

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

shm_dll.doc



SHM
32 Bit Dynamic Link Libraries
User Manual
Version 1.1 January 2003

Introduction

The SHM 32 bits Dynamic Link Library contains all functions to control the SHM modules. The functions work under Windows 9x/NT/ME/2K/XP. 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:

SHM.DLL	dynamic link library main file
SHM.LIB	import library file for Microsoft Visual C/C++, Borland C/C++, Watcom C/C++ and Symantec C/C++ compilers
SHM_DEF.H	Include file containing types definitions, functions prototypes and pre-processor statements
SHM180.INI	SHM DLL initialisation file
SHM_DLL.DOC	This description file
USE_SHM.C	A simple example of using SHM DLL functions. (Please choose the correct import library file to link in your compiler)

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

SHM-DLL Functions list

The following functions are implemented in the SHM-DLL:

Initialisation functions:

- SHM_init
- SHM_test_if_active
- SHM_get_init_status
- SHM_get_mode
- SHM_set_mode
- SHM_get_version
- SHM_get_module_info
- SHM_get_error_string

Setup functions:

- SHM_get_parameter
- SHM_set_parameter
- SHM_get_parameters
- SHM_set_parameters
- SHM_get_eeprom_data
- SHM_write_eeprom_data
- SHM_get_adjust_parameters
- SHM_set_adjust_parameters

Status functions:

- SHM_test_if_busy
- SHM_read_status

Measurement control functions:

SHM_start_measure

SHM_stop_measure

SHM memory r/w functions:

SHM_read_data

SHM_fill_memory

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

An identical set of functions is available for environments like Visual Basic which requires `_stdcall` calling convention. Names of these functions have 'std' letters after 'SHM', for example, `SHMstd_get_parameter` is the `_stdcall` version of `SHM_get_parameter`.

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

Application Guide

Initialisation of the SHM Measurement Parameters

Before the SHM module can be used the 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 SHM module. This is accomplished by the function **SHM_init**.

The SHM DLL Functions are able to control up to four SHM modules on several PCI bus(es).

The **SHM_init** function

- reads the parameter values from a specified initialisation file
- sends the parameter values to the SHM control registers in an active SHM module
- performs a hardware test (EEPROM checksum test) of active SHM module

