TMCM-1270 TMCL™ Firmware Manual

Firmware Version V1.00 | Document Revision V1.0 • 2017-Mar-02

The TMCM-1270 is an easy to use, single axis controller/driver module for 2-phase bipolar stepper motors with separate differential encoder and separate home and stop switch inputs. Dynamic current control, and quiet, smooth and efficient operation are combined with stealthChop™, dcStep™, stallGuard™and coolStep™features. Three digital inputs can be configured as home and endstop switches or incremental encoder inputs. A fourth digital input is available to enable or disable the motor.

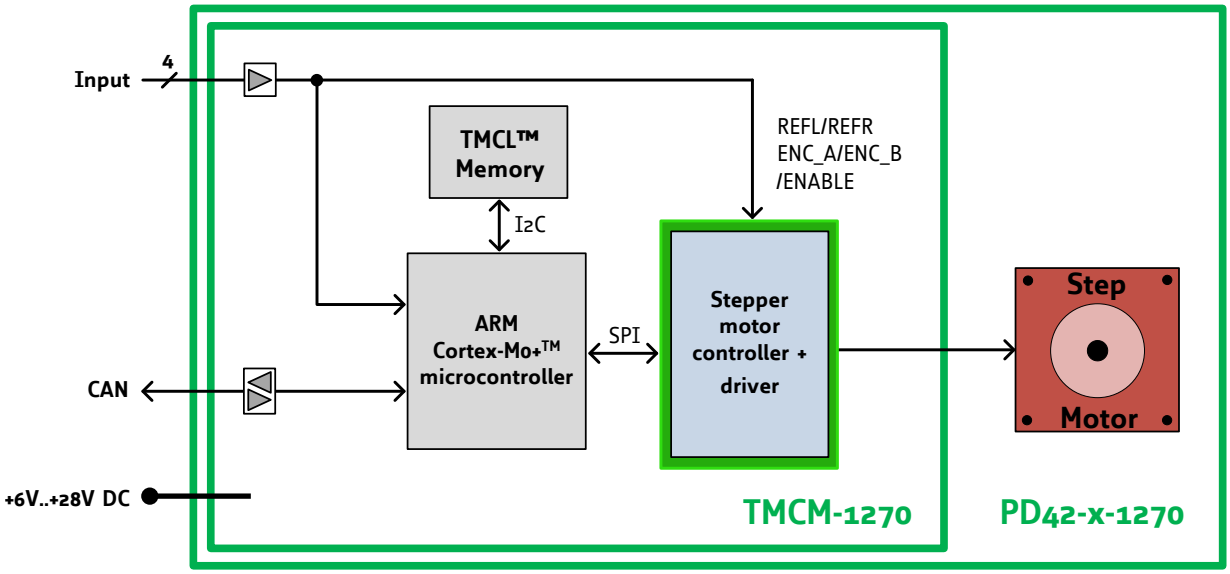
Features

- Single Axis Stepper motor control
- Supply voltage 24V DC
- TMCL™
- Motion Contoller
- CAN
- **dcStep™**
- Integrated **sixPoint™** ramp motion controller
- **stealthChop™** silent PWM mode
- **spreadCycle™** smart mixed decay
- **stallGuard2™** load detection
- **coolStep™** autom. current scaling

Applications

- Lab-Automation
- Semiconductor Handling
- Manufacturing
- Robotics
- Factory Automation
- CNC
- Laboratory Automation

Simplified Block Diagram



Read entire documentation.

TRINAMIC
MOTION CONTROL

Contents

1 Features

The TMCM-1270 is a single axis controller/driver module for 2-phase bipolar stepper motors with state of the art feature set. It is highly integrated, offers a convenient handling and can be used in many decentralized applications. The module has been designed for coil currents up to 1A RMS and 24V DC supply voltage. Three digital inputs can be configured as home and endstop switches or incremental encoder inputs. With its high energy efficiency from TRINAMIC's coolStep™ technology cost for power consumption is kept down. The TMCL firmware allows for both standalone and direct mode operation.

Main characteristics

- Motion controller & stepper motor driver:
  - Hardware motion profile calculation in real-time
  - On the fly alteration of motion parameters (e.g. position, velocity, acceleration)
  - High performance microcontroller for overall system control and communication protocol handling
  - 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
  - stealthChop™ feature for quiet operation and smooth motion
  - dcStep™ feature for load dependent speed control
- Interfaces
  - CAN
  - Low-active enable pin
  - Three I/O modes:
    - Incremental Encoder and Home Switch
    - Home and Endstop Switches
    - One analog and two digital inputs

Software

TMCL: remote controlled operation alone or during step/direction mode. PC-based application development software TMCL-IDE available for free.

Electrical data

- Supply voltage: +6V and +24V nominal.
- Motor current: up to 1A RMS / 1.41A peak (programmable).

Please see also the separate Hardware Manual.

1.1 stallGuard2

stallGuard2 is a high-precision sensorless load measurement using the back EMF of the coils. It can be used for stall detection as well as other uses at loads below those which stall the motor. The stallGuard2 measurement value changes linearly over a wide range of load, velocity, and current settings. At maximum motor load, the value reaches zero or is near zero. This is the most energy-efficient point of operation for the motor.

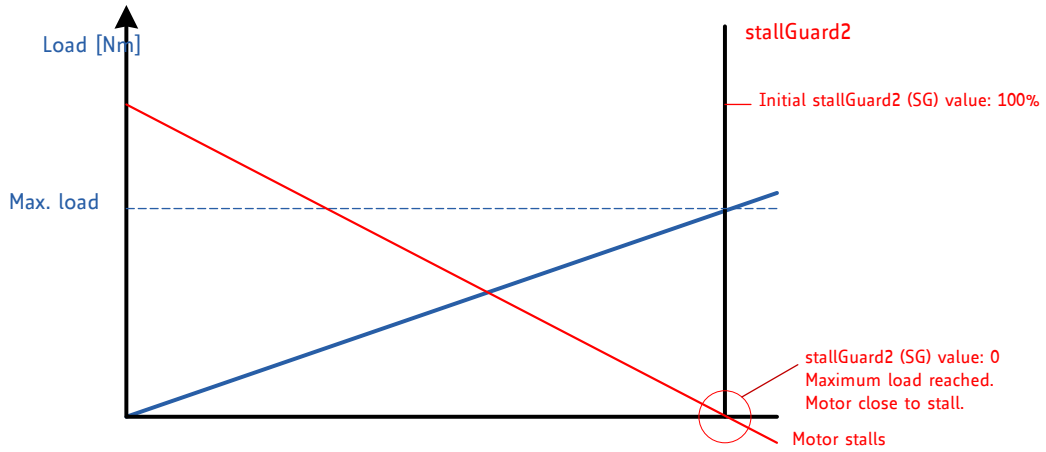


Figure 1: stallGuard2 Load Measurement as a Function of Load

1.2 coolStep

coolStep is a load-adaptive automatic current scaling based on the load measurement via stallGuard2 adapting the required current to the load. Energy consumption can be reduced by as much as 75%. coolStep allows substantial energy savings, especially for motors which see varying loads or operate at a high duty cycle. Because a stepper motor application needs to work with a torque reserve of 30% to 50%, even a constant-load application allows significant energy savings because coolStep automatically enables torque reserve when required. Reducing power consumption keeps the ystem cooler, increases motor life, and allows cost reduction.

