mV, default 5mV
cfd_limit_high= 80.0            ; 5 ...80 mV, default 80 mV, not for SPC130,140,15x,131,160,930
cfd_zc_level= 0.0               ; for SPCx3x(140,15x,131,160) -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(140,15x,131,160) -96 ... 96mV, for SPCx0x -10 ... 10mV
                                ; default 0mV
sync_freq_div= 4                ; for SPC130,140,15x,131,160,930    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 SPC130,140,15x,131,160,930 doesn't exist
sync_threshold= -20.0           ; for SPCx3x(140,15x,131,160) -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%, for SPC160,150N(151)  0 .. 50%
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 SPC830,140,15x,131,160,930)
ext_latch_delay= 0              ; 0 ... 255 ns, default 0, for SPC130 doesn't exist
                                ; for SPC140,15x,131,160,930 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, 15x ,131, 160:  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),
                                ;    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 SPC15x,160, 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. > C0
                                ; for SPC131 (= SPC-130-EM), default 0
                                ;    0 - normal operation (routing in)
                                ;    1 - FIFO mode 32 bits
scan_size_x=1                   ; for SPC7x0,830,140,15x,160,930 modules in scanning modes 1 ... 65536, default 1
scan_size_y=1                   ; for SPC7x0,830,140,15x,160,930 modules in scanning modes 1 ... 65536, default 1
scan_rout_x=1                   ; number of X routing channels in Scan In & Scan Out modes
                                ; for SPC7x0,830,140,15x,160,930 modules
                                ;     1 .. 128, (SPC7x0,830), 1 ... 16 (SPC140,15x,160,930), default 1
scan_rout_y=1                   ; number of Y routing channels in Scan In & Scan Out modes
                                ; for SPC7x0,830,140,15x,160, 930 modules
                                ;     1 .. 128, (SPC7x0,830), 1 ... 16 (SPC140,15x,160,930), 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 SPC7x0,140,15x,160)  -     12
                                ;     14 (12 for SPC7x0,140,15x,160)  -     10
                                ;     16 (14 for SPC7x0,140,15x,160)  -      8
                                ;     18 (16 for SPC7x0,140,15x,160)  -      6
                                ;     20 (18 for SPC140,15x,160)      -      4
                                ;     22 (20 for SPC140,15x,160)      -      2
                                ;     24 (22 for SPC140,15x,160)      -      0
scan_polarity=0                 ; for SPC7x0,830,140,15x,160,930 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 SPC140,15x,160,830 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
scan_flyback=0                  ; for SPC7x0,830,140,15x,160,930 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
scan_borders=0                  ; for SPC7x0,830,140,15x,160,930 modules in Scan In mode, default 0
                                ; bits 15-0 Upper boarder, bits 31-16 Left boarder
pixel_time= 200e-9              ; pixel time in sec for SPC7x0,830,140,15x,160,930 modules in Scan In mode,
                                ; 50e-9 ... 1.0, default 200e-9
pixel_clock= 0                  ; source of pixel clock for SPC7x0,830,140,15x,160,930 modules in Scan In mode
                                ; 0 - internal, 1 - external, default 0
                                ; for SPC140,15x,160,830 in FIFO_32M mode it disables/enables pixel markers
                                ;     in photons stream
line_compression= 1             ; line compression factor for SPC7x0,830,140,15x,160,930 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 SPC130,6x0,15x,160,131 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 SPC140,15x,160,131 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 SPC130,15x,160,131,830,930 can be: 1.0 s, 0.25 s, 0.1 s, 0.05 s
                                ; for SPC140 fixed to 50 ms
macro_time_clk= 0             ; macro time clock definition for SPC130,140,15x,160,131,830,930 in FIFO mode
                                ; for SPC130,140,15x,160,131:
                                ;     0 – 50 ns (default), 25 ns for SPC15x,160,131 & 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 – 50 ns (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 SPC830,140,15x,160 & DPC230,930 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 (SPC140,15x,160,830 in FIFO_32M, SPC930 in Camera mode),
                                ;      1 ... 1024, default 1
img_size_y = 1                ; image Y size (SPC140,15x,160,830 in FIFO_32M, SPC930 in Camera mode),
                                ; actually equal to img_size_x (quadratic image)
img_rout_x = 1                ; no of X routing channels (SPC140,15x,160,830 in FIFO_32M, SPC930 in Camera
                                mode),
                                ;       1 ... 16, default 1
img_rout_y = 1                ; no of Y routing channels (SPC140,15x,160,830 in FIFO_32M, 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 SPC140,15x,160,131 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 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(140,15x,160,131) -500 ... 0 mV, for SPCx0x 5 ... 80 mV
float cfd_limit_high	5 ... 80 mV, default 80 mV, not for SPC130,140,15x,160,131,930
float cfd_zc_level	SPCx3x(140,15x,160,131) -96 ... 96 mV, SPCx0x -10 ... 10mV
float cfd_holdoff	SPCx0x: 5 ... 20 ns, other modules: no influence
float sync_zc_level	SPCx3x(140,15x,160,131): -96 ... 96 mV, SPCx0x: -10 ... 10 mV

float sync_holdoff	4 ... 16 ns (SPC130,140,15x,160,131,930: no influence)
float sync_threshold	SPCx3x(140,15x,160,131): -500 ... -20 mV, SPCx0x: no influence
float tac_range	50 ... 5000 ns
short sync_freq_div	1,2,4,8,16 (SPC130,140,15x,160,131,930: 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, 140,15x,160,131, 930)
short ext_latch_delay	0 ... 255 ns, SPC130: no influence
	SPC140,15x,160,131, 930: only values 0,10,20,30,40,50 ns are possible
float collect_time	0.0001 ... 100000 s, default 0.01 s
float display_time	0.0001 ... 100000 s, default 0.01 s
float repeat_time	0.0001 ... 100000 s, default 0.01 s
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 SPC130,600,630,15x,160,131: 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(140,131, 151,160)
	- bit 6 - in FIFO_32M mode,
	= 0 (default) Marker 3 not used,
	= 1 waiting for Marker 3 to start collection timer,
	(used in accumulated Mosaic measurements)
	- bit 7 - in FIFO_32M mode,
	= 0 (default) Frame pulses on Marker 2,
	= 1 Frame pulses on Marker 3,
	- bits 8 - 11 – enable (1) / disable (0), default 0
	of recording Markers 0-3 entries in FIFO mode
	- bits 12 - 15 - active edge 0 (falling), 1 (rising), default 0
	of Markers 0-3 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 SPC15x,160, 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
for SPC131, default 0
0 - normal operation (routing in)
1 - FIFO mode 32 bits

unsigned long scan_size_x; for SPC7x0,830,140,15x,160,930 modules in scanning modes 1 ... 65536, default 1

unsigned long scan_size_y; for SPC7x0,830,140,15x,160,930 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 SPC7x0,830,140,15x,160,930 modules
1 ... 128, (SPC7x0,830), 1 … 16 (SPC140,15x,160,930), default 1

unsigned long scan_rout_y; number of Y routing channels in Scan In & Scan Out modes
for SPC7x0,830,140,15x,160,930 modules
1 … 128, (SPC7x0,830), 1 .. 16 (SPC140,15x,160,930), 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 current adc_resolution:

