Firmware Version V1.29

TMCL™ FIRMWARE MANUAL

TMCM-1021

1-Axis Stepper
Controller / Driver
24V DC
up-to 0.7A RMS / 1.4A RMS
RS485 Interface
sensOstep™ Encoder

UNIQUE FEATURES:

coolStep™

stallGuard2™

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

www.trinamic.com

TRINAMIC
MOTION CONTROL

Table of Contents

1 Features

The TMCM-1021 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 can be mounted on the back of NEMA11 (28mm flange size) and has been designed for coil currents up to 0.7A RMS (low current range, programmable) or 1.4A RMS (high current range, programmable, new additional range since hardware version 1.4) and 24V DC supply voltage. With its high energy efficiency from TRINAMIC's coolStep™ technology cost for power consumption is kept down. The TMCL™ firmware supports remote control (direct mode) and standalone operation (with TMCL program being executed on the TMCM-1021 itself).

MAIN CHARACTERISTICS

Highlights

- Motion profile calculation in real-time
- On the fly alteration of motor parameters (e.g. position, velocity, acceleration)
- High performance microcontroller for overall system control and serial communication protocol handling
- For position movement applications, where larger motors do not fit and higher torques are not required

Bipolar stepper motor driver

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

Encoder

- sensOstep magnetic encoder (max. 1024 increments per rotation) e.g. for step-loss detection under all operating conditions and positioning supervision

Interfaces

- Up to 4 multi-purpose inputs (2 shared with general purpose outputs)
- 2 general purpose outputs
- RS485 2-wire communication interface

Software

- TMCL: standalone operation or remote controlled operation,
  program memory (non volatile) for up to 876 TMCL commands, and
  PC-based application development software TMCL-IDE available for free.

Electrical and mechanical data

- Supply voltage: +24V DC nominal (9… 28V DC max.)
- Motor current: up to 0.7A RMS (low current range, programmable) or 1.4A RMS (high current range, programmable, new additional range since hardware version 1.4)

Refer to separate Hardware Manual, too.

TRINAMICs Unique Features – Easy to Use with TMCL

stallGuard2™ stallGuard2 is a high-precision sensorless load measurement using the back EMF on 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 goes to zero or near to zero. This is the most energy-efficient point of operation for the motor.

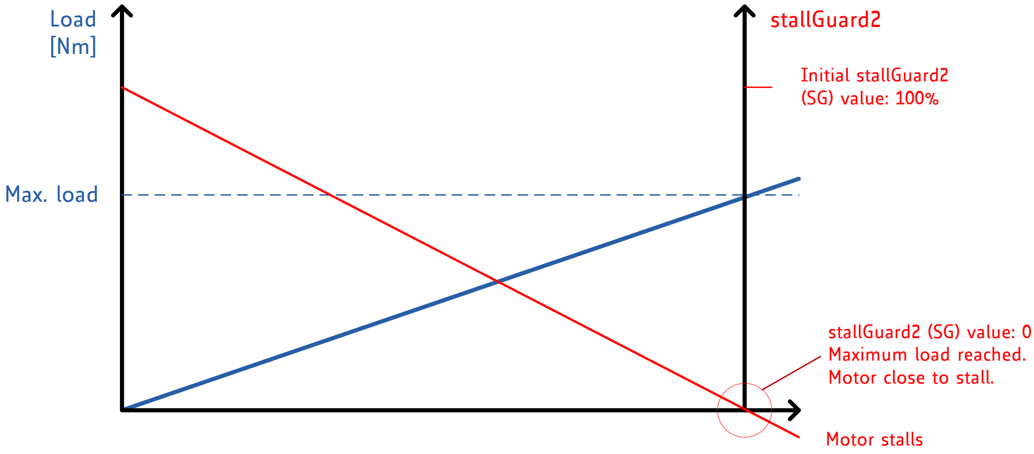


Figure 1.1 stallGuard2 load measurement SG as a function of load

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 system cooler, increases motor life, and allows reducing cost.

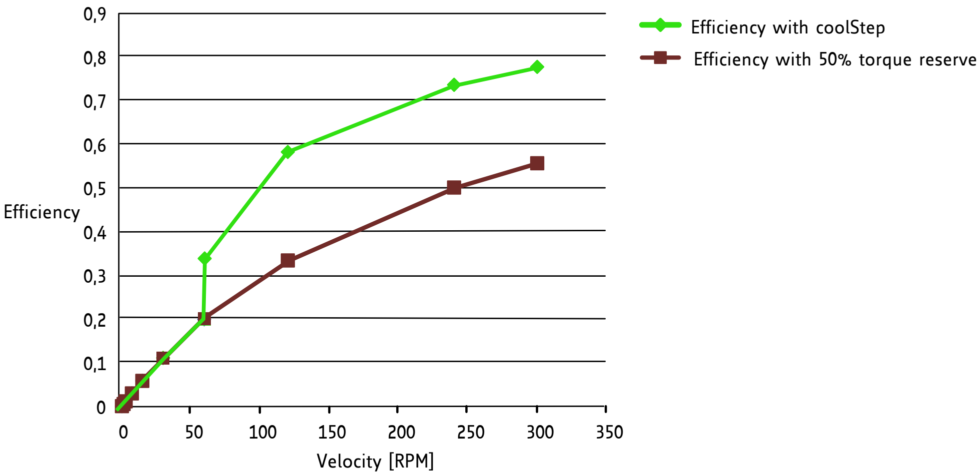


Figure 1.2 Energy efficiency example with coolStep

2 Putting the Module into Operation

Here you can find basic information for putting your TMCM-1021 into operation. If you are already common with TRINAMICs modules you may skip this chapter.

The things you need:

- TMCM-1021 with fitting motor
- RS485 interface converter with cables
- Nominal supply voltage +24V DC for your module
- TMCL-IDE program and PC

PRECAUTIONS
Do not connect or disconnect the TMCM-1021 while powered!
Do not connect or disconnect the motor while powered!
Do not exceed the maximum power supply voltage of 28V DC!
Note, that the module is not protected against reverse polarity!
START WITH POWER SUPPLY OFF!

2.1 Basic Set-Up

The following paragraph will guide you through the steps of connecting the unit and making first movements with the motor.

2.1.1 Connecting the Module

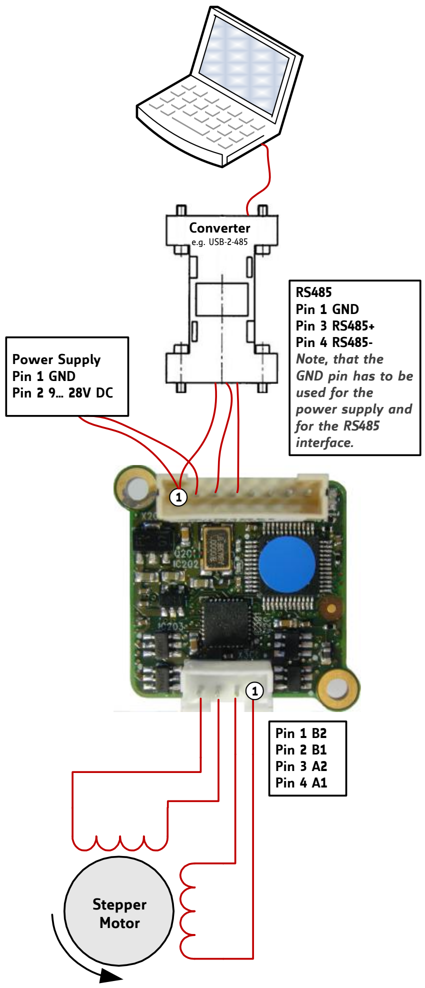


Figure 2.1: Starting up

1. **Connect RS485 interface and power supply.**

Pin	Label	Description
1	GND	GND
2	VDD	VDD (+9V…+28V)
3	RS485+	RS485 interface, diff. signal (non-inverting)
4	RS485-	RS485 interface, diff. signal (inverting)
5	IN_0	Digital input (+24V compatible)
		Alternate function 1: step input
		Alternate function 2: left stop switch
6	IN_1	Digital input (+24V compatible)
		Alternate function 1: direction input
		Alternate function 2: right stop switch
7	OUT_0 / IN_2	Open drain output with freewheeling diode (max. 100mA)
		Alternate function 1: digital input (+24V compatible)
		Alternate function 2:home switch
8	OUT_1 / IN_3	Open drain output with freewheeling diode (max. 100mA)
		Alternate function 1: digital input (+24V compatible)
		Alternate function 2: analog input

2. Connect motor

Pin	Label	Description
1	OB2	Pin 2 of motor coil B
2	OB1	Pin 1 of motor coil B
3	OA2	Pin 2 of motor coil A
4	OA1	Pin 1 of motor coil A

3. Switch ON the power supply

Turn power ON. The green LED for power lights up slowly and the motor is powered but in standstill now.

