

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

pcsdll32.doc 8.3.00



PCS-150 PCI-200
PCS-150A PCI-200A
32 Bit Dynamic Link Libraries
User Manual

Version 2.0

March 2000

Introduction

The PCS 32 bits Dynamic Link Library contains all functions to control the PCS-150, PCS-150A, PCI-200 and PCI-200A 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 32 bit DLL for Windows 95 and Windows NT is installed by a setup program. For installation, execute setup.exe from diskette No. 1 and follow the instructions of the setup program. The following files are installed:

PCSDLL32.DLL	Dynamic link library main file
PCSDLL32.LIB	import library file for Microsoft Visual C/C++, Borland C/C++, Watcom C/C++ and Symantec C/C++ compilers
PCS_DEF.H	Include file containing Types definitions, Functions Prototypes and Pre-processor statements
PCS150.INI	PCS DLL initialisation file
USE_DLL.XXX	Set of files to compile and run a simple example of using PCS DLL functions. The Source file of the example is the file use_pcs.c. The example was originally prepared under Borland C/C++ v.4.52. For use with other compilers choose the correct import library file for linking.

PCS-DLL Functions list

The following functions are implemented in the PCS-DLL:

Initialisation functions

PCS_init
PCS_get_init_status
PCS_get_id
PCS_test_id

Setup Functions to set and to read parameters

PCS_get_parameters
PCS_set_trig_slope
PCS_get_trig_slope
PCS_set_trig_source
PCS_get_trig_source
PCS_set_trig_threshold
PCS_get_trig_threshold
PCS_set_delay_range
PCS_get_delay_range
PCS_set_ini_delay
PCS_get_ini_delay
PCS_set_time_per_step
PCS_get_time_per_step
PCS_set_steps_per_curve
PCS_get_steps_per_curve
PCS_set_offset
PCS_get_offset
PCS_set_feedback
PCS_get_feedback
PCS_use_channel
PCS_is_channel_used
PCS_set_meas_mode
PCS_get_meas_mode
PCS_set_average_no
PCS_get_average_no
PCS_set_gate_width
PCS_get_gate_width
PCS_get_eeprom_data
PCS_LV_get_eeprom_data
PCS_write_eeprom_data
PCS_get_adjust_parameters
PCS_set_adjust_parameters

Status and Device Control Functions

PCS_init_measure
PCS_stop_measure
PCS_set_offset_dacs
PCS_set_delay_dac
PCS_arm
PCS_get_status
PCS_get_ini_dac
PCS_get_tps_dac

Functions to read results

PCS_get_averaged_sample
PCS_get_curve
PCS_get_sampleA
PCS_get_sampleB

Miscellaneous

PCS_set_mode

PCS_get_mode

Application Guide

Initialisation of the PCS 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 PCS module. This is accomplished by the function **PCS_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 PCS module
- sends the parameter values to the PCS control registers
- checks the module identification and the checksum of the EEPROM parameters

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file PCS150.INI or to start with PCS150.INI and to introduce the desired changes.

[pcs_base]

- ; Only pcs_base section with baseadr item has to be included in .ini file
- ; Other parameters are optional (default values are used if a parameter is not found)

```
baseadr= 0x380           ;base I/O address
simulation = 0           ; 0 - hardware mode(default) ,
                        ; >0 - simulation mode (see pcs_def.h for possible values)
```

