

Firmware Version V4.45

# TMCL™ FIRMWARE MANUAL

+



+

+

## TMCM-1180 PD86-1180

1-Axis Stepper  
Controller / Driver  
5.5A RMS/ 24 or 48V DC  
USB, RS232, RS485, and CAN

+



**stallGuard<sup>TM</sup>2**

TRINAMIC Motion Control GmbH & Co. KG  
Hamburg, Germany

[www.trinamic.com](http://www.trinamic.com)



# Table of Contents

1	Features.....	4
2	Overview.....	5
3	Putting the PD86-1180 into Operation.....	6
3.1	Starting up.....	6
3.2	Testing with a Simple TMCL Program.....	9
3.3	Operating the Module in Direct Mode.....	10
4	TMCL and TMCL-IDE.....	11
4.1	Binary Command Format.....	11
4.2	Reply Format.....	12
4.2.1	Status Codes.....	13
4.3	Standalone Applications.....	13
4.4	TMCL Command Overview.....	14
4.4.1	TMCL Commands.....	14
4.4.2	Commands Listed According to Subject Area.....	15
4.5	The ASCII Interface.....	19
4.5.1	Format of the Command Line.....	19
4.5.2	Format of a Reply.....	19
4.5.3	Configuring the ASCII Interface.....	20
4.6	Commands.....	21
4.6.1	ROR (rotate right).....	21
4.6.2	ROL (rotate left).....	22
4.6.3	MST (motor stop).....	23
4.6.4	MVP (move to position).....	24
4.6.5	SAP (set axis parameter).....	26
4.6.6	GAP (get axis parameter).....	27
4.6.7	STAP (store axis parameter).....	28
4.6.8	RSAP (restore axis parameter).....	29
4.6.9	SGP (set global parameter).....	30
4.6.10	GGP (get global parameter).....	31
4.6.11	STGP (store global parameter).....	32
4.6.12	RSGP (restore global parameter).....	33
4.6.13	RFS (reference search).....	34
4.6.14	SIO (set input / output).....	35
4.6.15	GIO (get input/output).....	37
4.6.16	CALC (calculate).....	39
4.6.17	COMP (compare).....	40
4.6.18	JC (jump conditional).....	41
4.6.19	JA (jump always).....	42
4.6.20	CSUB (call subroutine).....	43
4.6.21	RSUB (return from subroutine).....	44
4.6.22	WAIT (wait for an event to occur).....	45
4.6.23	STOP (stop TMCL program execution).....	46
4.6.24	SCO (set coordinate).....	47
4.6.25	GCO (get coordinate).....	48
4.6.26	CCO (capture coordinate).....	49
4.6.27	ACO (accu to coordinate; valid from TMCL version 4.18 on).....	50
4.6.28	CALCX (calculate using the X register).....	51
4.6.29	AAP (accumulator to axis parameter).....	52
4.6.30	AGP (accumulator to global parameter).....	53
4.6.31	CLE (clear error flags).....	54
4.6.32	VECT (set interrupt vector).....	55
4.6.33	EI (enable interrupt).....	56
4.6.34	DI (disable interrupt).....	57
4.6.35	RETI (return from interrupt).....	58
4.6.36	Customer Specific TMCL Command Extension (UF0... UF7 / User Function).....	58
4.6.37	Request Target Position Reached Event.....	59

4.6.38	BIN (return to binary mode) .....	59
4.6.39	TMCL Control Functions .....	60
5	Axis Parameters .....	62
5.1	coolStep Related Parameters .....	68
6	Global Parameters .....	70
6.1	Bank 0 .....	70
6.2	Bank 1 .....	72
6.3	Bank 2 .....	73
6.4	Bank 3 .....	73
7	Hints and Tips .....	74
7.1	Reference Search .....	74
7.2	Changing the Prescaler Value of an Encoder .....	75
7.3	stallGuard2 .....	76
7.4	Using the RS485 interface .....	76
8	TMCL Programming Techniques and Structure .....	77
8.1	Initialization .....	77
8.2	Main Loop .....	77
8.3	Using Symbolic Constants .....	77
8.4	Using Variables .....	78
8.5	Using Subroutines .....	78
8.6	Mixing Direct Mode and Standalone Mode .....	79
9	Life Support Policy .....	80
10	Revision History .....	81
10.1	Firmware Revision .....	81
10.2	Document Revision .....	81
11	References .....	82

# 1 Features

The PD86-1180 is a full mechatronic solution with state of the arte feature set. It is highly integrated and offers a convenient handling. The PD86-1180 consists of a NEMA 34 (flange size 86mm) stepper motor, controller/driver electronics and integrated encoder.

The TMCM-1180 is an intelligent stepper motor controller/driver module featuring the new outstanding coolStep™ technology for sensorless load dependent current control. This allows energy efficient motor operation. With the advanced stallGuard2™ feature the load of the motor can be detected with high resolution. The module is designed to be mounted directly on an 86mm flange QMot stepper motor.

## 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

## 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 (TMC428/429 motion controller)
- On the fly alteration of motor parameters (e.g. position, velocity, acceleration)
- High performance microcontroller for overall system control and serial communication protocol handling

## Bipolar stepper motor driver

- Up to 256 microsteps per full step
- High-efficient operation, low power dissipation
- Dynamic current control
- Integrated protection
- stallGuard2 feature for stall detection
- coolStep feature for reduced power consumption and heat dissipation

## 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
- Standalone TMCL operation or remote controlled operation
- Program memory (non volatile) for up to 2048 TMCL commands
- PC-based application development software TMCL-IDE available for free
- CANopen: CiA 301 + CiA 402 (homing mode, profile position mode and velocity mode) supported

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

## 2 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 TMC428/429 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.

## 3 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!

### 3.1 Starting up

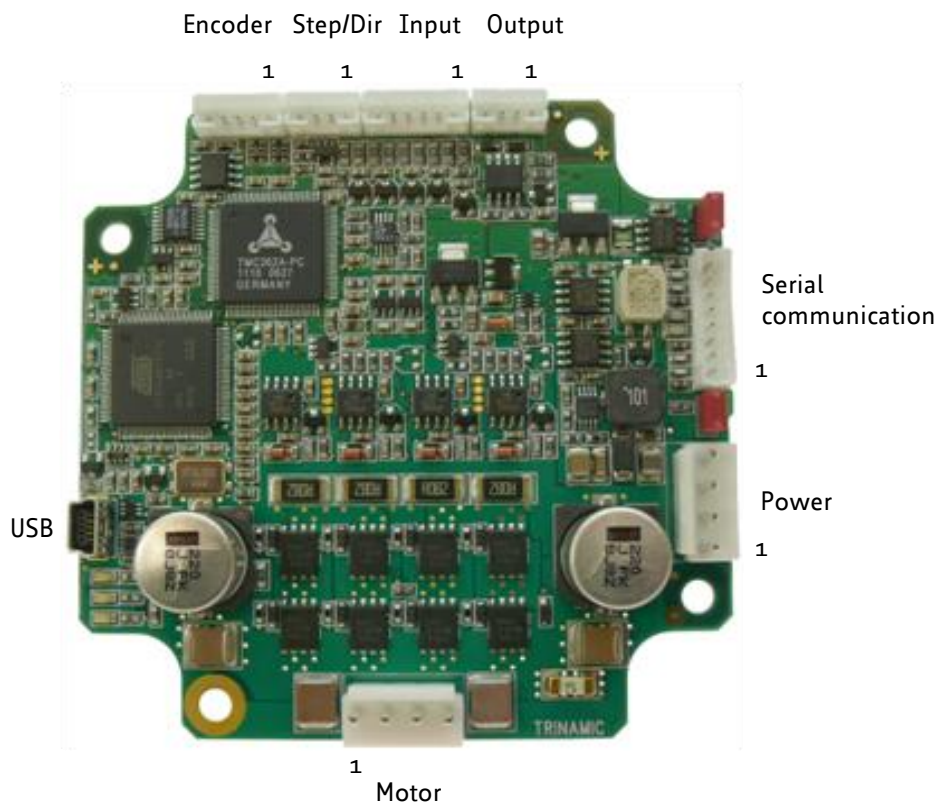


Figure 3.1 Overview connectors

## 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.


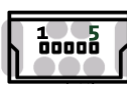
	Pin	Label	Description
	1	RS232_TxD	RS232 transmit data
	2	RS232_RxD	RS232 receive data
	3	GND	Module ground (system and signal ground)
	4	CAN_H	CAN_H bus line (dominant high)
	5	CAN_L	CAN_L bus line (dominant low)
	6	GND	Module ground (system and signal ground)
	7	RS485+	RS485 non-inverted bus signal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

Figure 3.2: RS232, RS485, and CAN connector

### b) Connect the USB interface


A 5-pin mini-USB connector is available on the board.