12 (10 for SPC7x0,140,15x,160)	-	12
14 (12 for SPC7x0,140,15x,160)	-	10
16 (14 for SPC7x0,140,15x,160)	-	8
18 (16 for SPC7x0,140,15x,160)	-	6
20 (18 for SPC140,15x,160)	-	4
22 (20 for SPC140,15x,160)	-	2
24 (22 for SPC140,15x,160)	-	0

unsigned long scan_flyback; for SPC7x0,830,140,15x,160,930 modules in Scan Out or Rout Out mode,
default & minimum = 1
bits 15-0 Flyback X in number of pixels
bits 31-16 Flyback Y in number of lines

unsigned long scan_borders; for SPC7x0,830,140,15x,160,930 modules in Scan In mode, default 0
bits 15-0 Upper boarder, bits 31-16 Left boarder

unsigned short scan_polarity; for SPC7x0,830,140,15x,160,930 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 SPC140,15x,160,830 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 SPC7x0,830,140,15x,160,930 modules in Scan In mode,
pixel clock source, 0 - internal,1 - external, default 0
for SPC140,15x,160,830 in FIFO_32M mode it disables/enables pixel markers in photons stream

unsigned short line_compression; line compression factor for SPC7x0,830,140,15x,160,930 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 SPC130,6x0,15x,160,131 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 SPC140,15x,160,131 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 SPC7x0,830,140,15x,160,930 modules in Scan In mode, 50e-9 ... 1.0, default 200e-9
unsigned long ext_pixclk_div;	divider of external pixel clock for SPC7x0,830,140,15x,160,930 modules in Scan In mode, 1 ... 0x3fe, default 1

float rate_count_time;	rate counting time in sec default 1.0 sec
	for SPC130,830,930,15x,160,131 can be: 1.0 s, 0.25 s, 0.1 s, 0.05 s
	for SPC140 fixed to 50 ms

short macro_time_clk;	macro time clock definition for SPC130,140,15x,160,131,830,930 in FIFO mode for SPC130, 140,15x,160,131: 0 – 50 ns (default), 25 ns for SPC15x,160,131 & 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 – 50 ns (default), 1 - SYNC freq., for SPC930: 0 – 50 ns (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 SPC830,140,15x,160,131,930 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 (SPC140,15x,160,830 in FIFO_32M mode, SPC930 in Camera mode), 1 ... 1024, default 1
unsigned long img_size_y;	image Y size (SPC140,15x,160,830 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 (SPC140,15x,160,830 in FIFO_32M mode) 1 ... 16, default 1
unsigned long img_rout_y;	no of Y routing channels (SPC140,15x,160,830 in FIFO_32M 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, 15x,160,131 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
unsigned long 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 long 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 18-19 - not used
	bits 20-23 - active channels of TDC1 for DPC-330 Hardware Histogram modes
	bits 24-27 - active channels of TDC2 for DPC-330 Hardware Histogram modes
	bits 28-31 - not used
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

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.

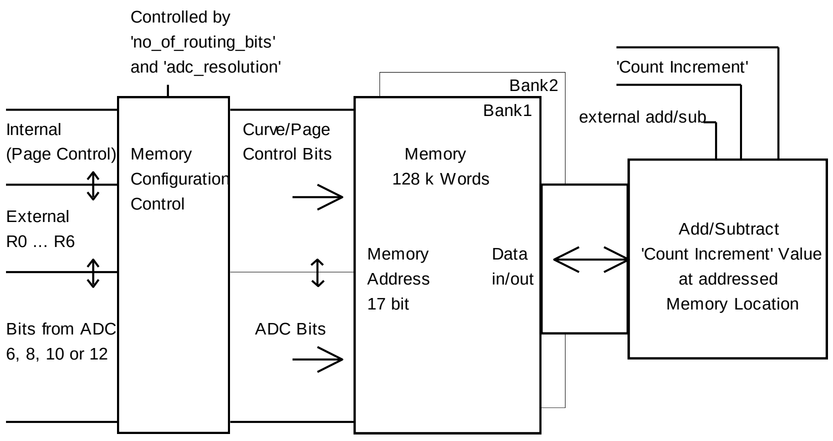
Single parameter values can be transferred to or from the DLL and module level by the functions **SPC_set_parameter** and **SPC_get_parameter**. To identify the desired parameter, the parameter identification par_id is used. For the parameter identification keywords are defined in spcm_def.h.

Memory Configuration

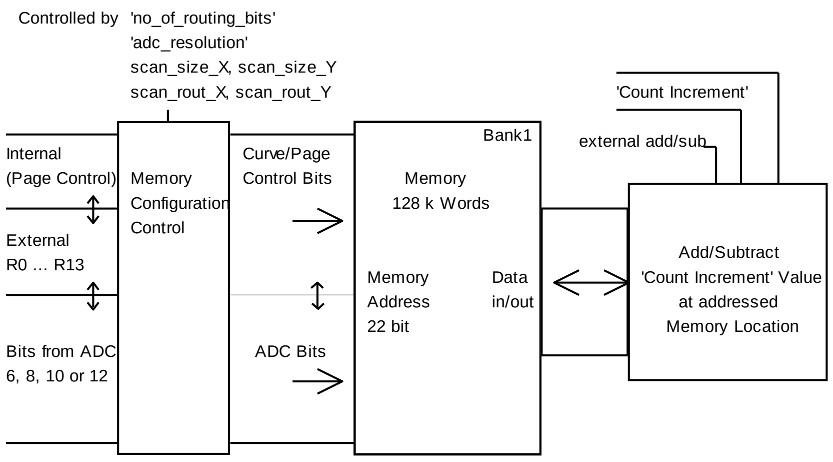
The SPC memory is interpreted as a set of 'pages'. One page contains a number of 'blocks'. One block contains one decay curve. The number of points per block (per curve) is defined by the ADC resolution. The number of blocks per page depends on the number of points per block and on the number of detector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

In the scanning modes of the SPC-7(8)(9)(140,15x,160), 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-130/600/630/15x/160/131 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-15x/160/131) 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-130/600/630/830/140/15x/160/131/930 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-700/730/830/140/15x/160/131/930 modules, Normal operation modes:

The length of the recorded curves is determined by the ADC resolution and can range from 0(SPC-830/140/15x/160/131/930) 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-700/730/830/140/15x/160/930 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-6x0,13x, 15x,160)
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 points16-bits words per block (curve)

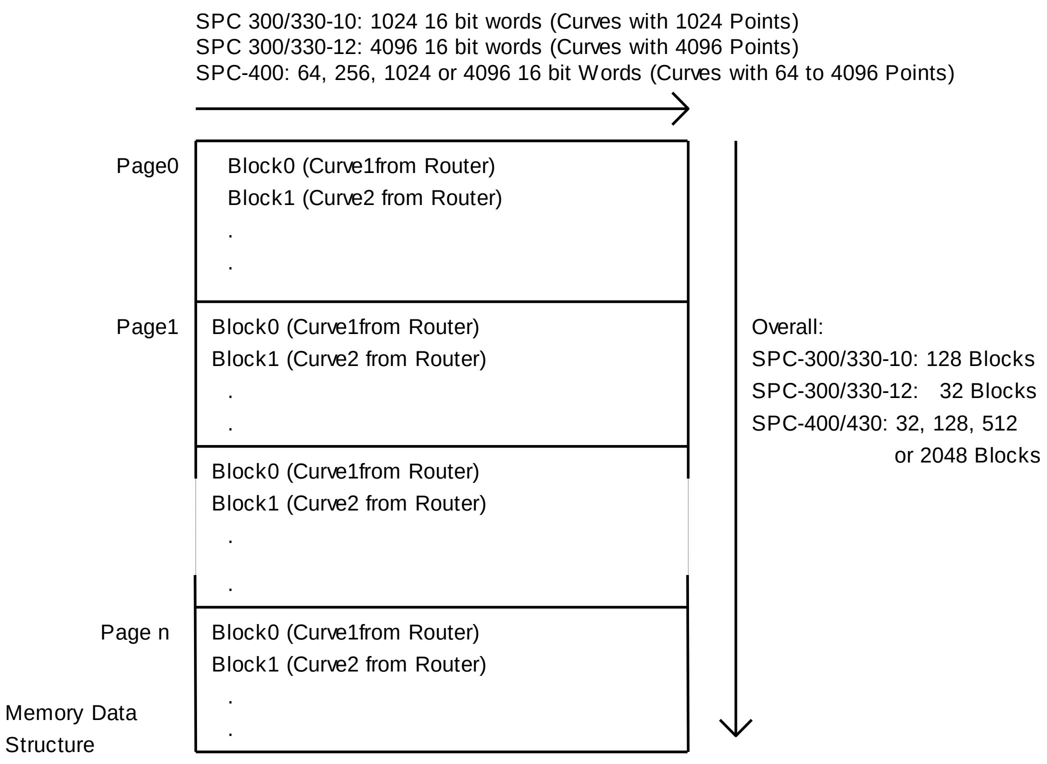
Memory Read/Write Functions

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-7x0/830/140/15x/160/930 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.