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

2.1.2 Start the TMCL-IDE Software Development Environment

The TMCL-IDE is available on www.trinamic.com.

Installing the TMCL-IDE:
Make sure the COM port you intend to use is not blocked by another program.
Open TMCL-IDE by clicking ***TMCL.exe***.
Choose ***Setup*** and ***Options*** and thereafter the ***Connection tab***.
Choose ***COM port*** and ***type*** with the parameters shown in Figure 2.2 (baud rate 9600). Click ***OK***.

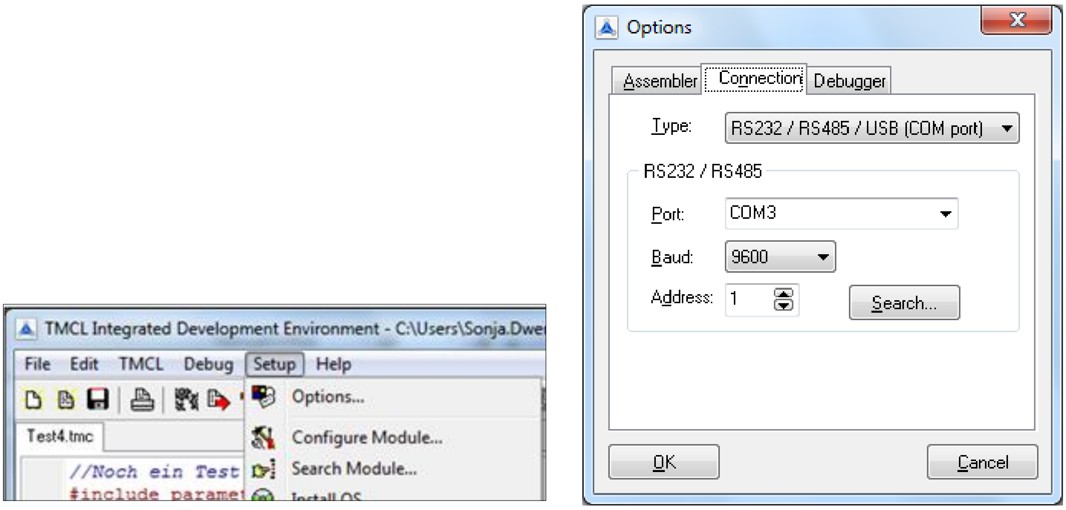
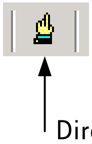


Figure 2.2 Setup dialogue and connection tab of the TMCL-IDE.

Please refer to the TMCL-IDE User Manual for more information (see www.TRINAMIC.com).

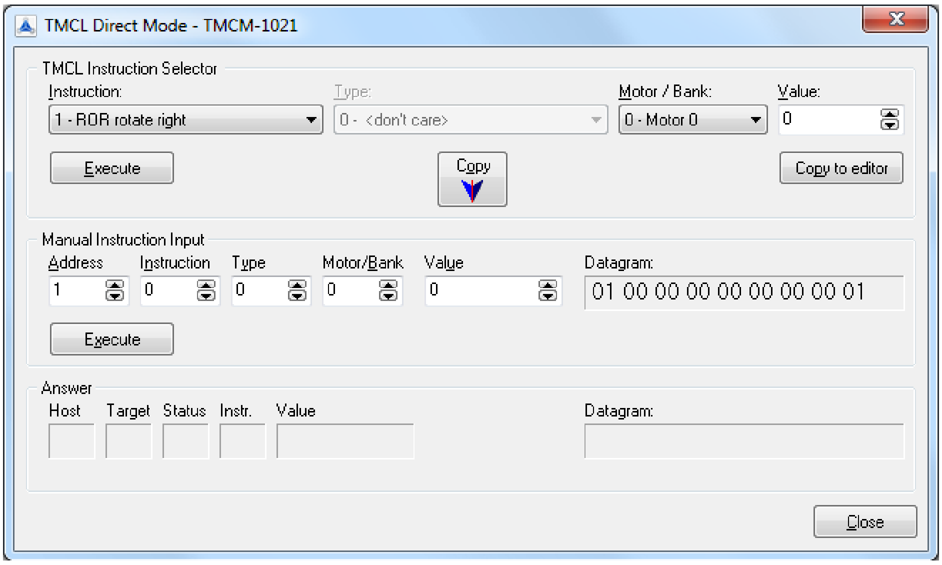
2.2 Using TMCL Direct Mode

1. Start TMCL ***Direct Mode***.



Direct Mode

2. If the communication is established the TMCM-1021 is automatically detected. ***If the module is not detected, please check all points above (cables, interface, power supply, COM port, baud rate).***



3. Issue a command by choosing ***Instruction***, ***Type*** (if necessary), ***Motor***, and ***Value*** and click ***Execute*** to send it to the module.

Examples:

- ROR rotate right, motor 0, value 10000 -> Click *Execute*. The first motor is rotating now.
- MST motor stop, motor 0 -> Click *Execute*. The first motor stops now.

Top right of the *TMCL Direct Mode* window is the button *Copy to editor*. Click here to copy the chosen command and create your own TMCL program. The command will be shown immediately on the editor.

NOTE
Please mind chapter 7 (programming techniques) of the TMCL-IDE User Manual on *www.trinamic.com*. Here you will find information about creating general structures of TMCL programs. In particular initialization, main loop, symbolic constants, variables, and subroutines are described there. Further you can learn how to mix direct mode and stand alone mode.

Chapter 4.3 of this manual includes a diagram which points out the coolStep related axis parameters and their functions. This can help you configuring your module to meet your needs.

2.2.1 Important Motor Settings

There are some axis parameters which have to be adjusted right in the beginning after installing your module. Please set the upper limiting values for the speed (axis parameter 4), the acceleration (axis parameter 5), and the current (axis parameter 6). Further set the standby current (axis parameter 7) and choose your microstep resolution with axis parameter 140. Please use the *SAP* (Set Axis Parameter) command for adjusting these values. The SAP command is described in paragraph 3.5.5. You can use the TMCM-IDE direct mode for easily configuring your module.

> ATTENTION
> The most important motor setting is the *absolute maximum motor current* setting, since too high values might cause motor damage!

IMPORTANT AXIS PARAMETERS FOR MOTOR SETTING

Number	Axis Parameter	Description	Range [Unit]
4	Maximum positioning speed	Maximum feasible positioning speed. Has to be adapted to motor and application	0… +268.435.454 [pps/s]
5	Maximum acceleration	Limit for acceleration and deceleration. Has to be adapted to motor and application.	1… +33554431 [pps/s]
6	Absolute max. current (CS / Current Scale)	The maximum value is 255. This value means 100% of maximum programmable current of the selected motor current range (see axis parameter 179). Current can be adjusted / scaled down by specifying a lower value between 0 and 255. This value is transformed into 32 different internal current settings supported by the hardware (see hardware manual for more details and complete table with possible current settings). ***Please note: high current range is available for hardware version V1.4, only!***	0… 255 *Low current range scaling (axis parameter 179 set to 1):* $I_{peak} = < value > \times \frac{1A}{255}$ $I_{RMS} = < value > \times \frac{0.7A}{255}$ *High current range scaling (axis parameter 179 set to 0):* $I_{peak} = < value > \times \frac{2A}{255}$ $I_{RMS} = < value > \times \frac{1.4A}{255}$
7	Standby current	The current limit two seconds after the motor has stopped.	0… 255 $I_{peak} = < value > \times \frac{1A}{255}$ $I_{RMS} = < value > \times \frac{0.7A}{255}$
140	Microstep resolution	0 full step 1 half step 2 4 microsteps 3 8 microsteps 4 16 microsteps 5 32 microsteps 6 64 microsteps 7 128 microsteps 8 256 microsteps	0… 8

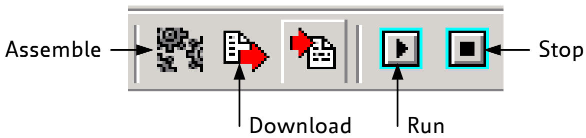
Number	Axis Parameter	Description	Range [Unit]
179	VSENSE	sense resistor voltage based current scaling 0: high current range up-to 1.4A RMS / 2A peak 1: low current range up-to 0.7A RMS / 1A peak (default value) ***Please note: this parameter should not and cannot be changed for hardware V1.2! The high current range is available for hardware V1.4, only!***	0/1

