DDG
Dynamic Link Libraries

USER MANUAL

Version 3.2, February 2023

Content

Introduction

The **DDG** Dynamic Link Library contains all functions to control the DDG modules. The functions work under 64 bit Windows 10/11. Both 32 and 64-bit DLL versions are available. The program which calls the DLLs must be compiled with the compiler option 'Structure Alignment' set to '1 Byte'.

The distribution package contain the following files:

DDG32.DLL	32-bit dynamic link library main file for use on 32-bit systems
DDG32x64.DLL	32-bit dynamic link library main file for use on 64-bit systems
DDG32.LIB	import library file for Microsoft Visual C/C++ for use on 32-bit systems
DDG32x64.LIB	import library file for Microsoft Visual C/C++ for use on 64-bit systems
DDG64.DLL	64-bit dynamic link library main file for use on 64-bit systems
DDG64.LIB	import library file for Microsoft Visual C/C++ for use on 64-bit systems
DDG_DEF.H	Include file containing Types definitions, Functions Prototypes and Pre-processor statements
DDG200.SET	default DDG DLL settings file
DDG_DLL.DOC	This description file
USE_DDG.C	The simple example of using DDG DLL functions. Choose the correct import library file to link in your compiler.

The installation program installs correct versions of the DLL depending on the operating system (32 or 64 bit systems require different DLL versions) together with the driver required to control DDG resources.

DDG-DLL Functions List

The following functions are implemented in the DDG-DLL:

Initialisation Functions:

DDG_init
DDG_test_if_active
DDG_get_init_status
DDG_get_module_info
DDG_get_version
DDG_get_mode
DDG_set_mode
DDG_get_version
DDG_get_error_string

Setup Functions:

DDG_get_parameter
DDG_set_parameter
DDG_get_parameters
DDG_set_parameters
DDG_get_pulses
DDG_get_pulses_from_file

DDG_set_pulses
DDG_add_pulse
DDG_get_chan_info
DDG_read_settings_from_file
DDG_save_settings_to_file
DDG_get_eeprom_data
DDG_write_eeprom_data
DDG_get_adjust_parameters
DDG_set_adjust_parameters
DDG_set_idle_state

Status Functions:

DDG_read_status
DDG_read_curr_count

Control Functions:

DDG_start
DDG_stop

DDG Channel's Memory Read/Write Functions:

DDG_read_chan_data
DDG_write_chan_data

Functions listed above must be called with C calling convention which is default for C and C++ programs.
Identical set of functions is available for environments like Visual Basic which requires _stdcall calling convention. Names of these functions have 'std' letters after 'DDG', for example, DDGstd_start it is _stdcall version of DDG_start.
Description and behaviour of these functions are identical to the functions from the first (default) set – the only difference is calling convention.

Another set of the functions is added for use **only in LabView environment**. The names of these functions have '_LV' letters after 'DDG'
' for example, DDG_LV_set_pulses it is LabView version of DDG_set_pulses.
These functions are added to treat correctly LabView clusters with strings (inside) used as an input parameter to DDG functions – it affects all functions which have a structure with string field(s) as a parameter.
Description and behaviour of these functions are identical to the functions from the first (default) set.

Application Guide

Initialisation of the DDG 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 DDG module. This is accomplished by the function **DDG_init**.

The DDG DLL Functions can control up to four DDG modules on PCI bus (DDG-200/210/220)

The **DDG_init** function

- reads the parameter values from a specified settings file, or set them to default
- sends the parameter values to the DDG control registers on active DDG module
- reads channels pulses from the file, verifies them and loads to channels memory all verified pulses
- performs a hardware test (memory test & EEPROM checksum test) of active DDG module

