



MS 64E User Manual

C-842 API

Application Programming Interface

Windows 95/98

Windows NT/2000

V 5.00

***Function Reference
&
Programming Instructions***

This Document is valid for the Products:

C-842.xx DC-Motor Controller

Release: 5.00
Release Date: 2002-02-28

© **PI** (Physik Instrumente) **GmbH & Co.**
D-76228 Karlsruhe, Germany FAX: (+49)721-4846-100

Email:info@pi.ws

www.pi.ws

Table of Contents :

0.	MANUFACTURER DECLARATIONS	3
0.1.1.	Disclaimers	3
1.	C-842 LIBRARY SUPPORT	4
1.1.1.	Windows 95/98.....	4
1.1.2.	Windows NT.....	4
2.	LIBRARY FOR WINDOWS 95/98	5
2.1.	QWLW500.DLL FUNCTION OVERVIEW:	6
2.2.	TYPES, CONSTANTS AND STRUCTURES	7
2.2.1.	For Delphi Programmers	8
2.2.2.	For MSVC++ Programmers.....	9
2.3.	QFLW500 LIBRARY FUNCTION REFERENCE :	11
2.3.1.	General Functions for C-842 Access	11
2.3.2.	System Initializing Functions	12
2.3.3.	Moving Functions.....	14
2.3.4.	Functions to Set Values.....	17
2.3.5.	Functions to Read Values.....	18
2.3.6.	Low Level Functions.....	20
3.	USING C-842 WITH WINDOWSNT AND WINDOWS2000	22
3.1.	HOW CAN I WORK UNDER NT?	22
3.2.	INSTALLING THE C-842 NT DRIVER	22
3.3.	NT-DRIVER INSTALLATION	23
4.	NT-DLL LIBRARY (VERSION 3.50)	23
4.1.	C842NT35.DLL FUNCTION LIST :	23
4.2.	BUILDING APPLICATIONS.....	24
4.3.	HEADER FILE	24
4.4.	FUNCTION REFERENCE	26
4.4.1.	NT-driver Functions.....	26
4.4.2.	Initializing	26
4.4.3.	General Communication.....	27
4.4.4.	Moving Functions.....	30
4.4.5.	Digital I/O	31
4.4.6.	Status, Limit and Reference Signal Requests.....	31
5.	LIST OF QFL COMMANDS.....	35

0. Manufacturer Declarations

The contents of the manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Physik Instrumente (PI).

Physik Instrumente (PI) assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual.

0.1.1. Disclaimers

The software described in this manual is distributed as it is.

Physik Instrumente (PI) expressly disclaims any and all warranties including any implied warranty of merchantability and fitness for a particular purpose. Physik Instrumente (PI) does not warrant, guarantee, or make any obligations regarding the use or the results of the use of this software, in terms of correctness, accuracy, reliability or otherwise, and does not warrant that operation of the software will be uninterrupted or error-free.

© Copyright 2002 by Physik Instrumente GmbH

Release: 5.00

File:MS64E500.doc, 145920 Bytes

PDate: 01.03.02 15:18

1. C-842 Library Support

The functionality of the C-842 motor controller is available for windows programmers by *Dynamic Link Libraries* offering a variety of functions for board handling and motion control.

Since the C-842 controller card is an ISA bus card, programmers have to deal with the fact, that Windows NT does not offer direct access to hardware components. So different approaches are required for both developing platforms:

1.1.1. Windows 95/98

The dynamic link library offers functions accessing directly the I/O port space of the ISA bus. No special driver is required and the library can be used from all 32 bit developing environments like VC++, Delphi, VB and others.

1.1.2. Windows NT

Windows NT does not allow direct hardware access. for this reason a NT kernel null driver is available that has to installed in the NT system prior using the NT-DLL. Unlike the Windows 95 DLL this library calls the driver functions in order to access the C-842 card. The functionality is almost the same but functions may differ in details depending on the evolution level of the file.

2. Library for Windows 95/98

QFLW500.DLL

Version 5.00, February 2001

Windows DLL for C-842 DC-Motor Controllers

Frequently used commands like *Move Absolute*, *Move Relative* or position queries like *Tell Position* can be called by library functions. Other commands like *Set P-gain* or *Tell following error* can be accessed via the TRANSLATE and/or EXECUTE function. The actual command then is passed as a parameter to the library function.

More than 100 commands are defined as constants to be accessed with the EXECUTE function. Packaging the command in a string in conjunction with axis identifier and parameter is the other way to send a command via the TRANSLATE function.

Examples: `execute(3,MR,45000,report);`
`translate("3MR45000",report);`

Both functions perform the same action, moving motor#3 relative for 45000 counts.

In the same manner, the complete QMove commands set of more than 100 commands can be called.

The command transferred is executed immediately when received.

*For detailed description on the commands implemented refer to **Programming Manuals MS45E and MS46E.***

2.1. QWLW500.DLL Function Overview:

<u>Ordinal</u>	<u>Entry Point</u>	<u>Name</u>
0045	00000000	AutoFindEdge
0011	0001f594	Axis_Installed
0017	00000000	Board_Installed
0015	0001f5d8	InitAxis
0013	00023cb4	InitBoard
0016	0001f5f8	InitStage
001f	0001f0bc	Init_LS
0019	0001f618	MoveA
0031	00000000	MoveA_12
0032	0001fbcc	MoveA_123
001b	00000000	MoveR
0033	000223c0	MoveR_12
0034	00000000	MoveR_123
000f	00023bfc	Set_BaseAddress
003b	00000000	VectorA
003c	0001f670	VectorR
001d	00000000	WaitStop
0029	0001f6a4	autodetect
0023	00000000	axis_select
0009	00021c00	execute
0027	00000000	getQMC
0021	00021c9c	get_BoardStatus
002f	00021cf4	get_DIN
0004	00023bb0	get_ErrorStatus
002e	00000000	get_Limit
004b	0001f7f0	get_LimitAxis
0003	00000000	get_PosErr
0006	0001f948	get_RefSignal
0001	00000000	get_pos
0007	0001faf8	get_pos4
0005	00000000	get_stat
002d	0002230c	get_v
000d	00000000	moving
0025	0001f0a8	setQMC
0047	00000000	set_Amplifier
0030	0001f120	set_DOUT
004f	00000000	set_acc
0049	0001f180	set_pid
002b	00000000	set_pos
004d	0001f35c	set_vel
000b	00000000	translate

2.2. Types, Constants and Structures

Types, constants and functions are defined in a definition text files, that can be easily used by C or Pascal programmers.