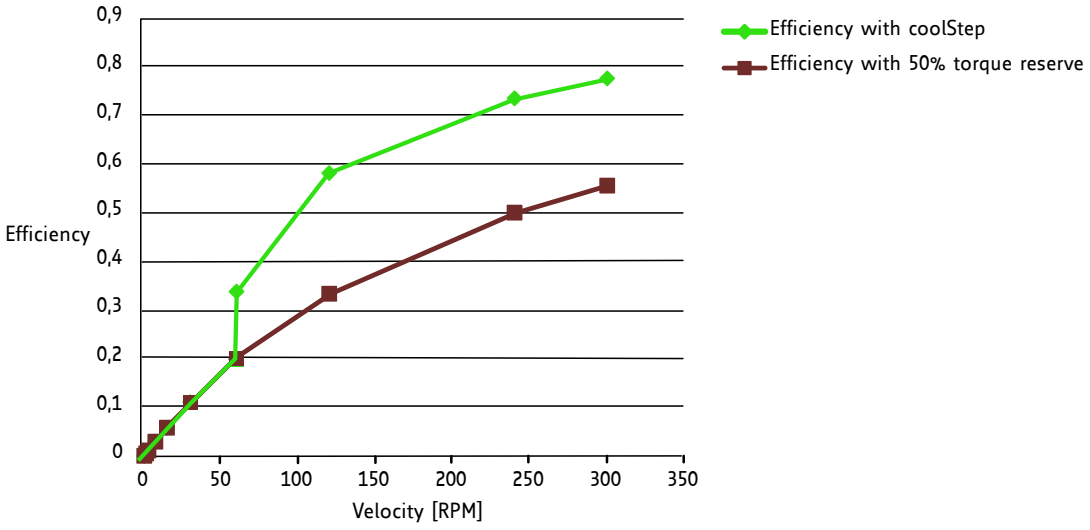


Figure 2: Energy Efficiency Example with coolStep

2 First Steps with TMCL

In this chapter you can find some hints for your first steps with the TMCM-1270 and TMCL. You may skip this chapter if you are already familiar with TMCL and the TMCL-IDE.

Things that you will need

- Your TMCM-1270 Module.
- A CAN adapter.
- A power supply (24V DC) for your TMCM-1270 module.
- The TMCL-IDE 3.x already installed on your PC
- A two-phase bipolar stepper motor.

2.1 Basic Setup

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

Please also ensure that your TMCM-1270 is properly connected to your power supply and that the stepper motor is properly connected to the module. Please see the TMCM-1270 hardware manual for instructions on how to do this. **Do not connect or disconnect a stepper motor to or from the module while the module is powered!**
Then, please start up the TMCL-IDE. After that you can connect your TMCM-1270 via CAN and switch on the power supply for the module (while the TMCL-IDE is running on your PC). The module will be recognized by the TMCL-IDE, and necessary driver registrations in Windows will automatically done by the TMCL-IDE.

2.2 Using the TMCL Direct Mode

At first try to use some TMCL commands in direct mode. In the TMCL-IDE a tree view showing the TMCM-1270 and all tools available for it is displayed. Click on the Direct Mode entry of the tool tree. Now, the Direct Mode tool will pop up.

In the Direct Mode tool you can choose a TMCL command, enter the necessary parameters and execute the command.

2.3 Changing Axis Parameters

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

Most important axis parameters					
Number	Axis Parameter	Description	Range [Units]	Default	Access
4	Maximum speed	The maximum speed used for positioning ramps.	0 ... 7999774 [pps]	150000	RWE

Number	Axis Parameter	Description	Range [Units]	Default	Access
5	Maximum acceleration	Maximum acceleration in positioning ramps. Acceleration and deceleration value in velocity mode.	0 ... 7629278 [pps/s]	1000	RWE
6	Maximum current	Motor current used when motor is running. The maximum value is 31 which means 100% of the maximum current of the module.	0 ... 31	24	RWE
7	Standby current	The current used when the motor is not running. The maximum value is 31 which means 100% of the maximum current of the module. This value should be as low as possible so that the motor can cool down when it is not moving.	0 ... 31	3	RWE
140	Microstep Resolution	Microstep resolutions per full step: 0 - fullstep 1 - halfstep 2 - 4 microsteps 3 - 8 microsteps 4 - 16 microsteps 5 - 32 microsteps 6 - 64 microsteps 7 - 128 microsteps 8 - 256 microsteps	0 ... 8	4	RWE
141	Microstep Interpolation	Interpolation of the current microstep resolution to 256 microsteps: 0 - No interpolation 1 - Interpolation to 256 microsteps	0 ... 1	1	RWE
179	VSense	Sense resistor voltage: 0 - High sense resistor voltage 1 - Low sense resistor voltage	0 ... 1	0	RWE

Table 1: Most important Axis Parameters

2.4 Testing with a simple TMCL Program

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

```
ROL 0, 51200              //Rotate motor 0 with speed 10000
WAIT TICKS, 0, 500
MST 0
ROR 0, 51200               //Rotate motor 0 with 50000
WAIT TICKS, 0, 500
MST 0

SAP 4, 0, 51200             //Set max. Velocity
SAP 5, 0, 51200             //Set max. Acceleration
```


```
Loop:
    MVP ABS, 0, 512000          //Move to Position 512000
    WAIT POS, 0, 0              //Wait until position reached
    MVP ABS, 0, -512000         //Move to Position -512000
    WAIT POS, 0, 0              //Wait until position reached
    JA Loop                     //Infinite Loop
```

After you have done that, take the following steps:

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

Also try out the debugging functions in the TMCL creator:

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

3 TMCL and the TMCL-IDE — An Introduction

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-1270 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 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 TMCM-1270 supports TMCL direct mode (binary commands). It also implements standalone TMCL program execution. This makes it possible to write TMCL programs using the TMCL-IDE and store them in the memory of the module.
In direct mode the TMCL communication over RS-232, RS-485, CAN and USB follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCM-1270. The TMCL interpreter on the module will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over the interface to the bus master. Only then should the master transfer the next command.
Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.
The Trinamic Motion Control Language [TMCL] provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM module to form programs that run standalone on the module. For this purpose there are not only motion control commands but also commands to control the program structure (like conditional jumps, compare and calculating).
Every command has a binary representation and a mnemonic. The binary format is used to send commands from the host to a module in direct mode, whereas the mnemonic format is used for easy usage of the commands when developing standalone TMCL applications using the TMCL-IDE (IDE means Integrated Development Environment).
There is also a set of configuration variables for the axis and for global parameters which allow individual configuration of nearly every function of a module. This manual gives a detailed description of all TMCL commands and their usage.

3.1 Binary Command Format

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes. When a command is to be sent via RS-232, RS-485, RS-422 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In these cases it consists of nine bytes.

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

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

Table 2: TMCL Command Format

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

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

3.1.1 Checksum Calculation

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

Checksum calculation in C:

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

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

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

Checksum calculation in Delphi:

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

    //Set the Command array to the desired command

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


3.2 Reply Format

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

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

Table 3: TMCL Reply Format

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

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

3.2.1 Status Codes

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

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

Table 4: TMCL Status Codes

3.3 Standalone Applications

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

3.4 TMCL Command Overview

This sections gives a short overview of all TMCL commands.

3.4.1 TMCL Commands

Overview of all TMCL Commands			
Command	Number	Parameter	Description
SAP	5	<parameter>, <motor number>, <value>	Set axis parameter (motion control specific settings)
GAP	6	<parameter>, <motor number>	Get axis parameter (read out motion control specific settings)
STAP	7	<parameter>, <motor number>, <value>	Store axis parameter (store motion control specific settings)
RSAP	8	<parameter>, <motor number>	Restore axis parameter (restore motion control specific settings)
SGP	9	<parameter>, <bank number>, <value>	Set global parameter (module specific settings e.g. communication settings or TMCL user variables)
GGP	10	<parameter>, <bank number>	Get global parameter (read out module specific settings e.g. communication settings or TMCL user variables)
STGP	11	<parameter>, <bank number>	Store global parameter (TMCL user variables only)
RSGP	12	<parameter>, <bank number>	Restore global parameter (TMCL user variables only)
RFS	13	<START|STOP|STATUS>, <motor number>	Reference search
GIO	15	<port number>, <bank number>	Get value of analog/digital input
CALC	19	<operation>, <value>	Process accumulator and value
COMP	20	<value>	Compare accumulator with value
JC	21	<condition>, <jump address>	Jump conditional
JA	22	<jump address>	Jump absolute
CSUB	23	<subroutine address>	Call subroutine
RSUB	24		Return from subroutine
WAIT	27	<condition>, <motor number>, <ticks>	Wait with further program execution
STOP	28		Stop program execution
SCO	30	<coordinate number>, <motor number>, <position>	Set coordinate
GCO	31	<coordinate number>, <motor number>	Get coordinate

Command	Number	Parameter	Description
CCO	32	<coordinate number>, <motor number>	Capture coordinate
CALCX	33	<operation>	Process accumulator and X-register
AAP	34	<parameter>, <motor number>	Accumulator to axis parameter
AGP	35	<parameter>, <bank number>	Accumulator to global parameter
CLE	36	<flag>	Clear an error flag
ACO	39	<coordinate number>, <motor number>	Accu to coordinate

Table 5: Overview of all TMCL Commands

3.5 TMCL Commands by Subject

3.5.1 Parameter Commands

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

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

Table 6: Parameter Commands

3.5.2 Branch Commands

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

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

Table 7: Branch Commands

3.5.3 I/O Port Commands

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

I/O Port Commands		
Mnemonic	Command number	Meaning
GIO	15	Get input

Table 8: I/O Port Commands

3.5.4 Calculation Commands

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

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

Table 9: Calculation Commands

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

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

3.6 Detailed TMCL Command Descriptions

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

3.6.1 SAP (Set Axis Parameter)

With this command most of the motion control parameters of the module can be specified. The settings will be stored in SRAM and therefore are volatile. That is, information will be lost after power off.

