



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

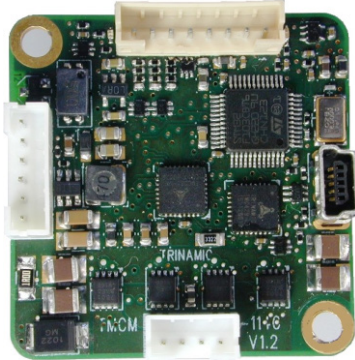
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



Firmware Version V1.27

TMCL™ FIRMWARE MANUAL

+



+

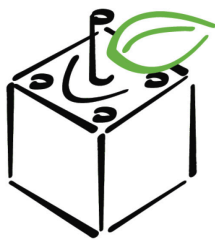
+

TMCM-1140

1-Axis Stepper
Controller / Driver
2 A / 24 V
sens0step™ Encoder
USB, RS485, and CAN

+

UNIQUE FEATURES:



coolStep™

stallGuard²™

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

www.trinamic.com



TRINAMIC
MOTION CONTROL

Table of Contents

1	Features.....	4
2	Putting the Module into Operation	6
2.1	Basic Set-Up	6
2.1.1	Start the TMCL-IDE Software Development Environment.....	8
2.2	Using TMCL Direct Mode.....	9
2.2.1	Important Motor Settings.....	10
2.3	Testing with a Simple TMCL Program.....	11
3	TMCL and the TMCL-IDE: Introduction	12
3.1	Binary Command Format.....	12
3.1.1	Checksum Calculation	13
3.2	Reply Format.....	13
3.2.1	Status Codes.....	14
3.3	Standalone Applications.....	14
3.4	TMCL Command Overview.....	15
3.4.1	TMCL Commands.....	15
3.4.2	Commands Listed According to Subject Area	16
3.5	The ASCII Interface.....	20
3.5.1	Format of the Command Line.....	20
3.5.2	Format of a Reply	20
3.5.3	Configuring the ASCII Interface	21
3.6	Commands.....	22
3.6.1	ROR (rotate right).....	22
3.6.2	ROL (rotate left)	23
3.6.3	MST (motor stop).....	24
3.6.4	MVP (move to position)	25
3.6.5	SAP (set axis parameter).....	27
3.6.6	GAP (get axis parameter)	28
3.6.7	STAP (store axis parameter)	29
3.6.8	RSAP (restore axis parameter).....	30
3.6.9	SGP (set global parameter)	31
3.6.10	GGP (get global parameter).....	32
3.6.11	STGP (store global parameter)	33
3.6.12	RSGP (restore global parameter)	34
3.6.13	RFS (reference search).....	35
3.6.14	SIO (set input / output).....	36
3.6.15	GIO (get input /output)	38
3.6.16	CALC (calculate).....	40
3.6.17	COMP (compare).....	41
3.6.18	JC (jump conditional)	42
3.6.19	JA (jump always)	43
3.6.20	CSUB (call subroutine)	44
3.6.21	RSUB (return from subroutine).....	45
3.6.22	WAIT (wait for an event to occur).....	46
3.6.23	STOP (stop TMCL program execution)	47
3.6.24	SCO (set coordinate).....	48
3.6.25	GCO (get coordinate).....	49
3.6.26	CCO (capture coordinate)	50
3.6.27	ACO (accu to coordinate)	51
3.6.28	CALCX (calculate using the X register)	52
3.6.29	AAP (accumulator to axis parameter).....	53
3.6.30	AGP (accumulator to global parameter).....	54
3.6.31	CLE (clear error flags).....	55
3.6.32	VECT (set interrupt vector).....	56
3.6.33	EI (enable interrupt).....	57
3.6.34	DI (disable interrupt).....	58
3.6.35	RETI (return from interrupt)	59

