

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

pmmdl32.doc



**PMM-328**  
**32 Bit Dynamic Link Libraries**  
**User Manual**

Version 1.0, December 98

## Introduction

The PMM 32 bits Dynamic Link Library contains all functions to control the PMM-328 and PMA-328 modules. The functions work under Windows 95 and Windows NT. The program which calls the DLLs must be compiled with the compiler option 'Structure Alignment' set to '1 Byte'.

The distribution disks contain the following files:

PMMDLL32.DLL	dynamic link library main file
PMMDLL32.LIB	import library file for Microsoft Visual C/C++, Borland C/C++, Watcom C/C++ and Symantec C/C++ compilers
PMM_DEF.H	Include file containing Types definitions, Functions Prototypes and Pre-processor statements
PMM328.INI	PMM DLL initialisation file
PMMDLL32.DOC	This description file
USE_PMM.XXX	Set of files to compile and run a simple example of using PMM DLL functions. The source file of the example is the file use_pmm.c. The example was originally prepared under Borland C/C++ v.4.52. For use in other compilers choose the correct import library file to link.

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

## PMM-DLL Functions list

The following functions are implemented in the PMM-DLL:

### Initialisation functions:

- PMM\_init
- PMM\_test\_if\_active
- PMM\_get\_init\_status
- PMM\_get\_sync\_adr
- PMM\_set\_sync\_adr
- PMM\_get\_mode
- PMM\_set\_mode
- PMM\_get\_version

### Setup functions:

- PMM\_get\_parameter
- PMM\_set\_parameter
- PMM\_get\_parameters
- PMM\_set\_parameters
- PMM\_get\_eeprom\_data
- PMM\_write\_eeprom\_data
- PMM\_get\_adjust\_parameters
- PMM\_set\_adjust\_parameters

### Status functions:

PMM\_test\_if\_busy  
PMM\_read\_status

#### Measurement control functions:

PMM\_start\_measure  
PMM\_stop\_measure

#### PMM transfer functions:

PMM\_fill\_memory  
PMM\_read\_data  
PMA\_get\_ADC\_value (for PMA modules only)

#### Test functions:

PMM\_test\_id  
PMM\_test\_counter  
PMM\_get\_test\_error\_string

## Application Guide

### Initialisation of the PMM 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 PMM module. This is accomplished by the function **PMM\_init**.

The PMM DLL Functions can control up to four PMM modules. All modules can be synchronously started and stopped.

The **PMM\_init** function

- reads the parameter values from a specified initialisation file
- checks the sync\_adr and base I/O addresses for all active modules to avoid hardware conflicts
- checks and recalculates the parameters depending on the hardware restrictions and the adjust parameters from the EEPROM on each active PMM module
- sends the parameter values to the PMM control registers on each active PMM module
- performs a hardware test of each active PMM module

