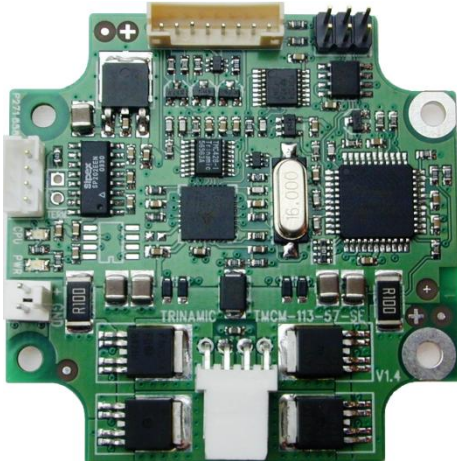


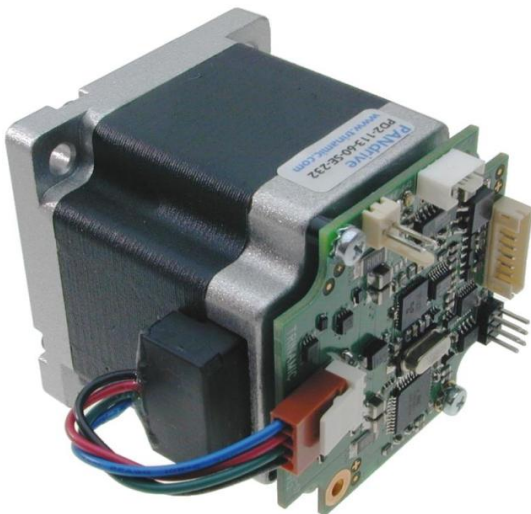
V 1.04

TMCL™ FIRMWARE MANUAL



TMCM-113-60-SE

controller / driver
up to 2.8A RMS / 24V
RS232 or RS485
integrated sens0step™ encoder
chopSync™
stallGuard™



PDx-113-57/60-SE

full mechatronic device
up to 2.8A RMS / 24V
RS232 or RS485
integrated sens0step™ encoder
chopSync™
stallGuard™

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

www.trinamic.com



Table of contents

1	Life support policy	4
2	Features	5
3	Overview	6
4	Putting the PDx-113-57/60-SE into operation	7
4.1	Starting up	8
4.2	Writing a simple TMCL™ program	9
4.3	Operating the module in direct mode	10
5	TMCL™ and TMCL-IDE	11
5.1	Binary command format	11
5.2	Reply format	12
5.2.1	Status codes	13
5.3	Standalone applications	13
5.4	TMCL™ command overview	13
5.4.1	Motion commands	13
5.4.2	Parameter commands	14
5.4.3	I/O port commands	14
5.4.4	Control commands	14
5.4.5	Calculation commands	14
5.5	TMCL™ commands	15
5.6	The ASCII interface	17
5.6.1	Format of the command line	17
5.6.2	Format of a reply	17
5.6.3	Commands that can be used in ASCII mode	17
5.6.4	Configuring the ASCII interface	17
5.7	Commands	19
5.7.1	ROR (rotate right)	19
5.7.2	ROL (rotate left)	20
5.7.3	MST (motor stop)	21
5.7.4	MVP (move to position)	22
5.7.5	SAP (set axis parameter)	24
5.7.6	GAP (get axis parameter)	25
5.7.7	STAP (store axis parameter)	26
5.7.8	RSAP (restore axis parameter)	27
5.7.9	SGP (set global parameter)	28
5.7.10	GGP (get global parameter)	29
5.7.11	STGP (store global parameter)	30
5.7.12	RS GP (restore global parameter)	31
5.7.13	RFS (reference search)	32
5.7.14	SIO (set output)	33
5.7.15	GIO (get input/output)	34
5.7.16	CALC (calculate)	36
5.7.17	COMP (compare)	37
5.7.18	JC (jump conditional)	38
5.7.19	JA (jump always)	39
5.7.20	CSUB (call subroutine)	40
5.7.21	RSUB (return from subroutine)	41
5.7.22	WAIT (wait for an event to occur)	42
5.7.23	STOP (stop TMCL™ program execution)	43
5.7.24	SCO (set coordinate)	44
5.7.25	GCO (get coordinate)	45
5.7.26	CCO (capture coordinate)	46
5.7.27	CALCX (calculate using the X register)	47
5.7.28	AAP (accumulator to axis parameter)	48
5.7.29	AGP (accumulator to global parameter)	49
5.7.30	CLE (clear error flags)	50
5.7.31	Customer specific TMCL™ command extension (UF0... UF7 / user function)	51

5.7.32	Request target position reached event.....	52
5.7.33	BIN (return to binary mode)	53
5.7.34	TMCL™ control functions	54
6	Axis parameters	56
6.1	Calculation: velocity and acceleration vs. microstep and fullstep frequency	60
6.1.1	Microstep frequency.....	60
6.1.2	Fullstep frequency.....	60
6.2	Stall detection.....	62
6.2.1	Finding the reference point with stall detection.....	62
6.3	Encoder parameters	63
6.3.1	Selecting the prescaler.....	63
6.4	Reference search and homing mode.....	64
7	Global parameters	66
7.1	Bank 0	66
7.2	Bank 1	68
7.3	Bank 2	68
8	Hints and tips	69
8.1	Fixing microstep errors	69
8.2	Using the RS485 interface	69
8.2.1	Using RS485 with converter	69
8.2.2	Using RS485 with USB-2-X interface converter	69
9	Revision history	70
9.1	Document revision	70
9.2	Firmware revision.....	70
10	References.....	71

1 Life support policy

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specific written consent of TRINAMIC Motion Control GmbH & Co. KG.

Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

© TRINAMIC Motion Control GmbH & Co. KG 2011

Information given in this data sheet is believed to be accurate and reliable. However neither responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties, which may result from its use.

Specifications are subject to change without notice.



2 Features

The PDx-113-57/60-SE is a full mechatronic device consisting of a NEMA 23 (flange size 57mm) or NEMA 24 (flange size 60mm) stepper motor, controller/driver electronics and integrated encoder. The electronics itself is also available without the motor as TCM-113-60-SE module.

Applications

- Compact single-axis stepper motor solutions
- Encoder feedback for high reliability operation (-SE option)

Electrical data

- Supply voltage: +24V DC nominal (+7V... +28.5V DC)
- Motor current: up to 2.8A RMS (programmable)

Integrated motor (for PDx-113-57/60-SE only)

- Two phase bipolar stepper motor with 2.8A RMS nom. coil current
- Holding torque with 57mm motor: 0.55Nm, 1.01Nm, 1.26Nm or 1.89Nm
- Holding torque with 60mm motor: 1.1Nm, 1.65Nm, 2.1Nm or 3.1Nm

Integrated encoder (for -SE option only)

- Integrated sensOstep™ magnetic encoder (max. 256 increments per rotation) for step-loss detection under all operating conditions

Integrated motion controller

- Motion profile calculation in real-time (TMC428/429 motion controller)
- On the fly alteration of motor parameters (e.g. position, velocity, acceleration)

Integrated bipolar stepper motor driver

- Up to 16 microsteps per full step
- High-efficient operation, low power dissipation (TMC249 stepper driver with external MOSFETs)
- Dynamic current control
- Integrated protection
- Integrated stallGuard™ for motor stall detection (e.g. elimination of end switches)
- Integrated chopSync™ for high velocity operation

Interfaces

- 2 inputs for reference switches, 2 general purpose inputs and 2 general purpose outputs
- either RS232 or RS485 serial communication interfaces

Software

- Available with TMCL™ (both interface options)
- standalone operation or remote controlled operation
- program memory (non volatile) for up to 2048 TMCL commands
- PC-based application development software TMCL-IDE available for free

3 Overview

As with most TRINAMIC modules the software running on the microprocessor of the PDx-113-57/60-SE consists of two parts, a boot loader and the firmware itself. Whereas the boot loader is installed during production and testing at TRINAMIC and remains – normally – untouched throughout the whole lifetime, the firmware can be updated by the user. New versions can be downloaded free of charge from the TRINAMIC website (<http://www.trinamic.com>).

The firmware shipped with this PANdrive™ is related to the standard TMCL™ firmware shipped with most of TRINAMIC modules with regard to protocol and commands. Corresponding, the module of the PANdrive™ is based on the TMC428/429 motion controller for stepper motors and the TMC249 high power driver and supports the standard TMCL™ with a special range of values. All commands and parameters available with this unit are explained on the following pages.

4 Putting the PDx-113-57/60-SE into operation

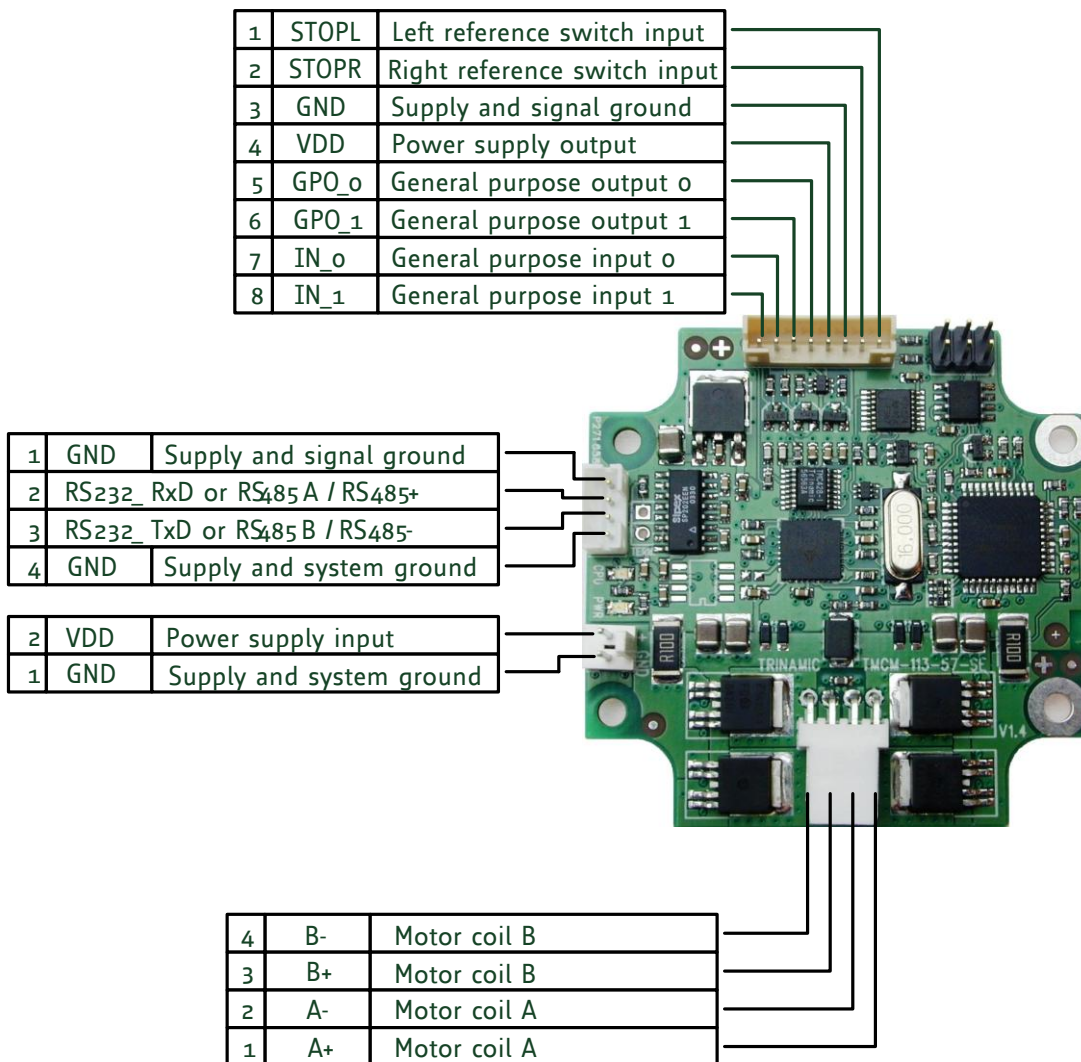
Here you can find basic information for putting your PANdrive into operation. The text contains a simple example for a TMCL™ program and a short description of operating the module in direct mode.