3.6.36	Customer Specific TMCL Command Extension (UF0... UF7 / User Function)	60
3.6.37	Request Target Position Reached Event.....	60
3.6.38	TMCL Control Functions.....	61
4	Axis Parameters	63
4.1	stallGuard2.....	70
4.2	coolStep Related Axis Parameters	70
5	Global Parameters.....	72
5.1	Bank 0.....	72
5.2	Bank 1.....	74
5.3	Bank 2.....	74
5.4	Bank 3.....	75
6	Hints and Tips	76
6.1	Reference Search	76
6.2	Changing the Prescaler Value of an Encoder.....	79
6.3	Using the RS485 Interface	80
7	TMCL Programming Techniques and Structure	81
7.1	Initialization	81
7.2	Main Loop.....	81
7.3	Using Symbolic Constants	81
7.4	Using Variables	82
7.5	Using Subroutines.....	82
7.6	Mixing Direct Mode and Standalone Mode.....	83
8	Life Support Policy.....	84
9	Revision History.....	85
9.1	Firmware Revision	85
9.2	Document Revision.....	85
10	References	86

1 Features

The TMCM-1140 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 NEMA 17 (42mm flange size) stepper motors and has been designed for coil currents up to 2 A RMS and 24 V DC supply voltage. With its high energy efficiency from TRINAMIC's coolStep™ technology cost for power consumption is kept down. The TMCL™ firmware allows for both, standalone operation and direct mode.

MAIN CHARACTERISTICS

Motion controller

- 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

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

Interfaces

- RS485 2-wire communication interface
- CAN 2.0B communication interface
- USB full speed (12Mbit/s) device interface
- 4 multipurpose inputs:
 - 3x general-purpose digital inputs
(Alternate functions: STOP_L / STOP_R / HOME switch inputs or A/B/N encoder input)
 - 1x dedicated analog input
- 2 general purpose outputs
 - 1x open-drain 1A max.
 - 1x +5V supply output (can be switched on/off in software)

Software

- TMCL: standalone operation or remote controlled operation, program memory (non volatile) for up to 2048 TMCL commands, and PC-based application development software TMCL-IDE available for free.

Electrical and mechanical data

- Supply voltage: +24 V DC nominal (9... 28 V DC)
- Motor current: up to 2 A RMS / 2.8 A peak (programmable)

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.

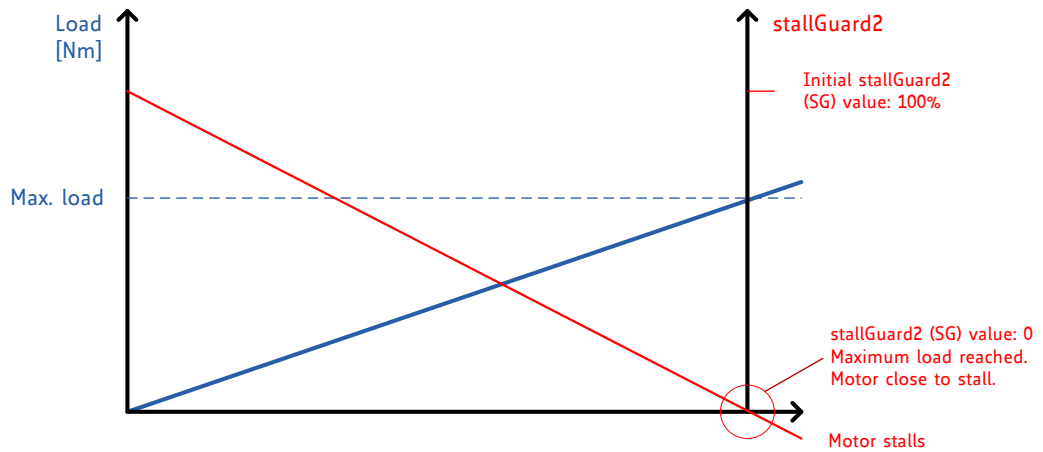


Figure 1.1 stallGuard2 load measurement SG as a function of load

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.