The settings file is an ASCII file with a structure shown in the table below. We recommend either to use the file DDG200.SET or to start with DDG200.SET and to introduce the desired changes.

```
;   DDG settings file
;   DDG parameters have to be included in .set file only when parameter
;   value is different from default.

[ddg_base]
simulation = 0                  ; 0 - hardware mode(default),
                                ; >0 - simulation mode (see ddg_def.h for possible values)
pci_bus_no = -1                 ; PCI bus on which DDG modules will be looking for
                                ; 0 - 255, default -1 (all PCI busses will be scanned)
pci_card_no = 0                 ; number of DDG module on PCI bus to be initialised
                                ; 0 - 3, default 0 (1st module on PCI bus)

[ddg_module]                    ; DDG module hardware parameters

repeat = 2                      ; 0 - single cycle, 1 - cyc_count cycles,
                                ; 2 - continuous work (default)
cyc_count = 100                 ; cycles counter (1 ... 4095), default 100
                                ; valid when repeat = 1
rep_time = 1000                 ; repeat time (in 10ns units), 3 ... 0xffffffff, default 1000
pulse_width = 100               ; start pulse width (in 10ns units), default 100
                                ; DDG-200 2 ... 0xfffffffe, DDG-210 5 ... 0xfffffffe, DDG-220 1...0xfffffffe
pulse_offset = 0                ; start pulse offset (in 2.5 ns units), default 0 (only for DDG-220)
                                ; DDG-220 0 ... 0x00007fff
polarity = 0x3f                 ; polarity of delayed pulses - 6-bit value
                                ; bit 0 ... 5 polarity of channels 1 ... 6
                                ; bit = 0 negative, = 1 positive,
                                ; default 0x3f - all channels positive
trigger = 0                     ; external trigger condition, only for DDG-210/DDG-220
                                ; bits 1 & 0 mean:
                                ;     00 - (value 0) external trigger disabled (default),
                                ;     01 - (value 1) external trigger active low,
                                ;     10 - (value 2) external trigger active high
                                ; bit 2 = 1 (value 4) - trigger each cycle
                                ;        = 0   - trigger only sequence
                                ; bit 11 = 1 - 50 Ohm termination, only for DDG-220
                                ;      0 - High-Impedance
trig_threshold= 0.1             ; trigger threshold level (only for DDG-210/DDG-220)
                                ; (-1.0 ... 1.0 V, default 0.1 V)

                                ; Channels pulses - 6 channels, max 64 pulses per channel
                                ; Start Width Fine offset (times in 10 ns units)

                                [ddg_chan1]
                                     5     10
                                    50     10
                                   100     10
                                [ddg_chan2]
                                [ddg_chan3]
                                [ddg_chan4]
                                [ddg_chan5]
                                [ddg_chan6]
```

The module will be initialised, but only when the DLL was registered correctly during installation (if required, using License number) and when it is not in use (locked) by other application.

If, for some reasons, the module which was locked must be initialised, it can be done using the function DDG_set_mode with the parameter 'force_use' = 1.

After successful initialisation the module is locked to prevent that other application can access it.

After an **DDG_init** call we recommend to call the **DDG_test_if_active** function to check whether required (module pci_card_no on PCI bus) DGG module is active. Only active module can be operated further. It is recommended (but not required) to check also the initialisation status (by **DDG_get_init_status**) of the used module. In case of a wrong initialisation the initialisation status shows the reason of the error (see ddg_def.h for possible values).

Additional information about DDG modules can be obtained by calling **DDG_get_module_info** function. The function fills DDGModInfo structure which is described below.

short module_type	module type: 200- DDG-200
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
int init	set to initialisation result code
unsigned long * base_adr	pointer to DDG resources (memory transfers only)
char serial_no[16]	module serial number

After calling the **DDG_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 **DDG_get_parameters** is provided. This function transfers the parameter values from the internal structures of the DLLs into a structure of the type DDGdata (see ddg_def.h) which has to be declared by the user. The parameter values in this structure are described below.

short mod_no	module no on PCI bus (0-3)
short init	set to initialisation result code
short polarity	polarity of delayed pulses - 6-bit value bit 0 ... 5 polarity of channels 1 ... 6 bit = 0 negative = 1 positive, default 0x3f - all channels positive
short repeat	0 - single cycle, 1 - cyc_count cycles, 2 - continuous work (default)
unsigned long cyc_count	cycles counter (1 ... 4095), default 100 valid when repeat = 0
unsigned long rep_time	repeat time (in 10ns units), 3 ... 0xffffffff, default 1000
unsigned long pulse_width	start pulse width (in 10ns units), default 100 DDG-200 2 ... 0xfffffffe, DDG-210 5 ... 0xfffffffe, DDG-220 1 ... 0xfffffffe
pulse_offset = 0	start pulse offset (in 2.5ns units), default 0 (only for DDG-220) ; DDG-220 0 ... 0x00007fff
short trigger;	external trigger condition, only for DDG-210/220 bits 1 & 0 mean: 00 - (value 0) external trigger disabled (default), 01 - (value 1) external trigger active low, 10 - (value 2) external trigger active high bit 2 = 1 (value 4) - trigger each cycle = 0 - trigger only sequence ; bit 11 = 1 - 50 Ohm termination, only for DDG-220 ; 0 - High-Impedance

float trig_threshold	trigger threshold level (only for DDG-210/220) (-1.0 ... 1.0 V, default 0.1 V)

To send the complete parameters set back to the DLLs and to the DDG module (e.g. after changing parameter values) the function **DDG_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 **DDG_set_parameters** by **DDG_get_parameters**.

