Becker & Hickl GmbH
Nahmitzer Damm
12277 Berlin
Tel. +49 30 212 800 20
Fax. +49 30 212 800 213
email: info@becker-hickl.com
http://www.becker-hickl.com

SPCM Dynamic Link Libraries

User Manual for DPC module

Version 4.0

April 2014

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 DPC-230 module. Up to eight DPC modules of the same type can be controlled using the SPCM DLL. The functions work under 32 or 64 bit Windows XP/Vista/7. 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 DPC-230 module:

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_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_scan_clk_state
SPC_get_fifo_usage

Measurement control functions:

SPC_start_measurement
SPC_stop_measurement

DPC memory transfer functions:

SPC_read_fifo

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_convert_dpc_raw_data
SPC_get_start_offset

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_fifo it is _stdcall version of SPC_read_fifo.
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
- sends the parameter values to the DPC control registers
- performs a hardware test of the DPC module(s)

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file dpc230.ini or to start with dpc230.ini and to introduce the desired changes.

```
;    SPCM DLL initialisation file for DPC230 module
;    DPC parameters have to be included in .ini file only when parameter
;    value is different from default.
;    for SPC modules use file spcm.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 DPC modules will be looking for
                                 ;   0 - 255, default -1 ( all PCI busses will be scanned)
pci_card_no= -1                  ; number of the DPC module on PCI bus
                                 ;   0 - 7, default -1 (all modules on PCI bus)

[spc_module]                     ; DPC hardware parameters
cfd_limit_low = -30.0            ; = CFD_TH1 threshold of CFD1 -510 ..0 mV, default -30
cfd_limit_high = -30.0           ; = CFD_TH2 threshold of CFD2 -510 ..0 mV, default -30
cfd_zc_level = -30.0             ; = CFD_TH3 threshold of CFD3 -510 ..0 mV, default -30
cfd_holdoff = -30.0              ; = CFD_TH4 threshold of CFD4 -510 ..0 mV, default -30

sync_zc_level = 0.0              ; = CFD_ZC1 Zero Cross Level of CFD1 -96 ..96 mV, default 0
sync_holdoff = 0.0               ; = CFD_ZC2 Zero Cross Level of CFD2 -96 ..96 mV, default 0
sync_threshold = 0.0             ; = CFD_ZC3 Zero Cross Level of CFD3 -96 ..96 mV, default 0
tac_limit_high = 0.0             ; = CFD_ZC4 Zero Cross Level of CFD4 -96 ..96 mV, default 0

mem_bank = 6                     ; bit 1 - DPC 1 active, bit 2 - DPC 2 active,
                                 ;       default 6 - both TDCs active
detector_type = 0                ; type of active inputs : bit 1 - TDC1, bit 2 - TDC2,
                                 ;     bit value 0 , CFD inputs active,
                                 ;     bit value 1 , TTL inputs active
                                 ; default 0 - both TDCs have CFD inputs active

chan_enable = 0x3ff3ff           ;   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
```

```
                                  ;        default 0x3ff3ff - all inputs enabled
chan_slope = 0                    ;   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
                                  ;     default 0    - all inputs falling edge
mode = 8                          ; module's operation mode , default 8
                                  ;    6 - TCSPC FIFO
                                  ;    7 - TCSPC FIFO Image mode
                                  ;    8 - Absolute Time FIFO mode
                                   ;    9 - Absolute Time FIFO Image mode
rate_count_time = 1.0             ; rate counting time in sec  default 1.0 sec
                                  ;       0.0 - don't count rate outside the measurement
collect_time = 2.0                ;  0.0001 .. 100000s , default 2.0 s
stop_on_time = 1                  ;  0,1 , default 1
sync_freq_div = 1                 ;  1,2,4 default 1  in TCSPC modes

trigger = 0                       ;  external trigger condition
                                  ;   bits 1 & 0 mean :   00 - ( value 0 ) none(default),
                                  ;                       01 - ( value 1 ) active low,
                                  ;                       10 - ( value 2 ) active high

pixel_clock = 0                   ; source of pixel clock in Image modes
                                  ;  0 - internal, 1 - external, default 0

chan_spec_no = 0x8813             ; channel numbers of special inputs
                                  ;  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
```

