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

July 2002

Step Motor Control Module STP-240/340
DLL Libraries

User Manual

Version 2.0

July 2002

Introduction

To support the user to create his own software, libraries for the basic module control functions are available.

The STP 32 bits Dynamic Link Library contains all functions to control STP-240 and STP-340 modules. STP-340 is a PCI bus module with the same functionality as the STP-240 module.
The functions work under Windows 9x/ NT / 2K / XP.
The program which calls the DLLs must be compiled with the compiler option 'Structure Alignment' set to '1 Byte'.

To facilitate the first steps of the library application we deliver the source code of the sample program use_stp.c. Program performs the initialisation of STP-240(340) module and some stepping actions.

For LabView users we deliver also STP_functions.llb library which contains implementation of STP DLL functions together with a simple LabView program to control STP module.

Installation

Execute "setup.exe" from diskette no. 1 and follow the instructions on the screen.

Configuration Files

The configuration of the system is described in a configuration file, normally stp.cfg. When the system is initialised by calling the function STP_init of the library, the information from this file is transferred to the internal data structures of the library and to the internal registers of the STP module.

The structure of the stp.cfg file is shown below. Comments within the configuration file are preceded by a semicolon. Texts which are contained in the configuration file are printed in type writer style. Explanations to the file items are printed as this text.

The first section contains general parameters:

```
[base]
baseadr= 0x3a0   ; base I/O address of the STP-240 module ,default=0x3a0
use_mot= 3       ; use motor (0- none(default) ,1 - motor1, 2 - motor2, 3 - both)
                 ;   !!! only used motors will be initialised
          ;  use value=0 ,if you work with SPC (PHC/PMS/PMM/MSA) module, but without
          ;    STP module and you want to use scale definitions
                 ;    from section device1(2)
use_dev= 0       ; parameters from section  device1 will be used
                 ;  (0- none(default) , 2 - device2)
                 ; this parameter will be used only when use_mot = 0
simulation =  0    ; 0 - hardware mode(default) ,
                   ; >0 - simulation mode (see stp_def.h for possible values)
pci_card_no = 0  ; number of STP module on PCI bus (if any exists) to be initialised
                 ;   0 - 3, default -1 ( STP-240 - ISA module )
pci_bus_no = -1  ; PCI bus on which STP-340 modules will be looking for
                 ;   0 - 255, default -1 ( all PCI busses will be scanned)
```

The next two sections contain the specific parameters of the motors:

```
[motor1]
; motor 1 config data - times in multiples of 2 us
tpeak =  1000      ;   1 - 65535 ,       default=150
tmin =    1500     ;   tpeak - 65535 ( must be >= 50 ), default = 250
tmax =    10000    ;   tmin  - 65535 , default=1500
tdelta =  250      ;   1 – 65535,       default=250
overlap =     0    ; stepping mode:
                   ;   0 - full step mode (default)
                   ;   1 - half step mode
phases = 4         ;  3 - 8  phases ,default=3
min_lim = 0        ; min_limit_switch :
                   ;    0 - not present (default) ,
                   ;   -1 - present and used for stepping,     logic state of switch at limit =0
                   ;   -2 - present but not used for stepping, logic state of switch at limit =0
                   ;    1 - present and used for stepping,     logic state of switch at limit =1
                   ;    2 - present but not used for stepping, logic state of switch at limit =1
max_lim = 0        ; max_limit_switch : possible values the same as for min_lim
                   ; (default = 0 - not present ) ,
device = 1         ; connection with device (default = 0 (none), 1-device1, 2-device2)
steps = 100        ; number of steps per one turn (default =200)
                   ;  be careful- sum of phases for both motors must be <= 8
                   ;  otherwise unused motor will be switched off
                   ; - device numbers should be different for both motors ,if
                   ;  they are both active (phases not equal 2)
[motor2]
  .
  .
```

The last 2 sections contain the specific parameters of the devices connected to the motors:

```
[device1]
scale = Wavelength[nm]   ; this text can be used as a scale description
                         ;   (max 16 char.)
unit = nm                ; nanometers - units of displayed values(max 3 char.)
min = 300.               ; min position  (in units described above)
max = 800.               ; max position  (in units described above)
turn_incr = 10.          ; increment per motor turn  (in units described above)

[device2]
   .
   .
   .
```

The Parameters must be included in the configuration file only if they are to be set to a value different from the default value. The name (and the path) of the configuration file is an input parameter of the library function STP_init. Therefore the user is free to have more configuration files, one for each measurement setup.