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file `SHM180.INI` or to start with `SHM180.INI` and to introduce the desired changes.

```
; SHM180 initialisation file
; SHM parameters have to be included in .ini file only when parameter
; value is different from default.
; module section (SHM_module1-4) is required for each existing SHM module

[shm_base]
simulation = 0                ; 0 - hardware mode(default) ,
                             ; >0 - simulation mode (see shm_def.h for possible values)

[shm_module1]                ; SHM module 1 hardware parameters

active = 1                    ; module active - can be used (default = 0 - not active)
enable_meas = 1               ; enable/disable(1/0) measurement , default = enable
trigger = 2                   ; external trigger condition
                             ; active low(1),active high(2) (default)
trig_level = 1.0              ; trigger level for ( -1.0V ... +1.0V , default 0.1)

delay = 10.e-9                ; delay for the 1st point in seconds,
                             ; 10 e-9(default).. 655.360E-6

                             ; all channel offsets have range -25mV ... 25mV and
```

```

; default value 0 mV
ch_offs_A1 = 0 ; offset of channel A1
ch_offs_A2 = 0 ; offset of channel A2
ch_offs_A3 = 0 ; offset of channel A3
ch_offs_A4 = 0 ; offset of channel A4
ch_offs_B1 = 0 ; offset of channel B1
ch_offs_B2 = 0 ; offset of channel B2
ch_offs_B3 = 0 ; offset of channel B3
ch_offs_B4 = 0 ; offset of channel B4

HP_filt_A = 0 ; HP filter group A setting
; 0- off (default), 1- 100µs, 2- 10µs, 3- 1µs
HP_filt_B = 0 ; HP filter group B setting
; 0- off (default), 1- 100µs, 2- 10µs, 3- 1µs

; possible values for all gains: 1-8,16,24,32,40,48,56
; default 1
ch_gain_A1 = 1 ; gain of channel A1
ch_gain_A2 = 1 ; gain of channel A2
ch_gain_A3 = 1 ; gain of channel A3
ch_gain_A4 = 1 ; gain of channel A4
ch_gain_B1 = 1 ; gain of channel B1
ch_gain_B2 = 1 ; gain of channel B2
ch_gain_B3 = 1 ; gain of channel B3
ch_gain_B4 = 1 ; gain of channel B4

LP_filt_A = 0 ; LP filter group A setting
; 0- 30ns ( default ), 1- 100ns, 2- 300ns, 3- 1000ns
LP_filt_B = 0 ; LP filter group B setting
; 0- 30ns ( default ), 1- 100ns, 2- 300ns, 3- 1000ns
invert_chan = 0 ; channels inverted , 8-bit value ,default = 0,
; bit = 0 - channel not inverted, bit = 1 - channel inverted
; bit0 - chan.A1... bit3 - chan.A4,
; bit4 - chan.B1... bit7 - chan.B4

; all channel digital(software) offsets have range -100% ... 100% and
; default value 0 %
ch_digoffs_A1 = 0.0 ; digital offset of channel A1
ch_digoffs_A2 = 0.0 ; digital offset of channel A2
ch_digoffs_A3 = 0.0 ; digital offset of channel A3
ch_digoffs_A4 = 0.0 ; digital offset of channel A4
ch_digoffs_B1 = 0.0 ; digital offset of channel B1
ch_digoffs_B2 = 0.0 ; digital offset of channel B2
ch_digoffs_B3 = 0.0 ; digital offset of channel B3
ch_digoffs_B4 = 0.0 ; digital offset of channel B4

; all channel digital(software) zoom have range 1 .. 64 and
; default value 1
ch_digzoom_A1 = 0.0 ; digital zoom of channel A1
ch_digzoom_A2 = 0.0 ; digital zoom of channel A2
ch_digzoom_A3 = 0.0 ; digital zoom of channel A3
ch_digzoom_A4 = 0.0 ; digital zoom of channel A4
ch_digzoom_B1 = 0.0 ; digital zoom of channel B1
ch_digzoom_B2 = 0.0 ; digital zoom of channel B2
ch_digzoom_B3 = 0.0 ; digital zoom of channel B3
ch_digzoom_B4 = 0.0 ; digital zoom of channel B4

start_ptr=0 ; start point of collection( 0(default) ... 16383)
samples = 100 ; no of samples to collect( 1 ... 16383 ), default 100

```

```

accu = 0                ; accumulation on(1) / off(0) - default
no_of_accum = 1        ; no of accumulations( 1(default) ... 65535 )

[shm_module2]         ; SHM module 2 hardware parameters

active = 1             ; module active - can be used (default = 0 - not active)

[shm_module3]         ; SHM module 3 hardware parameters

active = 1             ; module active - can be used (default = 0 - not active)

[shm_module4]         ; SHM module 4 hardware parameters

active = 1             ; module active - can be used (default = 0 - not active)

```

After successful initialisation the module is locked to prevent that other application can access it. Therefore a SHM module can only be initialised if it is not in use (i.e. locked) by another application. If, for any reason, a locked module must be initialised, it can be done by using the function **SHM_set_mode** with the parameter 'force_use' = 1.

After a **SHM_init** call we recommend to call the **SHM_test_if_active** function to check whether (and which) SHM module is active. Only active modules can be operated further. It is recommended (but not required) to check also the initialisation status (by **SHM_get_init_status**) of the used module. If the initialisation was not successful for any reason the initialisation status shows the error (see shm_def.h for possible values).

If several SHM modules are present the modules are numbered in the order of their serial numbers, i.e. module 1 is the module with the lowest serial number.