The things you need:

- PDx-113-57/60-SE
- Interface (RS232 or RS485) suitable to your PDx-113-57/60-SE version with cables
- Power supply for 24V
- TMCL-IDE program and PC

Precautions:

- **Do not connect or disconnect the motor while powered!**
- Do not mix up connections or short-circuit pins.
- Avoid bounding IO wires with motor power wires as this may cause noise picked up from the motor supply.
- Do not exceed the maximum power supply of 28.5V.
- **Start with power supply OFF!**



4.1 Starting up

Here we show you, how to connect the RS232 interface. Use the RS485 interface similarly please.

1. Connect the interface:

Please keep in mind that the RS232 **transmit** signal wire of the master has to be connected to the RS232 **receive** signal wire of the board and vice versa.

PC (D-SUB 9pin)		PDx-113-57/60-SE (Serial communication connector)		
Pin	Label	Pin	Label	
2	RS232_RxD	3	RS232_TxD	
3	RS232_TxD	2	RS232_RxD	
5	GND	1, 4	GND	

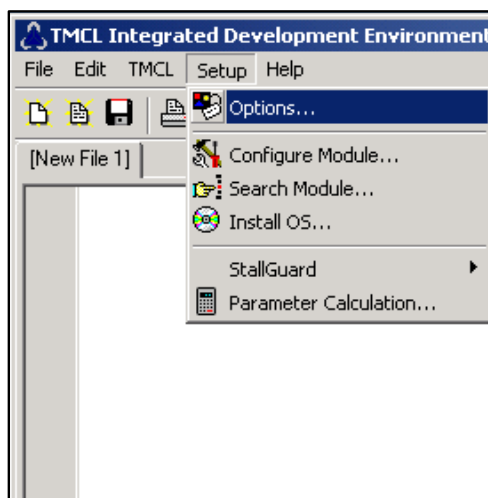
2. Connect power supply:

PDx-113-57/60-SE (Power connector)		
Pin	Label	
2	Power supply	
1	Ground	

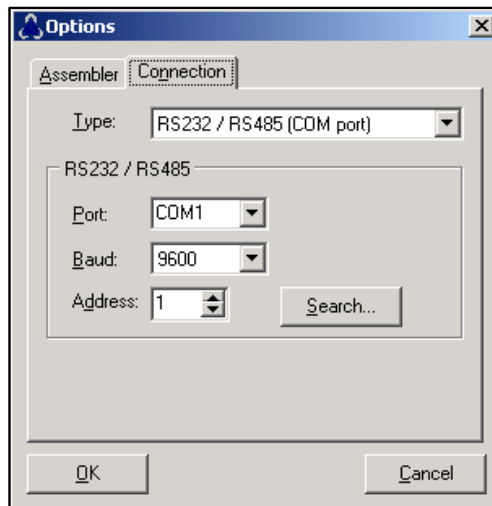
3. Turn power ON. The green LED of the module glows and gives notice that the module is powered with its supply voltage. ***If this does not occur, switch power OFF and check your connections as well as the power supply.***
4. Start the TMCL-IDE software development environment (available on the TechLibCD and www.trinamic.com).

Installing the TMCL-IDE:

- Make sure the COM port you intend to use is not blocked by another program.
- Open TMCL-IDE by clicking **TMCL.exe**.
- Choose **Setup** and **Options** and thereafter the **Connection tab**.



- Choose **COM port** and **type** with the parameters shown below (baud rate 9600). Click **OK**.



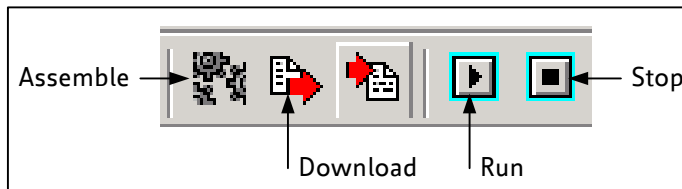
4.2 Writing a simple TMCL™ program

1. Type the following text in the open window.

```
//A simple example for using TMCL and the TMCL-IDE
SAP 4, 0, 1000           //Set the maximum speed

Loop: MVP ABS, 0, 150000 //Move to position 150000
WAIT POS, 0, 0
WAIT TICKS, 0, 200
MVP ABS, 0, 0           //Move back to position 0
WAIT POS, 0, 0
WAIT TICKS, 0, 100

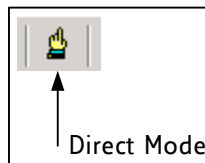
JA Loop                 // Infinite Loop
```



2. Click the **Assemble** icon.
3. Thereafter click the **Download** icon to download the code into the TCM-113-60-SE.
4. Click the **Run** icon. The downloaded program will now be executed.
5. Click **Stop** button to stop the program.

4.3 Operating the module in direct mode

1. Start TMCL *Direct Mode*.



2. If the communication is established the TCM-113-60-SE is automatically detected. ***If the module is not detected, please check all points above (cables, interface, power supply, COM port, baud rate).***
3. Issue a command by choosing ***instruction***, ***type*** (if necessary), ***motor***, and ***value*** and click *Execute* to send it to the module. As the TCM-113 controls and drives only one motor, always choose **0** for the motor number.



Examples:

- ROR rotate right, motor 0, value 500 -> Click *Execute*. The motor is rotating now.
- MST motor stop, motor 0 -> Click *Execute*. The motor stops now.



Please use the TMCL-IDE axis parameter calculation tool for getting best values.

5 TMCL™ and TMCL-IDE

The PDx-113-57/60-SE supports TMCL™ direct mode (binary commands or ASCII interface) and standalone TMCL™ program execution. You can store up to 2048 TMCL™ instructions on it.

In direct mode the TMCL™ communication over RS485 or RS232 follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the PDx-113-57/60-SE. The TMCL™ interpreter on the module will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over RS485/RS232 to the bus master. The master should not transfer the next command till then. Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.

The Trinamic Motion Control Language (TMCL™) provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TCM™ module to form programs that run standalone on the module. For this purpose there are not only motion control commands but also commands to control the program structure (like conditional jumps, compare and calculating).

Every command has a binary representation and a mnemonic. The binary format is used to send commands from the host to a module in direct mode, whereas the mnemonic format is used for easy usage of the commands when developing standalone TMCL™ applications using the TMCL-IDE (Integrated Development Environment).

There is also a set of configuration variables for the axis and for global parameters which allow individual configuration of nearly every function of a module. This manual gives a detailed description of all TMCL™ commands and their usage.

5.1 Binary command format

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes. When a command is to be sent via RS232 or RS485 interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In this case it consists of nine bytes.

The binary command format for RS232 and RS485 is as follows:

Bytes	Meaning
1	Module address
1	Command number
1	Type number
1	Motor or Bank number
4	Value (MSB first!)
1	Checksum

- The checksum is calculated by adding up all the other bytes using an 8-bit addition.

Checksum calculation

As mentioned above, the checksum is calculated by adding up all bytes (including the module address byte) using an 8-bit addition. Here are two examples to show how to do this:

- in C:

```
unsigned char i, Checksum;
unsigned char Command[9];

//Set the "Command" array to the desired command
Checksum = Command[0];
for(i=1; i<8; i++)
    Checksum+=Command[i];

Command[8]=Checksum; //insert checksum as last byte of the command
//Now, send it to the module
```
- in Delphi:

```
var
    i, Checksum: byte;
    Command: array[0..8] of byte;

//Set the "Command" array to the desired command

//Calculate the Checksum:
Checksum:=Command[0];
for i:=1 to 7 do Checksum:=Checksum+Command[i];
Command[8]:=Checksum;
//Now, send the "Command" array (9 bytes) to the module
```

5.2 Reply format

Every time a command has been sent to a module, the module sends a reply.

The reply format for RS485 and RS232 is as follows:

Bytes	Meaning
1	Reply address
1	Module address
1	Status (e.g. 100 means "no error")
1	Command number
4	Value (MSB first!)
1	Checksum

- The checksum is also calculated by adding up all the other bytes using an 8-bit addition.
- Do not send the next command before you have received the reply!

5.2.1 Status codes

The reply contains a status code.

The status code can have one of the following values:

Code	Meaning
100	Successfully executed, no error
101	Command loaded into TMCL™ program EEPROM
1	Wrong checksum
2	Invalid command
3	Wrong type
4	Invalid value
5	Configuration EEPROM locked
6	Command not available

5.3 Standalone applications

The module is equipped with an EEPROM for storing TMCL™ applications. You can use the TMCL-IDE for developing standalone TMCL™ applications. You can load your program down into the EEPROM and then it will run on the module. The TMCL-IDE contains an editor and a *TMCL™ assembler* where the commands can be entered using their mnemonic format. They will be assembled automatically into their binary representations. Afterwards this code can be downloaded into the module to be executed there.

5.4 TMCL™ command overview

In this section a short overview of the TMCL™ commands is given.

5.4.1 Motion commands

These commands control the motion of the motor. They are the most important commands and can be used in direct mode or in standalone mode.

Mnemonic	Command number	Meaning
ROL	2	Rotate left
ROR	1	Rotate right
MVP	4	Move to position
MST	3	Motor stop
RFS	13	Reference search
SCO	30	Store coordinate
CCO	32	Capture coordinate
GCO	31	Get coordinate

5.4.2 Parameter commands

These commands are used to set, read and store axis parameters or global parameters. Axis parameters can be set independently for the axis, whereas global parameters control the behavior of the module itself. These commands can also be used in direct mode and in standalone mode.

Mnemonic	Command number	Meaning
SAP	5	Set axis parameter
GAP	6	Get axis parameter
STAP	7	Store axis parameter into EEPROM
RSAP	8	Restore axis parameter from EEPROM
SGP	9	Set global parameter
GGP	10	Get global parameter
STGP	11	Store global parameter into EEPROM
RSGP	12	Restore global parameter from EEPROM

5.4.3 I/O port commands

These commands control the external I/O ports and can be used in direct mode and in standalone mode.