The initialisation file is an ASCII file with a structure shown in the table below. Each module has its own section in the initialisation file ([pmm\_module0..3]). Only modules which have an entry 'active = 1' are initialised. We recommend either to use the file pmm328.ini or to start with pmm328.ini and to introduce the desired changes.

```
; PMM328 initialisation file  
; PMM parameters have to be included in .ini file only when parameter  
; value is different from default.  
; module section (pmm_module0-3) is required for each existing PMM module
```

```
[pmm_base]  
simulation = 0 ; 0 - hardware mode(default) ,  
; >0 - simulation mode (see pmm_def.h for possible values)  
base_sync_adr= 0x398 ; sync_adr will be set on all active modules to start/stop  
; collection of photons (0 ... 0x3FC,default 0x398)
```

```
[pmm_module1] ; PMM module 1 hardware parameters
```

```

base_adr = 0x380           ;base I/O address (0 ... 0x3FC,default 0x380)
active = 1                 ;module active - can be used (default = 0 - not active)
enable_meas = 1           ; enable/disable(1/0) measurement , default = enable
trigger = 0                ; external trigger condition
                           ; none(0)(default),active low(1),active high(2)
gate_level= -0.2          ;discriminator level for gate signal
                           ; ( -0.2V ... +0.2V , default -0.2)
trig_level= 1.0           ; trigger level for ( -1.0V ... +1.0V , default 1.0)

                           ; all input thresholds have range -0.2 ... +0.2V and
                           ; default value -0.1V
inp_threshold_1= -0.1     ;input threshold level of channel 1
inp_threshold_2= -0.1     ;input threshold level of channel 2
inp_threshold_3= -0.1     ;input threshold level of channel 3
inp_threshold_4= -0.1     ;input threshold level of channel 4
inp_threshold_5= -0.1     ;input threshold level of channel 5
inp_threshold_6= -0.1     ;input threshold level of channel 6
inp_threshold_7= -0.1     ;input threshold level of channel 7
inp_threshold_8= -0.1     ;input threshold level of channel 8

collect_time= 1.          ;collection time in seconds (default 1.0 sec)
                           ; ( 0.25 mikrosec ... 100000 sec )
start_ptr=0               ; start point of collection( 0(default) ... 32767)
end_ptr=0                 ; end point of collection( start_ptr(default) ... 32767)

[pmm_module1]             ; PMM module 1 hardware parameters

base_adr= 0x280           ;base I/O address (0 ... 0x3FC)
active= 0                 ;module not active - cannot be used

[pmm_module2]             ; PMM module 2 hardware parameters

base_adr= 0x2a0           ;base I/O address (0 ... 0x3FC)
active= 0                 ;module not active - cannot be used

[pmm_module3]             ; PMM module 3 hardware parameters

base_adr= 0x2c0           ;base I/O address (0 ... 0x3FC)
active = 0                ;module not active - cannot be used

```

After a **PMM\_init** call we recommend to check which PMM modules are active by calling **PMM\_test\_if\_active** function. At least one module must be active, and only active modules can be operated further. It is recommended (but not required) to check also the initialisation status (**PMM\_get\_init\_status**) of each used module. In case of a wrong initialisation the initialisation status shows the reason of the error (see `pmm_def.h` for possible values ). In case of hardware test errors (values `INIT_WRONG_COUNTER` or `INIT_WRONG_DACs`) the function **PMM\_get\_test\_error\_string** returns additional information about the error.

After calling the **PMM\_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 **PMM\_get\_parameters** is provided. This function transfers the parameter values from the internal structures of the DLLs into a structure of the type `PMMdata` (see `pmm_def.h`) which has to be defined by the user. The parameter values in this structure are described below.

short base_adr	I/O base address
short init	set to initialisation result code
short active	most of the library functions are executed only when module is active ( not 0 )
short test_eep	test EEPROM checksum on start-up or not
short meas_mode	measurement mode
short enable_meas	measurement enabled/disabled(1/0)
short trigger	external trigger condition - none(0),active low(1),active high(2)
float gate_level	gate discriminator level
float inp_threshold[8]	input threshold level of channels 1 - 8
float collect_time	collection time interval
unsigned short start_ptr	memory start pointer
unsigned short end_ptr	memory end pointer
unsigned short count[8];	last value read from photon counter 1-8

To send the complete parameter set back to the DLLs and to the PMM module (e.g. after changing parameter values) the function **PMM\_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 **PMM\_set\_parameters** by **PMM\_get\_parameters**.

Single parameter values can be transferred to or from the DLL and module level by the functions **PMM\_set\_parameter** and **PMM\_get\_parameter**. To identify the desired parameter, the parameter identification par\_id is used. The parameter identification keywords are defined in pmm\_def.h.

## Memory Configuration

The memory is organised in two banks which contain the data for 8 channels. The channel data are 16 bit counter values. The measurement memory length 32768 words for each counter channel.

## Memory Read/Write Functions

Reading the memory of the PMM module is accomplished by the function **PMM\_read\_data**. To fill the memory with a constant value (or to clear the memory) the function **PMM\_fill\_memory** is available.

