V 1.07

# TMCL™ FIRMWARE MANUAL

## TMCM-1180

## PD86-1180

**1-axis stepper controller / driver**
**5.5A RMS/ 24 or 48V DC**
**USB, RS232, RS485, and CAN**

coolStep™

stallGuard2™

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

**www.trinamic.com**

# Table of contents

# 1 Life support policy

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

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

# 2 Features

The PD86-1180 is a full mechatronic device consisting of a NEMA 34 (flange size 86mm) stepper motor, controller/driver electronics and integrated encoder.

**Applications**

- Powerful single-axis stepper motor solutions
- Encoder feedback for high reliability operation

**Electrical data**

- Supply voltage: +24V DC or +48V DC nominal
- Motor current: up to 5.5A RMS (programmable)

**PANdrive motor**

- Two phase bipolar stepper motor with up to 5.5A RMS nom. coil current
- Holding torque: 7Nm

**Integrated encoder**

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

**Integrated motion controller**

- Motion profile calculation in real-time (TMC428 motion controller)
- On the fly alteration of motor parameters (e.g. position, velocity, acceleration)
- High performance ARM7 microcontroller for overall system control and serial communication protocol handling

**Integrated bipolar stepper motor driver**

- Up to 256 microsteps per full step
- High-efficient operation, low power dissipation (MOSFETs with low $R_{DS(ON)}$)
- Dynamic current control
- Integrated protection
- Automatic load dependent motor current adaptation for reduced power consumption and heat dissipation (coolStep™)

**Interfaces**

- inputs for stop switches (left and right) and home switch
- general purpose inputs and 2 general purpose outputs
- USB, RS232, RS485 and CAN (2.0B up to 1Mbit/s) communication interfaces

**Safety features**

- Shutdown input. The driver will be disabled in hardware as long as this pin is left open or shorted to ground
- Separate supply voltage inputs for driver and digital logic – driver supply voltage may be switched off externally while supply for digital logic and therefore digital logic remains active

**Software**

- Available with TMCL™ or CANopen
- TMCL™: standalone operation or remote controlled operation
- TMCL™: program memory (non volatile) for up to 2048 TMCL™ commands
- TMCL™: PC-based application development software TMCL-IDE available for free
- CANopen (*under development*): CiA 301 + CiA 402 (homing mode, profile position mode and velocity mode) supported

***Please refer to separate Hardware Manual for further information.***

# 3 Overview

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-1180 consists of two parts, a boot loader and the firmware itself. Whereas the boot loader is installed during production and testing at TRINAMIC and remains normally untouched throughout the whole lifetime, the firmware can be updated by the user. New versions can be downloaded free of charge from the TRINAMIC website (http://www.trinamic.com).

The firmware shipped with this module is related to the standard TMCL™ firmware shipped with most of TRINAMIC modules with regard to protocol and commands. Corresponding, this module is based on the TMC428-I stepper motor controller and the TMC262A-PC power driver and supports the standard TMCL™ with a special range of values.

The TMC262A-PC is a new energy efficient high current high precision microstepping driver IC for bipolar stepper motors and offers TRINAMICs patented coolStep™ feature with its special commands. Please mind this technical innovation.

All commands and parameters available with this unit are explained on the following pages.

# 4 Putting the PD86-1180 into operation

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

The things you need:

- PD83-1180
- Interface (RS232, RS485, USB or CAN) suitable to your PANdrive™ with cables
- Nominal supply voltage +24V DC (+24 or +48V DC) for your module
- TMCL-IDE program and PC
- External encoder optional. The PANdrive™ has an integrated sensOstep™ encoder.

Precautions:

- ***Do not connect or disconnect the PD86-1180 while powered!***
- ***Do not connect or disconnect the motor while powered!***
- ***Do not exceed the maximum power supply of 55V DC.***
- ***Start with power supply OFF!***

## 4.1 Starting up

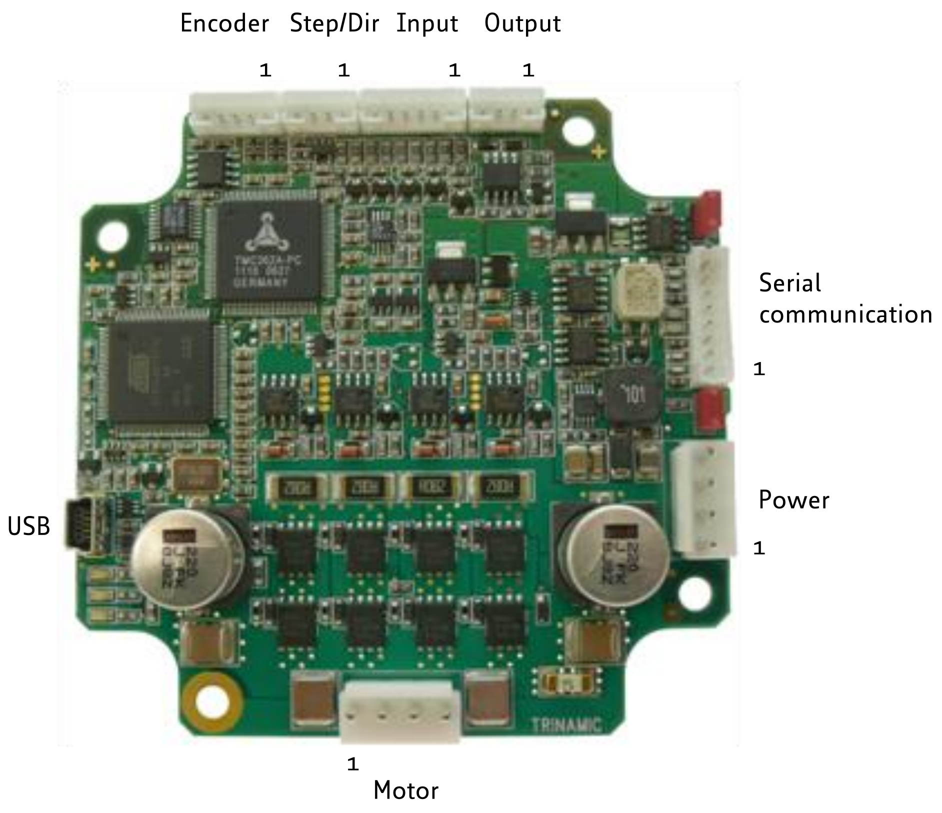


**Figure 4.1: Overview connectors**

1. **Connect the interface**

**a) Connect the RS232, the RS485, or the CAN interface**
A 2mm pitch 8 pin JST B8B-PH-K connector is used for serial communication. With this connector the module supports RS232, RS485, and CAN communication.

Please connect as follows:

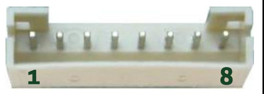
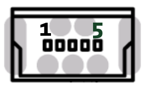
| | Pin | Label | Description |
|---|---|---|---|
|  | 1 | RS232_TxD | RS232 transmit data |
| | 2 | RS232_RxD | RS232 receive data |
| | 3 | GND | Module ground (system and signal ground) |
| | 4 | CAN_H | CAN_H bus line (dominant high) |
| | 5 | CAN_L | CAN_L bus line (dominant low) |
| | 6 | GND | Module ground (system and signal ground) |
| | 7 | RS485+ | RS485 non-inverted bus signal |
| | 8 | RS485- | RS485 inverted bus signal |

**Figure 4.2: RS232, RS485, and CAN connector**

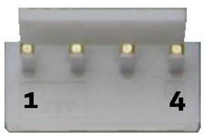
**b) Connect the USB interface**
A 5-pin mini-USB connector is available on the board.

Please connect as follows:

| | Pin | Label | Description |
|---|---|---|---|
|  | 1 | VBUS | +5V power |
| | 2 | D- | Data – |
| | 3 | D+ | Data + |
| | 4 | ID | Not connected |
| | 5 | GND | ground |

2. **Connect the power supply**
A 4-pin JST B04P-VL connector is used for power supply.

Please connect as follows:

| | Pin | Label | Description |
|---|---|---|---|
|  | 1 | $+U_{Driver}$ | Module + driver stage power supply input (nom. +48V DC) |
| | 2 | $+U_{Logic}$ | (Optional) separate digital logic power supply input (nom. +48V DC) |
| | 3 | GND | Module ground (power supply and signal ground) |
| | 4 | GND | Module ground (power supply and signal ground) |

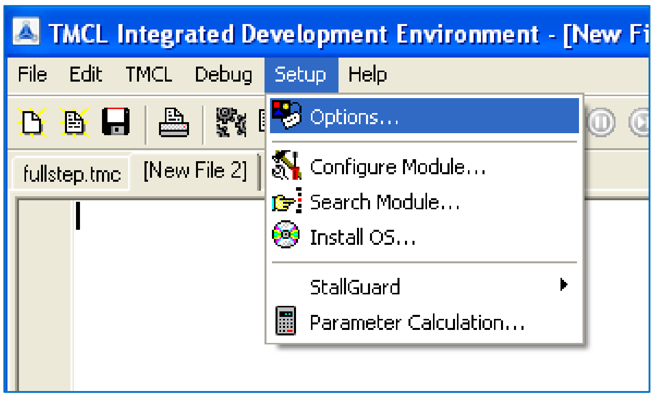
3. **Switch ON the power supply**
The LED for power should glow now. This indicates that the on-board +5V supply is available.

***If this does not occur, switch power OFF and check your connections as well as the power supply.***

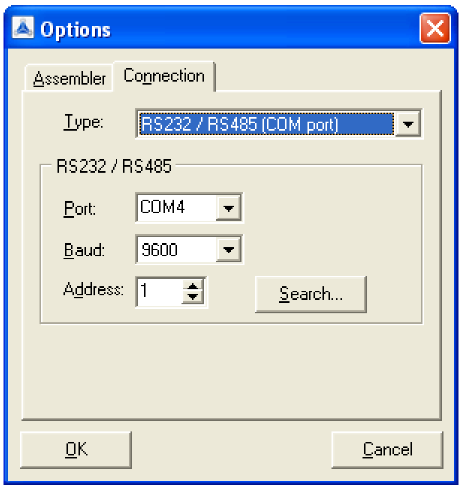
4. **Start the TMCL-IDE software development environment**
The TMCL-IDE is available on the TechLibCD and on www.trinamic.com.

Installing the TMCL-IDE:

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


TMCL Integrated Development Environment - [New Fi
File Edit TMCL Debug Setup Help
Options...
Configure Module...
Search Module...
Install OS...
StallGuard
Parameter Calculation...
fullstep.tmc [New File 2]

- For RS232 and RS485 choose ***COM port*** and ***type*** with the parameters shown below (baud rate 9600). Click *OK*.



Type: RS232 / RS485 (COM port)
RS232 / RS485
Port: COM4
Baud: 9600
Address: 1

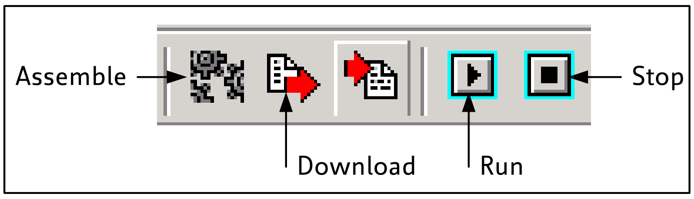
***Please refer to the TMCL-IDE User Manual for more information about connecting the other interfaces (see www.TRINAMIC.com).***

## 4.2 Testing with a simple TMCL™ program

Open the file test2.tmc. Change the *motor number 2* in the second paragraph in *motor number 0* (because there is only one motor involved). Now your test program looks as follows:

A description for the TMCL™ commands can be found in Appendix A.

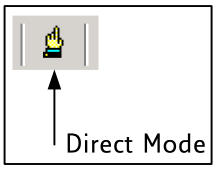
```
//A simple example for using TMCL™ and TMCL-IDE

        ROL 0, 500                  //Rotate motor 0 with speed 500
        WAIT TICKS, 0, 500
        MST 0
        ROR 0, 250                 //Rotate motor 0 with 250
        WAIT TICKS, 0, 500
        MST 0

        SAP 4, 0, 500               //Set max. Velocity
        SAP 5, 0, 50                //Set max. Acceleration
Loop:   MVP ABS, 0, 10000           //Move to Position 10000
        WAIT POS, 0, 0              //Wait until position reached
        MVP ABS, 0, -10000          //Move to Position -10000
        WAIT POS, 0, 0              //Wait until position reached
        JA Loop                     //Infinite Loop
```



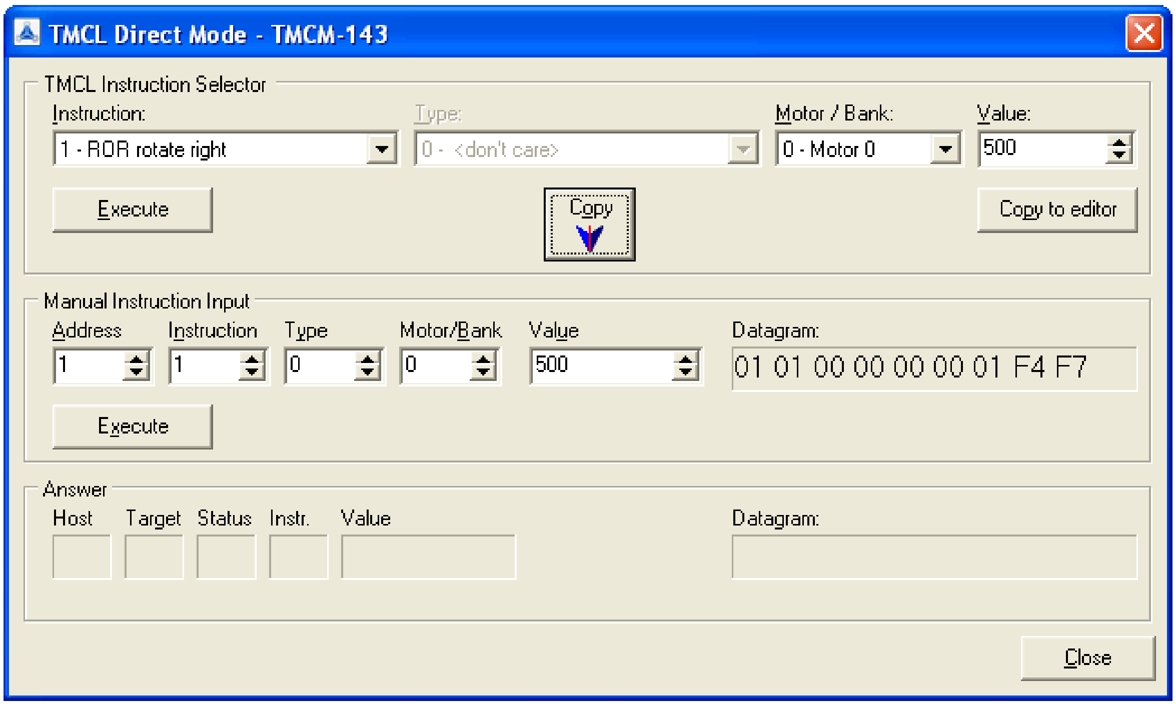
1. Click on Icon ***Assemble*** to convert the TMCL™ into machine code.
2. Then download the program to the TMCM-1180 module via the icon ***Download***.
3. Press icon ***Run***. The desired program will be executed.
4. Click ***Stop*** button to stop the program.