Mnemonic	Command number	Meaning
SIO	14	Set output
GIO	15	Get input
SAC	29	Access to external SPI device

5.4.4 Control commands

These commands are used to control the program flow (loops, conditions, jumps etc.). ***It does not make sense to use them in direct mode. They are intended for standalone mode only.***

Mnemonic	Command number	Meaning
JA	22	Jump always
JC	21	Jump conditional
COMP	20	Compare accumulator with constant value
CLE	36	Clear error flags
CSUB	23	Call subroutine
RSUB	24	Return from subroutine
WAIT	27	Wait for a specified event
STOP	28	End of a TMCL™ program

5.4.5 Calculation commands

These commands are intended to be used for calculations within TMCL™ applications. ***Although they could also be used in direct mode it does not make much sense to do so.***

Mnemonic	Command number	Meaning
CALC	19	Calculate using the accumulator and a constant value
CALCX	33	Calculate using the accumulator and the X register
AAP	34	Copy accumulator to an axis parameter
AGP	35	Copy accumulator to a global parameter

For calculating purposes there is an accumulator (or accu or A register) and an X register. When executed in a TMCL™ program (in standalone mode), all TMCL™ commands that read a value store the result in the accumulator. The X register can be used as an additional memory when doing calculations. It can be loaded from the accumulator.

When a command that reads a value is executed in direct mode the accumulator will not be affected. This means that while a TMCL™ program is running on the module (standalone mode), a host can still send commands like GAP, GGP or GIO to the module (e.g. to query the actual position of the motor) without affecting the flow of the TMCL™ program running on the module.

5.5 TMCL™ commands

The following TMCL™ commands are currently supported:

Command	Number	Parameter	Description
ROR	1	<motor number>, <velocity>	Rotate right with specified velocity
ROL	2	<motor number>, <velocity>	Rotate left with specified velocity
MST	3	<motor number>	Stop motor movement
MVP	4	ABS REL COORD, <motor number>, <position offset>	Move to position (absolute or relative)
SAP	5	<parameter>, <motor number>, <value>	Set axis parameter (motion control specific settings)
GAP	6	<parameter>, <motor number>	Get axis parameter (read out motion control specific settings)
STAP	7	<parameter>, <motor number>	Store axis parameter permanently (non volatile)
RSAP	8	<parameter>, <motor number>	Restore axis parameter
SGP	9	<parameter>, <bank number>, value	Set global parameter (module specific settings e.g. communication settings or TMCL™ user variables)
GGP	10	<parameter>, <bank number>	Get global parameter (read out module specific settings e.g. communication settings or TMCL™ user variables)
STGP	11	<parameter>, <bank number>	Store global parameter (TMCL™ user variables only)
RSGP	12	<parameter>, <bank number>	Restore global parameter (TMCL™ user variable only)
RFS	13	START STOP STATUS, <motor number>	Reference search
SIO	14	<port number>, <bank number>, <value>	Set digital output to specified value
GIO	15	<port number>, <bank number>	Get value of analogue/digital input
CALC	19	<operation>, <value>	Process accumulator & value
COMP	20	<value>	Compare accumulator <-> value
JC	21	<condition>, <jump address>	Jump conditional
JA	22	<jump address>	Jump absolute
CSUB	23	<subroutine address>	Call subroutine
RSUB	24		Return from subroutine
WAIT	27	<condition>, <motor number>, <ticks>	Wait with further program execution
STOP	28		Stop program execution
SCO	30	<coordinate number>, <motor number>, <position>	Set coordinate
GCO	31	<coordinate number>, <motor number>	Get coordinate
CCO	32	<coordinate number>, <motor number>	Capture coordinate
CALCX	33	<operation>	Process accumulator & X-register
AAP	34	<parameter>, <motor number>	Accumulator to axis parameter
AGP	35	<parameter>, <bank number>	Accumulator to global parameter

TMCL™ CONTROL COMMANDS:

Instruction	Description	Type	Mot/Bank	Value
128 – stop application	a running TMCL™ standalone application is stopped	don't care	don't care	don't care
129 – run application	TMCL™ execution is started (or continued)	0 - run from current address 1 - run from specified address	don't care	don't care starting address
130 – step application	only the next command of a TMCL™ application is executed	don't care	don't care	don't care
131 – reset application	the program counter is set to zero, and the standalone application is stopped (when running or stepped)	don't care	don't care	don't care
132 – start download mode	target command execution is stopped and all following commands are transferred to the TMCL™ memory	don't care	don't care	starting address of the application
133 – quit download mode	target command execution is resumed	don't care	don't care	don't care
134 – read TMCL™ memory	the specified program memory location is read	don't care	don't care	<memory address>
135 – get application status	one of these values is returned: 0 – stop 1 – run 2 – step 3 – reset	don't care	don't care	don't care
136 – get firmware version	return the module type and firmware revision either as a string or in binary format	0 – string 1 – binary	don't care	don't care
137 – restore factory settings	reset all settings stored in the EEPROM to their factory defaults This command does not send back a reply.	don't care	don't care	must be 1234
139 – enter ASCII mode	Enter ASCII command line (see chapter 5.6)	don't care	don't care	don't care

5.6 The ASCII interface

TMCL™ also offers an ASCII interface that can be used to communicate with the module and to send some commands as text strings.

- ***The ASCII command line interface is entered by sending the binary command 139 (enter ASCII mode).***
- Afterwards the commands are entered like in the TMCL-IDE. Please note that only those commands, which can be used in direct mode, also can be entered in ASCII mode.
- ***For leaving the ASCII mode and re-enter the binary mode enter the command BIN.***

5.6.1 Format of the command line

As the first character, the address character has to be sent. The address character is *A* when the module address is 1, *B* for modules with address 2 and so on. After the address character there may be spaces (but this is not necessary). Then, send the command with its parameters. At the end of a command line a <CR> character has to be sent.

Here are some examples for valid command lines:

```
AMVP ABS, 1, 50000
A MVP ABS, 1, 50000
AROL 2, 500
A MST 1
ABIN
```

These command lines would address the module with address 1. To address e.g. module 3, use address character *C* instead of *A*. The last command line shown above will make the module return to binary mode.

5.6.2 Format of a reply

After executing the command the module sends back a reply in ASCII format. This reply consists of:

- the address character of the host (host address that can be set in the module)
- the address character of the module
- the status code as a decimal number
- the return value of the command as a decimal number
- a <CR> character

So, after sending AGAP 0, 1 the reply would be BA 100 -5000 if the actual position of axis 1 is -5000, the host address is set to 2 and the module address is 1. The *value 100* is the *status code 100* that means *command successfully executed*.

5.6.3 Commands that can be used in ASCII mode

The following commands can be used in ASCII mode: ROL, ROR, MST, MVP, SAP, GAP, STAP, RSAP, SGP, GGP, STGP, RSGP, RFS, SIO, GIO, SAC, SCO, GCO, CCO, UF0, UF1, UF2, UF3, UF4, UF5, UF6, and UF7.

There are also special commands that are only available in ASCII mode:

- **BIN:** This command quits ASCII mode and returns to binary TMCL™ mode.
- **RUN:** This command can be used to start a TMCL™ program in memory.
- **STOP:** Stops a running TMCL™ application.

5.6.4 Configuring the ASCII interface

The module can be configured so that it starts up either in binary mode or in ASCII mode. **Global parameter 67 is used for this purpose** (please see also chapter 7.1). Bit 0 determines the startup mode: If this bit is set, the module starts up in ASCII mode, else it will start up in binary mode (default). Bit 4 and Bit 5 determine how the characters that are entered are echoed back. Normally, both bits are set to zero. In this case every

character that is entered is echoed back when the module is addressed. A Character can also be erased using the backspace character (press the backspace key in a terminal program). When bit 4 is set and bit 5 is clear the characters that are entered are not echoed back immediately but the entire line will be echoed back after the <CR> character has been sent. When bit 5 is set and bit 4 is clear there will be no echo, only the reply will be sent. This may be useful in RS485 systems.

5.7 Commands

The module specific commands are explained in more detail on the following pages. They are listed according to their command number.

5.7.1 ROR (rotate right)

With this command the motor will be instructed to rotate with a specified velocity in *right* direction (increasing the position counter).

Internal function: first, velocity mode is selected. Then, the velocity value is transferred to axis parameter #0 (*target velocity*).

The module is based on the TMC428/429 motor controller and the TMC249 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR 0, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
1	don't care	0*	<velocity> 0... 2047

*motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Rotate right, motor 0, velocity = 350

Mnemonic: ROR 0, 350

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$01	\$00	\$02	\$00	\$00	\$01	\$5e	\$61

5.7.2 ROL (rotate left)

With this command the motor will be instructed to rotate with a specified velocity (opposite direction compared to ROR, decreasing the position counter).

Internal function: first, velocity mode is selected. Then, the velocity value is transferred to axis parameter #0 (*target velocity*).

The module is based on the TMC428/429 motor controller and the TMC249 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL 0, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
2	don't care	0*	<velocity> 0... 2047

*motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Rotate left, motor 0, velocity = 1200

Mnemonic: ROL 0, 1200

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$02	\$00	\$00	\$00	\$00	\$04	\$b0	\$b7

5.7.3 MST (motor stop)

With this command the motor will be instructed to stop with deceleration ramp (soft stop). For information about hard stops refer to chapter 9 (hints and tips) please.

Internal function: the axis parameter *target velocity* is set to zero.

Related commands: ROL, ROR, SAP, GAP

Mnemonic: MST 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
3	don't care	0*	don't care

*motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Stop motor

Mnemonic: MST 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$03	\$00	\$00	\$00	\$00	\$00	\$00	\$04

5.7.4 MVP (move to position)

With this command the motor will be instructed to move to a specified relative or absolute position or a pre-programmed coordinate. It will use the acceleration/deceleration ramp and the positioning speed programmed into the unit. This command is non-blocking – that is, a reply will be sent immediately after command interpretation and initialization of the motion controller. Further commands may follow without waiting for the motor reaching its end position. The maximum velocity and acceleration are defined by axis parameters #4 and #5.

Three operation types are available:

- Moving to an absolute position in the range from - 8388608 to +8388607 (-2^{23} to $+2^{23}-1$).
- Starting a relative movement by means of an offset to the actual position. In this case, the new resulting position value must not exceed the above mentioned limits, too.
- Moving the motor to a (previously stored) coordinate (refer to SCO for details).

Please note, that the distance between the actual position and the new one should not be more than 8388607 microsteps. Otherwise the motor will run in the wrong direction for taking a shorter way. If the value is exactly 8388608 the motor maybe stops.

Internal function: a new position value is transferred to the axis parameter #2 target position.