## Standard Measurements

The most important measurement functions are listed below.

The **PMM\_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 PMM modules have finished the measurement,
- 1 - the measurement is still running (at least) in one PMM module, no modules are waiting for an external trigger
- 2 - at least one module is waiting for the external trigger

The **PMM\_read\_status** function returns the current status of a particular PMM module. The most important status bits delivered by the function are listed below (see also PMM\_DEF.H).

ARMED	0x1	module is armed
MEASURE	0x2	module collects data ( Armed and Triggered )
CT_EXP	0x40	collection timer expired (=0)
CT_OV	0x80	collection timer overflow
OVFL_1	0x100	overflow on counter 1
OVFL_2	0x200	overflow on counter 2
OVFL_3	0x400	overflow on counter 3
OVFL_4	0x800	overflow on counter 4
OVFL_5	0x1000	overflow on counter 5
OVFL_6	0x2000	overflow on counter 6
OVFL_7	0x4000	overflow on counter 7
OVFL_8	0x8000	overflow on counter 8

**PMM\_start\_measure** starts the measurement in all active PMM modules with the parameters set before by the PMM\_init, PMM\_set\_parameters or PMM\_set\_parameter functions. The measurement starts to collect photons in subsequent points in PMM memory from the address START\_PTR. The measurement stops when the memory pointer reaches END\_PTR, i.e. after recording (End\_ptr -Start\_ptr + 1) points

**PMM\_stop\_measure** is used to stop the measurement by a software command.

In the figures below some block diagrams of some simple measurement routines are given.