File: QFLW_CmdDef.PAS

Types used by the QFLW library:

```
Tstr = string[5];

Tcmd = record
    code    : byte;
    qclass  : byte;
    lab     : Tstr;
end;

TQ_long    = array[1..4] of longint;
TQ_integer = array[1..4] of integer;
```

All commands (cmd) used with the **execute** procedure are defines as constants of the type TCmd. Names of the variables comply with command names used in the QMove or WinMove program and are documented in the MS 45E operating manual.

Example of a command constant of TCmd type, here SA(Set Acceleration)

```
SA : Tcmd = (code:$12; qclass:wr_U1616; lab:'SA:'); (* SET_ACC    *)
```

In the same manner more than 100 QFL-commands are defined. For further information see the definition QFLW500.DEF file.

General structure of a program using the QFLW library:

Set_BaseAddress(\$214)	Define board address (optional if not default address hex210)
MaxAxis := Axis_Installed	Detects max axis number and adjusts sample rate to 400 µs
status := InitBoard(9)	Sets the board function register
status := autodetect()	Reads and defines the limit switch configuration
InitAxis(1)	Sets Motor #1 to default parameters
InitAxis(2)	Sets Motor #2 to default parameters
translate('1SV80000',report)	main program
translate('2SV120000',report)	
translate('1SA250',report)	
translate('2SA190',report)	
translate('1DP120',report)	
translate('2DP250',report)	
translate('1DI12',report)	

```

translate('2DI12',report)
translate('1DD230',report)
translate('1DD350',report)
MoveA(1,5000)    //move command
MoveA(2,5000)    //move command
...
...
end.

```

2.2.1. For Delphi Programmers

Declaration of functions: (Pascal style)

See file QFLW_CmdDef.PAS

```

(*****
(* Function Declarations :
(*****
// Case sensitive writing required !!
// ExtLib = DLL_FileName

function  get_pos(axis:byte):integer; stdcall external ExtLib; //2
function  get_PosErr(axis:byte):integer; stdcall external ExtLib; //4
function  get_ErrorStatus:integer; stdcall external ExtLib; //5
function  get_stat(axis:byte):integer; stdcall external ExtLib; //6
function  get_RefSignal(axis:byte):word; stdcall external ExtLib; //7
procedure execute(axis:byte;command:TCmd;para:integer;report:Pchar);
           stdcall external ExtLib; //10
procedure translate(cmd:Pchar;rep: Pchar); stdcall external ExtLib; //12
function  moving(axis:byte):boolean; stdcall external ExtLib; //14
procedure Set_BaseAddress(address:word); stdcall external ExtLib; //16
function  Axis_Installed:byte; stdcall external ExtLib; //18
function  InitBoard(mode:byte):integer; stdcall external ExtLib; //20
procedure InitAxis(axis:byte); stdcall external ExtLib; //22
procedure InitStage(axis,stage:byte); stdcall external ExtLib; //23
function  Board_Installed:boolean; stdcall external ExtLib; //24
procedure MoveA(axis:byte; position:integer); stdcall external ExtLib; //26
procedure MoveR(axis:byte; counts:integer); stdcall external ExtLib; // 28
function  WaitStop(axis:byte):integer; stdcall external ExtLib; //30
procedure Init_LS(drive1,drive2,drive3,drive4:byte); stdcall external ExtLib; //32
function  get_BoardStatus:word; stdcall external ExtLib; //34
procedure axis_select(axis:byte); stdcall external ExtLib; //36
procedure setQMC(cmd:byte; para:integer); stdcall external ExtLib; //38
function  getQMC(cmd:byte):integer; stdcall external ExtLib; //40
function  autodetect:integer; stdcall external ExtLib; //42
procedure set_pos(axis:byte; newpos:longint); stdcall external ExtLib; //44
function  get_v(axis:byte):integer; stdcall external ExtLib; //46

procedure AutoFindEdge(axis:byte; mode:byte); stdcall external ExtLib; //70
function  get_DIN(byte; stdcall external ExtLib;
procedure set_DOUT(digout:byte); stdcall external ExtLib;
function  set_Amplifier(mode:byte):integer; stdcall external ExtLib;
function  get_Limit:integer; stdcall external ExtLib;
function  get_LimitAxis(axis:byte):integer; stdcall external ExtLib;
procedure set_pid(axis:byte; pterm,iterm,dterm:integer); stdcall external ExtLib;
procedure set_vel(axis:byte; velocity:integer); stdcall external ExtLib;
procedure set_acc(axis:byte; acceleration:integer); stdcall external ExtLib;
procedure VectorA(axis_1, axis_2:byte; pos_1, pos_2, vv,va:integer);
           stdcall external ExtLib; //60
procedure VectorR(axis_1,axis_2:byte; move_1,move_2,vv,va:integer);
           stdcall external ExtLib; //61
procedure MoveA_12(p1,p2:integer); stdcall external ExtLib; // 50
procedure MoveA_123(p1,p2,p3:integer); stdcall external ExtLib; // 51
procedure MoveR_12(d1,d2:integer); stdcall external ExtLib; // 52
procedure MoveR_123(d1,d2,d3:integer); stdcall external ExtLib; // 53

```


2.2.2. For Microsoft VC++ Programmers

Programming examples for VC using:

QFLW500.DLL, QFL500_VC.LIB

Declaration of functions: (C style)

```
// File : QFLW500.H
// Date : 2002-03-01

#ifdef __cplusplus
    extern "C" {
#endif

// This is the header to use QFLW500.DLL functions with VC

// To use the DLL, include the static import
// library QFLW500_VC.LIB into the project

// definitions:
#define CMD_NOPARA    1

#define RD_NOFRAC     2
#define RD_U1616      4
#define RD_S1616      5
#define RD_U0032      6

#define WR_NOFRAC     12
#define WR_U1616      14
#define WR_S1616      15
#define WR_U0032      16
#define EXDEF         20

// types & structures:
typedef unsigned char BYTE;
typedef unsigned int  WORD;

typedef struct struct_CMD{
    BYTE code;
    BYTE cmd_group;
    char lab[6];
}TCMD;

typedef struct struct_Q_long{
    long nPosition[4];
}TQ_long;

// function prototypes :

long  __stdcall get_pos(BYTE axis);
long  __stdcall get_PosErr(BYTE axis);
long  __stdcall get_v(BYTE axis);
int   __stdcall get_stat(BYTE axis);
void  __stdcall get_pos4(TQ_long *pos4);
WORD  __stdcall get_BoardStatus(void);
BYTE  __stdcall get_DIN(void);
int   __stdcall get_ErrorStatus(void);
int   __stdcall get_Limit(void);
int   __stdcall get_LimitAxis(BYTE axis);
WORD  __stdcall get_RefSignal(BYTE axis);

void  __stdcall execute(BYTE axis, TCMD command, long para, char *report);
void  __stdcall translate(char *comstr, char *report);
BOOL  __stdcall moving(BYTE axis);
void  __stdcall Set_BaseAddress(unsigned short address);
BYTE  __stdcall Axis_Installed(void);
int   __stdcall InitBoard(BYTE mode);
void  __stdcall InitAxis(BYTE axis);
```

```
void __stdcall InitStage(BYTE axis, BYTE stagetype);
BOOL __stdcall Board_Installed(void);
void __stdcall MoveA(BYTE axis, long nPosition );
void __stdcall MoveR(BYTE axis, long nCounts );
void __stdcall VectorA(BYTE axis1, BYTE axis2, long p1, long p2, long vv, long va);
void __stdcall VectorR(BYTE axis1, BYTE axis2, long incl1, long inc2, long vv, long
va);
int __stdcall WaitStop(BYTE axis);
void __stdcall Init_LS(BYTE axis1, BYTE axis2, BYTE axis3, BYTE axis4);
//WORD __stdcall StatusBoard(void);
void __stdcall axis_select(BYTE axis);
void __stdcall setQMC(BYTE command, long parameter);
long __stdcall getQMC(BYTE command);
int __stdcall autodetect(void);
void __stdcall set_pos(BYTE axis, long newpos);
int __stdcall set_Amplifier(BYTE mode);
void __stdcall set_DOUT(BYTE digout);
void __stdcall set_acc(BYTE axis, int acceleration);
void __stdcall set_pid(BYTE axis, int pterm, int iterm, int dterm);
void __stdcall set_vel(BYTE axis, int velocity);

void __stdcall AutoFindEdge(BYTE axis, BYTE mode);

void __stdcall MoveA_12(long pos_1, long pos_2);
void __stdcall MoveA_123(long pos_1, long pos_2, long pos_3);
void __stdcall MoveR_12(long shift_1, long shift_2);
void __stdcall MoveR_123(long shift_1, long shift_2, long shift_3);
```

2.3. QFLW500 Library Function Reference :

Function reference uses C syntax.

Definitions:

BYTE represents unsigned 8 bit (char)

WORD represents unsigned 16 bit /(short)

2.3.1. General Functions for C-842 Access

```
void execute(BYTE axis, Tcmd SubCommand, long parameter, char *report)
```

This function calls one of the more than 100 sub-functions of the QFL library interface. The sub command of the type TCmd has a structure defined in the header or in the Pascal definition file. The *report parameter points to a 128 bit character array to hold a PChar type.

Ordinal index: 10

Input: *axis* as char (byte)
 command as Tcmd
 parameter as long
 report as pointer

Return: none

```
void translate(char *command, char *report)
```

This function can be used instead of the execute function if it is more convenient to work with strings.

Ordinal index: 12

input: *command*:
 Pointer to character array (PChar, zero terminated string), holds command string including axis identifier, command and parameter. Format example "1MR5000" with 1(motor identifier), MR(command) and 5000(parameter).
 report:
 Pointer to character array (PChar), supplied by the calling function. The calling program should define a character array of 256 characters to take the report message.

See also "Low Level Functions" for genuine board access.

2.3.2. System Initializing Functions

void Set_BaseAddress(unsigned short address)

This procedure allows to define the hardware I/O address selected by the DIP switches on the C-842 board. Default setting is 0x210.

If the C-842 board is used with the default address setting (0x210), this function is not required.

Input: address as unsigned short (word, 16 bit)

Return: none

BOOL Board_Installed(void)

This function returns a boolean value depending on a card is installed or not. The function looks for the board at the address defined by Set_BaseAddress function.

Ordinal index: 24

Input: none

Return: boolean (0 or 1), 1=installed, 0=not installed

BYTE Axis_Installed(void)

This function returns the number of motor axes supported by the processor. There are 2-axes and 4-axes version available (C-842.20 and C-842.40).

Ordinal index: 18

Input : none

Return: number of axes as BYTE

int InitBoard (BYTE mode)

This function initializes the C-842 and returns the board status. It has to be sent after the C-842 is powered up to enable the amplifiers and to define the position encoder functions. The mode value is transferred to the board function register. For normal operation set mode to 9.

Following actions are performed:

- Board reset
- Set board function register to mode value
- Set output to DAC16 (SOH) (amplifier to analog)
- Set limit sense register to 0x0FF
- Set capture to home (SCH)

Ordinal index: 20

Input: mode as BYTE

Return: board status register as integer

void Init_LS (BYTE axis1, BYTE axis2, BYTE axis3, BYTE axis4)

This function sets the Limit_Sense Register.

Note: It is more convenient to use the autodetect() function instead

After the system is powered up, the limit switch wiring and logical levels have to be defined. If no limit switches are connected, axis parameters have to be set to 0 for the corresponding axis.

If limit switches are wired compatible with PI standard stages, the corresponding Byte has to be set to 1.

Example: One PI-compatible stage is connected to axis #2. All other axes are not connected or have motors without limit switch connection, call Init_LS(0,1,0,0)

Note: Init_LS() overwrites autodetect() settings.

Ordinal index: 32
 Input: mode as BYTE
 Return: status as integer

int autodetect(void)

This function reads the current limit switch lines levels of all axes and defines the reading as normal state (no limit switch tripped). Any change of that status is interpreted as a limit switch hit event, triggered by the stage.

Note: When calling autodetect(), no limit switch must be hit.

Note: autodetect() overwrites prior Init_LS settings.

Actions performed:

- reads the current limit switch state
- analyses whether one limit switch is hit and returning error code in that case
- sets the limit_sense state of the board according to the values found
- sets LimitMode variable to limit_sense value

Ordinal index: 42
 Input: none
 Return: value of detected LS configuration (e.g. 255 if no LS connected)
 -1 if error (LS was hit)

void InitAxis(BYTE axis)

This function initializes the specified axis and sets default values. The settings may not match your specific application, so it might be required to set individual servo parameters, velocities and accelerations.

Actions performed: SET_KP, Set_VEL, Set_ACC, AUTO_STOP_OFF, Upate_Filter, CLR_STATUS

Ordinal index: 22
 Input: axis as BYTE
 Return: none

void InitStage(BYTE axis, BYTE stagetype)

This function can be used alternatively to the InitAxis() function. InitStage() allows to configure an axis using different sets of motion parameters, velocities and accelerations. The stagetype parameter selects one specific set.

stagetype : 1 : DC-Mikes like M-222.xx
 2 : M-227 DC Mikes or equivalent stages
 3 : C-136 motors or "normal" gear head drives
 4 : M-5x1.PD linear stages
 5 : M-5x1.DD linear stages
 15 : M-5x1.DD linear stages with brake ON

void AutoFindEdge(BYTE axis, BYTE mode)

This function can be used with stages having automatic reference heading option, e.g. M-500 or M-400 series. After this function is called, the stage starts in the right direction towards the origin point.