Additional information about SHM modules can be obtained by calling **SHM_get_module_info** function. The function fills the SHMModInfo structure which is described below:

```

short module_type      module type : 180- SHM-180
short bus_number       PCI bus number
short slot_number      slot number on PCI bus
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
char serial_no[18]     module serial number

```

After calling the **SHM_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 **SHM_get_parameters** is provided. This function transfers the parameter values from the internal structures of the DLLs into a structure of the type 'SHMdata' (see shm_def.h) which has to be declared by the user. The parameter values in this structure are described below:

```

unsigned short base_adr  base I/O address on PCI bus
short init              set to initialization result code

short active            most of the library functions are executed
                       only when module is active ( not 0 )

short trigger           external trigger condition -

```

	active low(1), active high(2)
float trig_level	trigger threshold level -1V.. +1V
float time_per_point	time of 1 point in mikrosec 0.01 .. 655.35
float delay	delay for 1st point, 10 ns .. 655.360E-6 s
float ch_offs[8]	offset of channel 1-8, -25 .. +25 mV
short HPfilter[2]	High Pass Filter setting for both groups of channels 0- off, 1- 100µs, 2- 10µs, 3- 1µs
short ch_gain[8]	gain of channel 1-8 possible values 1-8,16,24,32,40,48,56
short LPfilter[2]	Low Pass Filter setting for both groups of channels 0- 30ns, 1- 100ns, 2- 300ns, 3- 1000ns
unsigned short start_ptr	memory start pointer (0(default) ... 16383)
short accumulate	accumulate or overwrite values in memory
unsigned long samples	no of samples to collect (1 ... 16383), default 100*/
unsigned long no_of_accum	number of accumulations 1(default) ... 65535
short invert_chan	channels inverted , 8-bit value ,default = 0, bit = 0 - channel not inverted, bit = 1 - channel inverted bit0 - chan.A1... bit3 - chan.A4, bit4 - chan.B1... bit7 - chan.B4 */
short enable_meas	measurement enabled/disabled(1/0)
float ch_digoffs[8]	digital(software) offset of channel 1-8, -100 .. +100 %, default 0%
short ch_digzoom[8]	digital(software) zoom of channel 1-8, 1 .. 64, default 1

To send the complete parameter set back to the DLLs and to the SHM module (e.g. after changing parameter values) the function **SHM_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 **SHM_set_parameters** by **SHM_get_parameters**.

Single parameter values can be transferred to or from the DLL and module level by the functions **SHM_set_parameter** and **SHM_get_parameter**. To identify the desired parameter, the parameter identification par_id is used. The parameter identification keywords are defined in shm_def.h.

Memory Configuration

The memory is organised in two banks which contain the data for 8 channels. The channel data are 32 bit 2complement ADC values. The measurement memory length 16384 words for each counter channel.

Memory Read/Write Functions

Reading the memory of the SHM module is accomplished by the function **SHM_read_data**. To fill the memory with a constant value (or to clear the memory) the function **SHM_fill_memory** is available.

Standard Measurements

The most important measurement functions are listed below.

The **SHM_test_if_busy** function is used to control the measurement loop. It sets a busy variable according to the current state of the measurement. The state of all active modules is taken into account in the return value:

- 0 - all active SHM modules have finished the measurement,
- 1 - the measurement is still running (at least) in one SHM module, no modules are waiting for an external trigger
- 2 - at least one module is waiting for the external trigger

The **SHM_read_status** function returns the current status of a particular SHM module. The most important status bits delivered by the function are listed below (see also shm_def.h).

ARMED	0x1	module is armed
TRGED	0x10	module is triggered

SHM_start_measure starts the measurement in all active SHM modules with the parameters set before by the SHM_init, SHM_set_parameters or SHM_set_parameter functions. The measurement starts to collect ADC values in subsequent points in SHM memory from the address START_PTR. The measurement stops after recording SAMPLES points.

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

In the figure below block diagram of simple measurement routine is given.