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

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

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

Binary representation

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

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

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

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

3.6.2 GAP (Get Axis Parameter)

Most motion / driver related parameters of the TMCM-1270 can be adjusted using e.g. the SAP command. With the GAP parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditional jumps). In direct mode the value read is only output in the value field of the reply, without affecting the accumulator.

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

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

Mnemonic: GAP <parameter number>, <axis>

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

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

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

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

3.6.3 STAP (Store Axis Parameter)

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

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

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

Related commands: SAP, AAP, GAP, RSAP.

Mnemonic: STAP <parameter number>, <bank>

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

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

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

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

3.6.4 RSAP (Restore Axis Parameter)

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

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

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

Related commands: SAP, AAP, GAP, RSAP.

Mnemonic: RSAP <parameter number>, <bank>

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

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

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

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

3.6.5 SGP (Set Global Parameter)

With this command most of the module specific parameters not directly related to motion control can be specified and the TMCL user variables can be changed. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in banks to allow a larger total number for future products. Currently, bank 0 is used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.
All module settings in bank 0 will automatically be stored in non-volatile memory (EEPROM).

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

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

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

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

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

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

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

3.6.6 GGP (Get Global Parameter)

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

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

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

Mnemonic: GGP <parameter number>, <bank>

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

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

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

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

3.6.7 STGP (Store Global Parameter)

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

Info	For a table with parameters and values which can be used together with this command please refer to dection 5.2.

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

Related commands: SGP, AGP, GGP, RSGP.

Mnemonic: STGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
11	see chapter 5.2	2	0 (don't care)

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

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

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

3.6.8 RSGP (Restore Global Parameter)

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

Info For a table with parameters and values which can be used together with this command please refer to section 5.2.

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

Related commands: SGP, AGP, GGP, STGP.

Mnemonic: RSGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
12	see chapter 5.2	2	0 (don't care)

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

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

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

3.6.9 RFS (Reference Search)

The TMCM-1270 has a built-in reference search algorithm. The reference search algorithm provides different refrence search modes. This command starts or stops the built-in reference search algorithm. The status of the reference search can also be queried to see if it already has finished. (In a TMCL program it mostly is better to use the WAIT RFS 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 (please see chapter 4).

Internal function: The internal reference search state machine is started or stoped, or its state is queried.

Related commands: SAP, GAP, WAIT.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
13	0 START — start reference search 1 STOP — stop reference search 2 STATUS — get status	0	0 (don't care)

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

Reply in Direct Mode (RFS STATUS)		
Status	Value	
100 - OK	0	no ref. search active
	other values	reference search active

Example
Start reference search of motor 0.
Mnemonic: RFS START, 0.

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

3.6.10 GIO (Get Input)

With this command the status of the available general purpose outputs of the module can be read. The function reads a digital or an analog input port. Digital lines will read as 0 or 1, while the ADC channels deliver their 12 bit result in the range of 0...4095. In standalone mode the requested value is copied to the accumulator register for further processing purposes such as conditional jumps. In direct mode the value is only output in the value field of the reply, without affecting the accumulator. The actual status of a digital output line can also be read.

Internal function: The state of the i/o line specified by the type parameter and the bank parameter is read.

Related commands: SIO.

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

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

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

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

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

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

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

Digital Inputs in Bank 0			
Port	Description	Command	Range
0 - HOME	Home Input Switch (when not AIN1)	GIO 0, 0	0/1
1 - REFL_ENCA	Reference Left Switch / Encoder A Input	GIO 1, 0	0/1
2 - REFR_ENCB	Reference Right Switch / Encoder B Input	GIO 2, 0	0/1
3 - Enable	TMC5130 Enable Pin	GIO 3, 0	0/1
4 - DIAG0	TMC5130 Diag 0 Output	GIO 4, 0	0/1
5 - DIAG1	TMC5130 Diag 1 Output	GIO 5, 0	0/1

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

Analog Inputs in Bank 1			
Port	Description	Command	Range
0 - VSupply	Supply Voltage Measurement	GIO 0, 1	0...4095
1 - AIN1	Analog Input 1 (when not HOME)	GIO 1, 1	0...4095

3.6.11 CALC (Calculate)

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

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

Mnemonic: CALC <operation>, <operand>

Binary representation

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

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

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

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

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

3.6.12 COMP (Compare)

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

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

Mnemonic: COMP <operand>

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

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

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

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

3.6.13 JC (Jump conditional)

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

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

Related commands: JA, COMP, WAIT, CLE.

Mnemonic: JC <condition>, <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
21	0 ZE - zero 1 NZ - not zero 2 EQ - equal 3 NE - not equal 4 GT - greater 5 GE - greater/equal 6 LT - lower 7 LE - lower/equal 8 ETO - time out error 9 EAL - external alarm 10 EDV - deviation error 11 EPO - position error	0 (don't care)	<jump address>

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

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

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

3.6.14 JA (Jump always)

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

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

Related commands: JC, WAIT, CSUB.

Mnemonic: JA <label>

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

Example
An infinite loop in TMCL:

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

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

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

3.6.15 CSUB (Call Subroutine)

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

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

Related commands: RSUB, JA.

Mnemonic: CSUB <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
23	0 (don't care)	0 (don't care)	<subroutine address>

Example
Call a subroutine:

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

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

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

3.6.16 RSUB (Return from Subroutine)

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

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

Related commands: CSUB.

Mnemonic: RSUB

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

Example
Please see the CSUB example (section 3.6.15).
Binary form:

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

3.6.17 WAIT (Wait for an Event to occur)

This instruction interrupts the execution of the TMCL program until the specified condition is met. *This command is intended for standalone operation only.*
There are five different wait conditions that can be used:

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

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

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

Internal function: the TMCL program counter will be held at the address of this WAIT command until the condition is met or the timeout has expired.

Related commands: JC, CLE.

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

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

Example

[1] one tick is 10 milliseconds

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

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

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

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

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

Related commands: none.

Mnemonic: STOP

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

Example
Mnemonic: STOP

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

3.6.19 SCO (Set Coordinate)

Up to 20 position values (coordinates) can be stored for every axis for use with the MVP COORD command. This command sets a coordinate to a specified value. Depending on the global parameter 84, the coordinates are only stored in RAM or also stored in the EEPROM and copied back on startup (with the default setting the coordinates are stored in RAM only).

Note Coordinate #0 is always stored in RAM only.

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

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

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

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

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

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

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

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

3.6.20 GCO (Get Coordinate)

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

Note	Coordinate #0 is always stored in RAM only.

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

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

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

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

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

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

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

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

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

3.6.21 CCO (Capture Coordinate)

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

Note Coordinate #0 is always stored in RAM only.

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

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

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

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

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

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

Mnemonic: CCO 3, 0

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

3.6.22 ACO (Accu to Coordinate)

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

Note Coordinate #0 is always stored in RAM only.