2.3 Testing with a Simple TMCL Program

Type in the following program:

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

        SAP 4, 0, 50000             //Set max. Velocity
        SAP 5, 0, 50000             //Set max. Acceleration
Loop:   MVP ABS, 0, 100000          //Move to Position 100000
        WAIT POS, 0, 0              //Wait until position reached
        MVP ABS, 0, -100000         //Move to Position -100000
        WAIT POS, 0, 0              //Wait until position reached
        JA Loop                     //Infinite Loop
```



1. Click the ***Assemble*** icon to convert the TMCL into machine code.
2. Then download the program to the TMCM-1021 module by clicking the ***Download*** icon.
3. Press icon ***Run***. The desired program will be executed.
4. Click the ***Stop*** button to stop the program.

3 TMCL and the TMCL-IDE: Introduction

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-1061 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-1021 supports TMCL direct mode (binary commands) and standalone TMCL program execution. You can store up to 876 TMCL instructions on it.

In direct mode and most cases the TMCL communication over RS485 follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCL-1021. The TMCL interpreter on the module will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over RS485 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 RS485 interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In this case it consists of nine bytes.

The binary command format for RS485 is as follows:

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

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

Checksum calculation

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

- in C:

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

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

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

- in Delphi:

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

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

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

3.2 Reply Format

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

The reply format for RS485 is as follows:

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

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

3.2.1 Status Codes

The reply contains a status code.

The status code can have one of the following values:

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

3.3 Standalone Applications

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

3.4 TMCL Command Overview

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

3.4.1 TMCL Commands

Command	Number	Parameter	Description
ROR	1	<motor number>, <velocity>	Rotate right with specified velocity
ROL	2	<motor number>, <velocity>	Rotate left with specified velocity
MST	3	<motor number>	Stop motor movement
MVP	4	ABS|REL|COORD, <motor number>, <position|offset>	Move to position (absolute or relative)
SAP	5	<parameter>, <motor number>, <value>	Set axis parameter (motion control specific settings)
GAP	6	<parameter>, <motor number>	Get axis parameter (read out motion control specific settings)
STAP	7	<parameter>, <motor number>	Store axis parameter permanently (non volatile)
RSAP	8	<parameter>, <motor number>	Restore axis parameter
SGP	9	<parameter>, <bank number>, value	Set global parameter (module specific settings e.g. communication settings or TMCL user variables)
GGP	10	<parameter>, <bank number>	Get global parameter (read out module specific settings e.g. communication settings or TMCL user variables)
STGP	11	<parameter>, <bank number>	Store global parameter (TMCL user variables only)
RSGP	12	<parameter>, <bank number>	Restore global parameter (TMCL user variable only)
RFS	13	START|STOP|STATUS, <motor number>	Reference search

Command	Number	Parameter	Description
SIO	14	<port number>, <bank number>, <value>	Set digital output to specified value
GIO	15	<port number>, <bank number>	Get value of analogue/digital input
CALC	19	<operation>, <value>	Process accumulator & value
COMP	20	<value>	Compare accumulator <-> value
JC	21	<condition>, <jump address>	Jump conditional
JA	22	<jump address>	Jump absolute
CSUB	23	<subroutine address>	Call subroutine
RSUB	24		Return from subroutine
EI	25	<interrupt number>	Enable interrupt
DI	26	<interrupt number>	Disable interrupt
WAIT	27	<condition>, <motor number>, <ticks>	Wait with further program execution
STOP	28		Stop program execution
SCO	30	<coordinate number>, <motor number>, <position>	Set coordinate
GCO	31	<coordinate number>, <motor number>	Get coordinate
CCO	32	<coordinate number>, <motor number>	Capture coordinate
CALCX	33	<operation>	Process accumulator & X-register
AAP	34	<parameter>, <motor number>	Accumulator to axis parameter
AGP	35	<parameter>, <bank number>	Accumulator to global parameter
VECT	37	<interrupt number>, <label>	Set interrupt vector
RETI	38		Return from interrupt
ACO	39	<coordinate number>, <motor number>	Accu to coordinate

3.4.2 Commands Listed According to Subject Area

3.4.2.1 Motion Commands

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

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

3.4.2.2 Parameter Commands

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

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

3.4.2.3 Control Commands

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

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

3.4.2.4 I/O Port Commands

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

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

3.4.2.5 Calculation Commands

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

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

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

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

3.4.2.6 Interrupt Commands

Due to some customer requests, interrupt processing has been introduced in the TMCL firmware for ARM based modules.

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

3.4.2.6.1 Interrupt Types

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

3.4.2.6.2 Interrupt Processing

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

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

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

3.4.2.6.3 Interrupt Vectors

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached
15	stallGuard2
21	Deviation
27	Left stop switch
28	Right stop switch
39	Input change 0
40	Input change 1
255	Global interrupts

3.4.2.6.4 Further Configuration of Interrupts

Some interrupts need further configuration (e.g. the timer interval of a timer interrupt). This can be done using SGP commands with parameter bank 3 (SGP <type>, 3, <value>). Please refer to the SGP command (paragraph 3.5.9) for further information about that.

3.4.2.6.5 Using Interrupts in TMCL

For using an interrupt proceed as follows:

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

The following example shows the use of a timer interrupt:

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

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

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

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

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

Please also take a look at the other example programs.

3.5 Commands

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

3.5.1 ROR (rotate right)

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

Like on all other TMCL modules, the motor will be accelerated or decelerated to the speed given with the command. The speed is given in microsteps per second (pps). For conversion of this value into rounds per minute etc. please refer to chapter 0, also.

The range is -268.435.455… +268.435.454.

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

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR 0, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
1	(don't care)	0*	<velocity> -268.435.455… +268.435.454

***motor number is always 0 as the module supports just one axis**

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

Rotate right, velocity = 10000
Mnemonic: ROR 0, 10000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$01	$00	$00	$00	$00	$27	$10	$39

3.5.2 ROL (rotate left)

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

Like on all other TMCL modules, the motor will be accelerated or decelerated to the speed given with the command. The speed is given in microsteps per second (pps). For conversion of this value into rounds per minute etc. please refer to chapter 5.2, also.

The range is -268.435.455… +268.435.454.

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

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL 0, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
2	(don't care)	0*	<velocity> -268.435.455… +268.435.454

*** motor number is always 0 as the module supports just one axis**

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

Rotate left, velocity = 10000
Mnemonic: ROL 0, 10000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$02	$00	$00	$00	$00	$27	$10	$3a

3.5.3 MST (motor stop)

With this command the motor will be instructed to stop. The command uses the normal acceleration parameter (soft stop / deceleration ramp possible).

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

Related commands: ROL, ROR, SAP, GAP

Mnemonic: MST 0

Binary representation:

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

*** motor number is always 0 as the module support just one axis**

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

Stop motor
Mnemonic: MST 0

Binary:

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

3.5.4 MVP (move to position)

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

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

Attention:

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

Two operation types are available:

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

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