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

After initialisation **DDG_read_settings_from_file** function can also be used to load all parameters and channels pulses from the settings file.

Channel Pulses Read/Write Functions

Channel pulse is defined as a set of three unsigned 32-bit values which mean:

- Start time of the pulse [10ns], min = 2 for DDG-200, 5 for DDG-210, 1 for DDG-220, max = 4294967294 (0xfffffffe)
- Width of the pulse [10ns], 1 ... 524288
- Fine offset for the pulse start [1.25 ns], 0 ... 7

Channel pulses can be stored in [ddg_chan1] ... [ddg_chan6] sections of DDG settings files.

Non-existing Fine offset value in ddg_chan sections means offset equal 0.

Each DDG channel can contain up to 64 pulses.

The minimum distance between the end of the pulse and the start of the next pulse is equal 1[10ns] except the situation, when fine offsets of the pulses are not equal. Then the distance must be minimum 4 [10ns].

The minimum distance between start times of two subsequent pulses is equal 3 [10ns].

DDG_init, **DDG_read_settings_from_file** and **DDG_get_pulses_from_file** functions read channels pulses from the settings file, verify them and load to channels memory all verified pulses.

Pulses can be read from the DDG channel to the buffer using **DDG_get_pulses** function. **DDG_write_pulses** function verifies pulses from the buffer and writes all verified pulses to one of the DDG channels. **DDG_add_pulse** extracts the pulse from the text string and after verification writes it to one of the DDG channels.

Information about the channel's pulses is written to DDGChanInfo structure (see ddg_def.h file) and can be read using **DDG_get_chan_info**. In case of errors during verification of the pulses DDGChanInfo structure delivers error explanation.

Channel Memory Read/Write Functions

These functions are obsolete and they are present only for compatibility with the older DLL versions.

Reading and writing the channel's memory of the DDG module is accomplished by the functions **DDG_read_chan_data** and **DDG_write_chan_data**.

It is strongly recommended to use described earlier channel pulses r/w functions instead of reading and writing the channel's memory. Without detailed information about DDG hardware these functions are useless.

Standard Measurements

The most important measurement functions are listed below.

The **DDG_read_status** function returns the current status of the DDG module. The status bits delivered by the function are listed below (see also ddg_def.h).

For DDG-200/210/220 modules

STATUS_RUN	8	sequence is running
STATUS_END	4	sequence was finished (after last cycle)
For DDG-210/220 modules only		
STATUS_WAIT_TRG	16	sequence waits for trigger

DDG_start starts the sequence on the DDG module. Depending on the parameter 'REPEAT' setting the sequence runs continuously or it runs number of cycles set for parameter 'CYC_CYCLE' or runs one cycle only.

DDG-210 module checks trigger condition and if the trigger is enabled it will wait for trigger pulse before the 1st or every cycle depending on the TRIGGER parameter value.

To check whether a measurement is finished, the **DDG_read_status** function is used.

A running sequence can be stopped by the **DDG_stop** function.

Error Handling

Each DDG 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 ddg_def.h file and can be read using **DDG_get_error_string**. We recommend checking the return value after each function call.

Using DLL Functions in LabView Environment

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

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

Type in C programs	Type in LabView
char	signed 8-bit integer, byte (I8)
unsigned char	unsigned 8-bit integer, unsigned byte (U8)
short	signed 16-bit integer, word (I16)
unsigned short	unsigned 16-bit integer, unsigned word (U16)
long, int	signed 32-bit integer, long (I32)
unsigned long, int	unsigned 32-bit integer, unsigned long (U32)
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 DDG DLL Functions

---

short **DDG_init** (char * set_file);

Input parameters:

* set_file: pointer to a string containing the name of the initialisation file in use (including file name and extension) (can be NULL)

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 DDG module. This is accomplished by the function **DDG_init**. The **DDG_init** function

- reads the parameter values from a specified initialisation file,
  or set parameters to default if parameter 'set_file' = NULL
- sends the parameter values to the DDG control registers on active DDG module
- reads channels pulses from the file, verifies them and loads to channels memory all verified pulses
- performs a hardware test (memory test & EEPROM checksum test) of active DDG module