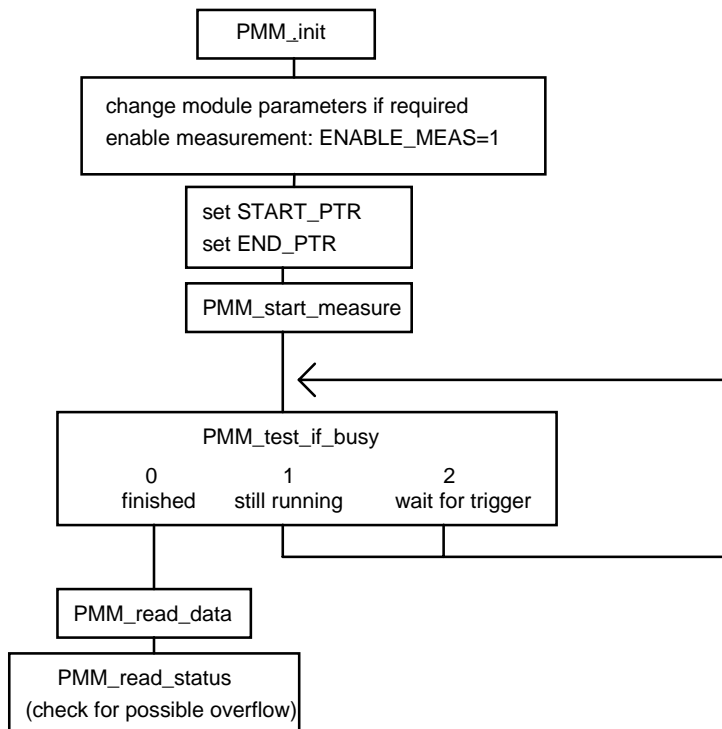


Fig1: Multiscaler measurement without accumulation

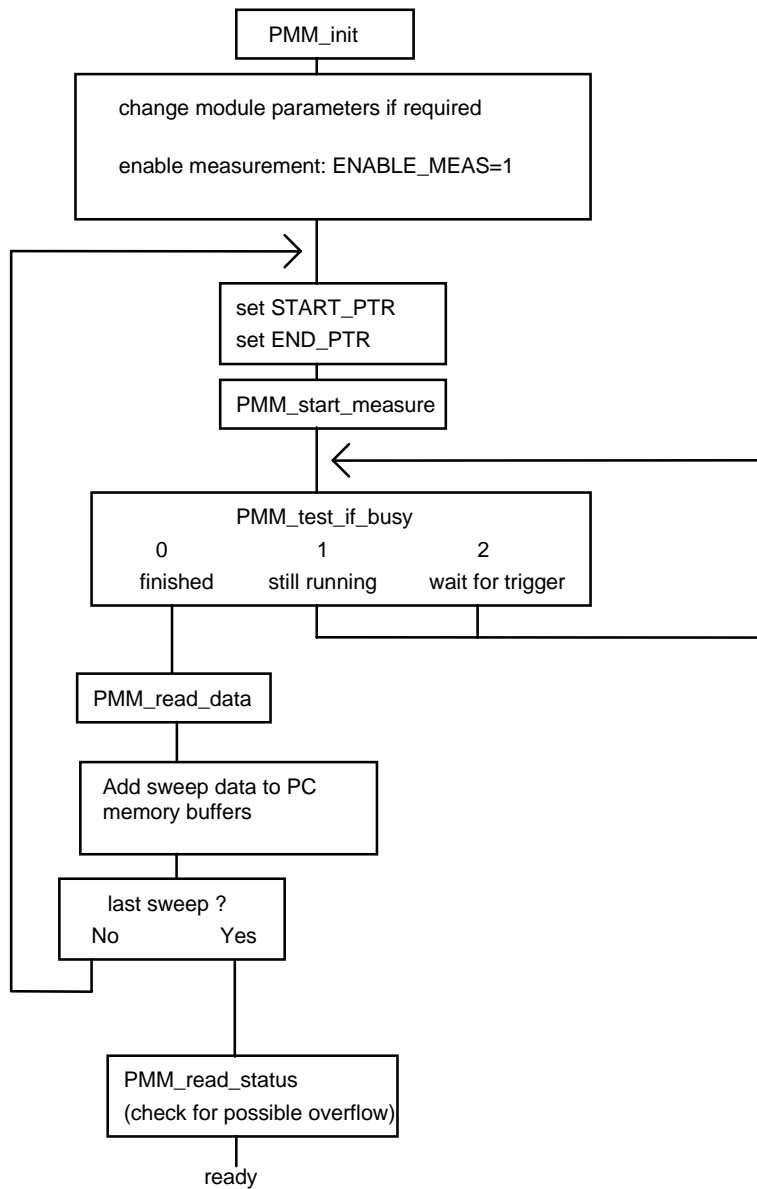


Fig. 2: Multiscaler Measurement with software accumulation

### Error Handling

Each PMM DLL function returns an error status. Return values  $\geq 0$  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 `pmm_def.h` file. We recommend to check the return value after each function call.

## Description of the PMM DLL Functions

---

```
short CVICDECL PMM_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 PMM module. This is accomplished by the function **PMM\_init**. The function

- reads the parameter values from the specified file ini\_file
- checks sync\_adr and base I/O addresses for all active modules to avoid hardware conflicts
- checks and recalculates the parameters depending on hardware restrictions and adjust parameters from the EEPROM in each active PMM module
- sends the parameter values to the PMM control registers of each active PMM module
- performs a hardware test of each active PMM module

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file pmm328.ini or to start with pmm328.ini and introduce the desired changes.

```
; PMM328 initialisation file  
; PMM parameters have to be included in .ini file only when parameter  
; value is different from default.  
; module section (pmm_module0-3) is required for each existing PMM module
```

```
[pmm_base]  
simulation = 0 ; 0 - hardware mode(default) ,  
; >0 - simulation mode (see pmm_def.h for possible values)  
base_sync_adr= 0x398 ; sync_adr will be set on all active modules to start/stop  
; collection of photons (0 ... 0x3FC,default 0x398)
```