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 structure is common for all SPC and DPC module types, therefore some fields are not used for DPC modules. 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	= CFD_TH1 threshold of CFD1 -510 ..0 mV
float cfd_limit_high	= CFD_TH2 threshold of CFD2 -510 ..0 mV
float cfd_zc_level	= CFD_TH3 threshold of CFD3 -510 ..0 mV
float cfd_holdoff	= CFD_TH4 threshold of CFD4 -510 ..0 mV
float sync_zc_level	= CFD_ZC1 Zero Cross level of CFD1 -96 ..96 mV
float sync_holdoff	= CFD_ZC2 Zero Cross level of CFD2 -96 ..96 mV
float sync_threshold	= CFD_ZC3 Zero Cross level of CFD3 -96 ..96 mV
float tac_range	DPC range in TCSPC and Multiscaler mode,0.16461 .. 1e7 ns
short sync_freq_div	1,2,4
short tac_gain	not used for DPC230
float tac_offset	TDC offset in TCSPC and Multiscaler mode -100 .. 100%
float tac_limit_low	not used for DPC230
float tac_limit_high	= CFD_ZC4 Zero Cross level of CFD4 -96 ..96 mV
short adc_resolution	no of points of decay curve in TCSPC and Multiscaler mode 0,2,4,6,8,10,12,14 bits
short ext_latch_delay	not used for DPC230
float collect_time	0.0001 .. 100000s , default 2.0s

float display_time	not used for DPC230
float repeat_time	not used for DPC230
short stop_on_time	1 (stop) or 0 (no stop)
short stop_on_ovfl	not used for DPC230
short dither_range	not used for DPC230
short count_incr	not used for DPC230
short mem_bank	bit 1 - DPC 1 active, bit 2 - DPC 2 active, default 6 (both TDCs active)
short dead_time_comp	not used for DPC230
unsigned short scan_control	not used for DPC230
short routing_mode	DPC230 bits 0-7 - control bits
float tac_enable_hold	macro time clock in ps, default 82.305 ps
short pci_card_no	module no on PCI bus (0-7)
unsigned short mode;	module's operation mode , default 8 6 - TCSPC FIFO 7 - TCSPC FIFO Image mode 8 - Absolute Time FIFO mode 9 - Absolute Time FIFO Image mode
unsigned long scan_size_x;	not used for DPC230
unsigned long scan_size_y;	not used for DPC230
unsigned long scan_rout_x;	not used for DPC230
unsigned long scan_rout_y;	not used for DPC230
unsigned long scan_flyback;	not used for DPC230
unsigned long scan_borders;	not used for DPC230
unsigned short scan_polarity;	not used for DPC230
unsigned short pixel_clock;	source of pixel clock in Image modes 0 - internal,1 - external, default 0
unsigned short line_compression;	not used for DPC230
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
float pixel_time;	not used for DPC230
unsigned long ext_pixclk_div;	not used for DPC230
float rate_count_time;	rate counting time in sec default 1.0 sec 0.0 - don't count rate outside the measurement
short macro_time_clk;	not used for DPC230
short add_select;	not used for DPC230
short test_eep	0: EEPROM is not read and not tested 1: EEPROM is read and tested for correct checksum
short adc_zoom;	not used for DPC230
unsigned long img_size_x;	not used for DPC230
unsigned long img_size_y;	not used for DPC230
unsigned long img_rout_x;	not used for DPC230
unsigned long img_rout_y;	not used for DPC230
short xy_gain;	not used for DPC230
short master_clock;	not used for DPC230
short adc_sample_delay;	not used for DPC230
short detector_type;	type of active inputs : bit 1 - TDC1, bit 2 - TDC2, bit value 0 , CFD inputs active, bit value 1 , TTL inputs active
unsigned long chan_enable;	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;	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;	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;	not used for DPC230

To send the complete parameter set back to the DLLs and to the DPC 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 DPC 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 module has two independent FIFO buffers. Each of them can contain up to 4 million events. The streams of collected photons are written to the FIFO buffers in parallel. A SPC_configure_memory function call is not required.

Memory Read Functions

Reading the memory of the DPC module is accomplished by the function **SPC_read_fifo**. There is no need to fill (write to) FIFO buffers. **SPC_read_fifo** delivers stream of events collected in one TDC FIFO buffer. The function must be called for both TDC (if both are active) to read data from all active input channels. The data contain unsorted photon entries and additional TDC control events in a **raw** format.

User should save the raw data from both TDCs to separate files and then use **SPC_convert_dpc_raw_data** function **(**in a way described in use_dpc.c file) to convert the files to a result file in **.spc** format (described in DPC Manual). **.spc** format is easy to use and understand in comparison to a raw data.

Extracting photons information from the FIFO data is done by the functions **SPC_init_phot_stream** and **SPC_get_photon**.

During the measurement user should only read the raw data (by **SPC_read_fifo)** to avoid FIFO overruns and to increase max possible photons rate. Converting the data to .spc format and extracting photons information is a time consuming task and should be done after the measurement.

A new set of functions to control FIFO measurements is available starting from SPCM DLL v.4.0. The functions use stream of photons inserted to the buffers (called as 'buffered stream') instead of stream of .spc files.

