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

SPC- 32 Bits Dynamic Link Libraries

User Manual

Version 2.2

June 2001

Introduction

The SPC 32 bits Dynamic Link Library contains all functions to control the whole family of SPC modules. The functions work under Windows 95 and Windows NT. 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:

SPCDLL32.DLL	dynamic link library main file
SPCDLL32.LIB	import library file for Microsoft Visual C/C++, Borland C/C++, Watcom C/C++ and Symantec C/C++ compilers
SPC_DEF.H	Include file containing Types definitions, Functions Prototypes and Pre-processor statements
SPC400.INI	SPC DLL initialisation file
SPCDLL32.DOC	This description file
USE_SPC.XXX	Set of files to compile and run a simple example of using SPC DLL functions. Source file of the example is the file use_spc.c. Example was originally prepared under Borland C/C++ v.4.52. For use in other compilers choose correct import library file to link.

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

SPC-DLL Functions list

The following functions are implemented in the SPC-DLL:

Initialisation functions:

SPC_init
SPC_test_id
SPC_get_init_status
SPC_set_mode
SPC_get_mode

Setup functions:

SPC_get_parameter
SPC_set_parameter
SPC_get_parameters
SPC_set_parameters
SPC_get_eeprom_data
SPC_write_eeprom_data
SPC_get_adjust_parameters
SPC_set_adjust_parameters

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

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

Other functions:

SPC_get_error_string
SPC_close

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. This is accomplished by the function **SPC_init**. This function

- 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
- sends the parameter values to the SPC control registers
- performs a hardware test of the SPC module

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file spc400.ini or to start with spc400.ini and to introduce the desired changes.

```
;    SPC400 initialisation file
;    SPC parameters have to be included in .ini file only when parameter
;    value is different from default.
;    only baseadr in spc_base section  is required
;    other parameters can have default values
[spc_base]
baseadr=  0x380                          ; base I/O address   (0 ... 0x3E0,default 0x380)
```

```
simulation = 0                      ; 0 - hardware mode(default) ,
                                    ; >0 - simulation mode (see spc_def.h for possible values)
pci_bus_no= 0                       ; PCI bus on which SPC module will be looking for
                                    ;   0 - 255, default 0 ( 1st found PCI bus with SPC modules will be scanned)
pci_card_no= 0                      ; number of module on PCI bus if PCI version of SPC module
                                    ;   0 - 7, default -1 ( ISA module)

[spc_module]                        ; SPC hardware parameters
cfd_limit_low= 5.0                  ; for SPCx3x -500 .. -20mV ,for SPCx0x  5 .. 80mV
                                    ; default 5mV
cfd_limit_high= 80.0                ; 5 ..80 mV, default 80 mV, for SPC130 doesn't exist
cfd_zc_level= 0.0                   ;for SPCx3x -96 .. 96mV ,for SPCx0x -10 .. 10mV
                                    ; default 0mV
cfd_holdoff= 5.0                     ; for SPCx0x 5 .. 20 ns , default 5ns
                                    ; for SPCx3x (130,230) doesn't exist
sync_zc_level= 0.0                  ; for SPCx3x -96 .. 96mV ,for SPCx0x -10 .. 10mV
                                    ; default 0mV
sync_freq_div= 4                    ; for SPC130,230 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 SPC130,230 doesn't exist
sync_threshold= -20.0               ; for SPCx3x -500 .. -20mV ,default -20 mV
                                    ; for SPCx0x doesn't exist

tac_range= 50.0                      ; 50 .. 2000 ns , default 50 ns
tac_gain= 1                         ; 1 .. 15 ,default 1
tac_offset=0.0                       ; 0 .. 100% ,default 0%
tac_limit_low=                      10.0   ; 0 .. 100% ,default 10%
tac_limit_high= 80.0                ; 0 .. 100% ,default 80%

adc_resolution= 10                  ; for SPC300(330) fixed to 10 or 12
                                    ; for SPC401(431) fixed to 12
                                    ; for SPC402(432) fixed to 8
                                    ; for other SPC modules 6,8,10,12 bits, default 10
ext_latch_delay= 0                  ; 0 ..255 , default 0
                                    ; for SPC130/230 doesn't exist

collect_time= 0.01                  ; for SPC300(330) 0.01   .. 100000s
                                    ; for other SPC modules 0.0001 .. 100000s , default 0.01s

display_time= 1.0                   ; for SPC300(330) 0.01   .. 100000s , default 1.0s
                                    ; for other SPC modules display timer doesn't exist
repeat_time=  10.0                  ; for SPC300(330) 0.01   .. 100000s
                                    ; for other SPC modules 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                     ; for SPC300(330) 0,32,64,128,256 ,default 0
                                    ; for other SPC modules subsequent values have
                                    ;       different meaning  0, 1/64, 1/32, 1/16, 1/8
count_incr= 1                       ; 1 .. 255 , default 1
mem_bank= 0                         ; for SPC400(130,230,430,600,630) 0 , 1 , default 0
                                    ; for other SPC modules always 0
dead_time_comp= 1                   ; for SPC300(330) always 1
                                    ; for other SPC modules 0 , 1 , default 1
scan_control= 0                     ; for SPC505(535,506,536) , default 0xf000
                                    ;   16-bit hex value to control scanning in SPC505(535)
                                    ;    or routing in SPC506(536)
                                    ;   bits are defined in spc_def.h file
                                    ; for other SPC modules always 0
routing_mode= 0                     ; for SPC230 0 , 1 , default 0
                                    ; for other SPC modules always 0
tac_enable_hold= 50.0               ; for SPC230  10.0 .. 265.0 ns, default 50.0 ns
                                    ; for other SPC modules always 0
mode= 0                             ; for SPC7xx      , default 0
                                    ;   0 - normal operation (routing in), 1 -  Scan In,
                                    ;   2 - block address out, 3 - Scan Out
                                    ; for SPC6xx      , 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 other SPC modules not used
scan_size_x=1                       ; for SPC7xx modules in scanning modes 1 .. 65536, default 1
scan_size_y=1                       ; for SPC7xx modules in scanning modes 1 .. 65536, default 1
```

```
scan_rout_x=1                  ; for SPC7xx modules in scanning modes 1 .. 128, default 1
scan_rout_y=1                  ; for SPC7xx modules in scanning modes 1 .. 128, 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:
                               ;       10            -          12
                               ;       12            -          10
                               ;       14            -           8
                               ;       16            -           6
scan_polarity=0                ; for SPC7xx 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)
scan_flyback=0                 ; for SPC7xx modules in Scan Out mode, default 0
                               ; bits 7-0  Flyback X in number of pixels
                               ; bits 15-8 Flyback Y in number of lines
scan_borders=0                 ; for SPC7xx modules in Scan In mode, default 0
                               ; bits 7-0  Upper boarder, bits 15-8 Left boarder
pixel_time= 200e-9             ; pixel time in sec for SPC7xx modules in Scan In mode,
                               ;  50e-9 .. 1.0 , default 200e-9
pixel_clock= 0                 ; source of pixel clock for SPC7xx modules in Scan In mode
                               ;  0 - internal, 1 - external, default 0
line_compression= 1            ; line compression factor for SPC7xx modules in Scan In mode,
                               ;  1,2,4,8,16,32,64,128, default 1
trigger = 0                    ;  external trigger condition for SPC6xx,7xx,130 modules
                               ;      none(0)(default),active low(1),active high(2)
ext_pixclk_div= 1              ; divider of external pixel clock for SPC7xx 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 can be : 1.0s, 0.25s, 0.1s, 0.05s
macro_time_clk= 0              ; macro time clock definition for SPC130 in FIFO mode
                               ;    0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.,
                               ;    3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.
add_select= 0                  ; selects ADD signal source: 0 - internal (ADD only) (default),
                               ;                               1 - external
```