SPC 300/330-10: 1024 16 bit words (Curves with 1024 Points)
SPC 300/330-12: 4096 16 bit words (Curves with 4096 Points)
SPC-400: 64, 256, 1024 or 4096 16 bit Words (Curves with 64 to 4096 Points)
Page0
Block0 (Curve1from Router)
Block1 (Curve2 from Router)
Page1
Block0 (Curve1from Router)
Block1 (Curve2 from Router)
Block0 (Curve1from Router)
Block1 (Curve2 from Router)
Page n
Block0 (Curve1from Router)
Block1 (Curve2 from Router)
Memory Data Structure
Overall:
SPC-300/330-10: 128 Blocks
SPC-300/330-12: 32 Blocks
SPC-400/430: 32, 128, 512 or 2048 Blocks

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 SPC7x0/830/140/15x/160/930 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_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)(13x)(140)(830)(15x)(160)(131) and SPC930 modules only :

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

For SPC700(730)(140) (830) (15x)(160) (131) 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 (15x)(160) (131) and SPC930 modules only:

SPC_ SEQ_STOP	0x4000	disarmed (measurement stopped) by sequencer

For SPC15x (160) (131) modules only:

SPC_ SEQ_GAP150	0x2000	Sequencer is waiting for other bank to be armed

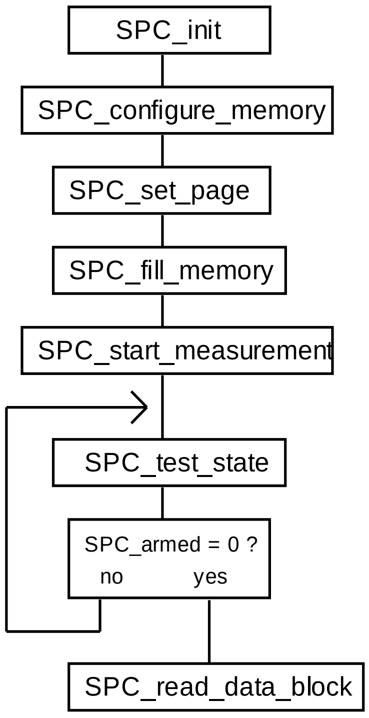
For SPC140 (15x)(160) and SPC830 modules in FIFO_32M mode

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 ore 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.

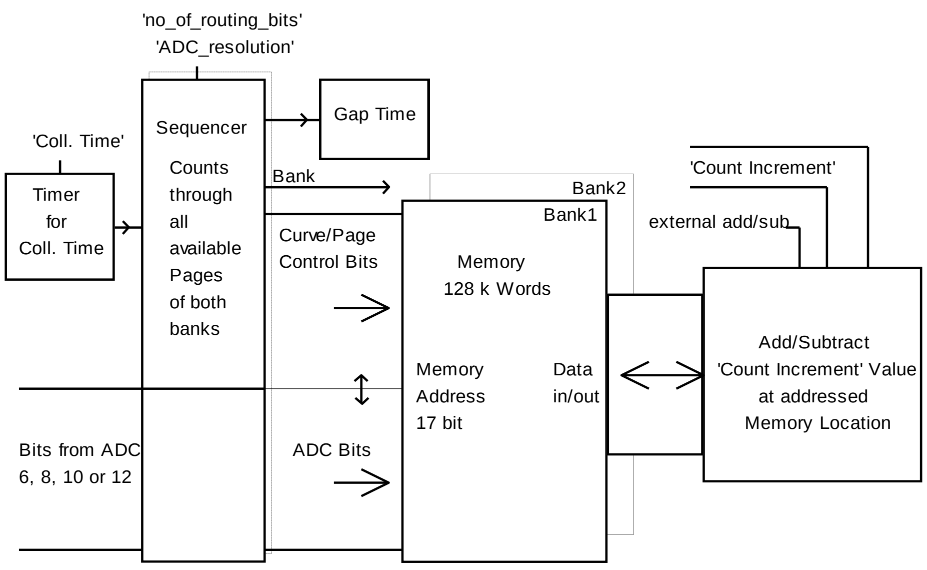


At the beginning, the measurement parameters are read from an initialisation file and send to the SPC module by SPC_init, and the memory is configured by SPC_configure_memory. The memory page in which the data is to be recorded is set by SPC_set_page. SPC_fill_memory is used to clear the data blocks (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-13x, SPC-160 and SPC-15x Sequencer

In the SPC-6x0, SPC-13x, SPC-160 and SPC-15x 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-15x(131)(160) bank is 16 times bigger than SPC-130 bank) and on ADC resolution (for SPC-15x(131)(160) 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.

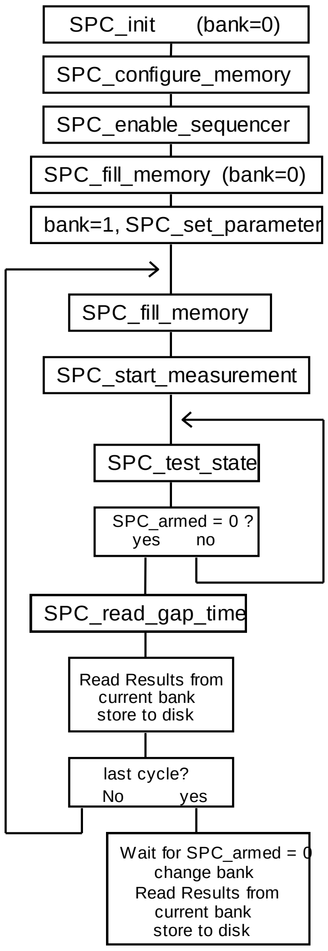
At the beginning, the measurement parameters are read from an initialisation file and sent to the SPC module by SPC_init. The memory is configured by SPC_configure_memory and the sequencer is enabled by **SPC_enable_sequencer**.

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. Than 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).