This simplifies extracting photons information from the FIFO data even during running measurements, without necessity to create .spc files.

The functions dedicated to use with buffered streams are :

SPC_init_buf_stream, SPC_add_data_to_stream, SPC_read_fifo_to_stream, SPC_stream_reset, SPC_stream_start(stop)_condition, SPC_get_photons_from_stream, SPC_get_stream_buffer_size, SPC_get_buffer_from_stream

User initializes the stream by calling SPC_init_buf_stream and starts the measurement (SPC_start_measurement). During running measurement raw FIFO contents is read to stream buffers (internal, on a DLL level) by calling SPC_read_fifo_to_stream procedure.

User can extract photons from the stream using SPC_get_photons_from_stream. Photons can be extracted from the stream directly after reading from FIFO, or in any other moment, also by using other program's thread.

It is possible to define the moment when extracting photons from the stream will start or stop using SPC_stream_start(stop)_condition.

Start(stop) time can be defined. After reaching start(stop) time appearance of photons or markers defined in start(stop)_OR(AND)_mask is tested.

See example of such measurement in use_dpc.c file.

Standard Measurements

DPC operation modes are described in DPC Manual.

The most important measurement functions are listed below.

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 and important for the DPC module are listed below (see also SPCM_DEF.H).

SPC_ARMED1	0x80	measurement in progress in TDC 1
SPC_ARMED2	0x40	measurement in progress in TDC 2
SPC_MEASURE	0x80	image measurement active, no wait for trigger or 1st frame pulse
SPC_FEMPTY1	0x100	FIFO empty in TDC 1
SPC_FEMPTY2	0x200	FIFO empty in TDC 2
SPC_FOVL1	0x400	FIFO overflow in TDC 1, data lost
SPC_FOVL2	0x800	FIFO overflow in TDC 2, data lost
SPC_CTIM_OVER1	0x8	collection timer of TDC 1 expired
SPC_CTIM_OVER2	0x20	collection timer of TDC 2 expired
SPC_TIME_OVER	0x4	measurement stopped on expiration of collection timer
SPC_WAIT_FR	0x2000	image measurement waits for frame signal to stop
SPC_WAIT_TRG	0x1000	wait for trigger

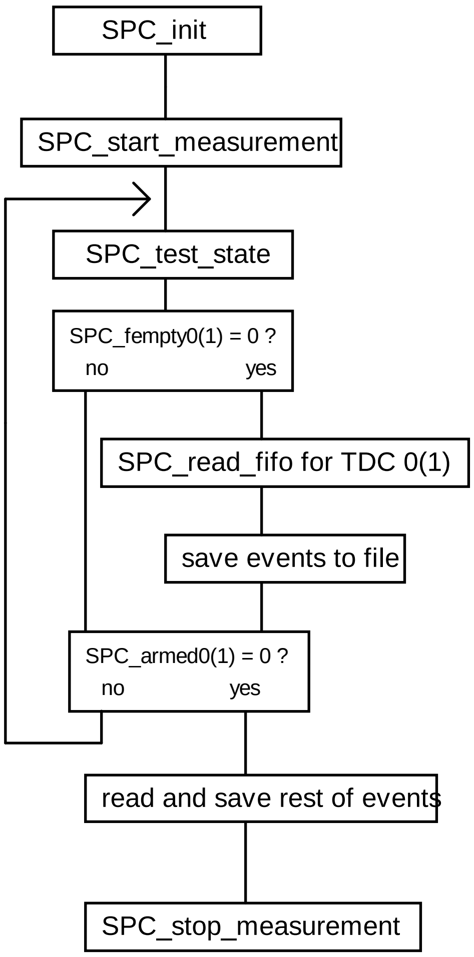
SPC_start_measurement starts the measurement in active TDCs with the parameters set before by the SPC_init, SPC_set_parameters or SPC_set_parameter functions. When the measurement is started by SPC_start_measurement, also the TDCs collection timers are started. In the standard mode, i.e. when intensity-versus-time functions are recorded in one ore more detector channels, the measurement stops when the specified stop condition appears (collection time expired, or stop by SPC_stop_measurement).

SPC_stop_measurement is used to stop the measurement by a software command.

For not imaging modes the measurement is stopped immediately in both TDCs.

For imaging modes real stop happens when next frame pulse appears (to ensure that whole frames are recorded). In this time period SPC_test_state will show the flag SPC_WAIT_FR. Another call to **SPC_stop_measurement** while waiting for the frame pulse will stop the measurement immediately.

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 DPC module by SPC_init. Additional parameter changes can be done before starting the measurement.