```
[pmm_module1] ; PMM module 1 hardware parameters  
base_adr = 0x380 ;base I/O address (0 ... 0x3FC,default 0x380)  
active = 1 ;module active - can be used (default = 0 - not active)  
enable_meas= 1 ; enable/disable(1/0) measurement , default = enable  
trigger = 0 ; external trigger condition  
; none(0)(default),active low(1),active high(2)  
gate_level= -0.2 ;discriminator level for gate signal  
; (-0.2V ... +0.2V , default -0.2)  
trig_level= 1.0 ; trigger level for (-1.0V ... +1.0V , default 1.0)
```

; all input thresholds have range -0.2 ... +0.2V and

```

;default value -0.1V
inp_threshold_1= -0.1 ;input threshold level of channel 1
inp_threshold_2= -0.1 ;input threshold level of channel 2
inp_threshold_3= -0.1 ;input threshold level of channel 3
inp_threshold_4= -0.1 ;input threshold level of channel 4
inp_threshold_5= -0.1 ;input threshold level of channel 5
inp_threshold_6= -0.1 ;input threshold level of channel 6
inp_threshold_7= -0.1 ;input threshold level of channel 7
inp_threshold_8= -0.1 ;input threshold level of channel 8

collect_time= 1. ;collection time in seconds (default 1.0 sec)
; ( 0.25 mikrosek ... 100000 sec )
start_ptr=0 ;start point of collection( 0(default) ... 32767)
end_ptr=0 ;end point of collection( start_ptr(default) ... 32767)

[pmm_module1] ;PMM module 1 hardware parameters

base_adr= 0x280 ;base I/O address (0 ... 0x3FC)
active = 0 ;module not active - cannot be used

[pmm_module2] ;PMM module 2 hardware parameters

base_adr= 0x2a0 ;base I/O address (0 ... 0x3FC)
active = 0 ;module not active - cannot be used

[pmm_module3] ;PMM module 3 hardware parameters

base_adr= 0x2c0 ;base I/O address (0 ... 0x3FC)
active = 0 ;module not active - cannot be used

```

After a **PMM\_init** call we recommend to check which PMM modules are active by calling **PMM\_test\_if\_active** function (minimum one module must be active and only active modules can be operated further). It is reasonable also to check the initialisation status (**PMM\_get\_init\_status**) of each used module. The initialisation status can show the reason of a wrong initialisation (see `pmm_def.h` for possible values ). In case of hardware test errors (values `INIT_WRONG_COUNTER` or `INIT_WRONG_DACs`) the function **PMM\_get\_test\_error\_string** delivers additional information.

```

-----
short CVICDECL PMM_test_if_active (short mod_no);
-----

```

Input parameters:

mod\_no module number (0 - 3)

Return value:

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

Description:

The procedure returns information whether the module specified by 'mod\_no' is active or not. A module is set active only if there is the entry 'active = 1' in the respective module section in the ini\_file. As a result of a wrong initialisation (PMM\_init function) a module can be deactivated. In this case a the PMM\_get\_init\_status function can explain the reason of deactivating the module.

-----  
short CVICDECL **PMM\_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 pmm\_def.h)

Description:

The procedure loads the ini\_status variable with the initialisation result code set by the function PMM\_init for module 'mod\_no'. The possible values are shown below (see also pmm\_def.h):

INIT_OK	0	no error
INIT_NOT_DONE	-1	initialisation not done
INIT_WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_WRONG_MOD_ID	-3	wrong module identification code
INIT_WRONG_BASE_ADR	-4	not unique base address
INIT_WRONG_SYNC_ADR	-5	sync address equal to base address
INIT_WRONG_COUNTER	-6	counter test failed
INIT_WRONG_DACs	-7	DAC's test failed

In case of hardware test errors (values INIT\_WRONG\_COUNTER or INIT\_WRONG\_DACs) the function **PMM\_get\_test\_error\_string** gives additional information on the test error.

-----  
short CVICDECL **PMM\_get\_sync\_adr**(short \* adr);  
-----

Input parameters:

\*adr            pointer to the sync address

Return value:     0 no errors, <0     error code (see pmm\_def.h)

Description:

The procedure loads the 'adr' variable with the actual sync address value.

The Sync address is common for all active modules and is used to synchronously start and stop all modules. The procedure returns an error, if the sync address has not the same value for all modules.

---

