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

DCC

32 Bit Dynamic Link Libraries

User Manual

Version 1.0 August 2002

Introduction

The DCC 32 bits Dynamic Link Library contains all functions to control the DCC 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:

DCC.DLL	dynamic link library main file
DCC.LIB	import library file for Microsoft Visual C/C++, Borland C/C++, Watcom C/C++ and Symantec C/C++ compilers
DCC_DEF.H	Include file containing types definitions, functions prototypes and pre-processor statements
DCC100.INI	DCC DLL initialisation file
DCC_DLL.DOC	This description file
USE_DCC.C	A simple example of using DCC 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.

DCC-DLL Functions list

The following functions are implemented in the DCC-DLL:

Initialisation functions:

DCC_init
DCC_test_if_active
DCC_get_init_status
DCC_get_mode
DCC_set_mode
DCC_get_module_info
DCC_get_error_string

Setup functions:

DCC_get_parameter
DCC_set_parameter
DCC_get_parameters
DCC_set_parameters
DCC_get_eeprom_data
DCC_write_eeprom_data
DCC_get_gain_HV_limit
DCC_set_gain_HV_limit

Status functions:

DCC_enable_outputs
DCC_clear_overload
DCC_get_overload_state
DCC_get_curr_lmt_state

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 'DCC', for example, DCCstd_ get_parameter is the _stdcall version of DCC_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 DCC Measurement Parameters

Before the DCC 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 DCC module. This is accomplished by the function **DCC_init**.

The DCC DLL Functions are able to control up to eight DCC modules on or several PCI bus(es).

The **DCC_init** function

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

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file DCC100.INI or to start with DCC100.INI and to introduce the desired changes.

```
;    DCC100 initialisation file
;    DCC parameters have to be included in .ini file only when parameter
;    value is different from default.
;    module section (dcc_module1-8) is required for each existing DCC module

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

[dcc_module1]                   ; DCC module 1 hardware parameters

active  = 1                     ; module active - can be used  (default = 0 - not active)
c1_p5V  = 0                     ; Connector 1 +5V On (1)/ Off (0), default = 0 (Off)
c1_m5V  = 0                     ; Connector 1 -5V On (1)/ Off (0), default = 0 (Off)
c1_p12V = 0                     ; Connector 1 +12V On (1)/ Off (0), default = 0 (Off)
c1_gain_HV = 0.0                ; Connector 1 Gain/HV : 0 - c1_gain_HV_limit % ( default 0%)
                                ;  c1_gain_HV_limit ( 0 - 100(default) % ) is stored in module EEPROM
c2_p5V  = 0                     ; Connector 2 +5V On (1)/ Off (0), default = 0 (Off)
c2_m5V  = 0                     ; Connector 2 -5V On (1)/ Off (0), default = 0 (Off)
c2_p12V = 0                     ; Connector 2 +12V On (1)/ Off (0), default = 0 (Off)
c2_digout = 0x0                 ; Connector 2 Digital Outputs State, 0 - ff(hex) , default 0
                                ;     each bit of the value represents one output
c3_p5V  = 0                     ; Connector 3 +5V On (1)/ Off (0), default = 0 (Off)
c3_m5V  = 0                     ; Connector 3 -5V On (1)/ Off (0), default = 0 (Off)
c3_p12V = 0                     ; Connector 3 +12V On (1)/ Off (0), default = 0 (Off)
c3_cooling = 0                  ; Connector 3 Cooler On (1) / Off(0), default 0 (Off)
c3_coolVolt = 0.0               ; Connector 3 Cooler Voltage 0 - 5V , default 0 V
c3_coolCurr = 0.0               ; Connector 3 Cooler Current Limit 0 - 2 Amperes , default 0
```

c3_gain_HV = 0.0	; Connector 3 Gain/HV : 0 - c3_gain_HV_limit % (default 0%) ; c3_gain_HV_limit (0 - 100(default) %) is stored in module EEPROM
[dcc_module2]	; DCC module 2 hardware parameters
active = 1	; module active - can be used (default = 0 - not active)
[dcc_module3]	; DCC module 3 hardware parameters
active = 1	; module active - can be used (default = 0 - not active)
[dcc_module4]	; DCC 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 DCC 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 **DCC_set_mode** with the parameter 'force_use' = 1.