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

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
4	0 ABS – absolute	0*	<position> −2.147.483.648… +2.147.483.647
	1 REL – relative	0*	<offset> −2.147.483.648… +2.147.483.647
	2 COORD – coordinate	0*	<coordinate number> 0… 20

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

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

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$04	$00	$00	$00	$01	$5f	$90	$f5

Example:

Move motor from current position 10000 steps backward (move relative −10000)
Mnemonic: MVP REL, 0, -10000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$04	$01	$00	$ff	$ff	$d8	$f0	$cc

Example:

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

Binary:

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

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

3.5.5 SAP (set axis parameter)

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

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

Related commands: GAP, STAP, RSAP, AAP

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

Binary representation:

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

*** motor number is always 0 as the module supports just one axis**

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Example:

Set the absolute maximum current of the motor during movements to approx. 78% of max. module
current:

Mnemonic: SAP 6, 0, 200

Binary:

Byte Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$05	$06	$00	$00	$00	$00	$c8	$d4

3.5.6 GAP (get axis parameter)

Most parameters of the TMCM-1021 can be adjusted individually for the axis. With this parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditioned jumps). In direct mode the value read is only output in the *value* field of the reply (without affecting the accumulator).

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

Related commands: SAP, STAP, AAP, RSAP

Mnemonic: GAP <parameter number>, 0

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

For a table with parameters which can be used together with this command please refer to chapter 4.

Example:

Get actual position of motor
Mnemonic: GAP 1, 0

Binary:

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

Reply:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Host-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$02	$01	$64	$06	$00	$00	$07	$d0	$44

⇨ **Status = 100 (no error), position = 2000**

3.5.7 STAP (store axis parameter)

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

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

Related commands: SAP, RSAP, GAP, AAP

Mnemonic: STAP <parameter number>, 0

Binary representation:

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

***1motor number is always 0 as only one motor is involved**
****2the value operand of this function has no effect. Instead, the current setting (e.g. previously set with SAP) is saved.***

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Parameter number	Motor number	Value
s. chapter 4	0	s. chapter 4

For a table with parameters which can be used together with this command please refer to chapter 4.

Example:

Store the maximum speed of motor
Mnemonic: STAP 4, 0

Binary:

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

Note: The STAP command will not have any effect when the configuration EEPROM is locked (refer to 5.1). In direct mode, the error code 5 (configuration EEPROM locked, see also section 0) will be returned in this case.

3.5.8 RSAP (restore axis parameter)

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

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

Relate commands: SAP, STAP, GAP, and AAP

Mnemonic: RSAP <parameter number>, 0

Binary representation:

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

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

Reply structure in direct mode:

STATUS	VALUE
100 – OK	(don't care)

For a table with parameters which can be used together with this command please refer to chapter 4.

Example:

Restore the maximum current of motor
Mnemonic: RSAP 6, 0

Binary:

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

3.5.9 SGP (set global parameter)

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

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

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

Related commands: GGP, STGP, RSGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Example:

Set the serial address of the target device (module) to 3
Mnemonic: SGP 66, 0, 3

Binary:

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

3.5.10 GGP (get global parameter)

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

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

Related commands: SGP, STGP, RSGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Example:

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

Binary:

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

Reply:

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

⇨ **Status = 100 (no error), Value = 1**

3.5.11 STGP (store global parameter)

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

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

Related commands: SGP, GGP, RSGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Example:

Store user variable #42
Mnemonic: STGP 42, 2

Binary:

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

3.5.12 RSGP (restore global parameter)

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

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

Relate commands: SGP, STGP, GGP, and AGP

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

Binary representation:

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

Reply structure in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Example:

Restore user variable #42

Mnemonic: RSGP 42, 2

Binary:

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

3.5.13 RFS (reference search)

The TMCL firmware has a built-in reference search algorithm which can be used. The reference search algorithm provides switching point calibration and three switch modes. The status of the reference search can also be queried to see if it has already finished. Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (chapter 4). The reference search can be started, stopped, and the actual status of the reference search can be checked.

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

Related commands: WAIT

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

Binary representation:

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

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

Reply in direct mode:
When using type 0 (START) or 1 (STOP):

STATUS	VALUE
100 – OK	(don't care)

When using type 2 (STATUS):

STATUS	VALUE
100 – OK	0 – no ref. search active other values – ref. search is active

Example:

Start reference search of motor
Mnemonic: RFS START, 0

Binary:

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

3.5.14 SIO (set output)

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

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

Related commands: GIO, WAIT

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

Binary representation:

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

Reply structure:

STATUS	VALUE
100 – OK	(don't care)

Example:

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

Binary:

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

I/O pin definition

Power / communication / I/Os

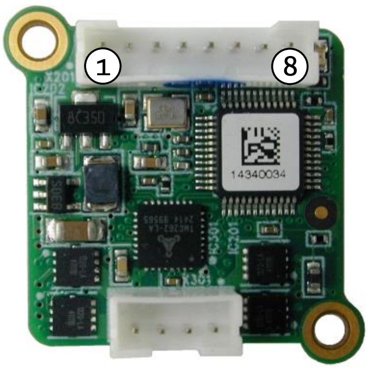


Figure 3.1: I/O connector of TMCM-1021

Please note, that the module has four I/O pins including two open drain outputs:

Pin 7: OUT_0 (open collector)
Pin 8: OUT_1 (open collector)

Please refer to the Hardware Manual for further information.

Available I/O ports of TMCM-1021:

Pin	I/O port	Command	Range <*n*>
7	OUT_0	SIO 0, 2, <*n*>	1/0
8	OUT_1	SIO 1, 2, <*n*>	1/0

Addressing both output lines with one SIO command:

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

Example:

Set both output pins high.
Mnemonic: SIO 255, 2, 127

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

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

3.5.15 GIO (get input/output)

With this command the status of the available general purpose inputs of the module can be read out. The function reads a digital or analogue input port. Digital lines will be read as 0 or 1, while the ADC channels deliver their 12bit result in the range of 0… 4095. In standalone mode the requested value is copied to the *accumulator* (accu) 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 specified signal pin is read.

Related commands: SIO, WAIT

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	<status of the port>

Example:

Get the analogue value of IN_3
Mnemonic: GIO 3, 1

Binary:

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

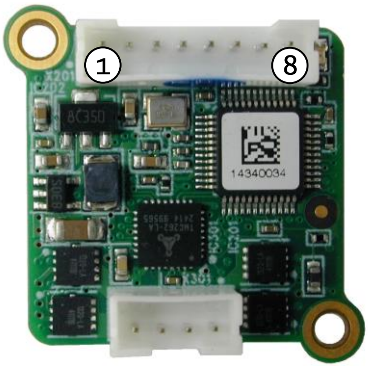
Reply:

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

⇨ value: 506

<u>**I/O pin definition**</u>

Power / communication / I/Os



Please note, that the module has four I/O pins including four input pins:

Pin 5: IN_0 (digital)
Pin 6: IN_1 (digital)
Pin 7: IN_2 (digital)
Pin 8: IN_3 (digital or analog)

Please refer to the Hardware Manual for further information.

Figure 3.2: I/O connector of TMCM-1021

3.5.15.1 I/O bank 0 – digital inputs:

IN_3 can be read as digital or analogue input. The analogue values can be accessed in bank 1.

Pin	I/O port	Command	Result range
5	IN_0	GIO 0, 0	0/1
6	IN_1	GIO 1, 0	0/1
7	IN_2 (same pin as OUT_0)	GIO 2, 0	0/1
8	IN_3 (same pin as OUT_1)	GIO 3, 0	0/1

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

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

<u>**Use following program to represent the states of the input lines on the output lines:**</u>

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

3.5.15.2 I/O bank 1 – analogue input:

IN_3 can be read back as digital or analogue input. The digital states can be accessed in bank 0.