Download and install the file *TMCM-1180.inf* ([www.trinamic.com](http://www.trinamic.com)).

	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.

	Pin	Label	Description
	1	+U <sub>Driver</sub>	Module + driver stage power supply input (nom. +48V DC)
	2	+U <sub>Logic</sub>	(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.

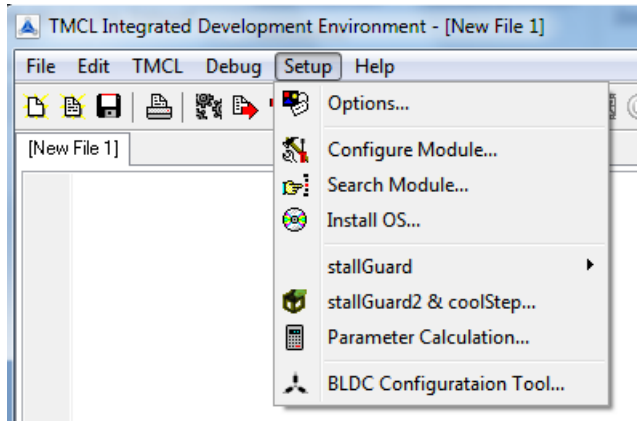
If this does not occur, switch power OFF and check your connections as well as the power supply.

#### 4. **Start the TMCL-IDE software development environment**

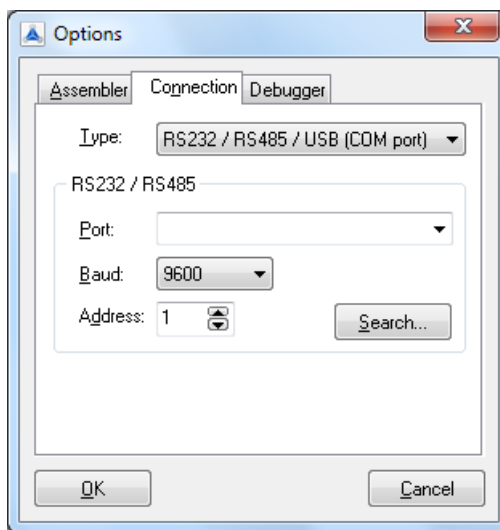
The TMCL-IDE is available on the TechLibCD and on [www.trinamic.com](http://www.trinamic.com).

##### 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**.



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



Please refer to the TMCL-IDE User Manual for more information about connecting the other interfaces (see [www.TRINAMIC.com](http://www.TRINAMIC.com)).



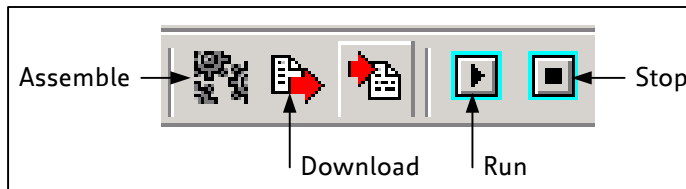
## 3.2 Testing with a Simple TMCL Program

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 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

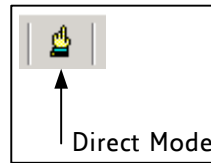
    SAP 4, 0, 500            //Set max. Velocity
    SAP 5, 0, 50            //Set max. Acceleration
Loop: MVP ABS, 0, 10000      //Move to Position 10000
    WAIT POS, 0, 0          //Wait until position reached
    MVP ABS, 0, -10000      //Move to Position -10000
    WAIT POS, 0, 0          //Wait until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

### 3.3 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.

The screenshot shows the 'TMCL Direct Mode - TMCM-1180' window. It has three main sections:

- TMCL Instruction Selector:** Contains dropdowns for 'Instruction' (set to '1 - ROR rotate right'), 'Type' (set to '0 - <don't care>'), 'Motor / Bank' (set to '0 - Motor 0'), and 'Value' (set to '100'). Below these are buttons for 'Execute', 'Copy', and 'Copy to editor'.
- Manual Instruction Input:** A table with columns: Address, Instruction, Type, Motor/Bank, Value, and Datagram. The first row contains: Address: 7, Instruction: 0, Type: 0, Motor/Bank: 0, Value: 0, Datagram: 07 00 00 00 00 00 00 00 07. Below the table is an 'Execute' button.
- Answer:** A table with columns: Host, Target, Status, Instr., Value, and Datagram. The first row contains: Host: 2, Target: 1, Status: 100, Instr.: 1, Value: 100, Datagram: 02 01 64 01 00 00 00 64 CC. Below the table is a 'Close' button.

#### Examples:

- ROR rotate right, motor 0, value 100 -> Click *Execute*. The first motor is rotating now.
- MST motor stop, motor 0 -> Click *Execute*. The first motor stops now.

#### Note

Chapter 5 (axis parameters) includes a diagram which shows important coolStep related axis parameters and their functions.

## 4 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.

### 4.1 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
```

## 4.2 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!

### 4.2.1 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

## 4.3 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.

## 4.4 TMCL Command Overview

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

### 4.4.1 TMCL Commands

Command	Number	Parameter	Description
ROR	1	<motor number>, <velocity>	Rotate right with specified velocity
ROL	2	<motor number>, <velocity>	Rotate left with specified velocity
MST	3	<motor number>	Stop motor movement
MVP	4	ABS REL COORD, <motor number>, <position offset>	Move to position (absolute or relative)
SAP	5	<parameter>, <motor number>, <value>	Set axis parameter (motion control specific settings)
GAP	6	<parameter>, <motor number>	Get axis parameter (read out motion control specific settings)
STAP	7	<parameter>, <motor number>	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

## 4.4.2 Commands Listed According to Subject Area

### 4.4.2.1 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

### 4.4.2.2 Parameter Commands

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

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

### 4.4.2.3 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
CSUB	23	Call subroutine
RSUB	24	Return from subroutine
WAIT	27	Wait for a specified event
STOP	28	End of a TMCL program

### 4.4.2.4 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

#### 4.4.2.5 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 and GGP to the module (e.g. to query the actual position of the motor) without affecting the flow of the TMCL program running on the module.

#### 4.4.2.6 Interrupt Commands

Due to some customer requests, interrupt processing has been introduced in the TMCL firmware s.

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

##### 4.4.2.6.1 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 included file *Interrupts.inc* for symbolic constants of the interrupt numbers.

##### 4.4.2.6.2 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).

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

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



#### 4.4.2.6.3 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	Stall (stallGuard2)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change

#### 4.4.2.6.4 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 4.6.9) for further information about that.

#### 4.4.2.6.5 Using Interrupts in TMCL

For using an interrupt proceed as follows:

- 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

#### EXAMPLE FOR A TIMER INTERRUPT:

```

VECT 0, TimeroIrq //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
TimeroIrq:
  GIO 0, 2 //check if OUTo is high
  JC NZ, OutoOff //jump if not
  SIO 0, 2, 1 //switch OUTo high
  RETI //end of interrupt
OutoOff:
  SIO 0, 2, 0 //switch OUTo low
  RETI //end of interrupt

```

In the example above, the interrupt numbers are used directly. To make the program better readable 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 program above 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.

#### 4.4.2.7 ASCII Commands

Mnemonic	Command number	Meaning
-	139	Enter ASCII mode
BIN	-	Quit ASCII mode and return to binary mode. This command can only be used in ASCII mode.

## 4.5 The ASCII Interface

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

### 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.

Only direct mode commands can be entered in ASCII mode!

### SPECIAL COMMANDS WHICH 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.

### ENTERING AND LEAVING ASCII MODE:

1. The ASCII command line interface is entered by sending the binary command 139 (*enter ASCII mode*).
2. Afterwards the commands are entered as in the TMCL-IDE.
3. For leaving the ASCII mode and re-enter the binary mode enter the command BIN.

### 4.5.1 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.

#### EXAMPLES FOR VALID COMMAND LINES:

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

*The command lines above 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.*

### 4.5.2 Format of a Reply

After executing the command the module sends back a reply in ASCII format.

The 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.*

### 4.5.3 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 6).

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. Characters 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.

## 4.6 Commands

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

### 4.6.1 ROR (rotate right)

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

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

The module is based on the TMC428/429 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> 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$01	\$00	\$00	\$00	\$00	\$01	\$5e

## 4.6.2 ROL (rotate left)

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

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

The module is based on the TMC428/429 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$02	\$00	\$00	\$00	\$00	\$04	\$b0

### 4.6.3 MST (motor stop)

With this command the motor will be instructed to stop.

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$03	\$00	\$00	\$00	\$00	\$00	\$00

## 4.6.4 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.

The range of the MVP command is 32 bit signed ( $-2.147.483.648... +2.147.483.647$ ). Positioning can be interrupted using MST, ROL or ROR commands.

### THREE OPERATION TYPES ARE AVAILABLE:

- Moving to an absolute position in the range from  $-2.147.483.648... +2.147.483.647$  ( $-2^{31}... 2^{31}-1$ ).
- Starting a relative movement by means of an offset to the actual position. In this case, the new resulting position value must not exceed the above mentioned limits, too.
- Moving the motor to a (previously stored) coordinate (refer to SCO for details).

*Please note, that the distance between the actual position and the new one should not be more than  $2.147.483.647$  ( $2^{31}-1$ ) microsteps. Otherwise the motor will run in the opposite direction in order to take the shorter distance.*

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$04	\$00	\$00	\$00	\$01	\$5f	\$90



**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$04	\$01	\$00	\$ff	\$ff	\$fc	\$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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$04	\$02	\$00	\$00	\$00	\$00	\$08

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

### 4.6.5 SAP (set axis parameter)

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

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

**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)

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$05	\$06	\$00	\$00	\$00	\$00	\$c8

## 4.6.6 GAP (get axis parameter)

Most parameters of the TMC1180 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).

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

**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)

**Example:**

Get the actual position of motor

*Mnemonic:* GAP 0, 1

*Binary:*

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

*Reply:*

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

⇒ **status=no error, position=711**

## 4.6.7 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.

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

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

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

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

**Binary representation:**

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

\*<sup>1</sup>motor number is always 0 as only one motor is involved

\*<sup>2</sup>the 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 5	0	s. chapter 5

**Example:**

Store the maximum speed of motor

*Mnemonic:* STAP 4, 0

*Binary:*

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

The STAP command will not have any effect when the configuration EEPROM is locked (refer to 6.1). In direct mode, the error code 5 (configuration EEPROM locked, see also section 4.2.1) will be returned in this case.

### 4.6.8 RSAP (restore axis parameter)

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

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

**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)

**Example:**

Restore the maximum current of motor

*Mnemonic:* RSAP 6, 0

*Binary:*

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

## 4.6.9 SGP (set global parameter)

With this command most of the module specific parameters not directly related to motion control can be specified and the TMCL user variables can be changed. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.

**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.**

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

**Internal function:** the parameter format is converted. 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)

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$09	\$42	\$00	\$00	\$00	\$00	\$03

### 4.6.10 GGP (get global parameter)

All global parameters can be read with this function. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.

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

**Internal function:** The parameter is read out of the correct position in the appropriate device.

**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)

**Example:**

Get the serial address of the target device

*Mnemonic:* GGP 66, 0

*Binary:*

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

*Reply:*

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

⇒ **Status=no error, Value=1**

### 4.6.11 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.

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

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

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

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

**Binary representation:**

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

**Reply in direct mode:**

STATUS	VALUE
100 – OK	(don't care)

**Example:**

Store the user variable #42

*Mnemonic:* STGP 42, 2

*Binary:*

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



## 4.6.12 RSGP (restore global parameter)

With this command the contents of a TMCL user variable can be restored from the EEPROM. By default, most parameters are automatically restored after power up. A single parameter that has been changed before can be reset by this instruction.

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

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

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

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

**Binary representation:**

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

**Reply structure in direct mode:**

STATUS	VALUE
100 – OK	(don't care)

**Example:**

Restore user variable #42

*Mnemonic:* RSGP 42, 2

*Binary:*

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

### 4.6.13 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 5). The reference search can be started, stopped, and the actual status of the reference search can be checked.

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

**Related commands:** WAIT

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

**Binary representation:**

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
13	0 START – start ref. search 1 STOP – abort ref. search 2 STATUS – get status	0	see below

**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
	other values	ref. search active

**Example:**

Start reference search of motor 0

*Mnemonic:* RFS START, 0

*Binary:*

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

With this module it is possible to use stall detection instead of a reference search.

#### 4.6.14 SIO (set input / output)

- SIO sets the status of the general digital output either to low (0) or to high (1). Bank 2 is used for this purpose.
- SIO is also used to switch the pull-up resistors for all digital inputs ON or OFF. Bank 0 is used for this purpose.

**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> 0/1

**Reply structure:**

STATUS	VALUE
100 – OK	don't care

**Example:**


Set OUT\_7 to high (bank 2, output 7)

*Mnemonic:* SIO 7, 2, 1

*Binary:*

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

#### 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.
- 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

```

**COMMAND FOR SWITCHING THE PULL-UP RESISTORS FOR STOP\_L, STOP\_R, AND HOME**

Every pull-up resistor can be switched individually:

- Bit 0 is used for switching the pull-up resistor of HOME.
- Bit 1 is used for switching the pull-up resistor of STOP\_L.
- Bit 2 is used for switching the pull-up resistor of STOP\_R.

	Pin	I/O port	Command	Range
	3	STOP_L	SIO 0, 0,<bitmask>	0... 7
	4	STOP_R		
	5	HOME		

### 4.6.15 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.

#### GIO IN STANDALONE MODE

In standalone mode the requested value is copied to the *accumulator* (accu) for further processing purposes such as conditioned jumps.

#### GIO IN DIRECT MODE

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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$0f	\$00	\$01	\$00	\$00	\$00	\$00


#### Reply:

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

⇒ value: 506

#### 4.6.15.1 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
	3	OUT_0	SIO 0, 2, <n>, (n=0/1)	1/0
	4	OUT_1	SIO 1, 2, <n>, (n=0/1)	1/0

**READING ALL DIGITAL INPUTS WITH ONE GIO COMMAND:**

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

**USE FOLLOWING PROGRAM TO REPRESENT THE STATES OF THE INPUT LINES ON THE OUTPUT LINES:**

```

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


```

**THE FOLLOWING COMMAND CAN BE USED FOR READING OUT THE ENABLE PIN OC\_EN OF THE STEP/DIR INTERFACE**

```
GIO 12, 0
```


**4.6.15.2 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

**4.6.15.3 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

### 4.6.16 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 1 SUB – subtract from accu 2 MUL – multiply accu by 3 DIV – divide accu by 4 MOD – modulo divide by 5 AND – logical and accu with 6 OR – logical or accu with 7 XOR – logical exor accu with 8 NOT – logical invert accu 9 LOAD – load operand to accu	(don't care)	<operand>

**Example:**

Multiply accu by -5000

*Mnemonic:* CALC MUL, -5000

*Binary:*

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

## 4.6.17 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.

**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/ CALC/CALCX). The internal arithmetic status flags are set according to the comparison result.