## 4.3 Operating the module in direct mode

1. Start TMCL™ ***Direct Mode***.



2. If the communication is established the PD86-1180 is automatically detected. ***If the module is not detected, please check all points above (cables, interface, power supply, COM port, baud rate).***
3. Issue a command by choosing ***Instruction***, ***Type*** (if necessary), ***Motor***, and ***Value*** and click ***Execute*** to send it to the module.



<u>Examples:</u>

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

*<u>Attention:</u>*
*Please mind the chapter 3 (programming techniques) of the TMCL-IDE User Manual on [www.trinamic.com](http://www.trinamic.com). Here you will find information about creating general structures of TMCL-programs. In particular initialization, main loop, symbolic constants, variables, and subroutines are described there. Further you can learn how to mix direct mode and stand alone mode.*

*Chapter 6 (axis parameters) includes a diagram which points the coolStep™ related axis parameters and their functions. This can help you configuring your module to meet your needs.*

***You will find a description of all TMCL™ commands in the following chapters.***

# 5 TMCL™ and TMCL-IDE

The TMCM-1180 supports TMCL™ direct mode (binary commands or ASCII interface) and standalone TMCL™ program execution. You can store up to 2048 TMCL™ instructions on it.

In direct mode and most cases the TMCL™ communication over RS485, RS232, USB or CAN 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 TMCL-1180. The TMCL™ interpreter on the module will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over RS485/RS232/USB/CAN 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.

## 5.1 Binary command format

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

This is different when communicating is via the CAN bus. Address and checksum are included in the CAN standard and do not have to be supplied by the user.

**The binary command format for RS232/RS485/USB is as follows:**

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

- The checksum is calculated by adding up all the other bytes using an 8-bit addition.
- When using CAN bus, just leave out the first byte (module address) and the last byte (checksum).

**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 to show how to do this:

- in C:

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

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

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

- in Delphi:

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

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

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

## 5.2 Reply format

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

**The reply format for RS485/RS232/USB is as follows:**

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

- The checksum is also calculated by adding up all the other bytes using an 8-bit addition.
- When using CAN bus, the first byte (reply address) and the last byte (checksum) are left out.
- Do not send the next command before you have received the reply!

### 5.2.1 Status codes

The reply contains a status code.

**The status code can have one of the following values:**

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

## 5.3 Standalone applications

The module is equipped with an EEPROM for storing TMCL™ applications. You can use TMCL-IDE for developing standalone TMCL™ applications. You can load them down into the EEPROM and then 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.

## 5.4 TMCL™ command overview

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

### 5.4.1 Motion commands

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

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

### 5.4.2 Parameter commands

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

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

### 5.4.3 I/O port commands

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

| Mnemonic | Command number | Meaning |
|---|---|---|
| SIO | 14 | Set output |
| GIO | 15 | Get input |

### 5.4.4 Control commands

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

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

### 5.4.5 Calculation commands

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

| Mnemonic | Command number | Meaning |
|---|---|---|
| CALC | 19 | Calculate using the accumulator and a constant value |
| CALCX | 33 | Calculate using the accumulator and the X register |
| AAP | 34 | Copy accumulator to an axis parameter |
| AGP | 35 | Copy accumulator to a global parameter |
| ACO | 39 | Copy accu to coordinate |

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

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

### 5.4.6 Interrupt commands

Due to some customer requests, interrupt processing has been introduced in the TMCL™ firmware for ARM based modules from revision 4.23 on. The TMCL-IDE supports the following commands from version 1.78 on.

| **Mnemonic** | **Command number** | **Meaning** |
|---|---|---|
| EI | 25 | Enable interrupt |
| DI | 26 | Disable interrupt |
| VECT | 37 | Set interrupt vector |
| RETI | 38 | Return from interrupt |

#### 5.4.6.1 Interrupt types:

There are many different interrupts in TMCL™, like timer interrupts, stop switch interrupts, position reached interrupts, and input pin change interrupts. Each of these interrupts has its own interrupt vector. Each interrupt vector is identified by its interrupt number. Please use the TMCL™ include file *Interrupts.inc* for symbolic constants of the interrupt numbers.

#### 5.4.6.2 Interrupt processing:

When an interrupt occurs and this interrupt is enabled and a valid interrupt vector has been defined for that interrupt, the normal TMCL™ program flow will be interrupted and the interrupt handling routine will be called. Before an interrupt handling routine gets called, the context of the normal program will be saved automatically (i.e. accumulator register, X register, TMCL™ flags).

There is no interrupt nesting, i.e. all other interrupts are disabled while an interrupt handling routine is being executed.

On return from an interrupt handling routine, the context of the normal program will automatically be restored and execution of the normal program will be continued.

#### 5.4.6.3 Interrupt vectors:

The following table shows all interrupt vectors that can be used.

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | Target position reached |
| 15 | stallGuard™ |
| 21 | Deviation |
| 27 | Left stop switch |
| 28 | Right stop switch |
| 39 | Input change 0 |
| 40 | Input change 1 |
| 255 | Global interrupts |

#### 5.4.6.4 Further configuration of interrupts

Some interrupts need further configuration (e.g. the timer interval of a timer interrupt). This can be done using SGP commands with parameter bank 3 (SGP <type>, 3, <value>). Please refer to the SGP command (paragraph 5.7.9) for further information about that.

#### 5.4.6.5 Using interrupts in TMCL™

To use an interrupt the following things have to be done:

- Define an interrupt handling routine using the VECT command.
- If necessary, configure the interrupt using an SGP <type>, 3, <value> command.
- Enable the interrupt using an EI <interrupt> command.
- Globally enable interrupts using an EI 255 command.
- An interrupt handling routine must always end with a RETI command

The following example shows the use of a timer interrupt:

```
    VECT 0, Timer0Irq   //define the interrupt vector
    SGP 0, 3, 1000      //configure the interrupt: set its period to 1000ms
    EI 0                //enable this interrupt
    EI 255              //globally switch on interrupt processing

//Main program: toggles output 3, using a WAIT command for the delay
Loop:
    SIO 3, 2, 1
    WAIT TICKS, 0, 50
    SIO 3, 2, 0
    WAIT TICKS, 0, 50
    JA Loop

//Here is the interrupt handling routine
Timer0Irq:
    GIO 0, 2            //check if OUT0 is high
    JC NZ, Out0Off      //jump if not
    SIO 0, 2, 1         //switch OUT0 high
    RETI                //end of interrupt
Out0Off:
    SIO 0, 2, 0         //switch OUT0 low
    RETI                //end of interrupt
```

In the above example, the interrupt numbers are used directly. To make the programme better readable please use the provided include file *Interrupts.inc.* This file defines symbolic constants for all interrupt numbers which can be used in all interrupt commands. The beginning of the above programme then looks like the following:

```
#include Interrupts.inc
    VECT TI_TIMER0, Timer0Irq
    SGP TI_TIMER0, 3, 1000
    EI TI_TIMER0
    EI TI_GLOBAL
```

***Please also take a look at the other example programs.***

## 5.5 TMCL™ list of commands

The following TMCL™ commands are currently supported:

| Command | Number | Parameter | Description |
|---|---|---|---|
| ROR | 1 | <motor number>, <velocity> | Rotate right with specified velocity |
| ROL | 2 | <motor number>, <velocity> | Rotate left with specified velocity |
| MST | 3 | <motor number> | Stop motor movement |
| MVP | 4 | ABS\|REL\|COORD, <motor number>, <position\|offset> | Move to position (absolute or relative) |
| SAP | 5 | <parameter>, <motor number>, <value> | Set axis parameter (motion control specific settings) |
| GAP | 6 | <parameter>, <motor number> | Get axis parameter (read out motion control specific settings) |
| STAP | 7 | <parameter>, <motor number> | Store axis parameter permanently (non volatile) |
| RSAP | 8 | <parameter>, <motor number> | Restore axis parameter |
| SGP | 9 | <parameter>, <bank number>, value | Set global parameter (module specific settings e.g. communication settings or TMCL™ user variables) |
| GGP | 10 | <parameter>, <bank number> | Get global parameter (read out module specific settings e.g. communication settings or TMCL™ user variables) |
| STGP | 11 | <parameter>, <bank number> | Store global parameter (TMCL™ user variables only) |
| RSGP | 12 | <parameter>, <bank number> | Restore global parameter (TMCL™ user variable only) |
| RFS | 13 | START\|STOP\|STATUS, <motor number> | Reference search |
| SIO | 14 | <port number>, <bank number>, <value> | Set digital output to specified value |
| GIO | 15 | <port number>, <bank number> | Get value of analogue/digital input |
| CALC | 19 | <operation>, <value> | Process accumulator & value |
| COMP | 20 | <value> | Compare accumulator <-> value |
| JC | 21 | <condition>, <jump address> | Jump conditional |
| JA | 22 | <jump address> | Jump absolute |
| CSUB | 23 | <subroutine address> | Call subroutine |
| RSUB | 24 | | Return from subroutine |
| EI | 25 | <interrupt number> | Enable interrupt |
| DI | 26 | <interrupt number> | Disable interrupt |
| WAIT | 27 | <condition>, <motor number>, <ticks> | Wait with further program execution |
| STOP | 28 | | Stop program execution |
| SCO | 30 | <coordinate number>, <motor number>, <position> | Set coordinate |
| GCO | 31 | <coordinate number>, <motor number> | Get coordinate |
| CCO | 32 | <coordinate number>, <motor number> | Capture coordinate |
| CALCX | 33 | <operation> | Process accumulator & X-register |
| AAP | 34 | <parameter>, <motor number> | Accumulator to axis parameter |
| AGP | 35 | <parameter>, <bank number> | Accumulator to global parameter |
| VECT | 37 | <interrupt number>, <label> | Set interrupt vector |
| RETI | 38 | | Return from interrupt |
| ACO | 39 | <coordinate number>, <motor number> | Accu to coordinate |

TMCL™ **control commands:**

| Instruction | Description | Type | Mot/Bank | Value |
|---|---|---|---|---|
| 128 – stop application | a running TMCL™ standalone application is stopped | (don't care) | (don't care) | (don't care) |
| 129 – run application | TMCL™ execution is started (or continued) | 0 - run from current address<br>1 - run from specified address | (don't care) | (don't care)<br><br>starting address |
| 130 – step application | only the next command of a TMCL™ application is executed | (don't care) | (don't care) | (don't care) |
| 131 – reset application | the program counter is set to zero, and the standalone application is stopped (when running or stepped) | (don't care) | (don't care) | (don't care) |
| 132 – start download mode | target command execution is stopped and all following commands are transferred to the TMCL™ memory | (don't care) | (don't care) | starting address of the application |
| 133 – quit download mode | target command execution is resumed | (don't care) | (don't care) | (don't care) |
| 134 – read TMCL™ memory | the specified program memory location is read | (don't care) | (don't care) | <memory address> |
| 135 – get application status | one of these values is returned:<br>0 – stop<br>1 – run<br>2 – step<br>3 – reset | (don't care) | (don't care) | (don't care) |
| 136 – get firmware version | return the module type and firmware revision either as a string or in binary format | 0 – string<br>1 – binary | (don't care) | (don't care) |
| 137 – restore factory settings | reset all settings stored in the EEPROM to their factory defaults<br>This command does not send back a reply. | (don't care) | (don't care) | must be 1234 |
| 138 – reserved | | | | |
| 139 – enter ASCII mode | Enter ASCII command line (see chapter 5.6) | (don't care) | (don't care) | (don't care) |

## 5.6 The ASCII interface

Since TMCL™ V3.21 there is also an ASCII interface that can be used to communicate with the module and to send some commands as text strings.

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

### 5.6.1 Format of the command line

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

**Here are some examples for valid command lines:**

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

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

### 5.6.2 Format of a reply

After executing the command the module sends back a reply in ASCII format. This reply consists of:
- the address character of the host (host address that can be set in the module)
- the address character of the module
- the status code as a decimal number
- the return value of the command as a decimal number
- a <CR> character

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

### 5.6.3 Commands that can be used in ASCII mode

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

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

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

### 5.6.4 Configuring the ASCII interface

The module can be configured so that it starts up either in binary mode or in ASCII mode. ***Global parameter 67 is used for this purpose*** (please see also chapter 7.1).

Bit 0 determines the startup mode: if this bit is set, the module starts up in ASCII mode, else it will start up in binary mode (default).

Bit 4 and Bit 5 determine how the characters that are entered are echoed back. Normally, both bits are set to zero. In this case every character that is entered is echoed back when the module is addressed. Character can also be erased using the backspace character (press the backspace key in a terminal program).

When bit 4 is set and bit 5 is clear the characters that are entered are not echoed back immediately but the entire line will be echoed back after the <CR> character has been sent.

When bit 5 is set and bit 4 is clear there will be no echo, only the reply will be sent. This may be useful in RS485 systems.

## 5.7 Commands

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

### 5.7.1 ROR (rotate right)

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

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

The module is based on the TMC428-I stepper motor controller and the TMC262A-PC power driver. This makes possible choosing a velocity between 0 and 2047.

**Related commands:** ROL, MST, SAP, GAP

**Mnemonic:** ROR 0, <velocity>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 1 | (don't care) | 0* | <velocity><br>0… 2047 |

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Rotate right, velocity = 350
*Mnemonic:* ROR 0, 350

*Binary:*

| **Byte Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $01 | $00 | $02 | $00 | $00 | $01 | $5e | $62 |

### 5.7.2 ROL (rotate left)

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

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

The module is based on the TMC428-I stepper motor controller and the TMC262A-PC power driver. This makes possible choosing a velocity between 0 and 2047.

**Related commands:** ROR, MST, SAP, GAP

**Mnemonic:** ROL 0, <velocity>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 2 | (don't care) | 0* | <velocity> 0… 2047 |

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Rotate left, velocity = 1200
*Mnemonic:* ROL 0, 1200

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| Value (hex) | $01 | $02 | $00 | $00 | $00 | $00 | $04 | $b0 | $b8 |

### 5.7.3 MST (motor stop)

With this command the motor will be instructed to stop. Please note: depending on motor speed a hard stop might lead to step losses.