After an **DCC_init** call we recommend to call the **DCC_test_if_active** function to check whether (and which) DCC module is active. Only active modules can be operated further. It is recommended (but not required) to check also the initialisation status (by **DCC_get_init_status**) of the used module. If the initialisation was not successful for any reason the initialisation status shows the error (see dcc_def.h for possible values).

If several DCC 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 DCC modules can be obtained by calling **DCC_get_module_info** function. The function fills the DCCModInfo structure which is described below:

short module_type	module type : 100- DCC-100
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[12]	module serial number

After calling the **DCC_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. For safety reasons all outputs are disabled.

To enable the outputs of the **DCC_enable_outputs** function must be called.

To give the user access to the parameters, the function **DCC_get_parameters** is provided. This function transfers the parameter values from the internal structures of the DLLs into a structure of the type 'DCCdata' (see dcc_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 active	most of the library functions are executed only when module is active (not 0)
short c1_p5V	Connector 1 +5V On/Off
short c1_m5V	Connector 1 -5V On/Off
float c1_gain_HV	Connector 1 Gain/HV [%]
short c1_p12V	Connector 1 +12V On/Off
short c2_p5V	Connector 2 +5V On/Off
short c2_m5V	Connector 2 -5V On/Off
short c2_p12V	Connector 2 +12V On/Off
short c2_digout	Connector 2 Digital Outputs State, 0 - ff(hex)
short c3_p5V	Connector 3 +5V On/Off
short c3_m5V	Connector 3 -5V On/Off
short c3_p12V	Connector 3 +12V On/Off
float c3_coolVolt	Connector 3 Cooler Voltage (0 - 5V)
float c3_coolCurr	Connector 3 Cooler Current Limit (0 - 2A)
float c3_gain_HV	Connector 3 Gain/HV [%]
short c3_cooling	Connector 3 Cooler On/Off

To send the complete parameter set back to the DLLs and to the DCC module (e.g. after changing parameter values) the function **DCC_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 **DCC_set_parameters** by **DCC_get_parameters**.

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

Error Handling

Each DCC 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 dcc_def.h file and can be read using **DCC_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.

Safety Note

Please stay alert that the program you develop possibly control an external high voltage power supply or a detector that can be damaged by exceeding the maximum operation voltage or the maximum output current. In particular, if you control a high voltage power supply, make sure that it is safe to turn on or increase the voltage. Although the DCC-100 contains some safety features, such as detector shutdown at power-on or overload, it cannot be made safe in terms of software glitches or operator errors, such as turning on HV power supplies with open or wrong connected output cables, or exceeding the maximum operating voltage for a given detector. bh will not take responsibility for accidents or detector damage resulting from software glitches, unintentional output enable, or setting or loading values exceeding maximum values for a given experiment setup.

Description of the DCC DLL Functions

----

short CVICDECL **DCC_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 DCC 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 DCC module. This is accomplished by the function **DCC_init**. The **DCC_init** function