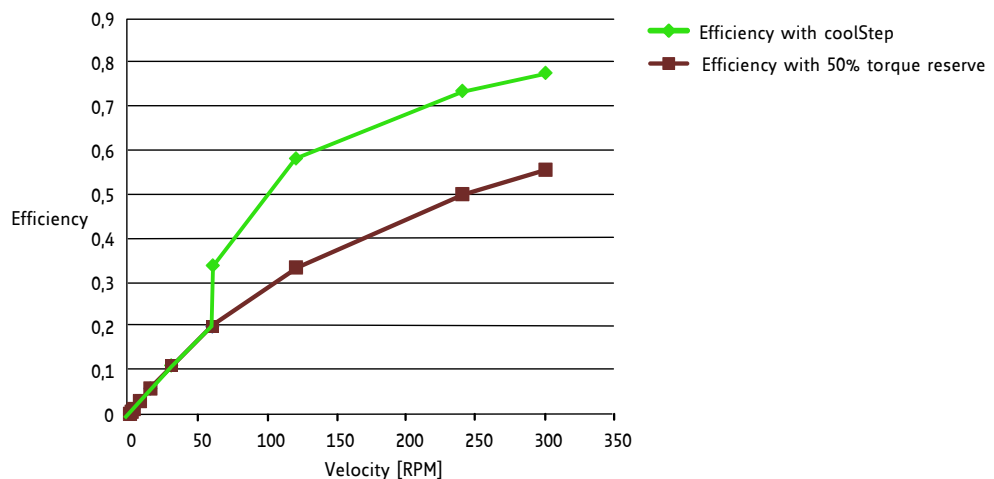


Figure 1.2 Energy efficiency example with coolStep

2 Putting the Module into Operation

Here you can find basic information for putting your TMCM-1140 into operation. If you are already common with TRINAMICs modules you may skip this chapter.

The things you need:

- TMCM-1140
- Interface (RS485/CAN/USB) suitable to your module with cables
- Nominal supply voltage +24V DC for your module
- TMCL-IDE program and PC
- Stepper motor

PRECAUTIONS

Do not connect or disconnect the TMCM-1140 while powered!
Do not connect or disconnect the motor while powered!
Do not exceed the maximum power supply voltage of 28 V 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.

CONNECTING THE MODULE

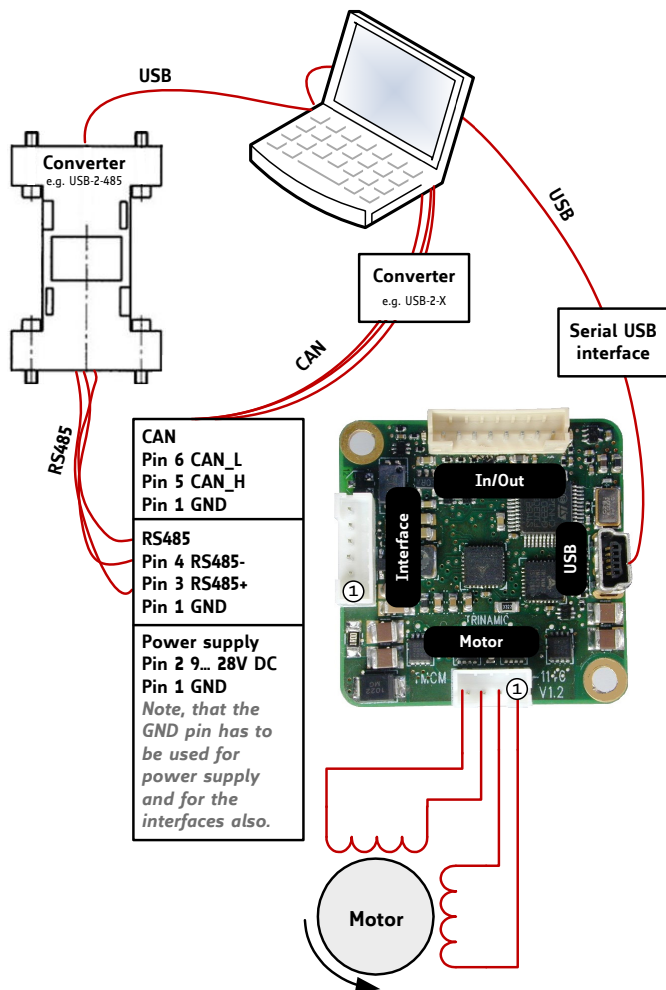


Figure 2.1: Starting up

1. Connect power supply and choose your interface

a) Connect CAN or RS485 and power supply

CAN interface will be de-activated in case USB is connected due to internal sharing of hardware resources.