**Related commands:** JC (jump conditional), GAP, GGP, 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$14	\$00	\$00	\$00	\$00	\$03	\$e8



## 4.6.18 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.

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

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

**Mnemonic:** JC <condition>, <label>

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

**Binary representation:**

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

**Example:**

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

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

COMP 1000     //compare actual value to 1000

JC GE, Label     //jump, type: 5 greater/equal

...

...

Label: ROL 0, 1000

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

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

## 4.6.19 JA (jump always)

Jump to a fixed address in the TMCL program memory.

This command is intended for standalone operation only.

**Internal function:** the TMCL program counter is set to the 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$16	\$00	\$00	\$00	\$00	\$00	\$14

## 4.6.20 CSUB (call subroutine)

This function calls a subroutine in the TMCL program memory.

This command is intended for standalone operation only.

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

**Related commands:** RSUB, JA

**Mnemonic:** CSUB <Label>

**Binary representation:**

INSTRUCTION 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$17	\$00	\$00	\$00	\$00	\$00	\$64

## 4.6.21 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.

**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 4.6.20).

*Binary format of RSUB:*

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

## 4.6.22 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.

### 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 <sup>*1</sup>	don't care	<no. of ticks <sup>*1</sup> >
	1 POS - target position reached	0 <sup>*2</sup>	<no. of ticks <sup>*1</sup> for timeout>, 0 for no timeout
	2 REFSW – reference switch	0 <sup>*2</sup>	<no. of ticks <sup>*1</sup> for timeout>, 0 for no timeout
	3 LIMSW – limit switch	0 <sup>*2</sup>	<no. of ticks <sup>*1</sup> for timeout>, 0 for no timeout
	4 RFS – reference search completed	0 <sup>*2</sup>	<no. of ticks <sup>*1</sup> for timeout>, 0 for no timeout

<sup>\*1</sup> one tick is 10 milliseconds (in standard firmware)

<sup>\*2</sup> 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$1b	\$01	\$00	\$00	\$00	\$00	\$00

### 4.6.23 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$1c	\$00	\$00	\$00	\$00	\$00	\$00

## 4.6.24 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).

**Note:** 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> $-2^{31} \dots +2^{31}$

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$1e	\$01	\$00	\$00	\$00	\$03	\$e8

Two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate to the EEPROM:

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.

## 4.6.25 GCO (get coordinate)

This command makes possible to read out a previously stored coordinate.

In *standalone mode* the requested value will be copied to the accumulator register for further processing purposes such as conditioned jumps.

In *direct mode*, the value is only output in the value field of the reply, without affecting the accumulator.

Please note that the coordinate number 0 is always stored in RAM only.

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).

**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

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$1f	\$01	\$00	\$00	\$00	\$00	\$00

*Reply:*

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

⇒ **Value: 0**

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:

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.



## 4.6.26 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 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

**Reply in direct mode:**

STATUS	VALUE
100 – OK	don't care

**Example:**

Store current position of the axis 0 to coordinate 3

*Mnemonic:* CCO 3, 0

*Binary:*

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

### 4.6.27 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 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

**Reply in direct mode:**

STATUS	VALUE
100 – OK	don't care

**Example:**

Copy the actual value of the accumulator to coordinate 1 of motor 0

*Mnemonic:* ACO 1, 0

*Binary:*

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

## 4.6.28 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 1 SUB – subtract X register from accu 2 MUL – multiply accu by X register 3 DIV – divide accu by X-register 4 MOD – modulo divide accu by x-register 5 AND – logical and accu with X-register 6 OR – logical or accu with X-register 7 XOR – logical exor accu with X-register 8 NOT – logical invert X-register 9 LOAD – load accu to X-register 10 SWAP – swap accu with X-register	(don't care)	(don't care)

**Example:**

Multiply accu by X-register

Mnemonic: CALCX MUL

*Binary:*

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

### 4.6.29 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.

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

**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)

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$22	\$00	\$00	\$00	\$00	\$00	\$00

### 4.6.30 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:**

The global parameters in bank 0 are EEPROM-only and thus should not be modified automatically by a standalone application.

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

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

**Binary representation:**

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

**Reply in direct mode:**

STATUS	VALUE
100 – OK	(don't care)

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

**Example:**

Copy accumulator to TMCL user variable #3

*Mnemonic:* AGP 3, 2

*Binary:*

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

### 4.6.31 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.

**Related commands:** JC

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

**Binary representation:**

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
36	0 – (ALL) all flags 1 – (ETO) timeout 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$24	\$01	\$00	\$00	\$00	\$00	\$00

### 4.6.32 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	Stall (stallGuard2)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change

**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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$25	\$03	\$00	\$00	\$00	\$01	\$F4

### 4.6.33 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	Stall (stallGuard2)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change

**Examples:**

Enable interrupts globally  
EI, 255

*Binary format of EI:*

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

Enable interrupt when target position reached  
EI, 3

*Binary format of EI:*

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



### 4.6.34 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	Stall (stallGuard2)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change

**Examples:**

Disable interrupts globally  
DI, 255

*Binary format of DI:*

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

Disable interrupt when target position reached  
DI, 3

*Binary format of DI:*

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

### 4.6.35 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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$26	\$00	\$00	\$00	\$00	\$01	\$00

### 4.6.36 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
Function	Target-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$02	\$01	(user defined)	64... 71	(user defined)	(user defined)	(user defined)	(user defined)

### 4.6.37 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 standalone mode, it is covered by the WAIT command) and hence does not have a mnemonic.

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

**Mnemonic:** ---

**Binary representation:**

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
138	0/1	(don't care)	1

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

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

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

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

### 4.6.38 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.

### 4.6.39 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).

TMCL control commands have no mnemonics, as they cannot be used in TMCL programs. These Functions are to be used only by the TMCL-IDE (e.g. to download a TMCL application into the module).

#### CONTROL COMMANDS THAT COULD BE USEFUL FOR A USER HOST APPLICATION ARE:

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

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

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

**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. 1180V1.30)

- 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)


## 5 Axis Parameters

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

### MEANING OF THE LETTERS IN COLUMN ACCESS:

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

 Basic parameters should be adjusted to motor / application for proper module operation.

 Parameters for the more experienced user – please do not change unless you are absolutely sure.