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

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

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

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

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

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

Mnemonic: ACO 1, 0

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

3.6.23 CALCX (Calculate using the X Register)

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

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

Mnemonic: CALCX <operation>

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

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

Example
Multiply accumulator and X register.
Mnemonic: CALCX MUL

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

3.6.24 AAP (Accu to Axis Parameter)

The content of the accumulator register is transferred to the specified axis parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction. *This command is mainly intended for use in standalone mode.*

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

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

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

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

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

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

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

Binary Form of AAP 0, 0	
Field	Value
Target address	01_h
Instruction number	22_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	23_h

3.6.25 AGP (Accu to Global Parameter)

The content of the accumulator register is transferred to the specified global parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction. *This command is mainly intended for use in standalone mode.*

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

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

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

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

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

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

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

3.6.26 CLE (Clear Error Flags)

This command clears the internal error flags. It is mainly intended for use in standalone mode.
The following error flags can be cleared by this command (determined by the <flag> parameter):

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

Related commands: JC, WAIT.

Mnemonic: CLE <flags>

Binary Representation			
Instruction	Type	Motor/Bank	Value
36	0 ALL – all flags 1 – (ETO) timeout flag 2 – (EAL) alarm flag 3 – (EDV) deviation flag 4 – (EPO) position flag 5 – (ESD) shutdown flag	0 (don't care)	0 (don't care)

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

Example
Reset the timeout flag.
Mnemonic: CLE ETO

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

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

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

Related commands: none.

Mnemonic: UF0...UF7

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

Reply in Direct Mode	
Status	Value
100 - OK	user defined

3.6.28 Request Target Position reached Event

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

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

Related commands: none.

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

With command 138 the value field is a bit vector. It shows for which motors one would like to have a position reached message. The value field contains a bit mask where every bit stands for one motor. For one motor modules like the TMCM-1270 it only makes sense to have bit 0 set. So, always set this parameter to 1 with the TMCM-1270 module. With the type field set to 0, only for the next MVP command that follows this command a position reached message will be generated. With type set to 1 a position reached message will be generated for every MVP command that follows this command. It is recommended to use the latter option.

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

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

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

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

3.6.29 TMCL Control Commands

There is a set of TMCL commands which are called TMCL control commands. These commands can only be used in direct mode and not in a standalone program. For this reason they only have opcodes, but no mnemonics. Most of these commands are only used by the TMCL-IDE (in order to implement e.g. the debugging functions in the TMCL creator). Some of them are also interesting for use in custom host applications, for example to start a TMCL routine on a module, when combining direct mode and standalone mode (please see also section 8.6. The following table lists all TMCL control commands.

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

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

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

Table 10: TMCL Control Commands

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

4 Axis Parameters

Most motor controller features of the TMCM-1270 module are controlled by axis parameters. Axis parameters can be modified or read using SAP, GAP, AAP, STAP and RSAP commands. This chapter describes all axis parameters that can be used on the TMCM-1270 module.

Axis 0 Parameters of the TMCM-1270 Module					
Number	Axis Parameter	Description	Range [Units]	Default	Access
0	Target position	The desired target position in position mode.	−2147483648 ... 2147483647 [usteps]	0	RW
1	Actual position	The actual position of the motor. Stop the motor before overwriting it. Should normally only be overwritten for reference position setting.	−2147483648 ... 2147483647 [usteps]	0	RW
2	Target speed	The desired speed in velocity mode. Not valid in position mode.	−4999999 ... 4999999 [pps]	0	RW
3	Actual speed	The actual speed of the motor.	−4999999 ... 4999999 [pps]	0	R
4	Maximum speed	The maximum speed used for positioning ramps.	0 ... 4999999 [pps]	51200	RWE
5	Maximum acceleration	Maximum acceleration in positioning ramps. Acceleration and deceleration value in velocity mode.	0 ... 2980186 [pps/s]	51200	RWE
6	Maximum current	Motor current used when motor is running. The maximum value is 31 which means 100% of the maximum current of the module.	0 ... 31	24	RWE
7	Standby current	The current used when the motor is not running. The maximum value is 31 which means 100% of the maximum current of the module. This value should be as low as possible so that the motor can cool down when it is not moving. Please see also parameter 214.	0 ... 31	3	RWE
8	Position reached flag	This flag is always set when target position and actual position are equal. 0 - Target position not reached 1 - Target position reached	0 ... 1	0	R
9	Home Switch	State of the home switch. 0 - Inactive 1 - Active	0 ... 1	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
10	Right Endstop	Reference switch right status. 0 - Inactive 1 - Active	0 ... 1	0	R
11	Left Endstop	Reference switch left status. 0 - Inactive 1 - Active	0 ... 1	0	R
12	Automatic Right Stop	Enables automatic motor stop during active right reference switch input. 0 - Inactive 1 - Active	0 ... 1	0	RW
13	Automatic Left Stop	Enables automatic motor stop during active left reference switch input. 0 - Inactive 1 - Active	0 ... 1	0	RW
14	Swap Switch Inputs	Swap the left and the right reference switch input. 0 - Inactive 1 - Active	0 ... 1	0	R
15	Acceleration A1	First acceleration between VSTART and V1 (in position mode only).	0 ... 2980186 [pps/s]	100	RW
16	Velocity V1	First acceleration / deceleration phase target velocity (in position mode only). Setting this value to 0 turns off the first acceleration / deceleration phase, maximum acceleration (axis parameter 5) and maximum deceleration (axis parameter 17) are used only.	0 ... 1000000 [pps]	2	RW
17	Maximum Deceleration	Maximum deceleration in positioning ramps. Used to decelerate from maximum positioning speed (axis parameter 4) to velocity V1	0 ... 2980186 [pps/s]	5	RW
18	Deceleration D1	Deceleration between V1 and VSTOP (in positioning mode only).	0 ... 2980186 [pps/s]	200	RW
19	Velocity VSTART	Motor start velocity (in position mode only). Do not set VSTART higher than VSTOP.	0 ... 249999 [pps]	1	RW
20	Velocity VSTOP	Motor stop velocity (in position mode only).	0 ... 249999 [pps]	2	RW
21	Ramp wait time	Defines the waiting time after ramping down to zero velocity before next movement or direction inversion can start. Time range is 0 to 2 seconds. This setting avoids excess acceleration e.g. from VSTOP to -VSTART.	0 ... 65535 [0.000032s]	0	RW

Number	Axis Parameter	Description	Range [Units]	Default	Access
22	Speed threshold for high speed mode	Speed threshold for de-activating coolStep or switching to fullstep mode.	0 ... 4999999 [pps]	0	RW
23	Minimum speed for switching to dcStep	Minimum speed for switching to dcStep.	0 ... 4999999 [pps]	0	RW
24	Right Switch Polarity	Sets the active polarity of the right reference switch input: 0 - non-inverted, high active: a high level on REFR stops the motor 1 - inverted, low active: a low level on REFR stops the motor	0 ... 1	0	RW
25	Left Switch Polarity	Sets the active polarity of the left reference switch input: 0 - non-inverted, high active: a high level on REFL stops the motor 1 - inverted, low active: a low level on REFL stops the motor	0 ... 1	0	RW
26	Soft Stop	The soft stop mode always uses the deceleration ramp settings DMAX, V1, D1, VSTOP and TZEROWAIT for stopping the motor. 0 - Hard stop 1 - Soft stop	0 ... 1	0	RW
27	High speed chopper mode	Switch to other chopper mode when measured speed is higher than axis parameter 22 when set to 1. 0 - Inactive 1 - Active	0 ... 1	0	RW
28	High speed fullstep mode	Switch to fullstep mode when measured speed is higher than axis parameter 22 when set to 1. 0 - Inactive 1 - Active	0 ... 1	0	RW
29	Measured Speed	Current speed value measured from the TMC5130 current position or from encoder.	−2147483648 ... 2147483647	0	R
31	Power down ramp	Controls the number of clock cycles for motor power down after a motion as soon as the motor has stopped and the setting time has expired. The smooth transition avoids a motor jerk upon power down. 0=instant power down, 15=longest possible power down ramp.	0 ... 15 [0.16384s]	0	RWE