[pcs_module]

```
meas_mode= 2           ;measurement mode 0- Sampling , 1- Boxcar ,2- Fixed(default)
ch_A_used= 1           ;0 -not used ,1 - used(default)
ch_B_used= 1           ;0 -not used ,1 - used(default)
offset_A= 0            ;offset of channel A [mV] ( should be in range -500 , 500)
                        ; (default =0)
offset_B= 0            ;offset of channel B [mV] ( should be in range -500 , 500)
                        ; (default =0)
trig_slope=1           ; 0 - positive(default) , 1 - negative
trig_source=0          ; 0 - internal , 1 - external(default)
threshold= -100        ;trigger threshold [mV] (-1000 , 1000) (default=0.0)
del_range=1000         ;delay range [ns] ( 10 - 20000) (default=10.0)
ini_delay=0            ;initial delay [ns] ( 0 - del_range)(default=0.0)
time_per_step=0.24     ;time per step [ns]
                        ;( 0.001 - 160 ,depends on actual delay range)
                        ;(default -will be calculated from delay range and steps)
steps=1024             ;number of steps per curve (64,128,256,512,1024(default))
feedback=0             ; 1 - feedback used, 0 - not used(default)
samples=1              ;number of samples averaged(default=1)
gate_width = 5.0       ; gate width [ns] for module PCI-200 (default=2.0 ns )
```

After calling the **PCS_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 individual parameters, the functions **PCS_get_....** return the individual parameter values. To set an individual parameter the functions **PCS_set_....** are provided.

To give the user access to the whole set of parameters, the function **PCS_get_parameters** is provided. This function transfers the parameter values from the internal structures of the DLLs into a structure of the type **PCSdata** (see **pcs_def.h**) which has to be defined by the user. The parameter values in this structure are described below.

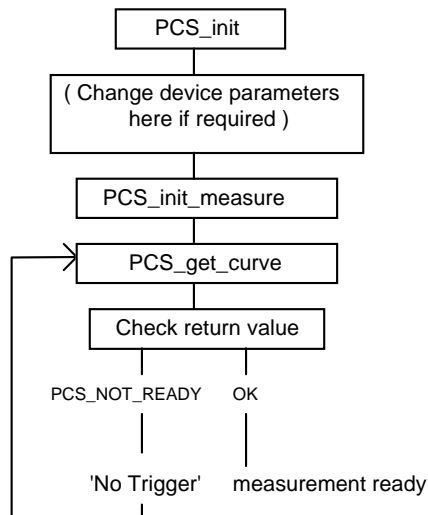
unsigned short base_adr	base I/O address (0..0x3E0,default 0x380)
short init;	set to init status value after initialisation
short hardware;	mode of driver operation - PCS_HARD or PCS_SIMUL..
short init;	set to initialisation result code
short module_type;	module type - PCS(_A) or PCI(_A) (read from EEPROM)
short meas_mode;	measurement mode - Sampling, Boxcar, Fixed
short ch_A_used;	channel A used in measurement
short ch_B_used;	channel B used in measurement
short offset_A;	offset of channel A
short offset_B;	offset of channel B
short trig_slope;	trigger slope 0 - positive,1 - negative
short trig_source;	trigger source 0 - internal,1 - external
float trig_threshold;	trigger threshold [mV] (-1000 , 1000)
float del_range;	delay range [ns]
float ini_delay;	initial delay [ns]
float tps;	time per step [ns]
short steps;	number of steps per curve
short samples;	no of samples averaged
short feedback_used;	1 - use feedback, 0 - not
short tps_dac;	time per step DAC increment
short ini_dac;	initial delay DAC value
short status;	actual state of PCS module - see Status defines
short sum_A;	actual sum in channel A (important, when feedback used)
short sum_B;	actual sum in channel B (important, when feedback used)
float gate_width;	gate width in ns - only for PCI modules

Simple Measurements Sequences

In the block diagram below a simple measurement sequence is shown which measures one curve in the operation mode and with the system parameters specified in the PCS150.INI file.

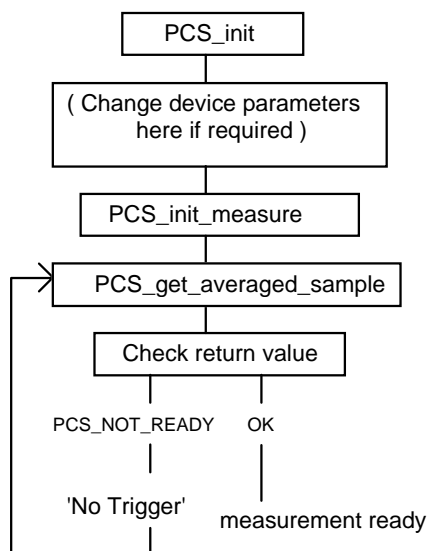
At the beginning PCS_init reads the system parameters from the PCS150.INI file and writes the parameter values into the PCS or PCI registers. We recommend to use a PCS150.INI file with the desired parameter set. However, if individual parameters have to be changed this should be done after the call of PCS_init.