Related commands: SAP, GAP, SCO, CCO, GCO, MST

Mnemonic: MVP <ABS|REL|COORD>, 0, <position|offset|coordinate number>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
4	0 ABS – absolute	0*	<position>
	1 REL – relative	0	<offset>
	2 COORD – coordinate	0	<coordinate number> 0..20

*motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Move motor to (absolute) position 9000

Mnemonic: MVP ABS, 0, 9000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$04	\$00	\$00	\$00	\$00	\$23	\$28	\$50

Example:

Move motor from current position 1000 steps backward (move relative -1000)

Mnemonic: MVP REL, 0, -1000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$04	\$01	\$00	\$ff	\$ff	\$fc	\$18	\$18

Example:

Move motor to previously stored coordinate #8
Mnemonic: MVP COORD, 0, 8

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$04	\$02	\$00	\$00	\$00	\$00	\$08	\$0f

- *When moving to a coordinate, the coordinate has to be set properly in advance with the help of SCO or CCO.*

5.7.5 SAP (set axis parameter)

With this command most of the motion control parameters of the module can be specified. The settings will be stored in SRAM and therefore are volatile. That is, information will be lost after power off. **Please use command STAP (store axis parameter) in order to store any setting permanently.**

Internal function: the parameter format is converted ignoring leading zeros (or ones for negative values). The parameter is transferred to the correct position in the appropriate device.

Related commands: GAP, STAP, RSAP, AAP

Mnemonic: SAP <parameter number>, 0, <value>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
5	<parameter number>	0*	<value>

*motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and values which can be used with this command please refer to chapter 6.

Example:

Set the absolute maximum current of motor to 2A.

Calculate the value with the formula for the RMS current unit. Please note, that there is also another formula which refers to the value of I_{peak}

$$I_{RMS} = <value> \times \frac{2.8A}{255}$$

Mnemonic: SAP 6, 0, 182

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$05	\$06	\$00	\$00	\$00	\$00	\$b6	\$c2

5.7.6 GAP (get axis parameter)

Most parameters of the PDx-113-57/60-SE can be adjusted individually for the axis. With this parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditioned jumps). In direct mode the value read is only put out in the value field of the reply (without affecting the accumulator).

Internal function: the parameter is read out of the correct position in the appropriate device. The parameter format is converted adding leading zeros (or ones for negative values).

Related commands: SAP, STAP, AAP, RSAP

Mnemonic: GAP <parameter number>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
6	<parameter number>	0*	don't care

*motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and values which can be used with this command please refer to chapter 6.

Example:

Get the actual position of motor

Mnemonic: GAP 0, 1

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$06	\$01	\$00	\$00	\$00	\$00	\$00	\$08

Reply:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Host-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	\$64	\$06	\$00	\$00	\$02	\$c7	\$36

⇒ status=no error, position=711

5.7.7 STAP (store axis parameter)

An axis parameter previously set with a *Set Axis Parameter* command (SAP) will be stored permanent. Most parameters are automatically restored after power up.

Internal function: an axis parameter value stored in SRAM will be transferred to EEPROM and loaded from EEPROM after next power up.

Related commands: SAP, RSAP, GAP, AAP

Mnemonic: STAP <parameter number>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
7	<parameter number>	0* ¹	don't care* ²

*¹motor number is always 0 as only one motor is involved

*²the value operand of this function has no effect. Instead, the currently used value (e.g. selected by SAP) is saved.

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and values which can be used with this command please refer to chapter 6.

Example:

Store the maximum positioning speed of motor

Mnemonic: STAP 4, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$07	\$04	\$00	\$00	\$00	\$00	\$00	\$0c

- *The STAP command will not have any effect when the configuration EEPROM is locked (refer to 7.1).*
- *In direct mode, the error code 5 (configuration EEPROM locked, see also section 5.2.1) will be returned in this case.*

5.7.8 RSAP (restore axis parameter)

For all configuration-related axis parameters non-volatile memory locations are provided. By default, most parameters are automatically restored after power up. A single parameter that has been changed before can be reset by this instruction also.

Internal function: the specified parameter is copied from the configuration EEPROM memory to its RAM location.

Relate commands: SAP, STAP, GAP, and AAP

Mnemonic: RSAP <parameter number>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
8	<parameter number>	0*	don't care

*motor number is always 0 as only one motor is involved

Reply structure in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and values which can be used with this command please refer to chapter 6.

Example:

Restore the maximum current of motor 0

Mnemonic: RSAP 6, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$08	\$06	\$00	\$00	\$00	\$00	\$00	\$0f

5.7.9 SGP (set global parameter)

With this command most of the module specific parameters not directly related to motion control can be specified and the TMCL™ user variables can be changed. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables.

All module settings will automatically be stored non-volatile (internal EEPROM of the processor). The TMCL™ user variables will not be stored in the EEPROM automatically, but this can be done by using STGP commands.

Internal function: the parameter format is converted ignoring leading zeros (or ones for negative values). The parameter is transferred to the correct position in the appropriate (on board) device.

Related commands: GGP, STGP, RSGP, AGP

Mnemonic: SGP <parameter number>, <bank number>, <value>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
9	<parameter number>	<bank number>	<value>

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 7.

Example:

Set the serial address of the target device to 3

Mnemonic: SGP 66, 0, 3

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$09	\$42	\$00	\$00	\$00	\$00	\$03	\$4f

5.7.10 GGP (get global parameter)

All global parameters can be read with this function. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables.

Internal function: the parameter is read out of the correct position in the appropriate device. The parameter format is converted adding leading zeros (or ones for negative values).

Related commands: SGP, STGP, RSGP, AGP

Mnemonic: GGP <parameter number>, <bank number>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
10	<parameter number>	<bank number>	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 7.

Example:

Get the serial address of the target device

Mnemonic: GGP 66, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0a	\$42	\$00	\$00	\$00	\$00	\$00	\$4d

Reply:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Host-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	\$64	\$0a	\$00	\$00	\$00	\$01	\$72

⇒ **Status=no error, Value=1**

5.7.11 STGP (store global parameter)

This command is used to store TMCL™ user variables permanently in the EEPROM of the module. Some global parameters are located in RAM memory, so without storing modifications are lost at power down. This instruction enables enduring storing. Most parameters are automatically restored after power up.

Internal function: the specified parameter is copied from its RAM location to the configuration EEPROM.

Related commands: SGP, GGP, RSGP, AGP

Mnemonic: STGP <parameter number>, <bank number>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
11	<parameter number>	<bank number>	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 7.

Example:

Store the serial address of the target device

Mnemonic: STGP 66, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0b	\$42	\$00	\$00	\$00	\$00	\$00	\$4e

- *The STAP command will not have any effect when the configuration EEPROM is locked (refer to 7.1).*
- *In direct mode, the error code 5 (configuration EEPROM locked, see also section 5.2.1) will be returned in this case.*

5.7.12 RSGP (restore global parameter)

With this command the contents of a TMCL™ user variable can be restored from the EEPROM. For all configuration-related axis parameters, non-volatile memory locations are provided. By default, most parameters are automatically restored after power up (see axis parameter list in chapter 0). A single parameter that has been changed before can be reset by this instruction.

Internal function: the specified parameter is copied from the configuration EEPROM memory to its RAM location.

Relate commands: SAP, STAP, GAP, and AAP

Mnemonic: RSAP <parameter number>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
12	<parameter number>	0*	don't care

*motor number is always 0 as only one motor is involved

Reply structure in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 7.

Example:

Restore user variable #42

Mnemonic: RSGP 42, 2

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0c	\$2a	\$02	\$00	\$00	\$00	\$00	\$39

5.7.13 RFS (reference search)

The PDx-113-57/60-SE has a built-in reference search algorithm which can be used. The reference search algorithm provides switching point calibration and three switch modes. The status of the reference search can also be queried to see if it has already finished. (In a TMCL™ program it is better to use the WAIT command to wait for the end of a reference search.) Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (chapter 6). The reference search can be started, stopped, and the actual status of the reference search can be checked.

Internal function: the reference search is implemented as a state machine, so interaction is possible during execution.

Related commands: WAIT

Mnemonic: RFS <START|STOP|STATUS>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
13	0 START – start ref. search 1 STOP – abort ref. search 2 STATUS – get status	0*	don't care

*motor number is always 0 as only one motor is involved

Reply in direct mode:

When using type 0 (START) or 1 (STOP):

STATUS	VALUE
100 – OK	don't care

When using type 2 (STATUS):

STATUS	VALUE	
100 – OK	0	No reference search active
	Other values	Reference search active

Example:

Start reference search of motor 0

Mnemonic: RFS START, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0d	\$00	\$00	\$00	\$00	\$00	\$00	\$0e

With this PANdrive it is possible to use stall detection instead of a reference search. Please refer to section 6.2 for details.

5.7.14 SIO (set output)

This command sets the status of the general digital output either to low (0) or to high (1).

Internal function: the passed value is transferred to the specified output line.

Related commands: GIO, WAIT

Mnemonic: SIO <port number>, <bank number>, <value>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
14	<port number>	<bank number>	<value>

Reply structure:

STATUS	VALUE
100 – OK	don't care

Example:

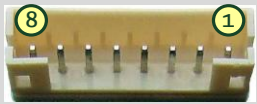
Set OUT_1 to high (bank 2, output 1; general purpose output)

Mnemonic: SIO 1, 2, 1

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0e	\$01	\$02	\$00	\$00	\$00	\$01	\$13

Available I/O ports of PDx-113-57/60-SE:

	Pin	I/O port	Command	Range
	5	OUT_0	SIO 0, 2, <n>	1/0
	6	OUT_1	SIO 1, 2, <n>	1/0

Addressing both output lines with one SIO command:

- Set the type parameter to 255 and the bank parameter to 2.
- The value parameter must then be set to a value between 0...255, where every bit represents one output line.
- Furthermore, the value can also be set to -1. In this special case, the contents of the lower 8 bits of the accumulator are copied to the output pins.

Example:

Set both output pins high.

Mnemonic: SIO 255, 2, 3

The following program will show the states of the input lines on the output lines:

```
Loop: GIO 255, 0
      SIO 255, 2, -1
      JA Loop
```

5.7.15 GIO (get input/output)

With this command the status of the two available general purpose inputs of the module can be read out. The function reads a digital or analogue input port. Digital lines will read 0 and 1, while the ADC channels deliver their 10 bit result in the range of 0... 1023. In standalone mode the requested value is copied to the *accumulator* (accu) for further processing purposes such as conditioned jumps. In direct mode the value is only output in the *value* field of the reply, without affecting the accumulator. The actual status of a digital output line can also be read.

Internal function: the specified line is read.

Related commands: SIO, WAIT

Mnemonic: GIO <port number>, <bank number>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
15	<port number>	<bank number>	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	<status of the port>

Example:

Get the analogue value of ADC channel 1

Mnemonic: GIO 1, 1

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$0f	\$00	\$01	\$00	\$00	\$00	\$00	\$11

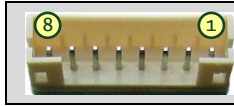
Reply:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Host-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	\$64	\$0f	\$00	\$00	\$01	\$fa	\$71

⇒ value: 506

5.7.15.1 I/O bank 0 – digital inputs:

The ADIN lines can be read as digital or analogue inputs at the same time. The analogue values can be accessed in bank 1.

	Pin	I/O port	Command	Range
	7	IN_0	GIO 0, 0	0/1
	8	IN_1	GIO 1, 0	0/1

Reading all digital inputs with one GIO command:

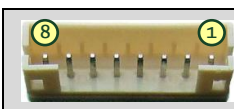
- Set the type parameter to 255 and the bank parameter to 0.
- In this case the status of all digital input lines will be read to the lower eight bits of the accumulator.