When the measurement has been started by SPC_start_measurement events can be read from TDCs FIFO buffers and saved to a file. The measurement runs until a stop condition (specified in the measurement parameters) is reached. Normally it is done after specified collection time when Stop_on_time parameter is set. Otherwise the measurement can be stopped by SPC_stop_measuremnt function, for example after reading specified amount of events. In this case the call of SPC_test_state returns SPC_armed0(1) = 0 indicating that the measurement has been stopped and the rest of data can be read from the module memory. Finally SPC_stop_measurement must be called to perform some necessary DLL actions.

See also use_dpc.c file for other measurement examples.

Count Rates

The count rates of the two TDCs of the DPC-230 are read by the **SPC_read_rates** function. Both rates are the total rates for all active channels of the TDCs.

Important: In contrast to the BH SPC modules the TDC chips have no counters to determine the count rates directly. The count rates have therefore to be determined by the SPCM software from the incoming photon data stream.

The results are stored into a structure of the following type – the same which is used for SPC modules:

float sync_rate	for DPC-230 - total photons rate in DPC1 in counts/s
float cfd_rate	for DPC-230 - total photons rate in DPC2 in counts/s
float tac_rate	for DPC-230 - not used
float adc_rate	for DPC-230 - not used

Integration time of rate values is equal normally 1sec and can be changed according to the parameter RATE_COUNT_TIME

To get correct results the **SPC_clear_rates** function must be called before the first call of SPC_read_rates

On-Board Timers

The DPC module is equipped with Collection timer which is used to control the measurement.

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 time from the start of a measurement is returned by the function **SPC_get_time_from_start**.

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 checking 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 DPC 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 of he DPC module(s)
- sends the parameter values to the internal structures of the DLL
- sends the parameter values to the DPC control registers
- performs a hardware test of the DPC module(s)

The DPC 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 DPC modules will be initialised:

- value 0 – 255 defines specific bus number (from the range: 0 to number of PCI busses with DPC modules) (if ‘pci_bus_no’ is greater than number of PCI busses with DPC modules, it is rounded to the number of busses –1)
- value –1 means that that the function will try to initialise DPC modules on all PCI busses

‘pci_card_no’ defines the number of DPC module on PCI bus to be initialised:

- value 0 – 7 defines one specific module (if ‘pci_card_no’ is greater than number of DPC 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 DPC 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 DPC230 module
;    DPC parameters have to be included in .ini file only when parameter
;    value is different from default.
;    for SPC modules use file spcm.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 DPC modules will be looking for
                                ;   0 - 255, default -1 ( all PCI busses will be scanned)
pci_card_no= -1                 ; number of the DPC module on PCI bus
                                ;   0 - 7, default -1 (all modules on PCI bus)

[spc_module]                    ; DPC hardware parameters
cfd_limit_low = -30.0           ; = CFD_TH1 threshold of CFD1 -510 ..0 mV, default -30
cfd_limit_high = -30.0          ; = CFD_TH2 threshold of CFD2 -510 ..0 mV, default -30
cfd_zc_level = -30.0            ; = CFD_TH3 threshold of CFD3 -510 ..0 mV, default -30
cfd_holdoff = -30.0             ; = CFD_TH4 threshold of CFD4 -510 ..0 mV, default -30

sync_zc_level = 0.0             ; = CFD_ZC1 Zero Cross Level of CFD1 -96 ..96 mV, default 0
sync_holdoff = 0.0              ; = CFD_ZC2 Zero Cross Level of CFD2 -96 ..96 mV, default 0
sync_threshold = 0.0            ; = CFD_ZC3 Zero Cross Level of CFD3 -96 ..96 mV, default 0
tac_limit_high = 0.0            ; = CFD_ZC4 Zero Cross Level of CFD4 -96 ..96 mV, default 0

mem_bank = 6                    ; bit 1 - DPC 1 active, bit 2 - DPC 2 active,
                                ;        default 6 - both TDCs active
detector_type = 0               ; type of active inputs : bit 1 - TDC1, bit 2 - TDC2,
                                ;      bit value 0 , CFD inputs active,
                                ;      bit value 1 , TTL inputs active
                                ; default 0 - both TDCs have CFD inputs active

chan_enable = 0x3ff3ff          ;   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
                                ;         default 0x3ff3ff - all inputs enabled
chan_slope = 0                  ;   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
                                ;      default 0    - all inputs falling edge
mode = 8                        ; module's operation mode , default 8
                                ;     6 - TCSPC FIFO
                                ;     7 - TCSPC FIFO Image mode
                                ;     8 - Absolute Time FIFO mode
                                 ;     9 - Absolute Time FIFO Image mode
rate_count_time = 1.0           ; rate counting time in sec  default 1.0 sec
                                ;       0.0 - don't count rate outside the measurement
collect_time = 2.0              ;  0.0001 .. 100000s , default 2.0 s
stop_on_time = 1                ;  0,1 , default 1
sync_freq_div = 1               ;  1,2,4 default 1  in TCSPC modes