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 spc_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 (0...0x3E0,default 0x380)
short init	Return value of SPC_init
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
float cfd_limit_low	SPCx3x -500 .. -20mV ,for SPCx0x 5 .. 80mV
float cfd_limit_high	5 ..80 mV, default 80 mV , not for SPC130
float cfd_zc_level	SPCx3x -96 .. 96mV, SPCx0x -10 .. 10mV
float cfd_holdoff	SPCx0x: 5 .. 20 ns, other modules: no influence
float sync_zc_level	SPCx3x: -96 .. 96mV, SPCx0x: -10..10mV
short sync_freq_div	1,2,4,8,16 (SPC130/230: 1,2,4)
float sync_holdoff	4 .. 16 ns (SPC130/230: no influence)
float sync_threshold	SPCx3x: -500 .. -20mV, SPCx0x: no influence
float tac_range	50 .. 2000 ns
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	SPC300(330) fixed to 10 or 12 depending on the module type SPC401(431) fixed to 12 SPC402(432) fixed to 8 other SPC modules: 6,8,10,12 bits
short ext_latch_delay	0 ..255, SPC130/230: no influence
float collect_time	SPC300/330: 0.01s .. 100000s

	other SPC modules: 0.0001s .. 100000s
float display_time	SPC300/330: 0.01 .. 100000s
	other SPC modules: no influence
float repeat_time	SPC300/330: 0.01 .. 100000s
	other SPC modules: 0.0001s .. 100000s
short stop_on_time	0 (stop) or 1 (no stop)
short stop_on_ovfl	0 (stop) or 1 (no stop)
short dither_range	SPC300/330: 0,32,64,128,256
	other SPC modules: 0=0, 32=1/64, 64=1/32, 128=1/16, 256=1/8
short count_incr	1 .. 255
short mem_bank	SPC400(130,230,430,600,630): 0 or 1
	other SPC modules: always 0
short dead_time_comp	0 (off) or 1 (on), SPC300/330: always 1
unsigned short scan_control	SPC505(535,506,536) scanning(routing) control word
	other SPC modules always 0
short routing_mode	SPC230: 0 (off) or 1 (on)
	other SPC modules always 0
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 for PCI module(0-7) or -1 for ISA module
unsigned short mode;	SPC6(7)XX, other SPC modules not used
	for SPC7xx , default 0
	0 - normal operation (routing in),
	1 - block address out, 2 - Scan In, 3 - Scan Out
	for SPC6xx , 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
unsigned long scan_size_x;	for SPC7xx modules in scanning modes 1 .. 65536, default 1
unsigned long scan_size_y;	for SPC7xx modules in scanning modes 1 .. 65536, default 1
unsigned long scan_rout_x;	for SPC7xx modules in scanning modes 1 .. 128, default 1
unsigned long scan_rout_y;	for SPC7xx modules in scanning modes 1 .. 128, 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:
	10 - 12
	12 - 10
	14 - 8
	16 - 6 */
unsigned short scan_polarity;	for SPC7xx 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)
unsigned short scan_flyback;	for SPC7xx modules in Scan Out or Rout Out mode, default 0
	bits 7-0 Flyback X in number of pixels
	bits 15-8 Flyback Y in number of lines
unsigned short scan_borders;	for SPC7xx modules in Scan In mode, default 0
	bits 7-0 Upper boarder, bits 15-8 Left boarder
float pixel_time;	pixel time in sec for SPC7xx modules in Scan In mode,
	50e-9 .. 1.0 , default 200e-9
unsigned short pixel_clock;	for SPC7xx modules in Scan In mode,
	pixel clock source, 0 - internal,1 - external, default 0
unsigned short line_compression;	line compression factor for SPC7xx modules
	in Scan In mode, 1,2,4,8,16,32,64,128, default 1
unsigned short trigger;	external trigger condition for SPC6xx,7xx,130 modules -
	none(0),active low(1),active high(2)
unsigned long ext_pixclk_div;	divider of external pixel clock for SPC7xx modules
	in Scan In mode, 1 .. 0x3fe, default 1
float rate_count_time;	rate counting time in sec default 1.0 sec
	for SPC130 can be : 1.0s, 0.25s, 0.1s, 0.05s
short macro_time_clk;	macro time clock definition for SPC130 in FIFO mode
	0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.,
	3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.
short add_select;	selects ADD signal source: 0 - internal (ADD only) (default),
	1 - external

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.

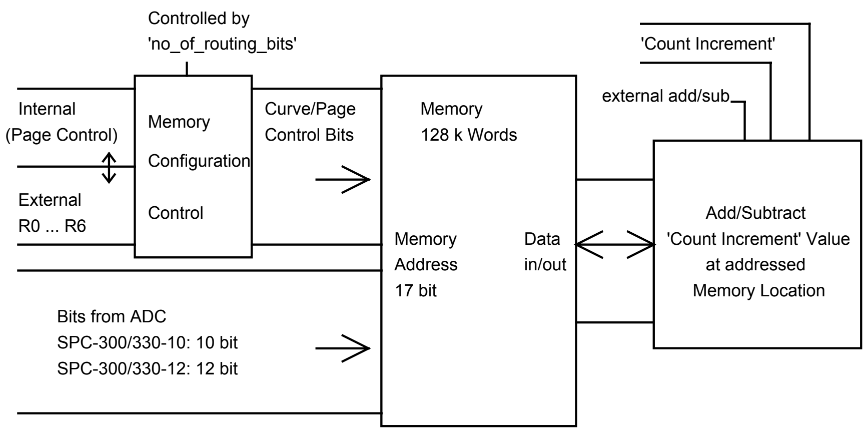
Single parameter values can be transferred to or from the DLL and module level by the functions **SPC_set_parameter** and **SPC_get_parameter**. To identify the desired parameter, the parameter identification par_id is used. For the parameter identification keywords are defined in spc_def.h.