Use following program to represent the states of the input lines on the output lines:

```
Loop: GIO 255, 0
      SIO 255, 2,-1
      JA Loop
```

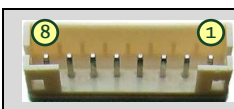
5.7.15.2 I/O bank 1 – analogue inputs:

The ADIN lines can be read back as digital or analogue inputs at the same time. The digital states can be accessed in bank 0.

	Pin	I/O port	Command	Range
	7	IN_0	GIO 0, 1	0... 1023
	8	IN_1	GIO 1, 1	0... 1023

5.7.15.3 I/O bank 2 – the states of digital outputs

The states of the OUT lines (that have been set by SIO commands) can be read back using bank 2.

	Pin	I/O port	Command	Range
	5	OUT_0	GIO 0, 2,	0/1
	6	OUT_1	GIO 1, 2,	0/1

5.7.16 CALC (calculate)

A value in the accumulator variable, previously read by a function such as GAP (get axis parameter), can be modified with this instruction. Nine different arithmetic functions can be chosen and one constant operand value must be specified. The result is written back to the accumulator, for further processing like comparisons or data transfer.

Related commands: CALCX, COMP, JC, AAP, AGP, GAP, GGP, GIO

Mnemonic: CALC <op>, <value>

where <op> is ADD, SUB, MUL, DIV, MOD, AND, OR, XOR, NOT or LOAD

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
19	0 ADD – add to accu 1 SUB – subtract from accu 2 MUL – multiply accu by 3 DIV – divide accu by 4 MOD – modulo divide by 5 AND – logical and accu with 6 OR – logical or accu with 7 XOR – logical exor accu with 8 NOT – logical invert accu 9 LOAD – load operand to accu	don't care	<operand>

Example:

Multiply accu by -5000

Mnemonic: CALC MUL, -5000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$13	\$02	\$00	\$FF	\$FF	\$EC	\$78	\$78

5.7.17 COMP (compare)

The specified number is compared to the value in the accumulator register. The result of the comparison can for example be used by the conditional jump (JC) instruction. This command is intended for use in standalone operation only.

The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. It does not make sense to use this command in direct mode.

Internal function: the specified value is compared to the internal *accumulator*, which holds the value of a preceding *get* or *calculate* instruction (see GAP/GGP/GIO/CALC/CALCX). The internal arithmetic status flags are set according to the comparison result.

Related commands: JC (jump conditional), GAP, GGP, GIO, CALC, CALCX

Mnemonic: COMP <value>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
20	don't care	don't care	<comparison value>

Example:

Jump to the address given by the label when the position of motor is greater than or equal to 1000.

GAP 1, 2, 0 //get axis parameter, type: no. 1 (actual position), motor: 0, value: 0 don't care

COMP 1000 //compare actual value to 1000

JC GE, Label //jump, type: 5 greater/equal, the label must be defined somewhere else in the program

Binary format of the COMP 1000 command:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$14	\$00	\$00	\$00	\$00	\$03	\$e8	\$00

5.7.18 JC (jump conditional)

The JC instruction enables a conditional jump to a fixed address in the TMCL™ program memory, if the specified condition is met. The conditions refer to the result of a preceding comparison. Please refer to COMP instruction for examples. This function is for standalone operation only.

The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. It does not make sense to use this command in direct mode. See the host-only control functions for details.

Internal function: the TMCL™ program counter is set to the passed value if the arithmetic status flags are in the appropriate state(s).

Related commands: JA, COMP, WAIT, CLE

Mnemonic: JC <condition>, <label>

where <condition>=ZE|NZ|EQ|NE|GT|GE|LT|LE|ETO|EAL|EDV|EPO

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
21	0 ZE - zero 1 NZ - not zero 2 EQ - equal 3 NE - not equal 4 GT - greater 5 GE - greater/equal 6 LT - lower 7 LE - lower/equal 8 ETO - time out error 9 EAL - external alarm 12 ESD - shutdown error	don't care	<jump address>

Example:

Jump to address given by the label when the position of motor is greater than or equal to 1000.

```
GAP 1, 0, 0    //get axis parameter, type: no. 1 (actual position), motor: 0, value: 0 don't care
COMP 1000     //compare actual value to 1000
JC GE, Label   //jump, type: 5 greater/equal
...
...
Label: ROL 0, 1000
```

Binary format of "JC GE, Label" when Label is at address 10:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$15	\$05	\$00	\$00	\$00	\$00	\$0a	\$25

5.7.19JA (jump always)

Jump to a fixed address in the TMCL™ program memory. This command is intended for standalone operation only.

Note that the host address and the reply is only used to transfer this instruction to the TMCL™ program memory. This command cannot be used in direct mode.

Internal function: the TMCL™ program counter is set to the passed value.

Related commands: JC, WAIT, CSUB

Mnemonic: JA <Label>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
22	don't care	don't care	<jump address>

Example:

```
An infinite loop in TMCL™
Loop: MVP ABS, 0, 10000
      WAIT POS, 0, 0
      MVP ABS, 0, 0
      WAIT POS, 0, 0
      JA Loop      //Jump to the label Loop
```

Binary format of "JA Loop" assuming that the label "Loop" is at address 20:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$16	\$00	\$00	\$00	\$00	\$00	\$14	\$2b

5.7.20 CSUB (call subroutine)

This function calls a subroutine in the TMCL™ program memory. It is intended for standalone operation only.

Note that the host address and the reply is only used to transfer this instruction to the TMCL™ program memory. This command cannot be used in direct mode.

Internal function: the actual TMCL™ program counter value is saved to an internal stack, afterwards overwritten with the passed value. The number of entries in the internal stack is limited to 8. This also limits nesting of subroutine calls to 8. The command will be ignored if there is no more stack space left.

Related commands: RSUB, JA

Mnemonic: CSUB <Label>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
23	don't care	don't care	<subroutine address>

Example:

```

Call a subroutine
Loop:  MVP ABS, 0, 10000
      CSUB SubW    //Save program counter and jump to label "SubW"
      MVP ABS, 0, 0
      JA Loop

SubW:  WAIT POS, 0, 0
      WAIT TICKS, 0, 50
      RSUB        //Continue with the command following the CSUB command

```

Binary format of the "CSUB SubW" command assuming that the label "SubW" is at address 100:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$17	\$00	\$00	\$00	\$00	\$00	\$64	\$7c

5.7.21 RSUB (return from subroutine)

Return from a subroutine to the command after the CSUB command. This command is intended for use in standalone mode only.

Note that the host address and the reply is only used to transfer this instruction to the TMCL™ program memory. This command cannot be used in direct mode.

Internal function: the TMCL™ program counter is set to the last value of the stack. The command will be ignored if the stack is empty.

Related command: CSUB

Mnemonic: RSUB

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
24	don't care	don't care	don't care

Example: Please see the CSUB example (section 5.7.20).

Binary format of RSUB:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$18	\$00	\$00	\$00	\$00	\$00	\$00	\$19

5.7.22 WAIT (wait for an event to occur)

This instruction interrupts the execution of the TMCL™ program until the specified condition is met. This command is intended for standalone operation only.

Note, that the host address and the reply is only used to transfer this instruction to the TMCL™ program memory. This command is not to be used in direct mode.

There are five different wait conditions that can be used:

- Ticks: Wait until the number of timer ticks specified by the <ticks> parameter has been reached.
- POS: Wait until the target position of the motor specified by the <motor> parameter has been reached. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- REFSW: Wait until the reference switch of the motor specified by the <motor> parameter has been triggered. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- LIMSW: Wait until a limit switch of the motor specified by the <motor> parameter has been triggered. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- RFS: Wait until the reference search of the motor specified by the <motor> field has been reached. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.

The timeout flag (ETO) will be set after a timeout limit has been reached. You can then use a JC ETO command to check for such errors or clear the error using the CLE command.

Internal function: the TMCL™ program counter is held until the specified condition is met.

Related commands: JC, CLE

Mnemonic: WAIT <condition>, 0, <ticks>
where <condition> is TICKS|POS|REFSW|LIMSW|RFS

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
27	0 TICKS - timer ticks* ¹	don't care	<no. of ticks* ² >
	1 POS - target position reached	0* ¹	<no. of ticks* for timeout>, 0 for no timeout
	2 REFSW – reference switch	0	<no. of ticks* for timeout>, 0 for no timeout
	3 LIMSW – limit switch	0	<no. of ticks* for timeout>, 0 for no timeout
	4 RFS – reference search completed	0	<no. of ticks* for timeout>, 0 for no timeout

*¹ motor number is always 0 as only one motor is involved

*² one tick is 10 milliseconds (in standard firmware)

Example:

Wait for motor 0 to reach its target position, without timeout

Mnemonic: WAIT POS, 0, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$1b	\$01	\$00	\$00	\$00	\$00	\$00	\$1d

5.7.23 STOP (stop TMCL™ program execution)

This function stops executing a TMCL™ program. The host address and the reply are only used to transfer the instruction to the TMCL™ program memory.

This command should be placed at the end of every standalone TMCL™ program. It is not to be used in direct mode.

Internal function: TMCL™ instruction fetching is stopped.

Related commands: none

Mnemonic: STOP

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
28	don't care	don't care	don't care

Example:

Mnemonic: STOP

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$1c	\$00	\$00	\$00	\$00	\$00	\$00	\$1d

5.7.24 SCO (set coordinate)

Up to 20 position values (coordinates) can be stored for every axis for use with the MVP COORD command. This command sets a coordinate to a specified value.

Please note that the coordinate number 0 is always stored in RAM only. All others are also stored in the EEPROM.

Internal function: the passed value is stored in the internal position array.

Related commands: GCO, CCO, MVP

Mnemonic: SCO <coordinate number>, 0, <position>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
30	<coordinate number> 0...20	0*	<position> $-2^{23} \dots +2^{23}$

* motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Set coordinate #1 of motor to 1000

Mnemonic: SCO 1, 0, 1000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$1e	\$01	\$00	\$00	\$00	\$03	\$e8	\$0b

5.7.25 GCO (get coordinate)

This command makes possible to read out a previously stored coordinate. In standalone mode the requested value is copied to the accumulator register for further processing purposes such as conditioned jumps. In direct mode, the value is only output in the value field of the reply, without affecting the accumulator.

Please note that the coordinate number 0 is always stored in RAM only. All others are also stored in the EEPROM.

Internal function: the desired value is read out of the internal coordinate array, copied to the accumulator register and -in direct mode- returned in the *value* field of the reply.

Related commands: SCO, CCO, MVP

Mnemonic: GCO <coordinate number>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
31	<coordinate number> 0...20	0*	don't care

* motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Get motor value of coordinate 1

Mnemonic: GCO 1, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$1f	\$01	\$00	\$00	\$00	\$00	\$00	\$21

Reply:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	\$64	\$0a	\$00	\$00	\$00	\$00	\$71

⇒ **Value: 0**

5.7.26 CCO (capture coordinate)

The actual position of the axis is copied to the selected coordinate variable.

Please note that the coordinate number 0 is always stored in RAM only. All others are also stored in the EEPROM.

Internal function: the selected (24 bit) position values are written to the 20 by 3 bytes wide coordinate array.