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

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file DCC100.INI or to start with DCC100.INI and to introduce the desired changes.

```
;   DCC100 initialisation file
;   DCC parameters have to be included in .ini file only when parameter
;   value is different from default.
;   module section (dcc_module1-8) is required for each existing DCC module

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

[dcc_module1]               ; DCC module 1 hardware parameters

active  = 1                 ; module active - can be used  (default = 0 - not active)
c1_p5V  = 0                 ; Connector 1 +5V On (1)/ Off (0), default = 0 (Off)
c1_m5V  = 0                 ; Connector 1 -5V On (1)/ Off (0), default = 0 (Off)
c1_p12V = 0                 ; Connector 1 +12V On (1)/ Off (0), default = 0 (Off)
c1_gain_HV = 0.0            ; Connector 1 Gain/HV : 0 - c1_gain_HV_limit % ( default 0%)
                            ;  c1_gain_HV_limit ( 0 - 100(default) % ) is stored in module EEPROM
c2_p5V  = 0                 ; Connector 2 +5V On (1)/ Off (0), default = 0 (Off)
c2_m5V  = 0                 ; Connector 2 -5V On (1)/ Off (0), default = 0 (Off)
c2_p12V = 0                 ; Connector 2 +12V On (1)/ Off (0), default = 0 (Off)
c2_digout = 0x0             ; Connector 2 Digital Outputs State, 0 - ff(hex) , default 0
                            ;     each bit of the value represents one output
c3_p5V  = 0                 ; Connector 3 +5V On (1)/ Off (0), default = 0 (Off)
c3_m5V  = 0                 ; Connector 3 -5V On (1)/ Off (0), default = 0 (Off)
c3_p12V = 0                 ; Connector 3 +12V On (1)/ Off (0), default = 0 (Off)
```

```
c3_cooling = 0              ; Connector 3 Cooler On (1) / Off(0), default 0 (Off)
c3_coolVolt = 0.0           ; Connector 3 Cooler Voltage 0 - 5V , default 0 V
c3_coolCurr = 0.0           ; Connector 3 Cooler Current Limit 0 - 2 Amperes , default 0
c3_gain_HV = 0.0            ; Connector 3 Gain/HV : 0 - c3_gain_HV_limit % ( default 0%)
                            ;  c3_gain_HV_limit ( 0 - 100(default) % ) is stored in module EEPROM

[dcc_module2]               ; DCC module  2 hardware parameters

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

[dcc_module3]               ; DCC module 3 hardware parameters

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

[dcc_module4]               ; DCC 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 DCC 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 **DCC_set_mode** with the parameter ‘force_use’ = 1.

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

If several DCC 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 DCC modules can be obtained by calling **DCC_get_module_info** function. The function fills DCCModInfo structure (see dcc_def.h for definition).

----

short CVICDECL **DCC_test_if_active** (short mod_no);

Input parameters:

mod_no   module number (0 - 7)

Return value:

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

Description:

The procedure returns information whether the DCC module ‘mod_no’ is active or not. As a result of a wrong initialisation (DCC_init function) a module can be deactivated. To find out the reason of deactivating the module, run the DCC_get_init_status function.

---

short CVICDECL **DCC_get_init_status**(short mod_no, short * ini_status);

Input parameters:

mod_no	module number (0 – 7)
*ini_status	pointer to the initialisation status

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

Description:

The procedure loads the ini_status variable with the initialisation result code set by the function DCC_init for module 'mod_no'. The possible values are shown below (see also dcc_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

---

short CVICDECL **DCC_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 dcc_def.h file:

#define DCC_HARD	0	hardware mode
#define DCC_SIMUL100	100	simulation mode of DCC-100

----

short CVICDECL **DCC_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 dcc_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 8 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 8 possible modules:

-if 'in_use' entry = 1 : the proper module is locked and initialised (if it wasn't) with the initial parameters set (from ini_file) but only when it was not locked by another application or when 'force_use' = 1.

-if 'in_use' entry = 0 : the proper module is unlocked and can be used further.

When one of the simulation modes is requested for each of 8 possible modules:

-if 'in_use' entry = 1 : the proper module is initialised (if it wasn't) with the initial parameters set (from ini_file).

-if 'in_use' entry = 0 : the proper module is unlocked and can be used further.