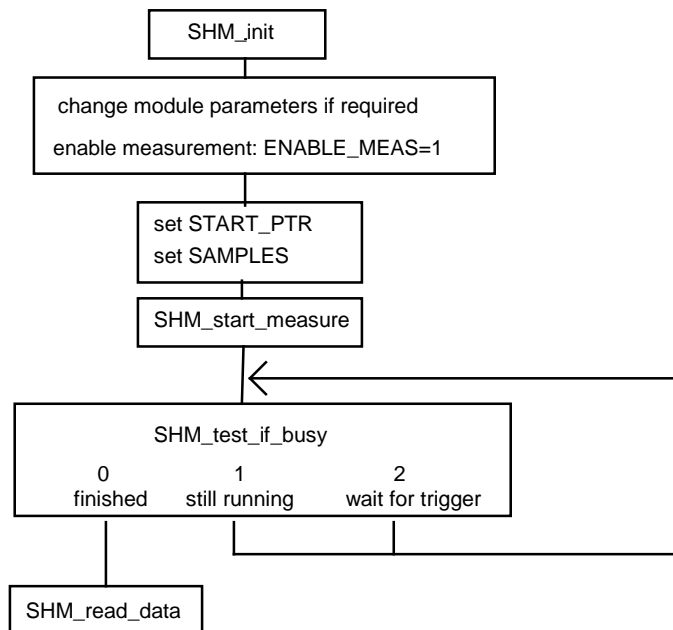


Fig1: Simple measurement

Error Handling

Each SHM DLL function returns an error status. Return values ≥ 0 indicate error free execution. A value < 0 indicates that an error has occurred. The meaning of a particular error code can be found in the shm_def.h file and can be read using **SHM_get_error_string**. We recommend to check the return value after each function call.

Using DLL functions in LabView environment

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

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

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

For structures defined in include file xxx_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 'XXX' (for example XXX_LV_get_module_info), and if found in xxx_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 SHM DLL Functions

short CVICDECL **SHM_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 the SHM module can be used the 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 SHM module. This is accomplished by the function **SHM_init**. The **SHM_init** function

- reads the parameter values from a specified initialisation file
- sends the parameter values to the SHM control registers on active SHM module
- performs a hardware test (EEPROM checksum test) of active SHM module

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file SHM180.INI or to start with SHM180.INI and to introduce the desired changes.

; SHM180 initialisation file
; SHM parameters have to be included in .ini file only when parameter
; value is different from default.
; module section (SHM_module1-4) is required for each existing SHM module

[shm_base]
simulation = 0 ; 0 - hardware mode(default) ,
; >0 - simulation mode (see shm_def.h for possible values)

[shm_module1] ; SHM module 1 hardware parameters

active = 1 ; module active - can be used (default = 0 - not active)
enable_meas = 1 ; enable/disable(1/0) measurement , default = enable
trigger = 2 ; external trigger condition
; active low(1),active high(2) (default)
trig_level = 1.0 ; trigger level for (-1.0V ... +1.0V , default 0.1)

delay = 10.e-9 ; delay for the 1st point in seconds,
; 10 e-9(default).. 655.360E-6