Pin	I/O port	Command	Result range
8	IN_3 (same pin as OUT_1)	GIO 3, 1	0… 4095

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

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

Pin	I/O port	Command	Result range
7	OUT_0	GIO 0, 2	0/1
8	OUT_1	GIO 1, 2	0/1

3.5.16 CALC (calculate)

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

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

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

Binary representation:

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

Example:

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

Binary:

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

3.5.17 COMP (compare)

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

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

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

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

Mnemonic: COMP <value>

Binary representation:

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

Example:

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

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

Binary format of the COMP 1000 command:

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

3.5.18 JC (jump conditional)

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

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

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

Related commands: JA, COMP, WAIT, CLE

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

Binary representation:

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

Example:

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

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

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

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

3.5.19 JA (jump always)

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

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

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

Related commands: JC, WAIT, CSUB

Mnemonic: JA <Label>

Binary representation:

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

Example:
An infinite loop in TMCL™

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

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

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

3.5.20 CSUB (call subroutine)

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

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

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

Related commands: RSUB, JA

Mnemonic: CSUB <Label>

Binary representation:

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

Example:

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

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

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

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

3.5.21 RSUB (return from subroutine)

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

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

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

Related command: CSUB

Mnemonic: RSUB

Binary representation:

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

Example: please see the CSUB example (section 3.5.20).

Binary format of RSUB:

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

3.5.22 WAIT (wait for an event to occur)

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

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

There are five different wait conditions that can be used:

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

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

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

Related commands: JC, CLE

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

Binary representation:

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

***[1] motor number is always 0 as only one motor is involved**
***[2] one tick is 10 milliseconds**

Example:

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

Binary:

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

3.5.23 STOP (stop TMCL program execution)

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

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

Internal function: TMCL instruction fetching is stopped.

Related commands: none
Mnemonic: STOP

Binary representation:

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

Example:

Mnemonic: STOP

Binary:

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

3.5.24 SCO (set coordinate)

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

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

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

Related commands: GCO, CCO, MVP

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

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

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

Binary:

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

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

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

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

3.5.25 GCO (get coordinate)

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

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

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

Related commands: SCO, CCO, MVP

Mnemonic: GCO <coordinate number>, 0

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

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

Binary:

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

Reply:

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

⇨ **Value: 0**

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

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

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

3.5.26 CCO (capture coordinate)

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

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

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

Related commands: SCO, GCO, MVP

Mnemonic: CCO <coordinate number>, 0

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

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

Binary:

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

3.5.27 ACO

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

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

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

Related commands: GCO, CCO, MVP COORD, SCO

Mnemonic: ACO <coordinate number>, 0

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

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

Binary:

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

3.5.28 CALCX (calculate using the X register)

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

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

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

Binary representation:

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

Example:

Multiply accu by X-register
Mnemonic: CALCX MUL

Binary:

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

3.5.29 AAP (accumulator to axis parameter)

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

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

Mnemonic: AAP <parameter number>, 0

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Example:

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

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

Binary format of the AAP 0,0 command:

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

3.5.30 AGP (accumulator to global parameter)

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

Related commands: SGP, GGP, STGP, RSGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Example:

Copy accumulator to TMCL user variable #3
Mnemonic: AGP 3, 2

Binary:

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

3.5.31 CLE (clear error flags)

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

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

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

Related commands: JC

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

Binary representation:

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

Example:

Reset the timeout flag
Mnemonic: CLE ETO

Binary:

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

3.5.32 VECT (set interrupt vector)

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

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

Related commands: EI, DI, RETI

Mnemonic: VECT <interrupt number>, <label>

Binary representation:

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

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	(Target) position reached
15	Stall (stallGuard2™)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change

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

Binary format of VECT:

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

3.5.33 EI (enable interrupt)

The EI command enables an individual interrupt and activates interrupts in general (global interrupt enable). Please make sure to always issue a global interrupt enable in order to actually activate the interrupts individually enabled.

Related command: DI, VECT, RETI

Mnemonic: EI <interrupt number>

Binary representation:

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

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	(Target) position reached
15	Stall (stallGuard2™)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change
255	Global interrupts

Examples:

Enable interrupts globally
EI, 255

Binary format of EI:

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

Enable interrupt when target position reached
EI, 3

Binary format of EI:

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

3.5.34 DI (disable interrupt)

The DI command disables an individual interrupt or using parameter 255 will de-activate any interrupt.

Related command: EI, VECT, RETI

Mnemonic: DI <interrupt number>

Binary representation:

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