Description of the STP Library Functions

Important Note: All library functions (except STP_init, STP_get_mode, STP_get_module_info) return an error when they are called before an initialisation by STP_init was not done before. All functions, which have a motor number as an input parameter, will return an error if the specified motor is not used - see use_mot setting in base section in STP configuration file.

Functions listed below 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 'STP', for example, STPstd_init it is _stdcall version of STP_init.
Description and behaviour of these functions are identical to the functions from the first (default) set – the only difference is calling convention.

---

short CVICDECL STP_init (char far config_file[]);

Input parameters: config_file: pointer to string containing configuration file name and path

Return value: returns error code, 0 - success, <0 - error

STP_init initialises the library-internal structures using parameters from the configuration file config_file (recommended extension .cfg) and initialises the internal registers of the STP module for used motors.

The STP-240 module, which will be initialised, is defined by 'baseadr' parameter from ini_file.
The STP-340 module, which will be initialised, is defined by 'pci_bus_no' and 'pci_card_no' parameter from ini_file.
'pci_bus_no' defines which PCI bus with STP-340 modules will be initialised:

- value 0 – 255 defines specific bus number (from the range: 0 to number of PCI busses with STP modules) (if 'pci_bus_no' is greater than number of PCI busses with STP modules, it is rounded to the number of busses –1)
- value –1 means that that all PCI busses will be scanned

'pci_card_no' defines the number of STP-340 module on PCI bus to be initialised:

- value 0 – 3 defines one specific module (if 'pci_card_no' is greater than number of STP modules on PCI bus, it is rounded to the number of modules –1)
- value –1 means that that the function will not try to initialise any STP-340 modules on PCI bus but only STP-240 module (if any exists)

The STP-340 module will be initialised, but only when it is not in use (locked) by other application.
After successful initialisation the STP-340 module is locked to prevent that other application can access it.
If, for some reasons, the STP-340 module which was locked must be initialised, it can be done using the function STP_set_mode with the parameter 'force_use' = 1.

During initialisation of the STP-340 module checksum of the internal EEPROM is tested.

The user should check initialisation status of the STP-340 module he wants to use by calling SPC_get_module_info function.

The possible values of initialisation result code are shown below (see also stp_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 in use (locked) - cannot initialise

---

short CVICDECL STP_calibrate(short mot_no, float far * pos);

Input parameters: mot_no : 0, 1 - number of motor
*pos : pointer to variable containing position value

Return value: returns error code, 0 - success, <0 - error ,

STP_calibrate sets the current position for the device connected to motor ‘mot_no’. The ‘pos’ value is recalculated according to the device limits and set to the actual current position. The used device has to be calibrated before stepping functions are used and after a call to the function ‘STP_cancel_stepping’.

---

short CVICDECL STP_get_position(short mot_no, float far *pos)

Input parameters: mot_no: motor number (0 - 1)
*pos: pointer to the position variable to be set

Return value: 0 - on success,
<0 - on error

STP_get_position sets ‘pos’ according to the current position of the device connected to motor ‘mot_no’.

---

short CVICDECL STP_set_position(short mot_no, float pos);

Input parameters: mot_no: motor number (0 - 1)
pos: new position value

Return value: 0 - on success,
<0 - on error

STP_set_position calculates the direction and the number of steps required to achieve the position ‘pos’ by the device connected to motor ‘mot_no’ and starts the stepping action. The function returns an error if device was not calibrated. After a call of STP_set_position, use the function STP_test_if_stepping to check whether the stepping has been finished.

---

short CVICDECL STP_step(short mot_no, long steps);

Input parameters: mot_no: motor number (0 - 1)
steps: number of steps

Return value: 0 - success,
<0 - error

STP_step starts the stepping action for motor ‘mot_no’. Motor ‘mot_no’ will make 'steps' number of steps in the proper direction. The function returns an error, if the device connected to motor ‘mot_no’ was not calibrated or the number steps is too large (i.e. if the position of the device connected to motor ‘mot_no’ would go through its limits min limit or max limit). After a call of STP_step, use the function STP_test_if_stepping to check whether stepping has been finished.

