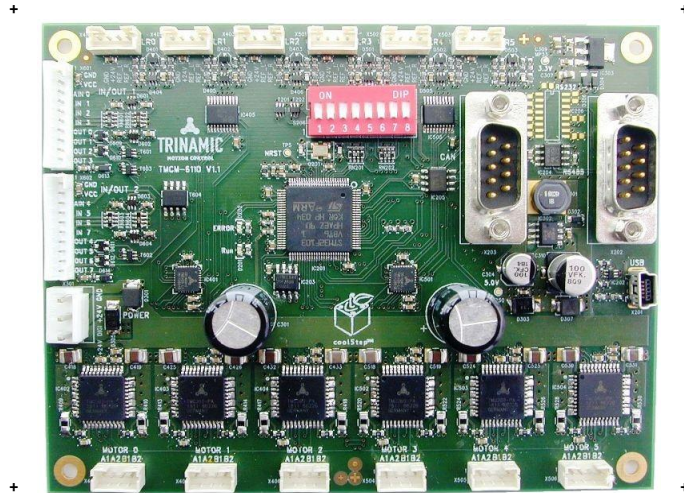


Firmware Version V1.31

TMCL™ FIRMWARE MANUAL



TMCM-6110

6 Axes Stepper
Controller / Driver
1.1A RMS / 24V DC
USB, CAN, RS485 [or RS232]



stallGuard²

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

www.trinamic.com

TRINAMIC
MOTION CONTROL

Table of Contents

1	Features.....	4
2	Overview.....	5
3	Putting the Module into Operation	6
3.1	Basic Set-up.....	6
3.1.1	Connecting the Module	6
3.1.2	Start the TMCL-IDE Software Development Environment	11
3.1.3	Using TMCL Direct Mode	12
3.1.4	Important Motor Settings	13
3.1.5	TMCL Program Example	14
4	TMCL and TMCL-IDE	15
4.1	Binary Command Format	15
4.2	Reply Format.....	16
4.2.1	Status Codes	16
4.3	Standalone Applications.....	16
4.4	TMCL Command Overview	17
4.4.1	TMCL Commands	17
4.4.2	Commands Listed According to Subject Area	18
4.5	The ASCII Interface.....	22
4.5.1	Command Line Format.....	22
4.5.2	Format of a Reply	22
4.5.3	Commands Used in ASCII Mode.....	22
4.5.4	Configuring the ASCII Interface	23
4.6	Commands.....	24
4.6.1	ROR (rotate right)	24
4.6.2	ROL (rotate left)	25
4.6.3	MST (motor stop).....	26
4.6.4	MVP (move to position)	27
4.6.5	SAP (set axis parameter)	29
4.6.6	GAP (get axis parameter)	30
4.6.7	STAP (store axis parameter)	31
4.6.8	RSAP (restore axis parameter)	32
4.6.9	SGP (set global parameter)	33
4.6.10	GGP (get global parameter).....	34
4.6.11	STGP (store global parameter)	35
4.6.12	RSGP (restore global parameter)	36
4.6.13	RFS (reference search).....	37
4.6.14	SIO (set input / output).....	38
4.6.15	GIO (get input /output)	41
4.6.16	CALC (calculate).....	44
4.6.17	COMP (compare).....	45
4.6.18	JC (jump conditional)	46
4.6.19	JA (jump always)	47
4.6.20	CSUB (call subroutine)	48
4.6.21	RSUB (return from subroutine).....	49
4.6.22	WAIT (wait for an event to occur).....	50
4.6.23	STOP (stop TMCL program execution)	51
4.6.24	SCO (set coordinate).....	52
4.6.25	GCO (get coordinate).....	53
4.6.26	CCO (capture coordinate)	54
4.6.27	ACO (accu to coordinate)	55
4.6.28	CALCX (calculate using the X register)	56
4.6.29	AAP (accumulator to axis parameter).....	57
4.6.30	AGP (accumulator to global parameter).....	58
4.6.31	CLE (clear error flags).....	59
4.6.32	VECT (set interrupt vector).....	60
4.6.33	EI (enable interrupt).....	62
4.6.34	DI (disable interrupt).....	64

4.6.35	RETI (return from interrupt)	66
4.6.36	Customer Specific TMCL Command Extension (user function).....	67
4.6.37	Request Target Position Reached Event.....	68
4.6.38	BIN (return to binary mode).....	69
4.6.39	TMCL Control Functions.....	70
5	Axis Parameters	71
5.1	stallGuard2 Related Parameters	78
5.2	coolStep Related Parameters	80
5.3	Reference Search	82
5.3.1	Reference Search Modes (Axis Parameter 193).....	83
5.4	Grouping Motors.....	86
5.5	Calculation: Velocity and Acceleration vs. Microstep- and Fullstep-Frequency	87
5.5.1	Microstep Frequency.....	87
5.5.2	Fullstep Frequency.....	88
6	Global Parameters.....	89
6.1	Bank 0.....	89
6.2	Bank 1.....	92
6.3	Bank 2.....	92
6.4	Bank 3.....	93
7	TMCL Programming Techniques and Structure	94
7.1	Initialization	94
7.2	Main Loop.....	94
7.3	Using Symbolic Constants	94
7.4	Using Variables	95
7.5	Using Subroutines.....	95
7.6	Mixing Direct Mode and Standalone Mode.....	95
8	Life Support Policy.....	97
9	Revision History.....	98
9.1	Firmware Revision	98
9.2	Document Revision	98
10	References	99

1 Features

The TMC6110 is a compact 6-axis stepper motor controller/driver standalone board. It supports up to 6 bipolar stepper motors with up to 1.1A RMS coil current. There are separate motor and reference/end switch connectors for each motor. In addition, the module offers 8 general purpose inputs and 8 general purpose outputs.

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

Interfaces

- Up to 8 multi-purpose inputs (+24V compatible, incl. 2 dedicated analog inputs)
- Up to 8 multi-purpose outputs (open-drain, incl. 2 outputs for currents up to 1A)
- RS485 2-wire communication interface
- USB 2.0 full speed (12Mbit/s) communication interface (mini-USB connector)
- CAN 2.0B communication interface (9pin D-SUB)

Features

- Uses TMC429 stepper motor controller for on-the-fly alteration of many motion specific parameters
- Uses TMC260 advanced stepper motor driver IC
- Up to 256 microsteps per fullstep
- Integrated protection: overtemperature/undervoltage

Software

- TMCL remote (direct mode) and standalone operation (memory for up to 2048 TMCL commands)
- Fully supported by TMCL-IDE (PC based integrated development environment)

Electrical data

- Supply voltage: +9V... +28V DC
- Motor current: up to 1.1A RMS (programmable) per axis

Mechanical data

- Board size: 130mm x 100mm, height 30mm max.
- 4 mounting holes for M3 screws

Please see separate Hardware Manual for additional information

2 Overview

The software running on the microprocessor of the TMC6110 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 firmware of this module is related to the standard TMCL firmware shipped with regard to protocol and commands. Corresponding, this module is based on the TMC429 stepper motor controller and the TMC260 power driver and supports the standard TMCL with a special range of values.

The TMC260 is a new energy efficient high current high precision microstepping driver IC for bipolar stepper motors and offers TRINAMICs patented coolStep™ feature with its special commands. Please mind this technical innovation.

3 Putting the Module into Operation

In this chapter you will find basic information for putting your module into operation. This includes a simple example for a TMCL program and a short description of operating the module in direct mode.

THINGS YOU NEED

- TMCM-6110 with up to six appropriate motors (e.g. QSH4218)
- Interface (RS485, CAN or USB) suitable to your TMCM-6110 module with cables
- Nominal supply voltage +24V DC (+9... +28V DC) for your module
- TMCL-IDE program (can be downloaded free of charge from www.trinamic.com)
- Appropriate cables

PRECAUTIONS

Do not mix up connections or short-circuit pins.
Avoid bounding I/O wires with motor power wires.
Do not exceed the maximum power supply of 28V DC!
Do not connect or disconnect the motor while powered!
START WITH POWER SUPPLY OFF!

3.1 Basic Set-up

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

3.1.1 Connecting the Module

For first steps you will need a power supply and a communication between PC and one of the serial interfaces of the module (USB, RS485 or CAN).