If the mode parameter equals 1, the stage starts in positive direction with reference level high, other parameter values start in negative direction.

Actions performed: After analyzing which side the stage is located, the FEP or FEN commands are called.

Ordinal index: 70
 Input: axis as BYTE,
 mode as BYTE
 Return: none

2.3.3. Moving Functions

void MoveA(BYTE axis, long position)

Move Absolute: This function starts one or all motors to move to an absolute position. If the axis identifier is 0, all axes available are moved to the same position.

After the function is called, the motion is started and the function is left immediately. If the next call of MoveA occurs before the motor(s) have terminated their moves, the old target is replaced by the new one and the motors immediately head towards their new targets, even if the previous target(s) were not yet reached. In order to synchronize motions and to start the next movement not before the first target is reached, the WaitStop command has to be called before the next move command is executed.

Example 1 :At first, motor#1 will run to position 5000 and stop there, then run to zero position:

```
MoveA(1,5000)
WaitStop(1)
MoveA(1,0)
```

Example 2: Assume the motor is at zero position. Without the WaitStop command the first move to 85000 will not be performed and the motor will already stop at position 1000:

MoveA(1,85000)**MoveA(1,1000)**

Ordinal index: 26

Input: axis as BYTE, position as long

Return: none

void MoveA_12(long position1, long position2)

Move Absolute XY with synchronized start. This function starts axis 1 and 2 with individual velocities at the same time.

After the function is called, the 2-axis motion is started and the function is left immediately. The general behavior is similar like the MoveA function.

Example : Set velocities of axes 1 and 2 using translate or execute function and then start simultaneous motion:

```
translate("1SV20000",report)
translate("2SV80000",report)
MoveA_12(25000,75000)
```

Ordinal index: 50

Input: position1 as long, position 2 as long

Return: none

void MoveA_123(long position1, long position2, long position3)

Description same as MoveA_12()

void MoveR(BYTE axis, long counts)

This function performs a relative move of a given number of steps (counts). The specified count number is added to the current position and the result is commanded as the new target.

After the function called, the motion is started and the function is left immediately. If the next call of MoveR occurs before the motor(s) have terminated their moves, a continuous motion to the new target is performed. In order to synchronize motions and to start the next movement not before the last move is terminated, the WaitStop command has to be used.

If the axis identifier is 0, all axis available are moved by the same number of counts.

Ordinal index: 28

Input: axis as BYTE, counts as long

Return: none

void MoveR_12(long position1, long position2)

Move Relative XY with synchronized start. This function starts axis 1 and 2 with individual velocities at the same time.

After the function is called, the 2-axis motion is started and the function is left immediately. The general behavior is similar like the MoveR function.

Example : Set velocities of axes 1 and 2 and start simultaneous motion:

```
translate("1SV20000",report)
translate("2SV80000",report)
MoveR_12(25000,75000)
```

Ordinal index: 52
 Input: position1 as long, position 2 as long
 Return: none

void MoveR_123(long position1, long position2, long position3)

Description same as MoveR_12()

void VectorA(BYTE axis1, BYTE axis2, long p1, long p2, long vv, long va)

This functions performs a linear interpolation and starts axis1 and axis2 at the same time towards the coordinate point (p1/p2) with the vector velocity vv and the vector acceleration va.

Example : Vector move of axes #1 and #2 from the current position to the absolute point 12000/30000 with the vector velocity of 6500 counts/s and the vector acceleration of 250 :

VectorA(1,2,12000,30000,6500,250)

Ordinal index: 60
 Input: axes as BYTE, positions and velocity as long
 Return: none

void VectorR(BYTE axis1, BYTE axis2, long shift1, long shift2, long vv, long va)

This functions performs a linear interpolation and starts axis1 and axis2 at the same point in time for a vector with the components shift1/shift2 with the vector velocity vv and the vector acceleration va.

Example : Relative Vector move of axes #3 and #4 relative from the current position for 15000/19500 counts with the vector velocity of 26500 counts/s and the vector acceleration of 400:

VectorR(3,4,15000,19500,26000,400)

Ordinal index: 61
 Input: axes as BYTE, shifts and velocity as long
 Return: none

long WaitStop(BYTE axis)

The WaitStop function controls the dynamic position of the specified axis and returns if the motor has reached the target. It can be used to synchronize multiple movements.

Ordinal index: 30

Parameter: axis as BYTE
Return: none

BOOL moving(BYTE axis)

This function returns true if the trajectory of the specified axis has not yet reached the target position. The function reads the on-target flag of the condition register.

Ordinal index: 14
Parameter: axis as BYTE
Return: boolean

2.3.4. Functions to Set Values

void set_pos(BYTE axis, long newpos)

This function redefines the current position.

Ordinal index: 44
Input: axis BYTE,
newpos as long
Return: none

int set_Amplifier(BYTE mode)

This function sets the amplifier mode for all axes on the board. The amplifier mode is global for the board.

Ordinal index: 72
Input: mode as BYTE,
return: errorcode as integer, 0=no error, 1= error

void set_pid(BYTE axis, int p-term, int i-term, int d-term)

This function sets p-i-d parameters

Ordinal index: 74
Input: axis BYTE,
p,i,d valus as integer
Return: none

void set_vel(BYTE axis, int velocity)

This function sets the maximum velocity

Ordinal index: 78
Input: axis BYTE,
velocity as integer
Return: none

void set_accl(BYTE axis, int acceleration)

This function sets the acceleration in trapezoidal mode

Ordinal index: 80
Input: axis BYTE,
acceleration as integer
Return: none

void set_DOUT(BYTE digout)

This function sets the digital output channels

Ordinal index: 49
Input: digout as BYTE, (digital output Byte)
Return: none

For general purposes output bits 0 to 5 can be used, bits 6 and 7 are reserved for motor brake control.

2.3.5. Functions to Read Values

long get_pos(BYTE axis)

This function reads the current position of the indicated motor.

Ordinal index: 2
Parameter: axis as BYTE
Return: Current encoder position as long

long get_PosErr(BYTE axis)

This function returns the current deviation of the encoder position from the profile position in encoder counts.

Ordinal index: 4
Parameter: axis as BYTE
Return: error counts as *long*

long get_v(BYTE axis)

This function returns the current velocity of the defined axis and returns it in counts/s.

