PD-*x*-1670 TMCL™ Firmware Manual

Firmware Version V2.0 | Document Revision V1.0 • 2018-Oct-29

PD-*x*-1670 is an easy to use and rather compact PANdrive™ smart BLDC motor. The module is controlled via CAN bus interface and comes with two firmware options – TMCL and CANopen. The PD-*x*-1670 line offer an integrated hall sensor based encoder for closed-loop FOC based operation.

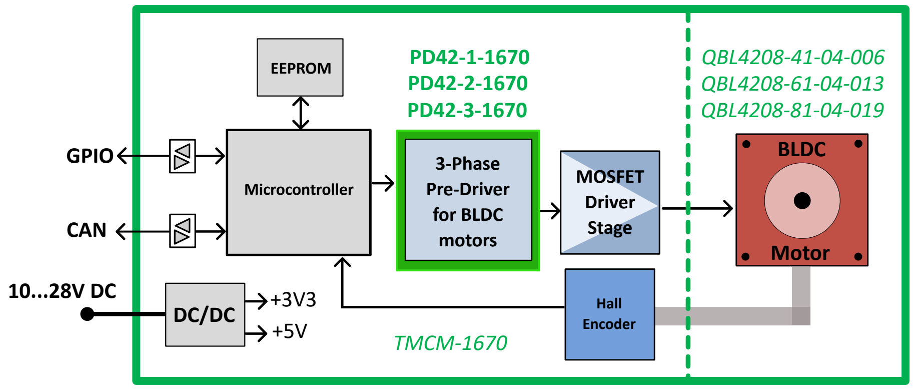
Features

- **PANdrive™** smart BLDC motor
- Supply Voltage +10 to +28V DC
- CAN bus interface
- TMCL or CANopen protocol
- integrated encoder (hall sensor based)
- FOC based + SVPWM

Applications

- Lab-Automation
- Manufacturing
- Robotics
- Factory Automation
- CNC

Simplified Block Diagram



Read entire documentation.

Contents

1 Features

The PD42-x-1670 is a single axis PANdrive™ for three-phase BLDC motors with state of the art feature set. It is highly integrated, offers a convenient handling and can be used in many decentralized applications. The module has been designed for coil currents up to 6.8A Peak at 24V DC supply voltage and has four digital inputs.

The PANdrive™ can be controlled via CAN or UART interface using TMCL™ communication protocoll or in a standalone mode with a TMCL™ script.

Main characteristics

- Motion controller & BLDC motor driver:
  - High performance microcontroller for overall system control and communication protocol handling
  - BLDC/PMSM sine commutation with field-oriented control (FOC)
  - Cascaded motor regulation modes in software (torque, velocity, and position)
  - On the fly adaptation of motion parameters (e.g. max. velocity, acceleration, PI parameter)
- Interfaces
  - CAN
- Software
  - The PD-1670 software runs on a microprocessor and consists of two parts. The boot loader is installed by TRINAMIC during production. It remains untouched throughout its entire product lifetime. The firmware can be updated by the user. New versions can be downloaded free of charge from the product's web page [TMCM-1670].

Electrical data

- Supply voltage: 24V nominal (10. . . 28V DC supply range)
- Motor current: up to 6.8A peak

Please see also the separate Hardware Manual.

2 First Steps with TMCL

In this chapter you can find some hints for your first steps with the PD-*x*-1670 and TMCL. You may skip this chapter if you are already familiar with TMCL and the TMCL-IDE.

Things that you will need

- Your PD-*x*-1670 PANdrive.
- A power supply (24V DC) for your PD-*x*-1670 PANdrive.
- A CAN adapter.
- The TMCL-IDE 3.x already installed on your PC

2.1 Basic Setup

First of all, you will need a PC with Windows (at least Windows 7) and the TMCL-IDE 3.x installed on it. If you do not have the TMCL-IDE installed on your PC then please download it from the TMCL-IDE product page of Trinamic's website (www.trinamic.com/support/software/) and install it on your PC.

Please also ensure that your PD-*x*-1670 is properly connected to your power supply. Please see the PD-*x*-1670 hardware manual for instructions on how to do this. **Do not connect or disconnect the motor to or from the module while the module is powered!** Then, please start up the TMCL-IDE. Afterwards you can connect your PD-*x*-1670. Just connect the module via CAN and switch on the power supply for the PANdrive.

2.2 Using the TMCL Direct Mode

At first try to use some TMCL commands in direct mode. In the TMCL-IDE a tree view showing the PD-*x*-1670 and all tools available for it is displayed. Click on the Direct Mode entry of the tool tree. Now, the Direct Mode tool will pop up.

In the Direct Mode tool you can choose a TMCL command, enter the necessary parameters and execute the command. For example, choose the command ROL (rotate left). Then choose the appropriate motor (motor 0 if your motor is connected to the motor 0 connector). Now, enter the desired speed. Try entering 500 rpm as the value and then click the Execute button. The motor will now run. Choose the MST (motor stop) command and click Execute again to stop the motor.

Next you can try changing some settings (also called axis parameters) using the SAP command in direct mode. Choose the SAP command. Then choose the parameter type and the motor number. Last, enter the desired value and click execute to execute the command which then changes the desired parameter. Please see chapter 4 for a complete list of all axis parameters.

2.3 Testing with a simple TMCL Program

Now, test the TMCL stand alone mode with a simple TMCL program. To type in, assemble and download the program, you will need the TMCL creator. This is also a tool that can be found in the tool tree of the TMCL-IDE. Click the TMCL creator entry to open the TMCL creator. In the TMCL creator, type in the following little TMCL program:

```
ROL 0, 500                    // rotate motor 0 with -500 rpm
WAIT TICKS, 0, 500
MST 0
```

```
    ROR 0, 500                        // rotate motor 0 with +500 rpm
    WAIT TICKS, 0, 500
    MST 0

    SAP 4, 0, 2000                    // set max. velocity
    SAP 11, 0, 1000                   // set max. acceleration
Loop:
    MVP ABS, 0, 512000                // move to position 512000
    WAIT POS, 0, 0                    // wait until position is reached
    MVP ABS, 0, -512000               // move to position -512000
    WAIT POS, 0, 0                    // wait until position os reached
    JA Loop                           // infinite Loop
```

After you have done that, take the following steps:

1. Click the Assemble icon (or choose Assemble from the TMCL menu) in the TMCL creator to assemble the program.
2. Click the Download icon (or choose Download from the TMCL menu) in the TMCL creator to donwload the program to the module.
3. Click the Run icon (or choose Run from the TMCL menu) in the TMCL creator to run the program on the module.

Also try out the debugging functions in the TMCL creator:

1. Click on the Bug icon to start the debugger.
2. Click the Animate button to see the single steps of the program.
3. You can at any time pause the program, set or reset breakpoints and resume program execution.
4. To end the debug mode click the Bug icon again.

3 TMCL and the TMCL-IDE — An Introduction

The PD-*x*-1670 supports TMCL direct mode (binary commands). It also implements standalone TMCL program execution. This makes it possible to write TMCL programs using the TMCL-IDE and store them in the memory of the module.

In direct mode the TMCL communication over RS-232, RS-485, CAN, and USB 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 PD-*x*-1670. 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 the interface to the bus master. Only then should the master transfer the next command.

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 TMCM 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 (IDE means 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.

3.1 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 RS-232, RS-485, RS-422 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In these cases it consists of nine bytes.

The binary command format with RS-232, RS-485, RS-422 and USB is as follows:

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

Table 1: TMCL Command Format

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

Note When using the CAN interface, leave out the address byte and the checksum byte. With CAN, the CAN-ID is used as the module address and the checksum is not needed because CAN bus uses hardware CRC checking.

3.1.1 Checksum Calculation

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

Checksum calculation 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
```

Checksum calculation 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
```

3.2 Reply Format

Every time a command has been sent to a module, the module sends a reply. The reply format with RS-232, RS-485, RS-422 and USB is as follows:

TMCL Reply Format	
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

Table 2: TMCL Reply Format

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

Note When using CAN interface, the reply does not contain an address byte and a checksum byte. With CAN, the CAN-ID is used as the reply address and the checksum is not needed because the CAN bus uses hardware CRC checking.

3.2.1 Status Codes

The reply contains a status code. The status code can have one of the following values:

TMCL Status Codes	
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

Table 3: TMCL Status Codes

3.3 Standalone Applications

The module is equipped with a TMCL memory for storing TMCL applications. You can use the TMCL-IDE for developing standalone TMCL applications. You can download a program into the EEPROM and afterwards it will run on the module. The TMCL-IDE contains an editor and the 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.

3.4 TMCL Command Overview

This sections gives a short overview of all TMCL commands.

Overview of all TMCL Commands			
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>, <value>	Store axis parameter (store motion control specific settings)
RSAP	8	<parameter>, <motor number>	Restore axis parameter (restore motion control specific settings)
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 variables only)
SIO	14	<port number>, <bank number>, <value>	Set digital output to specified value
GIO	15	<port number>, <bank number>	Get value of analog/digital input
CALC	19	<operation>, <value>	Aithmetical operation between accumulator and direct value
COMP	20	<value>	Compare accumulator with 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

Command	Number	Parameter	Description
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>	Arithmetical operation between accumulator and X-register
AAP	34	<parameter>, <motor number>	Accumulator to axis parameter
AGP	35	<parameter>, <bank number>	Accumulator to global parameter
CLE	36	<flag>	Clear an error flag
ACO	39	<coordinate number>, <motor number>	Accu to coordinate

Table 4: Overview of all TMCL Commands

3.5 TMCL Commands by Subject

3.5.1 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.