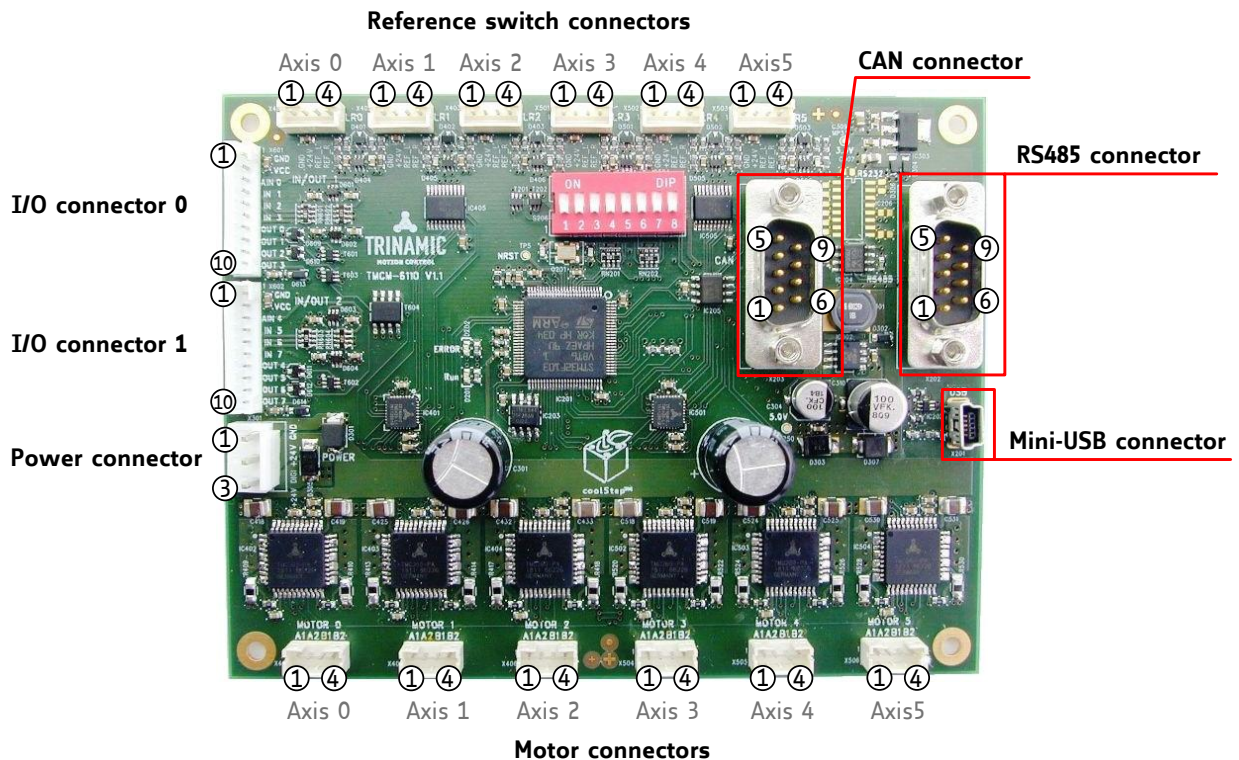


Figure 3.1 Connectors of TMCM-6110

3.1.1.1 Power Supply

Connect the power supply with the power supply connector (see Figure 3.1).

Label	Connector type	Mating connector type
Power Connector	JST B3P-VH (JST VH series, 3pins, 3.96mm pitch)	Connector housing: JST VHR-3N Contacts: JST SVH-21T-P1.1 Wire: 0.83mm ² , AWG 18

Pin	Label	Direction	Description
1	GND	Power (GND)	Common system supply and signal ground
2	V _{DRIVER}	Power (Supply input)	Stepper driver supply voltage. Without this voltage stepper driver ICs and therefore any motor connected will not be energized
3	V _{DIGITAL}	Power (Supply input)	Supply voltage for everything else apart from the stepper motor driver ICs. On-board voltage regulator will generate the necessary voltages for the digital circuits from this supply. This pin can be left unconnected. In this case a diode between V _{DRIVER} and V _{DIGITAL} will ensure supply of the digital parts. <i>Please note:</i> <ul style="list-style-type: none"> - The diode has a current rating of 3A. As V_{DIGITAL} is available at the I/O connectors and the reference switch connectors also, always connect this pin to positive supply voltage in case substantial amount of current is withdrawn from these pins for external circuits. - It is expected that V_{DIGITAL} and V_{DRIVER} are connected to the same power supply output when both pins are used. Otherwise please ensure that V_{DIGITAL} is always equal or higher than V_{DRIVER} when connected (due to the diode).

Do not exceed the maximum power supply of +28V DC!

The device is protected against wrong polarity by a diode that shorts the power supply when the polarity is wrong.

3.1.1.2 Communication

Choose your communication interface out of three serial interfaces: RS485, USB and CAN. If you need more information about the interfaces (e.g. pin assignments), refer to the Hardware Manual, please.

3.1.1.2.1 RS485

Connect the RS485 interface with the appropriate connector (see Figure 3.1).

Label	Connector type	Mating connector type
RS485 Connector	Tyco electronics 3-1634218-2 D-SUB socket with 4-40 female screw locks	Any standard D-SUB female 9-pin

Pin	Label	Direction	Description
1			
2	RS485-	Bi-directional	Differential RS485 bus signal (inverting)
3	GND	Power (GND)	Signal and system ground
4			
5			
6			
7	RS485+	Bi-directional	Differential RS485 bus signal (non-inverting)
8			
9			

RS485 as field bus interface normally requires an adapter. TRINAMIC offers the USB-2-485 converter between USB and RS485.

3.1.1.2.2 USB

Connect the USB interface with the appropriate connector (see Figure 3.1). To make use of the USB interface, a *device driver* has to be installed first. When the TMCM-6110 module is connected to the USB interface of a PC for the first time, you will be prompted for a driver by the operating system. The *tmcm-6110.inf* file is available on www.trinamic.com. After downloading and installing the driver the module is ready for use.

Label	Connector type	Mating connector type
Mini-USB connector	Molex 500075-1517 Mini USB Type B vertical receptacle	Any standard mini-USB plug

Pin	Label	Direction	Description
1	VBUS	Power (+5V input)	+5V supply from Host
2	D-	Bi-directional	USB Data -
3	D+	Bi-directional	USB Data +
4	ID		Connected to signal and system ground
5	GND	Power (GND)	Signal and System ground

CAN interface will be deactivated as soon as USB is connected (V_{BUS} voltage available)

On-board digital core logic (mainly processor and EEPROM) will be powered via USB in case no other supply is connected. This can be used to set parameters / download TMCL programs or perform firmware updates with the module connected via USB only or inside the machine while the machine is powered off.

3.1.1.2.3 CAN

Connect the CAN interface with the appropriate connector (see Figure 3.1). The dip switch can be used for setting the address of the module. If all switches are off the address set by global parameter 71 (CAN ID) is valid.

Label	Connector type	Mating connector type
CAN connector	Male D-SUB 9-pin	Any standard D-SUB female 9-pin

Pin	Label	Direction	Description
1			
2	CAN_L	Bi-directional	Differential CAN bus signal (inverting)
3	GND	Power (GND)	Signal and system ground
4			
5			
6			
7	CAN_H	Bi-directional	Differential CAN bus signal (non-inverting)
8			
9			

TRINAMIC offers the CANnes card, which is a CAN-bus PCI-card and provides a simple and easy use of the CAN interface.

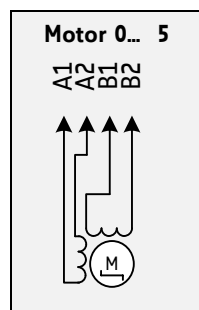
CAN interface will be de-activated in case USB is connected due to internal sharing of hardware resources.

3.1.1.3 Motor

The TMC6110 controls up to six 2-phase stepper motors. Connect one coil of each motor to the terminals marked *A0* and *A1* and the other coil to the connectors marked *B0* and *B1*.

Label	Connector type	Mating connector type
Motor Connectors	CI0104P1VK0-LF CVIlux CI01 series, 4 pins, 2mm pitch	Connector housing CVIlux: CI01045000-A Contacts CVIlux: CI01T011PE0-A <i>or</i> Connector housing JST: PHR-4 Contacts JST: SPH-002T-P0.5S Wire: 0.22mm ²

Pin	Label	Direction	Description
1	A1	Output	Pin 1 of motor coil A
2	A2	Output	Pin 2 of motor coil A
3	B1	Output	Pin 1 of motor coil B
4	B2	Output	Pin 2 of motor coil B



Before connecting a motor please make sure which cable belongs to which coil. Wrong connections may lead to damage of the driver chips or the motor!

Figure 3.2 Motor connection

3.1.1.4 Reference / Home Switches

Connect the switches with the appropriate connectors (see Figure 3.1), if you need them.

Label	Connector type	Mating connector type
Reference Switch Connectors	CI0104P1VK0-LF CVIlux CI01 series, 4 pins, 2mm pitch	Connector housing CVIlux: CI01045000-A Contacts CVIlux: CI01T011PE0-A <i>or</i> Connector housing JST: PHR-4 Contacts JST: SPH-002T-P0.5S Wire: 0.22mm ²

Pin	Label	Direction	Description
1	GND	Power (GND)	Signal and system ground
2	V _{DIGITAL}	Power (Supply input)	Connected to V _{DIGITAL}
3	REF_L	Input	Input for reference / limit switch left
4	REF_R	Input	Input for reference / limit switch right

3.1.1.5 I/Os

Connect inputs and outputs with the appropriate connectors (see Figure 3.1), if you want to use them.

Label	Connector type	Mating connector type
I/O Connector 0 + 1	CI01010P1VK0-LF CVIlux CI01 series, 10 pins, 2mm pitch	Connector housing CVIlux: CI010105000-A Contacts CVIlux: CI01T011PE0-A <i>or</i> Connector housing JST: PHR-10 Contacts JST: SPH-002T-P0.5S Wire: 0.22mm ²

I/O CONNECTOR 0

Pin	Label	Direction	Description
1	GND	Power (GND)	GND
2	V _{DIGITAL}	Power (Supply output)	Connected to V _{DIGITAL} of Power connector
3	AIN_0	Input	Dedicated analog input, input voltage range: 0... +10V, resolution: 12bit (0... 4095)
4	IN_1	Input	Digital input (+24V compatible)
5	IN_2	Input	Digital input (+24V compatible)
6	IN_3	Input	Digital input (+24V compatible)
7	OUT_0	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
8	OUT_1	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
9	OUT_2	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
10	OUT_3	Output	Open-drain output (max. 1A) Integrated freewheeling diode

I/O CONNECTOR 1

Pin	Label	Direction	Description
1	GND	Power (GND)	GND
2	V _{DIGITAL}	Power (Supply output)	Connected to V _{DIGITAL} of Power connector
3	AIN_4	Input	Dedicated analog input, input voltage range: 0... +10V, resolution: 12bit (0... 4095)
4	IN_5	Input	Digital input (+24V compatible)
5	IN_6	Input	Digital input (+24V compatible)
6	IN_7	Input	Digital input (+24V compatible)
7	OUT_4	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
8	OUT_5	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
9	OUT_6	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
10	OUT_7	Output	Open-drain output (max. 1A) Integrated freewheeling diode

Please refer to the Hardware Manual for more information about the connectors.

3.1.2 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**.

For RS485 choose **COM port** and **type** with the parameters shown in Figure 3.3 (baud rate 9600). Click **OK**.