Memory Configuration

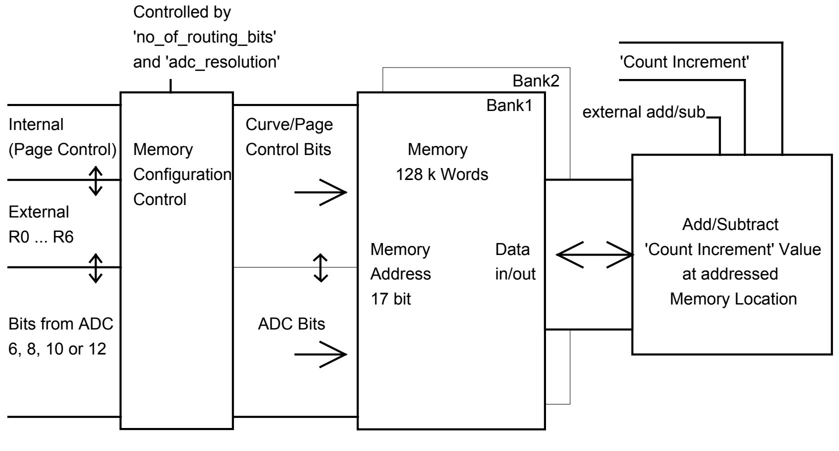
The SPC memory is interpreted as a set of 'pages'. One page contains a number of 'blocks'. One block contains one decay curve. The number of points per block (per curve) is defined by the module type (SPC-3 modules) or by the ADC resolution (SPC-1, 2, 4, 5, 6, 7). The number of blocks per page depends 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, 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).

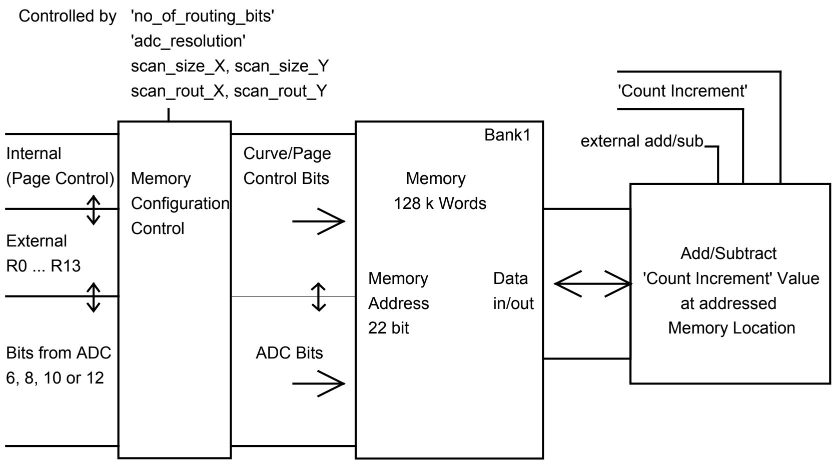
The figures below show the influence of these parameters on the memory configuration.



Memory Control (SPC-300/330)



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 and the scan parameters 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. The memory configuration determines in which part of the SPC memory the measurement data is recorded (see also SPC_set_page).

SPC-300/330 modules:

The length of the recorded curves (waveforms) depends on the module type. The -10 versions generally record curves with 1024, the -12 versions with 4096 points. The selected ADC resolution acts on the display only. The whole memory has space for 128 curves in the -10 version or 32 curves in the -12 version. Depending on the number of routing bits used the memory can hold a different number of measurement data sets or 'pages'.

SPC-130/400/430/500/530 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.

For SPC-505/535 modules:

The length of the recorded curves is determined by the ADC resolution and can range from 64 to 4096. The number of complete measurement data sets (or 'pages') depends on the ADC resolution, and on the size of the scan frame (lines per frame and number of pixels per line) defined by the SCAN_CONTROL parameters.

SPC-506/536 modules:

The length of the recorded curves is determined by the ADC resolution and can range from 64 to 1024. The number of complete measurement data sets (or 'pages') depends on the ADC resolution, and on the number of lines per image and no of pixels per line defined by the SCAN_CONTROL parameters.

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

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 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. An SPC_configure_memory function call is not required.

SPC-700/730 modules, Normal operation modes:

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-700/730 modules, Scanning modes:

The Memory configuration is not done not by SPC_configure_memory. Instead, the memory is configured by but by setting the parameters:

ADC_RESOLUTION – defines block_length,
SCAN_SIZE_X, SCAN_SIZE_Y – defines blocks_per_frame
SCAN_ROUT_X, SCAN_ROUT_Y – defines frames_per_page

However, after setting these parameters SPC_configure_memory should be called with 'adc_resolution' = 0 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:

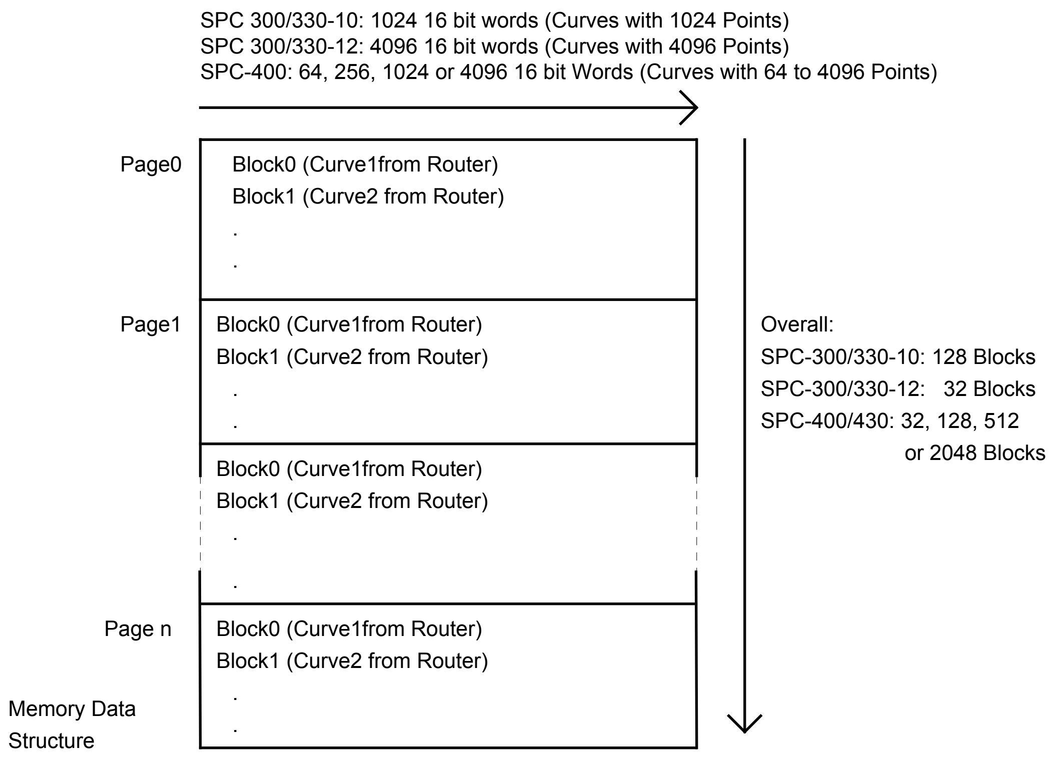
long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-4(6)00)
short block_length	Number of curve points16-bits words per block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page (Normally 1)
long maxpage	max number of pages to use in a measurement

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.

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 modules in scanning modes. Therefore, the memory structure is determined by the number of routing bits and the ADC resolution (or the module type in the case of the SPC-300/330). The memory is configured by the function **SPC_configure_memory** (see 'Memory Configuration).