trigger = 0                     ;  external trigger condition
                                ;    bits 1 & 0 mean :   00 - ( value 0 ) none(default),
                                ;                        01 - ( value 1 ) active low,
                                ;                        10 - ( value 2 ) active high

pixel_clock = 0                 ; source of pixel clock in Image modes
```

```
                                      ;  0 - internal, 1 - external, default 0

chan_spec_no = 0x8813                 ; channel numbers of special inputs
                                      ;  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 CVICDECL SPC_get_init_status(short mod_no);

Input parameters:

mod_no 0 .. 7, DPC module number

Return value: initialisation result code of the DPC module 'mod_no'

Description:

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

INIT_OK	0	no error
INIT_NOT_DONE	-1	init not done
INIT_WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_WRONG_MOD_ID	-3	wrong module identification code
INIT_HARD_TEST_ERR	-4	hardware test failed
INIT_CANT_OPEN_PCI_CARD	-5	cannot open PCI card
INIT_MOD_IN_USE	-6	module in use (locked) - cannot initialise
INIT_WINDRVR_VER	-7	incorrect WinDriver version
INIT_ SPC_WRONG_LICENSE	-8	corrupted license key
INIT_SPC_FIRMWARE_VER	-9	incorrect firmware version of DPC 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_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, DPC 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	DPC 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 a DPC 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 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  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, DPC 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 structure is common for all SPC and DPC module types, therefore some fields are not used for DPC modules. 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	= CFD_TH1 threshold of CFD1 -510 ..0 mV
float cfd_limit_high	= CFD_TH2 threshold of CFD2 -510 ..0 mV
float cfd_zc_level	= CFD_TH3 threshold of CFD3 -510 ..0 mV
float cfd_holdoff	= CFD_TH4 threshold of CFD4 -510 ..0 mV
float sync_zc_level	= CFD_ZC1 Zero Cross level of CFD1 -96 ..96 mV
float sync_holdoff	= CFD_ZC2 Zero Cross level of CFD2 -96 ..96 mV
float sync_threshold	= CFD_ZC3 Zero Cross level of CFD3 -96 ..96 mV
float tac_range	DPC range in TCSPC and Multiscaler mode,0.16461 .. 1e7 ns
short sync_freq_div	1,2,4
short tac_gain	not used for DPC230
float tac_offset	TDC offset in TCSPC and Multiscaler mode -100 .. 100%
float tac_limit_low	not used for DPC230
float tac_limit_high	= CFD_ZC4 Zero Cross level of CFD4 -96 ..96 mV
short adc_resolution	no of points of decay curve in TCSPC and Multiscaler mode 0,2,4,6,8,10,12,14 bits
short ext_latch_delay	not used for DPC230
float collect_time	0.0001 .. 100000s , default 2.0s

float display_time	not used for DPC230
float repeat_time	not used for DPC230
short stop_on_time	1 (stop) or 0 (no stop)
short stop_on_ovfl	not used for DPC230
short dither_range	not used for DPC230
short count_incr	not used for DPC230
short mem_bank	bit 1 - DPC 1 active, bit 2 - DPC 2 active, default 6 (both TDCs active)
short dead_time_comp	not used for DPC230
unsigned short scan_control	not used for DPC230
short routing_mode	DPC230 bits 0-7 - control bits
float tac_enable_hold	macro time clock in ps, default 82.305 ps
short pci_card_no	module no on PCI bus (0-7)
unsigned short mode;	module's operation mode , default 8
	6 - TCSPC FIFO
	7 - TCSPC FIFO Image mode
	8 - Absolute Time FIFO mode
	9 - Absolute Time FIFO Image mode
unsigned long scan_size_x;	not used for DPC230
unsigned long scan_size_y;	not used for DPC230
unsigned long scan_rout_x;	not used for DPC230
unsigned long scan_rout_y;	not used for DPC230
unsigned long scan_flyback;	not used for DPC230
unsigned long scan_borders;	not used for DPC230
unsigned short scan_polarity;	not used for DPC230
unsigned short pixel_clock;	source of pixel clock in Image modes
	0 - internal,1 - external, default 0
unsigned short line_compression;	not used for DPC230
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
float pixel_time;	not used for DPC230
unsigned long ext_pixclk_div;	not used for DPC230
float rate_count_time;	rate counting time in sec default 1.0 sec
	0.0 - don't count rate outside the measurement
short macro_time_clk;	not used for DPC230
short add_select;	not used for DPC230
short test_eep	0: EEPROM is not read and not tested
	1: EEPROM is read and tested for correct checksum
short adc_zoom;	not used for DPC230
unsigned long img_size_x;	not used for DPC230
unsigned long img_size_y;	not used for DPC230
unsigned long img_rout_x;	not used for DPC230
unsigned long img_rout_y;	not used for DPC230
short xy_gain;	not used for DPC230
short master_clock;	not used for DPC230
short adc_sample_delay;	not used for DPC230
short detector_type;	type of active inputs : bit 1 - TDC1, bit 2 - TDC2,
	bit value 0 , CFD inputs active,
	bit value 1 , TTL inputs active
unsigned long chan_enable;	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;	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;	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;	not used for DPC230