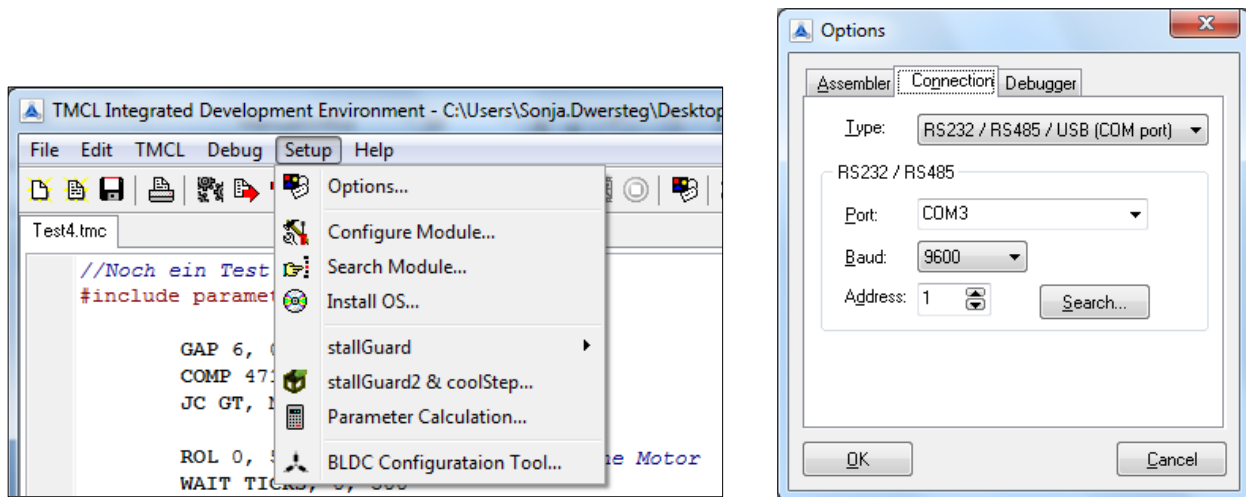
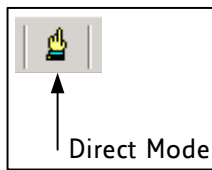


Figure 3.3 Setup dialogue and connection tab of the TMCL-IDE

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

3.1.3 Using TMCL Direct Mode

Start TMCL *Direct Mode*.



If the communication is established the TMCM-6110 is automatically detected.

If the module is not detected, please check cables, interface, power supply, COM port, and baud rate.

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 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.

3.1.4 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 4.6.5. You can use the TCMC-IDE direct mode for easily configuring your module.

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 (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 429 datasheet for calculation of physical units.	0... 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\text{microsteps}}{\text{sec}} \right]$																																								
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>	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 $I_{\text{peak}} = < \text{value} > \times \frac{1.6A}{255}$ $I_{\text{RMS}} = < \text{value} > \times \frac{1.1A}{255}$
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 $I_{\text{peak}} = < \text{value} > \times \frac{1.6A}{255}$ $I_{\text{RMS}} = < \text{value} > \times \frac{1.1A}{255}$																																								
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: $\frac{16\text{MHz}^2}{536870912 \cdot 2^{\text{puls_divisor} + \text{ramp_divisor}} \frac{\text{microsteps}}{\text{sec}^2}}$

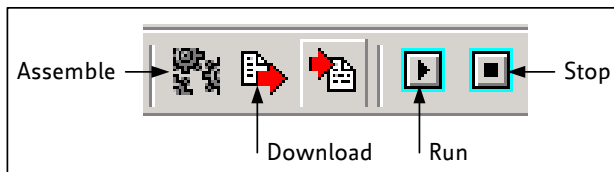
3.1.5 TMCL Program Example

Type the following test program in the editor.

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

    ROL 0, 500           //Rotate motor 0 with speed 500
    WAIT TICKS, 0, 500
    MST 0
    ROR 1, 250           //Rotate motor 0 with 250
    WAIT TICKS, 0, 500
    MST 1

    SAP 4, 2, 500       //Set max. Velocity
    SAP 5, 2, 50        //Set max. Acceleration
Loop: MVP ABS, 2, 10000 //Move to Position 10000
    WAIT POS, 2, 0      //Wait until position reached
    MVP ABS, 2, -10000  //Move to Position -10000
    WAIT POS, 2, 0      //Wait until position reached
    JA Loop             //Infinite Loop
```



1. Click on Icon **Assemble** to convert the TMCL into machine code.
2. Then download the program to the TMCM-6110 module via the icon **Download**.
3. Press icon **Run**. The desired program will be executed.
4. Click **Stop** button to stop the program.

4 TMCL and TMCL-IDE

The TMCM-6110 supports TMCL direct mode (binary commands or ASCII interface) and standalone TMCL program execution. You can store up to 2048 TMCL instructions on it.

In direct mode and most cases the TMCL communication over RS485, USB or CAN follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCM-6110. 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/USB/CAN to the bus master. Only then should the master transfer the next command. Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus. The Trinamic Motion Control Language [TMCL] provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM module to form programs that run standalone on the module. For this purpose there are not only motion control commands but also commands to control the program structure (like conditional jumps, compare and calculating).

Every command has a binary representation and a mnemonic. The binary format is used to send commands from the host to a module in direct mode, whereas the mnemonic format is used for easy usage of the commands when developing standalone TMCL applications using the TMCL-IDE (IDE means *Integrated Development Environment*).

There is also a set of configuration variables for the axis and for global parameters which allow individual configuration of nearly every function of a module. This manual gives a detailed description of all TMCL commands and their usage.

4.1 Binary Command Format

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 RS485/USB is as follows:

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

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

CHECKSUM CALCULATION

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

in C:

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

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

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

4.2 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 "no error")
1	Command number
4	Value (MSB first!)
1	Checksum

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

4.2.1 Status Codes

The reply contains a status code.

The status code can have one of the following values:

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

4.3 Standalone Applications

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

4.4 TMCL Command Overview

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

4.4.1 TMCL Commands

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

4.4.2 Commands Listed According to Subject Area

4.4.2.1 Motion Commands

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

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

4.4.2.2 Parameter Commands

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

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

4.4.2.3 Control Commands

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

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

4.4.2.4 I/O Port Commands

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

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

4.4.2.5 Calculation Commands

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

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

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

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

4.4.2.6 Interrupt Commands

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

4.4.2.6.1 Interrupt Types:

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

4.4.2.6.2 Interrupt Processing:

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

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

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

4.4.2.6.3 Interrupt Vectors:

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached axis 0
4	Target position reached axis 1
5	Target position reached axis 2
6	Target position reached axis 3
7	Target position reached axis 4
8	Target position reached axis 5
15	stallGuard axis 0
16	stallGuard axis 1
17	stallGuard axis 2
18	stallGuard axis 3
19	stallGuard axis 4
20	stallGuard axis 5
27	Left stop switch axis 0
28	Right stop switch axis 0
29	Left stop switch axis 1
30	Right stop switch axis 1
31	Left stop switch axis 2
32	Right stop switch axis 2
33	Left stop switch axis 3
34	Right stop switch axis 3
35	Left stop switch axis 4
36	Right stop switch axis 4
37	Left stop switch axis 5
38	Right stop switch axis 5
39	Input change 0
40	Input change 1
41	Input change 2
42	Input change 3
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

4.4.2.6.4 Further Configuration of Interrupts

Some interrupts need further configuration (e.g. the timer interval of a timer interrupt). This can be done using SGP commands with parameter bank 3 (SGP <type>, 3, <value>). Please refer to the SGP command (paragraph 4.6.9) for further information about that.

4.4.2.6.5 Using Interrupts in TMCL

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

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

THE FOLLOWING EXAMPLE SHOWS THE USE OF A TIMER INTERRUPT:

```

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

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

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

```

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

```

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

```

Please also take a look at the other example programs (see www.trinamic.com).

4.4.2.7 ASCII Commands

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

4.5 The ASCII Interface

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

PROCEED AS FOLLOWS

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

4.5.1 Command Line Format

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
```

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

4.5.2 Format of a Reply

After executing the command the module sends back a reply in ASCII format. The reply consists of:

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

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

4.5.3 Commands Used in ASCII Mode

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

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.

4.5.4 Configuring the ASCII Interface

The module can be configured so that it starts up either in binary mode or in ASCII mode. **Global parameter 67 is used for this purpose** (please see also chapter 6.1).

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. Characters can also be erased using the backspace character (press the backspace key in a terminal program).

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

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

4.6 Commands

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

4.6.1 ROR (rotate right)

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 #2 (*target velocity*).

The module is based on the TMC429 stepper motor controller and the TMC260 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR <motor>, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
1	don't care	<motor> 0... 5	<velocity> 0... 2047

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Rotate right motor 2, velocity = 350

Mnemonic: ROR 2, 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	\$02	\$00	\$00	\$01	\$5e

4.6.2 ROL (rotate left)

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 #2 (*target velocity*).

The module is based on the TMC429 stepper motor controller and the TMC260 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL <motor>, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
2	don't care	<motor> 0... 5	<velocity> 0... 2047

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Rotate left motor 0, velocity = 1200

Mnemonic: ROL 0, 1200

Binary:

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

4.6.3 MST (motor stop)

The motor will be instructed to stop.