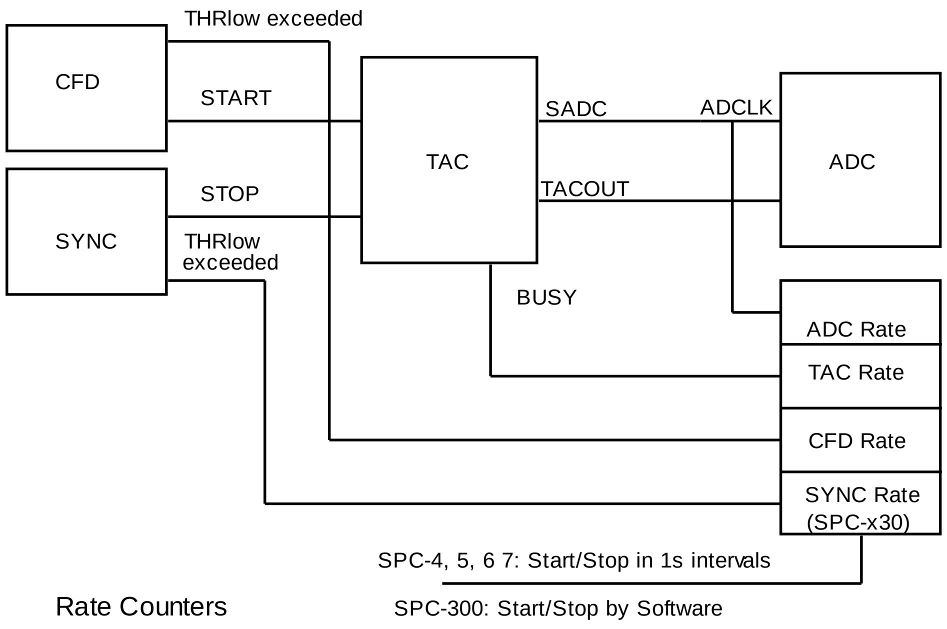
To measure an unlimited number of data blocks, the sequence is repeated in a loop. Thus, after reading the data from the current bank, this bank is cleared (SPC_fill_memory), while the measurement is still running in the alternate bank. The subsequent restarting of the measurement (SPC_start_measurement) 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.



Rate Counters

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-13x, -140, -15x, -160, -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-13x/830/930/15x/160 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.

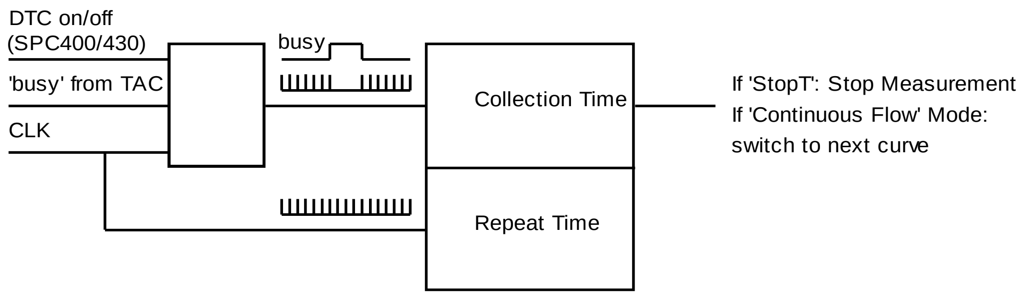
The rates are read by the **SPC_read_rates** function. The results are stored into a structure of the following type:

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 >= 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)
__int64	signed 64-bit integer, quad (I64)
unsigned __int64	unsigned 64-bit integer, unsigned quad (U64)
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]                   ; SPC hardware parameters
cfd_limit_low= 5.0             ; for SPCx3x(140,15x,131,160) -500 ... 0 mV, for SPCx0x 5 ... 80mV, default 5mV
cfd_limit_high= 80.0           ; 5 ... 80 mV, default 80 mV, not for SPC130,140,15x,131,160,930
cfd_zc_level= 0.0              ; for SPCx3x(140,15x,131,160) -96 ... 96 mV, for SPCx0x -10 ... 10 mV
                               ; default 0 mV
cfd_holdoff= 5.0               ; for SPCx0x 5 ... 20 ns, default 5 ns
                               ; for other modules doesn't exist
sync_zc_level= 0.0             ; for SPCx3x(140,15x,131,160) -96 ... 96 mV, for SPCx0x -10 ... 10 mV
                               ; default 0 mV
sync_freq_div= 4               ; for SPC130,140,15x,131,160,930    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 SPC130,140,15x,131,160,930 doesn't exist
sync_threshold= -20.0          ; for SPCx3x(140,15x,131,160) -500 ... -20 mV, 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%, for SPC160,150N(151)  0 ... 50%
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 SPC830,140,15x,131,160,930)
ext_latch_delay= 0             ; 0 ... 255 ns, default 0, for SPC130 doesn't exist
                               ; for SPC140,15x,131,160,930 only values 0,10,20,30,40,50 ns are possible
collect_time= 0.01             ; 0.0001 ... 100000 s, default 0.01 s
repeat_time= 10.0              ; 0.0001 ... 100000 s, default 10.0 s
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, 15x,131,160:  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),
                                        ;    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 SPC15x,160, 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. > C0
                                        ; for SPC131 (= SPC-130-EM), default 0
                                        ;    0 - normal operation (routing in)
                                        ;    1 - FIFO mode 32 bits
scan_size_x=1                           ; for SPC7x0,830,140,15x,160,930 modules in scanning modes 1 ... 65536, default 1
scan_size_y=1                           ; for SPC7x0,830,140,15x,160,930 modules in scanning modes 1 ... 65536, default 1
scan_rout_x=1                           ; number of X routing channels in Scan In & Scan Out modes
                                        ; for SPC7x0,830,140,15x,160,930 modules
                                        ;    1 ... 128, (SPC7x0,830), 1 ... 16 (SPC140,15x,160,930), default 1
scan_rout_y=1                           ; number of Y routing channels in Scan In & Scan Out modes
                                        ; for SPC7x0,830,140,15x,160, 930 modules
                                        ;    1 ... 128, (SPC7x0,830), 1 ... 16 (SPC140,15x,160,930), 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 SPC7x0,140,15x,160)  -        12
                                        ;       14 (12 for SPC7x0,140,15x,160)  -        10
                                        ;       16 (14 for SPC7x0,140,15x,160)  -         8
                                        ;       18 (16 for SPC7x0,140,15x,160)  -         6
                                        ;       20 (18 for SPC140,15x,160)      -         4
                                        ;       22 (20 for SPC140,15x,160)      -         2
                                        ;       24 (22 for SPC140,15x,160)      -         0
scan_polarity=0                         ; for SPC7x0,830,140,15x,160,930 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 SPC140,15x,160,830 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
scan_flyback=0                          ; for SPC7x0,830,140,15x,160,930 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
scan_borders=0                          ; for SPC7x0,830,140,15x,160,930 modules in Scan In mode, default 0
                                        ; bits 15-0 Upper boarder, bits 31-16 Left boarder
pixel_time= 200e-9                      ; pixel time in sec for SPC7x0,830,140,15x,160,930 modules in Scan In mode,
                                        ; 50e-9 ... 1.0, default 200e-9
pixel_clock= 0                          ; source of pixel clock for SPC7x0,830,140,15x,160,930 modules in Scan In mode
                                        ; 0 - internal, 1 - external, default 0
                                        ; for SPC140,15x,160,830 in FIFO_32M mode it disables/enables pixel markers
                                        ; in photons stream
line_compression= 1                     ; line compression factor for SPC7x0,830,140,15x,160,930 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 SPC130,6x0,15x,160,131 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 SPC140,15x,160,131 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 SPC130,15x,160,131,830,930 can be: 1.0 s, 0.25 s, 0.1 s, 0.05 s
                              ; for SPC140 fixed to 50 ms
macro_time_clk= 0             ; macro time clock definition for SPC130,140,15x,160,131,830,930 in FIFO mode
                              ; for SPC130,140,15x,160,131:
                              ;    0 – 50 ns (default), 25ns for SPC15x,160,131 & 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 – 50 ns (default), 1 - SYNC freq.,
                              ; for SPC930:
                              ;    0 – 50 ns (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 SPC830,140,15x,160 & DPC230,930 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 (SPC140,15x,160,830 in FIFO_32M, SPC930 in Camera mode),
                              ;     1 ... 1024, default 1
img_size_y = 1                ; image Y size (SPC140,15x,160,830 in FIFO_32M, SPC930 in Camera mode),
                              ; actually equal to img_size_x (quadratic image)
img_rout_x = 1                ; no of X routing channels (SPC140,15x,160,830 in FIFO_32M, SPC930 in Camera
                              mode),
                              ;     1 ... 16, default 1
img_rout_y = 1                ; no of Y routing channels (SPC140,15x,160,830 in FIFO_32M, 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 SPC140,15x,160,131 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 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_ SPC_OK	0	no error
INIT_ SPC_NOT_DONE	-1	init not done
INIT_ SPC_WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_ SPC_WRONG_MOD_ID	-3	wrong module identification code
INIT_ SPC_HARD_TEST_ERR	-4	hardware test failed
INIT_ SPC_CANT_OPEN_PCI_CARD	-5	cannot open PCI card
INIT_ SPC_MOD_IN_USE	-6	module in use (locked) - cannot initialise
INIT_ SPC_WINDRVR_VER	-7	incorrect WinDriver version
INIT_ SPC_WRONG_LICENSE	-8	corrupted license key
INIT_SPC_FIRMWARE_VER	-9	incorrect firmware version of DPC/SPC module
INIT_SPC_NO_LICENSE	-10	license key not read from registry
INIT_SPC_LICENSE_NOT_VALID	-11	license is not valid for SPCM DLL
INIT_SPC_LICENSE_DATE_EXP	-12	license date expired
INIT_SPC_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
#define M_DPC230            230         DPC-230
#define M_SPC131            131         130 with extended memory = SPC-130-EM
#define M_SPC151            151         150N = 150 with faster discriminators and reduced timing wobble
#define M_SPC160            160         160 - special module for optical tomography
```

----

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 */
#define SPC_SIMUL131        131     /* simulation mode of SPC131 (= SPC-130-EM) module */
#define SPC_SIMUL151        151     /* simulation mode of SPC150N module */
#define SPC_SIMUL160        160     /* simulation mode of SPC160 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(140,15x,160,131) -500 ... 0 mV, for SPCx0x 5 ... 80 mV
float cfd_limit_high	5 ... 80 mV, default 80 mV, not for SPC130,140,15x,160,131,930
float cfd_zc_level	SPCx3x(140,15x,160,131) -96 ... 96 mV, SPCx0x -10 ... 10 mV
float cfd_holdoff	SPCx0x: 5 ... 20 ns, other modules: no influence
float sync_zc_level	SPCx3x(140,15x,160,131): -96 ... 96 mV, SPCx0x: -10 ...10 mV
float sync_holdoff	4 ... 16 ns (SPC130,140,15x,160,131,930: no influence)
float sync_threshold	SPCx3x(140,15x,160,131): -500 ... -20 mV, SPCx0x: no influence
float tac_range	50 ... 5000 ns
short sync_freq_div	1,2,4,8,16 (SPC130,140,15x,160,131,930: 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, 140,15x,160,131, 930)
short ext_latch_delay	0 ... 255 ns, SPC130: no influence
	SPC140,15x,160,131,930: only values 0,10,20,30,40,50 ns are possible
float collect_time	0.0001 ... 100000 s, default 0.01 s
float display_time	0.0001 ... 100000 s, default 0.01 s
float repeat_time	0.0001 ... 100000 s, default 0.01 s
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 SPC130,600,630,15x,160,131: 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(140,131, 151,160)

	- bit 6 - in FIFO_32M mode, = 0 (default) Marker 3 not used, = 1 waiting for Marker 3 to start collection timer, (used in accumulated Mosaic measurements) - bit 7 - in FIFO_32M mode, = 0 (default) Frame pulses on Marker 2, = 1 Frame pulses on Marker 3, - bits 8 - 11 – enable (1) / disable (0), default 0 of recording Markers 0-3 entries in FIFO mode - bits 12 - 15 - active edge 0 (falling), 1 (rising), default 0 of Markers 0-3 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 SPC15x,160, 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 for SPC131, default 0 0 - normal operation (routing in) 1 - FIFO mode 32 bits
unsigned long scan_size_x;	for SPC7x0,830,140,15x,160,930 modules in scanning modes 1 ... 65536, default 1
unsigned long scan_size_y;	for SPC7x0,830,140,15x,160,930 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 SPC7x0,830,140,15x,160,930 modules 1 ... 128, (SPC7x0,830), 1 ... 16 (SPC140,15x,160,930), default 1
unsigned long scan_rout_y;	number of Y routing channels in Scan In & Scan Out modes for SPC7x0,830,140,15x,160,930 modules 1 ... 128, (SPC7x0,830), 1 ... 16 (SPC140,15x,160,930), 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 current adc_resolution: 12 (10 for SPC7x0,140,15x,160) - 12 14 (12 for SPC7x0,140,15x,160) - 10 16 (14 for SPC7x0,140,15x,160) - 8 18 (16 for SPC7x0,140,15x,160) - 6 20 (18 for SPC140,15x,160) - 4 22 (20 for SPC140,15x,160) - 2 24 (22 for SPC140,15x,160) - 0

unsigned long scan_flyback;	for SPC7x0,830,140,15x,160,930 modules in Scan Out or Rout Out mode, default & minimum = 1 bits 15-0 Flyback X in number of pixels bits 31-16 Flyback Y in number of lines
unsigned long scan_borders;	for SPC7x0,830,140,15x,160,930 modules in Scan In mode, default 0 bits 15-0 Upper boarder, bits 31-16 Left boarder

unsigned short scan_polarity;	for SPC7x0,830,140,15x,160,930 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 SPC140,15x,160,830 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 SPC7x0,830,140,15x,160,930 modules in Scan In mode,
	pixel clock source, 0 - internal,1 - external, default 0
	for SPC140,15x,160,830 in FIFO_32M mode it disables/enables pixel markers
	in photons stream

unsigned short line_compression;	line compression factor for SPC7x0,830,140,15x,160,930 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 SPC130,6x0,15x,160,131 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 SPC140,15x,160,131 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 SPC7x0,830,140,15x,160,930 modules in Scan In mode, 50e-9 ... 1.0, default 200e-9
unsigned long ext_pixclk_div;	divider of external pixel clock for SPC7x0,830,140,15x,160,930 modules in Scan In mode, 1 ... 0x3fe, default 1

float rate_count_time;	rate counting time in sec default 1.0 sec
	for SPC130,830,930,15x,160,131 can be: 1.0 s, 0.25 s, 0.1s, 0.05 s
	for SPC140 fixed to 50 ms

short macro_time_clk;	macro time clock definition for SPC130,140,15x,160,131,830,930 in FIFO mode for SPC130, 140,15x,160,131: 0 – 50 ns (default), 25 ns for SPC15x,160,131 & 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 – 50 ns (default), 1 - SYNC freq., for SPC930: 0 – 50 ns (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 SPC830,140,15x,160,131,930 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 (SPC140,15x,160,830 in FIFO_32M mode, SPC930 in Camera mode),
	1 ... 1024, default 1
unsigned long img_size_y;	image Y size (SPC140,15x,160,830 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 (SPC140,15x,160,830 in FIFO_32M mode)
	1 ... 16, default 1
unsigned long img_rout_y;	no of Y routing channels (SPC140,15x,160,830 in FIFO_32M 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, 15x,160,131 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
unsigned long 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 long 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 18-19 - not used
	bits 20-23 - active channels of TDC1 for DPC-330 Hardware Histogram modes
	bits 24-27 - active channels of TDC2 for DPC-330 Hardware Histogram modes
	bits 28-31 - not used
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

----

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_eeprom_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' filea 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 SPC6x0(13x)(140)(830)(15x)(160) modules only :

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

For SPC7x0(140) (830) (15x) (131) (160) 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 (15x) (131) and SPC160 modules only:

SPC_ SEQ_STOP	0x4000	disarmed (measurement stopped) by sequencer

For SPC15x (131) (160) modules only:

SPC_ SEQ_GAP150	0x2000	Sequencer is waiting for other bank to be armed

For SPC140 (15x) (160) 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 <u>before</u> 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-13x/830/930/15x/160 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 SPC6x0/13x/15x/160 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 SPC6x0/130/150/131 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-830/140/15x/930/160 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-13x/140/830/6x0/930/15x/160 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-6x0/13x/830/140/930/15x/160 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-6x0/13x/830/140/930/15x/160 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-6x0/13x/830/140/930/15x/160 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-6x0/13x/830/140/930/15x/160 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-6x0/13x/830/140/930/15x/160 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-6x0/13x/830/140/930/15x/160 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-6x0/130/131) in SCAN_IN mode:

If the measurement was started in SCAN_IN mode, 1st 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. 2nd call will stop the measurement without waiting for the end of frame.

For SPC-6x0/13x/830/140/930/15x/160 in FIFO mode:

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

For SPC-830/140/15x/160 in FIFO_32M mode (Fifo Image):

The functionality is added which enables higher level software to stop the measurement after collecting the whole frames.
Stopping the measurement requires two calls of SPC_stop_measurement.
1st call of SPC_stop_measurement:
- photon collection is not yet stopped
- current position of write pointer of FIFO is remembered as a stop position
- subsequent SPC_read_fifo calls will return 1 when this place in FIFO is reached

- this is a sign for higher level software that all photons collected up to the stop moment were read.

From this moment photons should be read until a frame marker will appear in photons stream - then higher level software should call SPC_stop_measurement again to stop collecting photons.

2nd call of SPC_stop_measurement:

- 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 (memory bank for the SPC-6x0/13x/15x/160)
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
long block_length	Number of 16-bits words per one block (curve)

----

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-130/6x0/150/131. 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 2nd 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 13x/15x/160 and 6x0 - normal operation (routing in) with sequencer
- scanning modes (Scan In, Scan Out) for SPC modules 7x0, 830, 140, 930, 15x, 160
- Continuous flow in Scan In mode for SPC-15x/160 module
- 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-830/140/930/15x/131/160) * 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-6x0/140/15x/131/160) (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-7/8/9/140/15x/160, 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.

<u>SPC-13x/6x0/15x/160 modules</u>:

The length of the recorded curves is determined by the ADC resolution and can range from 64 (0 for SPC-15x/131/160) to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

<u>SPC-13x/6x0/15x/160 modules in the Histogram modes</u>:

The length of the recorded curves is determined by the ADC resolution and can range from 64 (0 for SPC-15x/131/160) to 4096. Therefore, the number of complete measurement data sets (or 'pages') depends on the ADC resolution and the number of routing bits used.

<u>SPC-13x/6x0/830/140/15x/160/930 modules in the Fifo modes</u>:

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.

<u>SPC-7x0/830/140/15x/131/160 modules, Normal operation modes</u>:

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.

<u>SPC-7x0/830/140/15x/160/930 modules, Scanning modes</u>:

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(13x/15x/160))
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 points16-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 (memory bank for the SPC-6x0/13x/15x/160)
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
long 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 (memory bank for the SPC-6x0/13x/15x/160)
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
long 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 (1) no errors, <0 error code

The procedure reads data from the FIFO memory of SPC modules SPC-6x0/13x/830/140/15x/160 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 SPC13x/830/140/15x/160/930 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.

For SPC830/140/15x/160 modules in FIFO_32M (FIFO IMAGE) mode:

works as above with a difference after calling SPC_stop_measurement - function will return 1 (instead of 0) when photons are read up to stop pointer in FIFO - see explanation in SPC_stop_measurement function description.

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 (memory bank for the SPC-6x0/13x/15x/160)
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
long 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 (memory bank for the SPC-6x0/13x/15x/160)
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
long 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 * frames_per_page * (last_page – 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 SPC7x0/830/140/15x/160 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 (memory bank for the SPC-6x0/13x/15x/160)
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
long 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_minfo.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 1st module’s data the first byte of 2nd 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 SPC6x0/13x/15x/160 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-13x/15x/160 module types, when DLL parameter MODE = NORMAL and sequencer is enabled
- mode 'Scan Sync In': SPC-7x0/830/140/930/15x/160 module types, when DLL parameter MODE = SCAN_IN and sequencer is enabled
- mode 'Scan Sync Out': SPC-7x0/830/140/930/15x/160 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-6x0/830/13x/140/930/15x/160).

Functions to Manage Photons Streams:

```
-----------------------------------------------------------------------------------------------------

short CVICDECL SPC_init_phot_stream (short fifo_type, char * spc_file, short files_to_use,
                                              short stream_type, short what_to_read);

-----------------------------------------------------------------------------------------------------
```

Input parameters:

fifo_type	2 (FIFO_48), 3 (FIFO_32), 4 (FIFO_130), 5 (FIFO_830), 6 (FIFO_140), 7 (FIFO_150), 9 (FIFO_IMG)
* spc_file	file path of the 1st spc file
files_to_use	number of subsequent spc files to use, 1 ...999 or –1 for all files
stream_type	bit 0 = 1- stream of BH .spc files (1st entry in each file contains MT clock, flags & rout. chan info)
	bit 0 = 0 - no special meaning of the first entry in the file
	bit 9 = 1 - stream contains marker entries (FIFO_IMG mode)
	bit 10 = 1 - stream contains raw data (diagnostics only)
what_to_read	defines which entries will be extracted
	bit 0 = 1 read valid photons, bit 1 = 1 read invalid photons,
	bit 2 = 1 read markers 0 (pixel), bit 3 = 1 read markers 1 (line),
	bit 4 = 1 read markers 2 (frame), bit 5 = 1 read markers 3

Return value: >=0: stream handle, no errors, <0: error code

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, 1st 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 1st 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 1st file will be extracted, the 2nd 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’. It accepts only streams of spc files.

To extract photons from buffered photons streams SPC_get_photons_from_stream function can be used.

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_fifo_init_vars (short mod_no, short *fifo_type,
                                        short *stream_type, int *mt_clock,
                                        unsigned int *spc_header);
```

Input parameters:

mod_no	0 ... 7, SPC module number
fifo_type	pointer to variable which will be set to FIFO type of module mod_no
stream_type	pointer to variable which will be set to initial value of stream type

mt_clock	pointer to variable which will be set to macro time clock of the module mod_no
spc_header	pointer to variable which will be set to .spc file header (1st word of .spc file), if saving .spc files is required

Return value:

0 no errors, <0 error code

This procedure sets variables to be used as input parameters for SPC_init_buf_stream function. It can also prepare .spc file header (1st word of .spc file), if saving .spc files is required. The procedure is intended to use directly before starting FIFO measurement to initialize 'buffered' stream.