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

**Related commands:** ROL, ROR, SAP, GAP

**Mnemonic:** MST 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 3 | (don't care) | 0* | (don't care) |

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Stop motor
*Mnemonic:* MST 0

*Binary:*

| **Byte Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $03 | $00 | $00 | $00 | $00 | $00 | $00 | $05 |

### 5.7.4 MVP (move to position)

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

**Three operation types are available:**

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

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

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

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

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

**Binary representation:**

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

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Move motor to (absolute) position 90000
*Mnemonic:* MVP ABS, 0, 9000

*Binary:*

| **Byte Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $04 | $00 | $00 | $00 | $01 | $5f | $90 | $f6 |

**Example:**

Move motor from current position 1000 steps backward (move relative –1000)
*Mnemonic:* MVP REL, 0, -1000

*Binary:*

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

**Example:**

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

*Binary:*

| **Byte Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $04 | $02 | $00 | $00 | $00 | $00 | $08 | $11 |

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

### 5.7.5 SAP (set axis parameter)

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

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

**Related commands:** GAP, STAP, RSAP, AAP

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

**Binary representation:**

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

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**List of parameters, which can be used for SAP:**

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 0 | target (next) position | The desired position in position mode (see ramp mode, no. 138). | $\pm 2^{23}$ [µsteps] |
| 1 | actual position | The current position of the motor. Should only be overwritten for reference point setting. | $\pm 2^{23}$ [µsteps] |
| 2 | target (next) speed | The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 3 | actual speed | The current rotation speed. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 4 | maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet for calculation of physical units. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 5 | maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 428 datasheet for calculation of physical units. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 6 | absolute max. current (CS / Current Scale) | The most important motor setting, since too high values might cause motor damage! The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on). | 0… 255 $\left[\frac{\text{max. module current}}{255}\right]$ |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 7 | standby current | The current limit two seconds after the motor has stopped. | 0… 255 $\left[\frac{\text{max. module current}}{255}\right]$ |
| 12 | right limit switch disable | If set, deactivates the stop function of the right switch | 0/1 |
| 13 | left limit switch disable | Deactivates the stop function of the left switch resp. reference switch if set. | 0/1 |
| 130 | minimum speed | Should always be set 1 to ensure exact reaching of the target position. Do not change! | 0… 2047 $\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 138 | ramp mode | Automatically set when using ROR, ROL, MST and MVP. 0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided. 2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed. For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected. | 0/1/2 |
| 140 | microstep resolution | 0 full step | 0… 8 |
| | | 1 half step | |
| | | 2 4 microsteps | |
| | | 3 8 microsteps | |
| | | 4 16 microsteps | |
| | | 5 32 microsteps | |
| | | 6 64 microsteps | |
| | | 7 128 microsteps | |
| | | 8 256 microsteps | |
| 141 | ref. switch tolerance | For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop. | 0… 4095 [µsteps] |
| 149 | soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. | 0/1 |
| 153 | ramp divisor | The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one). | 0… 13 |
| 154 | pulse divisor | The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one). | 0… 13 |
| 160 | step interpolation enable | Step interpolation is supported with a 16 microstep setting only. In this setting, each step impulse at the input causes the execution of 16 times 1/256 microsteps. This way, a smooth motor movement like in 256 microstep resolution is achieved. 0 – step interpolation off 1 – step interpolation on | 0/1 |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 161 | double step enable | Every edge of the cycle releases a step/microstep. *It does not make sense to activate this parameter for internal use.*<br>Double step enable can be used with Step/Dir interface.<br>0 – double step off<br>1 – double step on | 0/1 |
| 162 | chopper blank time | Selects the comparator *blank time*. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. For low current drivers, a setting of 1 or 2 is good. For higher current applications like the TMCM-1180 a setting of 2 or 3 will be required. | 0… 3 |
| 163 | chopper mode | Selection of the chopper mode:<br>0 – spread cycle<br>1 – classic const. off time | 0/1 |
| 164 | chopper hysteresis decrement | Hysteresis decrement setting. This setting determines the slope of the hysteresis during on time and during fast decay time.<br>0 – fast decrement<br>3 – very slow decrement | 0… 3 |
| 165 | chopper hysteresis end | Hysteresis end setting. Sets the hysteresis end value after a number of decrements. Decrement interval time is controlled by axis parameter 164.<br>-3… -1: negative hysteresis end setting<br>0: zero hysteresis end setting<br>1… 12: positive hysteresis end setting | -3… 12 |
| 166 | chopper hysteresis start | Hysteresis start setting. Please remark, that this value is an offset to the hysteresis end value. | 0… 8 |
| 167 | chopper off time | The off time setting controls the minimum chopper frequency. An off time within the range of 5µs to 20µs will fit.<br>Off time setting for constant $t_{OFF}$ chopper:<br>$N_{CLK}$= 12 + 32*$t_{OFF}$ (Minimum is 64 clocks)<br>Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel. | 0 / 2… 15 |
| 168 | smartEnergy current minimum (SEIMIN) | Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value.<br>minimum motor current:<br>0 – 1/2 of CS<br>1 – 1/4 of CS | 0/1 |
| 169 | smartEnergy current down step | Sets the number of stallGuard2™ readings above the upper threshold necessary for each current decrement of the motor current.<br>Number of stallGuard2™ measurements per decrement:<br>Scaling: 0… 3: 32, 8, 2, 1<br>0: slow decrement<br>3: fast decrement | 0… 3 |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 170 | smartEnergy hysteresis | Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased.<br>Hysteresis:<br>(smartEnergy hysteresis value + 1) * 32<br>Upper stallGuard threshold:<br>(smartEnergy hysteresis start + smartEnergy hysteresis + 1) * 32 | 0… 15 |
| 171 | smartEnergy current up step | Sets the current increment step. The current becomes incremented for each measured stallGuard2™ value below the lower threshold (see smartEnergy hysteresis start).<br>current increment step size:<br>Scaling: 0… 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load | 1… 3 |
| 172 | smartEnergy hysteresis start | The lower threshold for the stallGuard2™ value (see smart Energy current up step). | 0… 15 |
| 173 | stallGuard2™ filter enable | Enables the stallGuard2™ filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps.<br>*In most cases it is expedient to set the filtered mode before using coolStep™.*<br>*Use the standard mode for step loss detection.*<br>0 – standard mode<br>1 – filtered mode | 0/1 |
| 174 | stallGuard2™ threshold | This signed value controls stallGuard2™ *threshold* level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2™ less sensitive and requires more torque to indicate a stall.<br>0 – Indifferent value<br>1… 63 – less sensitivity<br>-1… -64 – higher sensitivity | -64… 63 |
| 175 | slope control high side | Determines the slope of the motor driver outputs. *Set to 2 or 3 for this module or rather use the default value.*<br>0: lowest slope<br>3: fastest slope | 0… 3 |
| 176 | slope control low side | Determines the slope of the motor driver outputs. *Set identical to slope control high side.* | 0… 3 |
| 177 | short protection disable | 0: Short to GND protection is on<br>1: Short to GND protection is disabled<br>*Use default value!* | 0/1 |
| 178 | short detection timer | 0: 3.2µs<br>1: 1.6µs<br>2: 1.2µs<br>3: 0.8µs<br>*Use default value!* | 0..3 |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 181 | stop on stall | Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2™ load value reaches zero. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed. | 0… 255 $\left[\frac{\text{max. module current}}{255}\right]$ |
| 193 | referencing mode | 1 – Only the left reference switch is searched.<br>2 – The right switch is searched and afterwards the left switch is searched.<br>3 – Three-switch-mode: the right switch is searched first and afterwards the reference switch will be searched. | 1/2/3 |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. | 0… 2047 |
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. | 0… 2047 |
| 204 | freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. | 0… 65535<br>0 = never<br>[msec] |
| 209 | encoder position | The value of an encoder register can be read out or written. | [encoder steps] |
| 210 | encoder prescaler | Prescaler for the encoder. | |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. | 0… 65535<br><br>[encoder steps] |
| 214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). | 1… 65535<br>[10msec] |

**Example:**

Set the absolute maximum current of motor to 200mA
*Mnemonic:* SAP 6, 0, 200

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| Value (hex) | $01 | $05 | $06 | $00 | $00 | $00 | $00 | $c8 | $d5 |

### 5.7.6 GAP (get axis parameter)

Most parameters of the TMCM-1180 can be adjusted individually for the axis. With this parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditioned jumps). In direct mode the value read is only output in the "value" field of the reply (without affecting the accumulator).

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

**Related commands:** SAP, STAP, AAP, RSAP

**Mnemonic:** GAP <parameter number>, 0

**Binary representation:**

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

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**List of parameters, which can be used for GAP:**

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 0 | target (next) position | The desired position in position mode (see ramp mode, no. 138). | $\pm 2^{23}$ [µsteps] |
| 1 | actual position | The current position of the motor. Should only be overwritten for reference point setting. | $\pm 2^{23}$ [µsteps] |
| 2 | target (next) speed | The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 3 | actual speed | The current rotation speed. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 4 | maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet for calculation of physical units. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 5 | maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 428 datasheet for calculation of physical units. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 6 | absolute max. current (CS / Current Scale) | The most important motor setting, since too high values might cause motor damage! The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on). | 0… 255 $\left[\frac{\text{max. module current}}{255}\right]$ |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 7 | standby current | The current limit two seconds after the motor has stopped. | 0… 255 $\left[\frac{\text{max. module current}}{255}\right]$ |
| 8 | target pos. reached | Indicates that the actual position equals the target position. | 0/1 |
| 9 | ref. switch status | The logical state of the reference (left) switch. See the TMC 428 data sheet for the different switch modes. The default has two switch modes: the left switch as the reference switch, the right switch as a limit (stop) switch. | 0/1 |
| 10 | right limit switch status | The logical state of the (right) limit switch. | 0/1 |
| 11 | left limit switch status | The logical state of the left limit switch (in three switch mode) | 0/1 |
| 12 | right limit switch disable | If set, deactivates the stop function of the right switch | 0/1 |
| 13 | left limit switch disable | Deactivates the stop function of the left switch resp. reference switch if set. | 0/1 |
| 130 | minimum speed | Should always be set 1 to ensure exact reaching of the target position. Do not change! | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 135 | actual acceleration | The current acceleration (read only). | 0… 2047* |
| 138 | ramp mode | Automatically set when using ROR, ROL, MST and MVP.<br>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.<br>2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.<br>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected. | 0/1/2 |
| 140 | microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps | 0… 8 |
| 141 | ref. switch tolerance | For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop. | 0… 4095 [µsteps] |
| 149 | soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. | 0/1 |
| 153 | ramp divisor | The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one). | 0… 13 |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 154 | pulse divisor | The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one). | 0… 13 |
| 160 | step interpolation enable | Step interpolation is supported with a 16 microstep setting only. In this setting, each step impulse at the input causes the execution of 16 times 1/256 microsteps. This way, a smooth motor movement like in 256 microstep resolution is achieved.<br>0 – step interpolation off<br>1 – step interpolation on | 0/1 |
| 161 | double step enable | Every edge of the cycle releases a step/microstep. *It does not make sense to activate this parameter for internal use.*<br>Double step enable can be used with Step/Dir interface.<br>0 – double step off<br>1 – double step on | 0/1 |
| 162 | chopper blank time | Selects the comparator *blank time*. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. For low current drivers, a setting of 1 or 2 is good. For higher current applications like the TMCM-1180 a setting of 2 or 3 will be required. | 0… 3 |
| 163 | chopper mode | Selection of the chopper mode:<br>0 – spread cycle<br>1 – classic const. off time | 0/1 |
| 164 | chopper hysteresis decrement | Hysteresis decrement setting. This setting determines the slope of the hysteresis during on time and during fast decay time.<br>0 – fast decrement<br>3 – very slow decrement | 0… 3 |
| 165 | chopper hysteresis end | Hysteresis end setting. Sets the hysteresis end value after a number of decrements. Decrement interval time is controlled by axis parameter 164.<br>-3… -1 : negative hysteresis end setting<br>0 : zero hysteresis end setting<br>1… 12 : positive hysteresis end setting | -3… 12 |
| 166 | chopper hysteresis start | Hysteresis start setting. Please remark, that this value is an offset to the hysteresis end value. | 0… 8 |
| 167 | chopper off time | The off time setting controls the minimum chopper frequency. An off time within the range of 5µs to 20µs will fit.<br>Off time setting for constant $t_{OFF}$ chopper:<br>$N_{CLK}$= 12 + 32*$t_{OFF}$ (Minimum is 64 clocks)<br>Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel. | 0 / 2… 15 |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 168 | smartEnergy current minimum (SEIMIN) | Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value.<br>minimum motor current:<br>0 – 1/2 of CS<br>1 – 1/4 of CS | 0/1 |
| 169 | smartEnergy current down step | Sets the number of stallGuard2™ readings above the upper threshold necessary for each current decrement of the motor current.<br>Number of stallGuard2™ measurements per decrement:<br>Scaling: 0… 3: 32, 8, 2, 1<br>0: slow decrement<br>3: fast decrement | 0… 3 |
| 170 | smartEnergy hysteresis | Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased.<br>Hysteresis:<br>(smartEnergy hysteresis value + 1) * 32<br>Upper stallGuard threshold:<br>(smartEnergy hysteresis start + smartEnergy hysteresis + 1) * 32 | 0… 15 |
| 171 | smartEnergy current up step | Sets the current increment step. The current becomes incremented for each measured stallGuard2™ value below the lower threshold (see smartEnergy hysteresis start).<br>current increment step size:<br>Scaling: 0… 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load | 1… 3 |
| 172 | smartEnergy hysteresis start | The lower threshold for the stallGuard2™ value (see smart Energy current up step). | 0… 15 |
| 173 | stallGuard2™ filter enable | Enables the stallGuard2™ filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps.<br>*In most cases it is expedient to set the filtered mode before using coolStep™.*<br>*Use the standard mode for step loss detection.*<br>0 – standard mode<br>1 – filtered mode | 0/1 |
| 174 | stallGuard2™ threshold | This signed value controls stallGuard2™ *threshold* level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2™ less sensitive and requires more torque to indicate a stall.<br>0: Indifferent value<br>1… 63: less sensitivity<br>-1… -64: higher sensitivity | -64… 63 |

| **Number** | **Axis Parameter** | **Description** | **Range [Unit]** |
|---|---|---|---|
| 175 | slope control high side | Determines the slope of the motor driver outputs. *Set to 2 or 3 for this module or rather use the default value.*<br>0: lowest slope<br>3: fastest slope | 0… 3 |
| 176 | slope control low side | Determines the slope of the motor driver outputs. *Set identical to slope control high side.* | 0… 3 |
| 177 | short protection disable | 0: Short to GND protection is on<br>1: Short to GND protection is disabled<br>*Use default value!* | 0/1 |
| 178 | short detection timer | 0: 3.2µs<br>1: 1.6µs<br>2: 1.2µs<br>3: 0.8µs<br>*Use default value!* | 0..3 |
| 180 | smartEnergy actual current | This status value provides the *actual motor current* setting as controlled by coolStep™. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN.<br><u>actual motor current scaling factor:</u><br>0 … 31: 1/32, 2/32, … 32/32 | 0… 31 |
| 181 | stop on stall | Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2™ load value reaches zero. | 0… 2047<br>$\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. | 0… 2047<br>$\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$ |
| 183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed. | 0… 255<br>$\left[\frac{\text{max. module current}}{255}\right]$ |
| 193 | referencing mode | 1 – Only the left reference switch is searched.<br>2 – The right switch is searched and afterwards the left switch is searched.<br>3 – Three-switch-mode: the right switch is searched first and afterwards the reference switch will be searched. | 1/2/3 |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. | 0… 2047 |
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. | 0… 2047 |
| 196 | distance end switches | This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3). | 0… 8388307 |
| 204 | freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. | 0… 65535<br>0 = never<br>[msec] |
| 206 | actual load value | Readout of the actual load value with used for stall detection (stallGuard2™). | 0… 1023 |

| Number | Axis Parameter | Description | Range [Unit] |
|---|---|---|---|
| 208 | TMC262 driver error flags | Bit 0: stallGuard™ status (1: threshold reached)<br>Bit 1: Overtemperature (1: driver is shut down due to overtemperature)<br>Bit 2: Pre-warning overtemperature (1: Threshold is exceeded)<br>Bit 3: Short to ground A (1: Short condition detected, driver currently shut down)<br>Bit 4: Short to ground B (1: Short condition detected, driver currently shut down)<br>Bit 5: Open load A (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 6: Open load B (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 7: Stand still (1: No step impulse occurred on the step input during the last 2^20 clock cycles)<br>*Please refer to the TMC262 Datasheet for more information.* | 0/1 |
| 209 | encoder position | The value of an encoder register can be read out or written. | [encoder steps] |
| 210 | encoder prescaler | Prescaler for the encoder. | |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. | 0… 65535<br><br>[encoder steps] |
| 214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). | 1… 65535<br>[10msec] |
| 215 | absolute encoder value | Absolute value of the encoder. | 0… 255<br>[encoder steps] |