```
short CVICDECL PMM_set_sync_adr(short adr);
```

---

Input parameters:

mod_no	module number (0 - 3)
adr	new value of sync address

Return value:            0 no errors, <0        error code (see pmm\_def.h)

Description:

The procedure sets the sync address value to 'adr'.

The Sync address is common for all active modules and is used to synchronously start/stop all modules. The procedure checks for conflicts with I/O addresses used by the active PMM modules and, - if there are no conflicts- changes the sync address.

The procedure should be used only if the actual sync address (set during the initialisation to a value taken from ini\_file) causes conflicts with other devices.

---

```
short CVICDECL PMM_get_mode(void);
```

---

Input parameters:

none

Return value:            current mode of DLL operation

Description:

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

```
#define PMM_HARD            0            /* hardware mode */  
#define PMM_SIMUL328       30        /* simulation mode of PMM328 */  
#define PMA_SIMUL328       40        /* simulation mode of PMA328 */
```

---

```
short CVICDECL PMM_set_mode(short mode);
```

---

Input parameters:

mode: mode of DLL operation

Return value: 0 no errors, <0 error code (see pmm\_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 to switch the DLL to the simulation mode if hardware errors occur during the initialisation.

Use the function `PMM_get_mode` to check which mode is actually set. Possible 'mode' values are defined in the `pmm_def.h` file.

---

```
short CVICDECL PMM_get_version(short mod_no , short * version);
```

---

Input parameters:

mod\_no            module number (0 - 3)  
\*version           pointer to the version variable

Return value: 0 no errors, <0 error code (see pmm\_def.h)

Description:

The procedure loads the 'version' variable with the FPGA version of the module specified by `mod_no`. This is low a level procedure, not needed normally.

---

```
short CVICDECL PMM_get_parameters(short mod_no, PMMdata * data);
```

---

Input parameters:

mod\_no            module number (0 - 3)  
\*data              pointer to result structure (type PMMdata)

Return value: 0 no errors, <0 error code (see pmm\_def.h)

Description:

After calling the `PMM_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 **`PMM_get_parameters`** is provided. This function transfers the parameter values of the module 'mod\_no' from the internal structures

of the DLL into a structure of the type PMMdata (see pmm\_def.h) which has to be defined by the user. The parameter values in this structure are described below.

short base_adr	I/O base address
short init	set to initialisation result code
short active	most of the library functions are executed only when module is active ( not 0 )
short test_eep	test EEPROM checksum on start-up or not
short meas_mode	measurement mode
short enable_meas	measurement enabled/disabled(1/0)
short trigger	external trigger condition - none(0),active low(1),active high(2)
float gate_level	gate discriminator level
float inp_threshold[8]	input threshold level of channels 1 - 8
float collect_time	collection time interval
unsigned short start_ptr	memory start pointer
unsigned short end_ptr	memory end pointer
unsigned short count[8];	last value read from photon counter 1-8

---