Pin	Label	Description
1	GND	System and signal ground
2	VDD	VDD (+9V...+28V)
3	RS485+	RS485 interface, diff. signal (non-inverting)
4	RS485-	RS485 interface, diff. signal (inverting)
5	CAN_H	CAN interface, diff. signal (non-inverting)
6	CAN_L	CAN interface, diff. signal (inverting)

b) Connect USB interface (as alternative to CAN and RS485; use a normal USB cable)

Download and install the file *TMCM-1140.inf* (www.trinamic.com).

Pin	Label	Description
1	VBUS	+5V power
2	D-	Data -
3	D+	Data +
4	ID	ground
5	GND	ground

2. Connect In/Out connector

If you like to work with the GPIOs or switches, use the In/Out connector.

Pin	Label	Description
1	GND	System and signal ground
2	VDD	VDD, connected to VDD pin of the power and communication connector
3	OUT_1	Open-drain output (max. 1A) Integrated freewheeling diode to VDD
4	OUT_0	+5V supply output (max. 100mA) Can be switched on/off in software
5	AIN_0	Dedicated analog input, Input voltage range: 0..+10V Resolution: 12bit (0..4095)
6	IN_0, STOP_L, ENC_A	General purpose digital input (+24V compatible)
		Alternate function 1: left stop switch input
		Alternate function 2: external incremental encoder channel A input
7	IN_1, STOP_R, ENC_B	General purpose digital input (+24V compatible)
		Alternate function 1: right stop switch input
		Alternate function 2: external incremental encoder channel B input
8	IN_2, HOME, ENC_N	General purpose digital input (+24V compatible)
		Alternate function 1: home switch input
		Alternate function 2: external incremental encoder index / zero channel input

3. Connect the 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

4. **Switch ON the power supply**

Turn power ON. The green LED for power lights up 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.

2.1.1 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**.

USB interface

If the file *TMCM-1140.inf* is installed correctly, the module will be identified automatically.