----

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

Input parameters:

mod_no	0... 7, DPC 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 DPC module 'mod_no'.

The new parameter values are recalculated according to the parameter limits, hardware restriction and the DPC 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, DPC 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, DPC 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 DPC module ‘mod_no’.

If ‘mod_no’ = -1, parameter will be changed for all DPC modules which are actually in use.

The new parameter value is recalculated according to the parameter limits, hardware restrictions and DPC 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, DPC 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 DPC module ‘mod_no’. The EEPROM contains production data 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 DPC 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, DPC 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 a DPC 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_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 using 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 DPC module.

----

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

Input parameters:

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

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

Description:

The parameters set from the 'SPCdata' structure 'data' is saved to the section [spc_module] in dest_inifile file. [spc_base] section and initial comment lines are copied to dest_inifile from the source_inifile file.

If the parameter 'source_inifile' is equal NULL, ini_file used in SPC_init function call is used as the source file for dest_inifile.

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

The 'dest_inifile' and 'source_inifile' files are ASCII files with a structure shown in SPC_init description.

Use SPC_read_parameters_from_inifile to read back the parameters set from the file and then SPC_set_parameters to send it to the DPC module.

Status functions:

----

short CVICDECL SPC_test_state(short mod_no, short *state);

Input parameters:

mod_no	0 .. 7, DPC 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 DPC module 'mod_no'. The function is used to control the measurement loop. The status bits delivered by the function and important for the DPC module are listed below (see also SPCM_DEF.H).

SPC_ARMED1	0x80	measurement in progress in TDC 1
SPC_ARMED2	0x40	measurement in progress in TDC 2
SPC_MEASURE	0x80	image measurement active, no wait for trigger or 1st frame pulse
SPC_FEMPTY1	0x100	FIFO empty in TDC 1

SPC_FEMPTY2	0x200	FIFO empty in TDC 2
SPC_FOVL1	0x400	FIFO overflow in TDC 1, data lost
SPC_FOVL2	0x800	FIFO overflow in TDC 2, data lost
SPC_CTIM_OVER1	0x8	collection timer of TDC 1 expired
SPC_CTIM_OVER2	0x20	collection timer of TDC 2 expired
SPC_TIME_OVER	0x4	measurement stopped on expiration of collection timer
SPC_WAIT_FR	0x2000	image measurement waits for frame signal to stop
SPC_WAIT_TRG	0x1000	wait for trigger

----

short CVICDECL SPC_get_sync_state(short mod_no, short *sync_state);

Input parameters:

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

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

In the relative timing modes (multichannel scaler or TCSPC) reference pulses are recorded in one channel of the DPC-230. The times of the reference pulses are recorded together with the photon times, and relative times between the photons and the reference pulses are calculated. The presence of the signal in a reference channel can be tested by the procedure.

Reference channel is defined by setting the parameter CHAN_SPEC_NO.

The procedure sets "sync_state" according to the reference signal presence on the DPC module ‘mod_no’.

0:	Reference signal not detected
1:	Reference signal detected

----

short CVICDECL SPC_get_time_from_start(short mod_no, float *time);

Input parameters:

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

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

The procedure reads the DPC collection timer and calculates the time from the start of the measurement for the DPC module ‘mod_no’. It should be called during the measurement, because the timer starts to run after starting the measurement.

The procedure can be used to test the progress of the measurement.

----

short CVICDECL SPC_get_break_time(short mod_no, float *time);

Input parameters:

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

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

The procedure calculates for the DPC 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). 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, DPC module number
*time	pointer to result variable

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

The procedure reads the collection timer and calculates the actual collection time value for the DPC module 'mod_no'. During the measurement this value decreases from the specified collection time to 0.

----

short CVICDECL SPC_read_rates(short mod_no, rate_values *rates);

Input parameters:

mod_no	0 .. 7, DPC 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 counts for the DPC module 'mod_no', calculates the rate values and writes the results to the 'rates' structure.