The initialisation file is an ASCII file with a structure shown in the table below. We recommend either to use the file DDG200.SET or to start with DDG200.SET and to introduce the desired changes.

```
;   DDG settings file
;   DDG parameters have to be included in .set file only when parameter
;   value is different from default.

[ddg_base]
simulation = 0                  ; 0 - hardware mode(default),
                                ; >0 - simulation mode (see ddg_def.h for possible values)
pci_bus_no = -1                 ; PCI bus on which DDG modules will be looking for
                                ; 0 - 255, default -1 (all PCI busses will be scanned)
pci_card_no = 0                 ; number of DDG module on PCI bus to be initialised
                                ; 0 - 3, default 0 (1st module on PCI bus)

[ddg_module]                    ; DDG module hardware parameters
repeat = 2                      ; 0 - single cycle, 1 - cyc_count cycles,
                                ; 2 - continuous work (default)
cyc_count = 100                 ; cycles counter (1 ... 4095), default 100
                                ; valid when repeat = 1
rep_time = 1000                 ; repeat time (in 10ns units), 3 ... 0xffffffff, default 1000
pulse_width = 100               ; start pulse width (in 10ns units), default 100
                                ; DDG-200 2 ... 0xfffffffe, DDG-210 5 ... 0xfffffffe, DDG-220 1 ... 0xfffffffe
pulse_offset = 0                ; start pulse offset (in 2.5 ns units), default 0 (only for DDG-220)
                                ; DDG-220 0 ... 0x00007fff

polarity = 0x3f                 ; polarity of delayed pulses - 6-bit value
                                ; bit 0 ... 5 polarity of channels 1 ... 6
```

```
                                ; bit = 0 negative, = 1 positive,
                                ; default 0x3f - all channels positive
trigger = 0                     ; external trigger condition, only for DDG-210/DDG-220
                                ; bits 1 & 0 mean:
                                ;     00 - (value 0) external trigger disabled (default),
                                ;     01 - (value 1) external trigger active low,
                                ;     10 - (value 2) external trigger active high
                                ; bit 2 = 1 (value 4) - trigger each cycle
                                ;         = 0 - trigger only sequence
                                ; bit 11 = 1 - 50 Ohm termination, only for DDG-220
                                ;      0 - High-Impedance
trig_threshold= 0.1             ; trigger threshold level (only for DDG-210/DDG-220)
                                ;    (-1.0 ... 1.0 V , default 0.1 V)

                                ; Channels pulses - 6 channels, max 64 pulses per channel
                                ; Start Width Fine offset (times in 10ns units)

                                [ddg_chan1]
                                   5      10
                                  50      10
                                 100      10

                                [ddg_chan2]
                                [ddg_chan3]
                                [ddg_chan4]
                                [ddg_chan5]
                                [ddg_chan6]
```

The module will be initialised, but only when the DLL was registered correctly during installation (if required, using License number) and when it is not in use (locked) by other application.

If, for some reasons, the module which was locked must be initialised, it can be done using the function DDG_set_mode with the parameter ‘force_use’ = 1.

After successful initialisation the module is locked to prevent that other application can access it.

After a **DDG_init** call we recommend to call the **DDG_test_if_active** function to check whether required (module pci_card_no on PCI bus) DGG module is active. Only active module can be operated further. It is recommended (but not required) to check also the initialisation status (by **DDG_get_init_status**) of the used module. In case of a wrong initialisation the initialisation status shows the reason of the error (see ddg_def.h for possible values).

Additional information about DDG modules can be obtained by calling **DDG_get_module_info** function. The function fills DDGModInfo structure (see ddg_def.h for definition).

---

short **DDG_test_if_active** (short mod_no);

Input parameters:

mod_no 0 ... 3, DDG module number

Return value:

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

Description:

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

---

short **DDG_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 ddg_def.h)

Description:

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

INIT_DDG_OK	0	no error
INIT_DDG _NOT_DONE	-1	init not done
INIT_DDG _WRONG_EEP_CHKSUM	-2	wrong EEPROM checksum
INIT_DDG _CANT_OPEN_PCI_CARD	-3	cannot open PCI card
INIT_DDG _MOD_IN_USE	-4	module already in use
INIT_DDG _HARD_TEST_ERR	-5	hardware test failed
INIT_DDG _WRONG_LICENSE	-6	wrong or missing license key
INIT_DDG _NO_LICENSE	-7	license key not read from registry
INIT_DDG _LICENSE_NOT_VALID	-8	license is not valid for DDG DLL
INIT_DDG _LICENSE_DATE_EXP	-9	license date expired
INIT_DDG _FIRMWARE_VER	-10	incorrect firmware version of DDG module

Up to 4 modules can be initialised.