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

Table 5: Motion Commands

3.5.2 Parameter Commands

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

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

Table 6: Parameter Commands

3.5.3 Branch Commands

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

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

Table 7: Branch Commands

3.5.4 I/O Port Commands

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

I/O Port Commands		
Mnemonic	Command number	Meaning
SIO	14	Set output
GIO	15	Get input

Table 8: I/O Port Commands

3.5.5 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.

Calculation Commands		
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
ACO	39	Copy accu to coordinate

Table 9: Calculation Commands

For calculating purposes there is an accumulator (also called 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 and GGP 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.

3.6 Detailed TMCL Command Descriptions

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

3.6.1 ROR (Rotate Right)

The motor is instructed to rotate with a specified velocity in right direction (increasing the position counter). The velocity is given in microsteps per second (pulse per second [pps]).

Internal function:

- First, velocity mode is selected.
- Then, the velocity value is transferred to axis parameter #2 (target velocity).

Related commands: ROL, MST, SAP, GAP.

Mnemonic: ROR <axis>, <velocity>

Binary Representation			
Instruction	Type	Motor/Bank	Value
1	0	0	-2147483648...2147583647

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Rotate right motor 0, velocity 500.
Mnemonic: ROR 0, 500.

Binary Form of ROR 0, 51200	
Field	Value
Target address	01_h
Instruction number	01_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	$C8_h$
Value (Byte 0)	00_h
Checksum	CA_h

3.6.2 ROL (Rotate Left)

The motor is instructed to rotate with a specified velocity in left direction (decreasing the position counter). The velocity is given in microsteps per second (pulse per second [pps]).

Internal function:

- First, velocity mode is selected.
- Then, the velocity value is transferred to axis parameter #2 (target velocity).

Related commands: ROR, MST, SAP, GAP.

Mnemonic: ROL <axis>, <velocity>

Binary Representation			
Instruction	Type	Motor/Bank	Value
2	0	0	-2147483648. . . 2147583647

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Rotate left motor 0, velocity 500.
Mnemonic: ROL 0, 500.

Binary Form of ROL 0, 51200	
Field	Value
Target address	01_h
Instruction number	02_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	$C8_h$
Value (Byte 0)	00_h
Checksum	CB_h

3.6.3 MST (Motor Stop)

The motor is instructed to stop with a soft stop.

Internal function: The velocity mode is selected. Then, the target speed (axis parameter #0) is set to zero.
Related commands: ROR, ROL, SAP, GAP.

Mnemonic: MST <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
3	0	0	0

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Stop motor 0.
Mnemonic: MST 0.

Binary Form of MST 0	
Field	Value
Target address	01_h
Instruction number	03_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	04_h

3.6.4 MVP (Move to Position)

With this command the motor will be instructed to move to a specified relative or absolute position. 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 as well as other ramp parameters are defined by the appropriate axis parameters. For a list of these parameters please refer to section 4.
The range of the MVP command is 32 bit signed (-2147483648. . . 2147483647). Positioning can be interrupted using MST, ROL or ROR commands.

Three operation types are available:

- Moving to an absolute position in the range from -2147483648. . . 2147483647 ($-2^{31}...2^{31}-1$).
- 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).

Note The distance between the actual position and the new position must not be more than 2147483647 ($2^{31}-1$) position steps . Otherwise the motor will run in the opposite direction in order to take the shorter distance (caused by 32 bit overflow).

Internal function: A new position value is transferred to the axis parameter #0 (target position).
Related commands: SAP, GAP, SCO, GCO, CCO, ACO, MST.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
4	0 – ABS – absolute	0	<position>
	1 – REL – relative	0	<offset>
	2 – COORD – coordinate	0. . . 255	<coordinate number (0..20)>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Move motor 0 to position 90000.
Mnemonic: MVP ABS, 0, 90000

Binary Form of MVP ABS, 0, 90000	
Field	Value
Target address	01_h
Instruction number	04_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	01_h
Value (Byte 1)	$5F_h$
Value (Byte 0)	90_h
Checksum	$F5_h$

Example
Move motor 0 from current position 10000 steps backward.
Mnemonic: MVP REL, 0, -10000

Binary Form of MVP REL, 0, -10000	
Field	Value
Target address	01_h
Instruction number	04_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	FF_h
Value (Byte 2)	FF_h
Value (Byte 1)	$D8_h$
Value (Byte 0)	$F0_h$
Checksum	CC_h

Example
Move motor 0 to stored coordinate #8.

Mnemonic: MVP COORD, 0, 8

Binary Form of MVP COORD, 0, 8	
Field	**Value**
Target address	01_h
Instruction number	04_h
Type	02_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	08_h
Checksum	$0F_h$

Note	Before moving to a stored coordinate, the coordinate has to be set using an SCO, CCO or ACO command.

3.6.5 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.

Info For a table with parameters and values which can be used together with this command please refer to section 4.

Internal function: The specified value is written to the axis parameter specified by the parameter number.
Related commands: GAP, AAP.

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

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
5	see chapter 4	0	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example Set the maximum positioning speed for motor 0 to 51200 pps.
Mnemonic: SAP 4, 0, 51200.

Binary Form of SAP 4, 0, 51200	
Field	Value
Target address	01_h
Instruction number	05_h
Type	04_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	$C8_h$
Value (Byte 0)	00_h
Checksum	$D2_h$

3.6.6 GAP (Get Axis Parameter)

Most motion / driver related parameters of the PD-*x*-1670 can be adjusted using e.g. the SAP command. With the GAP 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 conditional jumps). In direct mode the value read is only output in the value field of the reply, without affecting the accumulator.

Info	For a table with parameters and values that can be used together with this command please refer to section 4.

Internal function: The specified value gets copied to the accumulator.
Related commands: SAP, AAP.

Mnemonic: GAP <parameter number>, <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
6	see chapter 4	0	<value>

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

Example
Get the actual position of motor 0.
Mnemonic: GAP 1, 0.

Binary Form of GAP 1, 0	
Field	Value
Target address	01_h
Instruction number	06_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	08_h

3.6.7 STAP (Store Axis Parameter)

This command is used to store TMCL axis parameters permanently in the EEPROM of the module. This command is mainly needed to store the default configuration of the module. The contents of the user variables can either be automatically or manually restored at power on.

Info For a table with parameters and values which can be used together with this command please refer to dection 4.

Internal function: The axis parameter specified by the type and bank number will be stored in the EEPROM.

Related commands: SAP, AAP, GAP, RSAP.

Mnemonic: STAP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
7	see chapter 4	0	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Store axis parameter #6.
Mnemonic: STAP 7, 6.

Binary Form of STAP 6, 12	
Field	Value
Target address	01_h
Instruction number	07_h
Type	06_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$0E_h$

3.6.8 RSAP (Restore Axis Parameter)

With this command the contents of an axis parameter can be restored from the EEPROM. By default, all axis parameters are automatically restored after power up. An axis parameter that has been changed before can be reset to the stored value by this instruction.

Info For a table with parameters and values which can be used together with this command please refer to section 4.

Internal function: The axis parameter specified by the type and bank number will be restored from the EEPROM.

Related commands: SAP, AAP, GAP, RSAP.

Mnemonic: RSAP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
8	see chapter 4	0	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Restore axis parameter #6.
Mnemonic: RSAP 8, 6.

Binary Form of RSAP 8, 6	
Field	Value
Target address	01_h
Instruction number	08_h
Type	06_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$0A_h$

3.6.9 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, bank 0 is used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.
All module settings in bank 0 will automatically be stored in non-volatile memory (EEPROM).

Info For a table with parameters and values which can be used together with this command please refer to section 5.

Internal function: The specified value will be copied to the global parameter specified by the type and bank number. Most parameters of bank 0 will automatically be stored in non-volatile memory.
Related commands: GGP, AGP.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
9	see chapter 5	0/2/3	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Set the serial address of the device to 3.
Mnemonic: SGP 66, 0, 3.

Binary Form of SGP 66, 0, 3	
Field	Value
Target address	01_h
Instruction number	09_h
Type	42_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	03_h
Checksum	$4F_h$

3.6.10 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, bank 0 is used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.

Info For a table with parameters and values which can be used together with this command please refer to section 5.

Internal function: The global parameter specified by the type and bank number will be copied to the accumulator register.
Related commands: SGP, AGP.

Mnemonic: GGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
10	see chapter 5	0/2/3	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

Example
Get the serial address of the device.
Mnemonic: GGP 66, 0.

Binary Form of GGP 66, 0	
Field	Value
Target address	01_h
Instruction number	$0A_h$
Type	42_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$4D_h$

3.6.11 STGP (Store Global Parameter)

This command is used to store TMCL global parameters permanently in the EEPROM of the module. This command is mainly needed to store the TMCL user variables (located in bank 2) in the EEPROM of the module, as most other global parameters (located in bank 0) are stored automatically when being modified. The contents of the user variables can either be automatically or manually restored at power on.

Info For a table with parameters and values which can be used together with this command please refer to dection **??**.

Internal function: The global parameter specified by the type and bank number will be stored in the EEPROM.

Related commands: SGP, AGP, GGP, RSGP.

Mnemonic: STGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
11	see chapter **??**	2	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Store user variable #42.
Mnemonic: STGP 42, 2.

Binary Form of STGP 42, 2	
Field	Value
Target address	01_h
Instruction number	$0B_h$
Type	$2A_h$
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	38_h

3.6.12 RSGP (Restore Global Parameter)

With this command the contents of a TMCL user variable can be restored from the EEPROM. By default, all user variables are automatically restored after power up. A user variable that has been changed before can be reset to the stored value by this instruction.

Info	For a table with parameters and values which can be used together with this command please refer to section **??**.