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	(Target) position reached
15	Stall (stallGuard2™)
21	Deviation
39	IN_0 change
40	IN_1 change
255	Global interrupts

Examples:

Disable interrupts globally
DI, 255

Binary format of DI:

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

Disable interrupt when target position reached
DI, 3

Binary format of DI:

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

3.5.35 RETI (return from interrupt)

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

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

Related commands: EI, DI, VECT

Mnemonic: RETI

Binary representation:

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

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

Binary format of RETI:

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

3.5.36 Customer Specific TMCL Command Extension (UF0… UF7/user function)

The user definable functions UF0… UF7 are predefined functions for user specific purposes. Contact TRINAMIC for the customer specific programming of these functions.

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

Related commands: none

Mnemonic: UF0… UF7

Binary representation:

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

Reply in direct mode:

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

3.5.37 Request Target Position Reached Event

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

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

Mnemonic: ---

Binary representation:

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

*** Motor number**

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

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

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

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

3.5.38 TMCL Control Functions

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

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

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

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

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

Special reply format of command 136:

Type set to 0 - reply as a string:

Byte index	Contents
1	Host Address
2… 9	Version string (8 characters, e.g. *1021V129*)

There is no checksum in this reply format!

Type set to 1 - version number in binary format:

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

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

4 Axis Parameters

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

Meaning of the letters in column *Access*:

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

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

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

Number	Axis Parameter	Description	Range [Unit]	Acc.
0	Target (next) position	The desired position in position mode (see ramp mode, no. 128).	–2.147.483.648… +2.147.483.647 [µsteps]	RW
1	Actual position	The current position of the motor. Should only be overwritten for reference point setting.	–2.147.483.648… +2.147.483.647 [µsteps]	RW
2	Target (next) speed	The desired speed in velocity mode (see ramp mode, no. 128). In position mode, this parameter is set automatically: to the maximum speed during acceleration, and to zero during deceleration and rest.	-268.435.455… +268.435.454 [pps]	RW
3	Actual speed	The current rotation speed.	-268.435.455… +268.435.454 [pps]	RW
4	Maximum positioning speed	Maximum feasible positioning speed. Has to be adapted to motor and application	0… +268.435.454 [pps]	RWE
5	Maximum acceleration	Limit for acceleration and deceleration. Has to be adapted to motor and application.	1… +33554431 [pps/s]	RWE
6	Max. motor run current	Attention: setting motor current too high might permanently damage the motor! The maximum value is 255. This value means 100% of maximum programmable current of the selected motor current range (see axis parameter 179). Current can be adjusted / scaled down by specifying a lower value between 0 and 255. This value is transformed into 32 different internal current settings supported by the hardware (see hardware manual for more details). ***Please note: high current range is available for hardware version V1.4, only!***	0… 255 *Low current range scaling (axis parameter 179 set to 1):* $I_{peak} = <value> \times \frac{1A}{255}$ $I_{RMS} = <value> \times \frac{0.7A}{255}$ *High current range scaling (axis parameter 179 set to 0):* $I_{peak} = <value> \times \frac{2A}{255}$ $I_{RMS} = <value> \times \frac{1.4A}{255}$	RWE

Number	Axis Parameter	Description	Range [Unit]	Acc.
7	Standby current	Current limit after the motor has stopped plus power down delay time (see parameter 214).	0… 255 *(Same range and current scaling as for axis parameter 6)*	RWE
8	Position reached	1 when target position = actual position 0 otherwise	0/1	R
9	Home switch status	The logical state of the home switch.	0/1	R
10	Right limit switch status	The logical state of the (right) limit switch.	0/1	R
11	Left limit switch status	The logical state of the left limit switch (in three switch mode)	0/1	R
12	Right limit switch disable	If set, deactivates the stop function of the right switch (default: right limit switch disabled)	0/1	RWE
13	Left limit switch disable	Deactivates the stop function of the left switch resp. reference switch if set (default: left limit switch disabled).	0/1	RWE
128	Ramp mode	Automatically set when using ROR, ROL, MST and MVP. 0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided. 1: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.	0/1	RW
130	Minimum speed	Ramp generation for acceleration and deceleration begins and ends with this start and stop value.	0… +268.435.454 [pps] Default = 0	RWE
140	Microstep resolution	0 full step 1 half step 2 4 microsteps 3 8 microsteps 4 16 microsteps 5 32 microsteps 6 64 microsteps 7 128 microsteps 8 256 microsteps (default)	0… 8	RWE
161	Double step enable	Every edge of the cycle releases a step/microstep. *It does not make sense to activate this parameter for internal use.* Double step enable can be used with step/dir interface. 0 – double step off 1 – double step on	0/1	RW
162	Chopper blank time	Selects the comparator *blank time*. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. For low current drivers, a setting of 1 or 2 is good.	0… 3	RW
163	Chopper mode	Selection of the chopper mode: 0 – spread cycle 1 – classic const. off time	0/1	RW

Number	Axis Parameter	Description	Range [Unit]	Acc.
164	Chopper hysteresis decrement	Hysteresis decrement setting. This setting determines the slope of the hysteresis during on time and during fast decay time. 0 – fast decrement 3 – very slow decrement	0… 3	RW
165	Chopper hysteresis end	Hysteresis end setting. Sets the hysteresis end value after a number of decrements. Decrement interval time is controlled by axis parameter 164. -3… -1 negative hysteresis end setting 0 zero hysteresis end setting 1… 12 positive hysteresis end setting	-3… 12	RW
166	Chopper hysteresis start	Hysteresis start setting. Please remark, that this value is an offset to the hysteresis end value.	0… 8	RW
167	Chopper off time	The off time setting controls the minimum chopper frequency. An off time within the range of 5µs to 20µs will fit. Off time setting for constant t_{OFF} chopper: N_{CLK}= 12 + 32*t_{OFF} (Minimum is 64 clocks) Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel.	0 / 2… 15	RW
168	smartEnergy current minimum	Sets the lower motor current limit for coolStep™ operation by scaling the max. motor run current value (parameter 6). minimum motor current: 0 – 1/2 of parameter 6 1 – 1/4 of parameter 6	0/1	RW
169	smartEnergy current down step	Sets the number of stallGuard2 readings above the upper threshold necessary for each current decrement of the motor current. Number of stallGuard2 measurements per decrement: Scaling: 0… 3: 32, 8, 2, 1 0: slow decrement 3: fast decrement	0… 3	RW
170	smartEnergy hysteresis	Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased. Hysteresis: (smartEnergy hysteresis value + 1) * 32 Upper stallGuard threshold: (smartEnergy hysteresis start + smartEnergy hysteresis + 1) * 32	0… 15	RW
171	smartEnergy current up step	Sets the current increment step. The current becomes incremented for each measured stallGuard2 value below the lower threshold (see smartEnergy hysteresis start). current increment step size: Scaling: 0… 3: 1, 2, 4, 8 0: slow increment 3: fast increment / fast reaction to rising load	1… 3	RW

Number	Axis Parameter	Description	Range [Unit]	Acc.
172	smartEnergy hysteresis start	The lower threshold for the stallGuard2 value (see smart Energy current up step).	0… 15	RW
173	stallGuard2™ filter enable	Enables the stallGuard2 filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps. *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	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. 0 Indifferent value 1… 63 less sensitivity -1… -64 higher sensitivity	-64… 63	RW
175	Slope control high side	Determines the slope of the motor driver outputs. *Set to 2 or 3 for this module or rather use the default value.* 0: lowest slope 3: fastest slope	0… 3	RW
176	Slope control low side	Determines the slope of the motor driver outputs. *Set identical to slope control high side.*	0… 3	RW
177	short protection disable	0: Short to GND protection is on 1: Short to GND protection is disabled *Use default value!*	0/1	RW
178	Short detection timer	0: 3.2µs 1: 1.6µs 2: 1.2µs 3: 0.8µs *Use default value!*	0..3	RW
179	VSENSE	sense resistor voltage based current scaling 0: high current range up-to 1.4A RMS / 2A peak 1: low current range up-to 0.7A RMS / 1A peak (default value) ***Please note: this parameter should not and cannot be changed for hardware V1.2! The high current range is available for hardware V1.4, only!***	0/1	RW
180	smartEnergy actual current	This status value provides the *actual motor current* setting as controlled by coolStep. The value goes up to the max. internal motor run current scaling value (0..31) specified via parameter 6 and down to the smartEnergy current minimum (fraction of max. motor run current set via parameter 168). motor current scaling factor: 0 … 31: 1/32, 2/32, … 32/32 (32/32: maximum current supported by module)	0… 31	RW