---

short CVICDECL STP_step_to_limit(short mot_no, short direction);

Input parameters:	mot_no:	motor number (0 - 1)
	direction:	0 - min limit, 1 - max limit
Return value:	0- success,	<0 - error

STP_step_to_limit causes that motor ‘mot_no’ goes to its limit position (detected by the limit switch).The function returns an error if the motor has no required limit switch (please see min_lim or max_lim setting in the motor0(1) section of the configuration file).

Use the function STP_test_if_stepping to check whether the action has been completed. STP_step_to_limit uses the full speed of the motor. Thus, it can happen that the motor does not stop exactly at the switch position. Use the STP_off_limit function to bring the motor slowly back to the exact switching position.

---

short CVICDECL STP_step_off_limit(short mot_no, short direction);

Input parameters:	mot_no:	motor number (0 - 1)
	direction:	0 - min limit, 1 - max limit
Return value:	0 - success, <0 - error	

The STP_off_limit function is used to bring the motor slowly back to the exact switching position after it has run beyond the limit position. The motor stops when the limit switch switches to the ‘off’ state .

The action starts only if the limit switch is ‘on’ (i.e. if the motor ran outside the limits during the last stepping action). The function returns an error if the motor has no limit switch (see min_lim or max_lim setting in the motor0(1) section of the configuration file).

Use the function STP_test_if_stepping to check whether the motor action has been completed.

---

short CVICDECL STP_test_if_stepping(short mot_no, float far * current_wl);

Input parameters: mot_no: motor number (0 - 1)
*current_wl: pointer to the position variable to be set

Return value: 0 - stepping completed, 1 - stepping in progress, <0 - error

STP_test_if_stepping checks whether the stepping for motor ‘mot_no’ is completed and sets ‘pos’ according to the current position of the device connected to the motor ‘mot_no’. If there were errors during the stepping action the procedure returns an error code.

---

short CVICDECL STP_cancel_stepping(short mot_no, float far * current_wl);

Input parameters: mot_no: motor number (0 - 1)
*current_wl: pointer to current position variable to be set

Return value: 0 - success, <0 - error

STP_cancel_stepping breaks current stepping action (if any) performed by motor mot_no and sets current_wl according to the current position of the device connected to the motor mot_no. Remember that after this call current position can be not defined - STP_calibrate call is required to execute further stepping actions on this device.

---

short CVICDECL STP_get_parameters(STPdata far * data);

Input parameters: *data: pointer to the structure to be set with actual STP library parameters

Return value: 0 - success, <0 - error

STP_get_parameters fills the 'data' structure with the actual STP library parameters .

After calling the STP_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 **STP_get_parameters** is provided. This function transfers the parameter values from the internal DLL structures into a structure of the type STPdata (see stp_def.h) which has to be defined by the user.

---

short CVICDECL STP_activate_motor(short mot_no, short active);

Input parameters: mot_no: motor number (0 - 1)
active: 0 - switch off, 1- switch on the motor

Return value: 0 - success, <0 - error

STP_activate_motor switches on or off motor ‘mot_no’. Further stepping actions will be not possible as long as the motor is switched off.

---

short CVICDECL STP_test_if_motor_active(short mot_no, short far *active);

Input parameters: mot_no: motor number (0 - 1)
*active: pointer to the variable to be set (0 - motor switched off ,1- switched on)

Return value: 0 - success, <0 - error

STP_test_if_motor_active returns information about the actual state of the motor ‘mot_no’. Stepping actions are not possible as long as the motor is switched off.

---

short CVICDECL STP_get_switch_state(short mot_no, short far *state);

Input parameters: mot_no: motor number (0 - 1)
*state: pointer to the variable to be set (bit 0 - min_lim, bit 1 - max_lim switch, bit = 0 switch is off, bit = 1 switch is on)

Return value: 0 - success, <0 - error

STP_get_switch_state returns the actual state of the switches of motor 'mot_no'. The function returns an error, if the motor has no limit switches. Switch 'on' means that it has the logical 'on' state defined in the configuration file (see min_lim or max_lim setting in the motor0(1) section of the configuration file).