---

short **DDG_get_module_info** (short mod_no, DDGModInfo * 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 ddg_def.h)

Description:

After calling the DDG_init function (see above) the DDGModInfo internal structures for all 4 modules are filled. This function transfers the contents of the internal structure of the DLL of module 'mod_no' into a structure of the type DDGModInfo (see ddg_def.h) which has to be defined by the user. The parameters included in this structure are described below.

short module_type	module type: 200- DDG-200, 210 – DDG-210, 220 – DDG-220
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
int init	set to initialisation result code
unsigned long * base_adr	pointer to DDG resources (memory transfers only)
char serial_no[16]	module serial number

---

short **DDG_get_version** (short mod_no, long * version);

Input parameters:

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

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

Description:

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

---

short **DDG_get_mode** (void);

Input parameters:

none

Return value: current mode of DLL operation

Description:

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

#define DDG_HARD	0	hardware mode

#define DDG_SIMUL200	200	simulation mode of DDG-200
#define DDG_SIMUL210	210	simulation mode of DDG-210
#define DDG_SIMUL220	220	simulation mode of DDG-220

---

short **DDG_set_mode** (short mode, short force_use, short *in_use);

Input parameters:

mode: mode of DLL operation

force_use force using the module if they are locked (in use)

*in_use pointer to the table with information which module must be used

Return value: 0 no errors, <0 error code (see ddg_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.

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 DDG_get_init_status and/or DDG_get_module_info to check which modules are correctly initialised and can be use further.

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

---

short **DDG_get_error_string** (short error_id, char * dest_string,
short max_length);

Input parameters:

error_id	DDG DLL error id (0 – number of DDG errors-1) (see ddg_def.h file)
*dest_string	pointer to destination string
max_length	max number of characters which can be copied to 'dest_string'

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

The procedure copies to 'dest_string' the string which contains the explanation of the DDG DLL error with id equal 'error_id'. Up to 'max_length characters will be copied.

Possible 'error_id' values are defined in the ddg_def.h file.

---

short **DDG_get_parameter** (short mod_no, short par_id, float * value);

Input parameters:

mod_no	0 ... 3, DDG module number
par_idparameter identification number (see ddg_def.h)	
*value	pointer to the parameter value

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

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

---

short **DDG_set_parameter** (short mod_no, short par_id, float value);

Input parameters:

mod_no	0 ... 3, DDG module number
par_idparameter identification number	
value	new parameter value

Return value:

0 no errors, <0 error code (see ddg_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 DDG module 'mod_no'. 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 ddg_def.h file as DDG_PARAMETERS_KEYWORDS.

---

short **DDG_get_parameters** (short mod_no, DDGdata * data);

Input parameters:

mod_no	0 ... 3, DDG module number
*data	pointer to result structure (type DDGdata)

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

Description:

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

short mod_no	module no on PCI bus (0-3)
short init	set to initialisation result code
short polarity	polarity of delayed pulses - 6-bit value bit 0 ... 5 polarity of channels 1 ... 6 bit = 0 negative, = 1 positive, default 0x3f - all channels positive
short repeat	0 - single cycle, 1 - cyc_count cycles, 2 - continuous work (default)
unsigned long cyc_count	cycles counter (1 ... 4095), default 100 valid when repeat = 0
unsigned long rep_time	repeat time (in 10ns units), 3 ... 0xffffffff, default 1000
unsigned long pulse_width	start pulse width (in 10ns units), default 100 DDG-200 2 ... 0xfffffffe, DDG-210 5 ... 0xfffffffe, DDG-220 1 ... 0xfffffffe
pulse_offset = 0	start pulse offset (in 2.5 ns units), default 0 (only for DDG-220) ; DDG-220 0 ... 0x00007fff
short trigger;	external trigger condition, only for DDG-210/220 bits 1 & 0 mean: 00 - (value 0) external trigger disabled (default), 01 - (value 1) external trigger active low, 10 - (value 2) external trigger active high bit 2 = 1 (value 4) - trigger each cycle = 0 - trigger only sequence ; bit 11 = 1 - 50 Ohm termination, only for DDG-220

	; 0 - High-Impedance
float trig_threshold	trigger threshold level (only for DDG-210/220) (-1.0 ... 1.0 V, default 0.1 V)

---

short **DDG_set_parameters** (short mod_no, DDGdata * data);

Input parameters:

mod_no	0 ... 3, DDG module number
*data	pointer to parameters structure (type DDGdata, see ddg_def.h)

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

Description:

The procedure sends all parameters from the 'DDGdata' structure to the internal DLL structures and to the control registers of the DDG 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 'mod_no' and 'init' are not changed. They can be changed only by a new set_file in a DDG_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 **DDG_get_pulses** (short mod_no, short chan_no, unsigned long * data);

Input parameters:

mod_no	0 ... 3, DDG module number
chan_no	channel number (0 - 5)
*data	pointer to the channel pulses buffer

Return value: >= 0 no errors – number of channel pulses,
<0 error code (see ddg_def.h)

Description:

The procedure fills buffer 'data' with channel 'chan_no' pulses of the module 'mod_no' and returns on success number of channel pulses.