**Example:**

Get the actual position of motor

*Mnemonic:* GAP 0, 1

*Binary:*

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

*Reply:*

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

⇨ **status=no error, position=711**

### 5.7.7 STAP (store axis parameter)

An axis parameter previously set with a *Set Axis Parameter* command (SAP) will be stored permanent. Most parameters are automatically restored after power up (refer to axis parameter list in chapter 6).

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

**Related commands:** SAP, RSAP, GAP, AAP

**Mnemonic:** STAP <parameter number>, 0

**Binary representation:**

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

***¹motor number is always 0 as only one motor is involved**
***²the value operand *of this function has no effect. Instead, the currently used value (e.g. selected by SAP) is saved.***

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Parameter ranges:**

| Parameter number | Motor number | Value |
|---|---|---|
| s. chapter 6 | 0 | s. chapter 6 |

**List of parameters, which can be used for STAP:**

| Number | Axis Parameter | Description |
|---|---|---|
| 4 | maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet for calculation of physical units. |
| 5 | maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 428 datasheet for calculation of physical units. |
| 6 | absolute max. current (CS / Current Scale) | The most important motor setting, since too high values might cause motor damage! The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on). |
| 7 | standby current | The current limit two seconds after the motor has stopped. |
| 12 | right limit switch disable | If set, deactivates the stop function of the right switch |
| 13 | left limit switch disable | Deactivates the stop function of the left switch resp. reference switch if set. |
| 130 | minimum speed | Should always be set 1 to ensure exact reaching of the target position. Do not change! |

| Number | Axis Parameter | Description |
|---|---|---|
| 138 | ramp mode | Automatically set when using ROR, ROL, MST and MVP.<br>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.<br>2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.<br>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected. |
| 140 | microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps |
| 149 | soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. |
| 153 | ramp divisor | The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one). |
| 154 | pulse divisor | The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one). |
| 193 | referencing mode | 1 – Only the left reference switch is searched.<br>2 – The right switch is searched and afterwards the left switch is searched.<br>3 – Three-switch-mode: the right switch is searched first and afterwards the reference switch will be searched. |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. |
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. |
| 204 | freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. |
| 210 | encoder prescaler | Prescaler for the encoder. |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. |
| 214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). |

**Example:**

Store the maximum speed of motor

*Mnemonic:* STAP 4, 0

*Binary:*

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

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

### 5.7.8 RSAP (restore axis parameter)

For all configuration-related axis parameters non-volatile memory locations are provided. By default, most parameters are automatically restored after power up (refer to axis parameter list in chapter 6). A single parameter that has been changed before can be reset by this instruction also.

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

**Relate commands:** SAP, STAP, GAP, and AAP

**Mnemonic:** RSAP <parameter number>, 0

**Binary representation:**

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

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

**Reply structure in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**List of parameters, which can be used for RSAP:**

| Number | Axis Parameter | Description |
|---|---|---|
| 4 | maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet for calculation of physical units. |
| 5 | maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 428 datasheet for calculation of physical units. |
| 6 | absolute max. current (CS / Current Scale) | The most important motor setting, since too high values might cause motor damage! The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on). |
| 7 | standby current | The current limit two seconds after the motor has stopped. |
| 12 | right limit switch disable | If set, deactivates the stop function of the right switch |
| 13 | left limit switch disable | Deactivates the stop function of the left switch resp. reference switch if set. |
| 130 | minimum speed | Should always be set 1 to ensure exact reaching of the target position. Do not change! |

| Number | Axis Parameter | Description |
|---|---|---|
| 138 | ramp mode | Automatically set when using ROR, ROL, MST and MVP.<br>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.<br>2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.<br>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected. |
| 140 | microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps |
| 149 | soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. |
| 153 | ramp divisor | The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one). |
| 154 | pulse divisor | The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one). |
| 193 | referencing mode | 1 – Only the left reference switch is searched.<br>2 – The right switch is searched and afterwards the left switch is searched.<br>3 – Three-switch-mode: the right switch is searched first and afterwards the reference switch will be searched. |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. |
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. |
| 204 | freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. |
| 210 | encoder prescaler | Prescaler for the encoder. |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. |
| 214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). |

**Example:**

Restore the maximum current of motor

*Mnemonic:* RSAP 6, 0

*Binary:*

| **Byte Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $08 | $06 | $00 | $00 | $00 | $00 | $00 | $10 |

### 5.7.9 SGP (set global parameter)

With this command most of the module specific parameters not directly related to motion control can be specified and the TMCL™ user variables can be changed. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. Refer to chapter 7 for a complete parameter list.

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

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

**Related commands:** GGP, STGP, RSGP, AGP

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

**Binary representation:**

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Global parameters of bank 0, which can be used for SGP:**

| Number | Global parameter | Description | Range |
|---|---|---|---|
| 64 | EEPROM magic | Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration. | 0… 255 |
| 65 | RS232/RS485 baud rate | 0 – 9600 baud – *Default*<br>1 – 14400 baud<br>2 – 19200 baud<br>3 – 28800 baud<br>4 – 38400 baud<br>5 – 57600 baud<br>6 – 76800 baud – *Not supported by Windows!*<br>7 – 115200 baud<br>8 – 230400 baud<br>9 – 250000 baud – *Not supported by Windows!*<br>10 – 500000 baud – *Not supported by Windows!*<br>11 – 1000000 baud – *Not supported by Windows!* | 0… 11 |
| 66 | serial address | The module (target) address for RS-232/RS-485. | 0… 255 |
| 67 | ASCII mode | Configure the TMCL™ ASCII interface:<br>Bit 0: 0 – start up in binary (normal) mode<br>1 – start up in ASCII mode<br>Bits 4 and 5:<br>00 – Echo back each character<br>01 – Echo back complete command<br>10 – Do not send echo, only send command reply | |

| Number | Global parameter | Description | Range |
|---|---|---|---|
| 69 | CAN bit rate | 2 20kBit/s<br>3 50kBit/s<br>4 100kBit/s<br>5 125kBit/s<br>6 250kBit/s<br>7 500kBit/s<br>8 1000kBit/s *Default* | 2… 8 |
| 70 | CAN reply ID | The CAN ID for replies from the board (default: 2) | 0… 7ff |
| 71 | CAN ID | The module (target) address for CAN (default: 1) | 0… 7ff |
| 73 | configuration EEPROM lock flag | Write: 1234 to lock the EEPROM, 4321 to unlock it.<br>Read: 1=EEPROM locked, 0=EEPROM unlocked. | 0/1 |
| 75 | telegram pause time | Pause time before the reply via RS232 or RS485 is sent. For RS232 set to 0.<br>For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin).<br>For CAN interface this parameter has no effect! | 0… 255 |
| 76 | serial host address | Host address used in the reply telegrams sent back via RS232 or RS485. | 0… 255 |
| 77 | auto start mode | 0: Do not start TMCL™ application after power up (default).<br>1: Start TMCL™ application automatically after power up. | 0/1 |
| 80 | shutdown pin functionality | Select the functionality of the SHUTDOWN pin<br>0 – no function<br>1 – high active<br>2 – low active | 0… 2 |
| 81 | TMCL™ code protection | Protect a TMCL™ program against disassembling or overwriting.<br>0 – no protection<br>1 – protection against disassembling<br>2 – protection against overwriting<br>3 – protection against disassembling and overwriting<br>***If you switch off the protection against disassembling, the program will be erased first! Changing this value from 1 or 3 to 0 or 2, the TMCL™ program will be wiped off.*** | 0,1,2,3 |
| 83 | CAN secondary address | Second CAN ID for the module. Switched off when set to zero. | 0… 7ff |
| 84 | coordinate storage | 0 – coordinates are stored in the RAM only (but can be copied explicitly between RAM and EEPROM)<br>1 – coordinates are always stored in the EEPROM only | 0 or 1 |
| 132 | tick timer | A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value. | |

**Global parameters of bank 1, which can be used for SGP:**

The global parameter bank 1 is normally not available, but can be used for customer specific extensions of the firmware.

**Global parameters of bank 2, which can be used for SGP:**

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

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

**Global parameters of bank 3, which can be used for SGP:**

Bank 3 contains interrupt parameters. Some interrupts need configuration (e.g. the timer interval of a timer interrupt). This can be done using the SGP commands with parameter bank 3 (SGP <type>, 3, <value>). ***The priority of an interrupt depends on its number. Interrupts with a lower number have a higher priority.***

The following table shows all interrupt parameters that can be set.

| Number | Global parameter | Description | Range |
|---|---|---|---|
| 0 | Timer 0 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 1 | Timer 1 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 2 | Timer 2 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 39 | Input 0 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 |
| 40 | Input 1 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 |

**Example:**

Set the serial address of the target device to 3
*Mnemonic:* SGP 66, 0, 3

*Binary:*

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

### 5.7.10 GGP (get global parameter)

All global parameters can be read with this function. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. Please refer to chapter 7 for a complete parameter list.

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

**Related commands:** SGP, STGP, RSGP, AGP

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 10 | (see chapter 6) | <bank number> | (don't care) |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Global parameters of bank 0, which can be used for GGP:**

| Number | Global parameter | Description | Range |
|---|---|---|---|
| 64 | EEPROM magic | Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration. | 0… 255 |
| 65 | RS232/RS485 baud rate | 0 – 9600 baud (default)<br>1 – 14400 baud<br>2 – 19200 baud<br>3 – 28800 baud<br>4 – 38400 baud<br>5 – 57600 baud<br>6 – 76800 baud – *Not supported by Windows!*<br>7 – 115200 baud<br>8 – 230400 baud<br>9 – 250000 baud – *Not supported by Windows!*<br>10 – 500000 baud – *Not supported by Windows!*<br>11 – 1000000 baud – *Not supported by Windows!* | 0… 11 |
| 66 | serial address | The module (target) address for RS-232/RS-485. | 0… 255 |
| 67 | ASCII mode | Configure the TMCL™ ASCII interface:<br>Bit 0: 0 – start up in binary (normal) mode<br>1 – start up in ASCII mode<br>Bits 4 and 5:<br>00 – Echo back each character<br>01 – Echo back complete command<br>10 – Do not send echo, only send command reply | |

| Number | Global parameter | Description | Range |
|---|---|---|---|
| 69 | CAN bit rate | 2 20kBit/s<br>3 50kBit/s<br>4 100kBit/s<br>5 125kBit/s<br>6 250kBit/s<br>7 500kBit/s<br>8 1000kBit/s *Default* | 2… 8 |
| 70 | CAN reply ID | The CAN ID for replies from the board (default: 2) | 0… 7ff |
| 71 | CAN ID | The module (target) address for CAN (default: 1) | 0… 7ff |
| 73 | configuration EEPROM lock flag | Write: 1234 to lock the EEPROM, 4321 to unlock it.<br>Read: 1=EEPROM locked, 0=EEPROM unlocked. | 0/1 |
| 75 | telegram pause time | Pause time before the reply via RS232 or RS485 is sent. For RS232 set to 0.<br>For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin).<br>For CAN interface this parameter has no effect! | 0… 255 |
| 76 | serial host address | Host address used in the reply telegrams sent back via RS232 or RS485. | 0… 255 |
| 77 | auto start mode | 0: Do not start TMCL™ application after power up (default).<br>1: Start TMCL™ application automatically after power up. | 0/1 |
| 80 | shutdown pin functionality | Select the functionality of the SHUTDOWN pin<br>0 – no function<br>1 – high active<br>2 – low active | 0… 2 |
| 81 | TMCL™ code protection | Protect a TMCL™ program against disassembling or overwriting.<br>0 – no protection<br>1 – protection against disassembling<br>2 – protection against overwriting<br>3 – protection against disassembling and overwriting<br>***If you switch off the protection against disassembling, the program will be erased first! Changing this value from 1 or 3 to 0 or 2, the TMCL™ program will be wiped off.*** | 0,1,2,3 |
| 83 | CAN secondary address | Second CAN ID for the module. Switched off when set to zero. | 0… 7ff |
| 84 | coordinate storage | 0 – coordinates are stored in the RAM only (but can be copied explicitly between RAM and EEPROM)<br>1 – coordinates are always stored in the EEPROM only | 0 or 1 |
| 128 | TMCL™ application status | 0 –stop<br>1 – run<br>2 – step<br>3 – reset | 0… 3 |
| 129 | download mode | 0 – normal mode<br>1 – download mode | 0/1 |
| 130 | TMCL™ program counter | The index of the currently executed TMCL™ instruction. | |
| 132 | tick timer | A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value. | |
| 133 | random number | Choose a random number. ***Read only!*** | 0… 2147483 647 |

**Global parameters of bank 1, which can be used for GGP:**

The global parameter bank 1 is normally not available, but can be used for customer specific extensions of the firmware.