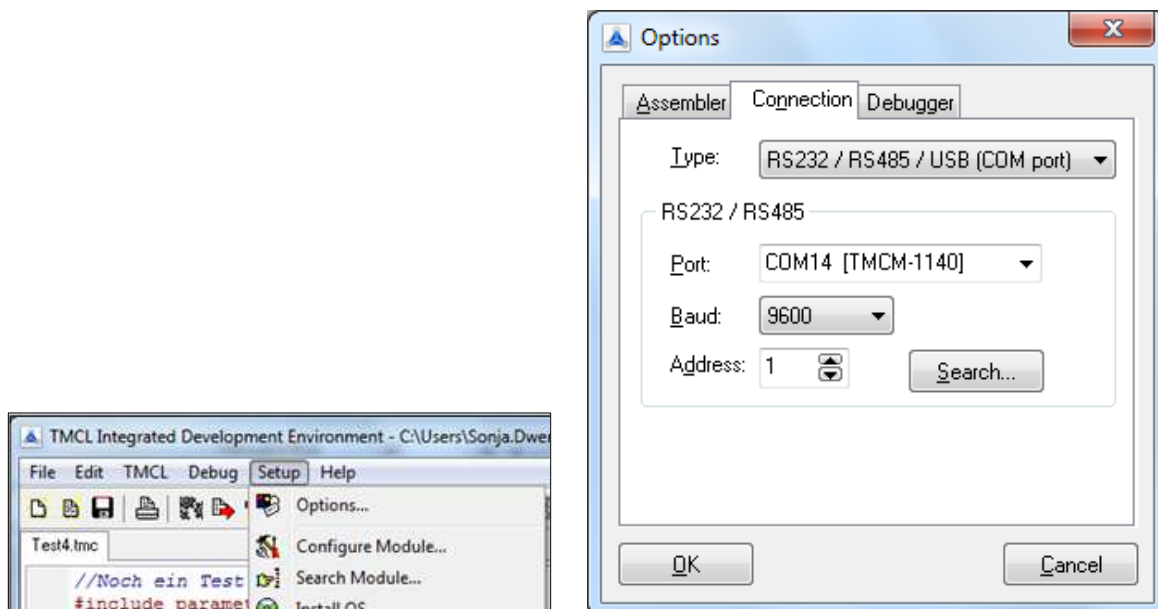


Figure 2.2 Setup dialogue and connection tab of the TMCL-IDE.

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

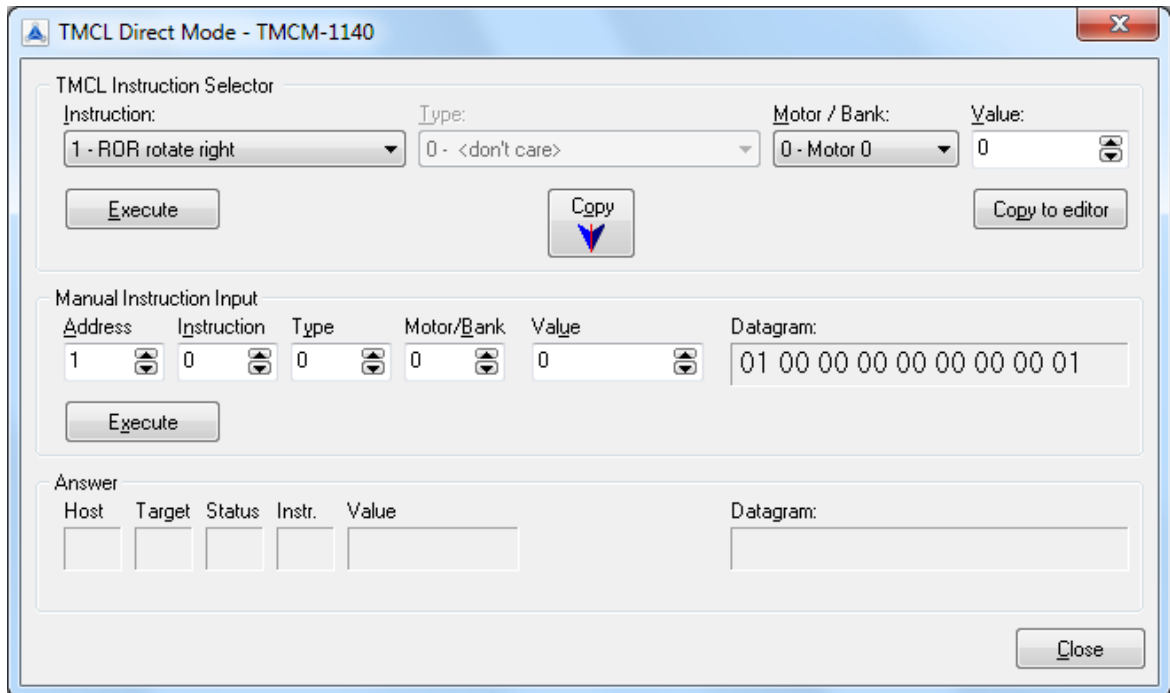
2.2 Using TMCL Direct Mode

1. Start TMCL *Direct Mode*.



↑
Direct Mode

2. If the communication is established the TMCM-1140 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 500 -> Click *Execute*. The motor is rotating now.
- MST motor stop, motor 0 -> Click *Execute*. The 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:
Chapter 4 of this manual (axis parameters) includes a diagram which points out the coolStep related axis parameters and their functions.