; all channel offsets have range -25mV ... 25mV and
; default value 0 mV
ch_offs_A1 = 0 ; offset of channel A1
ch_offs_A2 = 0 ; offset of channel A2
ch_offs_A3 = 0 ; offset of channel A3

```

ch_offs_A4 = 0           ; offset of channel A4
ch_offs_B1 = 0           ; offset of channel B1
ch_offs_B2 = 0           ; offset of channel B2
ch_offs_B3 = 0           ; offset of channel B3
ch_offs_B4 = 0           ; offset of channel B4

HP_filt_A = 0           ; HP filter group A setting
                        ; 0- off (default), 1- 100µs, 2- 10µs, 3- 1µs
HP_filt_B = 0           ; HP filter group B setting
                        ; 0- off (default), 1- 100µs, 2- 10µs, 3- 1µs

                        ; possible values for all gains: 1-8,16,24,32,40,48,56
                        ; default 1
ch_gain_A1 = 1          ; gain of channel A1
ch_gain_A2 = 1          ; gain of channel A2
ch_gain_A3 = 1          ; gain of channel A3
ch_gain_A4 = 1          ; gain of channel A4
ch_gain_B1 = 1          ; gain of channel B1
ch_gain_B2 = 1          ; gain of channel B2
ch_gain_B3 = 1          ; gain of channel B3
ch_gain_B4 = 1          ; gain of channel B4

LP_filt_A = 0           ; LP filter group A setting
                        ; 0- 30ns ( default ), 1- 100ns, 2- 300ns, 3- 1000ns
LP_filt_B = 0           ; LP filter group B setting
                        ; 0- 30ns ( default ), 1- 100ns, 2- 300ns, 3- 1000ns
invert_chan = 0         ; channels inverted , 8-bit value ,default = 0,
                        ; bit = 0 - channel not inverted, bit = 1 - channel inverted
                        ; bit0 - chan.A1... bit3 - chan.A4,
                        ; bit4 - chan.B1... bit7 - chan.B4

                        ; all channel digital(software) offsets have range -100% ... 100% and
                        ; default value 0 %
ch_digoffs_A1 = 0.0     ; digital offset of channel A1
ch_digoffs_A2 = 0.0     ; digital offset of channel A2
ch_digoffs_A3 = 0.0     ; digital offset of channel A3
ch_digoffs_A4 = 0.0     ; digital offset of channel A4
ch_digoffs_B1 = 0.0     ; digital offset of channel B1
ch_digoffs_B2 = 0.0     ; digital offset of channel B2
ch_digoffs_B3 = 0.0     ; digital offset of channel B3
ch_digoffs_B4 = 0.0     ; digital offset of channel B4

                        ; all channel digital(software) zoom have range 1 .. 64 and
                        ; default value 1
ch_digzoom_A1 = 0.0     ; digital zoom of channel A1
ch_digzoom_A2 = 0.0     ; digital zoom of channel A2
ch_digzoom_A3 = 0.0     ; digital zoom of channel A3
ch_digzoom_A4 = 0.0     ; digital zoom of channel A4
ch_digzoom_B1 = 0.0     ; digital zoom of channel B1
ch_digzoom_B2 = 0.0     ; digital zoom of channel B2
ch_digzoom_B3 = 0.0     ; digital zoom of channel B3
ch_digzoom_B4 = 0.0     ; digital zoom of channel B4

start_ptr=0             ; start point of collection( 0(default) ... 16383)
samples = 100           ; no of samples to collect( 1 ... 16383 ), default 100

accu = 0                ; accumulation on(1) / off(0) - default
no_of_accum = 1         ; no of accumulations( 1(default) ... 65535 )

[shm_module2]          ; SHM module 2 hardware parameters

```

```

active = 1                ; module active - can be used (default = 0 - not active)

[shm_module3]            ; SHM module 3 hardware parameters

active = 1                ; module active - can be used (default = 0 - not active)

[shm_module4]            ; SHM module 4 hardware parameters

active = 1                ; module active - can be used (default = 0 - not active)

```

After successful initialisation the module is locked to prevent that other application can access it. Therefore a SHM module can only be initialised if it is not in use (i.e. locked) by another application. If, for any reason, a locked module must be initialised, it can be done by using the function **SHM_set_mode** with the parameter 'force_use' = 1.

After an **SHM_init** call we recommend to call the **SHM_test_if_active** function to check whether (and which) SHM module is active. Only active module can be operated further. It is recommended (but not required) to check also the initialisation status (by **SHM_get_init_status**) of the used module. In case of a wrong initialisation the initialisation status shows the reason of the error (see shm_def.h for possible values).

If several SHM modules are present the modules are numbered in the order of their serial numbers, i.e. module 1 is the module with the lowest serial number.

Additional information about the SHM modules can be obtained by calling **SHM_get_module_info** function. The function fills SHMModInfo structure (see shm_def.h for definition).

```

-----
short CVICDECL SHM_test_if_active ( short mod_no);
-----

```

Input parameters:

mod_no module number (0 - 3)

Return value:

0 - module not active (cannot be used) , 1 - module active

Description:

The procedure returns information whether the SHM module 'mod_no' is active or not. As a result of a wrong initialisation (SHM_init function) a module can be deactivated. To find out the reason of deactivating the module, run the SHM_get_init_status function.

```
short CVICDECL SHM_get_init_status( short mod_no, short * ini_status);
```

Input parameters:

mod_no	module number (0 – 3)
*ini_status	pointer to the initialisation status

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

Description:

The procedure loads the ini_status variable with the initialisation result code set by the function SHM_init for module 'mod_no'. The possible values are shown below (see also shm_def.h):