**Global parameters of bank 2, which can be used for GGP:**

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

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

**Global parameters of bank 3, which can be used for GGP:**

Bank 3 contains interrupt parameters. ***The priority of an interrupt depends on its number. Interrupts with a lower number have a higher priority.***

The following table shows all interrupt parameters that can be read.

| Number | Global parameter | Description | Range |
|---|---|---|---|
| 0 | Timer 0 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 1 | Timer 1 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 2 | Timer 2 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 39 | Input 0 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 |
| 40 | Input 1 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 |

**Example:**

Get the serial address of the target device
*Mnemonic:* GGP 66, 0

*Binary:*

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

*Reply:*

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

⇨ **Status=no error, Value=1**

### 5.7.11 STGP (store global parameter)

This command is used to store TMCL™ user variables permanently in the EEPROM of the module. Some global parameters are located in RAM memory, so without storing modifications are lost at power down. This instruction enables enduring storing. Most parameters are automatically restored after power up (see the list of global parameters in chapter 7).

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

**Related commands:** SGP, GGP, RSGP, AGP

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 11 | (see chapter 8) | <bank number> (see chapter 8) | (don't care) |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Global parameters of bank 0, which can be used for STGP:**

The global parameter bank 0 is not required for the STGP command, because these parameters are automatically stored with the SGP command in EEPROM.

**Global parameters of bank 1, which can be used for STGP:**

The global parameter bank 1 is normally not available, but can be used in customer specific extensions of the firmware.

**Global parameters of bank 2, which can be used for STGP:**

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

| Number | Global parameter | Description |
|---|---|---|
| 0 | general purpose variable #0 | for use in TMCL™ applications |
| 1 | general purpose variable #1 | for use in TMCL™ applications |
| 2 | general purpose variable #2 | for use in TMCL™ applications |
| 3 | general purpose variable #3 | for use in TMCL™ applications |
| 4 | general purpose variable #4 | for use in TMCL™ applications |
| 5 | general purpose variable #5 | for use in TMCL™ applications |
| 6 | general purpose variable #6 | for use in TMCL™ applications |
| 7 | general purpose variable #7 | for use in TMCL™ applications |
| 8 | general purpose variable #8 | for use in TMCL™ applications |
| 9 | general purpose variable #9 | for use in TMCL™ applications |
| 10 | general purpose variable #10 | for use in TMCL™ applications |
| 11 | general purpose variable #11 | for use in TMCL™ applications |
| 12 | general purpose variable #12 | for use in TMCL™ applications |
| 13 | general purpose variable #13 | for use in TMCL™ applications |
| 14 | general purpose variable #14 | for use in TMCL™ applications |
| 15 | general purpose variable #15 | for use in TMCL™ applications |
| 16 | general purpose variable #16 | for use in TMCL™ applications |
| 17 | general purpose variable #17 | for use in TMCL™ applications |
| 18 | general purpose variable #18 | for use in TMCL™ applications |

| Number | Global parameter | Description |
|---|---|---|
| 19 | general purpose variable #19 | for use in TMCL™ applications |
| 20… 55 | general purpose variables #20… #55 | for use in TMCL™ applications |

**Global parameters of bank 3, which can be used for STGP:**

The global parameter bank 0 is not required for the STGP command.

**Example:**

Store the user variable #42
*Mnemonic:* STGP 42, 2

*Binary:*

| **Byte Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $0b | $2a | $02 | $00 | $00 | $00 | $00 | $38 |

### 5.7.12 RSGP (restore global parameter)

With this command the contents of a TMCL™ user variable can be restored from the EEPROM. For all configuration-related axis parameters, non-volatile memory locations are provided. By default, most parameters are automatically restored after power up (see axis parameter list in chapter 7). A single parameter that has been changed before can be reset by this instruction.

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

**Relate commands:** SAP, STAP, GAP, and AAP

**Mnemonic:** RSAP <parameter number>, 0

**Binary representation:**

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

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

**Reply structure in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Global parameters of bank 0, which can be used for RSGP:**

The global parameter bank 0 is not required for the RSGP command, because these parameters are automatically stored with the SGP command in EEPROM.

**Global parameters of bank 1, which can be used for RSGP:**

The global parameter bank 1 is normally not available, but can be used in customer specific extensions of the firmware.

**Global parameters of bank 2, which can be used for RSGP:**

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

| Number | Global parameter | Description |
|---|---|---|
| 0 | general purpose variable #0 | for use in TMCL™ applications |
| 1 | general purpose variable #1 | for use in TMCL™ applications |
| 2 | general purpose variable #2 | for use in TMCL™ applications |
| 3 | general purpose variable #3 | for use in TMCL™ applications |
| 4 | general purpose variable #4 | for use in TMCL™ applications |
| 5 | general purpose variable #5 | for use in TMCL™ applications |
| 6 | general purpose variable #6 | for use in TMCL™ applications |
| 7 | general purpose variable #7 | for use in TMCL™ applications |
| 8 | general purpose variable #8 | for use in TMCL™ applications |
| 9 | general purpose variable #9 | for use in TMCL™ applications |
| 10 | general purpose variable #10 | for use in TMCL™ applications |
| 11 | general purpose variable #11 | for use in TMCL™ applications |
| 12 | general purpose variable #12 | for use in TMCL™ applications |
| 13 | general purpose variable #13 | for use in TMCL™ applications |
| 14 | general purpose variable #14 | for use in TMCL™ applications |
| 15 | general purpose variable #15 | for use in TMCL™ applications |
| 16 | general purpose variable #16 | for use in TMCL™ applications |
| 17 | general purpose variable #17 | for use in TMCL™ applications |

| Number | Global parameter | Description |
|---|---|---|
| 18 | general purpose variable #18 | for use in TMCL™ applications |
| 19 | general purpose variable #19 | for use in TMCL™ applications |
| 20… 55 | general purpose variables #20… c#55 | for use in TMCL™ applications |

**Global parameters of bank 3, which can be used for RSGP:**

The global parameter bank 3 is not required for the RSGP command.

**Example:**

Restore the maximum current of motor

*Mnemonic:* RSGP 6, 0

*Binary:*

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

### 5.7.13 RFS (reference search)

The TMCM-1180 has a built-in reference search algorithm which can be used. The reference search algorithm provides switching point calibration and three switch modes. The status of the reference search can also be queried to see if it has already finished. (In a TMCL™ program it is better to use the WAIT command to wait for the end of a reference search.) Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (chapter 6). The reference search can be started, stopped, and the actual status of the reference search can be checked.

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

**Related commands:** WAIT

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 13 | 0 START – start ref. search<br>1 STOP – abort ref. search<br>2 STATUS – get status | 0* | (don't care) |

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

**Reply in direct mode:**

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

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

When using type 2 (STATUS):

| STATUS | VALUE |
|---|---|
| 100 – OK | 0 – no ref. search active<br>other values – ref. search is active |

**Example:**

Start reference search of motor
*Mnemonic:* RFS START, 0

*Binary:*

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

***With this PANdrive it is possible to use stall detection instead of a reference search. Please see section 8.1 for details.***

## 5.7.14 SIO (set output)

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

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

**Related commands:** GIO, WAIT

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

**Binary representation:**

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

**Reply structure:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Set OUT_7 to high (bank 2, output 7; general purpose output)
*Mnemonic:* SIO 7, 2, 1

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Function | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| Value (hex) | $01 | $0e | $07 | $02 | $00 | $00 | $00 | $01 | $19 |

**Available I/O ports of TMCM-1180:**

|  | Pin | I/O port | Command | Range |
|---|---|---|---|---|
| | 3 | OUT_0 | SIO 0, 2, <n>, (n=0/1) | 1/0 |
|  | 4 | OUT_1 | SIO 1, 2, <n>, (n=0/1) | 1/0 |

**Addressing both output lines with one SIO command:**

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

**Example:**

Set both output pins high.
*Mnemonic:* SIO 255, 2, 3

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

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

## 5.7.15 GIO (get input/output)

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

**Internal function:** The specified line is read.

**Related commands:** SIO, WAIT

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

**Binary representation:**

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | <status of the port> |

**Example:**

Get the analogue value of ADC channel 0
*Mnemonic:* GIO 0, 1

*Binary:*

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

*Reply:*

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

⇨ value: 506

### 5.7.15.1 I/O bank 0 – digital inputs:

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

| | Pin | I/O port | Command | Range |
|---|---|---|---|---|
| | 1 | IN_0 | GIO 0, 0 | 0/1 |
| | 2 | IN_1 | GIO 1, 0 | 0/1 |

<u>**Reading all digital inputs with one GIO command:**</u>

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

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

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

### 5.7.15.2 I/O bank 1 – analogue inputs:

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

| | Pin | I/O port | Command | Range |
|---|---|---|---|---|
| | **1** | IN_0 | GIO 0, 1 | 0… 1023 |
| | **2** | IN_1 | GIO 1, 1 | 0… 1023 |

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

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

| | Pin | I/O port | Command | Range |
|---|---|---|---|---|
| | **3** | OUT_0 | GIO 0, 2, <n> | 1/0 |
| | **4** | OUT_1 | GIO 1, 2, <n> | 1/0 |

### 5.7.16 CALC (calculate)

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

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

**Mnemonic:** CALC <operation>, <value>
where <op> is ADD, SUB, MUL, DIV, MOD, AND, OR, XOR, NOT or LOAD

**Binary representation:**

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

**Example:**

Multiply accu by -5000
*Mnemonic:* CALC MUL, -5000

*Binary:*

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

### 5.7.17 COMP (compare)

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

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

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

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

**Mnemonic:** COMP <value>

**Binary representation:**

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

**Example:**

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

```
GAP 1, 2, 0      //get axis parameter, type: no. 1 (actual position), motor: 0, value: 0 (don't care)
COMP 1000        //compare actual value to 1000
JC GE, Label     //jump, type: 5 greater/equal, the label must be defined somewhere else in the
                 program
```

*Binary format of the COMP 1000 command:*

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

### 5.7.18 JC (jump conditional)

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

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

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

**Related commands:** JA, COMP, WAIT, CLE

**Mnemonic:** JC <condition>, <label>
where <condition>=ZE|NZ|EQ|NE|GT|GE|LT|LE|ETO|EAL|EDV|EPO

**Binary representation:**

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

**Example:**

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

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

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

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

### 5.7.19 JA (jump always)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. This command cannot be used in direct mode.***

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

**Related commands:** JC, WAIT, CSUB

**Mnemonic:** JA <Label>

**Binary representation:**

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

**Example:** An infinite loop in TMCL™

```
Loop:   MVP ABS, 0, 10000
        WAIT POS, 0, 0
        MVP ABS, 0, 0
        WAIT POS, 0, 0
        JA Loop         //Jump to the label Loop
```

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

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

### 5.7.20 CSUB (call subroutine)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. This command cannot be used in direct mode.***

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

**Related commands:** RSUB, JA

**Mnemonic:** CSUB <Label>

**Binary representation:**

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

**Example: Call a subroutine**

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

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

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

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

### 5.7.21 RSUB (return from subroutine)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. This command cannot be used in direct mode.***

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

**Related command:** CSUB

**Mnemonic:** RSUB

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 24 | (don't care) | (don't care) | (don't care) |

**Example:** please see the CSUB example (section 5.7.20).

*Binary format of RSUB:*

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

### 5.7.22 WAIT (wait for an event to occur)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. This command cannot be used in direct mode.***

**There are five different wait conditions that can be used:**

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

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

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

**Related commands:** JC, CLE

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 27 | 0 TICKS - timer ticks*[1] | (don't care) | <no. of ticks*> |
| | 1 POS - target position reached | 0*[2]<br>0… 2 resp. 0… 5 | <no. of ticks* for timeout>,<br>0 for no timeout |
| | 2 REFSW – reference switch | 0<br>0… 2 resp. 0… 5 | <no. of ticks* for timeout>,<br>0 for no timeout |
| | 3 LIMSW – limit switch | 0<br>0… 2 resp. 0… 5 | <no. of ticks* for timeout>,<br>0 for no timeout |
| | 4 RFS – reference search completed | 0<br>0… 2 resp. 0… 5 | <no. of ticks* for timeout>,<br>0 for no timeout |

***[1] one tick is 10 milliseconds (in standard firmware)**
***[2] motor number is always 0 as only one motor is involved**

**Example:**

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

*Binary*:

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

### 5.7.23 STOP (stop TMCL™ program execution)

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

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

**Internal function:** TMCL™ instruction fetching is stopped.

**Related commands:** none
**Mnemonic:** STOP

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 28 | (don't care) | (don't care) | (don't care) |

**Example:**
*Mnemonic:* STOP

*Binary:*

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

### 5.7.24 SCO (set coordinate)

Up to 20 position values (coordinates) can be stored for every axis for use with the MVP COORD command. This command sets a coordinate to a specified value. 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 note that the coordinate number 0 is always stored in RAM only.***

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

**Related commands:** GCO, CCO, MVP

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

**Binary representation:**

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

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

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

*Binary:*

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

With TMCL™ version 4.18 and higher, two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate to the EEPROM.

<u>These special functions can be accessed using the following special forms of the SCO command:</u>

| | |
|---|---|
| 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. |

## 5.7.25 GCO (get coordinate)

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

**Mnemonic:** GCO <coordinate number>, 0

**Binary representation:**

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

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Get motor value of coordinate 1
*Mnemonic:* GCO 1, 0

*Binary:*

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

*Reply:*

| **Byte Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Target-address | Status | Instruction | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $02 | $01 | $64 | $0a | $00 | $00 | $00 | $00 | $86 |

⇨ **Value: 0**

With TMCL™ version 4.18 and higher, 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.

<u>These special functions can be accessed using the following special forms of the GCO command:</u>

| | |
|---|---|
| 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. |

### 5.7.26 CCO (capture coordinate)

The actual position of the axis is copied 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 that the coordinate number 0 is always stored in RAM only.***

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

**Related commands:** SCO, GCO, MVP

**Mnemonic:** CCO <coordinate number>, 0

**Binary representation:**

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

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Store current position of the axe to coordinate 3
*Mnemonic:* CCO 3, 0

*Binary:*

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

### 5.7.27 ACO (accu to coordinate; valid from TMCL™ version 4.18 on)

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).