If photons are added to the stream not from running FIFO measurement, but e.g. .spc files, then input parameters for SPC_init_buf_stream must be taken from .spc file header.

```
short CVICDECL SPC_init_buf_stream (short fifo_type, short stream_type,
                                     short what_to_read, int mt_clock,
                                     unsigned int start01_offs);
```

Input parameters:

fifo_type 2 (FIFO_48), 3 (FIFO_32), 4 (FIFO_130), 5 (FIFO_830), 6 (FIFO_140), 7 (FIFO_150), 9 (FIFO_IMG)

stream_type bit 0 has no special meaning for buffered streams, is set to 1

bits 1,2,8 = 0, used only for DPC-230 FIFO data

bit 9 = 1 - stream contains marker entries (FIFO_IMG mode)

bit 10 = 1 - stream contains raw data (diagnostics only)

bit 12 = 1 – indicates buffered stream type

bit13 = 1 – stream buffer freed automatically after extracting photons from it

= 0. stream buffer freed using SPC_close_phot_stream

what_to_read defines which entries will be extracted

bit 0 = 1 read valid photons, bit 1 = 1 read invalid photons,

bit 2 = 1 read markers 0 (pixel), bit 3 = 1 read markers 1 (line),

bit 4 = 1 read markers 2 (frame), bit 5 = 1 read markers 3

mt_clock macro time clock, for SPC modules 0.1ns units, for DPC-230 1fs units (femto, 1e-15)

start01_offs 0, no meaning for SPC modules, for DPC-230 Start01 offset from ACAM register 10

Return value: >=0: stream handle, no errors, <0: error code

The procedure is needed to initiate the process of extracting photons from a stream of photons placed into PC memory buffers called longer 'buffered' stream.

To get input parameters needed to call SPC_init_buf_stream (fifo_type, stream_type, mt_clock) call SPC_get_fifo_init_vars function, when you are ready to start the FIFO measurement.

If the photons are taken from .spc files, input parameters should be taken from .spc file header (1st word).

Set bit 9 of 'stream_type' if the file contains markers (was created in FIFO_IMG mode or FIFO mode with enabled markers).

Buffers are allocated/reallocated automatically while adding photons to the stream.

The buffers can be freed in two ways depending on an option FREE_BUF_STREAM (bit 13 in 'stream_type' parameter).

If bit 13 is not set, the buffers are freed when the stream is closed (SPCM_close_phot_stream).

If bit 13 is set, the buffer will be freed, when, during SPC_get_photons_from_stream call, all photons from the current buffer are extracted.

After this it will be not possible to use the buffer again, for example to get data from it using SPC_get_buffer_from_stream function.

FREE_BUF_STREAM option is recommended for long measurements with lots of data read from FIFO, which could make buffers allocated space very (too) big.

If the photons rate or measurement time is not very big/long, buffers can stay allocated and another extract action can be started (with different start/stop condition) or buffers contents can be stored in .spc file.

User can add photons to the stream buffers by using function:

SPC_add_data_to_stream – photons are taken from the buffer
(input buffer can be filled from .spc file)

SPC_read_fifo_to_stream - photons are read from FIFO during running FIFO measurement.

Adding photons to the stream or extracting photons can be done also during running measurement. During extracting or adding photons the stream is locked. Only one thread can access the thread safe stream at a time. If a thread requests the access to stream resources while another has it, the second thread waits in this function until the first thread releases the stream.

Use function SPC_get_photons_from_stream to extract photons information from the stream buffers.

'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 streams 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_photons_from_stream and so on).

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_photons_from_stream to extract subsequent photons from the stream.

Using SPC_stream_start_condition and SPC_stream_stop_condition user can define start/stop condition of extracting photons, which can be specific macro time and/or occurrence of markers or routing channels.

As long as stream buffers are not freed user can call SPC_reset_stream and then extract photons again with another start/stop condition.

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_add_data_to_stream (short stream_hndl, void * buffer,
                                                            unsigned int bytes_no);

-------------------------------------------------------------------------------------------------------
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream
buffer	pointer to the buffer containing photons data to be added to the stream
bytes_no	no of bytes to be added to the stream, > 0, max = STREAM_MAX_BUF_SIZE

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

The procedure can be used to add photons data to the opened 'buffered' photons stream defined by handle 'stream_hndl'.

Photons data in the input buffer can be taken from .spc files or read from module's FIFO.

Data are added to DLL internal buffers which have size between STREAM_MIN_BUF_SIZE and STREAM_MAX_BUF_SIZE. Internal buffers are allocated by DLL when required.

Maximum size of allocated stream buffers is equal STREAM_MAX_SIZE32 for 32-bit DLL and STREAM_MAX_SIZE64 for 64-bit DLL.

The buffers are freed, when the stream is closed (SPCM_close_phot_stream) or when, during SPC_get_photons_from_stream call, all photons from the current buffer are extracted (if an option FREE_BUF_STREAM (bit 13 in 'stream_type') is set).

Procedure returns error, if 'stream_hndl' is not the handle of the stream opened using SPC_init_buf_stream function.

```
short CVICDECL SPC_read_fifo_to_stream (short stream_hndl, short mod_no,
                                        unsigned long *count);
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream
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 added to the stream 'stream_hndl'

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

The procedure can be used to add photons data to the opened 'buffered' photons stream defined by handle 'stream_hndl'. Photons are read from the FIFO on SPC module 'mod_no' during running FIFO measurement.

The 'Count' variable is filled on exit with the number of 16-bit words added to the stream.

See also the description of SPC_read_fifo procedure because it is called internally.

Photons data read from FIFO are added to DLL internal buffers which have size between STREAM_MIN_BUF_SIZE and STREAM_MAX_BUF_SIZE. Internal buffers are allocated by DLL when required.

Maximum size of allocated stream buffers is equal STREAM_MAX_SIZE32 for 32-bit DLL and STREAM_MAX_SIZE64 for 64-bit DLL.

The buffers are freed, when the stream is closed (SPCM_close_phot_stream) or when, during SPC_get_photons_from_stream call, all photons from the current buffer are extracted (if an option FREE_BUF_STREAM (bit 13 in 'stream_type') is set).

Procedure returns error, if 'stream_hndl' is not the handle of the stream opened using SPC_init_buf_stream function.

```
short CVICDECL SPC_get_photons_from_stream (short stream_hndl,
                                            PhotInfo64 *phot_info, int *phot_no);
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream
phot_info	pointer to the buffer containing photons data extracted from the stream
phot_no	pointer to variable which: - on input contains required number of photons to extract from buffered stream 'stream_hndl', 1 ... 1000000 -on output will be filled with number of photons extracted from buffered stream 'stream_hndl'

Return value: >= 0: no errors, 1 – stop condition found, 2 – end of the stream reached

<0: error code

The procedure is used to extract photons data from the opened 'buffered' photons stream defined by handle 'stream_hndl'. Photons data are packed to the structures PhotInfo64 (defined in spcm_def.h file) in 'phot_info' buffer.

The 'phot_no' variable is filled on exit with the number of photons extracted from the stream.

Extracting photons can be done also during running measurement. During extracting or adding photons the stream is locked. Only one thread can access the thread safe stream at a time. If a thread requests the access to stream resources while another has it, the second thread waits in this function until the first thread releases the stream.

Using SPC_stream_start_condition and SPC_stream_stop_condition user can define start/stop condition of extracting photons, which can be specific macro time and/or occurrence of markers or routing channels.

User can define start and stop condition for extracting photons using functions SPC_stream_start(stop)_condition. The condition can be a specified macro time value and/or occurrence of specific markers and/or routing channels. See description of SPC_stream_start(stop)_condition functions.

Photons extracted to 'phot_info' buffer can be saved to .ph file.