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 SPC_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)

For all modules except SPC300(330) :

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 SPC400(430), SPC600(630) and SPC130 modules only :

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

For SPC401(431,402,432) SPC600(630) and SPC130 modules only:

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

For SPC505(535), SPC700(730) modules only:

SPC_SCRDY	0x400	Scan ready (data can be read)
SPC_FBRDY	0x800	Flow back of scan finished

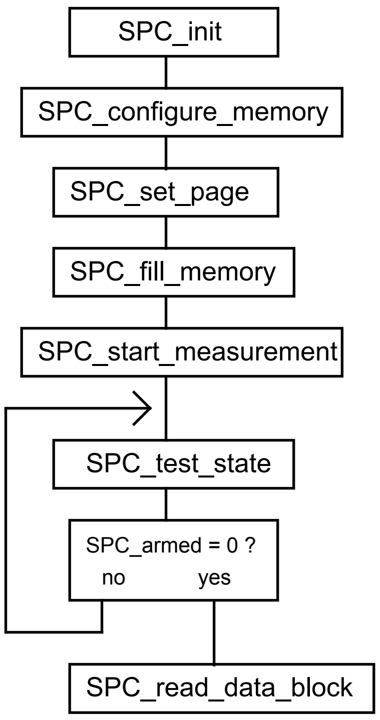
For SPC600(630), SPC700(730) and SPC130 modules only:

SPC_WAIT_TRG	0x1000	Wait for external trigger

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.

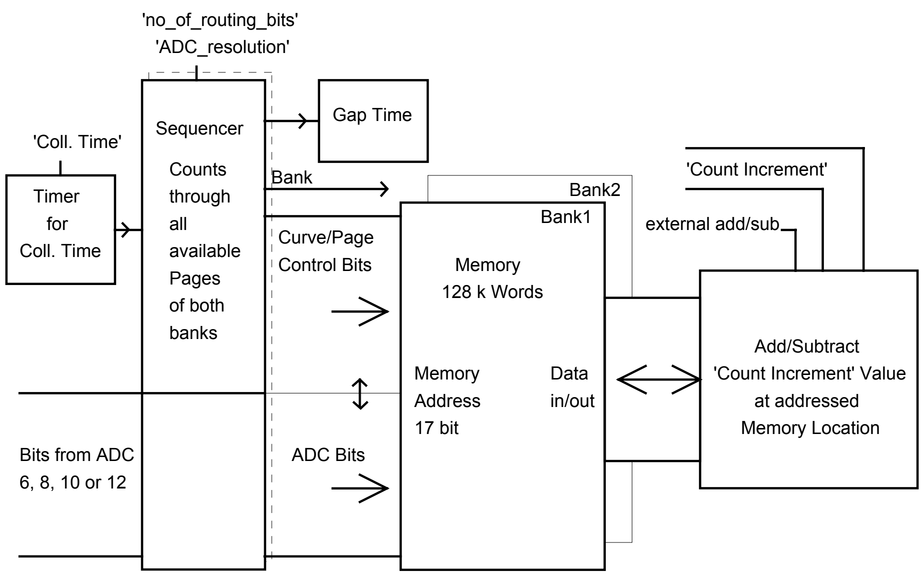


At the beginning, the measurement parameters are read from an initialisation file and send to the SPC module by SPC_init, and the memory is configured by SPC_configure_memory. The memory page in which the data is to be recorded is set by SPC_set_page. SPC_fill_memory is used to clear the data blocks (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-4,6 and SPC-130 Sequencer

In the SPC-4, SPC-6 and SPC-1 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. 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.

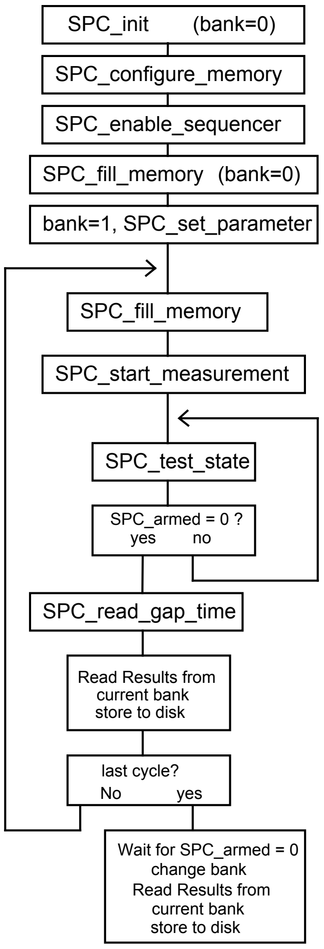
At the beginning, the measurement parameters are read from an initialisation file and sent to the SPC module by SPC_init. The memory is configured by SPC_configure_memory and the sequencer is enabled by **SPC_enable_sequencer**.

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).



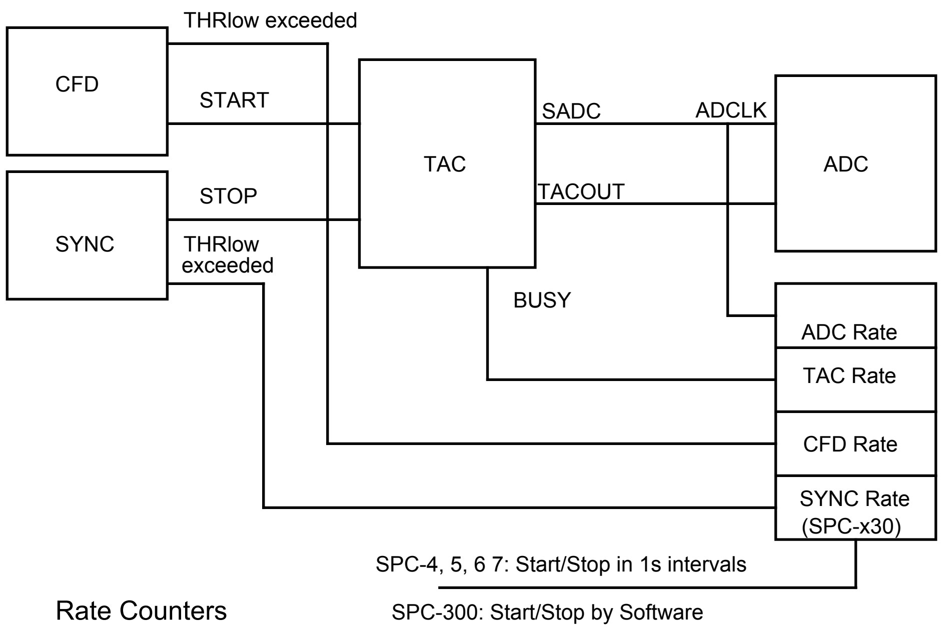
To measure an unlimited number of data blocks, the sequence is repeated in a loop. Thus, after reading the data from the current bank, this bank is cleared (SPC_fill_memory), while the measurement is still running in the alternate bank. The subsequent restarting of the measurement (SPC_start_measurement)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



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

The SYNC rate counter counts all pulses that exceed the lower discriminator threshold of the SYNC input. The sync rate counter is present only in the SPC-130, -430, -530, -630 and -730 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.