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

Use the function DCC_get_init_status and/or DCC_get_module_info to check which modules are correctly initialised and can be use further.

Use the function DCC_get_mode to check which mode is actually set. Possible 'mode' values are defined in the dcc_def.h file.

----

short CVICDECL **DCC_get_module_info** (short mod_no , DCCModInfo * mod_info);

Input parameters:

mod_no	module number (0 - 7)
* mod_info	pointer to the result structure

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

Description:

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

short module_type	module type : 100- DCC-100
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[12]	module serial number

----

short CVICDECL **DCC_get_error_string**(short error_id, char * dest_string, short max_length);

Input parameters:

error_id	DCC DLL error id (0 – number of DCC errors-1) (see dcc_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 DCC 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 dcc_def.h file.

---

short CVICDECL **DCC_get_parameter**(short mod_no, short par_id, float * value);

Input parameters:

mod_no	module number (0 - 7)
par_id	parameter identification number (see dcc_def.h)
*value	pointer to the parameter value

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

The procedure loads 'value' with the actual value of the requested parameter from the DLL-internal data structures of the DCC module 'mod_no'. The par_id values are defined in dcc_def.h file as DCC_PARAMETERS_KEYWORDS.

---

short CVICDECL **DCC_set_parameter**(short mod_no, short par_id,
short send_to_hard, float value);

Input parameters:

mod_no	module number (-1 .. 7)
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 dcc_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 DCC module 'mod_no' (if 'send_to_hard' parameter = 1).

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 par_id values are defined in dcc_def.h file as DCC_PARAMETERS_KEYWORDS.

----

short CVICDECL **DCC_get_parameters**(short mod_no, DCCdata * data);

Input parameters:

mod_no	module number (0 - 7)
*data	pointer to result structure (type DCCdata)

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

Description:

After calling the DCC_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 **DCC_get_parameters** is provided. This function transfers the parameter values of the DCC module 'mod_no' from the internal structures of the DLLs into a structure of the type DCCdata (see dcc_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 active	most of the library functions are executed only when module is active (not 0)
short c1_p5V	Connector 1 +5V On/Off
short c1_m5V	Connector 1 -5V On/Off
float c1_gain_HV	Connector 1 Gain/HV [%]
short c1_p12V	Connector 1 +12V On/Off
short c2_p5V	Connector 2 +5V On/Off
short c2_m5V	Connector 2 -5V On/Off
short c2_p12V	Connector 2 +12V On/Off
short c2_digout	Connector 2 Digital Outputs State, 0 - ff(hex)
short c3_p5V	Connector 3 +5V On/Off
short c3_m5V	Connector 3 -5V On/Off
short c3_p12V	Connector 3 +12V On/Off
float c3_coolVolt	Connector 3 Cooler Voltage (0 - 5V)
float c3_coolCurr	Connector 3 Cooler Current Limit (0 - 2A)
float c3_gain_HV	Connector 3 Gain/HV [%]
short c3_cooling	Connector 3 Cooler On/Off

----

short CVICDECL **DCC_set_parameters**(short mod_no, short send_to_hard,
DCCdata * data);

Input parameters:

mod_no	module number (0 - 7)
send_to_hard	send value to hardware (1) , or not (0)
*data	pointer to parameters structure (type DCCdata, see dcc_def.h)

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

The procedure sends all parameters from the 'DCCdata' structure to the internal DLL structures and, if 'send_to_hard' is equal 1, to the control registers of the DCC 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' and 'active' are not changed. They can be changed only by a new ini_file an a DCC_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 **DCC_get_eeprom_data**(short mod_no, DCC_EEP_Data *eep_data);

Input parameters:

mod_no	module number (0 - 7)
*eep_data	pointer to result structure

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

The structure "eep_data" is filled with the contents of the EEPROM of the DCC module 'mod_no'. The EEPROM contains the production data of the module. The structure "DCC_EEP_Data" is defined in the file dcc_def.h.

---

short CVICDECL **DCC_write_eeprom_data**(short mod_no, unsigned short write_enable, DCC_EEP_Data *eep_data);

Input parameters:

mod_no	module number (0 - 7)
write_enable	write enable password
*eep_data	pointer to result structure

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

The function is used to write data to the EEPROM of an DCC 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 **DCC_get_gain_HV_limit** (short mod_no, short lim_id, short * value);