The results are stored into a structure of the following type – the same which is used for SPC modules:

float sync_rate	for DPC-230 - total photons rate in DPC1 in counts/s
float cfd_rate	for DPC-230 - total photons rate in DPC2 in counts/s
float tac_rate	for DPC-230 - not used
float adc_rate	for DPC-230 - not used

Integration time of rate values is equal normally 1sec and can be changed according to the parameter RATE_COUNT_TIME

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.

----

short CVICDECL SPC_clear_rates(short mod_no);

Input parameters:

mod_no	0 .. 7, DPC module number

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

Description:

The procedure clears all rate counters for the DPC 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_scan_clk_state (short mod_no, short *scan_state);

Input parameters:

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

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

The procedure sets "scan_state" according to the presence of the scanning clocks on the DPC module ‘mod_no’.

Scan_state value is valid only when the module works in one of imaging modes (FIFO_TCSPC_IMG or FIFO_ABS_IMG) (setting parameter MODE using SPC_set_parameter procedure). Otherwise the procedure returns error code -SPC_BAD_FUNC.

In imaging modes some input channels are reserved for scan clocks (pixel, line, frame). Scan clocks channels can be defined by setting the parameter CHAN_SPEC_NO.

The presence of the signals in scan clocks channels is detected and “scan_state” is set accordingly.

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 .. 15, DPC module and TDC number
* usage_degree	pointer to result variable

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

The procedure sets "usage_degree" with the value in the range from 0 to 1, which tells how occupied is FIFO memory of one of the TDC on the DPC module.

mod_no values 0 .. 7 means TDC 1 of module 0 .. 7

mod_no values 8 .. 15 means TDC 2 of module 0 .. 7

FIFO memory usage is set to 0 at the start of the 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, DPC module number

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

The procedure is used to start the measurement on the DPC module ‘mod_no’.

Before a measurement is started DPC parameters must be set (SPC_init or SPC_set_parameter(s)).

The procedure clears Macro time and FIFO buffers in active TDCs and then active TDCs are armed i.e. the photon collection is started.

----

short CVICDECL SPC_stop_measurement(short mod_no);

Input parameters:

mod_no	0 .. 7, DPC module number

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

The procedure is used to terminate a running measurement on the DPC module ‘mod_no’. The procedure action is different for different measurement modes.

For not imaging modes the measurement is stopped immediately in both TDCs.

For imaging modes real stop happens when next frame pulse appears (to ensure that whole frames are recorded). In this time period SPC_test_state will show the flag SPC_WAIT_FR. Another call to **SPC_stop_measurement** while waiting for the frame pulse will stop the measurement immediately.

DPC memory transfer functions:

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

Input parameters:

mod_no	0 .. 15, DPC module and TDC number
*count	pointer to variable which: - on input contains required number of 16-bit words -on output will be filled with number of 16-bit words written to the buffer ‘data’
*data	pointer to data buffer which will be filled

Return value:

0 no errors, <0 error code

The procedure reads data from the FIFO memory of the specified TDC and DPC module.

mod_no values 0 .. 7 means TDC 1 of module 0 .. 7

mod_no values 8 .. 15 means TDC 2 of module 0 .. 7

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

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	8 (FIFO_D230) for DPC module,
	other values are reserved for different SPC module types
* 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)
	bit 0 = 0 - no special meaning of the first entry in the file
	bit 1 = 1 –stream with raw data from TDC1
	bit 2 = 1 –stream with raw data from TDC2
	bit 3 = 1 –stream with raw data from TDC with active TTL inputs
	bit 8 = 1 –stream with DPC module data
	bit 9 = 1 - stream created in imaging mode
	bit 10 = 1 - stream contains raw data
what_to_read	defines which entries will be extracted
	bit 0 = 1 read valid photons,
	bit 1 = 1 read invalid photons – no meaning for DPC module,
	bit 2 = 1 read pixel clock events, bit 3 = 1 read line clock events,
	bit 4 = 1 read frame clock events,
	bit 5 = 1 read markers 3 - no meaning for DPC module

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. 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 was created in one of imaging modes. Bit 8 must also be set for DPC data.

If the file(s) contains raw data (just read using SPC_read_fifo without conversion to DPC .spc format) bits 1,2,3 define which TDC and which inputs were active. This is important to calculate correct channel number (1 ... 20) while extracting photons from the stream.

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/DPC module type which was used during the measurement. Possible ‘fifo_type’ values are defined in the spcm_def.h file. For DPC data value 8 (FIFO_D230) should be used.

‘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 created in imaging modes (bit 9 set in ‘stream_type’) – pixel, line, frame events can also be extracted (bits 2-5) separately (CHAN_SPEC_NO parameter defines which channels are treated as the scan clock channels.