Related commands: SCO, GCO, MVP

Mnemonic: CCO <coordinate number>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
32	<coordinate number> 0...20	0*	don't care

* motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Store current position of the axe to coordinate 3

Mnemonic: CCO 3, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$20	\$03	\$00	\$00	\$00	\$00	\$00	\$24

5.7.27 CALCX (calculate using the X register)

This instruction is very similar to CALC, but the second operand comes from the X register. The X register can be loaded with the LOAD or the SWAP type of this instruction. The result is written back to the accumulator for further processing like comparisons or data transfer.

Related commands: CALC, COMP, JC, AAP, AGP

Mnemonic: CALCX <operation>

with <operation>=ADD|SUB|MUL|DIV|MOD|AND|OR|XOR|NOT|LOAD|SWAP

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
33	0 ADD – add X register to accu 1 SUB – subtract X register from accu 2 MUL – multiply accu by X register 3 DIV – divide accu by X-register 4 MOD – modulo divide accu by x-register 5 AND – logical and accu with X-register 6 OR – logical or accu with X-register 7 XOR – logical exor accu with X-register 8 NOT – logical invert X-register 9 LOAD – load accu to X-register 10 SWAP – swap accu with X-register	don't care	don't care

Example:

Multiply accu by X-register

Mnemonic: CALCX MUL

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$21	\$02	\$00	\$00	\$00	\$00	\$00	\$24

5.7.28 AAP (accumulator to axis parameter)

The content of the accumulator register is transferred to the specified axis parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction.

Related commands: AGP, SAP, GAP, SGP, GGP, GIO, GCO, CALC, CALCX

Mnemonic: AAP <parameter number>, 0

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
34	<parameter number>	0*	<don't care>

* motor number is always 0 as only one motor is involved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and values which can be used with this command please refer to chapter 6.

Example:

Positioning motor by a potentiometer connected to the analogue input #0:

```

Start:  GIO 0,1      // get value of analogue input line 0
        CALC MUL, 4  // multiply by 4
        AAP 0,0      // transfer result to target position of motor 0
        JA Start     // jump back to start

```

Binary format of the AAP 0,0 command:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$22	\$00	\$00	\$00	\$00	\$00	\$00	\$23

5.7.29 AGP (accumulator to global parameter)

The content of the accumulator register is transferred to the specified global parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction. **Note that the global parameters in bank 0 are EEPROM-only and thus should not be modified automatically by a standalone application.** (See chapter 0 for a complete list of global parameters).

Related commands: AAP, SGP, GGP, SAP, GAP, GIO

Mnemonic: AGP <parameter number>, <bank number>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
35	<parameter number>	<bank number>	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

For a table with parameters and values which can be used with this command please refer to chapter 7.

Example:

Copy accumulator to TMCL™ user variable #3

Mnemonic: AGP 2, 2

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$23	\$03	\$02	\$00	\$00	\$00	\$00	\$29

5.7.30 CLE (clear error flags)

This command clears the internal error flags.

It is intended for use in standalone mode only and must not be used in direct mode.

The following error flags can be cleared by this command (determined by the <flag> parameter):

- ALL: clear all error flags.
- ETO: clear the timeout flag.
- EAL: clear the external alarm flag
- EDV: clear the deviation flag
- EPO: clear the position error flag

Related commands: JC

Mnemonic: CLE <flags>

where <flags>=ALL|ETO|EDV|EPO

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
36	0 – (ALL) all flags 1 – (ETO) timeout flag 2 – (EAL) alarm flag 3 – (EDV) deviation flag 4 – (EPO) position flag 5 – (ESD) shutdown flag	don't care	don't care

Example:

Reset the timeout flag

Mnemonic: CLE ETO

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$24	\$01	\$00	\$00	\$00	\$00	\$00	\$26

5.7.31 Customer specific TMCL™ command extension (UF0... UF7 / user function)

The user definable functions UF0 through UF7 are predefined, *empty* functions for user specific purposes. Contact TRINAMIC for customer specific programming of these functions.

Internal function: call user specific functions implemented in **C** by TRINAMIC.

Related commands: none

Mnemonic: UF0... UF7

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
64... 71	(user defined)	(user defined)	(user defined)

Reply in direct mode:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	user defined	64...71	user defined	user defined	user defined	user defined	<checksum>

5.7.32 Request target position reached event

This command is the only exception to the TMCL™ protocol, as it sends two replies: One immediately after the command has been executed (like all other commands also), and one additional reply that will be sent when the motor has reached its target position. ***This instruction can only be used in direct mode (in standalone mode, it is covered by the WAIT command) and hence does not have a mnemonic.***

Internal function: send an additional reply when the motor has reached its target position

Mnemonic: ---

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
138	don't care	don't care	0*

* motor number

Reply in direct mode (right after execution of this command):

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	100	138	\$00	\$00	\$00	Motor bit mask	<checksum>

Additional reply in direct mode (after motors have reached their target positions):

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$02	\$01	128	138	\$00	\$00	\$00	Motor bit mask	<checksum>

5.7.33 BIN (return to binary mode)

This command can only be used in ASCII mode. It quits the ASCII mode and returns to binary mode.

Related Commands: none

Mnemonic: BIN

Binary representation: this command does not have a binary representation as it can only be used in ASCII mode.

5.7.34 TMCL™ control functions

The following functions are for host control purposes only and are not allowed for standalone mode. In most cases, there is no need for the customer to use one of those functions (except command 139). They are mentioned here only for reasons of completeness. These commands have no mnemonics, as they cannot be used in TMCL™ programs. The Functions are to be used only by the TMCL-IDE to communicate with the module, for example to download a TMCL™ application into the module.

The only control commands that could be useful for a user host application are:

- *get firmware revision* (command 136, please note the special reply format of this command, described at the end of this section)
- *run application* (command 129)

All other functions can be achieved by using the appropriate functions of the TMCL-IDE.

Instruction	Description	Type	Mot/Bank	Value
128 – stop application	a running TMCL™ standalone application is stopped	don't care	don't care	don't care
129 – run application	TMCL™ execution is started (or continued)	0 - run from current address 1 - run from specified address	don't care	don't care starting address
130 – step application	only the next command of a TMCL™ application is executed	don't care	don't care	don't care
131 – reset application	the program counter is set to zero, and the standalone application is stopped (when running or stepped)	don't care	don't care	don't care
132 – start download mode	target command execution is stopped and all following commands are transferred to the TMCL™ memory	don't care	don't care	starting address of the application
133 – quit download mode	target command execution is resumed	don't care	don't care	don't care
134 – read TMCL™ memory	the specified program memory location is read	don't care	don't care	<memory address>
135 – get application status	one of these values is returned: 0 – stop 1 – run 2 – step 3 – reset	don't care	don't care	don't care
136 – get firmware version	return the module type and firmware revision either as a string or in binary format	0 – string 1 – binary	don't care	don't care
137 – restore factory settings	reset all settings stored in the EEPROM to their factory defaults This command does not send back a reply.	don't care	don't care	must be 1234
139 – enter ASCII mode	Enter ASCII command line (see chapter 5.6)	don't care	don't care	don't care

Special reply format of command 136:**Type set to 0 - reply as a string:**

Byte index	Contents
1	Host Address
2... 9	Version string (8 characters, e.g. 113V2.50)

- There is no checksum in this reply format!

Type set to 1 - version number in binary format:

- Please use the normal reply format.
- The version number is output in the *value* field of the reply in the following way:

Byte index in value field	Contents
1	Version number, low byte
2	Version number, high byte
3	Type number, low byte (currently not used)
4	Type number, high byte (currently not used)

6 Axis parameters

The following sections describe all axis parameters that can be used with the SAP, GAP, AAP, STAP and RSAP commands.

MEANING OF THE LETTERS IN COLUMN ACCESS

Access type	Related command	Description
R	GAP	Parameter readable
W	SAP, AAP	Parameter writable
E	STAP, RSAP	Parameter automatically restored from EEPROM after reset or power-on. These parameters can be stored permanently in EEPROM using STAP command and also explicitly restored (copied back from EEPROM into RAM) using RSAP

 *Basic parameters. Parameters should be adjusted to motor / application for proper module operation. Please refer to chapter 6.1 also for more details regarding calculation of parameters.*

AXIS PARAMETERS OF PDX-113-57/60-SE AND TCM-113-60-SE

Number	Axis Parameter	Description	Range [Unit]	Access
0	target (next) position	The desired position in position mode.	$\pm 2^{23}$ [μsteps]	RW
1	actual position	The current position of the motor. Should only be overwritten for reference point setting.	$\pm 2^{23}$ [μsteps]	RW
2	target (next) speed	The desired speed in velocity mode. In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest.	$\pm 2047 \cdot \frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}$	RW
3	actual speed	The current rotation speed.	$\pm 2047 \cdot \frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}$	RW
4	maximum positioning speed	Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet (p.24) for calculation of physical units.	$0 \dots 2047 \cdot \frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}$	RWE
5	maximum acceleration	The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no.137), which is done automatically. See TMC 428 datasheet (p.24) for calculation of physical units.	$0 \dots 2047$ *)	RWE
6	absolute max. current	The most important motor setting, since too high values might cause motor damage! The maximum value is 255 (which mean 100% of the maximum current of the module). <i>Only PDX-113-57/60-SE and TCM-113-60-SE with pcb version 1.2 or newer support specified max. current over full temperature range. Older or pre-series versions may be limited with regard to temperature range and/or max. motor current.</i>	$0 \dots 255$ $I_{\text{peak}} = < \text{value} > \times \frac{4\text{A}}{255}$ $I_{\text{RMS}} = < \text{value} > \times \frac{2.8\text{A}}{255}$	RWE

Number	Axis Parameter	Description	Range [Unit]	Access
7	standby current	The current limit two seconds after the motor has stopped.	0... 255 $I_{peak} = < value > \times \frac{4A}{255}$ $I_{RMS} = < value > \times \frac{2.8A}{255}$	RWE
8	target pos. reached	Indicates that the actual position equals the target position.	0/1	R
9	ref. switch status	The logical state of the reference (left) switch. See the TMC 428 data sheet for the different switch modes. Default is two switch modes: the left switch as the reference switch, the right switch as a limit (stop) switch.	0/1	R
10	right limit switch status	The logical state of the (right) limit switch.	0/1	R
11	left limit switch status	The logical state of the left limit switch (in three switch mode)	0/1	R
12	right limit switch disable	if set, deactivates the stop function of the right switch	0/1	RWE
13	left limit switch disable	Deactivates the stop function of the left switch resp. reference switch if set.	0/1	RWE
130	minimum speed	Should always be set 1 to ensure exact reaching of the target position. Do not change!	0... 2047 $\left[\frac{\text{max. module current}}{255} \right]$	RWE
135	actual acceleration	The current acceleration (read only).	0... 2047*	R
138	ramp mode	Automatically set when using ROR, ROL, MST and MVP. 0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided. 2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter <i>target speed</i> is changed. For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected.	0/1/2	RWE