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

Related commands: ROL, ROR, SAP, GAP

Mnemonic: MST <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
3	don't care	<motor> 0... 5	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Stop motor 0

Mnemonic: MST 0

Binary:

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

4.6.4 MVP(move to position)

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

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

THREE OPERATION TYPES ARE AVAILABLE:

- Moving to an absolute position in the range from -2.147.483.648... +2.147.483.647 ($-2^{31} \dots 2^{31}-1$).
- 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>, <motor>, <position|offset|coordinate number>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
4	0 ABS – absolute	<motor> 0... 5	<position>
	1 REL – relative		<offset>
	2 COORD – coordinate		<coordinate number> 0... 20

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

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

Binary:

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

Example:

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

Binary:

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

Versatile options for the MVP COORD command:

- Moving only one motor: Set the <motor> parameter to the motor number (0...2).
- Moving multiple motors without interpolation: Set bit 7 of the <motor> parameter. Now the bits 0...2 of the <motor> parameter define which motors are to be started. Each of these bits stands for one motor.
- Moving multiple motors using interpolation: Set bit 6 of the <motor> parameter. Now the bits 0...2 of the <motor> parameter define which motors are to be moved using the interpolation. Each of these bits stands for one motor.

Examples MVP COORD:

- Move motors 0, 1 and 2 to coordinate 2 using interpolation
Mnemonic: MVP COORD, \$47, 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	\$04	\$02	\$47	\$00	\$00	\$00	\$02	\$50

- Move motors 0, 1 and 2 to coordinate 5 without using interpolation
Mnemonic: MVP COORD, \$87, 5

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	\$87	\$00	\$00	\$00	\$05	\$93

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

Binary:

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

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

4.6.5 SAP (set axis parameter)

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

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Set the absolute maximum current of motor 2 to 200mA

Because of the current unit $I_{RMS} = \langle value \rangle \times \frac{11A}{255}$ the 200mA setting has the <value> 47 (value range for current setting: 0... 255). The value for current setting has to be calculated before using this special SAP command.

Mnemonic: SAP 6, 2, 47

Binary:

Byte Index	0	1	2	3	4	5	6	7
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$05	\$06	\$02	\$00	\$00	\$00	\$2f

4.6.6 GAP (get axis parameter)

Most parameters of the TMCM-6110 can be adjusted individually for the axis. With this parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditioned jumps). In direct mode the value read is only output in the *value* field of the reply (without affecting the accumulator).

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

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

Related commands: SAP, STAP, AAP, RSAP

Mnemonic: GAP <parameter number>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
6	<parameter number>	<motor> 0... 5	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Get the maximum current of motor 1

Mnemonic: GAP 6, 1

Binary:

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

Reply:

Byte Index	0	1	2	3	4	5	6	7
Function	Host-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$02	\$01	\$64	\$06	\$00	\$00	\$02	\$80

⇒ **Status = no error, value = 128**

4.6.7 STAP (store axis parameter)

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

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

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

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

Related commands: SAP, RSAP, GAP, AAP

Mnemonic: STAP <parameter number>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
7	<parameter number>	<motor> 0... 5	don't care*

* the value operand of this function has no effect. Instead, the currently used value (e.g. selected by SAP) is saved

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Store the maximum speed of motor 0

Mnemonic: STAP 4, 0

Binary:

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

4.6.8 RSAP (restore axis parameter)

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

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

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

Relate commands: SAP, STAP, GAP, and AAP

Mnemonic: RSAP <parameter number>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
8	<parameter number>	<motor> 0... 5	don't care

Reply structure in direct mode:

STATUS	VALUE
100 - OK	don't care

Example:

Restore the maximum current of motor 3
Mnemonic: RSAP 6, 0

Binary:

Byte Index	0	1	2	3	4	5	6	7
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$08	\$06	\$03	\$00	\$00	\$00	\$00

4.6.9 SGP (set global parameter)

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

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

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

Internal function: the parameter format is converted 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

Example:

Set the serial address of the target device to 3

Mnemonic: SGP 66, 0, 3

Binary:

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

4.6.10 GGP (get global parameter)

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

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

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

Example:

Get the serial address of the target device

Mnemonic: GGP 66, 0

Binary:

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

Reply:

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

⇒ Status = no error, value = 1

4.6.11 STGP (store global parameter)

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

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

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

Related commands: SGP, GGP, RSGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Store the user variable #42

Mnemonic: STGP 42, 2

Binary:

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

4.6.12 RSGP (restore global parameter)

With this command the contents of a TMCL user variable can be restored from the EEPROM. 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.

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

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

Relate commands: SGP, STGP, GGP, and AGP

Mnemonic: RSGP <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

Example:

Restore the user variable #42

Mnemonic: RSGP 42, 2

Binary:

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

4.6.13 RFS (reference search)

The TMCM-6110 has a built-in reference search algorithm which can be used. The reference search algorithm provides switching point calibration and supports up-to three switches. The status of the reference search can also be queried to see if it has already been finished. (In a TMCL program it is better to use the WAIT command to wait for the end of a reference search.) Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (chapter 5). The reference search can be started, stopped, and the actual status of the reference search can be checked.

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

Related commands: WAIT

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
13	0 START – start ref. search 1 STOP – abort ref. search 2 STATUS – get status	<motor> 0... 5	see below

Reply in direct mode:

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

STATUS	VALUE
100 – OK	don't care

When using type 2 (STATUS):

STATUS	VALUE	
100 – OK	0	No reference search active
	other values	Reference search active

Example:

Start reference search of motor 0

Mnemonic: RFS START, 0

Binary:

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

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

4.6.14 SIO (set input / output)

This command can be used as follows:

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

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

Related commands: GIO, WAIT

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
14	<port number>	<bank number> 2	<value> 0/1

Reply structure:

STATUS	VALUE
100 – OK	don't care

Example:

Set OUT_7 to high (bank 2, output 7)

Mnemonic: SIO 7, 2, 1

Binary:

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

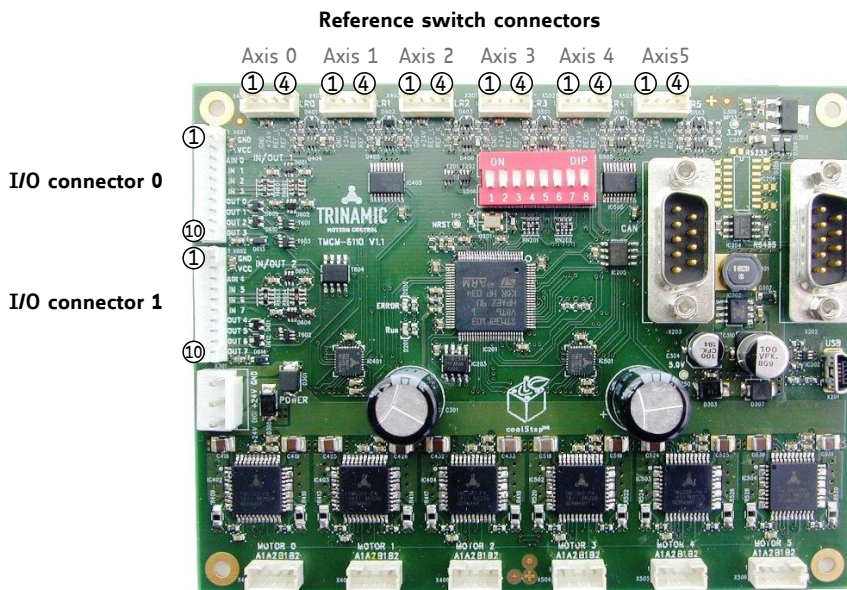


Figure 4.1 Connectors of TMCM-6110

I/O CONNECTOR 0 AND 1

Pin	Connector 0	Connector 1	Direction	Description
1	GND	GND	Power (GND)	GND
2	V _{DIGITAL}	V _{DIGITAL}	Power (supply output)	Connected to V _{DIGITAL} of power connector
3	AIN_0	AIN_4	Input	Dedicated analog input, input voltage range: 0... +10V, resolution: 12bit (0... 4095)
4	IN_1	IN_5	Input	Digital input (+24V compatible)
5	IN_2	IN_6	Input	Digital input (+24V compatible)
6	IN_3	IN_7	Input	Digital input (+24V compatible)
7	OUT_0	OUT_4	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
8	OUT_1	OUT_5	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
9	OUT_2	OUT_6	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
10	OUT_3	OUT_7	Output	Open-drain output (max. 1A) Integrated freewheeling diode

SETTING THE STATUS OF GENERAL DIGITAL OUTPUTS EITHER TO LOW (0) OR TO HIGH (1)

Pin	I/O Connector	I/O port	Command	Range
7	0	OUT_0	SIO 0, 2, <n>	1/0
8	0	OUT_1	SIO 1, 2, <n>	1/0
9	0	OUT_2	SIO 2, 2, <n>	1/0
10	0	OUT_3	SIO 3, 2, <n>	1/0
7	1	OUT_4	SIO 4, 2, <n>	1/0
8	1	OUT_5	SIO 5, 2, <n>	1/0
9	1	OUT_6	SIO 6, 2, <n>	1/0
10	1	OUT_7	SIO 7, 2, <n>	1/0

ADDRESSING ALL OUTPUT LINES WITH ONE SIO COMMAND:

Proceed as follows:

- 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 eight output pins.

Example:

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

THE FOLLOWING PROGRAM WILL SHOW THE STATES OF THE INPUT LINES ON THE OUTPUT LINES:

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

SWITCHING THE PULL-UP RESISTORS OF THE REFERENCE SWITCH INPUTS

SIO can be used to switch the pull-up resistors for all reference switch inputs on and off. Bank 0 is used for this purpose.

PIN ASSIGNMENT OF REFERENCE SWITCH CONNECTOR 0-5

Pin	Label	Description
3	REF_L	Input for reference / limit switch left
4	REF_R	Input for reference / limit switch right

Using the bitmask of this SIO command, the pull-up resistors can be switched on/off in two functional groups: bit 0 is used for motor 0, 1, and 2. Bit 1 is used for motor 3, 4, and 5. Setting the bit will switch on the pull-ups, clearing it will switch them off.

COMMAND FOR SWITCHING THE PULL-UP RESISTORS

Motor	Bit	Command	Range
0, 1, 2	0	SIO 0, 0,<bitmask>	0... 3
3, 4, 5	1		

4.6.15 GIO (get input /output)

With this command the status of the two available general purpose inputs of the module can be read out. The function reads a digital or analogue input port. Digital lines will read 0 and 1, while the ADC channels deliver their 12 bit 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 conditioned jumps.