Number	Axis Parameter	Description	Range [Unit]	Acc.
181	Stop on stall	Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2 load value reaches zero.	0… +268.435.454 [pps]	RW
182	smartEnergy threshold speed	Above this speed coolStep will be enabled.	0… +268.435.454 [pps]	RW
183	smartEnergy slow run current	Sets the motor current which is used below the threshold speed.	0… 255 *(Same range and current scaling as for axis parameter 6)*	RW
193	Ref. search mode	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 Additional functions: - *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 mode 1… 4).*	1… 8	RWE
194	Referencing search speed	For the reference search this value directly specifies the search speed.	-268.435.455… +268.435.454 [pps]	RWE
195	Referencing switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.	-268.435.455… +268.435.454 [pps]	RWE
196	End switch distance	This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3).	0… +268.435.454	R
197	Last reference position	Reference search: the last position before setting the counter to zero can be read out.	-2^{31}… $2^{31}-1$ [µsteps]	R
200	Boost current	Current used for acceleration and deceleration phases. If set to 0 the same current as set by axis parameter 6 will be used.	0… 255 *(Same range and current scaling as for axis parameter 6)*	RWE
204	Freewheeling delay	Time after which the power to the motor will be cut when its velocity has reached zero.	0… 65535 0 = never [msec]	RWE
206	Actual load value	Readout of the actual load value used for stall detection.	0… 1023	R

Number	Axis Parameter	Description	Range [Unit]	Acc.
208	TMC262 driver error flags	Bit 0: stallGuard™ status (1: threshold reached) Bit 1: Overtemperature (1: driver is shut down due to overtemperature) Bit 2: Pre-warning overtemperature (1: threshold is exceeded) Bit 3: Short to ground A (1: short condition detected, driver currently shut d□□n) Bi□ 4: Short to ground B (1: short condition detected, driver currently shut down) Bit 5: Open load A (1: no chopper event has happened during the last period with constant coil polarity) Bit 6: Open load B (1: no chopper event has happened during the last period with constant coil polarity) Bit 7: Always set (1: no step impulse occurred on the step input during the last 2^20 clock cycles) *Please refer to the TMC262 Datasheet for more information.*	0/1	R
209	Encoder position	The value of an encoder register can be read out or written.	[encoder steps] 32bit counter value	RW
210	Encoder prescaler	Prescaler for the encoder.	See paragraph 6.1	RWE
212	Maximum encoder deviation	When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero.	0… 65535 [encoder steps]	RWE
214	Power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec).	1… 65535 [10msec]	RWE
215	Absolute encoder value	Absolute value of the encoder.	0… 1023 [encoder steps]	R
254	Step/Dir mode	0: Turn OFF step/dir mode 1: Use of the ENABLE input on step/dir connector to switch between hold current and run current (no automatic switching) 2: Automatic switching between hold and run current: after the first step pulse the module automatically switches over to run current, and a configurable time after the last step pulse the module automatically switches back to hold current. The ENABLE input on the step/dir connector does not have any functionality. 3: Always use run current, never switch to hold current. The ENABLE input on the step/dir connector does not have any functionality. 4: Automatic current switching like (2), but the ENABLE input is used to switch the driver stage completely off or on. 5: Always use run current like (3), but the ENABLE pin is used to switch the driver stage completely off or on.	0… 5	RWE

4.1 Velocity Calculation

The axis parameters listed below are related to the speed of the motor. The table is an excerpt of the complete table of axis parameters in this chapter.

The unit of the velocity *<value>* is pulse per second (pps). For calculating the speed it is necessary to set the microstep resolution of the driver (axis parameter 140) first. Further, the fullsteps of the motor must be given. Now, calculate as follows:

$$rounds\ per\ second\ (rps) = \frac{< value >}{\text{microstep resolution of driver} \ * \ \text{fullsteps of motor}}$$

$$rounds\ per\ minute\ (rpm) = \text{rps} * 60$$

4.2 stallGuard2

The module is equipped with TMC262 motor driver chip. The TMC262 features load measurement that can be used for stall detection. stallGuard2 delivers a sensorless load measurement of the motor as well as a stall detection signal. The measured value changes linear with the load on the motor in a wide range of load, velocity and current settings. At maximum motor load the stallGuard2 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 when the load gets too high. It is configured by axis parameter #174.

Stall detection can also be used for finding the reference point. Do not use RFS in this case.

4.3 coolStep Related Axis Parameters

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

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

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

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

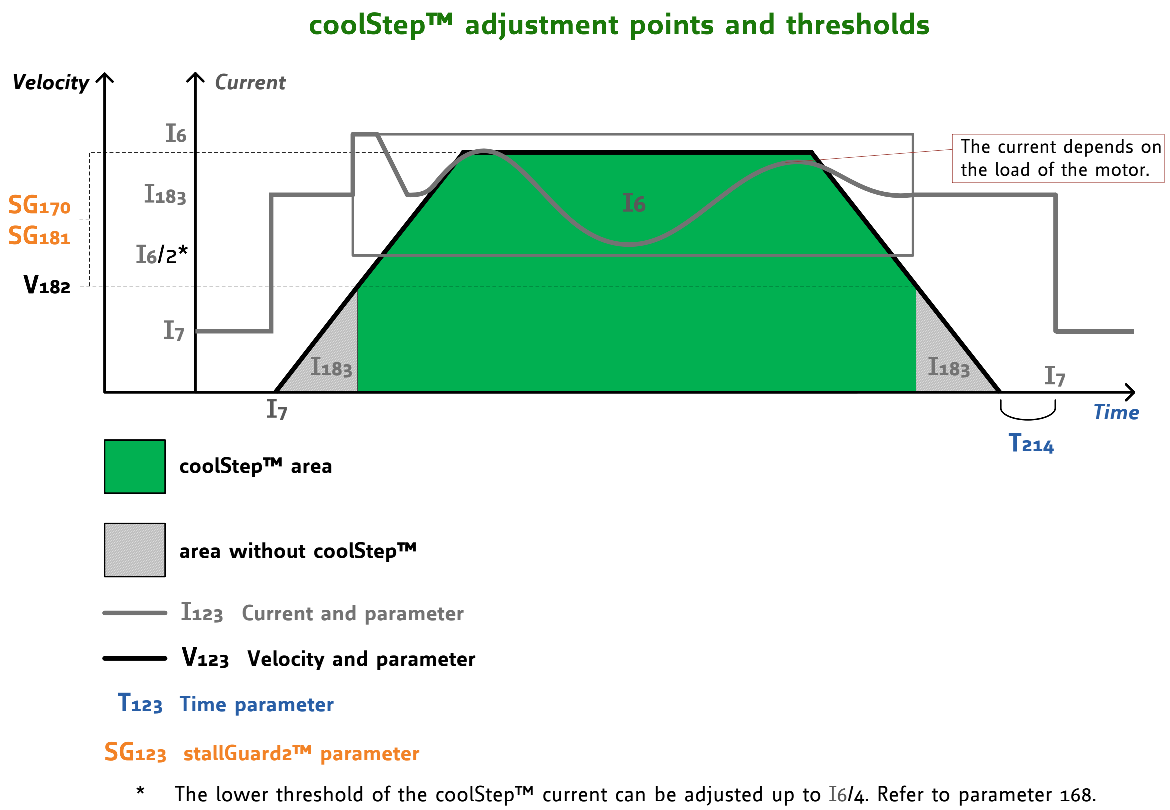


Figure 4.1: coolStep adjustment points and thresholds

Number	Axis parameter	Description
I6	absolute max. current (CS / Current Scale)	The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps (0… 255 divided by eight; e.g. step 0 = 0… 7, step 1 = 8… 15 and so on). *The most important motor setting, since too high values might cause motor damage!*
I7	standby current	The current limit two seconds after the motor has stopped.
I168	smartEnergy current minimum (SEIMIN)	Sets the lower motor current limit for coolStep™ operation by scaling the CS (Current Scale, see axis parameter 6) value. 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 / fast reaction to rising load
I183	smartEnergy slow run current	Sets the motor current which is used below the threshold speed. Please adjust the threshold speed with axis parameter 182.
SG170	smartEnergy hysteresis	Sets the distance between the lower and the upper threshold for stallGuard2™ reading. Above the upper threshold the motor current becomes decreased.
SG181	stop on stall	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 (value equates 2000msec).

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

5 Global parameters

Global parameters are grouped into 4 banks:

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

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

5.1 Bank 0

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

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

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

Meaning of the letters in column *Access*:

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

The TMCM-1021 does not support parameters 0… 38. They are used for modules which address more than one motor.

Number	Global parameter	Description	Range	Access
64	EEPROM magic	Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration.	0… 255	RWE
65	RS485 baud rate	0: 9600 baud (default) 1: 14400 baud 2: 19200 baud 3: 28800 baud 4: 38400 baud 5: 57600 baud 6: 76800 baud *Not supported by Windows!* 7: 115200 baud	0… 7	RWE
66	serial address	The module (target) address (default: 1) *Please note: address 0 has a special meaning. This address is accepted by all modules regardless of their particular address setting. Sending a command with this address might be useful in case a module has been set to an unknown address.*	0… 255	RWE

Number	Global parameter	Description	Range	Access
68	serial heartbeat	Serial heartbeat for the RS485 interface. If this time limit is up and no further command is noticed the motor will be stopped. 0 – parameter is disabled	[ms]	RWE
73	configuration EEPROM lock flag	Write: 1234 to lock the EEPROM, 4321 to unlock it. Read: 1=EEPROM locked, 0=EEPROM unlocked.	0/1	RWE
75	telegram pause time	Pause time before the reply is sent. For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin).	0… 255	RWE
76	serial host address	Host address used in the reply telegrams sent back via RS485 (default: 2).	0… 255	RWE
77	auto start mode	0: do not start TMCL application after power up (default). 1: start TMCL application automatically after power up.	0/1	RWE
79	End switch polarity	0: normal polarity 1: reverse polarity	0/1	RWE
81	TMCL code protection	Protect a TMCL program gainst disassembling or overwriting. 0 – no protection 1 – protection against disassembling 2 – protection against overwriting 3 – protection against disassembling and overwriting ***If you switch off the protection against disassembling, the program will be erased first!***	0,1,2,3	RWE
84	coordinate storage	0 – coordinates are stored in the RAM only (but can be copied explicitly between RAM and EEPROM) 1 – coordinates are always stored in the EEPROM only	0 or 1	RWE
87	Serial secondary address	Second module (target) address. This is the group or broadcast address of the module. Using this address a single command e.g. ROR or MVP sent by the master is sufficient in order to initiate a movement of several (group) or even all (broadcast) modules connected to one bus. The first serial address (66) might then still be used to set parameters of the modules individually. In order to avoid bus collisions, the module will **not** sent a reply for commands with this address.	0… 255	RWE
128	TMCL application status	0 – stop 1 – run 2 – step 3 – reset	0… 3	R
129	download mode	0 – normal mode 1 – download mode	0/1	R
130	TMCL program counter	The index of the currently executed TMCL instruction.		R
132	tick timer	A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value.		RW
133	random number	Choose a random number.	0… 147483647	R

5.2 Bank 1

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

5.3 Bank 2

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