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_dpc.c file for the extraction example.

----

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

Input parameters:

stream_hndl	handle of the initialised photons stream
stream_info	pointer to the stream info structure

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

The procedure fills ‘stream_info’ structure with the contents of DLL internal structure of the stream defined by handle ‘stream_hndl’. Procedure returns error, if ‘stream_hndl’ is not the handle of the stream opened using SPC_init_phot_stream function.

PhotStreamInfo structure is defined in the spcm_def.h file.

----

short CVICDECL SPC_get_photon (short stream_hndl, PhotInfo * phot_info);

Input parameters:

stream_hndl	handle of the initialised photons stream
phot_info	pointer to the photon info structure

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

The procedure can be used in a loop to extract subsequent photons from the opened photons stream defined by handle ‘stream_hndl’.

The procedure fills ‘phot_info’ structure with the information of the photon extracted from the current stream position. After extracting the 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, DPC 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	8 (FIFO_D230) for DPC module,
	other values are reserved for different SPC module types
stream_type	bit 0 has no special meaning for buffered streams, is set to 1
	bit 1 = 1 –stream with raw data from TDC1

bit 2 = 1 –stream with raw data from TDC2

bit 3 = 1 –stream with raw data from TDC with active TTL inputs

bit 8 = 1 –stream with DPC module data

bit 9 = 1 - stream created in imaging mode

bit 10 = 1 - stream contains raw data

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 – no meaning for DPC module,

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 - no meaning for DPC module

mt_clock	macro time clock, for DPC-230 1fs units (femto, 1e-15)
start01_offs	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 events– markers 0-2 (pixel, line, frame) can also be extracted (bits 2-4).

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 .. 15, DPC module and TDC 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 (during running FIFO measurement) from FIFO memory of the specified TDC and DPC module.

mod_no values 0 .. 7 means TDC 1 of module 0 .. 7

mod_no values 8 .. 15 means TDC 2 of module 0 .. 7

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.

Second frame(4bytes) of the header contains DPC start01 offset, which is TDC offset used to calculate real time scale. To get the value use SPC_get_start_offset procedure called after starting the measurement.

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_convert_dpc_raw_data (short tdc1_stream_hndl,
short tdc2_stream_hndl, short init,
char * spc_file, int max_per_call);

Input parameters:

tdc1_stream_hndl	handle of the initialised stream of photons read from TDC1
tdc2_stream_hndl	handle of the initialised stream of photons read from TDC2
init	1 for 1st call, 0 for subsequent calls
spc_file	file name of the destination .spc file
max_per_call	maximum number of events read from input streams in one call

Return value: 0: conversion finished, no errors, > 0 estimated rest to do in range 1000…1, <0: error code

The procedure is used in a loop to convert events from the opened photons streams of both TDC defined by handle 'tdc1(2)_stream_hndl'. One of the input stream handles can be equal NULL – it means that events from corresponding TDC are not used for conversion.

To get input stream handles use the procedure SPC_init_phot_stream.

To avoid blocking the computer the procedure reads and converts in one call up to 'max_per_call' events from each stream. First call must be done with 'init' = 1 – this call initialises conversion process. Subsequent calls, up to the end of conversion, must be done with 'init' = 0.

Converted photon entries are written to the destination 'spc_file'.

If error occurs the procedure returns error code < 0.

If no error, the procedure returns estimated progress value (rest to do) of the conversion process starting from 1000 down to 0. Value 0 means that all events from the opened input streams were analyzed and conversion was successfully finished.

Normally the input streams contain raw data read during the measurement from both TDCs using SPC_read_fifo procedure.

If the input streams contain data from both TDCs already converted to .spc format, the procedure will combine them into one .spc file sorted by macro time value.

See use_dpc.c file for the conversion example.

See DPC manual for .spc format description.

```
short CVICDECL SPC_get_start_offset (          short mod_no,
                                               short bank, unsigned long *ticks);
```

Input parameters:

mod_no	0 .. 7, DPC module number
bank	0 .. 1, TDC number
*ticks	pointer to value which will be filled with start offset ticks

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

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

The procedure is used to get start offset of the 1st photon recorded in fifo memory of TDC's number 'bank' on DPC module 'mod_no'. It must be called after starting the measurement (SPC_start_measurement).

The procedure returns SPC_RATES_NOT_RDY error, if start offset is not ready yet (nothing was written to the DPC fifo).

Start offset is used to calculate correct time scale of the photons. It is included as a 2nd frame in .spc raw data files.

If error occurs the procedure returns error code < 0.

See use_dpc.c file for the .spc files conversion example.

See DPC manual for .spc format description.

```
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.

===