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 3.6.5. You can use the TMCL-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	Should not exceed the physically highest possible value. Adjust the pulse divisor (axis parameter 154), if the speed value is very low (<50) or above the upper limit.	0... 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{pulse_divisor}} \frac{\mu\text{steps}}{\text{sec}} \right]$																																								
5	Maximum acceleration	The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 429 datasheet for calculation of physical units.	0... 2047* ¹																																								
6	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. <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <tr><td>0... 7</td><td>79...87</td><td>160... 167</td><td>240... 247</td></tr> <tr><td>8... 15</td><td>88... 95</td><td>168... 175</td><td>248... 255</td></tr> <tr><td>16... 23</td><td>96... 103</td><td>176... 183</td><td></td></tr> <tr><td>24... 31</td><td>104... 111</td><td>184... 191</td><td></td></tr> <tr><td>32... 39</td><td>112... 119</td><td>192... 199</td><td></td></tr> <tr><td>40... 47</td><td>120... 127</td><td>200... 207</td><td></td></tr> <tr><td>48... 55</td><td>128... 135</td><td>208... 215</td><td></td></tr> <tr><td>56... 63</td><td>136... 143</td><td>216... 223</td><td></td></tr> <tr><td>64... 71</td><td>144... 151</td><td>224... 231</td><td></td></tr> <tr><td>72... 79</td><td>152... 159</td><td>232... 239</td><td></td></tr> </table> <p><i>The most important motor setting, since too high values might cause motor damage!</i></p>	0... 7	79...87	160... 167	240... 247	8... 15	88... 95	168... 175	248... 255	16... 23	96... 103	176... 183		24... 31	104... 111	184... 191		32... 39	112... 119	192... 199		40... 47	120... 127	200... 207		48... 55	128... 135	208... 215		56... 63	136... 143	216... 223		64... 71	144... 151	224... 231		72... 79	152... 159	232... 239		0... 255 $I_{\text{peak}} = < \text{value} > \times \frac{2.8\text{A}}{255}$ $I_{\text{RMS}} = < \text{value} > \times \frac{2\text{A}}{255}$
0... 7	79...87	160... 167	240... 247																																								
8... 15	88... 95	168... 175	248... 255																																								
16... 23	96... 103	176... 183																																									
24... 31	104... 111	184... 191																																									
32... 39	112... 119	192... 199																																									
40... 47	120... 127	200... 207																																									
48... 55	128... 135	208... 215																																									
56... 63	136... 143	216... 223																																									
64... 71	144... 151	224... 231																																									
72... 79	152... 159	232... 239																																									
7	Standby current	The current limit two seconds after the motor has stopped.	0... 255 $I_{\text{peak}} = < \text{value} > \times \frac{2.8\text{A}}{255}$ $I_{\text{RMS}} = < \text{value} > \times \frac{2\text{A}}{255}$																																								
140	Microstep resolution	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>0</td><td>full step</td></tr> <tr><td>1</td><td>half step</td></tr> <tr><td>2</td><td>4 microsteps</td></tr> <tr><td>3</td><td>8 microsteps</td></tr> <tr><td>4</td><td>16 microsteps</td></tr> <tr><td>5</td><td>32 microsteps</td></tr> <tr><td>6</td><td>64 microsteps</td></tr> <tr><td>7</td><td>128 microsteps</td></tr> <tr><td>8</td><td>256 microsteps</td></tr> </table>	0	full step	1	half step	2	4 microsteps	3	8 microsteps	4	16 microsteps	5	32 microsteps	6	64 microsteps	7	128 microsteps	8	256 microsteps	0... 8																						
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																																										

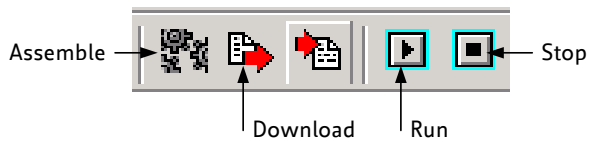
*1 Unit of acceleration: $\frac{16\text{MHz}^2}{536870912 \cdot 2^{\text{puls_divisor} + \text{ramp_divisor}}} \frac{\text{microsteps}}{\text{sec}^2}$