Meaning of the letters in column *Access*:

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

Number	Global parameter	Description	Range	Access
0	general purpose variable #0	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
1	general purpose variable #1	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
2	general purpose variable #2	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
3	general purpose variable #3	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
4	general purpose variable #4	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
5	general purpose variable #5	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
6	general purpose variable #6	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
7	general purpose variable #7	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
8	general purpose variable #8	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
9	general purpose variable #9	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
10	general purpose variable #10	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
11	general purpose variable #11	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
12	general purpose variable #12	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
13	general purpose variable #13	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
14	general purpose variable #14	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
15	general purpose variable #15	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
16	general purpose variable #16	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
17	general purpose variable #17	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
18	general purpose variable #18	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
19	general purpose variable #19	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
20… 55	general purpose variables #20… #55	for use in TMCL applications	-2^{31}… $+2^{31}$	RWE
56… 255	general purpose variables #56… #255	for use in TMCL applications	-2^{31}… $+2^{31}$	RW

5.4 Bank 3

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

Meaning of the letters in column *Access*:

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

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

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

6 Hints and Tips

This chapter gives some hints and tips on using the functionality of TMCL™, for example how to use and parameterize the built-in reference point search algorithm or the incremental sensOstep™ encoder. Further you will find basic information about stallGuard2™ and coolStep™.

6.1 Reference Search

The built-in reference search features switching point calibration and support of one or two reference switches. The internal operation is based on a state machine that can be started, stopped and monitored (instruction RFS, no. 13, chapter 3.5.13). The settings of the automatic stop functions corresponding to the switches (axis parameters 12 and 13) have no influence on the reference search.

Please note:

- Until the reference switch is found for the first time, the searching speed is identical to the maximum positioning speed (axis parameter 4), unless reduced by axis parameter 194.
- After hitting the reference switch, the motor slowly moves 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. This low calibrating speed is a quarter of the maximum positioning speed by default (axis parameter 195).
- The reference switch is connected in series with the left limit switch. The differentiation between the left limit switch and the home switch is made through software. Switches with open contacts (normally closed) are used.

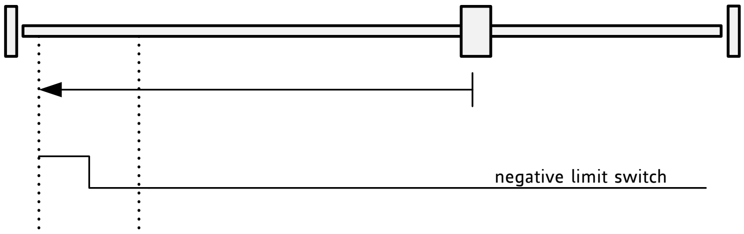
Choose one of these values for axis parameter 193:

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

Adding 128 to these values reverses the polarity of the home switch input.

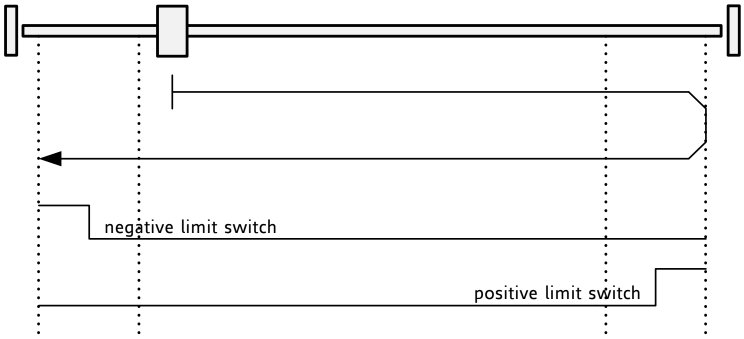
The next two pages show all possible modes of reference search according to the specific commands on top of each drawing.

SAP 193, 0, 1



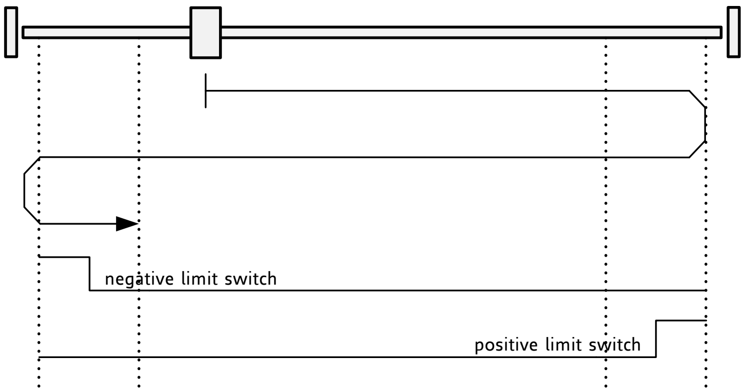
Search left stop switch only.

SAP 193, 0, 2



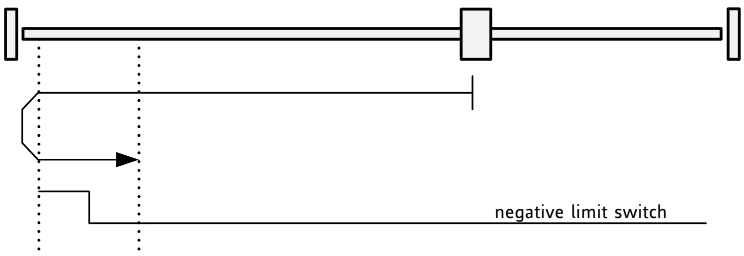
Search right stop switch, then search left stop switch.

SAP 193, 0, 3



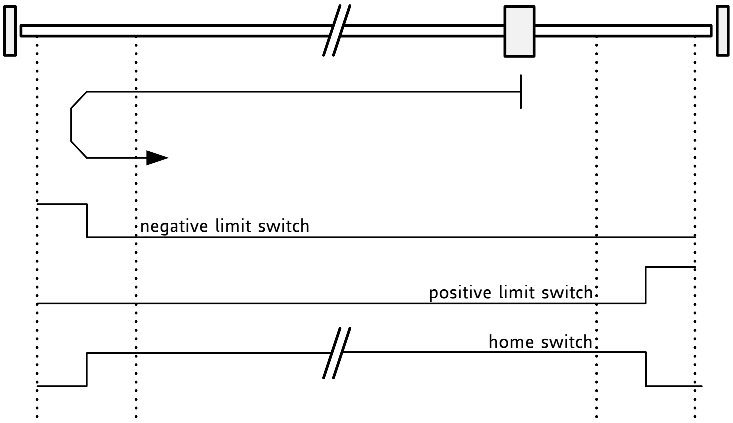
Search right stop switch, then search left stop switch from both sides.

SAP 193, 0, 4



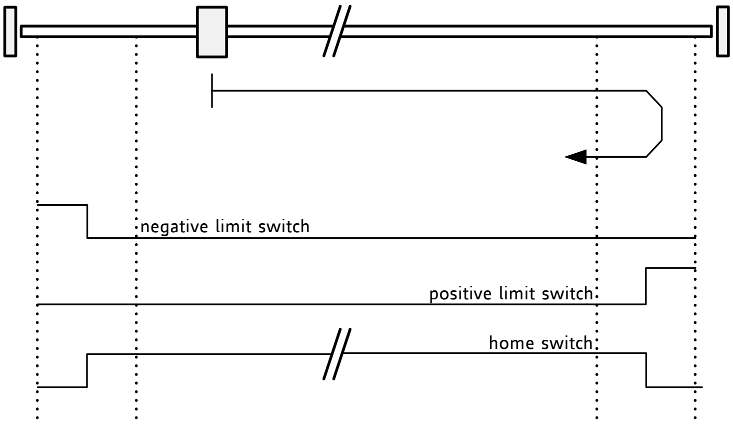
Search left stop switch from both sides.

SAP 193, 0, 5



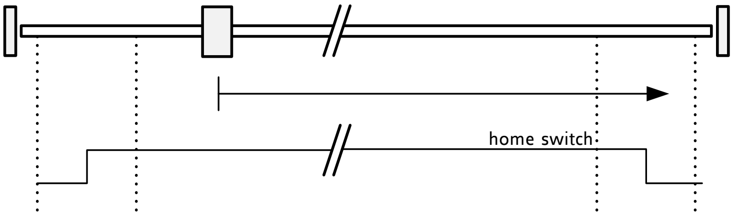
Search home switch in negative direction, reverse the direction when left stop switch reached.

SAP 193, 0, 6



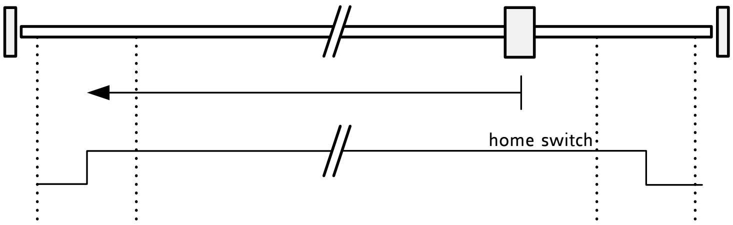
Search home switch in positive direction, reverse the direction when right stop switch reached.

SAP 193, 0, 7



Search home switch in positive direction, ignore end switches.

SAP 193, 0, 8



Search home switch in negative direction, ignore end switches.

6.2 Changing the Prescaler Value of an Encoder

The built-in encoder has 1024 steps per rotation.

For the operation with encoder please consider the following hints:

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

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

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

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

Consider the following formula for your calculation:

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

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

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

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

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

6.3 Using the RS485 Interface

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

7 TMCL Programming Techniques and Structure

7.1 Initialization

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

7.2 Main Loop

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

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

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

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

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

7.3 Using Symbolic Constants

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

Example:

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

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

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

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

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

7.4 Using Variables

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

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

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

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

Example:

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

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

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

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

7.5 Using Subroutines

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

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

7.6 Mixing Direct Mode and Standalone Mode

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

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

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

Example:

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

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

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

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

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

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

8 Life Support Policy

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

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

9 Revision History

9.1 Document Revision

Version	Date	Author	Description
0.90	2011-AUG-25	SD	Preliminary version
0.91	2011-AUG-31	GE	- Table with axis and global parameters now centralized in dedicated chapters - RFS function added
1.00	2012-MAR-09	SD	First complete version. - Axis parameters 130 and 200 added. - Axis parameter 254 updated
1.01	2012-MAR-13	SD	Axis parameter 254 corrected
1.02	2012-MAY-20	SD	Minor changes
1.03	2012-MAY-31	SD	JC instruction: *type* corrected
1.04	2012-JUL-30	SD	Global parameter 79 added
1.05	2013-APR-12	JP	Firmware revisions updated
1.06	2014-APR-14	SD	- Axis parameter 87 added. - Axis parameter 197 added. - Axis parameter 193 updated: reference mode moving directions can be inverted now. - Firmware revisions updated. - Chapter 7 added.
1.07	2014-SEP-30	GE	- Axis parameter 179 added for hardware version V1.4

Figure 8.1 Document revision

9.2 Firmware Revision

Version	Date	Author	Description
1.09	2011-AUG-08	OK	First version
1.10	2011-AUG-23	OK	Several enhancements
1.15	2012-MAR-09	OK	Several enhancements
1.19	2012-JUL-25	OK	Global parameter 79 added
1.20	2012-OCT-04	OK	Second Address available for RS232/RS485 (global parameter 87). Answer suppressed for messages to 2nd address.
1.21	2012-DEC-07	OK	- Bug fix: stored axis parameter 140 correctly loaded at startup. - Bug fix: oscillations around target position at minimum velocity >0 (parameter 130) fixed.
1.22	2013-FEB-23	OK	- Axis parameter 193 updated: add 64 in order to invert the reference mode moving direction. - Bug fix (reference modes 7 and 8 / axis parameter 193): end switches are generally deactivated.
1.25	2013-MAR-13	OK	Not deployed for TMCM-1021
1.26	2013-APR-04	OK	Internal clock frequency increased to enable RS485 bit rates between 9600 and 115200.
1.27	2013-AUG-30	OK	Error correction related to magnetic encoder. Update rate increased.
1.28	2013-SEP-24	OK	Not deployed for TMCM-1021
1.29	2014-JAN-14	OK	- Firmware version 1.29 is intended for TMCM-1021 module versions from 1.0 to 1.4. *Earlier firmware versions are not compatible with TMCM-1021 V1.4. boards!* - Axis parameter 197 added.

Figure 8.2 Firmware revision

10 References

[TMC262]	TMC262 Datasheet
[TMCL-IDE]	TMCL-IDE User Manual
[QSH2818]	QSH2818 Manual

Please refer to www.trinamic.com.