```
short CVICDECL PMM_set_parameters(short mod_no, PMMdata * data);
```

---

Input parameters:

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

Return value: 0 no errors, <0 error code (see pmm\_def.h)

Description:

The procedure sends all parameters from the 'PMMdata' structure to the internal DLL structures and to the control registers of the PMM module 'mod\_no'.

The new parameter values are recalculated according to the parameter limits and hardware restrictions (e.g. DAC resolution). 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 values of 'base\_adr', 'init' and 'active' are not changed. They can be changed only by a new ini\_file an a PMM\_init call.

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 **PMM\_get\_parameter**(short mod\_no, short par\_id, float \* value);  
-----

Input parameters:

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

Return value:           0 no errors, <0       error code (see pmm\_def.h)

The procedure loads 'value' with the actual value of the requested parameter from the DLL-internal data structures of the module 'mod\_no'. The par\_id values are defined in pmm\_def.h file as PMM\_PARAMETERS\_KEYWORDS.

-----  
short CVICDECL **PMM\_set\_parameter**(short mod\_no, short par\_id, float value);  
-----

Input parameters:

mod_no	module number (0 - 3)
par_id	parameter identification number
value	new parameter value

Return value:

0 no errors, <0 error code (see pmm\_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 PMM module 'mod\_no'. The new parameter value is recalculated according to the parameter limits and hardware restrictions (e.g. DAC resolution). 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 BASE\_ADR and ACTIVE cannot be changed. They can be changed only by a new ini\_file and a PMM\_init call.

The par\_id values are defined in pmm\_def.h file as PMM\_PARAMETERS\_KEYWORDS.



The structure 'adjpara' is filled with adjust parameters that are currently in use. The parameters can either be previously loaded from the EEPROM by PMM\_init or PMM\_get\_eeprom\_data or - not recommended - set by PMM\_set\_adust\_parameters.

The structure "PMM\_Adjust\_Para" is defined in the file pmm\_def.h.

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

```
-----  
short CVICDECL PMM_set_adjust_parameters (short mod_no,  
                                           PMM_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 pmm\_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 PMM\_init. The next call to PMM\_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 "PMM\_Adjust\_Para" is defined in the file pmm\_def.h.

```
-----  
short CVICDECL PMM_test_if_busy(short * busy);  
-----
```

Input parameters:

- \*busy            pointer to result value

Return value:        0 no errors, <0 error code (see pmm\_def.h)

PMM\_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 PMM modules finished the measurement,
- 1 - the measurement is still running at least in one PMM module, no modules are waiting for the trigger
- 2 - at least one module is waiting for the trigger

-----  
short CVICDECL **PMM\_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 **PMM\_read\_status** function returns the current status of PMM module 'mod\_no'. The most important status bits delivered by the function are listed below (see also PMM\_DEF.H).

ARMED	0x1	module is armed
MEASURE	0x2	module collects data ( Armed and Triggered )
CT_EXP	0x40	collection timer expired (=0)
CT_OV	0x80	collection timer overflow
OVFL_1	0x100	overflow in counter 1
OVFL_2	0x200	overflow in counter 2
OVFL_3	0x400	overflow in counter 3
OVFL_4	0x800	overflow in counter 4
OVFL_5	0x1000	overflow in counter 5
OVFL_6	0x2000	overflow in counter 6
OVFL_7	0x4000	overflow in counter 7
OVFL_8	0x8000	overflow in counter 8

The function is a low level procedure which is normally used only to test whether an overflow occurred during the measurement and to get additional information about the PMM module state. To control the measurement, the **PMM\_test\_if\_busy** function is recommended.

-----  
short CVICDECL **PMM\_start\_measure**(void);  
-----

Input parameters:       none

Return value:           0 no errors, <0       error code (see pmm\_def.h)

The procedure is used to start the measurement.

Before a measurement is started by **PMM\_start\_measure**

- the parameters on all active modules must be set (**PMM\_init** or **PMM\_set\_parameter(s)** )
- the measurement must be enabled in all requested modules (parameter **ENABLE\_MEAS** must be set by **PMM\_set\_parameter**),
- **START\_PTR** and **END\_PTR** must be set to define the number of points to be measured.

The measurement stops the when the memory pointer reaches **END\_PTR**, i.e. after collecting **End\_ptr - Start\_ptr + 1** points.