2.3 Testing with a Simple TMCL Program

Type in the following program:

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

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



1. Click the **Assemble** icon to convert the TMCL program into binary code.
2. Then download the program to the TMCM-1140 module by clicking the **Download** icon.
3. Click the **Run** icon. 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-1140 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-1140 supports TMCL direct mode (binary commands) and standalone TMCL program execution. You can store up to 2048 TMCL instructions on it. In direct mode and most cases the TMCL communication over RS485, CAN, or USB follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCM-1140. 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/CAN/USB 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 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In this case it consists of nine bytes.

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

The binary command format for R485/USB is as follows:

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

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

3.1.1 Checksum Calculation

As mentioned above, the checksum is calculated by adding up all bytes (including the module address byte) using 8-bit addition. Here are two examples 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/ USB is as follows:

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

- The checksum is also calculated by adding up all the other bytes using an 8-bit addition.
- When using CAN bus, just leave out the first byte (module address) and the last byte (checksum).
- 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 a TMCL memory for storing TMCL applications. You can use TMCL-IDE for developing standalone TMCL applications. You can download a program into the EEPROM and afterwards it will run on the module. The TMCL-IDE contains an editor and the TMCL assembler where the commands can be entered using their mnemonic format. They will be assembled automatically into their binary representations. Afterwards this code can be downloaded into the module to be executed there.

3.4 TMCL Command Overview

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
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 included 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	Stall (stallGuard2)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change
41	IN_2 change
42	IN_3 change
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 3.6.9) 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

EXAMPLE FOR THE USE OF A TIMER INTERRUPT:

```

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

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

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

```

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

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

Please also take a look at the other example programs.

3.5 The ASCII Interface

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

THE FOLLOWING COMMANDS CAN BE USED IN ASCII MODE:

ROL, ROR, MST, MVP, SAP, GAP, STAP, RSAP, SGP, GGP, STGP, RSGP, RFS, SIO, GIO, SCO, GCO, CCO, UF0, UF1, UF2, UF3, UF4, UF5, UF6, and UF7.

Note:

Only direct mode commands can be entered in ASCII mode!

SPECIAL COMMANDS WHICH ARE ONLY AVAILABLE IN ASCII MODE:

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

ENTERING AND LEAVING ASCII MODE:

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

3.5.1 Format of the Command Line

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

EXAMPLES FOR VALID COMMAND LINES:

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

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

3.5.2 Format of a Reply

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

The reply consists of:

- the address character of the host (host address that can be set in the module)
- the address character of the module
- the status code as a decimal number
- the return value of the command as a decimal number
- a <CR> character

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

3.5.3 Configuring the ASCII Interface

The module can be configured so that it starts up either in binary mode or in ASCII mode. **Global parameter 67 is used for this purpose** (please see also chapter 5.1).

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

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

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

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

3.6 Commands

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

3.6.1 ROR (rotate right)

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

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

The module is based on the TMC429 stepper motor controller and the TMC262 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR 0, <velocity>

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 - OK	(don't care)

Example:

Rotate right, velocity = 350

Mnemonic: ROR 0, 350

Binary:

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

3.6.2 ROL (rotate left)

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

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

The module is based on the TMC429 stepper motor controller and the TMC262 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL 0, <velocity>

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 - OK	(don't care)

Example:

Rotate left, velocity = 1200

Mnemonic: ROL 0, 1200

Binary:

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

3.6.3 MST (motor stop)

With this command the motor will be instructed to stop with a soft stop.

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

Related commands: ROL, ROR, SAP, GAP

Mnemonic: MST 0

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 - OK	(don't care)

Example:

Stop motor

Mnemonic: MST 0

Binary:

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

3.6.4 MVP(move to position)

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

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

THREE OPERATION TYPES ARE AVAILABLE:

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

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

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

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

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

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

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

Binary:

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