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 (not for SPC-300/330)
float cfd_rate	CDF rate in counts/s

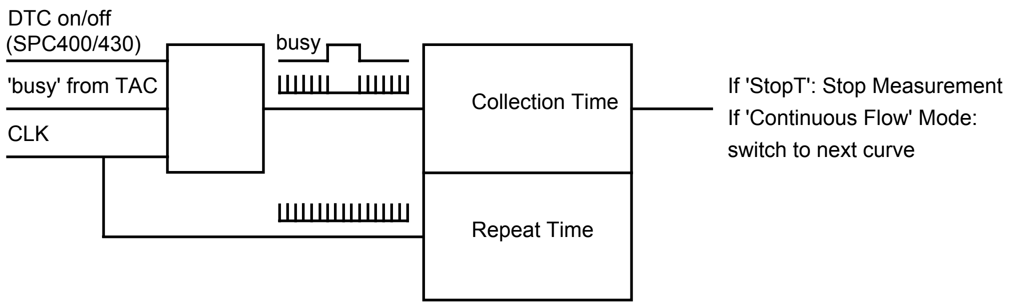
float tac_rate	TAC rate in counts/s
float adc_rate	ADC rate in counts/s

To get correct results the **SPC_clear_rates** function must be called

- for the SPC-300/330 before (re)starting the measurement and after each SPC_read_rates call
- for other SPC modules 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.



For all modules except SPC-300/330 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.

In the SPC-300/330 modules the dead time compensation cannot be switched off, it is always active.

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 spc_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**. In the SPC-400 a status bit is returned by SPC_test_state which indicates the expiration of the repeat counter.

When the sequencer of the SPC-400/430/600/630/130 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 spc_def.h file and can be read using **SPC_get_error_string**. We recommend to check the return value after each function call.

Description of the SPC DLL 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. This is accomplished by the function **SPC_init**. The function

- 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
- 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

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file spc400.ini or to start with spc400.ini and introduce the desired changes.

```
;   SPC400 initialisation file
;   SPC parameters have to be included in .ini file only when parameter
;   value is different from default.
;   only baseadr in spc_base section  is required
;   other parameters can have default values
[spc_base]
baseadr=  0x380                  ; base I/O address   (0 ... 0x3E0,default 0x380)
simulation = 0                   ; 0 - hardware mode(default) ,
                                 ; >0 - simulation mode (see spc_def.h for possible values)
pci_bus_no= 0                    ; PCI bus on which SPC module will be looking for
                                 ;   0 - 255, default 0 ( 1st found PCI bus with SPC modules will be scanned)
pci_card_no= 0                   ; number of module on PCI bus if PCI version of SPC module
                                 ;   0 - 7, default -1 ( ISA module)

[spc_module]                     ; SPC hardware parameters
cfd_limit_low= 5.0               ; for SPCx3x -500 .. -20mV ,for SPCx0x  5 .. 80mV
                                 ; default 5mV
cfd_limit_high= 80.0             ; 5 ..80 mV, default 80 mV, for SPC130 doesn't exist
cfd_zc_level= 0.0                ;for SPCx3x -96 .. 96mV ,for SPCx0x -10 .. 10mV
                                 ; default 0mV
cfd_holdoff= 5.0                  ; for SPCx0x 5 .. 20 ns , default 5ns
                                 ; for SPCx3x (130,230) doesn't exist
sync_zc_level= 0.0               ; for SPCx3x -96 .. 96mV ,for SPCx0x -10 .. 10mV
                                 ; default 0mV
sync_freq_div= 4                 ; for SPC130,230 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 SPC130,230 doesn't exist
```

```
sync_threshold= -20.0              ; for SPCx3x -500 .. -20mV ,default -20 mV
                                   ; for SPCx0x doesn't exist

tac_range= 50.0                     ; 50 .. 2000 ns , default 50 ns
tac_gain= 1                        ; 1 .. 15 ,default 1
tac_offset=0.0                      ; 0 .. 100% ,default 0%
tac_limit_low=                     10.0   ; 0 .. 100% ,default 10%
tac_limit_high= 80.0               ; 0 .. 100% ,default 80%

adc_resolution= 10                 ; for SPC300(330) fixed to 10 or 12
                                   ; for SPC401(431) fixed to 12
                                   ; for SPC402(432) fixed to 8
                                   ; for other SPC modules 6,8,10,12 bits, default 10
ext_latch_delay= 0                 ; 0 ..255 , default 0
                                   ; for SPC130/230 doesn't exist

collect_time= 0.01                 ; for SPC300(330) 0.01   .. 100000s
                                   ; for other SPC modules 0.0001 .. 100000s , default 0.01s

display_time= 1.0                  ; for SPC300(330) 0.01   .. 100000s , default 1.0s
                                   ; for other SPC modules display timer doesn't exist
repeat_time=  10.0                 ; for SPC300(330) 0.01   .. 100000s
                                   ; for other SPC modules 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                    ; for SPC300(330) 0,32,64,128,256 ,default 0
                                   ; for other SPC modules subsequent values have
                                   ;      different meaning  0, 1/64, 1/32, 1/16, 1/8
count_incr= 1                      ; 1 .. 255 , default 1
mem_bank= 0                        ; for SPC400(130,230,430,600,630) 0 , 1 , default 0
                                   ; for other SPC modules always 0
dead_time_comp= 1                  ; for SPC300(330) always 1
                                   ; for other SPC modules 0 , 1 , default 1
scan_control= 0                    ; for SPC505(535,506,536) , default 0xf000
                                   ;   16-bit hex value to control scanning in SPC505(535)
                                   ;    or routing in SPC506(536)
                                   ;   bits are defined in spc_def.h file
                                   ; for other SPC modules always 0
routing_mode= 0                    ; for SPC230 0 , 1 , default 0
                                   ; for other SPC modules always 0
tac_enable_hold= 50.0              ; for SPC230  10.0 .. 265.0 ns, default 50.0 ns
                                   ; for other SPC modules always 0
mode= 0                            ; for SPC7xx      , default 0
                                   ;   0 - normal operation (routing in), 1 -  Scan In,
                                   ;   2 - block address out, 3 - Scan Out
                                   ; for SPC6xx      , 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 other SPC modules not used
scan_size_x=1                      ; for SPC7xx modules in scanning modes 1 .. 65536, default 1
scan_size_y=1                      ; for SPC7xx modules in scanning modes 1 .. 65536, default 1
scan_rout_x=1                      ; for SPC7xx modules in scanning modes 1 .. 128, default 1
scan_rout_y=1                      ; for SPC7xx modules in scanning modes 1 .. 128, 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:
                                   ;        10                  -            12
                                   ;        12                  -            10
                                   ;        14                  -             8
                                   ;        16                  -             6
scan_polarity=0                    ; for SPC7xx 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)
scan_flyback=0                     ; for SPC7xx modules in Scan Out mode, default 0
                                   ; bits 7-0  Flyback X in number of pixels
                                   ; bits 15-8 Flyback Y in number of lines
scan_borders=0                     ; for SPC7xx modules in Scan In mode, default 0
                                   ; bits 7-0  Upper boarder, bits 15-8 Left boarder
pixel_time= 200e-9                 ; pixel time in sec for SPC7xx modules in Scan In mode,
```