Number	Axis Parameter	Description	Range [Unit]	Access
140	microstep resolution	0 full step*)	0... 6	RWE
		1 half step*)		
		2 4 microsteps		
		3 8 microsteps		
		4 16 microsteps		
		5 32 microsteps**)		
		6 64 microsteps**)		
		Modifying this parameter effects the rotation speed in the same relation: *) The full-step setting and the half-step setting are not optimized for use without an adapted microstepping table. The settings step through the microstep table in steps of 64 respectively 32. To get real full stepping use axis parameter 211 or load an adapted microstepping table. **) If the module is specified for 16 microsteps only, switching to 32 or 64 microsteps brings an enhancement in resolution and smoothness. The position counter will use the full resolution, but, however, the motor will resolve a maximum of 24 different microsteps only for the 32 or 64 microstep units.		
141	ref. switch tolerance	For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop.	0... 4095 [μsteps]	RW
149	soft stop flag	If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit.	0/1	RWE
153	ramp divisor	The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one).	0... 13	RWE
154	pulse divisor	The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one).	0... 13	RWE
193	referencing mode	1 – Only the left reference switch is searched. 2 – The right switch is searched, and then the left switch is searched. 3 – Three-switch-mode: the right switch is searched first, and then the reference switch will be searched.	1/2/3	RWE
194	referencing search speed	For the reference search this value specifies the search speed as a fraction of the maximum velocity: 0 – full speed 1 – half of the maximum speed 2 – a quarter of the maximum speed 3 – 1/8 of the maximum speed (etc.)	0... 8	RWE
195	referencing switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.	0... 8	RWE

Number	Axis Parameter	Description	Range [Unit]	Access
203	mixed decay threshold	If the actual velocity is above this threshold, mixed decay will be used. This can also be set to -1 which turns on mixed decay permanently also in the rising part of the microstep wave. This can be used to fix microstep errors.	0... 2048 or -1 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RWE
204	freewheeling	Time after which the power to the motor will be cut when its velocity has reached zero	0... 65535 0 = never [msec]	RWE
205	stall detection threshold	Stall detection threshold. Set it to 0 for no stall detection or to a value between 1 (low threshold) and 7 (high threshold). The motor will be stopped if the load value exceeds the stall detection threshold. Switch off mixed decay to get usable results.	0... 7	RWE
206	actual load value	Readout of the actual load value used for stall detection.	0... 7	R
208	driver error flags	TMC249 error flags. Read only!		R
209	encoder position	Position of motor axis within one rotation	0... 255 [encoder steps]	RW
210	encoder prescaler	Prescaler for the encoder (since firmware version 3.38). For more information refer to 0 please.		RWE
211	fullstep threshold	When exceeding this speed the driver will switch to real full step mode. To disable this feature set this parameter to zero or to a value greater than 2047. Setting a full step threshold allows higher motor torque of the motor at higher velocity. When experimenting with this in a given application, try to reduce the motor current in order to be able to reach a higher motor velocity!	0... 2048 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RWE
212	maximum encoder deviation	When the actual position (#1) and the encoder position (#209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero.	0... 65535 [encoder steps]	RWE
214	power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec).	1... 65535 [10msec]	RWE
215	absolute encoder value	Absolute value of the encoder.	0... 255 [encoder steps]	R

*) Unit of acceleration: $\frac{16\text{MHz}^2}{536870912 \cdot 2^{\text{puls_divisor} + \text{ramp_divisor}} \frac{\text{microsteps}}{\text{sec}^2}}$



Please use the TMCL-IDE axis parameter calculation tool for getting best values.

6.1 Calculation: velocity and acceleration vs. microstep and fullstep frequency

The values of the parameters sent to the TMC428/429 do not have typical motor values like rotations per second as velocity. But these values can be calculated from the TMC428/429-parameters as shown in this document.

PARAMETERS OF TMC428/429

Signal	Description	Range
f_{CLK}	Clock-frequency	16 MHz
velocity	-	0... 2047
a_max	Maximum acceleration	0... 2047
pulse_div	Divider for the velocity. The higher the value, the less is the maximum velocity Default value = 0	0... 13
ramp_div	Divider for the acceleration. The higher the value, the less is the maximum acceleration Default value = 0	0... 13
Usrs	Microstep resolution (microsteps per fullstep = 2^{Usrs})	0... 6

Table 6.1: TMC428/429 velocity parameters

6.1.1 Microstep frequency

The **microstep frequency** of the stepper motor is calculated with

$$\mu sf[Hz] = \frac{f_{CLK}[Hz] \cdot velocity}{2^{pulse_div} \cdot 2048 \cdot 32} \quad \text{with } \mu sf: \text{microstep-frequency}$$

6.1.2 Fullstep frequency

To calculate the **fullstep frequency** from the microstep frequency, the microstep frequency must be divided by the number of microsteps per fullstep.

$$fsf[Hz] = \frac{\mu sf[Hz]}{2^{Usrs}} \quad \text{with } fsf: \text{fullstep-frequency}$$

The change in the pulse rate per time unit (pulse frequency change per second; the **acceleration a** is given by

$$a = \frac{f_{CLK}^2 \cdot a_{max}}{2^{pulse_div + ramp_div + 29}}$$

This results in acceleration in fullsteps of:

$$af = \frac{a}{2^{Usrs}} \quad \text{with } af: \text{acceleration in fullsteps}$$

Example:

Signal	Value
f _{CLK}	16 MHz
velocity	1000
a_max	1000
pulse_div	1
ramp_div	1
usrs	6

$$\mu sf = \frac{16 \text{ MHz} \cdot 1000}{2^1 \cdot 2048 \cdot 32} = \underline{\underline{12207031 \text{ Hz}}}$$

$$fsf[\text{Hz}] = \frac{12207031}{2^6} = \underline{\underline{190734 \text{ Hz}}}$$

$$a = \frac{(16 \text{ Mhz})^2 \cdot 1000}{2^{1+1+29}} = \underline{\underline{119.21 \frac{\text{MHz}}{\text{s}}}}$$

$$af = \frac{119.21 \frac{\text{MHz}}{\text{s}}}{2^6} = \underline{\underline{1.863 \frac{\text{MHz}}{\text{s}}}}$$

Calculation of the number of rotations:

A stepper motor has e.g. 72 fullsteps per rotation.

$$RPS = \frac{fsf}{fullstepsper rotation} = \frac{190734}{72} = 26.49$$

$$RPM = \frac{fsf \cdot 60}{fullstepsper rotation} = \frac{190734 \cdot 60}{72} = 158946$$

6.2 Stall detection

The module is equipped with TMC249 motor driver chips. These chips feature load measurement that can be used for stall detection. The stallGuard™ function effects, that the motor will be stopped when the load gets too high. It is controlled by axis parameter #205.

- If this parameter is set to a value between 1 and 7 the stall detection will be activated.
- Setting it to 0 means that stall detection is turned off.
- A greater value means a higher threshold. This also depends on the motor and on the velocity.
- There is no stall detection while the motor is being accelerated or decelerated.

STALLGUARD™ RELATED AXIS PARAMETERS OF PDx-113-57/60-SE AND TCM-113-60-SE

Number	Axis Parameter	Description	Range [Unit]	Access
205	stall detection threshold	Stall detection threshold. Set it to 0 for no stall detection or to a value between 1 (low threshold) and 7 (high threshold). The motor will be stopped if the load value exceeds the stall detection threshold. Switch off mixed decay to get usable results.	0... 7	RWE
206	actual load value	Readout of the actual load value used for stall detection.	0... 7	R

6.2.1 Finding the reference point with stall detection

Stall detection can be used for finding the reference point. You can do this by using the following TMCL™ code:

```

SAP 205, 0, 5    //Turn on Stall Detection (use other threshold if needed)
ROL 0, 500      //Let the motor run (or use ROR or other velocity)
Loop: GAP 3, 0
COMP 0
JC NE, Loop     //Wait until the motor has stopped
SAP 1, 0, 0     //Set this position as the zero position

```

Do not use RFS in this case.

Mixed decay should be switched off when stallGuard™ operational in order to get usable results.

6.3 Encoder parameters

The PD-113-57/60-SE PANdrive™ is a full mechatronic solution including a NEMA 23 (flange size 57mm) or NEMA 24 (flange size 60mm) stepper motor, a motion controller/driver and a sensOstep™ magnetic encoder (max. 256 increments per rotation), e.g. for step-loss detection under all operating conditions.

For the operation with encoder please consider the following hints:

- The encoder counter can be read by software and can be used to control the exact position of the motor. This also makes closed loop operation possible.
- To read out or to change the position value of the encoder, axis parameter #209 is used. So, to read out the position of your encoder 0 use *GAP 209, 0*.
- The position values can also be changed using command *SAP 209, 0, <n>*, with $n = \pm 0, 1, 2...$
- To change the encoder settings, axis parameter #210 is used. For changing the prescaler of the encoder 0 use *SAP 210, 0, <p>*.

ENCODER RELATED AXIS PARAMETERS OF PDX-113-57/60-SE AND TCM-113-60-SE

Number	Axis Parameter	Description	Range [Unit]	Access
209	encoder position	Position of motor axis within one rotation	0... 255 [encoder steps]	RW
210	encoder prescaler	Prescaler for the encoder (since firmware version 3.38)		RWE
215	absolute encoder value	Absolute value of the encoder	0... 255 [encoder steps]	R

6.3.1 Selecting the prescaler

To select a prescaler, the following values can be used for <p>.

Value for <p>	Resulting prescaler	SAP command for motor 0 SAP 210, 0, <p>	Suggestion for microstep solution (setting of axis parameter 140)
6400	12.5 (default)	SAP 210, 0, 6400	6 (64 micro steps)
3200	6.25	SAP 210, 0, 3200	5 (32 micro steps)
1600	3.125	SAP 210, 0, 1600	4 (16 micro steps)
800	1.5625	SAP 210, 0, 800	3 (8 micro steps)
400	0.78125	SAP 210, 0, 400	2 (4 micro steps)
200	0.390625	SAP 210, 0, 200	1 (2 micro steps)

The table above just shows a subset of those prescalers that can be selected. Also other values between those given in the table can be used. Only the values 1, 2, 4, and 16 must not be used for <p> (because they are needed to select the special encoder function below or rather are reserved for intern usage).

Consider the following formula for your calculation:

$$\text{Prescaler} = \frac{p}{512}$$

Example: <p> = 6400
6400/512 = 12.5 (prescaler)

There is one special function that can also be configured using <p>. To select it just add the following value to <p>:

Adder for <p>	SAP command for motor 0 SAP 210, M0, <p>
4	Clear encoder with next null channel event

Add up both <p> values from these tables to get the required value for the SAP 210 command. The resulting prescaler is Value/512.

6.4 Reference search and homing mode

The built-in reference search features switching point calibration. There is one reference switch. The internal operation is based on an individual state machine that can be started, stopped and monitored (instruction RFS, no. 13). The settings of the automatic stop function corresponding to the switch (axis parameters 12 and 13) do not have any influence on the reference search.