Please make sure that the buffer 'data' be allocated with enough memory for up to 64 pulses.

Channel's pulse is defined as a set of three unsigned 32-bit values which mean:

- Start time of the pulse [10ns], min = 2 for DDG-200, 5 for DDG-210, 1 for DDG-220, max = 4294967294 (0xfffffffe)
- Width of the pulse [10ns], 1 ... 524288
- Fine offset for the pulse start [1.25ns], 0 ... 7

---

short **DDG_get_pulses_from_file** (short mod_no, short chan_no, short send_to_hard,
char * file_name, unsigned long * data,
DDGChanInfo *info);

Input parameters:

mod_no	0 ... 3, DDG module number
chan_no	channel number (0 - 5)
send_to_hard	1(0) send or not verified pulses to channel memory
file_name	DDG settings file name
*data	pointer to the channel pulses buffer
*info	pointer to DDGChanInfo channel info structure filled on exit

Return value: >= 0 no errors, <0 error code (see ddg_def.h)

Description:

The procedure looks for [ddg_chan] section of the channel 'chan_no' in the file 'file_name'. The sections are numbered from 1 to 6, so the proper section number is 'chan_no' + 1.

If the section is found all its pulses lines are interpreted and added to the buffer 'data' (if no errors were found in the pulse line). The action is stopped when an error in a pulse line is found. Pulse line with error and error explanation together with the number of verified pulses are written to DDGChanInfo 'info' structure (see ddg_def.h file for definition).

If 'send_to_hard' equals 1, all correctly verified pulses are written to the DDG channel 'chan_no' memory of the module 'mod_no'.

Please make sure that the buffer 'data' be allocated with enough memory for up to 64 pulses.

Channel's pulse is defined as a set of three unsigned 32-bit values which mean:

- Start time of the pulse [10ns], min = 2 for DDG-200, 5 for DDG-210, 1 for DDG-220, max = 4294967294 (0xfffffffe)
- Width of the pulse [10ns], 1 ... 524288
- Fine offset for the pulse start [1.25ns], 0 ... 7

---

short **DDG_set_pulses** (short mod_no, short chan_no, unsigned long * data,
DDGChanInfo *info);

Input parameters:

mod_no	0 ... 3, DDG module number
chan_no	channel number (0 - 5)
*data	pointer to the channel pulses buffer
*info	pointer to DDGChanInfo channel info structure filled on exit

Return value: >= 0 no errors, <0 error code (see ddg_def.h)

Description:

The procedure analyses all pulses from the buffer ‘data’ (number of pulses is given in ‘info’ structure) and writes them to the DDG channel ‘chan_no’ memory (previous channel’s memory contents is overwritten) of the module ‘mod_no’.

The action is stopped when an error in a pulse definition is found.

Pulse line with error and error explanation together with the number of verified pulses are written to DDGChanInfo ‘info’ structure (see ddg_def.h file for definition).

Please make sure that the buffer ‘data’ be allocated with enough memory for required number of pulses.

Channel’s pulse is defined as a set of three unsigned 32-bit values which mean:

- Start time of the pulse [10ns], min = 2 for DDG-200, 5 for DDG-210, 1 for DDG-220, max = 4294967294 (0xfffffffe)
- Width of the pulse [10ns], 1 ... 524288
- Fine offset for the pulse start [1.25ns], 0 ... 7

---

short **DDG_add_pulse** (short mod_no, char * pulse, unsigned long * chan_buf, DDGChanInfo *info, short force, short send_to_chan);

Input parameters:

mod_no	0 ... 3, DDG module number
pulse	string with pulse definition
*chan_buf	pointer to the channel pulses buffer
*info	pointer to DDGChanInfo channel info structure filled on exit on input contains initial number of pulses in ‘chan_buf’ buffer
force	1(0) force or not adding the pulse in case of conflicts with other pulses

send_to_chan	0 ... 5 send all verified pulses to 'send_to_chan' channel's memory -1 do not send pulses to the memory

Return value: 0 – pulse not added because of conflicts, 'info' explains conflict

1 - pulse added (no conflicts),

2 – pulse added with resolving conflicts (when 'force' = 1),
< 0 error code (see ddg_def.h)

Description:
Initial number of pulses in 'chan_buf' is given in 'info' structure.