INIT_OK	0	no error
INIT_NOT_DONE	-1	init not done
INIT_WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_CANT_OPEN_PCI_CARD	-3	cannot open PCI card
INIT_MOD_IN_USE	-4	module already in use
INIT_WINDRVR_VER	-5	incorrect WinDriver version

```
short CVICDECL SHM_get_mode(void);
```

Input parameters:

none

Return value: current mode of DLL operation

Description:

The procedure returns the current mode of the DLL operation (hardware or simulation). Possible 'mode' values are defined in the shm_def.h file:

#define SHM_HARD	0	hardware mode
#define SHM_SIMUL180	180	simulation mode of SHM-180

short CVICDECL **SHM_set_mode**(short mode, short force_use, short *in_use);

Input parameters:

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

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

Description:

The procedure is used to change the mode of the DLL operation between the ‘hardware’ mode and the ‘simulation’ mode. It is a low level procedure and not intended to normal use. It is used for software test and demonstration, and to switch the DLL to the simulation mode if hardware errors occur during the initialisation.

The table ‘in_use’ should contain entries for all 4 modules but only one can be set to 1:

- 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 one of 4 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 can be used further.

When one of the simulation modes is requested for each of 4 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 can be used further.

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

Use the function `SHM_get_init_status` and/or `SHM_get_module_info` to check which modules are correctly initialised and can be use further.

Use the function `SHM_get_mode` to check which mode is actually set. Possible ‘mode’ values are defined in the `shm_def.h` file.

short CVICDECL **SHM_get_version** (short mod_no , unsigned short * version);

Input parameters:

mod_no module number (0 - 3)
* version pointer to the result structure

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

Description:

The procedure loads the version variable with the FPGA version of SHM module 'mod_no'.

short CVICDECL **SHM_get_module_info** (short mod_no , SHMModInfo * mod_info);

Input parameters:

mod_no module number (0 - 3)
* mod_info pointer to the result structure

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

Description:

After calling the SHM_init function (see above) the internal 'SHMModInfo' 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 SHMModInfo (see shm_def.h) which has to be defined by the user. The parameters in this structure are described below.

short module_type	module type : 180- SHM-180
short bus_number	PCI bus number
short slot_number	slot number on PCI bus
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
char serial_no[18]	module serial number

```
short CVICDECL SHM_get_error_string( short error_id, char * dest_string,  
short max_length);
```

Input parameters:

error_id SHM DLL error id (0 – number of SHM errors-1) (see shm_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 the string which contains the explanation of the SHM DLL error with the id equal ‘error_id’ to ‘dest_string’. Up to ‘max_length’ characters will be copied.

Possible ‘error_id’ values are defined in the shm_def.h file.

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

Input parameters:

mod_no module number (0 - 3)
par_id parameter identification number (see shm_def.h)
*value pointer to the parameter value

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

The procedure loads ‘value’ with the actual value of the requested parameter from the DLL-internal data structures of the SHM module ‘mod_no’. The par_id values are defined in shm_def.h file as SHM_PARAMETERS_KEYWORDS.

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

Input parameters:

mod_no module number (-0 .. 3)
par_id parameter identification number
send_to_hard send value to hardware (1), or not (0)
value new parameter value

Return value:

0 no errors, <0 error code (see shm_def.h)

The procedure sets the specified hardware parameter. The value of the specified parameter is transferred to the internal data structures of the DLL functions and to the SHM module 'mod_no'.

If 'mod_no' = -1, the parameter is set on all active modules.

The new parameter value is recalculated according to the parameter limits and hardware restrictions. 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 to get their real values after recalculation.

The parameters ACTIVE and PCI_BUS_NO cannot be changed. They can be changed only by a new ini_file and a SHM_init call.