Number	Axis Parameter	Description	Range [Unit]	Acc.																																								
0	Target (next) position	The desired position in position mode (see ramp mode, no. 138).	$\pm 2^{31}-1$ [μsteps]	RW																																								
1	Actual position	The current position of the motor. Should only be overwritten for reference point setting.	$\pm 2^{31}-1$ [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.	$\pm 2047$ $\left[ \frac{16\text{MHz}}{65536} \cdot 2^{\text{pulse\_divisor}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RW																																								
3	Actual speed	The current rotation speed.	$\pm 2047$ $\left[ \frac{16\text{MHz}}{65536} \cdot 2^{\text{pulse\_divisor}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RW																																								
4	Maximum positioning speed	Should not exceed the physically highest possible value. Adjust the pulse divisor (axis parameter 154), if the speed value is very low (<50) or above the upper limit.	0... 2047 $\left[ \frac{16\text{MHz}}{65536} \cdot 2^{\text{pulse\_divisor}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RWE																																								
5	Maximum acceleration	The limit for acceleration (and deceleration). Changing this parameter requires recalculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically.	0... 2047*	RWE																																								
6	Absolute max. current (CS / Current Scale)	<div>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.</div> <table><tr><td>0... 7</td><td>79... 87</td><td>160... 167</td><td>240... 247</td></tr><tr><td>8... 15</td><td>88... 95</td><td>168... 175</td><td>248... 255</td></tr><tr><td>16... 23</td><td>96... 103</td><td>176... 183</td><td></td></tr><tr><td>24... 31</td><td>104... 111</td><td>184... 191</td><td></td></tr><tr><td>32... 39</td><td>112... 119</td><td>192... 199</td><td></td></tr><tr><td>40... 47</td><td>120... 127</td><td>200... 207</td><td></td></tr><tr><td>48... 55</td><td>128... 135</td><td>208... 215</td><td></td></tr><tr><td>56... 63</td><td>136... 143</td><td>216... 223</td><td></td></tr><tr><td>64... 71</td><td>144... 151</td><td>224... 231</td><td></td></tr><tr><td>72... 79</td><td>152... 159</td><td>232... 239</td><td></td></tr></table> <div>The most important motor setting, since too high values might cause motor damage!</div>	0... 7	79... 87	160... 167	240... 247	8... 15	88... 95	168... 175	248... 255	16... 23	96... 103	176... 183		24... 31	104... 111	184... 191		32... 39	112... 119	192... 199		40... 47	120... 127	200... 207		48... 55	128... 135	208... 215		56... 63	136... 143	216... 223		64... 71	144... 151	224... 231		72... 79	152... 159	232... 239		$I_{peak} = < value > \times \frac{9.3A}{255}$ $I_{RMS} = < value > \times \frac{6.6A}{255}$	RWE
0... 7	79... 87	160... 167	240... 247																																									
8... 15	88... 95	168... 175	248... 255																																									
16... 23	96... 103	176... 183																																										
24... 31	104... 111	184... 191																																										
32... 39	112... 119	192... 199																																										
40... 47	120... 127	200... 207																																										
48... 55	128... 135	208... 215																																										
56... 63	136... 143	216... 223																																										
64... 71	144... 151	224... 231																																										
72... 79	152... 159	232... 239																																										

Number	Axis Parameter	Description	Range [Unit]	Acc.																		
7	Standby current	The current limit two seconds after the motor has stopped.	0... 255 $I_{peak} = <value> \times \frac{9.3A}{255}$ $I_{RMS} = <value> \times \frac{6.6A}{255}$	RWE																		
8	Target pos. reached	Indicates that the actual position equals the target position.	0/1	R																		
9	Ref. switch status	The logical state of the reference home switch.	0/1	R																		
10	Right limit switch status	The logical state of the (right) limit switch.	0/1	R																		
11	Left limit switch status	The logical state of the left limit switch (in three switch mode)	0/1	R																		
12	Right limit switch disable	If set, deactivates the stop function of the right switch	0/1	RWE																		
13	Left limit switch disable	Deactivates the stop function of the left switch resp. reference switch if set.	0/1	RWE																		
130	Minimum speed	Should always be set 1 to ensure exact reaching of the target position.	0... 2047 Default = 1 $\lceil \frac{16MHz}{65536} \cdot 2^{pulse\_divisor} \frac{\mu steps}{sec} \rceil$	RWE																		
135	Actual acceleration	The current acceleration (read only).	0... 2047*	R																		
138	Ramp mode	Automatically set when using ROR, ROL, MST and MVP. 0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided. 2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed. For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected.	0/1/2	RWE																		
140	Microstep resolution	<table><tr><td>0</td><td>full step</td></tr><tr><td>1</td><td>half step</td></tr><tr><td>2</td><td>4 microsteps</td></tr><tr><td>3</td><td>8 microsteps</td></tr><tr><td>4</td><td>16 microsteps</td></tr><tr><td>5</td><td>32 microsteps</td></tr><tr><td>6</td><td>64 microsteps</td></tr><tr><td>7</td><td>128 microsteps</td></tr><tr><td>8</td><td>256 microsteps</td></tr></table>	0	full step	1	half step	2	4 microsteps	3	8 microsteps	4	16 microsteps	5	32 microsteps	6	64 microsteps	7	128 microsteps	8	256 microsteps	0... 8	RWE
0	full step																					
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]	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																		

Number	Axis Parameter	Description	Range [Unit]	Acc.
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	RW
161	Double step enable	Every edge of the cycle releases a step/microstep. <i>It does not make sense to activate this parameter for internal use.</i> Double step enable can be used with Step/Dir interface. 0 – double step off 1 – double step on	0/1	RW
162	Chopper blank time	Selects the comparator <i>blank time</i> . 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.	0... 3	RW
163	Chopper mode	Selection of the chopper mode: 0 – spread cycle 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. 0 – fast decrement 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. -3... -1 negative hysteresis end setting 0 zero hysteresis end setting 1... 12 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.  Off time setting for constant $t_{off}$ chopper: $N_{CLK} = 12 + 32 \cdot t_{OFF}$ (Minimum is 64 clocks)  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. minimum motor current: 0 – 1/2 of CS 1 – 1/4 of CS	0/1	RW



Number	Axis Parameter	Description	Range [Unit]	Acc.	
169	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:  Scaling: 0... 3: 32, 8, 2, 1 0: slow decrement 3: fast decrement	0... 3	RW	
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.	0... 15	RW	
		Hysteresis: (smartEnergy hysteresis value + 1) * 32			
		Upper stallGuard threshold: (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).  current increment step size:  Scaling: 0... 3: 1, 2, 4, 8 0: slow increment 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. <i>In most cases it is expedient to set the filtered mode before using coolStep.</i> <i>Use the standard mode for step loss detection.</i> 0 – standard mode 1 – filtered mode	0/1	RW	
174	stallGuard2 threshold	This signed value controls stallGuard2 <i>threshold</i> 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.	-64... 63	RW	
		0			Indifferent value
		1... 63			less sensitivity
		-1... -64			higher sensitivity
175	Slope control high side	Determines the slope of the motor driver outputs. <i>Set to 2 or 3 for this module or rather use the default value.</i> 0: lowest slope 3: fastest slope	0... 3	RW	

Number	Axis Parameter	Description	Range [Unit]	Acc.						
176	Slope control low side	Determines the slope of the motor driver outputs. <i>Set identical to slope control high side.</i>	0... 3	RW						
177	short protection disable	0: Short to GND protection is on 1: Short to GND protection is disabled <i>Use default value!</i>	0/1	RW						
178	Short detection timer	0: 3.2µs 1: 1.6µs 2: 1.2µs 3: 0.8µs <i>Use default value!</i>	0..3	RW						
179	Vsense	sense resistor voltage based current scaling 0: Full scale sense resistor voltage is 1/18 VDD 1: Full scale sense resistor voltage is 1/36 VDD (refers to a current setting of 31 and DAC value 255) <i>Use default value. Do not change!</i>	0/1	RW						
180	smartEnergy actual current	This status value provides the <i>actual motor current</i> setting as controlled by coolStep. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN. <u>actual motor current scaling factor:</u> 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	RW						
182	smartEnergy threshold speed	Above this speed coolStep becomes enabled.	0... 2047 $\left[ \frac{16\text{MHz}}{65536} \cdot 2^{\text{pulse\_divisor}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RW						
183	smartEnergy slow run current	Sets the motor current which is used below the threshold speed.	0... 255 $I_{\text{peak}} = < \text{value} > \times \frac{9.3A}{255}$ $I_{\text{RMS}} = < \text{value} > \times \frac{6.6A}{255}$	RW						
193	Reference search mode	<table><tr><td>1</td><td>search left stop switch only</td></tr><tr><td>2</td><td>search right stop switch, then search left stop switch</td></tr><tr><td>3</td><td>search right stop switch, then search left stop switch from both sides</td></tr></table> <i>Adding 128 to these values reverses the polarity of the home switch input.</i>	1	search left stop switch only	2	search right stop switch, then search left stop switch	3	search right stop switch, then search left stop switch from both sides	1... 3	RWE
1	search left stop switch only									
2	search right stop switch, then search left stop switch									
3	search right stop switch, then search left stop switch from both sides									
194	Reference search speed	For the reference search this value directly specifies the search speed.	0... 2047	RWE						
195	Reference switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.	0... 2047	RWE						
196	Reference switch distance	This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3).	0... 2.147.483.647	R						
200	Boost current	Current used for acceleration and deceleration phases. If set to 0 the same current as set by axis parameter 6 will be used.	0... 255 $I_{\text{peak}} = < \text{value} > \times \frac{9.3A}{255}$ $I_{\text{RMS}} = < \text{value} > \times \frac{6.6A}{255}$	RWE						