Internal function: The global parameter specified by the type and bank number will be restored from the EEPROM.

Related commands: SGP, AGP, GGP, STGP.

Mnemonic: RSGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
12	see chapter **??**	2	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Restore user variable #42.
Mnemonic: RSGP 42, 2.

Binary Form of RSGP 42, 2	
Field	Value
Target address	01_h
Instruction number	$0C_h$
Type	$2A_h$
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	39_h

3.6.13 SIO (Set Output)

This command sets the states of the general purpose digital outputs.

Internal function: The state of the output line specified by the type parameter is set according to the value passed to this command.

Related commands: GIO.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
14	<port number>	<bank number> (2)	0/1

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Set output 0 (bank 2) to high.
Mnemonic: SIO 0, 2, 1.

Binary Form of SIO 0, 2, 1	
Field	Value
Target address	01_h
Instruction number	$0E_h$
Type	00_h
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	01_h
Checksum	12_h

Bank 2 – Digital Outputs
The following output lines can be set by the SIO commands) using bank 2.

Digital Outputs in Bank 2			
Port	Description	Command	Range
0 - OUTPUT_0	Digital output 0	SIO 0, 2, <value>	0/1

3.6.14 GIO (Get Input)

With this command the status of the available general purpose outputs of the module can be read. The function reads a digital or an analog input port. Digital lines will read as 0 or 1, while the ADC channels deliver their bit result in the range of 0.... In standalone mode the requested value is copied to the accumulator register for further processing purposes such as conditional 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 state of the i/o line specified by the type parameter and the bank parameter is read.

Related commands: SIO.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
15	<port number>	<bank number> (0/1/2)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	status of the port

Example
Get the value of ADC channel 0.
Mnemonic: GIO 0, 1.

Binary Form of GIO 0, 1	
Field	Value
Target address	01_h
Instruction number	$0F_h$
Type	00_h
Motor/Bank	01_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	11_h

Reply (Status=no error, Value=302)	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	$0F_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	01_h
Value (Byte 0)	$2E_h$
Checksum	$A5_h$

Bank 0 – Digital Inputs
The analog input lines can be read as digital or analog inputs at the same time. The digital input states can be accessed in bank 0.

Digital Inputs in Bank 0			
Port	Description	Command	Range
0 - REF_R	Digital input 0	GIO 0, 0	0/1
1 - REF_L	Digital input 1	GIO 1, 0	0/1
2 - INPUT_0	Digital input 2	GIO 2, 0	0/1
3 - ENABLE	Digital input 3	GIO 3, 0	0/1

Bank 1 – Analog Inputs
The analog input lines can be read back as digital or analog inputs at the same time. The analog values can be accessed in bank 1.

Analog Inputs in Bank 1			
Port	Description	Command	Range
0 - ADC_Phase_A	Analog input 0	GIO 0, 1	0...4095
1 - ADC_Phase_C	Analog input 1	GIO 1, 1	0...4095
2 - ADC_VOLTAGE	Analog input 2	GIO 2, 1	0...4095
3 - ADC_MOT_TEMP	Analog input 3	GIO 3, 1	0...4095

Bank 2 – States of the Digital Outputs
The states of the output lines (that have been set by SIO commands) can be read back using bank 2.

Digital Outputs in Bank 2			
Port	Description	Command	Range
0 - OUTPUT_0	Digital output 0	GIO 0, 2, <value>	0/1

3.6.15 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. *This command is mainly intended for use in standalone mode.*

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

Mnemonic: CALC <operation>, <operand>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
19	0 ADD – add to accumulator	0 (don't care)	<operand>
	1 SUB – subtract from accumulator		
	2 MUL – multiply accumulator by		
	3 DIV – divide accumulator by		
	4 MOD – modulo divide accumulator by		
	5 AND – logical and accumulator with		
	6 OR – logical or accumulator with		
	7 XOR – logical exor accumulator with		
	8 NOT – logical invert accumulator		
	9 LOAD – load operand into accumulator		

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Multiply accumulator by -5000.
Mnemonic: CALC MUL, -5000

Binary Form of CALC MUL, -5000	
Field	Value
Target address	01_h
Instruction number	13_h
Type	02_h
Motor/Bank	00_h
Value (Byte 3)	FF_h
Value (Byte 2)	FF_h
Value (Byte 1)	EC_h
Value (Byte 0)	78_h
Checksum	78_h

Reply (Status=no error, value=-5000:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	13_h
Value (Byte 3)	FF_h
Value (Byte 2)	FF_h
Value (Byte 1)	EC_h
Value (Byte 0)	78_h
Checksum	DC_h

3.6.16 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.*

Internal function: The accumulator register is compared with the sepcified value. The internal arithmetic status flags are set according to the result of the comparison. These can then control e.g. a conditional jump.
Related commands: JC, GAP, GGP, GIO, CALC, CALCX.

Mnemonic: COMP <operand>

Binary Representation			
Instruction	Type	Motor/Bank	Value
20	0 (don't care)	0 (don't care)	<operand>

Example
Jump to the address given by the label when the position of motor #0 is greater than or equal to 1000.

```
GAP 1, 0 //get actual position of motor 0
COMP 1000    //compare actual value with 1000
JC GE, Label    //jump to Lable if greter or equal to 1000
```

Binary Form of COMP 1000	
Field	Value
Target address	01_h
Instruction number	14_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	03_h
Value (Byte 0)	$E8_h$
Checksum	00_h

3.6.17 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 command is intended for standalone operation only.*

Internal function: The TMCL program counter is set to the value passed to this command if the status flags are in the appropriate states.

Related commands: JA, COMP, WAIT, CLE.

Mnemonic: JC <condition>, <label>

Binary Representation			
Instruction	Type	Motor/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 10 EDV - deviation error 11 EPO - position error	0 (don't care)	<jump address>

Example
Jump to the address given by the label when the position of motor #0 is greater than or equal to 1000.

```
GAP 1, 0  //get actual position of motor 0
COMP 1000   //compare actual value with 1000
JC GE, Label   //jump to Lable if greter or equal to 1000
...
Label: ROL 0, 1000
```

Binary form of JC GE, Label assuming Label at address 10	
Field	Value
Target address	01_h
Instruction number	15_h
Type	05_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	$0A_h$
Checksum	25_h

3.6.18 JA (Jump always)

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

Internal function: The TMCL program counter is set to the value passed to this command.

Related commands: JC, WAIT, CSUB.

Mnemonic: JA <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
22	0 (don't care)	0 (don't care)	<jump address>

Example
An infinite loop in TMCL:

```
Loop:
    MVP ABS, 0, 51200
    WAIT POS, 0, 0
    MVP ABS, 0, 0
    WAIT POS, 0, 0
    JA Loop
```

Binary form of the JA Loop command when the label Loop is at address 10:

Binary Form of JA Loop (assuming Loop at address 10)	
Field	Value
Target address	01_h
Instruction number	16_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	$0A_h$
Checksum	21_h

3.6.19 CSUB (Call Subroutine)

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

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	Type	Motor/Bank	Value
23	0 (don't care)	0 (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
    CSUB SubW   //Save program counter and jump to label SubW
    JA Loop

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

Binary form of CSUB SubW (assuming SubW at address 100)	
Field	Value
Target address	01_h
Instruction number	17_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	64_h
Checksum	$7C_h$

3.6.20 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.*

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

Related commands: CSUB.

Mnemonic: RSUB

Binary Representation			
Instruction	Type	Motor/Bank	Value
24	0 (don't care)	0 (don't care)	0 (don't care)

Example
Please see the CSUB example (section 3.6.19).
Binary form:

Binary Form of RSUB	
Field	Value
Target address	01_h
Instruction number	18_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	19_h

3.6.21 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.*
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.

Special case for the <ticks> parameter: When this parameter is set to -1 the contents of the accumulator register will be taken for this value. So for example WAIT TICKS, 0, -1 will wait as long as specified by the value store in the accumulator. *The accumulator must not contain a negative value when using this option.*

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 will be held at the address of this WAIT command until the condition is met or the timeout has expired.

Related commands: JC, CLE.

Mnemonic: WAIT <condition>, <motor number>, <ticks>

Binary Representation			
Instruction	Type	Motor/Bank	Value
27	0 TICKS – timer ticks	0 (don't care)	<no. of ticks to wait[1]>
	1 POS – target position reached	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout
	2 REFSW – reference switch	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout
	3 LIMSW – limit switch	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout
	4 RFS – reference search completed	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout

Example

[1] one tick is 10 milliseconds

Wait for motor 0 to reach its target position, without timeout.
Mnemonic: WAIT POS, 0, 0

Binary Form of WAIT POS, 0, 0	
Field	Value
Target address	01_h
Instruction number	$1B_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$1D_h$

3.6.22 STOP (Stop TMCL Program Execution – End of TMCL Program)

This command stops the execution of a TMCL program. *It is intended for use in standalone operation only.*

Internal function: Execution of a TMCL program in standalone mode will be stopped.

Related commands: none.

Mnemonic: STOP

Binary Representation			
Instruction	Type	Motor/Bank	Value
28	0 (don't care)	0 (don't care)	0 (don't care)

Example
Mnemonic: STOP

Binary Form of STOP	
Field	Value
Target address	01_h
Instruction number	$1C_h$
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$1D_h$

3.6.23 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. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only).

Note	Coordinate #0 is always stored in RAM only.