The par_id values are defined in shm_def.h file as SHM_PARAMETERS_KEYWORDS.

```
-----
short CVICDECL SHM_get_parameters( short mod_no, SHMdata * data);
-----
```

Input parameters:

mod_no	module number (0 - 3)
*data	pointer to result structure (type SHMdata)

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

Description:

After calling the SHM_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 **SHM_get_parameters** is provided. This function transfers the parameter values of the SHM module 'mod_no' from the internal structures of the DLLs into a structure of the type SHMdata (see shm_def.h). A suitable structure has to be defined by the user. The parameter values in this structure are described below.

unsigned short base_adr	base I/O address on PCI bus
short init	set to initialisation result code
short active	most of the library functions are executed only when module is active (not 0)
short trigger	external trigger condition - active low(1), active high(2)
float trig_level	trigger threshold level -1V.. +1V
float time_per_point	time of 1 point in mikrosec 0.01 .. 655.35
float delay	delay for 1st point, 10 ns .. 655.360E-6 s
float ch_offs[8]	offset of channel 1-8, -25 .. +25 mV
short HPfilter[2]	High Pass Filter setting for both groups of channels 0- off, 1- 100µs, 2- 10µs, 3- 1µs
short ch_gain[8]	gain of channel 1-8 possible values 1-8,16,24,32,40,48,56
short LPfilter[2]	Low Pass Filter setting for both groups of channels 0- 30ns, 1- 100ns, 2- 300ns, 3- 1000ns
unsigned short start_ptr	memory start pointer (0(default) ... 16383)

short accumulate	accumulate or overwrite values in memory
unsigned long samples	no of samples to collect (1 ... 16383), default 100*/
unsigned long no_of_accum	number of accumulations 1(default) ... 65535
short invert_chan	channels inverted , 8-bit value ,default = 0, bit = 0 - channel not inverted, bit = 1 - channel inverted bit0 - chan.A1... bit3 - chan.A4, bit4 - chan.B1... bit7 - chan.B4 */
short enable_meas	measurement enabled/disabled(1/0)
float ch_digoffs[8]	digital(software) offset of channel 1-8, -100 .. +100 %, default 0%
short ch_digzoom[8]	digital(software) zoom of channel 1-8, 1 .. 64, default 1

```
short CVICDECL SHM_set_parameters( short mod_no, SHMdata * data);
```

Input parameters:

mod_no	module number (0 - 3)
*data	pointer to parameters structure (type SHMdata, see shm_def.h)

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

The procedure sends all parameters from the 'SHMdata' structure to the internal DLL structures and to the control registers of the SHM module 'mod_no'.

The new parameter values are recalculated according to the parameter limits and hardware restrictions. 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 to get their true values after recalculation. The values of 'base_adr', 'init' and 'active' are not changed. They can be changed only by a new ini_file an a SHM_init call.

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

```
short CVICDECL SHM_get_eeeprom_data( short mod_no, SHM_EEP_Data *eep_data);
```

Input parameters:

mod_no	module number (0 - 3)
*eep_data	pointer to result structure

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

The structure "eep_data" is filled with the contents of the EEPROM of the SHM module 'mod_no'. The EEPROM contains the production data and adjust parameters of the module. The structure "SHM_EEP_Data" is defined in the file shm_def.h.

```
short CVICDECL SHM_write_eeprom_data(short mod_no, unsigned short write_enable,  
                                     SHM_EEP_Data *eep_data);
```

Input parameters:

mod_no	module number (0 - 3)
write_enable	write enable password
*eep_data	pointer to result structure

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

The function is used to write data to the EEPROM of an SHM module 'mod_no' by the manufacturer. To prevent corruption of the data by not allowed access the function writes the EEPROM only if the 'write_enable' password is correct.

```
short CVICDECL SHM_get_adjust_parameters (short mod_no,  
                                           SHM_Adjust_Para * adjpara);
```

Input parameters:

mod_no	module number (0 - 3)
* adjpara	pointer to result structure

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

The structure 'adjpara' is filled with adjust parameters that are currently in use. The parameters can either be previously loaded from the EEPROM by SHM_init or SHM_get_eeprom_data or - not recommended - set by SHM_set_adust_parameters.

The structure "SHM_Adjust_Para" is defined in the file shm_def.h.