When the parameters are set the measurement system is initialised by PCS_init_measure. Then the PCS_get_curve function is called to start the measurement and to write the measurement data into the data buffers specified in the PCS_get_curve call. When the specified number of points with the specified number of averagings have been measured the PCS_get_curve function returns with the error status OK (0). However, if the measurement could not be finished (e.g. because of a timeout during the waiting for trigger), the function returns with 'PCS_NOT_READY'. In this case the program gives a message that there was no trigger and repeats the call of PCS_get_curve until the trigger appears.



A program sequence to measure one sample is shown in the next figure. Again, at the beginning the measurement parameters are read from the initialisation file and sent to the PCS module by `PCS_init`. If some of the parameters are to be changed this can be done now by calling the appropriate `PCS_set_...` function.

When the parameters are set the measurement system is initialised by `PCS_init_measure`. Then the `PCS_get_averaged_sample` function is called to start the measurement and to write the measurement data into the data buffers specified in the `PCS_get_averaged_sample` call. When the signal point has been measured with the specified number of averagings the `PCS_get_averaged_sample` function returns with the error status 0. However, if the measurement could not be finished (e.g. because of a timeout during the waiting for trigger), the function returns with `'PCS_NOT_READY'`. In this case a the program gives the message `'No Trigger'` and repeats the call of `PCS_get_averaged_sample` until the trigger appears.



Error Handling

Most of the PCS DLL function return an error status. Return values ≥ 0 indicate error free execution. A value < 0 indicates that an error occurred during the execution. The error codes are

defined in the PCS_def.h file . The meaning of the error codes is described in the table below. Although not absolutely required we recommend to check the return value after each function call.

Error codes

PCS_NONE	0	no error
PCS_OPEN_FILE	-1	can't open configuration file
PCS_FILE_NVALID	-2	not valid configuration file
PCS_MEM_ALLOC	-3	memory allocation error
PCS_READ_STR	-4	can't read string from file
PCS_NOT_INIT	-5	module not initialised
PCS_WRONG_ID	-6	wrong module id read from PCS
PCS_MEAS_ON	-7	can't execute the function when measure in progress
PCS_EEPROM_READ	-8	error during reading EEPROM
PCS_EEPROM_WRITE	-9	error during writing to EEPROM
PCS_EEPROM_CHKSUM	-10	Wrong EEPROM checksum
PCS_BIG_INI	-11	initial delay changed - was too big
PCS_BIG_TPS	-12	time per step changed - was too big
PCS_BIG_OFF	-13	channel offset changed - was too big
PCS_BIG_TPS_INI	-14	time per step and init delay changed- were too big
PCS_BAD_THRE	-15	threshold value out of limits - is recalculated
PCS_BAD_RANGE	-16	delay range value out of limits - is recalculated
PCS_BAD_STEPS	-17	steps value out of limits - is recalculated
PCS_CHAN_OFF	-18	channel not active
PCS_PCS_ARMED	-19	PCS already armed
PCS_NOT_ARMED	-20	PCS not armed
PCS_NOT_READY	-21	sample value not available - PCS not triggered
PCS_TOUT	-22	timeout during reading sample
PCS_WRONG_MOD	-23	can't execute function - wrong module type
PCS_EEP_WR_DI S	-24	write access to EEPROM denied

Description of the PCS DLL Functions

short PCS_init (char far *config_file)

Input parameters: config_file - string with configuration file name

Return value: 0 no errors, <0 error code

PCS_init initialises the internal structures of the PCS library using the parameters from a configuration file config_file (suggested extension .ini). At the beginning the function reads and tests the module identification byte ('module id'). This byte is hard-wired in the PCS/PCI module. If the module id is correct (i.e. the module is present)

- the adjust parameters are read from module EEPROM
- the adjust DACs are set according to the adjust parameters
- the internal registers are initialised according to the system parameters in config_file

short PCS_get_init_status (void)

Input parameters: none

Return value: initialisation result code

PCS_get_init_status returns the initialisation result code set by the function PCS_init.