Ordinal index: 46
Input: axis as BYTE
Return: velocity as long

int get_stat(BYTE axis)

This function returns the status word of the indicated axis.

Ordinal index: 6

Input: axis as BYTE
 Return: status value as integer

void get_pos4(TQ_long *Pos4)

This function reads the current positions of all 4 axes and writes them into an array of 4 long integers.

Ordinal index: 8
 Input : array[1..4] of long as TQ_long
 Return: none

WORD get_BoardStatus(void)

This function reads the board status register and returns it as unsigned 16 bit value.

Ordinal index: 33
 Input : none
 Return: board status register as WORD (unsigned 16 bit)

WORD get_RefSignal(BYTE axis)

This function reads the current reference signal line level and returns 0 if the line is low or 1 if the line is high.

Ordinal index: 7
 Input : axis as BYTE
 Return: 0 if line is LOW, formatted as WORD (unsigned 16 bit)
 1 if line is HIGH, formatted as WORD (unsigned 16 bit)

int get_ErrorStatus(void)

This function reads error status code :

```

0 : no error
1 : setQMC : chipset
10 : WaitQMC timeout
11 : setQMC cmd not found
12 :      timeout
13 : getQMC cmd not found
101 : translate: command not found
230 : FEP/FEN limit hit
231 :      user break
232 :      wrong axis
  
```

Ordinal index: 5
 Input : none
 Return: error status number as integer

int get_Limit(void)

This function returns the status of all limit signals of all 4 axes.

Ordinal index: 47

Input : none
 Return: global limit status word as integer

Bit #	Description
0	Axis 1 positive limit switch (1 = high)
1	Axis 1 negative limit switch (1 = high)
2	Axis 2 positive limit switch (1 = high)
3	Axis 2 negative limit switch (1 = high)
4	Axis 3 positive limit switch (1 = high)
5	Axis 3 negative limit switch (1 = high)
6	Axis 4 positive limit switch (1 = high)
7	Axis 4 negative limit switch (1 = high)
8-15	not used (set to 0)

int get_LimitAxis(BYTE axis)

This function returns the current limit status of the indicated axis.

Ordinal index: 76
 Input : axis as BYTE
 Return: 0 : no limit line high
 1 : positive limit line high
 2 : negative limit line high
 3 : both limit lines high

BYTE get_DIN(void)

This function returns the input byte of the 8 digital input lines.

Ordinal index: 48
 Input : none
 Return: digital input byte

2.3.6. Low Level Functions

void axis_select (BYTE axis)

This function activates the specified axis and updates an internal axis flag. Normal move commands do not need the axis_select function, except setQMC or getQMC are used.

Ordinal Index: 36
Input: axis as BYTE
Return: none

void **setQMC**(BYTE hexcommand, long parameter)

This function writes a genuine command and the related parameter to the activated axis. Use **axis_select** function to choose one axis prior to call setQMC. axis_select is only required if the axis should be changed.

Ordinal index: 38
Input: hexcommand as BYTE
Return: none

Example:

In order to clear the status word of one axis, the following sequence can be used:

```
axis_select (axisnumber)
setQMC (0x33,0)
```

long **getQMC**(BYTE hexcommand)

This function returns the questioned value. as long as the query does not concern a global command, the value for the activated axis is returned. Use axis_select function to choose one axis prior to call getQMC. axis_select is only required if the axis should be changed.

Ordinal index: 40
Input : command in hexcode as char (byte)
Return: value as long integer

3. Using C-842 with WindowsNT and Windows2000

3.1. How can I work under NT?

Due to restrictions of hardware access, NT systems as well as Windows2000 do not allow direct bus communication with the C-842 card. This is the reason that the Win95 DLL will not work.

Accessing the C-842 card in a NT system, the program or DLL has to access a kernel-null driver that enables the actual hardware access to the board.

















So the first step for NT and Windows2000 users is to install the C-842 NT-driver, supplied on the PI-Motion CD or as download from our site.

3.2. Installing the C-842 NT Driver

The C-842 NT Application Package allows to run the C-842 DC-Motor Controller with NT operating systems.

The package includes all installation files required to install and to register the driver in the NT environment. The supplied kernel driver interfaces via a DLL with the application program. So programmers who already have used other QFL libraries with C-842 applications may use the same calls with the new 32-bit DLL.

Disk Contents

Name	Größe	Typ	Geändert am
 layout.bin	1 KB	BIN Datei	29.07.98 14:35
 setup.lid	1 KB	LID Datei	29.07.98 14:35
 Data.tag	1 KB	TAG Datei	29.07.98 14:35
 data1.cab	129 KB	CAB Datei	29.07.98 14:35
 Setup.ini	1 KB	Konfigurati...	29.07.98 14:35
 _user1.cab	46 KB	CAB Datei	29.07.98 14:35
 _sys1.cab	401 KB	CAB Datei	29.07.98 14:34
 setup.ins	56 KB	Internet Co...	29.07.98 14:34
 setup.bmp	112 KB	Bitmap	28.07.98 14:01
 Setup.exe	59 KB	Anwendung	15.07.97 11:56
 _isdel.exe	8 KB	Anwendung	14.07.97 17:37
 _setup.dll	11 KB	Programm...	14.07.97 17:35
 _inst16.ex_	275 KB	EX_ Datei	14.07.97 17:29
 _inst32i.ex_	313 KB	EX_ Datei	14.07.97 14:08
 lang.dat	5 KB	DAT Datei	30.05.97 12:31
 os.dat	1 KB	DAT Datei	06.05.97 15:15

Note: This installation package may copy not the latest version of the DLL. Just add the latest release to you project directory.

This document refers to this library:

Current DLL version: C842NT31.DLL, version 3.1, (February 2001)

For update package, email to info@physikinstrumente.com

3.3. NT-Driver Installation

To install the NT application package in the system, run the SETUP.EXE from the C842 NT installation disk. The setup routine installs the driver "UIO.SYS", modifies the registry and copies the DLL with an example application **DLLTEST** into the destination directory. The project is established with Visual C++ 4.0 and contains all sources required to build the application.

You can not run the installation on Windows NT and Windows 2000 systems.

4. NT-DLL Library (Version 3.50)

FileName: QFLNT35.DLL, Version : 3.50

The full complexity of the command set of the motion control processor used with the C-842 DC-Motor controller board can be handled easily using the Library functions.

More than 100 commands are supported by the library. The command set includes all commands with direct reference to the processor's command set and emulates over that many additional commands.

Error Messages:

Error messages are printed into the stream 'stderr'.

Constants defined:

All commands available like MR, MA, DH, GH, SOP.....(there are more than 100 of these commands) are predefined as constants of the type TCMD. For full command reference see operating manual MS45E.