Number	Axis Parameter	Description	Range [Units]	Default	Access
40	Supply voltage	The actual supply voltage.	0 ... 1000 [100mV]	240	R
127	Relative Positioning Option Code	The command MoveToRelativePosition uses as referene the following position: 0 - the target position 1 - the actual position 2 - the encoder position	0 ... 2	1	RW
140	Microstep Resolution	Microstep resolutions per full step: 0 - fullstep 1 - halfstep 2 - 4 microsteps 3 - 8 microsteps 4 - 16 microsteps 5 - 32 microsteps 6 - 64 microsteps 7 - 128 microsteps 8 - 256 microsteps	0 ... 8	4	RW
150	debug value 0	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
151	debug value 1	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
152	debug value 2	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
153	debug value 3	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
154	debug value 4	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
155	debug value 5	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
156	debug value 6	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
157	debug value 7	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
158	debug value 8	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
159	debug value 9	Freely usable debugging value.	−2147483648 ... 2147483647	0	RW
162	Chopper blank time	Selects the comparator blank time. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. Normally leave at the default value.	0 ... 3	0	RW

Number	Axis Parameter	Description	Range [Units]	Default	Access
163	Constant TOff Mode	Selection of the chopper mode. 0 - spread cycle 1 - classic constant off time	0 ... 1	0	RW
164	Disable fast decay comparator	See parameter 163. For "classic const. off time", setting this parameter to "1" will disable current comparator usage for termination of fast decay cycle. 0 - enable comparator 1 - disable comparator	0 ... 1	0	RW
165	Chopper hysteresis end / fast decay time	See parameter 163. For "spread cycle" chopper mode this parameter will set / return the hysteresis end setting (hysteresis end value after a number of decrements). For "classic const. off time" chopper mode this parameter will set / return the fast decay time.	0 ... 15	0	RW
166	Chopper hysteresis start / sine wave offset	See parameter 163. For "spread cycle" chopper mode this parameter will set / return the Hysteresis start setting (please note that this value is an offset to the hysteresis end value). For "classic const. off time" chopper mode this parameter will set / return the sine wave offset.	0 ... 8	0	RW
167	Chopper off time	The off time setting controls the minimum chopper frequency. An off time within the range of 5us to 20us will fit. Off time setting for constant t Off chopper: NCLK = 12 + 32 tOFF (Minimum is 64 clocks) Setting this parameter to zero completely disables all driver transistors and the motor can freewheel.	0 ... 15	0	RW
168	smartEnergy current minimum (SEIMIN)	Sets the lower motor current limit for coolStep operation by scaling the maximum current (see axis parameter 6) value. 0 - 1/2 of CS 1 - 1/4 of CS	0 ... 1	0	RW
169	smartEnergy current down step	Sets the number of stallGuard2 readings above the upper threshold necessary for each current decrement of the motor current. Number of stallGuard2 measurements per decrement: Scaling: 0. . . 3: 32, 8, 2, 1. 0: slow decrement, 3: fast decrement	0 ... 3	0	RW

Number	Axis Parameter	Description	Range [Units]	Default	Access
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. Hysteresis: ([AP172] + 1) * 32. Upper stallGuard threshold: ([AP172] + [AP170] + 1) * 32	0 ... 15	0	RW
171	smartEnergy current up step	Sets the current increment step. The current becomes incremented for each measured stallGuard2 value below the lower threshold see smartEnergy hysteresis start). Current increment step size: Scaling: 0. . . 3: 1, 2, 4, 8. 0: slow increment, 3: fast increment / fast reaction to rising load	0 ... 3	0	RW
172	smartEnergy hysteresis start	The lower threshold for the stallGuard2 value (see smart Energy current up step).	0 ... 15	0	RW
173	stallGuard2 filter enable	Enables the stallGuard2 filter for more precision of the movement. If set, reduces the measurement frequency to one measurement per four fullsteps. In most cases it is expedient to set the filtered mode before using coolStep. Use the standard mode for step loss detection. 0 - standard mode 1 - filtered mode	0 ... 1	0	RW
174	stallGuard2 threshold	This signed value controls stallGuard2 threshold level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2 less sensitive and requires more torque to indicate a stall.	−64 ... 63	0	RW
177	Short to Ground Protection	Disable short to ground protection. 0 - Short to GND protection is on 1 - Short to GND protection is disabled	0 ... 1	0	RW
179	VSense	Sense resistor voltage: 0 - High sense resistor voltage 1 - Low sense resistor voltage	0 ... 1	0	RWE

Number	Axis Parameter	Description	Range [Units]	Default	Access
180	smartEnergy actual current	This status value provides the actual motor current setting as controlled by coolStep. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN. Actual motor current scaling factor: 0. . . 31: 1/32, 2/32, . . . 32/32	0 . . . 31	0	R
181	smartEnergy stall velocity	Velocity from which stop on stall feature is active	0 . . . 4999999	0	RW
182	smartEnergy threshold speed	Above this speed coolStep becomes enabled.	0 . . . 4999999 [pps]	0	RW
184	Random TOff mode	Enable / disable random TOff mode. 0 - Chopper off time is fixed 1 - Chopper off time is random	0 . . . 1	0	RW
185	Chopper synchronization	This parameter allows synchronization of the chopper for both phases of a two phase motor in order to avoid the occurrence of a beat, especially at low velocities. 0: chopper sync function chopSync off 1. . . 15: chopper synchronization	0 . . . 15	0	RW
186	PWM threshold speed	The stealthChop feature will be switched on when this parameter is >= 0 and the actual velocity is higher than this value.	0 . . . 4999999 [pps]	0	RW
187	PWM gradient	Velocity dependent gradient for PWM amplitude (stealthChop). Setting this value to 0 turns off stealthChop.	0 . . . 15	0	RW
188	PWM amplitude	Maximum PWM amplitude when switching to stealthChop mode. Do not set too low. Values above 64 recommended.	0 . . . 255	0	RW
189	PWM scale	Actual PWM amplitude scaler (255=max. Voltage). In voltage mode PWM, this value allows to detect a motor stall.	0 . . . 255	0	R
190	PWM Mode	Status of StealthChop Voltage PWM Mode (depending on velocity thresholds). 0 - StealthChop disabled 1 - StealthChop enabled	0 . . . 1	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
191	PWM frequency	PWM frequency selection fro StealthChop. 0 - $f_{PWM} = 1/1024 f_{CLK}$ 1 - $f_{PWM} = 1/683 f_{CLK}$ 2 - $f_{PWM} = 1/512 f_{CLK}$ 3 - $f_{PWM} = 1/410 f_{CLK}$	0 ... 3	0	RW
192	PWM autoscale	PWM automatic amplitude scaling for StealthChop. 0 - User defined PWM amplitude. The current settings have no influence 1 - enable automatic current control	0 ... 1	0	RW
193	Ref Mode	Selects the reference search mode. Add 128 to a mode value for inverting the home switch (can be used with mode 5...8). Add 64 to a mode for driving the right instead of the left reference switch (can be used with modes 1-4). 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. 4 - Search left stop switch from both sides. 5 - Search home switch in negative direction, reverse the direction when left stop switch reached. 6 - Search home switch in positive direction, reverse the direction when right stop switch reached. 7 - Search home switch in positive direction, ignore end switches. 8 - Search home switch in negative direction, ignore end switches.	1 ... 8, 65 ... 72, 129 ... 136 and 193 ... 200	1	RWE
194	RefSearchSpeed	This value specifies the speed for roughly searching the reference switch.	0 ... 4999999 [pps]	50000	RWE
195	RefSwitchSpeed	This parameter specifies the speed for searching the switching point. It should be slower than parameter 194.	0 ... 4999999 [pps]	5000	RWE
196	Right Limit Switch Position	Position of the right limit switch when working in Reference Mode (AP193) 2 and 3.	−2147483648 ... 2147483647	0	R
197	Last Reference Position	Value of the last position before the reset when referencing.	−2147483648 ... 2147483647	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
201	Encoder mode	Encoder mode configuration. See bits 0-9 of TMC5130 ENCMODE register.	0 ... 1023	0	RW
202	Motor Fullstep Resolution	Number of fullsteps of the motor.	0 ... 4294967295 [fullsteps]	200	RWE
203	PWM symmetric	Force symmetric PWM for StealthChop. 0 - The PWM value may change within each PWM cycle (standard mode) 1 - A symmetric PWM cycle is enforced	0 ... 1	0	RW
204	Freewheeling mode	Stand still option when the standby current (parameter 7) is set to zero. 0 - normal operation 1 - freewheeling 2 - coil shorted using low side drivers 3 - coil shorted using high side drivers	0 ... 3	0	RWE
206	Load value	Actual current control scaling for monitoring smart energy current scaling or automatic current scaling.	0 ... 1023	0	R
207	Extended error flags	A combination of the following values: 1 - stallGuard error, 2 - deviation error. These error flags are cleared automatically when this parameter has been read out or when a motion command has been executed.	0 ... 3	0	R
208	Drive Status Flags	TMC5130 flags of register DRVSTATUS.	0 ... 255	0	R
209	Encoder position	The current encoder position	−2147483648 ... 2147483647 [usteps]	0	RW
210	Encoder Resolution	Resolution of the encoder.	0 ... 65535	32768	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]	0	RWE
214	Power down delay	Standstill period before the current will be ramped down to standby current. The standard value is 200 (which means 2000ms).	0 ... 417 [10ms]	0	RW
254	Module Specific Mode	Choose between pin configurations for GPI1, GPI2 and GPI3. 0 - MODE_REF: Home, RefL and RefR 1 - MODE_ENC: Home, EncA and EncB 2 - MODE_GPI: AIN1, DIN2 and DIN3	0 ... 2	0	RWE