The possible values are shown below (see also pcs_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

short PCS_get_id (short *id)

Input parameters: id - pointer to variable to be set to id value

Return value: 0 no errors, <0 error code

PCS_get_id returns the PCS module id.

short PCS_test_id (void)

Input parameters: none

Return value: 0 if module id is ok, <0 error code

PCS_test_id reads the PCS module id and compares it to correct value.

short PCS_get_parameters (PCSdata * data)

Input parameters: none

Return value: 0 no errors, <0 error code

After calling the PCS_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 **PCS_get_parameters** is provided. This function transfers the parameter values from the internal structures of the DLLs into a structure of the type PCSdata (see pcs_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;	set to init status value after initialisation
short hardware;	mode of driver operation - PCS_HARD or PCS_SIMUL..
short init;	set to initialisation result code
short module_type;	module type - PCS(_A) or PCI(_A) (read from EEPROM)
short meas_mode;	measurement mode - Sampling, Boxcar, Fixed
short ch_A_used;	channel A used in measurement
short ch_B_used;	channel B used in measurement
short offset_A;	offset of channel A
short offset_B;	offset of channel B
short trig_slope;	trigger slope 0 - positive,1 - negative
short trig_source;	trigger source 0 - internal,1 - external
float trig_threshold;	trigger threshold [mV] (-1000 , 1000)
float del_range;	delay range [ns]
float ini_delay;	initial delay [ns]
float tps;	time per step [ns]
short steps;	number of steps per curve
short samples;	no of samples averaged
short feedback_used;	1 - use feedback, 0 - not
short tps_dac;	time per step DAC increment
short ini_dac;	initial delay DAC value
short status;	actual state of PCS module - see Status defines
short sum_A;	actual sum in channel A (important, when feedback used)
short sum_B;	actual sum in channel B (important, when feedback used)
float gate_width;	gate width in ns - only for PCI modules

short PCS_set_trig_slope (short slope)

Input parameters: slope=0 - positive slope, slope=1 - negative slope

Return value: 0 no errors, <0 error code

PCS_set_trig_slope sets the trigger slope.

short PCS_get_trig_slope (short *slope)

Input parameters: slope - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_trig_slope loads the 'slope' variable with the actual trigger slope value.

short PCS_set_trig_source (short source)

Input parameters: source=0 - internal, source=1 - external

Return value: 0 no errors, <0 error code

PCS_set_trig_source sets the trigger source.

short PCS_get_trig_source (short *source)

Input parameters: source - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_trig_source loads the 'source' variable with the actual trigger source value.

short PCS_set_trig_threshold (float threshold)

Input parameters: threshold = value in mV, from -1000 to +1000

Return value: 0 no errors, <0 error code

PCS_set_trig_threshold sets the trigger threshold.

short PCS_get_trig_threshold (float *threshold)

Input parameters: threshold - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_trig_threshold loads the 'threshold' variable with the actual trigger threshold value in mV.

short PCS_set_delay_range (float range)

Input parameters: range = value in ns, from 10 to 20,000 (10 to 100,000 for the -A modules)

Return value: 0 no errors, <0 error code

PCS_set_delay_range sets the delay range.

Remarks: In the 'Sampling' or 'Boxcar' mode the setting of 'Delay Range' may require a recalculation of 'time per step' (tps). If tps is still below the minimum value, also the 'initial delay' parameter can be set to 0.

short PCS_get_delay_range (float *range)

Input parameters: range - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_delay_range loads the 'range' variable with the actual 'Delay Range' in ns.

short PCS_set_ini_delay (float delay)

Input parameters: delay = value in ns from 0 to the actual delay range (see also function PCS_set_delay_range)

Return value: 0 no errors, <0 error code

PCS_set_ini_delay sets the initial delay value.