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

Related commands: GCO, CCO, ACO, MVP COORD.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
30	<coordinate number> 0...20	<motor number> 0	<position> $-2^{31} \ldots 2^{31}-1$

Example
Set coordinate #1 of motor #0 to 1000.
Mnemonic: SCO 1, 0, 1000

Binary Form of SCO 1, 0, 1000	
Field	Value
Target address	01_h
Instruction number	$1E_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	03_h
Value (Byte 0)	$E8_h$
Checksum	$0B_h$

Two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate to the EEPROM. These functions can be accessed using the following special forms of the SCO command:

- SCO 0, 255, 0 copies all coordinates (except coordinate number 0) from RAM to the EEPROM.
- SCO <coordinate number>, 255, 0 copies the coordinate selected by <coordinate number> to the EEPROM. The coordinate number must be a value between 1 and 20.

3.6.24 GCO (Get Coordinate)

Using this command previously stored coordinate can be read back. In standalone mode the requested value is copied to the accumulator register for further processing purposes such as conditional jumps. In direct mode, the value is only output in the value field of the reply, without affecting the accumulator. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only).

Note	Coordinate #0 is always stored in RAM only.

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, ACO, MVP COORD.

Mnemonic: GCO <coordinate number>, <motor number>

Binary Representation			
Instruction	Type	Motor/Bank	Value
31	<coordinate number> 0. . . 20	<motor number> 0	0 (don’t care)

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

Example
Get coordinate #1 of motor #0.
Mnemonic: GCO 1, 0

Binary Form of GCO 1, 0	
Field	Value
Target address	01_h
Instruction number	$1F_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	21_h

Two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate from the EEPROM to the RAM.
These functions can be accessed using the following special forms of the GCO command:

- GCO 0, 255, 0 copies all coordinates (except coordinate number 0) from the EEPROM to the RAM.
- GCO <coordinate number>, 255, 0 copies the coordinate selected by <coordinate number> from the EEPROM to the RAM. The coordinate number must be a value between 1 and 20.

3.6.25 CCO (Capture Coordinate)

This command copies the actual position of the axis to the selected coordinate variable. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only). Please see the SCO and GCO commands on how to copy coordinates between RAM and EEPROM.

Note Coordinate #0 is always stored in RAM only.

Internal function: the actual position of the selected motor is copied to selected coordinate array entry.

Related commands: SCO, GCO, ACO, MVP COORD.

Mnemonic: CCO <coordinate number>, <motor number>

Binary Representation			
Instruction	Type	Motor/Bank	Value
32	<coordinate number> 0. . . 20	<motor number> 0	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

Example
Store current position of motor #0 to coordinate array entry #3.

Mnemonic: CCO 3, 0

Binary Form of CCO 3, 0	
Field	Value
Target address	01_h
Instruction number	20_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	22_h

3.6.26 ACO (Accu to Coordinate)

With the ACO command the actual value of the accumulator is copied to a selected coordinate of the motor. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only).

Note	Coordinate #0 is always stored in RAM only.

Internal function: the actual position of the selected motor is copied to selected coordinate array entry.

Related commands: SCO, GCO, CO, MVP COORD.

Mnemonic: ACO <coordinate number>, <motor number>

Binary Representation			
Instruction	Type	Motor/Bank	Value
39	<coordinate number> 0…20	<motor number> 0	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Copy the actual value of the accumulator to coordinate #1 of motor #0.

Mnemonic: ACO 1, 0

Binary Form of ACO 1, 0	
Field	Value
Target address	01_h
Instruction number	27_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	29_h

3.6.27 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. *This command is mainly intended for use in standalone mode.*

Related commands: CALC, COMP, JC, AAP, AGP, GAP, GGP, GIO.

Mnemonic: CALCX <operation>

Binary Representation			
Instruction	Type	Motor/Bank	Value
33	0 ADD – add X register to accumulator 1 SUB – subtract X register from accumulator 2 MUL – multiply accumulator by X register 3 DIV – divide accumulator by X register 4 MOD – modulo divide accumulator by X register 5 AND – logical and accumulator with X register 6 OR – logical or accumulator with X register 7 XOR – logical exor accumulator with X register 8 NOT – logical invert X register 9 LOAD – copy accumulator to X register 10 SWAP – swap accumulator and X register	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Multiply accumulator and X register.
Mnemonic: CALCX MUL

Binary Form of CALCX MUL	
Field	Value
Target address	01_h
Instruction number	21_h
Type	02_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	24_h

3.6.28 AAP (Accu 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. *This command is mainly intended for use in standalone mode.*

Info For a table with parameters and values which can be used together with this command please refer to section 4.