18

```
                                      ;  50e-9 .. 1.0 , default 200e-9
pixel_clock= 0                        ; source of pixel clock for SPC7xx modules in Scan In mode
                                      ;  0 - internal, 1 - external, default 0
line_compression= 1                   ; line compression factor for SPC7xx modules in Scan In mode,
                                      ;  1,2,4,8,16,32,64,128, default 1
trigger = 0                           ;  external trigger condition for SPC6xx,7xx,130 modules
                                      ;      none(0)(default),active low(1),active high(2)
ext_pixclk_div= 1                     ; divider of external pixel clock for SPC7xx 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 can be : 1.0s, 0.25s, 0.1s, 0.05s
macro_time_clk= 0                     ; macro time clock definition for SPC130 in FIFO mode
                                      ;    0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.,
                                      ;    3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.
add_select= 0                         ; selects ADD signal source: 0 - internal (ADD only) (default),
                                      ;                                1 - external
```

----

short CVICDECL SPC_get_init_status(void);

Input parameters: none

Return value: initialisation result code

Description:

The procedure returns the initialisation result code set by the function SPC_init. The possible values are shown below (see also spc_def.h):

INIT_OK	0	no error
INIT_NOT_DONE	-1	init not done
INIT_WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_WRONG_MOD_ID	-3	wrong module identification code
INIT_HARD_TEST_ERR	-4	hardware test failed
INIT_CANT_OPEN_PCI_CARD	-5	cannot open PCI card

----

short CVICDECL SPC_get_parameters(SPCdata * data);

Input parameters: *data pointer to result structure (type SPCdata)

Return value: 0

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 structures of the DLLs into a structure of the type SPCdata (see spc_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 (0...0x3E0,default 0x380)

short init	Return value of SPC_init
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
float cfd_limit_low	SPCx3x -500 .. -20mV ,for SPCx0x 5 .. 80mV
float cfd_limit_high	5 ..80 mV, default 80 mV , not for SPC130
float cfd_zc_level	SPCx3x -96 .. 96mV, SPCx0x -10 .. 10mV
float cfd_holdoff	SPCx0x: 5 .. 20 ns, other modules: no influence
float sync_zc_level	SPCx3x: -96 .. 96mV, SPCx0x: -10..10mV
short sync_freq_div	1,2,4,8,16 (SPC130/230: 1,2,4)
float sync_holdoff	4 .. 16 ns (SPC130/230: no influence)
float sync_threshold	SPCx3x: -500 .. -20mV, SPCx0x: no influence
float tac_range	50 .. 2000 ns
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	SPC300(330) fixed to 10 or 12 depending on the module type
	SPC401(431) fixed to 12
	SPC402(432) fixed to 8
	other SPC modules: 6,8,10,12 bits
short ext_latch_delay	0 ..255, SPC130/230: no influence
float collect_time	SPC300/330: 0.01s .. 100000s
	other SPC modules: 0.0001s .. 100000s
float display_time	SPC300/330: 0.01 .. 100000s
	other SPC modules: no influence
float repeat_time	SPC300/330: 0.01 .. 100000s
	other SPC modules: 0.0001s .. 100000s
short stop_on_time	0 (stop) or 1 (no stop)
short stop_on_ovfl	0 (stop) or 1 (no stop)
short dither_range	SPC300/330: 0,32,64,128,256
	other SPC modules: 0=0, 32=1/64, 64=1/32, 128=1/16, 256=1/8
short count_incr	1 .. 255
short mem_bank	SPC400(130,230,430,600,630): 0 or 1
	other SPC modules: always 0
short dead_time_comp	0 (off) or 1 (on), SPC300/330: always 1
unsigned short scan_control	SPC505(535,506,536) scanning(routing) control word
	other SPC modules always 0
short routing_mode	SPC230: 0 (off) or 1 (on)
	other SPC modules always 0
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 for PCI module(0-7) or -1 for ISA module
unsigned short mode;	SPC6(7)XX, other SPC modules not used
	for SPC7xx , default 0
	0 - normal operation (routing in),
	1 - block address out, 2 - Scan In, 3 - Scan Out
	for SPC6xx , 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
unsigned long scan_size_x;	for SPC7xx modules in scanning modes 1 .. 65536, default 1
unsigned long scan_size_y;	for SPC7xx modules in scanning modes 1 .. 65536, default 1
unsigned long scan_rout_x;	for SPC7xx modules in scanning modes 1 .. 128, default 1
unsigned long scan_rout_y;	for SPC7xx modules in scanning modes 1 .. 128, 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:
	10 - 12
	12 - 10
	14 - 8
	16 - 6 */
unsigned short scan_polarity;	for SPC7xx 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)
unsigned short scan_flyback;	for SPC7xx modules in Scan Out or Rout Out mode, default 0
	bits 7-0 Flyback X in number of pixels
	bits 15-8 Flyback Y in number of lines
unsigned short scan_borders;	for SPC7xx modules in Scan In mode, default 0
	bits 7-0 Upper boarder, bits 15-8 Left boarder
float pixel_time;	pixel time in sec for SPC7xx modules in Scan In mode,
	50e-9 .. 1.0 , default 200e-9

```
unsigned short pixel_clock;           for SPC7xx modules in Scan In mode,
                                              pixel clock source, 0 - internal,1 - external, default 0
unsigned short line_compression;      line compression factor for SPC7xx modules
                                              in Scan In mode,   1,2,4,8,16,32,64,128, default 1
unsigned short trigger;               external trigger condition for SPC6xx,7xx,130 modules -
                                              none(0),active low(1),active high(2)
unsigned long ext_pixclk_div;         divider of external pixel clock for SPC7xx modules
                                              in Scan In mode, 1 .. 0x3fe, default 1
float rate_count_time;                rate counting time in sec  default 1.0 sec
                                      ;       for SPC130 can be : 1.0s, 0.25s, 0.1s, 0.05s
short macro_time_clk;                 macro time clock definition for SPC130 in FIFO mode
                                          0 - 50ns (default), 1 - SYNC freq., 2 - 1/2 SYNC freq.,
                                          3 - 1/4 SYNC freq., 4 - 1/8 SYNC freq.
short add_select;                     selects ADD signal source: 0 - internal (ADD only) (default),
                                                          1 - external
```

----

short CVICDECL SPC_set_parameters(SPCdata *data);

Input parameters: *data pointer to parameters structure (type SPCdata, see spc_def.h)

Return value: 0 no errors, <0 error code (see spc_def.h)

Description:

The procedure sends all parameters from the ‘SPCdata’ structure to the internal DLL structures and to the control registers of the SPC module.

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 par_id, float * value);

Input parameters:

par_id	parameter identification number (see spc_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 data structures of the DLL library. The par_id values are defined in spc_def.h file as SPC_PARAMETERS_KEYWORDS.

----

short CVICDECL SPC_set_parameter(short par_id, float value);

Input parameters:

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 data structures of the DLL functions and to the SPC module. 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 spc_def.h file as SPC_PARAMETERS_KEYWORDS.

----

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

Input parameters:

adc_resolution	ADC resolution (0*,6,8,10,12) * With adc_resolution = 0 the procedure ‘mem_info’ is filled with the current values disregarding ‘no_of_routing_bits’.
no_of_routing_bits	number of routing bits (0 - 7) (0 – 14 for SPC-5(7)xx modules),
* mem_info	pointer to result memory info structure

Return value:

0 no errors, <0 error code

The procedure configures the SPC memory depending on the specified ADC resolution, the module type and the number of detector channels (if a router is used). The procedure has to be called before the first access to the SPC memory or before a measurement is started. 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 module type (SPC-3 modules) or by the ADC resolution (SPC-4, 5 ,6 ,7,1). 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, a page contains a number of ‘frames’ (normally 1). Each frame contains a number of blocks. The number of blocks per page depends on the number of points per block and on the number of detector channels (PointsX and PointsY) and on the scanning parameters (pixels per line and lines per frame).