Remarks: Initial Delay is recalculated to fit to

- the actual Delay Range (in all measurement modes)
- the actual No of Steps and Time per Step values (in the Sampling and Boxcar mode only)

short PCS_get_ini_delay (float *delay)

Input parameters : delay - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_ini_delay loads the 'delay' variable with the actual 'Initial Delay' value in ns.

short PCS_set_time_per_step (float tps)

Input parameters: tps = value in ns, from 0.001 to 160

Return value: 0 no errors, <0 error code

PCS_set_time_per_step sets the 'Time per Step' value.

Remarks: Time per step is recalculated in the 'Sampling' and 'Boxcar' mode to fit to the actual Delay Range, No of Steps and Initial Delay values.

short PCS_get_time_per_step (float *tps)

Input parameters: tps - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_time_per_step loads the 'tps' variable with actual 'Time per Step' value.

short PCS_set_steps_per_curve (short steps)

Input parameters: steps = 64, 128, 256, 512 or 1024

Return value: 0 no errors, <0 error code

PCS_set_steps_per_curve sets the 'Steps per Curve' parameter.

Remarks: steps_per_curve is recalculated in the 'Sampling' or 'Boxcar mode' to fit to the actual 'Delay Range', 'Steps per Curve' and 'Initial Delay' values.

short PCS_get_steps_per_curve (short *steps)

Input parameters: steps - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_steps_per_curve loads the 'steps' variable with actual 'Steps per Curve' value.

short PCS_set_offset (short chan, short offset)

Input parameters: chan = 0 - channel A , chan=1 - channel B

offset = new offset value in mV, from -500 to 500

PCS_is_channel_used gets an information whether a signal channel is switched on or off (state=0 - channel off, state=1 - channel on).

short PCS_set_meas_mode(short mode)

Input parameters: mode=0 - Sampling, mode=1 - Boxcar, mode=2 - Fixed Delay

Return value: 0 no errors, <0 error code

PCS_set_meas_mode defines the measurement mode.

Remarks: Setting Sampling or Boxcar mode may require to recalculate 'Time per Step' (tps). If tps is still below the minimum value, also 'Initial Delay' can be set to 0.

short PCS_get_meas_mode(short *mode)

Input parameters: mode - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_meas_mode loads the 'mode' variable with the actual measurement mode (mode=0 - Sampling, mode=1 - Boxcar, mode=2 - Fixed Delay).

short PCS_set_average_no(short averaged_no)

Input parameters: averaged_no, from 1 to 4096

Return value: 0 no errors, <0 error code

PCS_set_average_no defines the number of samples to be averaged. This number is used in the measurement functions PCS_get_averaged_sampleA(B) and PCS_measure_curve.

short PCS_get_average_no(short *averaged_no)

Input parameters: averaged_no - pointer to result value

Return value: 0 no errors, <0 error code

PCS_get_average_no loads the 'averaged_no' variable with the actual number of samples to be averaged by the measurement functions PCS_get_averaged_sampleA(B) and PCS_measure_curve.

The function is prepared for use in a LabView environment to read the contents of the EEPROM of the PCS/PCI module. It works in the same way as PCS_get_eeeprom_data function but instead of the pointer to PCS_EEP_Data structure it sends as parameters separate pointers to strings from PCS_EEP_Data structure.

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

short PCS_save_eeeprom_data (PCS_EEP_Data far *eep_data);

Input parameters: *eep_data: pointer to a structure which will be sent to EEPROM

Return value: 0 no errors, <0 error code

The function is used to write data to the EEPROM of an PCS/PCI module by the manufacturer. To prevent corruption of the adjust data the function can be used by B&H only.

short PCS_get_adjust_parameters (PCS_Adjust_Para far *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 PCS_init or PCS_get_eeeprom_data or - not recommended - set by PCS_set_adust_parameters.

The structure "PCS_Adjust_Para" is defined in the file pcs_def.h.