Number	Axis Parameter	Description		Range [Unit]	Acc.
204	Freewheeling	Time after which the power to the motor will be cut when its velocity has reached zero.		0... 65535 0 = never [msec]	RWE
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	stallGuard2 status (1: threshold reached)	0/1	R
		Bit 1	Overtemperature (1: driver is shut down due to overtemperature)		
		Bit 2	Pre-warning overtemperature (1: Threshold is exceeded)		
		Bit 3	Short to ground A (1: Short condition detected, driver currently shut down)		
		Bit 4	Short to ground B (1: Short condition detected, driver currently shut down)		
		Bit 5	Open load A (1: no chopper event has happened during the last period with constant coil polarity)		
		Bit 6	Open load B (1: no chopper event has happened during the last period with constant coil polarity)		
		Bit 7	Stand still (1: No step impulse occurred on the step input during the last 2^20 clock cycles)		
209	Encoder position	The value of an encoder register can be read out or written.		[encoder steps]	RW
210	Encoder prescaler	Prescaler for the sens0step encoder.		See paragraph 7.1	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 [encoder steps]	RWE
214	Power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec).		1... 65535 [10msec]	RWE
215	Absolute encoder value	Absolute value of the encoder.		0... 255 [encoder steps]	R
254	Step/Dir mode	1	Use of the ENABLE input on step/dir connector to switch between hold current and run current (no automatic switching)	1... 5	RWE
		2	Automatic switching between hold and run current: after the first step pulse the module automatically switches over to run current, and a configurable time after the last step pulse the module automatically switches back to hold current. The ENABLE input on the step/dir connector does not have any functionality.		
		3	Always use run current, never switch to hold current. The ENABLE input on the step/dir connector does not have any functionality.		
		4	Automatic current switching like (2), but the ENABLE input is used to switch the driver stage completely off or on.		
		5	Always use run current like (3), but the ENABLE pin is used to switch the driver stage completely off or on.		

\* Unit of acceleration:  $\frac{16\text{MHz}^2}{536870912 \cdot 2^{\text{puls\_divisor} + \text{ramp\_divisor}}} \frac{\text{microsteps}}{\text{sec}^2}$

## 5.1 coolStep Related Parameters

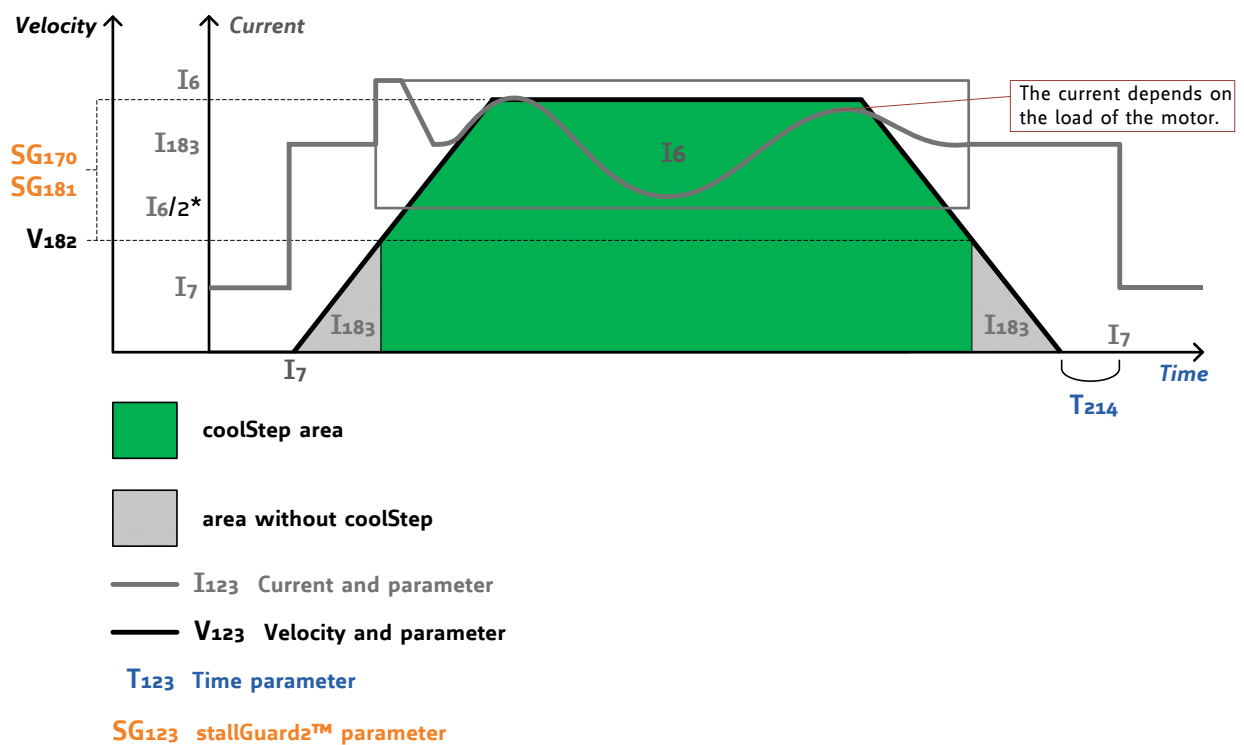
The figure below gives an overview of the coolStep related parameters. Please have in mind that the figure shows only one example for a drive. There are parameters which concern the configuration of the current. Other parameters are for velocity regulation and for time adjustment.

### THE FOLLOWING ADJUSTMENTS HAVE TO BE MADE:

- Thresholds for current ( $I_6$ ,  $I_7$  and  $I_{183}$ ) and velocity ( $V_{182}$ ) have to be identified and set.
- The stallGuard2 feature has to be adjusted and enabled with parameters  $SG_{170}$  and  $SG_{181}$ .
- The reduction or increasing of the current in the coolStep area (depending on the load) has to be configured with parameters  $I_{169}$  and  $I_{171}$ .

*In this chapter only basic axis parameters are mentioned which concern coolStep and stallGuard2. The complete list of axis parameters in chapter 5 contains further parameters which offer more possibilities for configuration.*

### coolStep™ adjustment points and thresholds



\* The lower threshold of the coolStep current can be adjusted up to  $I_6/4$ . Refer to parameter 168.

Number	Axis parameter	Description
<b>I6</b>	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). <i>The most important motor setting, since too high values might cause motor damage!</i>
<b>I7</b>	standby current	The current limit two seconds after the motor has stopped.
<b>I168</b>	smartEnergy current minimum (SEIMIN)	Sets the lower motor current limit for coolStep operation by scaling the CS (Current Scale, see axis parameter 6) value. Minimum motor current: 0 – 1/2 of CS 1 – 1/4 of CS
<b>I169</b>	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: Scaling: 0... 3: 32, 8, 2, 1 0: slow decrement 3: fast decrement
<b>I171</b>	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). current increment step size: Scaling: 0... 3: 1, 2, 4, 8 0: slow increment 3: fast increment / fast reaction to rising load
<b>I183</b>	smartEnergy slow run current	Sets the motor current which is used below the threshold speed. Please adjust the threshold speed with axis parameter 182.
<b>SG170</b>	smartEnergy hysteresis	Sets the distance between the lower and the upper threshold for stallGuard2 reading. Above the upper threshold the motor current becomes decreased.
<b>SG181</b>	stop on stall	Motor stop in case of stall.
<b>V182</b>	smartEnergy threshold speed	Above this speed coolStep becomes enabled.
<b>T214</b>	power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec).