All commands are defined as variables of the type TCMD. The name of the variable complies with the command names used in the WinMove program and documented in the MS45E operating manual.

4.1. C842NT35.DLL Function List :

see table at right:

<u>Ordinal</u>	<u>Entry Point</u>	<u>Name</u>
0000	000011f0	Axis_Installed
0001	00001300	Board_Installed
0002	00002190	ClearAxisStatus
0003	000021b0	InitAxis
0004	000022a0	InitBoard
0005	00001230	Init_LS
0006	00004cc0	MoveA
0007	00004d00	MoveR
0008	00004ba0	QMCBusy
0009	00004bd0	QMCReady
000a	00001160	Set_BaseAddress
000b	00001190	Set_Board
000c	00004c00	StatusBoard
000d	00001280	autodetect
000e	00004b20	axis_select
000f	00001030	c842_close
0010	00001000	c842_open
0011	00002320	execute
0012	000045f0	getQMC
0013	00004590	get_AxisLimit
0014	000044d0	get_AxisStatus
0015	00004c20	get_BoardStatus
0016	00004dc0	get_DLLversion
0017	00004580	get_Limit
0018	00004c40	get_RefStatus
0019	000045b0	get_din
001a	00004540	get_pos
001b	000044f0	get_pos4
001c	00004560	get_posErr
001d	000044a0	get_stat
001e	00001330	moving
001f	00003bb0	ref_board
0020	00003ec0	ref_board_ss
0021	00004850	setQMC
0022	00004d80	set_acc
0023	000045d0	set_dout
0024	00002250	set_pid
0025	00004d40	set_vel
0026	00001360	translate

4.2. Building Applications

- Include the header "C842NT35.H" into your application.
- Link or add the import library C842NT35.LIB into your project
- Prior to access the C-842 by the translate function, call the driver initializing function "c842_open(void)". When leaving the application program, close the driver with the function "c842_close(void)". These functions create the connection to the UIO.SYS driver.

4.3. Header File

```
// C842NT35.H
// 2002-02-10
//-----
// created 1998-10-29
// modified 2001-02-16 (version 3.10)
// modified 2001-06-02 (version 3.40)
// -----
#include <windows.h>

// #define TRUE 1
// #define FALSE 0
// #define BYTE unsigned char

#include "C842CMD26.H"

#ifdef __cplusplus
extern "C" {
#endif

#ifdef C842_DLL
#define FUNC_DECL __declspec(dllexport)
#else
#define FUNC_DECL __declspec(dllimport)
#endif

// Functions :
BOOL FUNC_DECL c842_open(void);
BOOL FUNC_DECL c842_close(void);

void FUNC_DECL Init_LS(BYTE ax1, BYTE ax2, BYTE ax3, BYTE ax4);
short FUNC_DECL Board_Installed(void);
short FUNC_DECL InitBoard(BYTE mode);
void FUNC_DECL InitAxis(BYTE axis);
int FUNC_DECL autodetect(void);

void FUNC_DECL translate(char *comstr, char *report);
short FUNC_DECL execute(BYTE axis, TCMD command, long para, char
*report);
void FUNC_DECL ClearAxisStatus(BYTE axis);
short FUNC_DECL axis_select(BYTE axis);

long FUNC_DECL ref_board(BYTE axis, BYTE mode); // added V34
long FUNC_DECL ref_board_ss(BYTE axis, BYTE mode); // added V34

short FUNC_DECL get_pos4(long *position);
long FUNC_DECL get_pos(BYTE axis);
long FUNC_DECL get_posErr(BYTE axis); // added V31
short FUNC_DECL get_stat(BYTE axis); // added V31 for compatibility
short FUNC_DECL get_BoardStatus(void); // added V25
short FUNC_DECL get_AxisStatus(BYTE axis); // added V31
short FUNC_DECL get_RefStatus(BYTE axis); // added V29
long FUNC_DECL get_Limit(void); // added V33
long FUNC_DECL get_AxisLimit(BYTE axis); // added V33
```



```
void FUNC_DECL set_pid(BYTE axis,int pterm,int iterm,int dterm); // added V33

void FUNC_DECL set_vel(BYTE axis, long velocity);
void FUNC_DECL set_acc(BYTE axis, long acceleration);

BYTE FUNC_DECL get_din(void); // added V33
void FUNC_DECL set_dout(BYTE outbyte); // added V33

void FUNC_DECL get_DLLversion(char *version); // added V25

short FUNC_DECL setQMC(BYTE cmd, long para);
long FUNC_DECL getQMC(BYTE cmd);
short FUNC_DECL QMCBusy(void);
short FUNC_DECL QMCReady(void);

void FUNC_DECL Set_BaseAddress(unsigned address);
void FUNC_DECL Set_Board(BYTE boardnumber); // added V28

BYTE FUNC_DECL Axis_Installed(void);
BYTE FUNC_DECL moving(BYTE axis);

void FUNC_DECL MoveA(BYTE axis, long position);
void FUNC_DECL MoveR(BYTE axis, long shift); // added V31
short FUNC_DECL StatusBoard(void); // added for compatibility reasons only,
                                   // use AxisStatus() instead

#ifdef __cplusplus
}
#endif
```

4.4. Function Reference

4.4.1. NT-driver Functions

BOOL c842_open(void)

This open function has to be called prior to call any other function that accesses the NT driver.

Return: TRUE if success

BOOL c842_close(void)

This open function has to be called prior the program is terminated.

Return: TRUE if success

4.4.2. Initializing

short Board_Installed (void)

Return: TRUE if a C-842 is installed in a PC slot
FALSE if no controller is found.

short InitBoard(BYTE mode)

This function initializes the C-842 performing these functions:

- Sets the output to analog,
- sets limit sense to 255,
- sets the boardmode to the specified value

This function has to be called after the C-842 is powered up or after a reset was performed.

For details see Operating Manual MS 45E, keyword "board function register".

Return: board status byte if success
-1 if failed

void Init_LS (BYTE axis1, BYTE axis2, BYTE axis3, BYTE axis4)

This function sets the Limit_Sense Register Byte. For unknown configurations it can be replaced by the autodetect function.

The purpose of this function is that after power up, limit switch wiring and levels have to be defined. If the limit switches are not used, the axis parameter has to be set to 0, otherwise they have to be set to 1.

Example: One PI-stage is connected to axis #2. All other axes are not used or have stages without limit switch option. Then call Init_LS(0,1,0,0).

Return : none

Int autodetect (void)

This function sets the limit sense register to the current level of the limit switch lines. It can be called once at the beginning of the program and replaces the function Init_LS.

No limit switch must be hit when autodetect is called !