***Please note, that this command is valid from TMCL™ version 4.18 and TMCL-IDE version 1.77 on.***

***Please note also that the coordinate number 0 is always stored in RAM only. For Information about storing coordinates refer to the SCO command.***

**Internal function:** The actual value of the accumulator is stored in the internal position array.

**Related commands:** GCO, CCO, MVP COORD, SCO

**Mnemonic:** ACO <coordinate number>, 0

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 39 | <coordinate number> (0… 20) | 0* | (don't care) |

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Example:**

Copy the actual value of the accumulator to coordinate 1 of motor
*Mnemonic:* ACO 1, 0

*Binary:*

| **Byte Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $27 | $01 | $00 | $00 | $00 | $00 | $00 | $29 |

### 5.7.28 CALCX (calculate using the X register)

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

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

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

**Binary representation:**

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

**Example:**

Multiply accu by X-register
*Mnemonic:* CALCX MUL

*Binary:*

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

### 5.7.29 AAP (accumulator to axis parameter)

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

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

**Mnemonic:** AAP <parameter number>, 0

**Binary representation:**

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

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**List of parameters, which can be used for AAP:**

| Number | Axis Parameter | Description |
|---|---|---|
| 0 | target (next) position | The desired position in position mode (see ramp mode, no. 138). |
| 1 | actual position | The current position of the motor. Should only be overwritten for reference point setting. |
| 2 | target (next) speed | The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest. |
| 3 | actual speed | The current rotation speed. |
| 4 | maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet for calculation of physical units. |
| 5 | maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 428 datasheet for calculation of physical units. |
| 6 | absolute max. current (CS / Current Scale) | The most important motor setting, since too high values might cause motor damage! The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on). |
| 7 | standby current | The current limit two seconds after the motor has stopped. |
| 12 | right limit switch disable | If set, deactivates the stop function of the right switch |

| Number | Axis Parameter | Description |
|---|---|---|
| 13 | left limit switch disable | Deactivates the stop function of the left switch resp. reference switch if set. |
| 130 | minimum speed | Should always be set 1 to ensure exact reaching of the target position. Do not change! |
| 138 | ramp mode | Automatically set when using ROR, ROL, MST and MVP.<br>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.<br>2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.<br>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected. |
| 140 | microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps |
| 141 | ref. switch tolerance | For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop. |
| 149 | soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. |
| 153 | ramp divisor | The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one). |
| 154 | pulse divisor | The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one). |
| 160 | step interpolation enable | Step interpolation is supported with a 16 microstep setting only. In this setting, each step impulse at the input causes the execution of 16 times 1/256 microsteps. This way, a smooth motor movement like in 256 microstep resolution is achieved.<br>0 – step interpolation off<br>1 – step interpolation on |
| 161 | double step enable | Every edge of the cycle releases a step/microstep. *It does not make sense to activate this parameter for internal use.*<br>Double step enable can be used with Step/Dir interface.<br>0 – double step off<br>1 – double step on |

| Number | Axis Parameter | Description |
|---|---|---|
| 162 | chopper blank time | Selects the comparator *blank time*. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. For low current drivers, a setting of 1 or 2 is good. For higher current applications like the TMCM-1180 a setting of 2 or 3 will be required. |
| 163 | chopper mode | Selection of the chopper mode:<br>0 – spread cycle<br>1 – classic const. off time |
| 164 | chopper hysteresis decrement | Hysteresis decrement setting. This setting determines the slope of the hysteresis during on time and during fast decay time.<br>0 – fast decrement<br>3 – very slow decrement |
| 165 | chopper hysteresis end | Hysteresis end setting. Sets the hysteresis end value after a number of decrements. Decrement interval time is controlled by axis parameter 164.<br>-3… -1: negative hysteresis end setting<br>0: zero hysteresis end setting<br>1… 12: positive hysteresis end setting |
| 166 | chopper hysteresis start | Hysteresis start setting. Please remark, that this value is an offset to the hysteresis end value. |
| 167 | chopper off time | The off time setting controls the minimum chopper frequency. An off time within the range of 5µs to 20µs will fit.<br>Off time setting for constant $t_{OFF}$ chopper:<br>$N_{CLK}$= 12 + 32*$t_{OFF}$ (Minimum is 64 clocks)<br>Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel. |
| 168 | smartEnergy current minimum (SEIMIN) | Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value.<br>minimum motor current:<br>0 – 1/2 of CS<br>1 – 1/4 of CS |
| 169 | smartEnergy current down step | Sets the number of stallGuard2™ readings above the upper threshold necessary for each current decrement of the motor current.<br>Number of stallGuard2™ measurements per decrement:<br>Scaling: 0… 3: 32, 8, 2, 1<br>0: slow decrement<br>3: fast decrement |
| 170 | smartEnergy hysteresis | Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased.<br>Hysteresis:<br>(smartEnergy hysteresis value + 1) * 32 |

| Number | Axis Parameter | Description |
|---|---|---|
| | | Upper stallGuard threshold:<br>(smartEnergy hysteresis start + smartEnergy hysteresis + 1) * 32 |
| 171 | smartEnergy current up step | Sets the current increment step. The current becomes incremented for each measured stallGuard2™ value below the lower threshold (see smartEnergy hysteresis start).<br>current increment step size:<br>Scaling: 0… 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load |
| 172 | smartEnergy hysteresis start | The lower threshold for the stallGuard2™ value (see smart Energy current up step). |
| 173 | stallGuard2™ filter enable | Enables the stallGuard2™ filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps.<br>*In most cases it is expedient to set the filtered mode before using coolStep™.*<br>*Use the standard mode for step loss detection.*<br>0 – standard mode<br>1 – filtered mode |
| 174 | stallGuard2™ threshold | This signed value controls stallGuard2™ *threshold* level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2™ less sensitive and requires more torque to indicate a stall.<br>0 — Indifferent value<br>1… 63 — less sensitivity<br>-1… -64 — higher sensitivity |
| 175 | slope control high side | Determines the slope of the motor driver outputs. *Set to 2 or 3 for this module or rather use the default value.*<br>0: lowest slope<br>3: fastest slope |
| 176 | slope control low side | Determines the slope of the motor driver outputs. *Set identical to slope control high side.* |
| 177 | short protection disable | 0: Short to GND protection is on<br>1: Short to GND protection is disabled<br>*Use default value!* |
| 178 | short detection timer | 0: 3.2µs<br>1: 1.6µs<br>2: 1.2µs<br>3: 0.8µs<br>*Use default value!* |
| 181 | stop on stall | Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2™ load value reaches zero. |
| 182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. |

| Number | Axis Parameter | Description |
|---|---|---|
| 183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed. |
| 193 | referencing mode | 1 – Only the left reference switch is searched.<br>2 – The right switch is searched and afterwards the left switch is searched.<br>3 – Three-switch-mode: the right switch is searched first and afterwards the reference switch will be searched. |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. |
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. |
| 204 | freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. |
| 209 | encoder position | The value of an encoder register can be read out or written. |
| 210 | encoder prescaler | Prescaler for the encoder. |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. |
| 214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). |

**Example:**

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

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

*Binary format of the AAP 0,0 command:*

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

### 5.7.30 AGP (accumulator to global parameter)

The content of the accumulator register is transferred to the specified global parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction. ***Note that the global parameters in bank 0 are EEPROM-only and thus should not be modified automatically by a standalone application.*** (See chapter 7 for a complete list of global parameters).

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

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

**Binary representation:**

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | (don't care) |

**Global parameters of bank 0, which can be used for AGP:**

| Number | Global parameter | Description | | |
|---|---|---|---|---|
| 64 | EEPROM magic | Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration. | | |
| 65 | RS232/RS485 baud rate | 0 | 9600 baud | *Default* |
| | | 1 | 14400 baud | |
| | | 2 | 19200 baud | |
| | | 3 | 28800 baud | |
| | | 4 | 38400 baud | |
| | | 5 | 57600 baud | |
| | | 6 | 76800 baud | *Not supported by Windows!* |
| | | 7 | 115200 baud | |
| | | 8 | 230400 baud | |
| | | 9 | 250000 baud | *Not supported by Windows!* |
| | | 10 | 500000 baud | *Not supported by Windows!* |
| | | 11 | 1000000 baud | *Not supported by Windows!* |
| 66 | serial address | The module (target) address for RS-232/RS-485. | | |
| 67 | ASCII mode | Configure the TMCL™ ASCII interface:<br>Bit 0: 0 – start up in binary (normal) mode<br>1 – start up in ASCII mode<br>Bits 4 and 5:<br>00 – Echo back each character<br>01 – Echo back complete command<br>10 – Do not send echo, only send command reply | | |
| 69 | CAN bit rate | 2 | 20kBit/s | |
| | | 3 | 50kBit/s | |
| | | 4 | 100kBit/s | |
| | | 5 | 125kBit/s | |
| | | 6 | 250kBit/s | |
| | | 7 | 500kBit/s | |
| | | 8 | 1000kBit/s | *Default* |
| 70 | CAN reply ID | The CAN ID for replies from the board (default: 2) | | |

| Number | Global parameter | Description |
|---|---|---|
| 71 | CAN ID | The module (target) address for CAN (default: 1) |
| 73 | configuration EEPROM lock flag | Write: 1234 to lock the EEPROM, 4321 to unlock it.<br>Read: 1=EEPROM locked, 0=EEPROM unlocked. |
| 75 | telegram pause time | Pause time before the reply via RS232 or RS485 is sent. For RS232 set to 0.<br>For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin).<br>For CAN interface this parameter has no effect! |
| 76 | serial host address | Host address used in the reply telegrams sent back via RS232 or RS485. |
| 77 | auto start mode | 0: Do not start TMCL™ application after power up (default).<br>1: Start TMCL™ application automatically after power up. |
| 80 | shutdown pin functionality | Select the functionality of the SHUTDOWN pin<br>0 – no function<br>1 – high active<br>2 – low active |
| 81 | TMCL™ code protection | Protect a TMCL™ program against disassembling or overwriting.<br>0 – no protection<br>1 – protection against disassembling<br>2 – protection against overwriting<br>3 – protection against disassembling and overwriting<br>***If you switch off the protection against disassembling, the program will be erased first! Changing this value from 1 or 3 to 0 or 2, the TMCL™ program will be wiped off.*** |
| 83 | CAN secondary address | Second CAN ID for the module. Switched off when set to zero. |
| 84 | coordinate storage | 0 – coordinates are stored in the RAM only (but can be copied explicitly between RAM and EEPROM)<br>1 – coordinates are always stored in the EEPROM only |
| 132 | tick timer | A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value. |

**Global parameters of bank 1, which can be used for AGP:**

The global parameter bank 1 is normally not available, but can be used in customer specific extensions of the firmware.

**Global parameters of bank 2, which can be used for AGP:**

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

| Number | Global parameter | Description |
|---|---|---|
| 0 | general purpose variable #0 | for use in TMCL™ applications |
| 1 | general purpose variable #1 | for use in TMCL™ applications |
| 2 | general purpose variable #2 | for use in TMCL™ applications |
| 3 | general purpose variable #3 | for use in TMCL™ applications |
| 4 | general purpose variable #4 | for use in TMCL™ applications |
| 5 | general purpose variable #5 | for use in TMCL™ applications |
| 6 | general purpose variable #6 | for use in TMCL™ applications |
| 7 | general purpose variable #7 | for use in TMCL™ applications |
| 8 | general purpose variable #8 | for use in TMCL™ applications |
| 9 | general purpose variable #9 | for use in TMCL™ applications |

| Number | Global parameter | Description |
|---|---|---|
| 10 | general purpose variable #10 | for use in TMCL™ applications |
| 11 | general purpose variable #11 | for use in TMCL™ applications |
| 12 | general purpose variable #12 | for use in TMCL™ applications |
| 13 | general purpose variable #13 | for use in TMCL™ applications |
| 14 | general purpose variable #14 | for use in TMCL™ applications |
| 15 | general purpose variable #15 | for use in TMCL™ applications |
| 16 | general purpose variable #16 | for use in TMCL™ applications |
| 17 | general purpose variable #17 | for use in TMCL™ applications |
| 18 | general purpose variable #18 | for use in TMCL™ applications |
| 19 | general purpose variable #19 | for use in TMCL™ applications |
| 20..55 | general purpose variables #20… #55 | for use in TMCL™ applications |

**Global parameters of bank 3, which can be used for AGP:**

Bank 3 contains interrupt parameters. Some interrupts need configuration (e.g. the timer interval of a timer interrupt). ***The priority of an interrupt depends on its number. Interrupts with a lower number have a higher priority.***

| Number | Global parameter | Description | Range |
|---|---|---|---|
| 0 | Timer 0 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 1 | Timer 1 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 2 | Timer 2 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] |
| 39 | Input 0 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 |
| 40 | Input 1 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 |

**Example:**

Copy accumulator to TMCL™ user variable #3

*Mnemonic:* AGP 3, 2

*Binary:*

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

## 5.7.31 CLE (clear error flags)

This command clears the internal error flags. ***It is intended for use in standalone mode only and must not be used in direct mode.***

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

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

**Related commands:** JC

**Mnemonic:** CLE <flags>
where <flags>=ALL|ETO|EDV|EPO

**Binary representation:**

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

**Example:**

Reset the timeout flag
*Mnemonic:* CLE ETO

*Binary:*

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

### 5.7.32 VECT (set interrupt vector)

The VECT command defines an interrupt vector. It needs an interrupt number and a label as parameter (like in JA, JC and CSUB commands).

***This label must be the entry point of the interrupt handling routine.***

**Related commands:** EI, DI, RETI

**Mnemonic:** VECT <interrupt number>, <label>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 37 | <interrupt number> | (don't care) | <label> |

The following table shows all interrupt vectors that can be used:

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | Target position reached |
| 15 | stallGuard™ |
| 21 | Deviation |
| 27 | Left stop switch |
| 28 | Right stop switch |
| 39 | Input change 0 |
| 40 | Input change 1 |

**Example:** Define interrupt vector at target position 500
VECT 3, 500

*Binary format of VECT:*

| **Byte Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $25 | $03 | $00 | $00 | $00 | $01 | $F4 | $1E |

### 5.7.33 EI (enable interrupt)

The EI command enables an interrupt. It needs the interrupt number as parameter. Interrupt number 255 globally enables interrupts.

**Related command:** DI, VECT, RETI

**Mnemonic:** EI <interrupt number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 25 | <interrupt number> | (don't care) | (don't care) |

The following table shows all interrupt vectors that can be used:

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | Target position reached |
| 15 | stallGuard™ |
| 21 | Deviation |
| 27 | Left stop switch |
| 28 | Right stop switch |
| 39 | Input change 0 |
| 40 | Input change 1 |
| 255 | Global interrupts |

**Examples:**

Enable interrupts globally
EI, 255

*Binary format of EI:*

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

Enable interrupt when target position reached
EI, 3

*Binary format of EI:*

| **Byte Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $19 | $03 | $00 | $00 | $00 | $00 | $00 | $1D |

### 5.7.34 DI (disable interrupt)

The DI command disables an interrupt. It needs the interrupt number as parameter. Interrupt number 255 globally disables interrupts.

**Related command:** EI, VECT, RETI

**Mnemonic:** DI <interrupt number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 26 | <interrupt number> | (don't care) | (don't care) |

The following table shows all interrupt vectors that can be used:

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | Target position reached |
| 15 | stallGuard™ |
| 21 | Deviation |
| 27 | Left stop switch |
| 28 | Right stop switch |
| 39 | Input change 0 |
| 40 | Input change 1 |
| 255 | Global interrupts |

**Examples:**

Disable interrupts globally
DI, 255

*Binary format of DI:*

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

Disable interrupt when target position reached
DI, 3

*Binary format of DI:*

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

### 5.7.35 RETI (return from interrupt)

This command terminates the interrupt handling routine, and the normal program execution continues.
***At the end of an interrupt handling routine the RETI command must be executed.***

**Internal function:** The saved registers (A register, X register, flags) are copied back. Normal program execution continues.