The differences between the modules are listed below.

SPC-300/330 modules:

The length of the recorded curves (waveforms) depends on the module type. The -10 versions generally record curves with 1024, the -12 versions with 4096 points. The selected ADC resolution acts on the display only. The whole memory has space for 128 curves in the -10 version or 32 curves in the -12 version. Depending on the number of routing bits used the memory can hold a different number of measurement data sets or 'pages'.

SPC-400/430/500/530 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.

For SPC-505/535 modules:

The length of the recorded curves is determined by the ADC resolution and can range from 64 to 4096. The number of complete measurement data sets (or 'pages') depends on the ADC resolution, and on the size of the scan frame (lines per frame and number of pixels per line) defined by the SCAN_CONTROL parameters.

SPC-506/536 modules:

The length of the recorded curves is determined by the ADC resolution and can range from 64 to 1024. The number of complete measurement data sets (or 'pages') depends on the ADC resolution, and on the number of lines per image and no of pixels per line defined by the SCAN_CONTROL parameters.

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

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 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. An SPC_configure_memory function call is not required.

SPC-700/730 modules, Normal operation modes:

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-700/730 modules, Scanning modes:

The Memory configuration is not done not by SPC_configure_memory. Instead, the memory is configured by but by setting the parameters:

ADC_RESOLUTION – defines block_length,

SCAN_SIZE_X, SCAN_SIZE_Y – defines blocks_per_frame
SCAN_ROUT_X, SCAN_ROUT_Y – defines frames_per_page

However, after setting these parameters SPC_configure_memory should be called with 'adc_resolution' = 0 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-4(6)(1)x0)
short block_length	Number of curve points16-bits words per block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement

Possible operation modes for the SPC-6x0/7x0 modules are defined in the spc_def.h file. The operation mode can be changed by setting the parameter MODE.

----

short CVICDECL SPC_fill_memory(long block, long page, unsigned short fill_value);

Input parameters:

block	block number to be filled
page	page number
fill_value	value written to SPC memory

Return value:

0: no errors, <0: error code

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

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 before the first call of SPC_fill_memory.

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.

----

short CVICDECL SPC_read_data_block(long block, long page, short reduction_factor, short from, short to, unsigned short *data);

Input parameters:

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 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 before the first call of SPC_read_data_block. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-400)
short block_length	Number of 16-bits words per one block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement

Please make sure that the buffer 'data' be allocated with enough memory for no_of_points.

----

short CVICDECL SPC_write_data_block(long block, long page, short from, short to, unsigned short *data);

Input parameters:

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 SPC memory defined by the parameters ‘block’ and ‘page’. 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 before the first call of SPC_write_data_block. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-4(6)(1)x0)
short block_length	Number of 16-bits words per one block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement

----

short CVICDECL SPC_read_fifo (unsigned long * count, unsigned short *data);

Input parameters:

*count	pointer to variable which will be filled with number of 16-bit words written to the buffer ‘data’
*data	pointer to data buffer which will be filled

Return value:

0 no errors, <0 error code

The procedure reads data from the FIFO memory of SPC modules SPC-401/431/402/432/600/630/130 and has no effect for other SPC module types. Because of hardware differences the procedure action is different for different SPC module types.

For SPC401(431) modules:

The function reads 48-bits frames from the FIFO memory and writes them to the buffer ‘data’ until the FIFO is empty. The ‘Count’ variable is filled with the number of 16-bit words written to the buffer.

For SPC402(432) modules:

The function reads 32-bits frames from the FIFO memory and writes them to the buffer ‘data’ until the FIFO is empty. The ‘Count’ variable is filled 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 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 like SPC4x1 modules) or FIFO_32 (32 bits frame like SPC4x2 modules) (fifo mode values are defined in spc_def.h file). After setting FIFO mode SPC6x0 module memory has FIFO structure. SPC_read_fifo function works now in the same way as described above depending on the fifo frame length.

For SPC130 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 SPC4x2/6x0 modules) (fifo mode values are defined in spc_def.h file). After setting FIFO mode SPC130 module memory has FIFO structure. SPC_read_fifo function works now in the same way as described above depending on the fifo frame length.

Please make sure that the buffer ‘data’ be allocated with enough memory for the expected number of frames.

----

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

Input parameters:

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 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 modules in scanning modes).

The procedure cannot be used for SPC-300(330) modules.

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 before the first call of SPC_read_data_frame. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-400)
short block_length	Number of 16-bits words per one block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement

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 (long first_page, long last_page, unsigned short *data);

Input parameters:

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 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 procedure cannot be used for SPC-300(330) modules.

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 before the first call of SPC_read_data_page. This function also delivers the required information about the block/page structure of the SPC memory:

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-400)
short block_length	Number of 16-bits words per one block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement

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_set_page(long page);

Input parameters:

page	page number, 0 to maxpage - 1

Return value: 0

The procedure defines the page of the SPC memory in which the data of a subsequent measurement will be recorded. SPC_set_page must be called before a measurement is started.

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

long max_block_no	total number of blocks (=curves) in the memory (per memory bank for the SPC-4(6)00)
short block_length	Number of 16-bits words per one block (curve)
long blocks_per_frame	Number of blocks (=curves) per frame
long frames_per_page	Number of frames per page
long maxpage	max number of pages to use in a measurement

----

short CVICDECL SPC_get_sync_state(short *sync_state);

Input parameters: *sync_state, pointer to a variable which will be set

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

The procedure sets "sync_state" according to the actual sync state.

For SPC-130 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_clear_rates(void);

Input parameters: none

Return value: 0

Description:

The procedure clears all SPC rate counters.

Because of hardware differences the effect of the procedure is different for the SPC300 and other SPC module types.

For SPC300/330 modules: To get correct rate values the procedure must be called before (re)starting the measurement and after each call of the SPC_read_rates function.

For other SPC module types modules: 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_read_rates(rate_values *rates);

Input parameters:

*rates	pointer to result rates structure

Return value:

For SPC300(330) modules: 0
For other SPC module types: 0 - OK, -1 - rate values not ready yet

The procedure reads the SPC rate counters, calculates the rate values and writes the results to the ‘rates’ structure. Because of hardware differences the procedure action is different for SPC300/330 and other SPC module types.

For SPC300(330) modules:

A SYNC rate is not available. For the other rates the function delivers correct results only during the measurement. After calling the function or before (re)starting the measurement procedure SPC_clear_rates must be called.

For other SPC module types:

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 -1.

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

----

short CVICDECL SPC_get_time_from_start(float *time);

Input parameters: *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. It should be called during the measurement, because the timer starts to run after (re)starting the measurement.

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 of the SPC-4(6)00/4(6)30/130 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(float *time);

Input parameters: *time: pointer to result variable

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

The procedure calculates 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(float *time);

Input parameters: *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. During the measurement this value decreases from the specified collection time to 0.