Return: Limit_Sense value

void InitAxis (BYTE axis)

This procedure initializes the specified axis with default parameters.

Default parameters are not optimized to the current stages connected. Parameter optimization has to be done by individual setting of terms using the translate or execute functions.

This function can be called after InitBoard to set the axis to a operational state.

Return: none

void Set_BaseAddress (unsigned address)

This function sets the base address of the C-842. With this function multiple boards can be accessed in one PC.

If 0x210 is used as address (default address) and only this board is installed, this function is not needed.

See MS 45E Operating Manual for details.

void Set_Board (BYTE boardnumber)

This function can be used if more than one board is installed in the PC. Depending on the boardnumber, the active board is selected:

boardnumber = 1 : board address 0x210

boardnumber = 2 : board address 0x214

boardnumber = 3 : board address 0x218

boardnumber = 4 : board address 0x21C

parameter: boardnumber: range 1..4

return: none

BYTE Axis_Installed (void)

Return: Number of Axis supported by the processor of the C-842. If a C842.40 is installed, 4 is returned, in case of C-842.20 the return value is 2.

4.4.3. General Communication

short execute(BYTE axis, TCMD command, long parameter, char *report);

execute an universal function that allows to transfer all commands to the board and to read the report string.

The full command set of the processor can be addressed as well as additional commands are available, having no direct implementation in the processor firmware.

Return: always 0

Example:

Move motor #3 for 5000 counts in positive direction relative to the present position, then wait until the new position is reached, then return home and indicate reaching the home position by a +5V signal at output channel # 8: :

```
execute( 3, MR, 5000, report);    (move motor # 3 relative 5000 counts)
execute( 3, WS, 0, report);      (wait until motor has reached the position)
execute( 3, GH, 0, report);      (Start motor # 3 to run home)
```

void **translate**(char *comstr, char *report)

Like the execute function also the translate function has an universal character allowing to address all commands as well as the group of emulated commands. Unlike the execute function, translate works with strings containing the axis identifier, the command and the parameter.

Return: none

Examples:

```
translate("3MR5000",rep);    (move motor # 3 relative 5000 by counts)
translate("3WS",rep);        (wait until motor has reached its position)
translate("3GH",rep)         (Start motor # 3 to run home)
translate("3WS",rep)         (Wait until motor has reached its home)
translate("3TP",rep)         (report the position)
```

short **axis_select** (BYTE axis)

This procedure activates the selected axis. All consecutive issued commands are directed to this axis. Using the execute and the translate procedures, axis_select() is not required.

Return: TRUE if success, FALSE if failed

long **get_pos** (BYTE axis)

This function offers a fast access to the current motor encoder position of the specified axis. The return value is the position in counts.

Return: position in counts

long **get_posErr** (BYTE axis)

This function returns the current position error in encoder counts, that is the difference between the current motor encoder position and the profile target.

Return: position error in counts

short get_pos4 (long *position)

This function offers a fast access to the current motor encoder positions of all 4 axes in counts. The parameter is a pointer to an array of 4 long integers representing the 4 motor positions.

Return: 0 if success, -1 if failed.

short setQMC (BYTE cmd, long para)

This procedure is used to transfer a processor command with the associated parameter value to the board.

Return: TRUE if success, FALSE if failed

long getQMC (BYTE cmd)

This function sends a query command to the C-842 and returns the numerical value of the requested register.

Return: requested value as long integer

short QMCBusy(void)

This function waits until the controller is ready to accept new data.

Return: TRUE: (not busy) number of timeout counter, max 2500,
FALSE (busy) if timeout

short QMCReady(void)

This function waits until the C-842 becomes ready to accept data.

Return: TRUE if ready
FALSE if not ready

void set_acc(BYTE axis, long acceleration)

This function sets the acceleration for the specified axis. The acceleration value is scaled and has to be multiplied by 2,500 to get the physical acceleration in c/s/s, e.g. a value of 200 sets the acceleration to 500,000 c/s/s.

Parameter: axis as BYTE
acceleration value as long

Return: none

void set_vel(BYTE axis, long velocity)

This function sets the velocity for the specified axis in c/s.

Parameter: axis as BYTE
velocity as long

Return: none

```
void set_pid(BYTE axis, int pterm, int item, int dterm)
```

This function sets the p-i-d values.

Parameters: axis as BYTE
 pterm as long (integer),
 item as long (integer)
 dterm as long (integer)

4.4.4. Moving Functions

```
BYTE moving (BYTE axis)
```

Depending on the moving status of the specified axis, this function returns TRUE if the motor is moving or FALSE if the motor profile position is on its target.

Return: 0 (FALSE) not moving
 1 (TRUE) moving

```
void MoveA (BYTE axis, long position)
```

This function moves the motor to a new position.

Return: none

```
void MoveR (BYTE axis, long movelength)
```

This function moves the motor by movelength counts starting from the current position.

Return: none

4.4.5. Digital I/O

BYTE get_din (void)

This function returns the signals applied to the digital input channels

Return: Input signals as BYTE

void set_dout (BYTE outbyte)

This function sets the digital output channels 1 to 6. Channels 7 and 8 are reserved for brake control.

Parameter: output byte

Return: none

4.4.6. Status, Limit and Reference Signal Requests

short get_BoardStatus (void)

Return: Board status register.

For more information see MS 45E Operating Manual, page 10.

short get_AxisStatus (BYTE axis)

This function returns the axis status register.

Return: Axis status register as short

For more information see MS 45E Operating Manual, page 10.

short get_stat (BYTE axis)

This function returns the axis status register and has the same functionality as the function get_AxisStatus(). It was added due to compatibility reasons.

Return: Axis status register.

For more information see MS 45E Operating Manual, page 10.

short get_RefStatus (BYTE axis)

New function since version 3.0

This function evaluates the board status register in order to get the voltage level of the reference signal. The value returned reflects the current signal level.

Unlike this function, the genuine processor command GET_HOME reads the home input if no static level is applied.

For more information see MS 45E Operating Manual, page 10.

Parameter: axis number, range 1 to 4

Return: 0 if reference input signal is low
 1 if reference input signal is high

short **StatusBoard**(void)

Function for compatibility reasons only. Use get_BoardStatus() instead.

Parameters: none

Return: Board function register

long **get_Limit**(void)

This function returns the limit status of all 4 axes. See command 0x67.

Return: Board limit status as long

long **get_AxisLimit**(BYTE axis)

This function returns the limit status of the specified axis.

Return: 0 : no limit hit
 1 : positive limit
 2 : negative Limit
 3 : both limits

void **get_DLLversion** (char *version)

This function delivers a string with library version information. Max. 80 characters.