## 6 Global Parameters

### GLOBAL PARAMETERS ARE GROUPED INTO 4 BANKS:

- 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.

### 6.1 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.

#### Attention:

- 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

Access Type	Related Command(s)	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable
E	STGP, RSGP	Parameter automatically restored from EEPROM after reset or power-on. These parameters can be stored permanently in EEPROM using STGP command and also explicitly restored (copied back from EEPROM into RAM) using RSGP.

Number	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!		
		<b>Warning:</b> The highest possible speed for RS232 is 115200 baud limited by the RS232 transceiver. The RS232 might work with higher speed but out of specification.				
66	Serial address	The module (target) address for RS232 / RS485.	0... 255	RWE		

Number	Parameter	Description	Range	Access																					
67	ASCII mode	Configure the TMCL ASCII interface: Bit 0: 0 – start up in binary (normal) mode 1 – start up in ASCII mode Bits 4 and 5: 00 – Echo back each character 01 – Echo back complete command 10 – Do not send echo, only send command reply		RWE																					
68	Serial heartbeat	Serial heartbeat for the RS485 interface. If this time limit is exceeded and no further command is noticed the motor will be stopped. 0 – parameter is disabled	[ms]	RWE																					
69	CAN bit rate	<table><tr><td>2</td><td>20kBit/s</td><td></td></tr><tr><td>3</td><td>50kBit/s</td><td></td></tr><tr><td>4</td><td>100kBit/s</td><td></td></tr><tr><td>5</td><td>125kBit/s</td><td></td></tr><tr><td>6</td><td>250kBit/s</td><td></td></tr><tr><td>7</td><td>500kBit/s</td><td></td></tr><tr><td>8</td><td>1000kBit/s</td><td>Default</td></tr></table>	2	20kBit/s		3	50kBit/s		4	100kBit/s		5	125kBit/s		6	250kBit/s		7	500kBit/s		8	1000kBit/s	Default	2... 8	RWE
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)	0... 7ff	RWE																					
71	CAN ID	The module (target) address for CAN (default: 1)	0... 7ff	RWE																					
73	Configuration EEPROM lock flag	Write: 1234 to lock the EEPROM, 4321 to unlock it. Read: 1=EEPROM locked, 0=EEPROM unlocked.	0/1	RWE																					
75	Telegram pause time	Pause time before the reply via RS232 or RS485 is sent. For RS232 set to 0. For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin). For CAN interface this parameter has no effect!	0... 255	RWE																					
76	Serial host address	Host address used in the reply telegrams sent back via RS232 / RS485.	0... 255	RWE																					
77	Auto start mode	0: Do not start TMCL application after power up (default). 1: Start TMCL application automatically after power up.	0/1	RWE																					
79	End switch polarity	0: normal polarity 1: reverse polarity	0/1	RWE																					
80	Shutdown pin functionality	Select the functionality of the SHUTDOWN pin 0 – no function 1 – high active 2 – low active	0... 2	RWE																					

Number	Parameter	Description	Range	Access
81	TMCL code protection	Protect a TMCL program against disassembling or overwriting. 0 – no protection 1 – protection against disassembling 2 – protection against overwriting 3 – protection against disassembling and overwriting <i>If you switch off the protection against disassembling, the program will be erased first!</i> <i>When changing this value from 1 or 3 to 0 or 2, the TMCL program will be erased.</i>	0,1,2,3	RWE
82	CAN heartbeat	Heartbeat for CAN interface. If this time limit is exceeded and no further command is noticed the motor will be stopped. 0 – parameter is disabled	[ms]	RWE
83	CAN secondary address	Second CAN ID for the module. Switched off when set to zero.	0... 7ff	RWE
85	Do not store user variables	0 – user variables are restored ( <i>default</i> ) 1 – user variables are not restored	0/1	RWE
87	Serial secondary address	Second module (target) address for RS232 / RS485.	0... 255	RWE
128	TMCL application status	0 – stop 1 – run 2 – step 3 – reset	0... 3	R
129	Download mode	0 – normal mode 1 – download mode	0/1	R
130	TMCL program counter	The index of the currently executed TMCL instruction.		R
132	Tick timer	A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value.	0... $2^{32}$	RW
133	Random number	Choose a random number.	0... 2147483647	RW

## 6.2 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 these variables form the interface between extensions of the firmware (written in C) and TMCL applications.



## 6.3 Bank 2

Bank 2 contains general purpose 32 bit variables for the use in TMCL applications. They are located in RAM and the first 56 variables can be stored permanently in EEPROM, also. After booting, their values are automatically restored to the RAM. Up to 256 user variables are available.

### MEANING OF THE LETTERS IN COLUMN ACCESS

Access Type	Related Command(s)	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable
E	STGP, RSGP	Parameter automatically restored from EEPROM after reset or power-on. These parameters can be stored permanently in EEPROM using STGP command and also explicitly restored (copied back from EEPROM into RAM) using RSGP.

### GENERAL PURPOSE VARIABLES FOR TMCL APPLICATIONS (BANK 2)

Number	Global parameter	Description	Range	Access
0... 55	general purpose variables #0... #55	for use in TMCL applications	$-2^{31} \dots +2^{31}$	RWE
56... 255	general purpose variables #56... #255	for use in TMCL applications	$-2^{31} \dots +2^{31}$	RW

## 6.4 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 parameter number defines the priority of an interrupt. Interrupts with a lower number have a higher priority.**

### MEANING OF THE LETTERS IN COLUMN ACCESS

Access type	Related command(s)	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable

### INTERRUPT PARAMETERS (BANK 3)

Number	Global parameter	Description	Range	Access
0	Timer 0 period (ms)	Time between two interrupts (ms)	0... 4.294.967.295 [ms]	RW
1	Timer 1 period (ms)	Time between two interrupts (ms)	0... 4.294.967.295 [ms]	RW
2	Timer 2 period (ms)	Time between two interrupts (ms)	0... 4.294.967.295 [ms]	RW
27	Stop left 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0... 3	RW
28	Stop right 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0... 3	RW
39	Input 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0... 3	RW
40	Input 1 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0... 3	RW