The procedure analyses 'pulse' string to get pulse definition from it.

Procedure checks then whether new pulse makes conflict with one or more pulses in 'chan_buf'.

If there are no conflicts or when 'force' is equal 1 new pulse is added to the chan_buf of the module 'mod_no'.

Remember that, if new pulse conflicts with some pulses but it is added due to 'force' = 1, resulting buffer contents and number of pulses can differ significantly from initial state, because all conflicting pulses will be removed.

When 'send_to_chan' is in range from 0 to 5, new 'chan_buf' contents is written to the proper DDG channel's memory (previous channel's memory contents is overwritten) of the module 'mod_no'.

Pulse line with error and error explanation (if any) together with the final number of pulses after adding the new pulse are written to DDGChanInfo 'info' structure (see ddg_def.h file for definition).

Please make sure that the buffer 'chan_buf' be allocated with enough memory for up to 64 pulses.

Channel's pulse is defined as a set of three unsigned 32-bit values which mean:

- Start time of the pulse [10ns], min = 2 for DDG-200, 5 for DDG-210, 1 for DDG-220, max = 4294967294 (0xfffffffe)
- Width of the pulse [10ns], 1 ... 524288
- Fine offset for the pulse start [1.25ns], 0 ... 7

---

short **DDG_get_chan_info** (short mod_no, short chan_no, DDGChanInfo *info);

Input parameters:

mod_no	0 ... 3, DDG module number
chan_no	channel number (0 - 5)
*info	pointer to DDGChanInfo structure filled on exit

Return value: >= 0 no errors, <0 error code (see ddg_def.h)

Description:

The procedure fills ‘info’ structure with the contents of internal DLL DDGChanInfo structure of DDG channel ‘chan_no’ on the module ‘mod_no’ (see ddg_def.h file for definition).

DDGChanInfo structure contains current number of pulses in channel’s memory and the information about the last pulse which was not verified and not added.

---

short **DDG_read_settings_from_file** (short mod_no, char * file_name);

Input parameters:

mod_no	0 ... 3, DDG module number
file_name	pathname of DDG settings file

Return value: >= 0 no errors, <0 error code (see ddg_def.h)

Description:

The procedure reads all DDG module ‘mod_no’ parameters from ‘file_name’ DDG settings file.

The function

- reads the parameter values from a specified settings file
- sends the parameter values to the DDG control registers on active DDG module
- reads channels pulses from the file, verifies them and loads to channels memory all verified pulses

---

short **DDG_save_settings_to_file** (short mod_no, char * file_name, char * comments);

Input parameters:

mod_no	0 ... 3, DDG module number
file_name	pathname of DDG settings file
comments	pointer to the buffer with comments lines which will be added to the destination file

Return value: >= 0 no errors, <0 error code (see ddg_def.h)

Description:

The procedure writes all DDG module ‘mod_no’ parameters and channels pulses to the file_name’ DDG settings file. The contents of the ‘comments’ buffer’ is also written to the file as comment lines before [ddg_base] section.

---

short **DDG_get_eeprom_data** (short mod_no, DDG_EEP_Data *eep_data);

Input parameters:

mod_no	0 ... 3, DDG module number
*eep_data	pointer to result structure

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

The structure "eep_data" is filled with the contents of the EEPROM of the DDG module ‘mod_no’. The EEPROM contains the production data and adjust parameters of the module. The structure "DDG_EEP_Data" is defined in the file ddg_def.h.

---

short **DDG_write_eeprom_data** (short mod_no, nsigned short write_enable,
DDG_EEP_Data *eep_data);

Input parameters:

mod_no	0 ... 3, DDG module number
write_enable	write enable password
*eep_data	pointer to result structure

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

The function is used to write data to the EEPROM of a DDG 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 **DDG_get_adjust_parameters** (short mod_no, DDG_Adjust_Para * adjpara);

Input parameters:

mod_no	0 ... 3, DDG module number
* adjpara	pointer to result structure

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

The structure 'adjpara' is filled with adjust parameters of the module ‘mod_no’ that are currently in use. The parameters can either be previously loaded from the EEPROM by DDG_init or DDG_get_eeprom_data or - not recommended - set by DDG_set_adust_parameters.

The structure "DDG_Adjust_Para" is defined in the file ddg_def.h. It contains adjust values for start pulse and all delayed pulses.