Number	Axis Parameter	Description	Range [Units]	Default	Access
255	Unit mode	Units of velocity and acceleration/deceleration settings 0 - internal units of TMC5130 1 - pps for velocity and pps/s for acceleration	0 . . . 1	1	RW

Table 11: All TMCM-1270 Axis 0 Parameters

5 Global Parameters

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

- Bank 0: Global configuration of the module.
- Bank 2: TMCL user variables.

5.1 Bank 0

Parameters with numbers from 64 on configure stuff like the serial address of the module 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 automatically stored in the EEPROM.

Note

- An SGP command on such a parameter will always store it permanently and no extra STGP command is needed.
- Take care when changing these parameters, and use the appropriate functions of the TMCL-IDE to do it in an interactive way!

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

Table 12: Meaning of the Letters in the Access Column

All Global Parameters of the TMCM-1270 Module in Bank 0					
Number	Global Parameter	Description	Range [Units]	Default	Access
69	CAN Bitrate	After next restart, the module will be configured with one of the following bitrates: 2 - 20 Kbps 3 - 50 Kbps 4 - 100 Kbps 5 - 125 Kbps 6 - 250 Kbps 7 - 500 Kbps 8 - 1 Mbps	2 ... 8	8	RWA
70	CAN Send Id	CAN Id used for TMCL replies.	0 ... 255	2	RWA
71	CAN Receive Id	CAN Messages with this target Id will be received and interpreted.	0 ... 255	1	RWA

Number	Global Parameter	Description	Range [Units]	Default	Access
72	CAN Secondary Id	CAN Messages with this target Id will also be received and interpreted. If the value is 0, this functionality will not be used.	0 ... 255	0	RWA
77	Auto start mode	Use automatic TMCL application start after power up. 0 - do not start TMCL application after power up 1 - start TMCL application automatically after power up	0 ... 1	1	RWA
81	TMCL Protection Mode	TMCL Storage will be deleted if the read protection bit is reset. 0 - not reset 1 - TMCL Storage will be deleted	0 ... 1	0	RWA
84	EEPROM Coordinate Store	If true, TMCL Coordinates will be stored in the EEPROM. 0 - Coordinates not stored 1 - coordinates stored	0 ... 1	0	RWA
85	Zero User Variables	0 - user variables are restored 1 - user variables are not restored	0 ... 1	0	RWA
128	Application status	Actual TMCL application status. 0 - stop 1 - run 2 - step 3 - reset	0 ... 3	0	R
130	Program counter	TMCL program counter.	0 ... 4294967295	0	R
131	Last error	TMCL Last Error.	0 ... 4294967295	0	RW
132	Tick timer	TMCL tick timer.	0 ... 4294967295	0	RW
133	RandomNumber	TMCL Random Number.	0 ... 2147483647	0	RW

Table 13: All Global Parameters of the TMCM-1270 Module in Bank 0

5.2 Bank 2

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

Meaning of the letters in the *Access* column		
Access type	Command	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable
E	STGP, RSGP	Parameter can be stored in the EEPROM

Table 14: Meaning of the Letters in the Access Column

User Variables in Bank 2				
Number	Global Parameter	Description	Range [Units]	Access
0...55	user variables #0...#55	TMCL user variables	-2147483648 ... 2147483647	RWE

Table 15: User Variables in Bank 2

6 Module Specific Modes

This section explains how the three availabe inputs can be configured. The TMCM-1270 implements three different predefined modes, each one with a different set of functions for these three pins. Axis Parameter number 254 “Module Specific Mode” (see Section 4) contains the current mode. In Table 16, the functionalities for all three pins in each configuration are shown.

Pin Functionalities of TMCM-1270 Specific Modes				
Mode	Id	GPI1	GPI2	GPI3
Mode Reference	0	Home Switch	Left Endstop Switch	Right Endstop Switch
Mode Encoder	1	Home Switch	Encoder A	Encoder B
Mode GPI	2	Analog Input 1	Digital Input 2	Digital Input 3

Table 16: Pin Functionalities of TMCM-1270 Specific Modes

In Mode Reference, the three digital inputs correspond to reference switches: Home, Left Endstop and Rigth Endstop for GPI 1, 2 and 3 respectively. The Reference Search feature is only available in this mode. See more information about Reference Seach in Section 7.1.

In Mode Encoder, GPI2 and 3 correspond to the encoder pins A and B.

Finally, all three inputs are function free inputs available for the user in Mode GPI.

All these inputs are read with a Get Input TMCL Command with Bank 0 (Digital Inputs). The only exception is GPI1 in Mode GPI (Analog Input 1), which is read with Bank 1 (analog inputs). See more information about how to read the state of an input in Section 3.6.10.

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 search algorithm or using an incremental encoder. You will also find basic information about stallGuard2™ and coolStep™ in this chapter.

7.1 Reference Search

The built-in reference search features switching point calibration and support for a home switch and/or one or two end switches. The internal operation is based on a state machine that can be started, stopped and monitored (instruction RFS, opcode 13). The settings of the automatic stop functions corresponding to the end switches (axis parameters 12 and 13) do not influence the reference search.

Notes:

- Until the reference switch is found for the first time, the searching speed set by axis parameter 194 is used.
- After hitting the reference switch, the motor slowly moves until the switch is released. Finally the switch is re-entered in the other direction, setting the reference point to the center of the two switching points. The speed used for this calibration is defined by axis parameter 195.

Axis parameter 193 defines the reference search mode to be used. Choose one of the reference search modes shown in table 17 and in the following subsections:

Reference Search Modes	
Value	Description
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
4	search left stop switch from both sides
5	search home switch in negative direction, reverse the direction when left stop switch reached
6	search home switch in positive direction, reverse the direction when right stop switch reached
7	search home switch in positive direction, ignore end switches
8	search home switch in negative direction, ignore end switches

Table 17: Reference Search Modes

The drawings in the following subsections show how each reference search mode works. A linear stage with two end points and a moving slider is used as example.

7.1.1 Mode 1

Reference search mode 1 only searches the left end switch. Select this mode by setting axis parameter #193 to 1. Figure 3 illustrates this.

Add 64 to the mode number (i.e. set axis parameter #193 to 65) to search the right end switch instead of the left end switch.

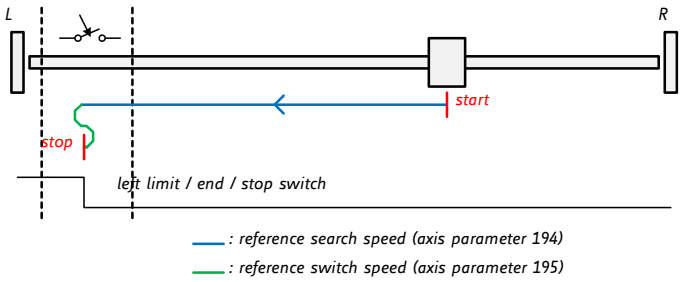


Figure 3: Reference search Mode 1

7.1.2 Mode 2

Reference search mode 2 first searches the right end switch and then the left end switch. The left end switch is then used as the zero point. Figure 4 illustrates this. Select this mode by setting axis parameter #193 to 2. After the reference search has finished, axis parameter #196 contains the distance between the two reference switches in microsteps.