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

Mnemonic: AAP <parameter number>, <motor number>

Binary Representation			
Instruction	Type	Motor/Bank	Value
34	see chapter 4	0	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Position motor #0 by a potentiometer connected to analog input #0:

```
Start:
    GIO 0,1       //get value of analog 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 Form of AAP 0, 0	
Field	Value
Target address	01_h
Instruction number	22_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	23_h

3.6.29 AGP (Accu 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. *This command is mainly intended for use in standalone mode.*

Info For an overview of parameter and bank indices that can be used with this command please see section 5.

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

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
35	<parameter number>	0/2/3 <bank number>	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Copy accumulator to user variable #42:
Mnemonic: AGP 42, 2

Binary Form of AGP 42, 2	
Field	Value
Target address	01_h
Instruction number	23_h
Type	$2A_h$
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	50_h

3.6.30 CLE (Clear Error Flags)

This command clears the internal error flags. It is mainly intended for use in standalone 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, WAIT.

Mnemonic: CLE <flags>

Binary Representation			
Instruction	Type	Motor/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	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Reset the timeout flag.
Mnemonic: CLE ETO

Binary Form of CLE ETO	
Field	Value
Target address	01_h
Instruction number	24_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	26_h

3.6.31 Customer specific Command Extensions (UF0. . . UF7 – User Functions)

These commands are used for customer specific extensions of TMCL. They will be implemented in C by Trinamic. Please contact the sales department of Trinamic Motion Control GmbH & Co KG if you need a customized TMCL firmware.

Related commands: none.

Mnemonic: UF0. . . UF7

Binary Representation			
Instruction	Type	Motor/Bank	Value
64. . . 71	<user defined>	0 <user defined>	0 <user defined>

Reply in Direct Mode	
Status	Value
100 - OK	user defined

3.6.32 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 a motor has reached its target position.

Related commands: none.

Binary Representation			
Instruction	Type	Motor/Bank	Value
138	0/1	0 (don't care)	always 1

With command 138 the value field is a bit vector. It shows for which motors one would like to have a position reached message. The value field contains a bit mask where every bit stands for one motor. For one motor modules like the PD-*x*-1670 it only makes sense to have bit 0 set. So, always set this parameter to 1 with the PD-*x*-1670 module. With the type field set to 0, only for the next MVP command that follows this command a position reached message will be generated. With type set to 1 a position reached message will be generated for every MVP command that follows this command. It is recommended to use the latter option.

Example
Get a target position reached message for each MVP command that follows.

Binary Form for this example	
Field	Value
Target address	01_h
Instruction number	$8A_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	01_h
Checksum	$8D_h$

Reply in Direct Mode	
Field	Value
Target address	01_h
Host address	02_h
Status	64_h (100)
Command	$8A_h$ (138)
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	Motor bit mask
Checksum	depends also on motor bit mask

Additional Reply after Motor has reached Target Position	
Field	Value
Target address	01_h
Host address	02_h
Status	80_h (128)
Command	$8A_h$ (138)
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	Motor bit mask
Checksum	depends also on motor bit mask

3.6.33 TMCL Control Commands

There is a set of TMCL commands which are called TMCL control commands. These commands can only be used in direct mode and not in a standalone program. For this reason they only have opcodes, but no mnemonics. Most of these commands are only used by the TMCL-IDE (in order to implement e.g. the debugging functions in the TMCL creator). Some of them are also interesting for use in custom host applications, for example to start a TMCL routine on a module, when combining direct mode and standalone mode (please see also section 7.6. The following table lists all TMCL control commands.

The motor/bank parameter is not used by any of these functions and thus is not listed in the table. It should always be set to 0 with these commands.

TMCL Control Commands			
Instruction	Description	Type	Value
128 – stop application	stop a running TMCL application	0 (don't care)	0 (don't care)
129 – run application	start or continue TMCL program execution	0 – from current address 1 – from specific address	0 (don't care) starting address
130 – step application	execute only the next TMCL command	0 (don't care)	0 (don't care)
131 – reset application	Stop a running TMCL program. Reset program counter and stack pointer to zero. Reset accumulator and X register to zero. Reset all flags.	0 (don't care)	0 (don't care)
132 – enter download mode	All following commands (except control commands) are not executed but stored in the TMCL memory.	0 (don't care)	start address for download
133 – exit download mode	End the download mode. All following commands are executed normally again.	0 (don't care)	0 (don't care)
134 – read program memory	Return contents of the specified program memory location (special reply format).	0 (don't care)	address of memory location

Instruction	Description	Type	Value
135 – get application status	Return information about the current status, depending on the type field.	0 - return mode, wait flag, memory pointer 1 - return mode, wait flag, program counter 2 - return accumulator 3 - return X register	0 (don't care)
136 – get firmware version	Return firmware version in string format (special reply) or binary format).	0 - string format 1 - binary format	0 (don't care)
137 – restore factory settings	Reset all settings in the EEPROM to their factory defaults. This command does not send a reply.	0 (don't care)	set to 1234
255 – software reset	Restart the CPU of the module (like a power cycle). The reply of this command might not always get through.	0 (don't care)	set to 1234

Table 10: TMCL Control Commands

Especially the commands 128, 129, 131, 136 and 255 are interesting for use in custom host applications. The other control commands are mainly being used by the TMCL-IDE.

4 Axis Parameters

Most motor controller features of the PD-*x*-1670 module are controlled by axis parameters. Axis parameters can be modified or read using SAP, GAP and AAP commands. Some axis parameters can also be stored to or restored from the EEPROM using STAP and RSAP commands. This chapter describes all axis parameters that can be used on the PD-*x*-1670 module.

Axis 0 Parameters of the PD-*x*-1670 Module					
Number	Axis Parameter	Description	Range [Units]	Default	Access
0	target position	The target position of a currently executed ramp.	-2147483648 $\ldots 2147483647$	0	RW
1	actual position	The actual position counter.	-2147483648 $\ldots 2147483647$	0	RW
2	target velocity	The desired target velocity.	-200000 $\ldots 200000$ [rpm]	0	RW
3	actual velocity	The actual velocity of the motor.	-2147483648 $\ldots 2147483647$ [rpm]	0	R
4	max velocity	Max. absolute velocity for velocity and positioning mode.	$0 \ldots 200000$ [rpm]	4000	RWEX
5	current limit	Max. allowed value for max current. (PANdrive limitation)	$0 \ldots 7400$ [mA] (peak)	7400	R
6	max current	Max. allowed absolute motor current. *This value can be temporarily exceeded marginal due to the operation of the current regulator.	$0 \ldots 7400$ [mA] (peak)	7400	RWEX
7	target reached velocity	Max. velocity at which end position flag can be set. Prevents issuing of end position flag when the target is passed at high velocity.	$0 \ldots 200000$ [rpm]	500	RWEX
9	motor halted velocity	If the actual velocity is below this value the motor halted flag will be set.	$0 \ldots 200000$ [rpm]	5	RWEX
10	target reached distance	Maximum distance at which the position end flag is set.	$0 \ldots 100000$	5	RWEX
11	acceleration	Acceleration parameter for ROL, ROR, and the velocity ramp of MVP.	$0 \ldots 100000$ [rpm/s]	2000	RWEX
13	ramp velocity	The actual velocity of the velocity ramp used for positioning and velocity mode.	-2147483648 $\ldots 2147483647$ [rpm]	0	R
14	ramp position	The actual position of the position ramp used for positioning mode.	-2147483648 $\ldots 2147483647$	0	R
31	reinit bldc regulation	Reinit bldc regulation.	$0 \ldots 0$	0	W

Number	Axis Parameter	Description	Range [Units]	Default	Access
100	Bode Control_Mode	0 - direct sine wave 1 - direct square wave 2 - sweeep sine wave	$0 \ldots 2$	0	R
101	Bode Target_Mode	0 - no target 1 - torque target 2 - velocity target 3 - position target 4 - stopped	$0 \ldots 4$	0	R
102	Bode Actual_Magnitude	Actual magnitude (ratio of actual to target value).	$-2147483648 \ldots 2147483647$ [0.1%]	0	R
103	Bode Actual_Phi	Actual phase angle error between target and actual value.	$-2147483648 \ldots 2147483647$ [1.0°]	0	R
104	Bode Actual_Frequency	Actual used bode frequency.	$0 \ldots 10000$ [0.1Hz]	0	R