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

short PCS_set_adjust_parameters (PCS_Adjust_Para far *adjpara);

Input parameters: *adjpara: pointer to result structure

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 PCS_init. The next call to PCS_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 impaired by wrong adjust values.

The structure "PCS_Adjust_Para" is defined in the file pcs_def.h.

Status and Device Control Functions

short PCS_init_measure (void)

Input parameters: none

Return value: 0 no errors, <0 error code

PCS_init_measure initialises the measurement system with the actual system parameters. It is called before a measurement is started. Therefore, all measurement parameters (trigger parameters, delay parameters, measurement mode, channel offset) must be set before calling the function.

After calling PCS_init_measure signal samples can be collected by low level functions (PCS_arm, PCS_get_sampleA(B)) or by high level functions PCS_get_averaged_sample or PCS_get_curve.

Remarks:

The function returns an error when the PCS is already armed.

The function sets the offset DACs for the used channels according to the actual offset values and the delay DAC according to the initial delay value.

The function must be used also after changing Delay Range, Time per Step, Steps per Curve or Channel Offset.

void PCS_stop_measure (void)

Input parameters : none

Return value : none

The function resets the measurement control variables of the library function. It must be called only if a measurement function did not finish the measurement correctly (i.e. returned with an error).

void PCS_set_offset_dacs (void);

Input parameters: none

Return value: none

PCS_set_offset_dacs is used to set the offset DACs in the used channels (switched by PCS_use_channel). It must be called before calling the low level functions PCS_get_sampleA or PCS_get_sampleB.

If the feedback is not used only one call at the beginning or after changing the offset value(s) is required. Use this function only when you build measurement procedures from low level functions calls.

void PCS_set_delay_dac (short val);

Input parameters: val = new DAC value, from 0 to 4095

Return value: none

PCS_set_delay_dac sets the Delay DAC. It can be used for special delay control in conjunction with the functions PCS_get_averaged_sample or PCS_get_sampleA(B). For normal curve recording we recommend the function PCS_get_curve which incorporates the delay control.

Use this function only when you build measurement procedures from low level functions calls.

Remarks:

The input parameters should be calculated by the equation:

$$\text{pcs.ini_dac} + i * \text{pcs.tps_dac},$$

where i= measurement step number (0 to steps-1), for Fixed Delay i=0

Values of pcs.ini_dac and pcs.tps_dac are calculated during the execution of functions set_delay_range, set_ini_delay, set_time_per_step.

short PCS_arm (void)

Input parameters: none

Return value: 0 no errors, <0 error code

PCS_arm can be used to arm the PCS/PCI (i.e. to start the measurement) in conjunction with the low level measurement functions PCS_get_sampleA or PCS_get_sampleB.

Remarks: Use this function only when you build measurement procedures from low level functions calls (PCS_get_sampleA or PCS_get_sampleB).

It is not possible to arm the PCS when it is already armed or a previous sample is not read from the used channel(s).

short PCS_get_status(void);

Input parameters: none

Return value: actual value of internal PCS/PCI status, error code when < 0

PCS_get_status returns the actual value of the internal PCS/PCI status.

The function is required only if when you build special measurement procedures from low level function calls. The status definitions are at the top of the pcs_def.h file.

short PCS_get_ini_dac(void)

Input parameters: none

Return value: actual ini_dac value, error code when < 0

PCS_get_ini_dac returns the internal DAC value of 'Initial Delay' (the internal ini_dac variable). The function is required only if when you build special measurement procedures from low level function calls.

short PCS_get_tps_dac(void);

Input parameters: none

Return value: actual tps_dac value, error code when < 0

PCS_get_tps_dac returns the value of the internal tps_dac variable (the internal DAC value).

The function is required only if when you build special measurement procedures from low level function calls.

Functions to read results

short PCS_get_averaged_sample (float far *aver_valA, float far *aver_valB)

Input parameters: aver_valA(B) - pointer to variable which is set with the averaged sample value

Return value: 0 no errors, <0 error code

PCS_get_averaged_sample is used to get averaged sample values from the used channels in the 'Fixed Delay' or 'Boxcar' measurement mode.