Input parameters:

mod_no	module number (0 - 7)
lim_id	0 – Connector 1, 1 – Connector 3 gain_HV limit
*value	pointer to the parameter value

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

The procedure loads 'value' with the actual value of the gain_HV limit from the EEPROM of the DCC module 'mod_no'. Depending on 'lim_id' Connector 1 or Connector 3 gain_HV will be used.

Gain_HV limits are expressed in % of the maximum voltage which can be sent to gain_HV outputs.

----

short CVICDECL **DCC_set_gain_HV_limit** (short mod_no, short lim_id, short * value);

Input parameters:

mod_no	module number (0 - 7)
lim_id	0 – Connector 1, 1 – Connector 3 gain_HV limit
*value	pointer to the limit value, 0 - 100

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

The procedure sets the gain_HV limit of the DCC module 'mod_no' to the value taken from parameter 'value'. Depending on 'lim_id' Connector 1 or Connector 3 gain_HV will be used.

The limits are stored in the module EEPROM of the DCC module. Then corresponding parameter C1_GAIN_HV or C3_GAIN_HV will be recalculating according to the limit.

On return the 'value' variable is set to the current limit value read from EEPROM.

In case of errors during writing or reading to/from DCC EEPROM gain_HV limit is set to the default value of 100. Therefore, please **make sure** that no error occurred and use the function DCC_get_gain_HV_limit to check that the correct limit is set.

Gain_HV limits are expressed in % of the maximum voltage which can be sent to gain_HV outputs.

Caution: This function limits the detector operation voltage essentially by software. It gives reasonable safety against unintentional overload of detectors or other connected devices. The function is **not safe** in terms of human safety, and it cannot be used to exclude hazard by the output voltage of externally connected high voltage power supplies. Please make sure that a connected high voltage power supply, the devices to which the high voltage is connected, or other devices connected to the DCC are safe for the **full** output voltage range.

----

short CVICDECL **DCC_enable_outputs** (short mod_no, short enable);

Input parameters:

mod_no	module number (-1 .. 7)
enable	0 – disable, 1 – enable outputs
Return value:	0 no errors, <0 error code (see dcc_def.h)

The **DCC_enable_outputs** function is used to enable/disable outputs of the DCC module 'mod_no'.

If 'mod_no' = -1, the outputs on all active modules are enabled/disabled.

Outputs should be disabled during connecting cables or setting up devices controlled with DCC module.

Caution: Please stay alert that the function may control an external high voltage power supply or a detector that can be damaged by exceeding the maximum operation voltage or the maximum output current. In particular, if you control a high voltage power supply, make sure that it is safe to turn on the voltage. Switching off a high voltage power supply by the function DCC_enable_outputs is **not safe** in terms of human safety, and disabling the outputs cannot be used to exclude hazard by the output voltage of externally connected high voltage power supplies.

----

short CVICDECL **DCC_clear_overload** (short mod_no);

Input parameters:

mod_no	module number (-1 .. 7)
Return value:	0 no errors, <0 error code (see dcc_def.h)

The **DCC_ clear_overload** function clears overload hardware flags on DCC module 'mod_no'. It will enable again the outputs disabled by overload.

If 'mod_no' = -1, overload will be cleared on all active DCC modules.

Caution: Please stay alert that the function may control an external high voltage power supply or a detector that can be damaged by exceeding the maximum operation voltage or the maximum output current. In particular, if you control a high voltage power supply, make sure that it is safe to turn the voltage.

----

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

Input parameters:

mod_no	module number (0 .. 7)
*state	pointer to result variable

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

The procedure is used to check whether an overload occurred on the DCC module 'mod_no'.

Bit 0 of the 'state' variable is set when Connector 1 Overload is present, otherwise it is 0.

Bit 1 of the 'state' variable is set when Connector 3 Overload is present, otherwise it is 0.

----

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

Input parameters:

mod_no	module number (0 .. 7)
*state	pointer to result variable

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

The procedure is used to check whether the current limit is reached (for cooling device on Connector 3) on the DCC module 'mod_no'.

'state' variable is set to 0 or 1 according to the Current limit flag.