Add 64 to the mode number (i.e. set axis parameter #193 to 66) to search the left end switch first and then use the right end switch as the zero point.

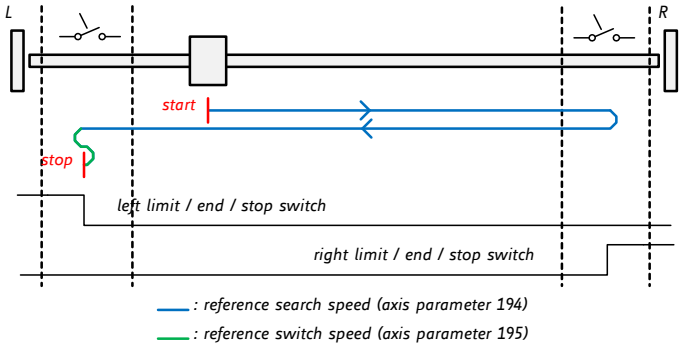


Figure 4: Reference search Mode 2

7.1.3 Mode 3

Reference search mode 3 first searches the right end switch and then the left end switch. The left end switch is then searched from both sides, to find the middle of the left end switch. This is then used as the zero point. Figure 5 illustrates this. Select this mode by setting axis parameter #193 to 3. After the reference search has finished, axis parameter #196 contains the distance between the right end switch and the middle of the left end switch in microsteps.

Add 64 to the mode number (i.e. set axis parameter #193 to 67) to search the left end switch first and then use the middle of the right end switch as the zero point.

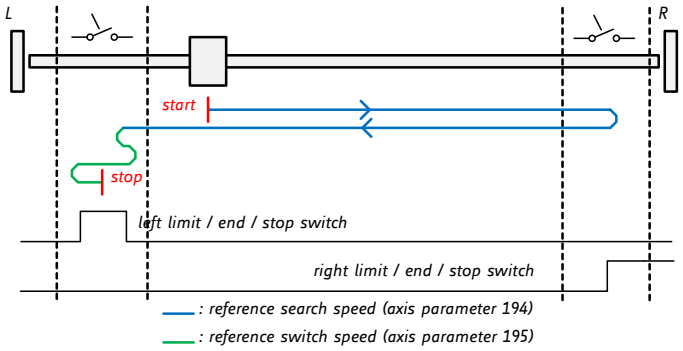


Figure 5: Reference search Mode 3

7.1.4 Mode 4

Reference search mode 4 searches the left end switch only, but from both sides so that the middle of the switch will be found and used as the zero point. This is shown in figure 6.

Add 64 to the mode number (i.e. set axis parameter #193 to 68) to search the right end switch instead.

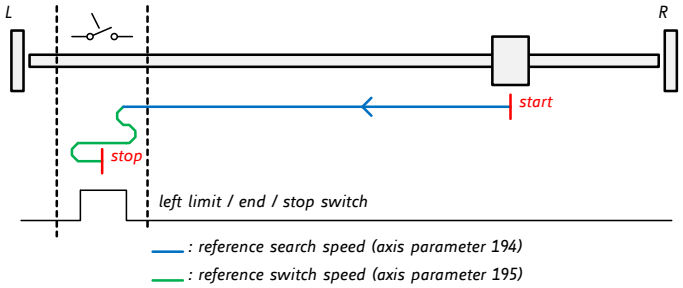


Figure 6: Reference search Mode 4

7.1.5 Mode 5

Refeerence search mode 5 searches the home switch in negative direction. The search direction will be reversed if the left limit switch is reached. This is shown in figure 7.
Add 128 to the mode number (i.e. set axis parameter #193 to 129) to reverse the polarity of the home switch input.

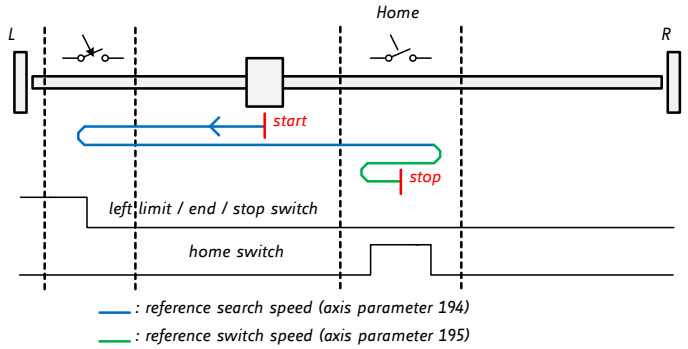


Figure 7: Reference search Mode 5

7.1.6 Mode 6

Reference search mode 6 searches the home switch in positive direction. The search direction will be reversed if the left limit switch is reached. This is shown in figure 8.
Add 128 to the mode number (i.e. set axis parameter #193 to 130) to reverse the polarity of the home switch input.

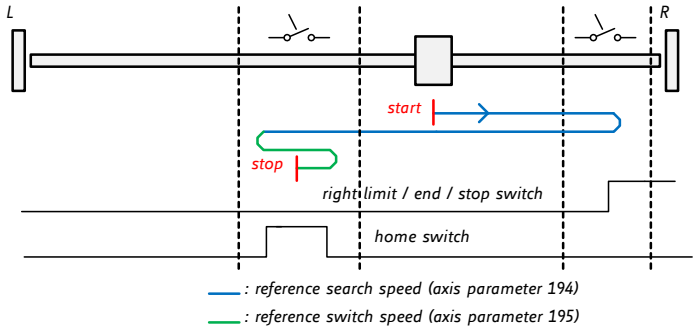


Figure 8: Reference search Mode 6

7.1.7 Mode 7

Reference search mode 7 searches the home switch in positive direction, ignoring the limit switch inputs. It is recommende mainly for use with a circular axis. The exact middle of the switch will be found and used as the zero point. Figure 9 illustrates this.
Add 128 to the mode number (i.e. set axis parameter #193 to 131) to reverse the polarity of the home switch input.

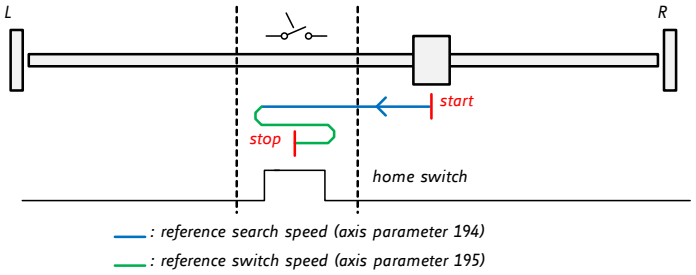


Figure 9: Reference search Mode 7

7.1.8 Mode 8

Reference search mode 8 searches the home switch in positive direction, ignoring the limit switch inputs. It is recommende mainly for use with a circular axis. The exact middle of the switch will be found and used as the zero point. Figure 10 illustrates this.
Add 128 to the mode number (i.e. set axis parameter #193 to 132) to reverse the polarity of the home switch input.

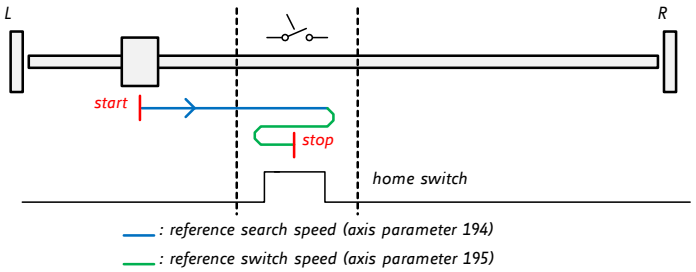


Figure 10: Reference search Mode 8

7.2 stallGuard2

The module is equipped with motor driver chips that feature load measurement. This load measurement 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.

Stall detection means that the motor will be stopped automatically when the load gets too high. This function is configured mainly using axis parameters #174 and #181.