The function starts the measurement, waits for the trigger and adds the samples taken from the signal. The function returns when either the specified number of samples have been added or when a timeout occurs during the waiting for the trigger pulse. If the return was caused by a timeout the function returns with an error code (PCS_not_ready). If PCS_get_averaged_sample is started again after a return by timeout it will proceed from the previous state, i.e. finish the measurement. If all samples were read correctly (without timeout), the procedure sets aver_valA(B) with the averaged value and returns with OK (0). If there were errors during the measurement (timeout), it does not set aver_valA(B) and returns with an error code (PCS_NOT_READY).

If the function is used in the sampling mode only one sample will be taken - independently of the averaged_no value.

Before using this function the user should set:

- the number of samples to be averaged (averaged_no), by PCS_set_average_no, default value=1)
- the Offset DACs, by PCS_set_offset_dacs
- the Delay DAC (function PCS_set_delay_dac), in the Fixed Delay mode before the first sample, for recording waveforms before each sample

short PCS_get_curve (float far *curveA, float far *curveB)

Input parameters: curveA(B) - pointers to buffers which are filled with the curve points

Return value: 0 no errors, <0 error code

PCS_get_curve is used to measure a complete curve with one or both signal channels. It can be used in all measurement modes.

The samples of the curve(s) are measured with waiting for trigger up to a timeout time of approximately 20 ms. If all samples were read correctly (without timeout) the function returns 'OK' (0). If there were errors during reading samples (timeout), the function returns with an error code (PCS_NOT_READY).

In the 'Boxcar' and 'Fixed Delay' modes the function measures 'steps_per_curve' averaged samples on one or both channels A(B) and writes the results to the input data buffers.

In the 'Sampling' mode the function measures 'steps_per_curve' samples in one or both channels A(B) and adds them to temporary buffers. This procedure is executed n times, with n = No of samples averaged. At the end the averaged values are calculated for all points of the Curve(s) and sent to the data buffers (curveA(B)).

If PCS_get_curve is started again after a return by timeout it will proceed from the previous state, i.e. finish the measurement.

Before using this function the following parameters must be set:

- 'No of Samples Averaged' (averaged_no), by the function PCS_set_average_no, default value is 1
- 'Delay Range', 'Initial Delay', 'Time per Step', 'No of Steps per Curve'

The Data buffers curveA(B) must have enough space for the whole curve. The number of elements is given by the 'steps_per_curve' parameter set by the PCS_set_steps_per_curve function.

short PCS_get_sampleA (short far *value)

short PCS_get_sampleB (short far *value)

Input parameters: value = pointer to a variable which is be set with measured sample value

Return value: 0 no errors, <0 error code

PCS_get_sample is used to read a sample value from one of the channel. It is used to create special measurement sequences which cannot be carried out by the normal high level functions PCS_get_averaged_sample and PCS_get_curve.

The procedure tests whether the sample in channel A(B) is ready. If yes, it sets the value and returns OK (0). If the PCS is not armed or the sample is still not available it returns an error code.

Remarks:

PCS_get_sample can produce wrong results, if the procedures PCS_init_measure and PCS_arm were not called before or if the 'Offset' DAC and the 'Delay' DAC are not set.

Miscellaneous functions

short PCS_set_mode (short mode)

Input parameters: mode=0 - hardware mode, simulation mode values – see pcs_def.h

Return value: 0 no errors, <0 error code

PCS_set_mode sets the mode of the driver operation. To operate the PCS/PCI module mode=0 is required. Mode=1 sets a ‘simulation’ mode which replaces all hardware functions by software simulation sequences. The simulation mode is used to create demonstration software or to develop and debug PCS/PCI applications on a computer without a PCS/PCI module.

short PCS_get_mode (void);

Input parameters: none

Return value: mode=0 - hardware mode, simulation mode values – see pcs_def.h

PCS_get_mode returns the mode of driver operation.