**Related commands:** EI, DI, VECT

**Mnemonic:** RETI

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 38 | (don't care) | (don't care) | (don't care) |

**Example:** Terminate interrupt handling and continue with normal program execution
RETI

*Binary format of RETI:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $26 | $00 | $00 | $00 | $00 | $01 | $00 | $27 |

### 5.7.36 Customer specific TMCL™ command extension (UF0… UF7/user function)

The user definable functions UF0…UF7 are predefined, functions without topic for user specific purposes. Contact TRINAMIC for the customer specific programming of these functions.

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

**Related commands:** none

**Mnemonic:** UF0… UF7

**Binary representation:**

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

**Reply in direct mode:**

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

### 5.7.37 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 stand alone mode, it is covered by the WAIT command) and hence does not have a mnemonic.***

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

**Mnemonic: ---**

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 138 | (don't care) | (don't care) | 0* |

*** Motor number**

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

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

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

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

### 5.7.38 BIN (return to binary mode)

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

**Related Commands:** none

**Mnemonic:** BIN

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

### 5.7.39 TMCL™ Control Functions

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

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

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

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

| Instruction | Description | Type | Mot/Bank | Value |
|---|---|---|---|---|
| 128 – stop application | a running TMCL™ standalone application is stopped | (don't care) | (don't care) | (don't care) |
| 129 – run application | TMCL™ execution is started (or continued) | 0 - run from current address<br>1 - run from specified address | (don't care) | (don't care)<br><br>starting address |
| 130 – step application | only the next command of a TMCL™ application is executed | (don't care) | (don't care) | (don't care) |
| 131 – reset application | the program counter is set to zero, and the standalone application is stopped (when running or stepped) | (don't care) | (don't care) | (don't care) |
| 132 – start download mode | target command execution is stopped and all following commands are transferred to the TMCL™ memory | (don't care) | (don't care) | starting address of the application |
| 133 – quit download mode | target command execution is resumed | (don't care) | (don't care) | (don't care) |
| 134 – read TMCL™ memory | the specified program memory location is read | (don't care) | (don't care) | <memory address> |
| 135 – get application status | one of these values is returned:<br>0 – stop<br>1 – run<br>2 – step<br>3 – reset | (don't care) | (don't care) | (don't care) |
| 136 – get firmware version | return the module type and firmware revision either as a string or in binary format | 0 – string<br>1 – binary | (don't care) | (don't care) |
| 137 – restore factory settings | reset all settings stored in the EEPROM to their factory defaults<br>This command does not send back a reply. | (don't care) | (don't care) | must be 1234 |
| 138 – reserved | | | | |
| 139 – enter ASCII mode | Enter ASCII command line (see chapter 5.6) | (don't care) | (don't care) | (don't care) |

**Special reply format of command 136:**

**Type set to 0 - reply as a string:**

| Byte index | Contents |
|---|---|
| 1 | Host Address |
| 2… 9 | Version string (8 characters, e.g. 140V2.50 |

- There is no checksum in this reply format!
- To get also the last byte when using the CAN bus interface, just send this command in an eight byte frame instead of a seven byte frame. Then, eight bytes will be sent back, so you will get all characters of the version string.

**Type set to 1 - version number in binary format:**

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

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

# 6 Axis parameters

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

**Meaning of the letters in column *Access*:**

R = readable (GAP)
W = writable (SAP)
E = automatically restored from EEPROM after reset or power-on

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 0 | target (next) position | The desired position in position mode (see ramp mode, no. 138). | $\pm 2^{23}$ [µsteps] | RW |
| 1 | actual position | The current position of the motor. Should only be overwritten for reference point setting. | $\pm 2^{23}$ [µsteps] | RW |
| 2 | target (next) speed | The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 3 | actual speed | The current rotation speed. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 4 | maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet for calculation of physical units. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RWE |
| 5 | maximum acceleration | The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 428 datasheet for calculation of physical units. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RWE |
| 6 | absolute max. current (CS / Current Scale) | The most important motor setting, since too high values might cause motor damage! The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on). | 0… 255 $\left[\frac{\text{max. module current}}{255}\right]$ | RWE |
| 7 | standby current | The current limit two seconds after the motor has stopped. | 0… 255 $\left[\frac{\text{max. module current}}{255}\right]$ | RWE |
| 8 | target pos. reached | Indicates that the actual position equals the target position. | 0/1 | R |
| 9 | ref. switch status | The logical state of the reference (left) switch. See the TMC 428 data sheet for the different switch modes. The default has two switch modes: the left switch as the reference switch, the right switch as a limit (stop) switch. | 0/1 | R |
| 10 | right limit switch status | The logical state of the (right) limit switch. | 0/1 | R |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 11 | left limit switch status | The logical state of the left limit switch (in three switch mode) | 0/1 | R |
| 12 | right limit switch disable | If set, deactivates the stop function of the right switch | 0/1 | RWE |
| 13 | left limit switch disable | Deactivates the stop function of the left switch resp. reference switch if set. | 0/1 | RWE |
| 130 | minimum speed | Should always be set 1 to ensure exact reaching of the target position. Do not change! | 0… 2047 $\left[\frac{16\mathrm{MHz}}{65536}\cdot 2^{\mathrm{PD}}\frac{\mu\mathrm{steps}}{\mathrm{sec}}\right]$ | RWE |
| 135 | actual acceleration | The current acceleration (read only). | 0… 2047* | R |
| 138 | ramp mode | Automatically set when using ROR, ROL, MST and MVP.<br>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.<br>2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.<br>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected. | 0/1/2 | RWE |
| 140 | microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps | 0… 8 | RWE |
| 141 | ref. switch tolerance | For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop. | 0… 4095 [µsteps] | RW |
| 149 | soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. | 0/1 | RWE |
| 153 | ramp divisor | The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one). | 0… 13 | RWE |
| 154 | pulse divisor | The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one). | 0… 13 | RWE |
| 160 | step interpolation enable | Step interpolation is supported with a 16 microstep setting only. In this setting, each step impulse at the input causes the execution of 16 times 1/256 microsteps. This way, a smooth motor movement like in 256 microstep resolution is achieved.<br>0 – step interpolation off<br>1 – step interpolation on | 0/1 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 161 | double step enable | Every edge of the cycle releases a step/microstep. *It does not make sense to activate this parameter for internal use.*<br>Double step enable can be used with Step/Dir interface.<br>0 – double step off<br>1 – double step on | 0/1 | RW |
| 162 | chopper blank time | Selects the comparator *blank time*. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. For low current drivers, a setting of 1 or 2 is good. For higher current applications like the TMCM-1180 a setting of 2 or 3 will be required. | 0… 3 | RW |
| 163 | chopper mode | Selection of the chopper mode:<br>0 – spread cycle<br>1 – classic const. off time | 0/1 | RW |
| 164 | chopper hysteresis decrement | Hysteresis decrement setting. This setting determines the slope of the hysteresis during on time and during fast decay time.<br>0 – fast decrement<br>3 – very slow decrement | 0… 3 | RW |
| 165 | chopper hysteresis end | Hysteresis end setting. Sets the hysteresis end value after a number of decrements. Decrement interval time is controlled by axis parameter 164.<br>-3… -1 &#124; negative hysteresis end setting<br>0 &#124; zero hysteresis end setting<br>1… 12 &#124; positive hysteresis end setting | -3… 12 | RW |
| 166 | chopper hysteresis start | Hysteresis start setting. Please remark, that this value is an offset to the hysteresis end value. | 0… 8 | RW |
| 167 | chopper off time | The off time setting controls the minimum chopper frequency. An off time within the range of 5µs to 20µs will fit.<br>Off time setting for constant $t_{OFF}$ chopper:<br>$N_{CLK}$= 12 + 32*$t_{OFF}$ (Minimum is 64 clocks)<br>Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel. | 0 / 2… 15 | RW |
| 168 | smartEnergy current minimum (SEIMIN) | Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value.<br>minimum motor current:<br>0 – 1/2 of CS<br>1 – 1/4 of CS | 0/1 | RW |
| 169 | smartEnergy current down step | Sets the number of stallGuard2™ readings above the upper threshold necessary for each current decrement of the motor current.<br>Number of stallGuard2™ measurements per decrement:<br>Scaling: 0… 3: 32, 8, 2, 1<br>0: slow decrement<br>3: fast decrement | 0… 3 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 170 | smartEnergy hysteresis | Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased.<br>Hysteresis:<br>(smartEnergy hysteresis value + 1) * 32<br>Upper stallGuard threshold:<br>(smartEnergy hysteresis start + smartEnergy hysteresis + 1) * 32 | 0… 15 | RW |
| 171 | smartEnergy current up step | Sets the current increment step. The current becomes incremented for each measured stallGuard2™ value below the lower threshold (see smartEnergy hysteresis start).<br>current increment step size:<br>Scaling: 0… 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load | 1… 3 | RW |
| 172 | smartEnergy hysteresis start | The lower threshold for the stallGuard2™ value (see smart Energy current up step). | 0… 15 | RW |
| 173 | stallGuard2™ filter enable | Enables the stallGuard2™ filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps.<br>*In most cases it is expedient to set the filtered mode before using coolStep™.*<br>*Use the standard mode for step loss detection.*<br>0 – standard mode<br>1 – filtered mode | 0/1 | RW |
| 174 | stallGuard2™ threshold | This signed value controls stallGuard2™ *threshold* level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2™ less sensitive and requires more torque to indicate a stall.<br>0: Indifferent value<br>1… 63: less sensitivity<br>-1… -64: higher sensitivity | -64… 63 | RW |
| 175 | slope control high side | Determines the slope of the motor driver outputs. *Set to 2 or 3 for this module or rather use the default value.*<br>0: lowest slope<br>3: fastest slope | 0… 3 | RW |
| 176 | slope control low side | Determines the slope of the motor driver outputs. *Set identical to slope control high side.* | 0… 3 | RW |
| 177 | short protection disable | 0: Short to GND protection is on<br>1: Short to GND protection is disabled<br>*Use default value!* | 0/1 | RW |
| 178 | short detection timer | 0: 3.2µs<br>1: 1.6µs<br>2: 1.2µs<br>3: 0.8µs<br>*Use default value!* | 0..3 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 180 | smartEnergy actual current | This status value provides the *actual motor current* setting as controlled by coolStep™. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN.<br>actual motor current scaling factor:<br>0 … 31: 1/32, 2/32, … 32/32 | 0… 31 | RW |
| 181 | stop on stall | Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2™ load value reaches zero. | 0… 2047<br>$\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. | 0… 2047<br>$\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed. | 0… 255<br>$\left[\frac{\text{max. module current}}{255}\right]$ | RW |
| 193 | referencing mode | 1 – Only the left reference switch is searched.<br>2 – The right switch is searched and afterwards the left switch is searched.<br>3 – Three-switch-mode: the right switch is searched first and afterwards the reference switch will be searched. | 1/2/3 | RWE |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. | 0… 2047 | RWE |
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. | 0… 2047 | RWE |
| 196 | distance end switches | This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3). | 0… 8388307 | R |
| 204 | freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. | 0… 65535<br>0 = never<br>[msec] | RWE |
| 206 | actual load value | Readout of the actual load value with used for stall detection (stallGuard2™). | 0… 1023 | R |
| 208 | TMC262 driver error flags | Bit 0: stallGuard™ status (1: threshold reached)<br>Bit 1: Overtemperature (1: driver is shut down due to overtemperature)<br>Bit 2: Pre-warning overtemperature (1: Threshold is exceeded)<br>Bit 3: Short to ground A (1: Short condition detected, driver currently shut down)<br>Bit 4: Short to ground B (1: Short condition detected, driver currently shut down)<br>Bit 5: Open load A (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 6: Open load B (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 7: Stand still (1: No step impulse occurred on the step input during the last 2^20 clock cycles)<br>*Please refer to the TMC262 Datasheet for more information.* | 0/1 | R |
| 209 | encoder position | The value of an encoder register can be read out or written. | [encoder steps] | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 210 | encoder prescaler | Prescaler for the encoder. | | RWE |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. | 0… 65535<br><br>[encoder steps] | RWE |
| 214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). | 1… 65535<br>[10msec] | RWE |
| 215 | absolute encoder value | Absolute value of the encoder. | 0… 255<br>[encoder steps] | R |

## 6.1 coolStep™ related parameters

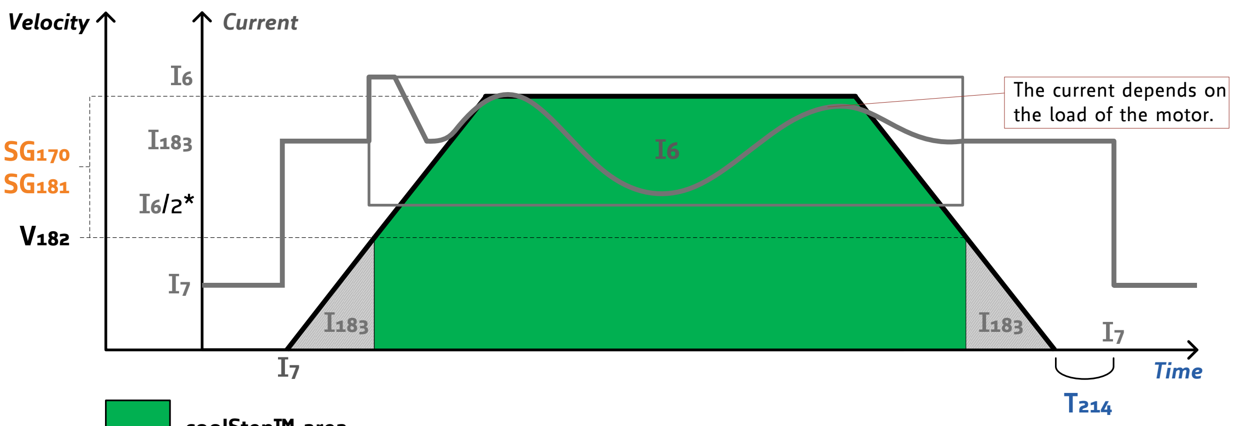
The figure below gives an overview of the coolStep™ related parameters. Please have in mind that the figure shows only one example for a drive. There are parameters which concern the configuration of the current. Other parameters are for velocity regulation and for time adjustment.

It is necessary to identify and configure the thresholds for current (I6, I7 and I183) and velocity (V182). Furthermore the stallGuard2™ feature has to be adjusted and enabled (SG170 and SG181).

The reduction or increasing of the current in the coolStep™ area (depending on the load) has to be configured with parameters I169 and I171.