---

short CVICDECL **STP_get_module_info** (short mod_no , STPModInfo * 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 stp_def.h)

Description:
After calling the STP_init function (see above) the STPModInfo internal structures for all 4 modules are filled. This function transfers the contents of the internal structure of the DLL into a structure of the type STPModInfo (see stp_def.h) which has to be defined by the user. The parameters included in this structure are described below.

short module_type	module type : 240, 340, -1 – unknown type (module doesn't exist)
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 address of STP-340 resources
char serial_no[12]	module serial number

---

short CVICDECL **STP_get_eeprom_data**(STP_EEP_Data *eep_data);

Input parameters:

*eep_data	pointer to result structure

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

The structure "eep_data" is filled with the contents of the EEPROM of the STP-340 module. The EEPROM contains the production data of the module. The structure "STP_EEP_Data" is defined in the file stp_def.h.
The function is not supported for STP-240 module.

---

short CVICDECL **STP_write_eeprom_data**(unsigned short write_enable,
STP_EEP_Data *eep_data);

Input parameters:

write_enable	write enable password
*eep_data	pointer to result structure
Return value:	0 no errors, <0 error code (see stp_def.h)

The function is used to write data to the EEPROM of an STP module 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.
The function is not supported for STP-240 module.

---

short CVICDECL STP_get_id (short far *id);

Input parameters: *id - pointer to result variable

Return value: 0 - success, <0 - error

STP_get_id sets 'id' according to the identification code of the module STP-240.

The function is not supported for STP-340 modules – use STP_get_module_info to get information about the module.

---

short CVICDECL STP_test_id (void);

Input parameters: none

Return value: 0 - success (ID correct), <0 - error

STP_test_id reads the identification code of the module STP-240 and compares it to the correct value.

The function is not supported for STP-340 modules – use STP_get_module_info to get information about the module.

---

short CVICDECL STP_set_parameter(short mot_no, short reg_no, short val)

Input parameters: mot_no: motor number (0 - 1)
reg_no: register number (1 - 11)
val : value written to register reg_no

Return value: 0 - success, <0 - error

Sets the register 'reg_no' of motor 'mot_no' to the value 'val'. STP_set_parameter is a low level procedure and not intended for normal use.

---

short CVICDECL STP_get_parameter(short mot_no, short reg_no, short far *val)

Input parameters: mot_no: motor number (0 - 1)
reg_no: register number (1 - 15)
*val: pointer to variable to set

Return value: 0 - success, <0 - error

Sets 'val' with the current contents of the register 'reg_no' of motor 'mot_no'. STP_get_parameter is low level procedure and not intended for normal use.

---

short CVICDECL STP_get_status(unsigned short far *val);

Input parameters: *val : pointer to variable to set

Return value: 0 - success, <0 - error

STP_get_status sets ‘val’ with the current contents of the STP status register. Status bits 0 - 7 contain an error code (if any) set by STP microcontroller.

---

short CVICDECL STP_set_mode (short mode, short force_use, short mod_no);

Input parameters:

mode:	mode of DLL operation
force_use	force using the module if they are locked (in use)
mod_no	module number on PCI bus which will be affected

Return value: 0 - success, <0 - error

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 is not intended to normal use.
It is used to switch the DLL to the simulation mode if hardware errors occur during the initialisation.
When the Hardware Mode is requested for one of 4 possible STP-340 modules:

-the ‚mod_no‘ 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 and when it exists.

-other modules (if any) are unlocked and can be used further.

When one of the simulation modes is requested for one of 4 possible modules:

- the ‚mod_no‘ module is initialised (if it wasn’t) with the initial parameters set (from ini_file).

-other modules are unused.

Errors during the module initialisation can cause that the module is excluded from use.
Use the function STP_get_module_info to check which modules are correctly initialised and can be use further.
Use the function STP_get_mode to check which mode is actually set. Possible ‘mode’ values are defined in the stp_def.h file.

---

short CVICDECL STP_get_mode (void);

Input parameters: none

Return value: current mode of DLL operation, <0 - error

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

```
#define STP_HARD        0       /* hardware mode */
#define STP_SIMUL240    240     /* simulation mode of STP-240 */
#define STP_SIMUL340    340     /* simulation mode of STP-340 */
```

---

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

Input parameters:

error_id	STP DLL error id (0 – number of STP errors-1) (see stp_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 STP DLL error with id equal 'error_id'. Up to 'max_length characters will be copied.
For errors during stepping (E_STEP_ERR) or writing parameters to STP module (E_MOD_ERR) status and motor's error codes are included.
Possible 'error_id' values are defined in the stp_def.h file.