106	Bode Target Value	Actual generator target value.	$-2147483648 \ldots 2147483647$	0	R
107	Bode Actual Value	Actual response value.	$-2147483648 \ldots 2147483647$	0	R
133	velocity regulation loop delay	Delay of the position and velocity regulator.	$0 \ldots 20$ [50us]	10	RWEX
134	current regulation loop delay	Delay of the current regulator.	$0 \ldots 10$ [50us]	1	RWEX
146	enable velocity ramp	An activated ramp allows a defined acceleration for velocity and position mode. 0 - Deactivate velocity ramp generator. 1 - Activate velocity ramp generator.	$0 \ldots 1$	1	RWEX
150	actual torque	The actual motor current.	$-2147483648 \ldots 2147483647$ [mA] (peak)	0	R
151	supply voltage	The actual supply voltage.	$0 \ldots 1000$ [100mV]	300	R
152	driver temperature	The actual temperature of the motor driver.	$-20 \ldots 150$ [°C]	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
155	target torque	Get desired target current or set target current to activate current regulation mode. (+= turn motor in right direction; -= turn motor in left direction)	$-7400 \ldots 7400$ [mA] (peak)	0	RW
156	status flags	Actual status flags: 0x0001 - OVERCURRENT 0x0002 - UNDERVOLTAGE 0x0004 - OVERVOLTAGE 0x0008 - OVERTEMPERATURE 0x0010 - MOTORHALTED 0x0020 - HALLERROR 0x0040 - DRIVER_ERROR 0x0080 - INIT_ERROR 0x0100 - STOP_MODE 0x0200 - VELOCITY_MODE 0x0400 - POSITION_MODE 0x0800 - TORQUE_MODE 0x1000 - EMERGENCYSTOP 0x2000 - FREERUNNING 0x4000 - POSITION_END 0x8000 - MODULE_INITIALIZED	$0 \ldots 65535$	0	R
159	commutation mode	Select a commutation mode that fits best to your motor's sensors. 7 - FOC (encoder) 8 - FOC (controlled)	$7 \ldots 8$	7	RWEX
161	clear on null	Clear the position counter on encoder N channel. 0 - do not clear position counter at next N channel event 1 - set position counter to zero at next N channel event	$0 \ldots 1$	0	RWEX
163	clear once	Clear the position counter on encoder N channel. 0 - clear position counter always at an N channel event 1 - set position counter to zero only once	$0 \ldots 1$	0	RWEX
165	encoder offset	This value represents the internal commutation offset. (0...max. encoder steps per rotation).	$0 \ldots 65535$	0	RWE
172	torque P	P parameter for current PID regulator	$0 \ldots 65535$	0	RWEX
173	torque I	I parameter for current PID regulator	$0 \ldots 65535$	0	RWEX
177	start current	Motor current for controlled commutation. This parameter is used in commutation mode 8.	$0 \ldots 7400$ [mA] (peak)	1000	RWEX
180	Main loops	Main loops per second.	$0 \ldots 4294967295$	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
181	PWM loops	PWM loops per second.	$0 \ldots 4294967295$	0	R
182	Torque loops	Torque loops per second.	$0 \ldots 4294967295$	0	R
183	Velocity loops	Velocity loops per second.	$0 \ldots 4294967295$	0	R
190	debug value 0	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
191	debug value 1	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
192	debug value 2	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
193	debug value 3	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
194	debug value 4	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
195	debug value 5	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
196	debug value 6	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
197	debug value 7	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
198	debug value 8	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
199	debug value 9	Free used debugging value.	$-2147483648 \ldots 2147483647$	0	RW
200	current PID error	Actual error of current PID regulator.	$-2147483648 \ldots 2147483647$ [mA]	0	R
201	current PID error sum	Sum of errors of current PID regulator.	$-2147483648 \ldots 2147483647$	0	R
202	flux PID error	Actual error of flux PID regulator.	$-2147483648 \ldots 2147483647$ [mA]	0	R
203	flux PID error sum	Sum of errors of flux PID regulator.	$-2147483648 \ldots 2147483647$	0	R
211	actual encoder angle	Actual encoder angle value.	$-32768 \ldots 32767$	0	R
212	actual controlled angle	Actual controlled angle value.	$-32768 \ldots 32767$	0	R
214	driver diagnostic value	Actual driver diagnostic value.	$0 \ldots 1000$ [0.1%]	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
215	driver status acknowledge	Acknowledge driver status.	$0 \ldots 0$	0	W
216	driver init spi	Disables the driver and initializes the driver SPI access.	$0 \ldots 0$	0	W
217	driver status register 2	TMC6130 status register 2.	$0 \ldots 65535$	0	RW
218	driver status register 3	TMC6130 status register 3.	$0 \ldots 65535$	0	RW
219	driver status register 4	TMC6130 status register 4.	$0 \ldots 65535$	0	RW
226	position PID error	Actual error of position PID regulator.	$-2147483648 \ldots 2147483647$	0	R
228	velocity PID error	Actual error of velocity PID regulator.	$-2147483648 \ldots 2147483647$ [rpm]	0	R
229	velocity PID error sum	Sum of errors of velocity PID regulator.	$-2147483648 \ldots 2147483647$	0	R
230	position P	P parameter for position PID regulator	$0 \ldots 65535$	0	RWEX
234	velocity P	P parameter for velocity PID regulator	$0 \ldots 65535$	0	RWEX
235	velocity I	I parameter for velocity PID regulator	$0 \ldots 65535$	0	RWEX
236	velocity filter	Filter for actual measured velocity. (Filter length = $2^{\text{(filter value)}}$)	$0 \ldots 8$	0	RWEX
241	init velocity	Init velocity for encoder initialization with mode 2.	$-200000 \ldots 200000$ [rpm]	100	RWEX
244	init sine delay	Duration for sine initialization sequence. This parameter should be set in a way, that the motor has stopped mechanical oscillations after the specified time.	$0 \ldots 10000$ [ms]	1000	RWEX
249	encoder init mode	Select an encoder init mode that fits best to your motor's sensors. 0 - initialize in controlled sine commutation (determine encoder offset) 2 - initialize in controlled sine commutation (use encoder offset)	$0 \ldots 2$	1	RWEX
250	encoder steps	Encoder steps per full motor rotation.	$0 \ldots 2147483647$	4000	RWEX
251	encoder direction	Set the encoder direction in a way, that ROR increases position counter. 0 - positive encoder direction 1 - negative encoder direction	$0 \ldots 1$	0	RWEX
253	motor poles	Number of motor poles.	$2 \ldots 254$	8	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
255	enable driver	0 - driver disabled 1 - driver enabled	$0 \ldots 1$	0	RW

Table 11: All PD-x-1670 Axis 0 Parameters

5 Global Parameters

The following sections describe all global parameters that can be used with the SGP, GGP, AGP, STGP and RSGP commands. Global parameters are grouped into banks:

- Bank 0: Global configuration of the module.

5.1 Bank 0

Parameters with numbers from 64 on configure all settings that affect the overall behaviour of a module. These are things like the serial address, the RS485 baud rate or the CAN bit rate (where appropriate). 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 automatically stored in the EEPROM.

Note

- 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.
- Some configurations of the interface (for example baud rates that are not supported by the PC) may leed to the fact that the module cannot be reached any more. In such a case please see the PD-*x*-1670 Hardware Manual on how to reset all parameters to factory default settings.
- Some settings (especially interface bit rate settings) do not take effect immediately. For those settings, power cycle the module after changing them to make the changes take effect.

There are different parameter access types, like read only or read/write. Table 12 shows the different parameter access types used in the global parameter tables.

Meaning of the Letters in the Access Column		
Access type	Command	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable
E	STGP, RSGP	Parameter can be stored in the EEPROM
A	SGP	Automatically stored in the EEPROM

Table 12: Meaning of the Letters in the Access Column

All Global Parameters of the PD-*x*-1670 Module in Bank 0					
Number	Global Parameter	Description	Range [Units]	Default	Access
65	serial baud rate	RS485/RS232 baud rate 0 - 9600 baud 1 - 14400 baud 2 - 19200 baud 3 - 28800 baud 4 - 38400 baud 5 - 57600 baud 6 - 76800 baud 7 - 115200 baud	$0 \ldots 7$	7	RW
66	serial address	The module (target) address for RS485, RS232, and virtual COM port.	$1 \ldots 255$	2	RWX
69	CAN bit rate	CAN bit rate 2 - 20 kBit/s 3 - 50 kBit/s 4 - 100 kBit/s 5 - 125 kBit/s 6 - 250 kBit/s 7 - 500 kBit/s 8 - 1000 kBit/s	$2 \ldots 8$	8	RW
70	CAN send ID	The CAN send ID of the module.	$0 \ldots 255$	2	RWX
71	CAN receive ID	The CAN receive ID of the module.	$0 \ldots 255$	1	RWX
75	telegram pause time	Pause time before the reply via RS485/RS232 is sent.	$0 \ldots 255$ [ms]	0	RWX
76	serial host address	Host address used in the reply telegrams sent back via RS485/RS232.	$1 \ldots 255$	1	RWX
77	auto start mode	Use automatic TMCL application start after power up. 0 - do not start TMCL application after power up 1 - start TMCL application automatically after power up	$0 \ldots 1$	1	RWX
128	application status	Actual TMCL application status. 0 - stop 1 - run 2 - step 3 - reset	$0 \ldots 3$	0	R
130	program counter	TMCL program counter.	$0 \ldots 4294967295$	0	R
132	tick timer	TMCL tick timer.	$0 \ldots 4294967295$	0	RW

Table 13: All Global Parameters of the PD-x-1670 Module in Bank 0

6 Motor Regulation

6.1 Structure of Cascaded Motor Regulation Modes