In *direct mode* the value is only output in the *value* field of the reply, without affecting the accumulator. The actual status of a digital output line can also be read.

Internal function: the specified line is read.

Related commands: SIO, WAIT

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	<status of the port>

Example:

Get the analogue value of ADC channel 0

Mnemonic: GIO 0, 1

Binary:

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

Reply:

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

Status = no error, value = 46

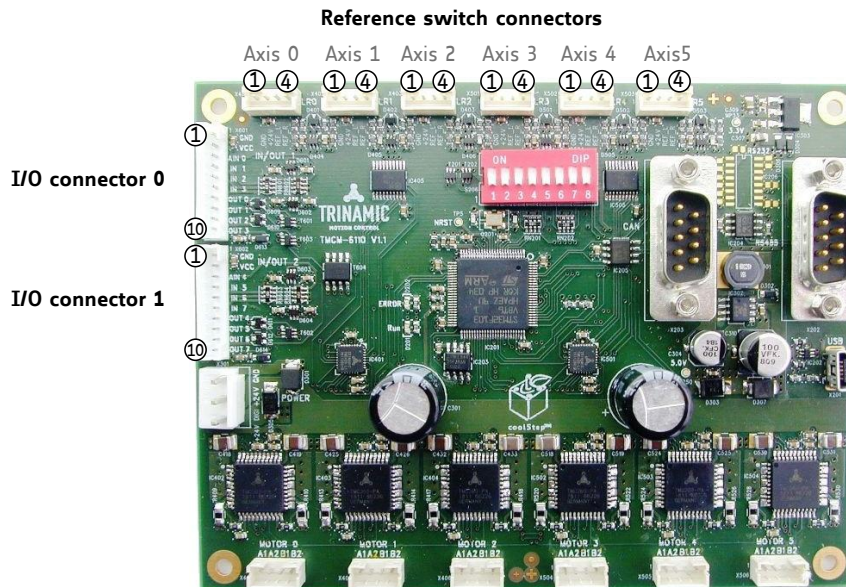


Figure 4.2 Connectors of TMCM-6110

I/O CONNECTOR 0 AND 1

Pin	Connector 0	Connector 1	Direction	Description
1	GND	GND	Power (GND)	GND
2	V _{DIGITAL}	V _{DIGITAL}	Power (supply output)	Connected to V _{DIGITAL} of Power connector
3	AIN_0	AIN_4	Input	Dedicated analog input, input voltage range: 0... +10V, resolution: 12bit (0... 4095)
4	IN_1	IN_5	Input	Digital input (+24V compatible)
5	IN_2	IN_6	Input	Digital input (+24V compatible)
6	IN_3	IN_7	Input	Digital input (+24V compatible)
7	OUT_0	OUT_4	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
8	OUT_1	OUT_5	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
9	OUT_2	OUT_6	Output	Open-drain output (max. 100mA) Integrated freewheeling diode
10	OUT_3	OUT_7	Output	Open-drain output (max. 1A) Integrated freewheeling diode

4.6.15.1 I/O bank 0 – Digital Inputs

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

Pin	I/O Connector	I/O port	Command	Range
4	0	IN_1	GIO 1, 0	0/1
5	0	IN_2	GIO 2, 0	0/1
6	0	IN_3	GIO 3, 0	0/1
4	1	IN_5	GIO 5, 0	0/1
5	1	IN_6	GIO 6, 0	0/1
6	1	IN_7	GIO 7, 0	0/1

READING ALL DIGITAL INPUTS WITH ONE GIO COMMAND:

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

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

```

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

```

4.6.15.2 I/O bank 1 – Analogue Inputs

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

Pin	I/O Connector	I/O port	Command	Range
3	0	AIN_0	GIO 0, 1	0... 4095
3	1	AIN_4	GIO 4, 1	0... 4095

4.6.15.3 I/O bank 2 – 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 Connector	I/O port	Command	Range
7	0	OUT_0	GIO 0, 2, <n>	1/0
8	0	OUT_1	GIO 1, 2, <n>	1/0
9	0	OUT_2	GIO 2, 2, <n>	1/0
10	0	OUT_3	GIO 3, 2, <n>	1/0
7	1	OUT_4	GIO 4, 2, <n>	1/0
8	1	OUT_5	GIO 5, 2, <n>	1/0
9	1	OUT_6	GIO 6, 2, <n>	1/0
10	1	OUT_7	GIO 7, 2, <n>	1/0

4.6.16 CALC (calculate)

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

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

Mnemonic: CALC <operation>, <value>

Binary representation:

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

Example:

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

Binary:

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

Reply:

Byte Index	0	1	2	3	4	5	6	7
Function	Host-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$02	\$01	\$64	\$13	\$ff	\$ff	\$ec	\$78

Status = no error, value = -5000

4.6.17 COMP (compare)

The specified number is compared to the value in the accumulator register. The result of the comparison can for example be used by the conditional jump (JC) instruction.

This command is intended for use in standalone operation only.

Internal function: the specified value is compared to the internal "accumulator", which holds the value of a preceding "get" or calculate instruction (see GAP/GGP/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
Function	Target-address	Instruction Number	Type	Motor/Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	\$01	\$14	\$00	\$00	\$00	\$00	\$03	\$e8

4.6.18 JC (jump conditional)

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

This function is for standalone operation only.

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

Related commands: JA, COMP, WAIT, CLE

Mnemonic: JC <condition>, <label>

Binary representation:

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

Example:

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

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

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

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

4.6.19 JA (jump always)

Jump to a fixed address in the TMCL program memory.

This command is intended for standalone operation only.

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

Related commands: JC, WAIT, CSUB

Mnemonic: JA <Label>

Binary representation:

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

Example: An infinite loop in TMCL

```

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

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

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

4.6.20 CSUB (call subroutine)

This function calls a subroutine in the TMCL program memory.

This command is intended for standalone operation only.

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

Related commands: RSUB, JA

Mnemonic: CSUB <Label>

Binary representation:

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

Example: Call a subroutine