First 4 bytes of .ph file it is a header (the same as for .spc files). Call SPC_get_fifo_init_vars to get the header value. After the header subsequent PhotInfo64 photons structures are stored. Such file can be used in SPCM software as an input file in 'Convert FIFO Files' panel.

```
    short CVICDECL SPC_stream_start_condition (short stream_hndl,
                                    double start_time, unsigned int start_OR_mask,
                                    unsigned int start_AND_mask);
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream
start_time	macro time (in sec) at which extracting photons will start (during call to SPC_get_photons_from_stream)
start_OR_mask	bitwise OR mask used to define start of extracting photons from the stream (in addition to 'start_time'), bits 31-28 – markers M3-M0, bits 27-0 - routing channels 27-0
start_AND_mask	bitwise AND mask used to define start of extracting photons from the stream (in addition to 'start_time'), bits 31-28 – markers M3-M0, bits 27-0 - routing channels 27-0

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

The procedure is used to define start of extracting photons from buffered stream 'stream_hndl' during SPC_get_photons_from_stream call.

Start_time and OR/AND masks can be used all together.

All photons (markers) are ignored until the macro time in the stream reaches 'start_time' value. From this moment appearance of specified markers/channels is tested according to start_OR(AND)_mask.

Start condition is found when minimum one of markers/channels defined in start_OR_mask appears in the stream (since 'start_time').

Start condition is also found when all of markers/channels defined in start_AND_mask appeared in the stream (since 'start_time').

SPC_get_photons_from_stream returns an error, when start condition cannot be found in the stream.

While extracting photons during running measurement, start condition (if defined) can be found later after reading new portion of photons data from FIFO to the stream (using SPC_read_fifo_to_stream).

```
-------------------------------------------------------------------------------------------------------

short CVICDECL SPC_stream_stop_condition (short stream_hndl,
                                           double stop_time, unsigned int stop_OR_mask,
                                           unsigned int stop_AND_mask);

-------------------------------------------------------------------------------------------------------
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream
stop_time	macro time (in sec) at which extracting photons will stop (during call to SPC_get_photons_from_stream) (if stop masks are not defined)
stop_OR_mask	bitwise OR mask used to define stop of extracting photons from the stream (in addition to 'stop_time'), bits 31-28 – markers M3-M0, bits 27-0 - routing channels 27-0
stop_AND_mask	bitwise AND mask used to define stop of extracting photons from the stream (in addition to 'stop_time'), bits 31-28 – markers M3-M0, bits 27-0 - routing channels 27-0

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

The procedure is used to define stop of extracting photons from buffered stream 'stream_hndl' during SPC_get_photons_from_stream call.

Stop_time and OR/AND masks can be used all together.

All photons (markers) are extracted until the macro time in the stream reaches 'stop_time' value. From this moment appearance of specified markers/channels is tested according to stop_OR(AND)_mask.

Stop condition is found when minimum one of markers/channels defined in stop_OR_mask appears in the stream (since 'stop_time').

Stop condition is also found when all of markers/channels defined in stop_AND_mask appeared in the stream (since 'stop_time').

SPC_get_photons_from_stream returns an error, when stop condition cannot be found in the stream.

While extracting photons during running measurement, stop condition (if defined) can be found later after reading new portion of photons data from FIFO to the stream (using SPC_read_fifo_to_stream).

```
-------------------------------------------------------------------------------------------------------

short CVICDECL SPC_stream_reset (short stream_hndl);

-------------------------------------------------------------------------------------------------------
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream

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

The procedure resets buffered stream 'stream_hndl' to the state before extracting the photons without affecting stream's internal data buffers. After this user can define new stream's start/stop condition and extract photons once more from the beginning of the stream using new conditions. But, attention, this is possible only when stream's data buffers are not freed after extracting photons.

An option FREE_BUF_STREAM (bit 13 in 'stream_type' parameter while initializing the stream using SPC_init_buf_stream function) defines a way in which stream's buffers are freed. If it is set, the buffer is freed, when, during SPC_get_photons_from_stream call, all photons from the current buffer are extracted.

After this it will be not possible to use the buffer again, for example to get data from it using SPC_get_buffer_from_stream function.

Therefore SPC_reset_stream can be used when FREE_BUF_STREAM is not used – buffers are not freed and extracting photons can be repeated.

```
-------------------------------------------------------------------------------------------------------
    short CVICDECL SPC_get_stream_buffer_size (short stream_hndl,
                                            unsigned short buf_no, unsigned int *buf_size);
-------------------------------------------------------------------------------------------------------
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream
buf_no	no of stream's buffer to be copied, 0 ... number of stream's buffers (use SPC_get_phot_stream_info to get it)
buf_size	pointer to variable which will be filled with the size of 'buf_no' buffer

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

Before calling SPC_get_buffer_from_stream user must know the buffer size and should allocate big enough data buffer.

The SPC_get_stream_buffer_size procedure is used to get size of stream's internal buffer number 'buf_no'. Stream is defined by handle 'stream_hndl'. The procedure works only with 'buffered' streams.

Call SPC_get_phot_stream_info to get info about current number of stream buffers (field no_of_buf of PhotStreamInfo structure).

```
short CVICDECL SPC_get_buffer_from_stream (short stream_hndl,
                                        unsigned short buf_no, unsigned int *buf_size,
                                        char * data_buf, short free_buf);
```

Input parameters:

stream_hndl	handle of the initialised 'buffered' photons stream
buf_no	no of stream's buffer to be copied, 0 ... number of stream's buffers (use SPC_get_phot_stream_info to get it)
buf_size	pointer to variable which: on input gives 'data_buf' size, on output will be filled with number of bytes copied to the 'data_buf' buffer
data_buf	pointer to the buffer which will be filled with the contents of stream's buffer 'buf_no'
free_buf	0,1 - if = 1, stream's buffer 'buf_no' is freed on procedure exit

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

The procedure is used to get contents of stream's internal buffers with photons data to the buffer 'data_buf'. Stream is defined by handle 'stream_hndl'. The procedure works only with 'buffered' streams.

The procedure fills 'data_buf' with contents of stream's buffer 'buf_no'.

'data_buf' must be allocated with minimum 'buf_size' bytes.

Use SPC_get_stream_buffer_size procedure to get size of buffer 'buf_no', which means the required 'buf_size' value.

On exit 'buf_size' is set to real number of bytes copied to 'data_buf'.

On demand, the buffer 'buf_no' can be freed on procedure exit, when 'free_buf' parameter equals 1.

Call SPC_get_phot_stream_info to get info about current number of stream buffers (field no_of_buf of PhotStreamInfo structure).

Be aware of option FREE_BUF_STREAM (bit 13 in 'stream_type' parameter while initializing the stream using SPC_init_buf_stream function) It defines a way in which stream's buffers are freed. If it is set, the buffer is freed, when, during SPC_get_photons_from_stream call, all photons from the current buffer are extracted.

After this it will be not possible to get data from it using SPC_get_buffer_from_stream function.

Data taken from stream' buffers can be stored in .spc file for future use (for example in SPCM application panel 'Convert FIFO files').

First 4 bytes of .spc file it is a header - call SPC_get_fifo_init_vars to get the header value. After the header, subsequent stream's buffers should be stored.

Other Functions:

----

short CVICDECL SPC_get_error_string (short error_id, char * dest_string, short max_length);

Input parameters:

error_id	SPC DLL error id (0 – number of SPC errors-1) (see spcm_def.h file)
*dest_string	pointer to destination string
max_length	max number of characters which can be copied to 'dest_string'

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

The procedure copies to 'dest_string' the string which contains the explanation of the SPC DLL error with id equal 'error_id'. Up to 'max_length characters will be copied.

Possible 'error_id' values are defined in the spcm_def.h file.

```
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_typepointer 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 1st 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.

===