Stall detection can for example be used for finding the reference point without the need for reference switches. A short routine written in TMCL is needed to use stallGuard for reference searching.

7.3 coolStep

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

Figure 11 shows all the adjustment points for coolStep. It is necessary to identify and configure the thresholds for current (I6, I7 and I183) and velocity (V182). Furthermore the stallGuard2 feature has to be adjusted (SG170). It can also be enabled if needed (SG181).

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

In this chapter only basic axis parameters are mentioned which concern coolStep and stallGuard2. The complete list of axis parameters in chapter 4 contains further parameters which offer more configuration options.

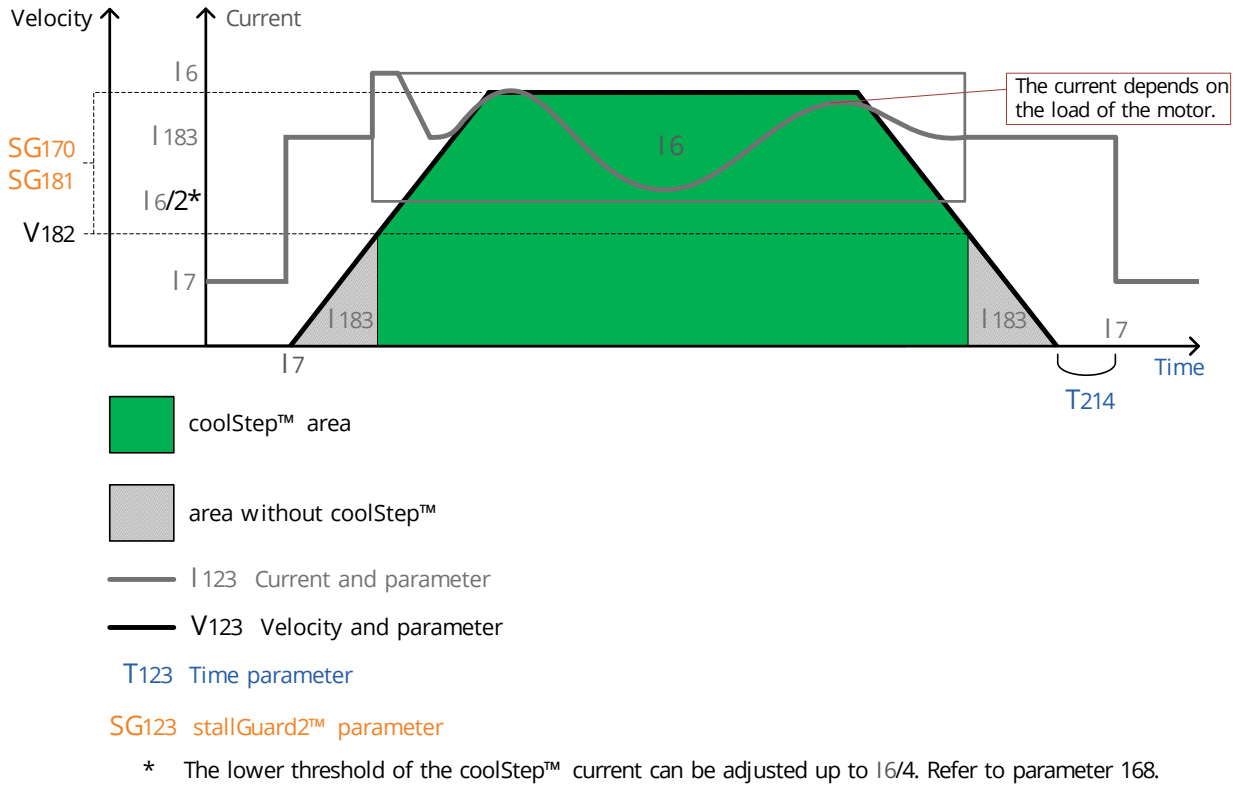


Figure 11: coolStep Adjustment Points and Thresholds

coolStep Adjustment Points and Thresholds		
Number	Axis Parameter	Description

Number	Axis Parameter	Description
I6	Absolute maximum current	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). The most important motor setting, since too high values might cause motor damage!
I7	Standby current	The current limit two seconds after the motor has stopped.
I168	smartEnergy current minimum	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
I169	smartEnergy current down step	Sets the number of stallGuard2 readings above the upper threshold necessary for each current decrement of the motor current. Number of stallGuard2 measurements per decrement: Scaling: 0...3: 32, 8, 2, 1 0: slow decrement 3: fast decrement
I171	smartEnergy current up step	Sets the current increment step. The current becomes incremented for each measured stallGuard2 value below the lower threshold (see smartEnergy hysteresis start). current increment step size: Scaling: 0...3: 1, 2, 4, 8 0: slow increment 3: fast increment
SG170	smartEnergy hysteresis	Sets the distance between the lower and the upper threshold for stallGuard2 reading. Above the upper threshold the motor current becomes decreased.
SG181	Stop on stall	Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2 load value reaches zero.
V182	smartEnergy threshold speed	Above this speed coolStep becomes enabled.
T214	Power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (which means 2000msec).

Table 18: coolStep Adjustment Points and Thresholds

7.4 Velocity and Acceleration Calculation

When the unit mode (axis parameter #255) is set to 1 (which is also the default value), all velocity parameters on the TMCM-1270 are given in microsteps per second (also called pulse per second or pps). Acceleration and deceleration units are given in pps^2.
When axis parameter #255 is set to 0 the internal units of the ramp generators are directly used. But this is only necessary in very special cases. Normally one should leave axis parameter #255 at 1 and use the pps units.
In order to convert between pps units and units like rounds per second (rps) or rounds per minute (rpm), one has to know the fullstep resolution of the motor (full steps per round) and the microstep resolution setting of the module (axis parameter #140, default setting is 256 microsteps per full step).
So to convert from pps to rps, use the following formula:

$$v_{rps} = \frac{v_{pps}}{r_{fullstep} \cdot r_{microstep}}$$

To convert from rps to rpm, use:

$$v_{rpm} = v_{rps} \cdot 60$$

With the following symbols:

- v_{rps}: velocity in rounds per second
- v_{rpm}: velocity in rounds per minute
- v_{pps}: velocity in pulses (microsteps) per second
- $r_{fullstep}$: fullstep resolution of the motor (with most motors 200 (1.8°))
- $r_{microstep}$: microstep setting of the module (default 256)

So, with a 200 fullsteps motor and a microstep setting of 256 (axis parameter #140 = 8), a velocity of 51200pps will result in 1rps (60rpm).

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.

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

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

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

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. Please consider the following example:

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

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

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


```
  WAIT POS, Motor0, 0
17 JA MainLoop
```

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

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

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 as well as STGP and RSGP are used to work with user variables:

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

Please see the following example:

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

5 SGP MyVariable, 2, 1234   //Initialize the variable with the value 1234
  ...
7 ...
  GGP MyVariable, 2    //Copy contents of variable to accumulator register
9 CALC MUL, 2          //Multiply accumulator register with two
  AAP MyVariable, 2    //Store contents of accumulator register to variable
11 ...
  ...
```

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 Combining Direct Mode and Standalone Mode

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

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

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

Example:

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

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

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

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

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

9 Figures Index

10 Tables Index

11 Supplemental Directives

11.1 Producer Information

11.2 Copyright

11.3 Trademark Designations and Symbols

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

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

11.4 Target User

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

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

11.5 Disclaimer: Life Support Systems

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

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

Information given in this document is believed to be accurate and reliable. However, no responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties which may result from its use. Specifications are subject to change without notice.

11.6 Disclaimer: Intended Use

The data specified in this user manual is intended solely for the purpose of product description. No representations or warranties, either express or implied, of merchantability, fitness for a particular purpose

or of any other nature are made hereunder with respect to information/specification or the products to which information refers and no guarantee with respect to compliance to the intended use is given.

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

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

11.7 Collateral Documents & Tools

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

12 Revision History

12.1 Document Revision

Version	Date	Author	Description
V0.8	2016-JUL-12	BS	Exported from xml.
V0.9	2016-SEP-26	BS	First complete version.
V1.00	2016-NOV-18	BS	Final version with changes from Fw V1.0.

Table 19: Document Revision