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

```
short CVICDECL SHM_set_adjust_parameters (short mod_no,  
                                           SHM_Adjust_Para * adjpara);
```

Input parameters:

mod_no	module number (0 - 3)
* adjpara	pointer to result structure

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

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 SHM_init. The next call to SHM_init replaces the adjust parameters by the values from the EEPROM. We strongly discourage to use modified adjust parameters, because the module function can be seriously corrupted.

The structure "SHM_Adjust_Para" is defined in the file shm_def.h.

short CVICDECL **SHM_test_if_busy**(short * busy);

Input parameters:

*busy pointer to result value

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

SHM_test_if_busy sets a busy variable according to the current state of the measurement. The function is used to control the measurement loop after starting the measurement. The current state of all active modules is taken into account. Possible values of busy are listed below.

- 0 - all active SHM modules finished the measurement,
- 1 - the measurement is still running at least in one SHM module, no modules are waiting for the trigger
- 2 - at least one module is waiting for the trigger

short CVICDECL **SHM_read_status**(short mod_no, unsigned short * status);

Input parameters:

mod_no module number (0 - 3)

*status pointer to result value

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

The **SHM_read_status** function returns the current status of SHM module 'mod_no'. The most important status bits delivered by the function are listed below (see also shm_def.h).

ARMED	0x1	module is armed
MEASURE	0x10	module is triggered

The function is a low level procedure which is normally used only to get additional information about the SHM module state. To control the measurement, the SHM_test_if_busy function is recommended.

short CVICDECL **SHM_start_measure**(void);

Input parameters: none

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

The procedure is used to start the measurement.

Before a measurement is started by **SHM_start_measure**

- the parameters on all active modules must be set (SHM_init or SHM_set_parameter(s))
- the measurement must be enabled in all requested modules (parameter ENABLE_MEAS must be set by SHM_set_parameter),
- START_PTR and SAMPLES must be set to define the number of points to be measured.

The measurement starts with the memory pointer START_PTR and stops after collecting SAMPLES points.

short CVICDECL **SHM_stop_measure**(void);

Input parameters: none

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

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

short CVICDECL **SHM_read_data**(short mod_no, short channel, unsigned short from, unsigned short to, long * buf, unsigned short * points_read, unsigned short * lastpoint_accu);

Input parameters:

mod_no	module number (0 - 3)
channel	channel number (0 - 7)
from	1st address to read (0 to 16383)
to	last address to read ('from' to 16383)
* buf	pointer to a data buffer to be filled with channel data
* points_read	pointer to a variable which will be set with number of read points
* lastpoint_accu	pointer to a variable which will be set with number of read points

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

The procedure is used to read measurement results from the channel number 'channel' on SHM module 'mod_no'.

The number of points read (32 bit 2complement ADC values) depends on the state of the measurement.

If the measurement is already finished, or not started yet, or when ‘from’ to ‘to’ is not in the measured range (START_PTR to START_PTR + SAMPLES) the procedure reads the SHM memory from the address ‘from’ to the address ‘to’ . The procedure sets ‘lastpoint_accu’ variable to the number of accumulations performed for the last point(which is equal to the parameter NO_OF_ACCUM) and sets the points_read variable to the value ‘ to - from +1’.

If the measurement is still running and from’ to ‘to’ is in the measured range (START_PTR to START_PTR + SAMPLES), the procedure reads the SHM memory from the address ‘from’ up to the current memory pointer. Then it reads the last (not finished) point value from the memory and sets ‘lastpoint_accu’ variable to the current number of accumulations that have been already completed. If this value is equal to the parameter NO_OF_ACCUM, it means that the point is already finished.

Finally, the procedure sets the ‘points_read’ variable to a value equal to the number of points collected from the start of the measurement.

Please make sure that the buffer ‘buf’ is allocated with enough memory for the required number of points (to - from +1).

short CVICDECL **SHM_fill_memory**(short mod_no, short channel, unsigned short from, unsigned short to, long fill_value);

Input parameters:

mod_no	module number (0 - 3)
channel	channel number 0 - 7 , all channels: -1
from	1st address to fill (0 to 16383)
to	last address to fill (‘from’ to 16383)
fill_value	value written into the SHM memory

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

The procedure is used to fill a specified part of the memory of the SHM module ‘mod_no’ with the value ‘fill_value’.