The PD-*x*-1670 supports a current, velocity, and position PID regulation mode for motor control in different application areas. These regulation modes are cascaded as shown in Figure 1. Individual modes are explained in the following subsections.

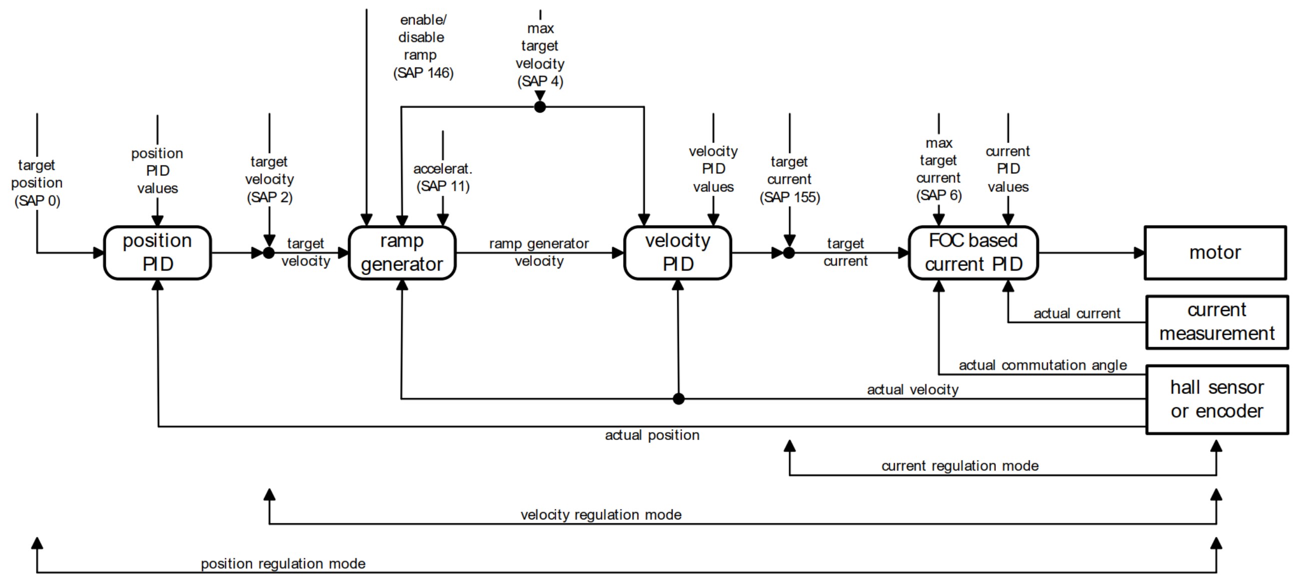


Figure 1: Cascaded Regulation

6.2 Current Regulation

The current regulation mode uses a FOC current and flux regulator to adjust a desired motor current. The target current can be set by axis parameter 155. The maximal target current is limited by axis parameter 6. The current regulation uses three basic parameters: The P and I value as well as the timing control value.

6.2.1 Timing Control Value

The timing control value (current regulation loop multiplier, axis parameter 134) determines how often the current regulation is invoked. The value is given in multiple of 50µs, following the next formula:

$$t_{PIDDELAY} = x_{PIDRLD} \cdot 50\mu s$$

$$t_{PIDDELAY} = resulting\ delay\ between\ two\ current\ regulation\ loops$$

$$x_{PIDRLD} = current\ regulation\ loop\ multiplier\ parameter$$

For most applications it is recommended to leave this parameter unchanged at its default of 1 $\times$ 50µs. Higher values may be necessary for very slow and less dynamic drives.

6.2.2 Structure of the Current Regulator

.

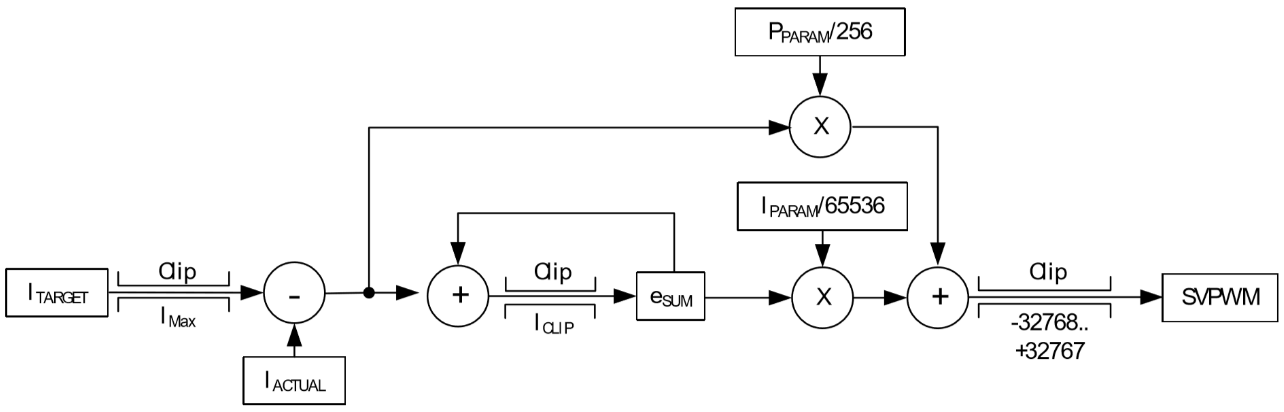


Figure 2: Current Regulation (See parameter descriptions in Table 14)

Current Regulation Parameters	
Parameter	Description
I_{ACTUAL}	Actual motor current (GAP 150)
I_{TARGET}	Target motor current (SAP 155)
I_{Max}	Max. motor current (SAP 6)
e_{SUM}	Error sum for integral calculation (GAP 201)
P_{PARAM}	Current P parameter (SAP 172)
I_{PARAM}	Current I parameter (SAP 173)

Table 14: Current Regulation Parameters

6.2.2.1 Parametrizing the Current Regulator Set

In order to parameterize properly the current regulator set, do as follows:

1. Set the P parameter and the I parameter to zero.
2. Start the motor by using a low target current (e.g. 1000mA).
3. Modify the current P parameter. Start from a low value and go to a higher value, until the actual current nearly reaches 50% of the desired target current.
4. Do the same with the current I parameter.

For all tests, set the motor current limitation to a realistic value, so that your power supply does not become overloaded during acceleration phases. If your power supply reaches current limitation, the unit may reset or undetermined regulation results may occur.

6.3 Velocity Regulation

Based on the current regulation the motor velocity can be controlled by the velocity PI regulator.

6.3.1 Timing Control Value

The velocity PI regulator uses a timing control value (velocity regulation loop multiplier, axis parameter 133) which determines how often the PID regulator is invoked. The value is given in multiple of 50µs, following the next formula:

$$t_{PIDDELAY} = x_{PIDRLD} \cdot 50\mu s$$

$$t_{PIDDELAY} = resulting\ delay\ between\ two\ PID\ calculations$$

$$x_{PIDRLD} = PID\ regulation\ loop\ delay\ parameter$$

For most applications it is recommended to leave this parameter unchanged at its default of 1 × 50µs. Higher values may be necessary for very slow and less dynamic drives.

6.3.2 Structure of the Velocity Regulator

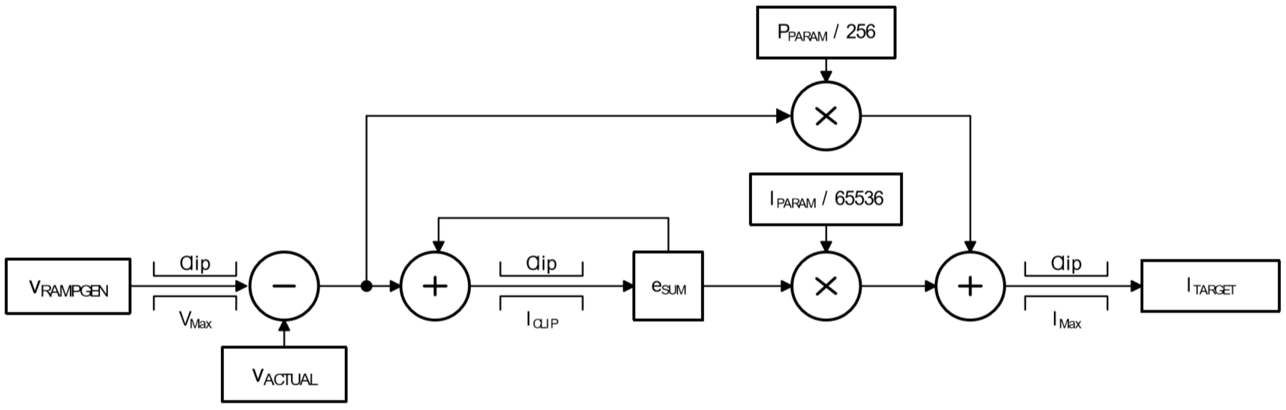


Figure 3: Velocity Regulation (See parameter descriptions in Table 15)

Velocity Regulation Parameters	
Parameter	Description
V$_{ACTUAL}$	Actual motor velocity (GAP 3)
V$_{RAMPGEN}$	Target velocity of ramp generator (SAP 2, GAP 13)
V$_{Max}$	Max. target velocity (SAP 4)
e$_{SUM}$	Error sum for integral calculation (GAP 229)
P$_{PARAM}$	Velocity P parameter (SAP 234)
I$_{PARAM}$	Velocity I parameter (SAP 235)
I$_{Max}$	Max. target current (SAP 6)
I$_{TARGET}$	Target current for current PID regulator (GAP 155)

Table 15: Velocity Regulation Parameters

6.3.2.1 Parametrizing the Velocity Regulator Set

In order to parameterize properly the velocity regulator set, do as follows:

1. Set the velocity I parameter to zero.
2. Start the motor by using a medium target velocity (e.g. 2000 rpm).
3. Modify the current P parameter.
   (a) Start from a low value and go to a higher value, until the actual motor speed reaches 80 or 90% of the target velocity.
   (b) The lasting 10 or 20% speed difference can be reduced by slowly increasing the velocity I parameter.

6.4 Velocity Ramp Generator

For a controlled startup of the motor's velocity, a velocity ramp generator can be activated/deactivated by axis parameter 146. The ramp generator uses the maximal allowed motor velocity (axis parameter 4), the acceleration (axis parameter 11) and the desired target velocity (axis parameter 2) to calculate a ramp generator velocity for the following velocity PI regulator.

6.5 Position Regulation

Based on current and velocity regulators, the PD-*x*-1670 supports a positioning mode based on encoder or hall sensor position. During positioning the velocity ramp generator can be activated to enable motor positioning with controlled acceleration or it can be disabled to support motor positioning with max allowed speed.

The PID regulation uses two basic parameters: the P regulation parameter and a timing control value.

6.5.1 Timing Control Value

The timing control value PID regulation loop parameter (axis parameter 133) determines how often the PID regulator is invoked. The value is given in multiple of 50µs, following the next formula:

$$t_{PIDDELAY} = x_{PIDRLD} \cdot 50\mu s$$

$$t_{PIDDELAY} = resulting\ delay\ between\ two\ position\ regulation\ loops$$

$$x_{PIDRLD} = PID\ regulation\ loop\ multiplier\ parameter$$

For most applications it is recommended to leave the timing control value unchanged at its default of 1 $\times$ 50µs. Higher values may be necessary for very slow and less dynamic drives.

6.5.2 Structure of the Position Regulator

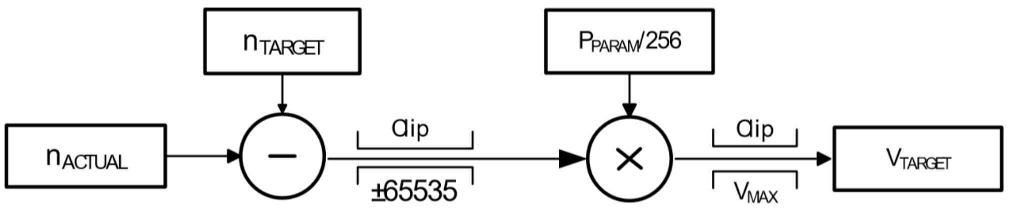


Figure 4: Positioning Regulation (See parameter descriptions in Table 16)

Position Regulation Parameters	
Parameter	Description
n_{ACTUAL}	Actual motor position (GAP 1)
n_{TARGET}	Target motor position (SAP 0)
P_{PARAM}	Position P parameter (SAP 230)
V_{MAX}	Max. allowed velocity (SAP 4)
V_{TARGET}	New target velocity for the ramp generator (GAP 13)

Table 16: Position Regulation Parameters

6.5.2.1 Parametrizing the Position Regulation

Based on the velocity regulator, only the position regulator P has to be parameterized.
In order to parameterize the position regulator, do as follows:

1. Disable the velocity ramp generator and set position P parameter to zero.
2. Choose a target position and increase the position P parameter until the motor reaches the target position approximately.
3. Switch on the velocity ramp generator. Based on the max. positioning velocity (axis parameter 4) and the acceleration value (axis parameter 11) the ramp generator automatically calculates the slow down point, i.e. the point at which the velocity has to be reduced in order to stop at the desired target position.
4. Reaching the target position is signaled by setting the position end flag.

In order to minimize the time until this flag becomes set, the positioning tolerance MVP target reached distance can be chosen with axis parameter 10.

Since the motor typically is assumed not to signal target reached when the target was just passed in a short moment at a high velocity, additionally the maximum target reached velocity (MVP target reached velocity) can be defined by axis parameter 7.

A value of zero for axis parameter 7 is the most universal, since it implies that the motor stands still at the target. But when a fast rising of the position end flag is desired, a higher value for the MVP target reached velocity parameter will save a lot of time. The best value should be tried out in the actual application.

6.5.3 Correlation of Axis Parameters 10 and 7, the Target Position and the Position End Flag

Depending on motor and mechanics, a low oscillation is normal. This can be reduced to at least +/-1 steps. Without oscillation the regulation cannot keep the position!

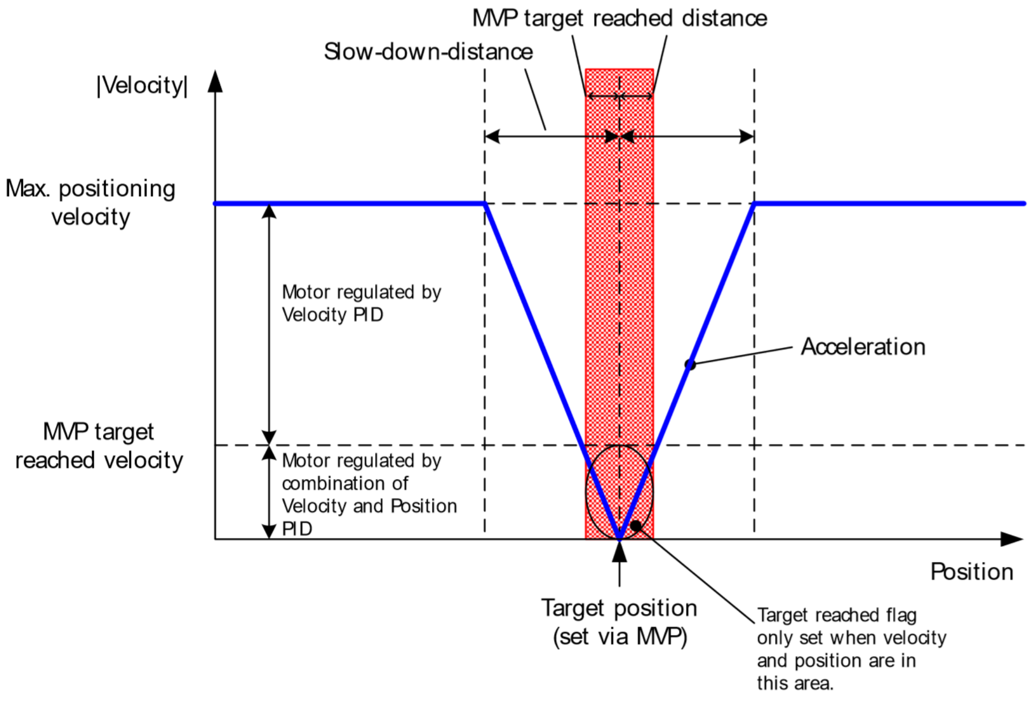

MVP target reached distance
Slow-down-distance
|Velocity|
Max. positioning velocity
Motor regulated by Velocity PID
Acceleration
MVP target reached velocity
Motor regulated by combination of Velocity and Position PID
Position
Target position (set via MVP)
Target reached flag only set when velocity and position are in this area.

Figure 5: Positioning Algorithm

7 TMCL Programming Techniques and Structure

7.1 Initialization

The first task in a TMCL program (like in other programs also) is to initialize all parameters where different values than the default values are necessary. For this purpose, SAP and SGP commands are used.

7.2 Main Loop

Embedded systems normally use a main loop that runs infinitely. This is also the case in a TMCL application that is running stand alone. Normally the auto start mode of the module should be turned on. After power up, the module then starts the TMCL program, which first does all necessary initializations and then enters the main loop, which does all necessary tasks end never ends (only when the module is powered off or reset).

There are exceptions to this, e.g. when TMCL routines are called from a host in direct mode.

So most (but not all) stand alone TMCL programs look like this:

```
  //Initialization
  SAP 4, 0, 50000 //define maximum positioning speed
  SAP 5, 0, 10000 //define maximum acceleration

MainLoop:
  //do something, in this example just running between two positions
  MVP ABS, 0, 5000
  WAIT POS, 0, 0
  MVP ABS, 0, 0
  WAIT POS, 0, 0
  JA MainLoop   //end of the main loop => run infinitely
```

7.3 Using Symbolic Constants

To make your program better readable and understandable, symbolic constants should be taken for all important numerical values that are used in the program. The TMCL-IDE provides an include file with symbolic names for all important axis parameters and global parameters. Please consider the following example:

```
//Define some constants
#include TMCLParam.tmc
MaxSpeed = 50000
MaxAcc = 10000
Position0 = 0
Position1 = 500000

//Initialization
  SAP APMaxPositioningSpeed, Motor0, MaxSpeed
  SAP APMaxAcceleration, Motor0, MaxAcc

MainLoop:
  MVP ABS, Motor0, Position1
  WAIT POS, Motor0, 0
  MVP ABS, Motor0, Position0
  WAIT POS, Motor0, 0
  JA MainLoop
```

Have a look at the file TMCLParam.tmc provided with the TMCL-IDE. It contains symbolic constants that define all important parameter numbers.

Using constants for other values makes it easier to change them when they are used more than once in a program. You can change the definition of the constant and do not have to change all occurrences of it in your program.

7.4 Using Variables

The user variables can be used if variables are needed in your program. They can store temporary values. The commands SGP, GGP and AGP as well as STGP and RSGP are used to work with user variables:

- SGP is used to set a variable to a constant value (e.g. during initialization phase).
- GGP is used to read the contents of a user variable and to copy it to the accumulator register for further usage.
- AGP can be used to copy the contents of the accumulator register to a user variable, e.g. to store the result of a calculation.
- The STGP command stores the contents of a user varaible in the EEPROM.
- The RSGP command copies the value stored in the EEPROM back to the user variable.
- Global parameter 85 controls if user variables will be restored from the EEPROM automatically on startup (default setting) or not (user variables will then be initialized with 0 instead).

Please see the following example:

```
MyVariable = 42
//Use a symbolic name for the user variable
//(This makes the program better readable and understandable.)

SGP MyVariable, 2, 1234   //Initialize the variable with the value 1234
...
...
GGP MyVariable, 2    //Copy contents of variable to accumulator register
CALC MUL, 2          //Multiply accumulator register with two
AGP MyVariable, 2    //Store contents of accumulator register to variable
...
...
```

Furthermore, these variables can provide a powerful way of communication between a TMCL program running on a module and a host. The host can change a variable by issuing a direct mode SGP command (remember that while a TMCL program is running direct mode commands can still be executed, without interfering with the running program). If the TMCL program polls this variable regularly it can react on such changes of its contents.

The host can also poll a variable using GGP in direct mode and see if it has been changed by the TMCL program.

7.5 Using Subroutines

The CSUB and RSUB commands provide a mechanism for using subroutines. The CSUB command branches to the given label. When an RSUB command is executed the control goes back to the command that follows the CSUB command that called the subroutine.

This mechanism can also be nested. From a subroutine called by a CSUB command other subroutines can be called. In the current version of TMCL eight levels of nested subroutine calls are allowed.

7.6 Combining Direct Mode and Standalone Mode

Direct mode and standalone mode can also be combined. When a TMCL program is being executed in standalone mode, direct mode commands are also processed (and they do not disturb the flow of the program running in standalone mode). So, it is also possible to query e.g. the actual position of the motor in direct mode while a TMCL program is running.

Communication between a program running in standalone mode and a host can be done using the TMCL user variables. The host can then change the value of a user variable (using a direct mode SGP command) which is regularly polled by the TMCL program (e.g. in its main loop) and so the TMCL program can react on such changes. Vice versa, a TMCL program can change a user variable that is polled by the host (using a direct mode GGP command).

A TMCL program can be started by the host using the run command in direct mode. This way, also a set of TMCL routines can be defined that are called by a host. In this case it is recommended to place JA commands at the beginning of the TMCL program that jump to the specific routines. This assures that the entry addresses of the routines will not change even when the TMCL routines are changed (so when changing the TMCL routines the host program does not have to be changed).

Example:

```
//Jump commands to the TMCL routines
Func1:  JA Func1Start
Func2:  JA Func2Start
Func3:  JA Func3Start

Func1Start:
    MVP ABS, 0, 1000
    WAIT POS, 0, 0
    MVP ABS, 0, 0
    WAIT POS, 0, 0
    STOP

Func2Start:
    ROL 0, 500
    WAIT TICKS, 0, 100
    MST 0
    STOP

Func3Start:
    ROR 0, 1000
    WAIT TICKS, 0, 700
    MST 0
    STOP
```

This example provides three very simple TMCL routines. They can be called from a host by issuing a run command with address 0 to call the first function, or a run command with address 1 to call the second function, or a run command with address 2 to call the third function. You can see the addresses of the TMCL labels (that are needed for the run commands) by using the ”Generate symbol file function” of the TMCL-IDE.

7.7 Make the TMCL Program start automatically

For stand-alone operation the module has to start the TMCL program in its memory automatically after power-on. In order to achieve this, switch on the Autostart option of the module. This is controlled by global parameter #77. There are different ways to switch on the Autostart option:

- Execute the command SGP 77, 0, 1 in direct mode (using the Direct Mode tool in the TMCL-IDE).
- Use the Global Parameters tool in the TMCL-IDE to set global parameter #77 to 1.
- Use the Autostart entry in the TMCL menu of the TMCL Creator in the TMCL-IDE. Go to the Autostart entry in the TMCL menu and select "’On"’.

8 Figures Index

9 Tables Index

10 Supplemental Directives

10.1 Producer Information

10.2 Copyright

TRINAMIC owns the content of this user manual in its entirety, including but not limited to pictures, logos, trademarks, and resources. © Copyright 2018 TRINAMIC. All rights reserved. Electronically published by TRINAMIC, Germany.

Redistributions of source or derived format (for example, Portable Document Format or Hypertext Markup Language) must retain the above copyright notice, and the complete Datasheet User Manual documentation of this product including associated Application Notes; and a reference to other available product-related documentation.

10.3 Trademark Designations and Symbols

Trademark designations and symbols used in this documentation indicate that a product or feature is owned and registered as trademark and/or patent either by TRINAMIC or by other manufacturers, whose products are used or referred to in combination with TRINAMIC's products and TRINAMIC's product documentation.

This TMCL™ Firmware Manual is a non-commercial publication that seeks to provide concise scientific and technical user information to the target user. Thus, trademark designations and symbols are only entered in the Short Spec of this document that introduces the product at a quick glance. The trademark designation /symbol is also entered when the product or feature name occurs for the first time in the document. All trademarks and brand names used are property of their respective owners.

10.4 Target User

The documentation provided here, is for programmers and engineers only, who are equipped with the necessary skills and have been trained to work with this type of product.

The Target User knows how to responsibly make use of this product without causing harm to himself or others, and without causing damage to systems or devices, in which the user incorporates the product.

10.5 Disclaimer: Life Support Systems

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.

Information given in this document is believed to be accurate and reliable. However, no 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.

10.6 Disclaimer: Intended Use

The data specified in this user manual is intended solely for the purpose of product description. No representations or warranties, either express or implied, of merchantability, fitness for a particular purpose or of any other nature are made hereunder with respect to information/specification or the products to which information refers and no guarantee with respect to compliance to the intended use is given.

In particular, this also applies to the stated possible applications or areas of applications of the product. TRINAMIC products are not designed for and must not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (safety-Critical Applications) without TRINAMIC's specific written consent.

TRINAMIC products are not designed nor intended for use in military or aerospace applications or environments or in automotive applications unless specifically designated for such use by TRINAMIC. TRINAMIC conveys no patent, copyright, mask work right or other trade mark right to this product. TRINAMIC assumes no liability for any patent and/or other trade mark rights of a third party resulting from processing or handling of the product and/or any other use of the product.

10.7 Collateral Documents & Tools

This product documentation is related and/or associated with additional tool kits, firmware and other items, as provided on the product page at: www.trinamic.com.

11 Revision History

11.1 Firmware Revision

Version	Date	Author	Description
2.00	26.10.2018	ED	First release.

Table 17: Firmware Revision

11.2 Document Revision

Version	Date	Author	Description
1.00	29.10.2018	ED	First release.

Table 18: Document Revision