## 7 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.

### 7.1 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.

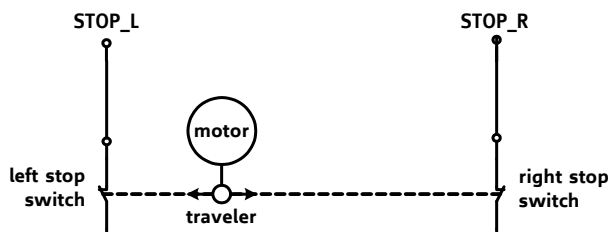


Figure 7.1 Left and right limit switches

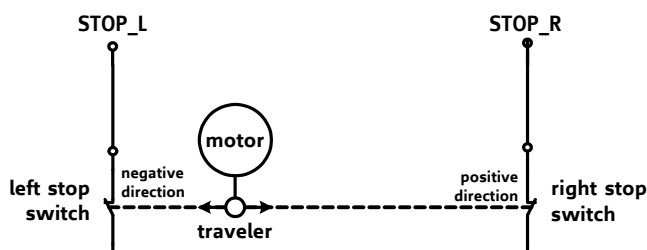


Figure 7.2 Limit switches and reference switch

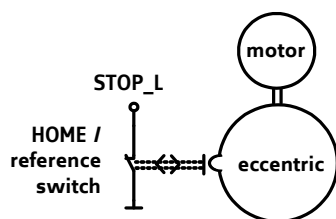


Figure 7.3 One reference switch

## 7.2 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 per rotation.

### FOR THE OPERATION WITH ENCODER CONSIDER THE FOLLOWING HINTS:

- The encoder counter can be read by software and can be used to control the exact position of the motor. This also makes closed loop operation possible.
- To read out or to change the position value of the encoder, axis parameter 209 is used.
- So, to read out the position of your encoder 0 use *GAP 209, 0*. The position values can also be changed using command *SAP 209, 0, <n>*, with  $n = \pm 0, 1, 2...$
- To change the encoder settings, axis parameter 210 is used. For changing the prescaler of the encoder 0 use *SAP 210, 0, <p>*.
- 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
102400	200	SAP 210, 0, 102400	8 (256 microsteps)
51200	100	SAP 210, 0, 51200	7 (128 microsteps)
25600	50	SAP 210, 0, 25600	6 (64 microsteps)
12800	25	SAP 210, 0, 12800	5 (32 microsteps)
6400 <i>default</i>	12.5	SAP 210, 0, 6400	4 (16 microsteps)
3200	6.25	SAP 210, 0, 3200	3 (8 microsteps)
1600	3.125	SAP 210, 0, 1600	2 (4 microsteps)
800	1.5625	SAP 210, 0, 800	1 (2 microsteps)

#### Note

- The table above shows a subset of prescalers that can be selected. Other values between those given in the table can be used.
- The values 1, 2, 4, and 16 must not be used for <p>.

Consider the following formula for your calculation:  $Prescaler = \frac{p}{512}$

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

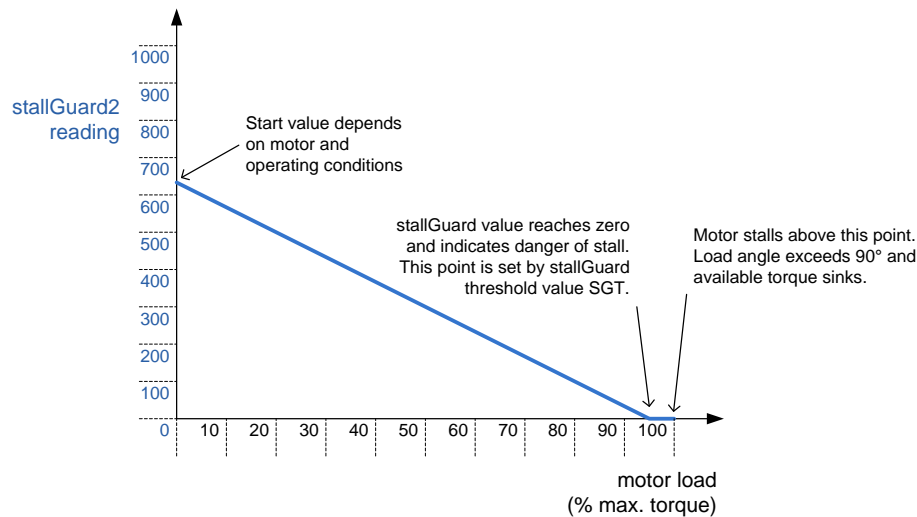
There is one special function that can also be configured using <p>. To select it just add the following value to <p>:

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.

### 7.3 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 7.4 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.***

### 7.4 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

## 8 TMCL Programming Techniques and Structure

### 8.1 Initialization

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

### 8.2 Main Loop

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

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

#### **MOST (BUT NOT ALL) STANDALONE TMCL PROGRAMS LOOK LIKE THIS:**

```
//Initialization
    SAP 4, 0, 500 //define max. positioning speed
    SAP 5, 0, 100 //define max. acceleration

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

### 8.3 Using Symbolic Constants

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

#### Example:

```
//Define some constants
#include TMCLParam.tmc
MaxSpeed = 500
MaxAcc = 100
Position0 = 0
Position1 = 5000

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

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

*Just have a look at the file [TMCLParam.tmc](#) provided with the TMCL-IDE. It contains symbolic constants that define all important parameter numbers.*

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

## 8.4 Using Variables

The *User Variables* can be used if variables are needed in your program. They can store temporary values. The commands SGP, GGP and AGP are used to work with user variables:

*SGP* is used to set a variable to a constant value (e.g. during initialization phase).

*GGP* is used to read the contents of a user variable and to copy it to the accumulator register for further usage.

*AGP* can be used to copy the contents of the accumulator register to a user variable, e.g. to store the result of a calculation.

### Example:

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

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

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

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

## 8.5 Using Subroutines

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

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

## 8.6 Mixing Direct Mode and Standalone Mode

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

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

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

### Example:

```
//Jump commands to the TMCL routines
```

```
Func1:      JA Func1Start
Func2:      JA Func2Start
Func3:      JA Func3Start
```

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

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

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

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

Please refer to the TMCL-IDE User Manual for further information about the TMCL-IDE.

## 9 Life Support Policy

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

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

© TRINAMIC Motion Control GmbH & Co. KG 2010-2014

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

Specifications are subject to change without notice.

All trademarks used are property of their respective owners.





## 10 Revision History

### 10.1 Firmware Revision

Version	Date	Description
4.26	2010-APR-26	First version supporting all TMCL features
4.27	2010-JUL-05	Firmware updates for other modules.
4.28	2010-AUG-09	RFS start resets deviation flags, too. Thus, a reference search is stopped if an encoder deviation is detected.
4.29-4.36	2011-DEC-01	Firmware updates for other modules.
4.37	2012-JAN-06	Axis parameter 200 (boost current) new.
4.38-4.40	2012-JUN-20	Firmware updates for other modules.
4.41	2012-SEP-21	Global parameter 87 new. Reference search: the last position before setting the counter to zero can be read out with axis parameter 197.
4.42	2012-NOV-20	<ul style="list-style-type: none"> <li>- Global parameter 82 (CAN heart beat) new</li> <li>- Global parameter 85 (do not store user variables) new</li> <li>- Axis parameter 254 (step/dir mode) new</li> <li>- Axis parameter 200 (boost current) new</li> <li>- Axis parameter 215 (absolute encoder value) new</li> </ul>
4.43	2013-FEB-20	Not deployed.
4.44	2013-OKT-15	Not deployed.
4.45	21.01.2014	Improved USB connection. Improved command <i>request target position reached</i> .

### 10.2 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 <i>Changing the Prescaler Value of an Encoder</i> 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.
1.08	2012-NOV-20	SD	Global parameter 65 updated.
1.09	2013-JAN-02	SD	<p>Chapter 8 new: TMCL programming techniques and structures. Changes related to TRINAMICs design.</p> <p>Global parameter list updated:</p> <ul style="list-style-type: none"> <li>- 87 (serial second address) new</li> <li>- 82 (CAN heart beat) new</li> <li>- 85 (do not store user variables) new</li> </ul> <p>Axis parameter list updated:</p> <ul style="list-style-type: none"> <li>- 254 (step/dir mode) new</li> <li>- encoder parameters updated: 209, 210, 212</li> <li>- 200 (boost current) new</li> <li>- 215 (absolute encoder value) new</li> <li>- unit for current parameters corrected</li> </ul> <p>SIO command updated</p>
1.10	2014-MAY-16	SD	Firmware revision updated.

## 11 References

[TMC262 / PD86-1180]  
[TMC262]  
[TMCL-IDE]  
[QSH8618]

TMC262 and PD86-1180 Hardware Manual  
TMC262 Datasheet  
TMCL-IDE User Manual  
QSH8618 Manual

Please refer to [www.trinamic.com](http://www.trinamic.com).