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

---

short **DDG_set_adjust_parameters** (short mod_no, DDG_Adjust_Para * adjpara);

Input parameters:

mod_no	0 ... 3, DDG module number
* adjpara	pointer to result structure

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

The adjust parameters in the internal DLL structures (not in the EEPROM) of the module ‘mod_no’ 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 DDG_init. The next call to DDG_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 "DDG_Adjust_Para" is defined in the file ddg_def.h.

---

short **DDG_set_idle_state** (short mod_no);

Input parameters:

mod_no	0 ... 3, DDG module number

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

DDG module ‘mod_no’ is set to an idle state – trigger is disabled and polarity of all pulses is set to positive (idle state is low).

---

short **DDG_read_status** (short mod_no, short * status);

Input parameters:

mod_no	0 ... 3, DDG module number
*status	pointer to result value

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

The **DDG_read_status** function returns the current status of the DDG module ‘mod_no’.
The status bits delivered by the function are listed below (see also ddg_def.h).

For DDG-200/210/220 modules

STATUS_RUN	8	sequence is running
STATUS_END	4	sequence was finished (after last cycle)

For DDG-210/220 modules only

STATUS_WAIT_TRG	16	sequence waits for trigger

The function is normally used to test whether the sequence is still running.

---

short **DDG_get_curr_count** (short mod_no, short *curr_count);

Input parameters:

mod_no	0 ... 3, DDG module number
* curr_count	pointer to result value

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

The **DDG_get_curr_count** function fills ‘curr_count’ with the current value of cycles counter on the DDG module ‘mod_no’.

The function is used to check how many cycles were already completed during the running sequence (after DDG_start).

---

short **DDG_start** (short mod_no);

Input parameters:

mod_no	0 ... 3, DDG module number	
none		
Return value:	0 no errors, <0	error code (see ddg_def.h)

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

The sequence continues dependently on ‘REPEAT’ parameter.

‘REPEAT’ = 0 - single cycle only,

‘REPEAT’ = 1 - continues until the ‘CYC_COUNT’ number of cycles has been completed,

‘REPEAT’ = 2 - continues endless until **DDG_stop** is called.

Additionally DDG-210 module checks trigger condition and if the trigger is enabled it will wait for trigger pulse before the 1st or every cycle depending on the TRIGGER parameter value.

The procedure **DDG_read_status** can be used to check whether the sequence was completed.

After the sequence was successfully completed **DDG_stop** should be called to set the module to the initial state.

---

short **DDG_stop** (short mod_no);

Input parameters:

mod_no	0 ... 3, DDG module number	
Return value:	0 no errors, <0	error code (see ddg_def.h)

DDG_stop is used to stop the running sequence on the module ‘mod_no’ by a software command.

---

short **DDG_read_chan_data** (short mod_no, short chan_no, short from,
short words_no, unsigned long * data);

Input parameters:

mod_no	0 ... 3, DDG module number
chan_no	channel number (0 – 5)
from	1st address to read (0 – 127)
words_no	number of 32-bit words to read (1 – (128- from))
*data	pointer to data buffer to be filled

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

The procedure is obsolete and it is present only for compatibility with the older DLL versions.

The procedure is used to read the memory of the channel ‘chan_no’ on the DDG module ‘mod_no’.

The procedure reads the channel’s memory from the address ‘from’ up to the address ‘from’ + ‘words_no’ and writes the values to the buffer ‘data’.

Please make sure that the buffer ‘data’ be allocated with enough memory for the required number of words (‘words_no’).

It is strongly recommended to use described earlier channel pulses r/w functions instead of reading and writing the channel’s memory. Without detailed information about DDG hardware these functions are useless.

---

short **DDG_write_chan_data** (short mod_no, short chan_no, short from,
short words_no, unsigned long * data);

Input parameters:

mod_no	0 ... 3, DDG module number
chan_no	channel number (0 – 5)
from	1st address to read (0 – 126) – must be even
words_no	number of 32-bit words to read (2 – (128- from)) – must be even
*data	pointer to data buffer

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

The procedure is obsolete and it is present only for compatibility with the older DLL versions.

The procedure is used to write the memory of the channel ‘chan_no’ of the DDG module ‘mod_no’.

The procedure writes the values from the buffer ‘data’ to the channel’s memory from the address ‘from’ up to the address ‘from’ + ‘words_no’.

Writing channel’s memory is always done from an even address and with an even number of words. Therefore parameters ‘from’ and ‘words_no’ must have even value.

It is strongly recommended to use described earlier channel pulses r/w functions instead of reading and writing the channel’s memory. Without detailed information about DDG hardware these functions are useless.

==========