```

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

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

```

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

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

4.6.21 RSUB (return from subroutine)

Return from a subroutine to the command after the CSUB command.

This command is intended for use in standalone mode only.

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

Related command: CSUB

Mnemonic: RSUB

Binary representation:

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

Example: please see the CSUB example (section 4.6.20).

Binary format of RSUB:

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

4.6.22 WAIT (wait for an event to occur)

This instruction interrupts the execution of the TMCL program until the specified condition is met.

This command is intended for standalone operation only.

THERE ARE FIVE DIFFERENT WAIT CONDITIONS THAT CAN BE USED:

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

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

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

Related commands: JC, CLE

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

Binary representation:

INSTRUCTION NO.	TYPE <condition>	MOT/BANK	VALUE
27	0 TICKS - timer ticks* ¹	don't care	<no. of ticks*>
	1 POS - target position reached	<motor> 0... 5	<no. of ticks* for timeout>, 0 for no timeout
	2 REFSW - reference switch	<motor> 0... 5	<no. of ticks* for timeout>, 0 for no timeout
	3 LIMSW - limit switch	<motor> 0... 5	<no. of ticks* for timeout>, 0 for no timeout
	4 RFS - reference search completed	<motor> 0... 5	<no. of ticks* for timeout>, 0 for no timeout

*¹ one tick is 10 milliseconds

Example:

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

Binary:

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

4.6.23 STOP (stop TMCL program execution)

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

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

Internal function: TMCL instruction fetching is stopped.

Related commands: none

Mnemonic: STOP

Binary representation:

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

Example:

Mnemonic: STOP

Binary:

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

4.6.24 SCO (set coordinate)

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

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>, <motor>, <position>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
30	<coordinate number> 0... 20	<motor> 0... 5	<position> -2 ³¹ ... 2 ³¹ -1

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Set coordinate #1 of motor to 1000

Mnemonic: SCO 1, 0, 1000

Binary:

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

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

These functions can be accessed using the following special forms of the SCO command:

SCO 0, 255, 0

copies all coordinates (except coordinate number 0) from RAM to the EEPROM.

SCO <coordinate number>, 255, 0

copies the coordinate selected by <coordinate number> to the EEPROM. The coordinate number must be a value between 1 and 20.

4.6.25 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>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
31	<coordinate number> 0... 20	<motor> 0... 5	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Get motor value of coordinate 1

Mnemonic: GCO 1, 0

Binary:

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

Reply:

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

⇒ **Value: 0**

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

These functions can be accessed using the following special forms of the GCO command:

GCO 0, 255, 0

copies all coordinates (except coordinate number 0) from the EEPROM to the RAM.

GCO <coordinate number>, 255, 0

copies the coordinate selected by <coordinate number> from the EEPROM to the RAM. The coordinate number must be a value between 1 and 20.

4.6.26 CCO (capture coordinate)

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

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

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

Related commands: SCO, GCO, MVP

Mnemonic: CCO <coordinate number>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
32	<coordinate number> 0... 20	<motor> 0... 5	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

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

Binary:

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

4.6.27 ACO (accu to coordinate)

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>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
39	<coordinate number> 0... 20	<motor> 0... 5	don't care

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

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

Mnemonic: ACO 1, 0

Binary:

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

4.6.28 CALCX (calculate using the X register)

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

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

Mnemonic: CALCX <operation>

Binary representation:

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

Example:

Multiply accu by X-register

Mnemonic: CALCX MUL

Binary:

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

4.6.29 AAP (accumulator to axis parameter)

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

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

Mnemonic: AAP <parameter number>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
34	<parameter number>	<motor> 0... 5	<don't care>

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

4.6.30 AGP (accumulator to global parameter)

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

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

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

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

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Copy accumulator to TMCL user variable #3

Mnemonic: AGP 3, 2

Binary:

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

4.6.31 CLE (clear error flags)

This command clears the internal error flags.

This command 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	don't care	don't care

Example:

Reset the timeout flag
Mnemonic: CLE ETO

Binary:

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

4.6.32 VECT (set interrupt vector)

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

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

Related commands: EI, DI, RETI

Mnemonic: VECT <interrupt number>, <label>

Binary representation:

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

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached axis 0
4	Target position reached axis 1
5	Target position reached axis 2
6	Target position reached axis 3
7	Target position reached axis 4
8	Target position reached axis 5
15	stallGuard axis 0
16	stallGuard axis 1
17	stallGuard axis 2
18	stallGuard axis 3
19	stallGuard axis 4
20	stallGuard axis 5
27	Left stop switch axis 0
28	Right stop switch axis 0
29	Left stop switch axis 1
30	Right stop switch axis 1
31	Left stop switch axis 2
32	Right stop switch axis 2
33	Left stop switch axis 3
34	Right stop switch axis 3
35	Left stop switch axis 4
36	Right stop switch axis 4
37	Left stop switch axis 5
38	Right stop switch axis 5
39	Input change 0
40	Input change 1
41	Input change 2
42	Input change 3
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

Example:

Define interrupt vector at target position 500
VECT 3, 500

Binary format of VECT:

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

4.6.33 EI (enable interrupt)

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

Related command: DI, VECT, RETI

Mnemonic: EI <interrupt number>

Binary representation:

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

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached axis 0
4	Target position reached axis 1
5	Target position reached axis 2
6	Target position reached axis 3
7	Target position reached axis 4
8	Target position reached axis 5
15	stallGuard axis 0
16	stallGuard axis 1
17	stallGuard axis 2
18	stallGuard axis 3
19	stallGuard axis 4
20	stallGuard axis 5
27	Left stop switch axis 0
28	Right stop switch axis 0
29	Left stop switch axis 1
30	Right stop switch axis 1
31	Left stop switch axis 2
32	Right stop switch axis 2
33	Left stop switch axis 3
34	Right stop switch axis 3
35	Left stop switch axis 4
36	Right stop switch axis 4
37	Left stop switch axis 5
38	Right stop switch axis 5
39	Input change 0
40	Input change 1
41	Input change 2
42	Input change 3
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

Examples:

Enable interrupts globally
EI, 255

Binary format of EI:

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

Enable interrupt when target position reached
EI, 3

Binary format of EI:

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

4.6.34 DI (disable interrupt)

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

Related command: EI, VECT, RETI

Mnemonic: DI <interrupt number>

Binary representation:

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

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached axis 0
4	Target position reached axis 1
5	Target position reached axis 2
6	Target position reached axis 3
7	Target position reached axis 4
8	Target position reached axis 5
15	stallGuard axis 0
16	stallGuard axis 1
17	stallGuard axis 2
18	stallGuard axis 3
19	stallGuard axis 4
20	stallGuard axis 5
27	Left stop switch axis 0
28	Right stop switch axis 0
29	Left stop switch axis 1
30	Right stop switch axis 1
31	Left stop switch axis 2
32	Right stop switch axis 2
33	Left stop switch axis 3
34	Right stop switch axis 3
35	Left stop switch axis 4
36	Right stop switch axis 4
37	Left stop switch axis 5
38	Right stop switch axis 5
39	Input change 0
40	Input change 1
41	Input change 2
42	Input change 3
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

Examples:

Disable interrupts globally
 DI, 255

Binary format of DI:

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

Disable interrupt when target position reached
 DI, 3

Binary format of DI:

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

4.6.35 RETI (return from interrupt)

This command terminates the interrupt handling routine, and the normal program execution continues.

At the end of an interrupt handling routine the RETI command must be executed.

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

Related commands: EI, DI, VECT

Mnemonic: RETI

Binary representation:

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

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

Binary format of RETI:

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

4.6.36 Customer Specific TMCL Command Extension (user function)

The user definable functions UF0... UF7 are predefined functions without topic for user specific purposes. A user function (UF) command uses three parameters. Please contact TRINAMIC for a customer specific programming.

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

Related commands: none

Mnemonic: UF0... UF7 <parameter number>

Binary representation:

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

Reply in direct mode:

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

4.6.37 Request Target Position Reached Event

This command is the only exception to the TMCL protocol, as it sends two replies: One immediately after the command has been executed (like all other commands also), and one additional reply that will be sent when the motor has reached its target position.

This instruction can only be used in direct mode (in standalone mode, it is covered by the WAIT command) and hence does not have a mnemonic.

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

Mnemonic: ---

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
138	don't care	don't care	<motor bit mask>

The *value* field contains a bit mask. Every bit stands for one motor:

- bit 0 = motor 0
- bit 1 = motor 1
- bit 2 = motor 2
- bit 3 = motor 3
- bit 4 = motor 4
- bit 5 = motor 5

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

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

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

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

4.6.38 BIN (return to binary mode)

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

Related Commands: none

Mnemonic: BIN

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

4.6.39 TMCL Control Functions

There are several TMCL control functions, but for the user are only 136 and 137 interesting. Other control functions can be used with axis parameters.