*In this chapter only basic axis parameters are mentioned which concern coolStep™ and stallGuard2™. The complete list of axis parameters in chapter 6 contains further parameters which offer more configuration possibilities.*

**coolStep™ adjustment points and thresholds**



🟩 **coolStep™ area**

▨ **area without coolStep™**

I123 Current and parameter

**V123 Velocity and parameter**

**T123 Time parameter**

**SG123 stallGuard2™ parameter**

* The lower threshold of the coolStep™ current can be adjusted up to I6/4. Refer to parameter 168.

| Number | Axis parameter | Description |
|---|---|---|
| I6 | absolute max. current (CS / Current Scale) | The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on).<br>*The most important motor setting, since too high values might cause motor damage!* |
| I7 | standby current | The current limit two seconds after the motor has stopped. |
| I168 | smartEnergy current minimum (SEIMIN) | Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value.<br>Minimum motor current:<br>0 – 1/2 of CS<br>1 – 1/4 of CS |
| I169 | smartEnergy current down step | Sets the number of stallGuard2™ readings above the upper threshold necessary for each current decrement of the motor current. Number of stallGuard2™ measurements per decrement:<br>Scaling: 0… 3: 32, 8, 2, 1<br>0: slow decrement<br>3: fast decrement |
| I171 | smartEnergy current up step | Sets the current increment step. The current becomes incremented for each measured stallGuard2™ value below the lower threshold (see smartEnergy hysteresis start).<br>current increment step size:<br>Scaling: 0… 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load |
| I183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed. Please adjust the threshold speed with axis parameter 182. |
| SG170 | smartEnergy hysteresis | Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased. |
| SG181 | stop on stall | Motor stop in case of stall. |
| V182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. |
| T214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). |

*For further information about the coolStep™ feature please refer to the TMC262 Datasheet.*

# 7 Global parameters

<u>Global parameters are grouped into 4 banks:</u>

- bank 0 (global configuration of the module)
- bank 1 (user C variables)
- bank 2 (user TMCL™ variables)
- bank 3 (interrupt configuration)

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

## 7.1 Bank 0

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

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

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

**Meaning of the letters in column *Access*:**

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

Note: The TMCM-1180 does not have the parameters 0…38. They are used for modules which address more than one motor.

| Number | Global parameter | Description | | | Range | Access |
|---|---|---|---|---|---|---|
| 64 | EEPROM magic | Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration. | | | 0… 255 | RWE |
| 65 | RS232/RS485 baud rate | 0 | 9600 baud (default) | | 0… 11 | RWE |
| | | 1 | 14400 baud | | | |
| | | 2 | 19200 baud | | | |
| | | 3 | 28800 baud | | | |
| | | 4 | 38400 baud | | | |
| | | 5 | 57600 baud | | | |
| | | 6 | 76800 baud | *Not supported by Windows!* | | |
| | | 7 | 115200 baud | | | |
| | | 8 | 230400 baud | | | |
| | | 9 | 250000 baud | *Not supported by Windows!* | | |
| | | 10 | 500000 baud | *Not supported by Windows!* | | |
| | | 11 | 1000000 baud | *Not supported by Windows!* | | |
| 66 | serial address | The module (target) address for RS-232/RS-485. | | | 0… 255 | RWE |

| Number | Global parameter | Description | Range | Access |
|---|---|---|---|---|
| 67 | ASCII mode | Configure the TMCL™ ASCII interface:<br>Bit 0: 0 – start up in binary (normal) mode<br>1 – start up in ASCII mode<br>Bits 4 and 5:<br>00 – Echo back each character<br>01 – Echo back complete command<br>10 – Do not send echo, only send command reply | | RWE |
| 69 | CAN bit rate | 2 20kBit/s<br>3 50kBit/s<br>4 100kBit/s<br>5 125kBit/s<br>6 250kBit/s<br>7 500kBit/s<br>8 1000kBit/s *Default* | 2… 8 | RWE |
| 70 | CAN reply ID | The CAN ID for replies from the board (default: 2) | 0… 7ff | RWE |
| 71 | CAN ID | The module (target) address for CAN (default: 1) | 0… 7ff | RWE |
| 73 | configuration EEPROM lock flag | Write: 1234 to lock the EEPROM, 4321 to unlock it.<br>Read: 1=EEPROM locked, 0=EEPROM unlocked. | 0/1 | RWE |
| 75 | telegram pause time | Pause time before the reply via RS232 or RS485 is sent. For RS232 set to 0.<br>For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin).<br>For CAN interface this parameter has no effect! | 0… 255 | RWE |
| 76 | serial host address | Host address used in the reply telegrams sent back via RS232 or RS485. | 0… 255 | RWE |
| 77 | auto start mode | 0: Do not start TMCL™ application after power up (default).<br>1: Start TMCL™ application automatically after power up. | 0/1 | RWE |
| 80 | shutdown pin functionality | Select the functionality of the SHUTDOWN pin<br>0 – no function<br>1 – high active<br>2 – low active | 0… 2 | RWE |
| 81 | TMCL™ code protection | Protect a TMCL™ program against disassembling or overwriting.<br>0 – no protection<br>1 – protection against disassembling<br>2 – protection against overwriting<br>3 – protection against disassembling and overwriting<br>***If you switch off the protection against disassembling, the program will be erased first!***<br>***Changing this value from 1 or 3 to 0 or 2, the TMCL™ program will be wiped off.*** | 0,1,2,3 | RWE |
| 83 | CAN secondary address | Second CAN ID for the module. Switched off when set to zero. | 0… 7ff | RWE |
| 84 | coordinate storage | 0 – coordinates are stored in the RAM only (but can be copied explicitly between RAM and EEPROM)<br>1 – coordinates are always stored in the EEPROM only | 0 or 1 | RWE |
| 128 | TMCL™ application status | 0 –stop<br>1 – run<br>2 – step<br>3 – reset | 0… 3 | R |
| 129 | download mode | 0 – normal mode<br>1 – download mode | 0/1 | R |
| 130 | TMCL™ program counter | The index of the currently executed TMCL™ instruction. | | R |

| Number | Global parameter | Description | Range | Access |
|---|---|---|---|---|
| 132 | tick timer | A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value. | | RW |
| 133 | random number | Choose a random number. ***Read only!*** | 0… 2147483 647 | R |

## 7.2 Bank 1

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

## 7.3 Bank 2

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

Up to 56 user variables are available.

**Meaning of the letters in column *Access*:**

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

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

## 7.4 Bank 3

Bank 3 contains interrupt parameters. Some interrupts need configuration (e.g. the timer interval of a timer interrupt). This can be done using the SGP commands with parameter bank 3 (SGP <type>, 3, <value>). ***The priority of an interrupt depends on its number. Interrupts with a lower number have a higher priority.***

The following table shows all interrupt parameters that can be set.

**Meaning of the letters in column *Access*:**

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

| Number | Global parameter | Description | Range | Access |
|---|---|---|---|---|
| 0 | Timer 0 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] | RWE |
| 1 | Timer 1 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] | RWE |
| 2 | Timer 2 period (ms) | Time between two interrupts (ms) | 32 bit unsigned [ms] | RWE |
| 39 | Input 0 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 40 | Input 1 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |

# 8 Hints and tips

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

## 8.1 Reference search

The built-in reference search features switching point calibration and support of one or two reference switches. The internal operation is based on a state machine that can be started, stopped and monitored (instruction RFS, no. 13). The settings of the automatic stop functions corresponding to the switches (axis parameters 12 and 13) have no influence on the reference search.

Definition of the switches

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

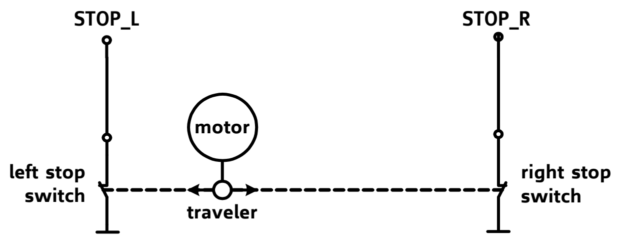


**Figure 8.1: Left and right limit switches**

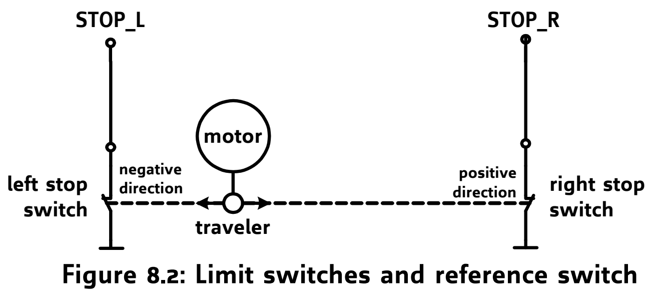


**Figure 8.2: Limit switches and reference switch**

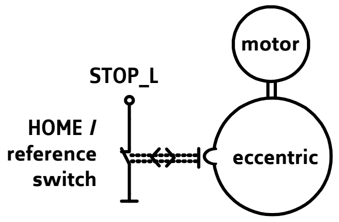


**Figure 8.3: One reference switch**

## 8.2 Changing the prescaler value of an encoder

The PD86-1180 PANdrive™ is a full mechatronic solution including a 86mm flange high torque motor, a motion controller/driver and a integrated sensOstep™ encoder. The built-in encoder has 256 steps/rotation.

For the operation with encoder please consider the following hints:

- The encoder counter can be read by software and can be used to control the exact position of the motor. This also makes closed loop operation possible.
- To read out or to change the position value of the encoder, axis parameter #209 is used.
  So, to read out the position of your encoder 0 use *GAP 209, 0*. The position values can also be changed using command SAP 209, 0, <n>, with n = ± 0,1,2,…
- To change the encoder settings, axis parameter #210 is used. For changing the prescaler of the encoder 0 use *SAP 210, 0, <p>*.
- Automatic motor stop on deviation error is also usable. This can be set using axis parameter 212 (maximum deviation). This function is turned off when the maximum deviation is set to 0.

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

| Value for <p> | Resulting prescaler | SAP command for motor 0 SAP 210, 0, <p> | Microstep solution of axis parameter 140 |
|---|---|---|---|
| 25600 | 50 | SAP 210, 0, 25600 | 8 (256 micro steps) |
| 12800 | 25 | SAP 210, 0, 12800 | 7 (128 micro steps) |
| 6400 | 12.5 *(default)* | SAP 210, 0, 6400 | 6 (64 micro steps) |
| 3200 | 6.25 | SAP 210, 0, 3200 | 5 (32 micro steps) |
| 1600 | 3.125 | SAP 210, 0, 1600 | 4 (16 micro steps) |
| 800 | 1.5625 | SAP 210, 0, 800 | 3 (8 micro steps) |
| 400 | 0.78125 | SAP 210, 0, 400 | 2 (4 micro steps) |
| 200 | 0.390625 | SAP 210, 0, 200 | 1 (2 micro steps) |

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

Consider the following formula for your calculation:

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

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

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

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

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

## 8.3 stallGuard2

The module is equipped with TMC262A-PC motor driver chip. The TMC262A-PC features load measurement that can be used for stall detection. stallGuard2™ delivers a sensorless load measurement of the motor as well as a stall detection signal. The measured value changes linear with the load on the motor in a wide range of load, velocity and current settings. At maximum motor load the stallGuard™ value goes to zero. This corresponds to a load angle of 90° between the magnetic field of the stator and magnets in the rotor. This also is the most energy efficient point of operation for the motor.

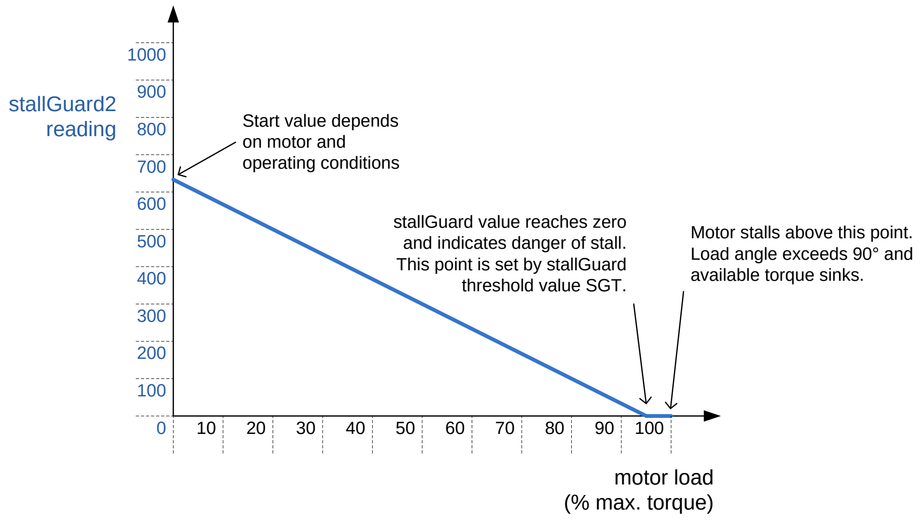


**Figure 8.4: Principle function of stallGuard2**

Stall detection means that the motor will be stopped when the load gets too high. It is configured by axis parameter #174.

Stall detection can also be used for finding the reference point. ***Do not use RFS in this case.***
***Mixed decay should be switched off when stallGuard2™ operational in order to get usable results.***

## 8.4 Using the RS485 interface

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

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

# 9 Revision history

## 9.1 Firmware revision

| Version | Date | Description |
|---|---|---|
| 4.26 | 2010-APR-26 | First version supporting all TMCL™ features |
| 4.27 | 2010-JUL-05 | |
| 4.28 | 2010-AUG-09 | |

## 9.2 Document revision

| Version | Date | Author | Description |
|---|---|---|---|
| 1.00 | 2010-JUN-28 | SD | Initial version |
| 1.01 | 2010-AUG-31 | SD | Minor corrections |
| 1.02 | 2010-SEP-16 | SD | Paragraph *Changing the prescaler value of an encoder* completed. |
| 1.03 | 2010-NOV-19 | SD | Value range of axis parameter 215 corrected. |
| 1.04 | 2010-DEC-22 | SD | Units of axis parameters 130, 182 and 183 corrected. Diagram for coolStep™ related parameters added. |
| 1.05 | 2011-FEB-21 | SD | Value range of axis parameter 206 corrected. Axis parameter 205 deleted. The functionality of this parameter is handled by parameter 174. |
| 1.06 | 2011-MAR-21 | SD | Minor changes |
| 1.07 | 2011-SEP-13 | SD | Axis parameter 181 corrected. |

# 10 References

| | |
|---|---|
| [TMCM-1180 / PD86-1180] | TMCM-1180 and PD86-1180 Hardware Manual |
| [TMC262] | TMC262 Datasheet |
| [TMCL-IDE] | TMCL-IDE User Manual |
| [QSH8618] | QSH8618 Manual |

Please refer to www.trinamic.com.