-----  
short CVICDECL **PMM\_stop\_measure**(void);  
-----

Input parameters: none

Return value: 0 no errors, <0 error code (see pmm\_def.h)

**PMM\_stop\_measure** is used to stop the measurement by a software command.

-----  
short CVICDECL **PMM\_fill\_memory**(short mod\_no, short channel, unsigned short from,  
unsigned short to, unsigned 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 32767 )
to	last address to fill ( 'from' to 32767)
fill_value	value written into the PMM memory

Return value: 0 no errors, <0 error code (see pmm\_def.h)

The procedure is used to fill a specified part of the memory of the PMM module 'mod\_no' with the value 'fill\_value'.

-----  
short CVICDECL **PMM\_read\_data**(short mod\_no, short channel, unsigned short from, unsigned short to, unsigned short \* buf, unsigned short \* points\_read);  
-----

Input parameters:

mod_no	module number (0 - 3)
channel	channel number (0 - 7)
from	1st address to read (0 to 32767)
to	last address to read ( 'from' to 32767)
* 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

Return value: 0 no errors, <0 error code (see pmm\_def.h)

The procedure is used to read measurement results from the channel number 'channel' on PMM module 'mod\_no'.

The number of points read (16 bit counter values) depends on the state of the measurement.

If the measurement is already finished (or not started yet), the procedure reads the PMM memory from the address 'from' to the address 'to' and sets the points\_read variable to the value 'to - from +1'.

If the measurement is still running, the procedure reads the PMM memory from the address 'from' up to the current memory pointer. Then it reads the last (not finished) point value directly from the counter. 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 PMA_get_ADC_value (short mod_no, short chan_no,  
                                unsigned short * value);  
-----
```

Input parameters:

mod_no	module number (0 - 3)
chan_no	ADC channel number (0 - 31)
* value	pointer to a variable which to be set with ADC value

Return value:            0 no errors, <0        error code (see pmm\_def.h)

The procedure is intended for use with PMA modules only (a version of the PMM-328 with an additional 32 channel multiplexed ADC). It sets the input multiplexer to the desired input channel (chan\_no, 0..31), waits until the multiplexer output has settled, starts the AD conversion and reads the result from the ADC. The range of the result is -8192 to +8191 for all input voltage ranges.

The procedure fills the 'value' variable with the value read from the ADC connected to the input channel 'chan\_no' on the PMA module 'mod\_no'.

```
-----  
short CVICDECL PMM_test_id(short mod_no) ;  
-----
```

Input parameters:

mod_no:	module number (0 - 3)
---------	-----------------------

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

The procedure is used to check the identification code of PMM module 'mod\_no'. It is a low level procedure that is called also during the initialisation in PMM\_init. The procedure returns a module type value, if the id is correct. Possible module type values are defined in the pmm\_def.h file.

-----  
short CVICDECL **PMM\_test\_counter**(short mod\_no, short \*active);  
-----

Input parameters:

mod\_no            module number (0 - 3)  
\* active           pointer to gate polarity variable

Return value:        0 no errors, <0 error code (see pmm\_def.h)

The procedure performs a trigger test and counter test in the PMM module ‘mod\_no’ and determines the state of the gate polarity jumpers. Possible values of the ‘active’ variable after the test is listed below:

- 1 - hardware error detected, gate polarity unknown
- 0 - test OK, gate polarity active high,
- 1 - test OK, gate polarity active low,
- 2 - hardware error detected. Probably jumper missing, check the module and insert the missing jumper

In case of a return value < 0 the function **PMM\_get\_test\_error\_string** can get additional information on test error. The error string is prepared during the test execution.

**Important:** The input signals must be disconnected from the module inputs during the test.

-----  
short CVICDECL **PMM\_get\_test\_error\_string**(char \*error\_string);  
-----

Input parameters:

error\_string        pointer to error message string

Return value:        <0 last error code (see pmm\_def.h)

The procedure fills ‘error\_string’ with the internal DLL string generated during the last execution of the **PMM\_test\_counter** or **PMM\_init** function. ‘Error string’ contains detailed information on a counter test error or a DAC test error (during PMM\_init).

After a call to **PMM\_get\_test\_error\_string** DLL’s internal error string is empty.

=====