Return: none


```
long ref_board (BYTE axismask, BYTE mode)
```

```
long ref_board_ss (BYTE axismask, BYTE mode)
```

since DLL version 3.4

These functions perform a simultaneous reference position search of all axes available on the C-842. The functions allow to start the motors specified by the "axismask" parameter (this can be each combination of motors) at the same time in the right direction towards the stage origin point.

The feature "starting in the right direction" is only available with stages having the "full range reference signal" option, that means defined signal levels at each side of the reference point (e.g. M-500 and M-400 series stages).

Over that, the **ref_board_ss** function performs a side correct approach of the reference point for hysteresis-free origin point finding. If the stage approaches the reference point while running in negative direction, the stage overruns the point by 50,000 counts, is then stopped and started for a new reference search in positive direction. *Only reference points found while moving in positive direction are assumed to be correct.*

The functions do not watch keyboard strokes.

The functions return as soon as break conditions have occurred for all axes defined in the axis parameter.

Break conditions are:

- reference signal found,
- limit hit,
- motion error
- timeout (set to 120 s)

If after 120 seconds not all axes have terminated by break conditions, the function returns by timeout condition with bit 16 set.

Parameters :

axismask (byte, unsigned character)
 The axis parameter allows to select one or multiple axes for the reference search.
 Bit 0 (value 1) enables axis #1
 Bit 1 (value 2) enables axis #2
 Bit 2 (value 4) enables axis #3
 Bit 3 (value 8) enables axis #4

mode (byte, unsigned character)
 The mode parameter allows for all axes to define the starting direction depending on the logic level of the reference signal.
 Bit 0 (value 1) if set, starts axis #1 in positive direction if reference signal is high.
 Bit 1 (value 2) if set, starts axis #2 in positive direction if reference signal is high.
 Bit 2 (value 4) if set, starts axis #3 in positive direction if reference signal is high.
 Bit 3 (value 8) if set, starts axis #4 in positive direction if reference signal is high.

Return: (long, 4 bytes)
 The return value indicates whether reference points have been found, limit

switches have been contacted or a motion errors have occurred. Also the approach direction of each axis is flagged.

bit 0 (value 1) axis #1 : Reference signal detected
bit 1 (value 2) axis #2 : Reference signal detected
bit 2 (value 4) axis #3 : Reference signal detected
bit 3 (value 8) axis #4 : Reference signal detected

bit 4 (value 16) axis #1 : Limit hit
bit 5 (value 32) axis #2 : Limit hit
bit 6 (value 64) axis #3 : Limit hit
bit 7 (value 128) axis #4 : Limit hit

bit 8 (value 256) axis #1 : Motion error
bit 9 (value 512) axis #2 : Motion error
bit 10 (value 1024) axis #3 : Motion error
bit 11 (value 2048) axis #4 : Motion error

bit 12 (value 4096) axis #1 : approach in negative direction
bit 13 (value 8192) axis #2 : approach in negative direction
bit 14 (value 16384) axis #3 : approach in negative direction
bit 15 (value 32768) axis #4 : approach in negative direction

bit 16 (value 65536) timeout occurred

5. List of QFL Commands

This list is valid for both, Win95 and NT libraries.

```

1  UP , // Update
   TP , // Tell Position
   GLS, // Get Limit Switch
   TT , // Tell Target
   MUP, // Multiple Update
   TS , // Tell Status
   GIP, // Get Index Position
   MA , // Move Absolute
   SV , // Set Velocity
10  GMO, // Get Mode of axis
   MR , // Move Relative
   GH , // Go Home
   TX , // Tell reference position
   WS , // Wait Stop
   MN , // Motor On
   MF , // Motor off
   TV , // Tell Velocity
   CLR, // Clear Status
   AB  // Abort motion
20  ST , // Stop
   STS // Stop Smooth
   SA , // Set Acceleration
   DH , // Define Home
   TY , // Tell programmed Velocity
   GPP, // Get Profile Position
   GPV, // Get Profile Velocity
   TF , // Tell actual Position Error
   RT , // Reset Board
   GB , // Get Breakpoint
30  GIR, // Get status of interrupting axis
   SAN, // Set Auto update on
   SAF, // Set Auto update off
   TL , // Tell acceleration
   ASN, // Auto Stop on
   ASF, // Auto Stop off
   SMO, // Set Motor Output
   TE , // Tell Error
   DP , // Define Proportional gain
   DD , // Define Differential gain
40  DI , // Define Integral gain
   DF , // Define velocity Feedforward gain
   DL , // Define integration Limit
   GP , // Get integral gain
   GD , // Get Differential gain
   GI , // Get Integral gain
   GF , // Get velocity Feedforward gain
   GL , // Get integration limit
   GPE, // Get Position Error
   GSI, // Get integrated position error
50  RTI, // Get Time
   GIM, // Get irq mask
   SCH, // Set Capture Home
   SCI, // Set Capture Index
   LN , // Set limit switch sensing on
   LF , // Set limit switch sensing off
   SPS, // Set Profile to S-curve
   SPT, // Set Profile to trapezoidal
   SMA, // Set Maximum Acceleration
   GMA, // Get Maximum Acceleration

```

```
60  SYN,  // Synchronize profile
    SPE,  // Set max Position Error Limit
    SLS,  // Set Limit Sense byte
    VE ,  // Tell Version
    GTI,  // Get system Time
    WA ,  // Wait Absolute
    SBT,  // Set Breakpoint to Time
    SBP,  // Set Breakpoint Positive Target
    SBN,  // Set Breakpoint Negative Target
    SBA,  // Set Breakpoint to positive act position
70  SBC,  // Set Breakpoint to negative act position
    SBF,  // Set Breakpoint mode off
    SB ,  // Set Breakpoint
    SIM,  // Set interrupt mask
    GJ ,  // Get Jerk
    SJ ,  // Set Jerk
    SRA,  // Set gear Ratio
    GRA,  // Get gear Ratio
    SPV,  // Set Profile mode to Velocity
    SPG,  // Set Profile mode to electronic Gear
80  AN ,  // Axis on
    AF ,  // Axis off
    SOP,  // Set output to PWM
    SOH,  // Set output to 16bit DAC
    SST,  // Set Sample Time
    GT ,  // Get sample Time
    GOM,  // Get current output mode
    TZ ,  // Tell channels in decimal
    TC ,  // Tell channel
    CN ,  // Channel on
90  CF ,  // Channel off
    IV ,  // Increment Velocity
    WC ,  // Wait Channel
    WT ,  // Wait Trigger
    FEP,  // find edge in positive direction
    FEN,  // find edge in negative direction
    RET  // return from motion error condition
    BN   // brake ON
    BF   // brake OFF
```