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 SPC4x0(230,6x0,130) modules only:

- If the sequencer is running, the collection timer cannot be accessed.
- The dead time compensation can be switched off (not for SPC3x0). 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_test_state(short *state);

Input parameters: *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. The function is used to control the measurement loop. The status bits delivered by the function are listed below (see also SPC_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)

for all modules except SPC300(330):

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 SPC400(430), SPC600(630) and SPC130 modules only :

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

for SPC401(431,402,432) SPC600(630) and SPC130 modules only:

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

for SPC5x5(7x0) modules only:

SPC_SCRDY	0x400	Scan ready (data can be read)
SPC_FBRDY	0x800	Flow back of scan finished

for SPC600(630), SPC700(730) and SPC130 modules only:

SPC_WAIT_TRG	0x1000	wait for external trigger

----

short CVICDECL SPC_start_measurement(void);

Input parameters: none

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

The procedure is used to start the measurement.

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),
- 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.

For SPC300(330) modules:

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

For all other SPC module types (except FIFO modules – SPC-401/431/402/432):

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!
- 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 FIFO modules – SPC-401/431/402/432 and SPC-6x0/130 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(void);

Input parameters: none

Return value: 0: not paused, because already finished,

> 0 paused, <0: error code

The procedure is used to pause a running measurement.

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

For SPC300(330) modules:

- The repeat and collection timers are tested to get current times used during restart,
- the SPC is disarmed (photon collection is stopped).

For all other SPC module types (except FIFO modules – SPC-4x1/4x2 and SPC-6x0/130 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 ('Continuous Flow' measurement):
-an error is returned - this measurement can’t be paused

For FIFO modules – SPC-401/431/402/432 and SPC-6x0/130 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(void);

Input parameters: none

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

The procedure is used to restart a measurement that was paused by SPC_pause_measurement.

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

For SPC300(330) modules:

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

For all other SPC module types (except FIFO modules – SPC-4x1/4x2 and SPC-6x0/130 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 ('Continuous Flow' measurement):

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

For FIFO modules – SPC-401/431/402/432 and SPC-6x0/130 in fifo mode:

The procedure should not be used for FIFO modules.

----

short CVICDECL SPC_stop_measurement(void);

Input parameters: none

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

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

For SPC300(330) modules:

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

For all other SPC module types (except FIFO modules – SPC-401/431/402/432):

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 ('Continuous Flow' measurement):

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

For FIFO modules – SPC-401/431/402/432 and SPC-6x0/130 in fifo mode:

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

----

short CVICDECL SPC_enable_sequencer(short enable);

Input parameters: enable 0 or 1 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 modules SPC4x0/230/5x5/5x6/6x0/130.

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:

If the sequencer was not enabled:

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

----

short CVICDECL SPC_get_sequencer_state(short *state);

Input parameters: * 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 SPC4x0/230/5x5/5x6/6x0/130 modules. The sequencer status bits are defined in the spc_def.h file:

```
SPC_SEQ_ENABLE               0x1     sequencer is enabled
SPC_SEQ_RUN                  0x2     sequencer is running
only for SPC4x0/230/6x0/130 modules
SPC_SEQ_GAP_BANK             0x4     sequencer is waiting for other bank to be armed
```

----

short CVICDECL SPC_read_gap_time(float *time);

Input parameters: * 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 SPC4x0/6x0/230/130 modules. ‘time’ is set to the last gap time in ms.

----

short CVICDECL SPC_get_eeprom_data(SPC_EEP_Data *eep_data);

Input parameters: *eep_data: pointer to result structure

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

The structure "eep_data" is filled with the contents of SPC module's EEPROM. The EEPROM contains production data and adjust parameters of the module. The structure "SPC_EEP_Data" is defined in the file spc_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 (unsigned short write_enable,
                                      SPC_EEP_Data *eep_data);
```

Input parameters:

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 SPC module's EEPROM 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 (SPC_Adjust_Para * adjpara);

Input parameters: *adjpara: pointer to result structure

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

The structure 'adjpara' is filled with adjust parameters 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 spc_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 (SPC_Adjust_Para * adjpara);

Input parameters: *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) 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 spc_def.h.

----

short CVICDECL SPC_test_id(void) ;

Input parameters: none

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

The procedure can be used to the check module identification code, i.e. 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, if the id is correct. Possible module type values are defined in spc_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 module types  - returned value from SPC_test_id */
#define M_SPC300          1
#define M_SPC330          2
#define M_SPC400          3
#define M_SPC430          4
#define M_SPC401          5         SPC FIFO with 48 bit data format
#define M_SPC431          6         SPC FIFO with 48 bit data format
#define M_SPC402          7         SPC FIFO with 32 bit data format
#define M_SPC432          8         SPC FIFO with 32 bit data format
#define M_SPC230          23        special version of 430
#define M_SPC500          50        400 with 8MB memory in one bank, no sequencer
#define M_SPC530          53        430 with 8MB memory in one bank, no sequencer
#define M_SPC505          505       500 with scan control
#define M_SPC535          535       530 with scan control
#define M_SPC506          506       500 with TV routing control
#define M_SPC536          536       530 with TV routing control
#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
```

----

short CVICDECL SPC_set_mode(short mode);

Input parameters: mode - mode of DLL operation

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 a low level procedure and not intended to normal use. It is used to switch the DLL to the simulation mode if hardware errors occur during the initialisation.

When switching to the Hardware Mode is requested, a hardware test is executed to check whether the hardware exists and is functional. If the hardware is OK, the SPC DLL is switched to the Hardware Mode.

Errors during the hardware test enforce the Simulation Mode. Use the function SPC_get_mode to check which mode is actually set. Possible 'mode' values are defined in the spc_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 spc_def.h file:

```
#define SPC_HARD            0        /* hardware mode */
#define SPC_SIMUL300_10     3010     /* simulation mode of SPC300 10 bits ADC */
#define SPC_SIMUL300_12     3012     /* simulation mode of SPC300 12 bits ADC */
#define SPC_SIMUL330_10     3310     /* simulation mode of SPC330 10 bits ADC */
#define SPC_SIMUL330_12     3312     /* simulation mode of SPC330 12 bits ADC */
#define SPC_SIMUL400        40       /* simulation mode of SPC400 module */
#define SPC_SIMUL430        43       /* simulation mode of SPC430 module */
#define SPC_SIMUL401        401      /* simulation mode of SPC401 module */
#define SPC_SIMUL431        431      /* simulation mode of SPC431 module */
#define SPC_SIMUL402        402      /* simulation mode of SPC402 module */
#define SPC_SIMUL432        432      /* simulation mode of SPC432 module */
#define SPC_SIMUL500        500      /* simulation mode of SPC500 module */
#define SPC_SIMUL530        530      /* simulation mode of SPC530 module */
#define SPC_SIMUL230        230      /* simulation mode of SPC230 module */
#define SPC_SIMUL505        505      /* simulation mode of SPC505 module */
#define SPC_SIMUL535        535      /* simulation mode of SPC535 module */
#define SPC_SIMUL506        506      /* simulation mode of SPC506 module */
#define SPC_SIMUL536        536      /* simulation mode of SPC536 module */
#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 */
```

----

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 spc_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 spc_def.h file.

----

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.

==