Instruction number	Type	Command	Description
136	0 – string 1 – binary	Firmware version	Get the module type and firmware revision as a string or in binary format. (<i>Motor/Bank</i> and <i>Value</i> are ignored.)
137	don't care	Reset to factory defaults	Reset all settings stored in the EEPROM to their factory defaults This command does not send back a reply. <i>Value must be 1234</i>

FURTHER INFORMATION ABOUT COMMAND 136

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

Byte index	Contents
1	Host address
2... 9	Version string (8 characters, e.g. 6110V100)

There is no checksum in this reply format!

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

The version number is output in the *value* field of the reply in the following way:


Byte index in value field	Contents
1	17
2	DE
3	Type number, low byte
4	Type number, high byte


5 Axis Parameters

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

MEANING OF THE LETTERS IN COLUMN ACCESS

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

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

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

Number	Axis Parameter	Description	Range [Unit]	Acc.
0	Target (next) position	The desired position in position mode (see ramp mode, no. 138).	$-2^{31} \dots 2^{31}-1$ [μsteps]	RW
1	Actual position	The current position of the motor. Should only be overwritten for reference point setting.	$-2^{31} \dots 2^{31}-1$ [μsteps]	RW
2	Target (next) speed	The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest.	± 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RW
3	Actual speed	The current rotation speed.	± 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RW
4	Maximum positioning speed	Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 429 datasheet for calculation of physical units.	0... 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RWE
5	Maximum acceleration	The limit for acceleration (and deceleration). Changing this parameter requires recalculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC 429 datasheet for calculation of physical units.	0... 2047*	RWE

Number	Axis Parameter	Description	Range [Unit]	Acc.																																								
6	Absolute max. current (CS / Current Scale)	<p>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.</p> <table border="1"> <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		<p>0... 255</p> $I_{peak} = < value > \times \frac{1.6A}{255}$ $I_{RMS} = < value > \times \frac{1.1A}{255}$	RWE
0... 7	79...87	160... 167	240... 247																																									
8... 15	88... 95	168... 175	248... 255																																									
16... 23	96... 103	176... 183																																										
24... 31	104... 111	184... 191																																										
32... 39	112... 119	192... 199																																										
40... 47	120... 127	200... 207																																										
48... 55	128... 135	208... 215																																										
56... 63	136... 143	216... 223																																										
64... 71	144... 151	224... 231																																										
72... 79	152... 159	232... 239																																										
7	Standby current	The current limit two seconds after the motor has stopped.	<p>0... 255</p> $I_{peak} = < value > \times \frac{1.6A}{255}$ $I_{RMS} = < value > \times \frac{1.1A}{255}$	RWE																																								
8	Target pos. reached	Indicates that the actual position equals the target position.	0/1	R																																								
9	Ref. switch status	The logical state of the reference home switch.	0/1	R																																								
10	Right limit switch status	The logical state of the (right) limit switch.	0/1	R																																								
11	Left limit switch status	The logical state of the left limit switch (in three switch mode)	0/1	R																																								
12	Right limit switch disable	If set, deactivates the stop function of the right switch	0/1	RWE																																								
13	Left limit switch disable	Deactivates the stop function of the left switch resp. reference switch if set.	0/1	RWE																																								
130	Minimum speed	Should always be set 1 to ensure exact reaching of the target position. Do not change!	<p>0... 2047</p> $\left[\frac{16MHz}{65536} \cdot 2^{PD} \frac{\mu steps}{sec} \right]$	RWE																																								
135	Actual acceleration	The current acceleration (read only).	0... 2047*	R																																								
138	Ramp mode	<p>Automatically set when using ROR, ROL, MST and MVP.</p> <p>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.</p> <p>2: velocity mode. The motor will run continuously and the speed will be changed with constant (maximum) acceleration, if the parameter target speed is changed.</p> <p>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected.</p>	0/1/2	RWE																																								

Number	Axis Parameter	Description	Range [Unit]	Acc.	
140	Microstep resolution	0	full step	0... 8	RWE
		1	half step		
		2	4 microsteps		
		3	8 microsteps		
		4	16 microsteps		
		5	32 microsteps		
		6	64 microsteps		
		7	128 microsteps		
8	256 microsteps				
141	Reference switch tolerance	For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop.	0... 4095 [μsteps]	RW	
149	Soft stop flag	If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit.	0/1	RWE	
153	Ramp divisor	The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one).	0... 13	RWE	
154	Pulse divisor	The exponent of the scaling factor for the pulse (step) generator - should be de/incremented carefully (in steps of one).	0... 13	RWE	
160	Step interpolation enable	Step interpolation is supported with a 16 microstep setting only. In this setting, each step impulse at the input causes the execution of 16 times 1/256 microsteps. This way, a smooth motor movement like in 256 microstep resolution is achieved. 0 - step interpolation off 1 - step interpolation on	0/1	RW	
161	Double step enable	Every edge of the cycle releases a step/microstep. <i>It does not make sense to activate this parameter for internal use.</i> Double step enable can be used with Step/Dir interface. 0 - double step off 1 - double step on	0/1	RW	
162	Chopper blank time	Selects the comparator <i>blank time</i> . This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. For low current drivers, a setting of 1 or 2 is good.	0... 3	RW	
163	Chopper mode	Selection of the chopper mode: 0 - spread cycle 1 - classic const. off time	0/1	RW	
164	Chopper hysteresis decrement	Hysteresis decrement setting. This setting determines the slope of the hysteresis during on time and during fast decay time. 0 - fast decrement 3 - very slow decrement	0... 3	RW	
165	Chopper hysteresis end	Hysteresis end setting. Sets the hysteresis end value after a number of decrements. Decrement interval time is controlled by axis parameter 164.	-3... 12	RW	
		-3... -1			negative hysteresis end setting
		0			zero hysteresis end setting
1... 12	positive hysteresis end setting				

Number	Axis Parameter	Description	Range [Unit]	Acc.
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 5 μ s to 20 μ s 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 (SEIMIN)	Sets the lower motor current limit for coolStep operation by scaling the CS (Current Scale, see axis parameter 6) value. minimum motor current: 0 – 1/2 of CS 1 – 1/4 of CS	0/1	RW
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 stallGuard2 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
172	smartEnergy hysteresis start	The lower threshold for the stallGuard2 value (see smart Energy current up step).	0... 15	RW

Number	Axis Parameter	Description	Range [Unit]	Acc.						
173	stallGuard2 filter enable	Enables the stallGuard2 filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps. <i>In most cases it is expedient to set the filtered mode before using coolStep.</i> <i>Use the standard mode for step loss detection.</i> 0 – standard mode 1 – filtered mode	0/1	RW						
174	stallGuard2 threshold	This signed value controls stallGuard2 <i>threshold</i> level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2 less sensitive and requires more torque to indicate a stall. <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>0</td> <td>Indifferent value</td> </tr> <tr> <td>1... 63</td> <td>less sensitivity</td> </tr> <tr> <td>-10 - 64</td> <td>higher sensitivity</td> </tr> </table>	0	Indifferent value	1... 63	less sensitivity	-10 - 64	higher sensitivity	-64... 63	RW
0	Indifferent value									
1... 63	less sensitivity									
-10 - 64	higher sensitivity									
175	Slope control high side	Determines the slope of the motor driver outputs. <i>Set to 2 or 3 for this module or rather use the default value.</i> 0: lowest slope 3: fastest slope	0... 3	RW						
176	slope control low side	Determines the slope of the motor driver outputs. <i>Set identical to slope control high side.</i>	0... 3	RW						
177	Short protection disable	0: Short to GND protection is on 1: Short to GND protection is disabled <i>Use default value!</i>	0/1	RW						
178	Short detection timer	0: 3.2µs 1: 1.6µs 2: 1.2µs 3: 0.8µs <i>Use default value!</i>	0... 3	RW						
179	Vsense	sense resistor voltage based current scaling 0: Full scale sense resistor voltage is 1/18 VDD 1: Full scale sense resistor voltage is 1/36 VDD (refers to a current setting of 31 and DAC value 255) <i>Use default value. Do not change!</i>	0/1	RW						
180	smartEnergy actual current	This status value provides the <i>actual motor current</i> setting as controlled by coolStep. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN. <u>Actual motor current scaling factor:</u> 0 ... 31: 1/32, 2/32, ... 32/32	0... 31	RW						
181	Stop on stall	Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2 load value reaches zero.	0... 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RW						
182	smartEnergy threshold speed	Above this speed coolStep becomes enabled.	0... 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}} \right]$	RW						

Number	Axis Parameter	Description	Range [Unit]	Acc.																
183	smartEnergy slow run current	Sets the motor current which is used below the threshold speed.	0... 255 $I_{peak} = < value > \times \frac{1.6A}{255}$ $I_{RMS} = < value > \times \frac{1.1A}{255}$	RW																
184	Random chopper off time	Enable randomizing the slow decay phase duration: 0: Chopper off time is fixed (see axis parameter 167). 1: Random mode. Chopper off time is modulated by dNCLK= -12 ... +3 clocks.	0/1	RWE																
193	Reference search mode	<table border="1"> <tr> <td>1</td> <td>search left stop switch only</td> </tr> <tr> <td>2</td> <td>search right stop switch, then search left stop switch</td> </tr> <tr> <td>3</td> <td>search right stop switch, then search left stop switch from both sides</td> </tr> <tr> <td>4</td> <td>search left stop switch from both sides</td> </tr> <tr> <td>5</td> <td>search home switch in negative direction, reverse the direction when left stop switch reached</td> </tr> <tr> <td>6</td> <td>search home switch in positive direction, reverse the direction when right stop switch reached</td> </tr> <tr> <td>7</td> <td>search home switch in negative direction, ignore end switches</td> </tr> <tr> <td>8</td> <td>search home switch in positive direction, ignore end switches</td> </tr> </table> <p>Additional functions:</p> <ul style="list-style-type: none"> - Add 128 to a mode value for inverting the home switch polarity (can be used with mode 5... 8). - Add 64 to a mode for exchanging left and right stop switches (can be used with mode 1... 4). 	1	search left stop switch only	2	search right stop switch, then search left stop switch	3	search right stop switch, then search left stop switch from both sides	4	search left stop switch from both sides	5	search home switch in negative direction, reverse the direction when left stop switch reached	6	search home switch in positive direction, reverse the direction when right stop switch reached	7	search home switch in negative direction, ignore end switches	8	search home switch in positive direction, ignore end switches	1... 8	RWE
1	search left stop switch only																			
2	search right stop switch, then search left stop switch																			
3	search right stop switch, then search left stop switch from both sides																			
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 negative direction, ignore end switches																			
8	search home switch in positive direction, ignore end switches																			