REFERENCE SEARCH RELATED AXIS PARAMETERS OF PDX-113-57/60-SE AND TCM-113-60-SE

Number	Axis Parameter	Description	Range [Unit]	Access
9	ref. switch status	The logical state of the reference (left) switch. See the TMC 428 data sheet for the different switch modes. Default is two switch modes: the left switch as the reference switch, the right switch as a limit (stop) switch.	0/1	R
10	right limit switch status	The logical state of the (right) limit switch.	0/1	R
11	left limit switch status	The logical state of the left limit switch (in three switch mode)	0/1	R
12	right limit switch disable	if set, deactivates the stop function of the right switch	0/1	RWE
13	left limit switch disable	Deactivates the stop function of the left switch resp. reference switch if set.	0/1	RWE
193	referencing mode	1 – Only the left reference switch is searched. 2 – The right switch is searched, and then the left switch is searched. 3 – Three-switch-mode: the right switch is searched first, and then the reference switch will be searched.	1/2/3	RWE
194	referencing search speed	For the reference search this value specifies the search speed as a fraction of the maximum velocity: 0 – full speed 1 – half of the maximum speed 2 – a quarter of the maximum speed 3 – 1/8 of the maximum speed (etc.)	0... 8	RWE
195	referencing switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.	0... 8	RWE

- Selecting the referencing mode (axis parameter 193): in modes 1 and 2, the motor will start by moving *left* (negative position counts). In mode 3 (three-switch mode), the right stop switch is searched first to distinguish the left stop switch from the reference switch by the order of activation when moving left (reference switch and left limit switch share the same electrical function).
- Until the reference switch is found for the first time, the searching speed is identical to the maximum positioning speed (axis parameter 4), unless reduced by axis parameter 194.
- After hitting the reference switch, the motor slowly moves right until the switch is released. Finally the switch is re-entered in left direction, setting the reference point to the center of the two switching points. This low calibrating speed is a quarter of the maximum positioning speed by default (axis parameter 195).
- In **Figure 6.2** the connection of the left and the right limit switch is shown. **Figure 6.3** shows the connection of three switches as left and right limit switch and a reference switch for the reference point. The reference switch is connected in series with the left limit switch. The differentiation between the left limit switch and the reference switch is made through software. Switches with open contacts (normally closed) are used.
- In circular systems there are no end points and thus only one reference switch is used for finding the reference point.

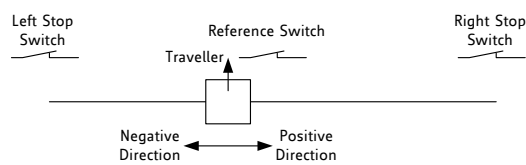


Figure 6.1: Definition of the switches

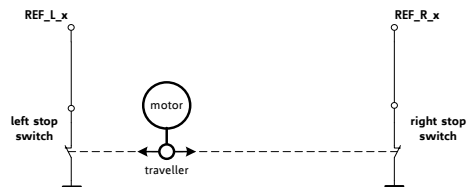


Figure 6.2: Two limit switches

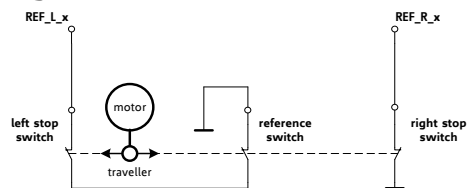


Figure 6.3: Limit switches with extra reference switch

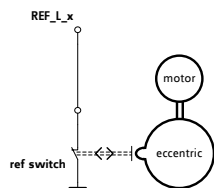


Figure 6.4: Circular system

7 Global parameters

The global parameters apply for all types of TCM modules.

They are grouped into 3 banks:

- bank 0 (global configuration of the module)
- bank 1 (normally not available; for customer specific extensions of the firmware)
- bank 2 (user TMCL™ variables)

Please use SGP and GGP commands to write and read global parameters.

7.1 Bank 0

Parameters with numbers from 64 on configure stuff like the serial address of the module RS232/RS485 baud rate or the CAN bit rate. Change these parameters to meet your needs. The best and easiest way to do this is to use the appropriate functions of the TMCL-IDE. The parameters with numbers between 64 and 128 are stored in EEPROM only.

An SGP command on such a parameter will always store it permanently and no extra STGP command is needed.

Take care when changing these parameters, and use the appropriate functions of the TMCL-IDE to do it in an interactive way.

Note: The PDX-113-57/60-SE does not have the parameters 0...38. They are used for modules which address more than one motor.

Meaning of the letters in column Access:

- R = readable (GGP)
- W = writeable (SGP)
- E = automatically restored from EEPROM after reset or power-on.

Number	Global parameter	Description			Range	Access
64	EEPROM magic	Setting this parameter to a different value as \$E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration.			0...255	RWE
65	RS232/RS485 baud rate	0	9600 baud	Default	0...7	RWE
		1	14400 baud			
		2	19200 baud			
		3	28800 baud			
		4	38400 baud			
		5	57600 baud			
		6	76800 baud	Not supported by Windows!		
		7	(115200 baud)			
66	Serial address	The module (target) address for RS232/RS485.			0...255	RWE
67	ASCII mode	Configure the TMCL™ ASCII interface: Bit 0: 0 – start up in binary (normal) mode 1 – start up in ASCII mode Bits 4 and 5: 00 – Echo back each character 01 – Echo back complete command 10 – Do not send echo, only send command reply				RWE

Number	Global parameter	Description	Range	Access
69	CAN bit rate	1 – 10kBit/s 2 – 20kBit/s 3 – 50kBit/s 4 – 100kBit/s 5 – 125kBit/s 6 – 250kBit/s (default) 7 – 500kBit/s 8 – 1000kBit/s	1..7	RWE
70	CAN reply ID	The CAN ID for replies from the board (default: 2)	0..7ff	RWE
71	CAN ID	The module (target) address for CAN (default: 1)	0..7ff	RWE
73	Configuration EEPROM lock flag	Write: 1234 to lock the EEPROM, 4321 to unlock it. Read: 1=EEPROM locked, 0=EEPROM unlocked.	0/1	RWE
75	Telegram pause time	Pause time before the reply via RS232 or RS485 is sent. For RS232 set to 0. For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin). For CAN interface this parameter has no effect!	0..255	RWE
76	Serial host address	Host address used in the reply telegrams sent back via RS232 or RS485.	0..255	RWE
77	Auto start mode	0: Do not start TMCL™ application after power up (default). 1: Start TMCL™ application automatically after power up.	0/1	RWE
80	Shutdown pin functionality	Select the functionality of the SHUTDOWN pin 0 – no function 1 – high active 2 – low active	0..2	RWE
81	TMCL™ code protection	Protect a TMCL™ program against disassembling or overwriting. 0 – no protection 1 – protection against disassembling 2 – protection against overwriting 3 – protection against disassembling and overwriting <i>If you switch off the protection against disassembling, the program will be erased first! Changing this value from 1 or 3 to 0 or 2, the TMCL™ program will be wiped off.</i>	0,1,2,3	RWE
128	TMCL™ application status	0 – stop 1 – run 2 – step 3 – reset	0..3	R
129	Download mode	0 – normal mode 1 – download mode	0/1	R
130	TMCL™ program counter	The index of the currently executed TMCL™ instruction.		R
132	Tick timer	A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value.		RW

7.2 Bank 1

The global parameter bank 1 is normally not available. It may be used for customer specific extensions of the firmware. Together with user definable commands (see section 7.3) these variables form the interface between extensions of the firmware (written in C) and TMCL™ applications.

7.3 Bank 2

Bank 2 contains general purpose 32 bit variables for the use in TMCL™ applications. They are located in RAM and can be stored to EEPROM. After booting, their values are automatically restored to the RAM.

Up to, 56 user variables are available.

Meaning of the letters in column Access:

- R = readable (GGP)
- W = writeable (SGP)
- E = automatically restored from EEPROM after reset or power-on.

Number	Global parameter	Description	Range	Access
0	General purpose variable #0	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
1	General purpose variable #1	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
2	General purpose variable #2	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
3	General purpose variable #3	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
4	General purpose variable #4	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
5	General purpose variable #5	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
6	General purpose variable #6	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
7	General purpose variable #7	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
8	General purpose variable #8	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
9	General purpose variable #9	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
10	General purpose variable #10	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
11	General purpose variable #11	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
12	General purpose variable #12	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
13	General purpose variable #13	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
14	General purpose variable #14	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
15	General purpose variable #15	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
16	General purpose variable #16	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
17	General purpose variable #17	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
18	General purpose variable #18	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
19	General purpose variable #19	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE
20..55	General purpose variables #20..#55	for use in TMCL™ applications	$-2^{31} \dots +2^{31}$	RWE

8 Hints and tips

This chapter gives some hints and tips on using the functionality of TMCL™, for example how to use and parameterize the built-in reference point search algorithm or the incremental encoder interface.

8.1 Fixing microstep errors

Due to the *zero crossing problem* of the TMC249 stepper motor drivers, microstep errors may occur with some motors as the minimum motor current that can be reached is slightly higher than zero (depending on the inductivity, resistance and supply voltage of the motor).

This can be solved by setting the *mixed decay threshold* parameter (axis parameter 203) to the value -1. This switches on mixed decay permanently, in every part of the microstepping waveform. Now the minimum reachable motor current is always near zero which gives better microstepping results.

A further optimization is possible by adapting the motor current shape and the zero offset of the wave. (For further information about TMCL-IDE please refer to the TMCL™ reference and programming manual.)

Use SAP 203, <motor number>, -1 to turn on this feature.

8.2 Using the RS485 interface

8.2.1 Using RS485 with converter

With most RS485 converters that can be attached to the COM port of a PC the data direction is controlled by the RTS pin of the COM port. Please note that this will only work with Windows 2000, Windows XP, Windows NT4 or Vista, not with Windows 95, Windows 98 or Windows ME (due to a bug in these operating systems). Another problem is that Windows 2000/XP/NT4 switches the direction back to *receive* too late. To overcome this problem, set the *telegram pause time* (global parameter #75) of the module to 15 (or more if needed) by issuing an *SGP 75, 0, 15* command in direct mode. The parameter will automatically be stored in the configuration EEPROM.

8.2.2 Using RS485 with USB-2-X interface converter

Another possibility offers the TRINAMIC USB-2-X device. It is an interface converter that is equipped with a CAN interface, an IIC interface, a LIN interface, an SPI interface and an RS485 interface. It is connected to the PC via an USB interface. The following data apply for RS485:

- half duplex communication mode
- industry standard 75179 transceiver
- 2400, 9600, and 19200 bps supported

Note: For RS232 set the *telegram pause time* to zero for maximum data throughput.

9 Revision history

9.1 Document revision

Version	Date	Author	Description
1.00	2009-SEP-11	SD	Initial version
1.01	2009-OCT-28	GE	Photos added
1.02	2009-DEC-01	GE	TMC-113-60-SE
1.03	2011-MAR-14	SD	Front page new
1.04	2011-SEP-3	SD	Chapter 6 updated, some examples for commands new, minor changes

Table 9.1: Document revision

9.2 Firmware revision

Version	Date	Description
3.37	2009-01-29	
3.38	2009-08-17	
3.39	2010-MAY-17	
3.40	2011-APR-18	Latest version

Table 9.2: Firmware revision

10 References

- [QSH5718] 57mm stepper motor manual (see <http://www.trinamic.com>)
[QSH6018] 60mm stepper motor manual (see <http://www.trinamic.com>)
[USB-2-485] USB-2-485 interface converter manual (see <http://www.trinamic.com>)