194	Reference search speed	For the reference search this value directly specifies the search speed.	0... 2047	RWE																
195	Reference switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.	0... 2047	RWE																
196	Reference switch distance	This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3).	0... 2.147.483.647	R																
197	Last reference position	Reference search: the last position before setting the counter to zero can be read out.	$-2^{31} \dots 2^{31}-1$ [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 $I_{peak} = < value > \times \frac{1.6A}{255}$ $I_{RMS} = < value > \times \frac{1.1A}{255}$	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 (stallGuard2).	0... 1023	R																

Number	Axis Parameter	Description	Range [Unit]	Acc.
208	TMC260 driver error flags	Bit 0 stallGuard2 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 down) Bit 4 Short to ground B (1: short condition detected, driver currently shut down) Bit 5 Open load A (1: no chopper event has happened during the last period with constant coil polarity) Bit 6 Open load B (1: no chopper event has happened during the last period with constant coil polarity) Bit 7 Stand still (1: no step impulse occurred on the step input during the last 2 ²⁰ clock cycles) <i>Please refer to the TMC260 Datasheet for more information.</i>	0/1	R
213	Group index	All motors on the module which have the same group index will get the same commands when a ROL, ROR, MST, MVP or RFS is issued for one of these motors.	0... 255	RW
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

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

5.1 stallGuard2 Related Parameters

The module is equipped with TMC260 motor driver chip. The TMC260 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.

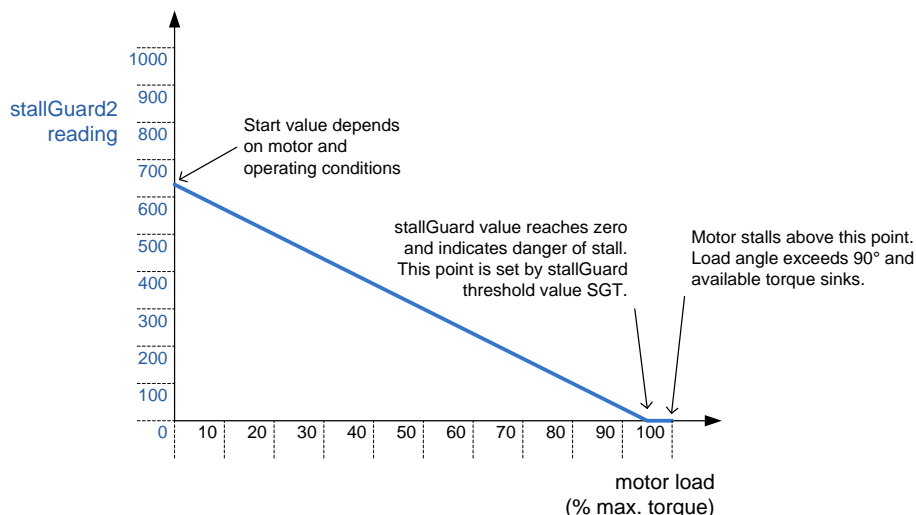


Figure 5.1 Principle function of stallGuard2

Stall detection means that the motor will be stopped when the load gets too high. This level is set using axis parameter 174 (stallGuard2 threshold). In order to exclude e.g. resonances during motor acceleration and deceleration phases it is also possible to set a minimum speed for motor being stopped due to stall detection using axis parameter #181. Stall detection can also be used for finding the reference point.

Do not use RFS in this case.

PARAMETERS NEEDED FOR ADJUSTING THE STALLGUARD2 FEATURE

Number	Axis Parameter	Description																																								
6	Absolute max. current (CS / Current Scale)	<p>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.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <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... 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																																								
173	stallGuard2 filter enable	<p>Enables the stallGuard2 filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps. <i>In most cases it is expedient to set the filtered mode before using coolStep. Use the standard mode for step loss detection.</i></p> <p>0 - standard mode 1 - filtered mode</p>																																								

Number	Axis Parameter	Description						
174	stallGuard2 threshold	This signed value controls stallGuard2 <i>threshold</i> level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2 less sensitive and requires more torque to indicate a stall. <table border="1" style="margin-left: 40px;"> <tr> <td>0</td> <td>Indifferent value</td> </tr> <tr> <td>1... 63</td> <td>less sensitivity</td> </tr> <tr> <td>-1... -64</td> <td>higher sensitivity</td> </tr> </table>	0	Indifferent value	1... 63	less sensitivity	-1... -64	higher sensitivity
0	Indifferent value							
1... 63	less sensitivity							
-1... -64	higher sensitivity							
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.						
206	Actual load value	Readout of the actual load value used for stall detection (stallGuard2).						

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

5.2 coolStep Related Parameters

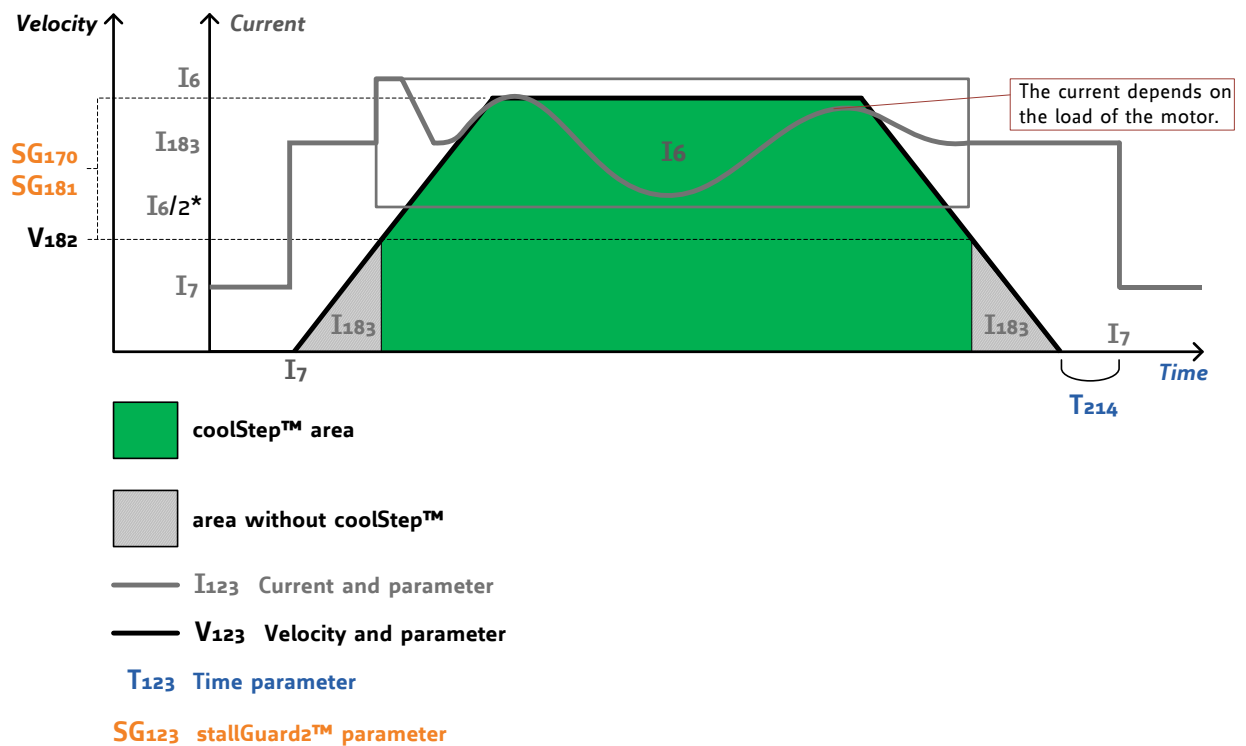
The figure below gives an overview of the coolStep related parameters. Please have in mind that the figure shows only one example for a drive. There are parameters which concern the configuration of the current. Other parameters are for velocity regulation and for time adjustment.

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

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

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

coolStep™ adjustment points and thresholds



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

PARAMETERS NEEDED FOR ADJUSTING THE COOLSTEP FEATURE

Number	Axis parameter	Description																																								
6	Absolute max. current (CS / Current Scale)	<p>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.</p> <table border="1"> <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... 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.																																								
168	smartEnergy current minimum (SEIMIN)	<p>Sets the lower motor current limit for coolStep operation by scaling the CS (Current Scale, see axis parameter 6) value.</p> <p>Minimum motor current: 0 – 1/2 of CS 1 – 1/4 of CS</p>																																								
169	smartEnergy current down step	<p>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:</p> <p>Scaling: 0... 3: 32, 8, 2, 1 0: slow decrement 3: fast decrement</p>																																								
171	smartEnergy current up step	<p>Sets the current increment step. The current becomes incremented for each measured stallGuard2 value below the lower threshold (see smartEnergy hysteresis start).</p> <p>current increment step size: Scaling: 0... 3: 1, 2, 4, 8 0: slow increment 3: fast increment / fast reaction to rising load</p>																																								
183	smartEnergy slow run current	Sets the motor current which is used below the threshold speed. Please adjust the threshold speed with axis parameter 182.																																								
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.																																								
181	Stop on stall	Below this speed motor will not be stopped. Above this speed motor will stop in case stallGuard2 load value reaches zero.																																								
182	smartEnergy threshold speed	Above this speed coolStep becomes enabled.																																								
214	Power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec).																																								

5.3 Reference Search

The built-in reference search feature (instruction RFS, no. 13) offers switching point calibration and supports up-to two end / stop / limit switches and one home switch. Depending on the selected reference search mode (see table below), one of the limit switches or the home switch might be used as reference switch and the end / limit switches are either respected or ignored during the reference search. The internal operation is based on a state machine that can be started, stopped and monitored at any time during the reference search operation (instruction RFS, no. 13).

REMARKS ON REFERENCE SEARCH OPERATION:

- End / limit switches: enable / disable of the automatic stop function corresponding to the end / limit switches (axis parameters 12 and 13) has no influence on the reference search.
- Initial search speed: until the reference switch is found for the first time, the reference search speed specified by axis parameter 194 is used. (see figures below for more details)
- Search speed after reference switch has been triggered: As soon as the reference switch has been detected the motor will switch to the reference switch speed (axis parameter 195). (see figures below for more details)
- Search one of the end / stop / limit switches (from one side): when hitting the reference switch during movement, the motor stops and slowly moves backwards until the switch is released again. Motor stops again, reverses direction and slowly moves forward until the reference switch is triggered again. The reference point will be set to the center between release and trigger point of the reference switch on one side. With this mode the reference switch is not "overrun" – instead the middle between the two transitions of the switch on one side open -> close and close -> open is used as reference.
- Search one of the end / stop / limit switches or the home switch from both sides: when hitting the reference switch during movement, the motor will continue moving until the switch is released again. Motor stops then, reverses direction and slowly moves forward until the switch is triggered again. The motor continues movement at low speed until the switch is released again, stops, reverses direction, moves towards the reference switch at low speed again until it is triggered again. It will then move to the middle of the switch between the two positions where the switch has been activated / triggered at low speed from both sides. With this mode the reference switch is "overrun" and the motor stops in the middle between activation of the switch from the left and the right side. This mode is usually used with the home switch.
- Using axis parameter 193 the reference search mode can be selected (see next chapter for more details on different reference search modes) before the actual reference search is started using the RFS command (command no. 13).
- End / limit / stop switches: With default configuration it is expected that both end / limit / stop switches are normally closed (N.C.) switches. These switches should be connected between the REF_L or REF_R input of one of the reference switch connectors 0..5 (depending on the desired motor axis) and ground (GND). With default configuration the inputs offer pull-up resistors to +5V. As soon as one of the switches is activated (open) the input level will go high due to the internal pull-ups. For other types of switches and / or other external connections this can be changed in software. Internal pull-ups can be deactivated using the command SIO 0, 0, 0 (for all axes – please see SIO command for more details and different configuration options). This way a normally open (N.O.) switch may be connected between supply voltage (e.g. +24V) and the REF_L or REF_R input of one of the reference switch connectors 0..5 (depending on the desired motor axis). The input will be low as long as the switch is not activated and turning high as soon as the switch is activated. For other configurations it is also possible to invert the polarity of the input (see global parameter 79).
- Home switch: as general purpose inputs are used for home switch connection there are no internal pull-ups which can be switched on or off. For the standard reference search modes (see axis parameter 193 for more details) it is expected that a normally open (N.O.) switch is connected between e.g. +24V supply voltage and one of the general purpose inputs used as home switch input (IN_1 for motor 0, IN_2 for motor1, IN_3 for motor2, IN_5 for motor3, IN_6 for motor4 and IN_7 for motor5). In case a normally closed (N.C.) switch is used between home switch input and supply ground (as with the End / Limit / Stop switches) an external pull-up resistor e.g. to +24V should be added.

AXIS PARAMETERS RELATED TO REFERENCE SEARCH

Number	Axis Parameter	Description																
9	Reference switch status	The logical state of the home switch.																
10	Right limit switch status	The logical state of the right limit switch.																
11	Left limit switch status	The logical state of the left limit switch (in three switch mode)																
12	Right limit switch disable	If set, deactivates the stop function of the right switch (not applicable for the reference search algorithm)																
13	Left limit switch disable	If set, deactivates the stop function of the left switch (not applicable for the reference search algorithm)																
149	Soft stop flag	If cleared, the motor will stop immediately (disregarding motor limits), when the end / stop / limit switch is hit.																
193	Reference search mode	<table border="1"> <tr><td>1</td><td>search left stop switch only</td></tr> <tr><td>2</td><td>search right stop switch, then search left stop switch</td></tr> <tr><td>3</td><td>search right stop switch, then search left stop switch from both sides</td></tr> <tr><td>4</td><td>search left stop switch from both sides</td></tr> <tr><td>5</td><td>search home switch in negative direction, reverse the direction when left stop switch reached</td></tr> <tr><td>6</td><td>search home switch in positive direction, reverse the direction when right stop switch reached</td></tr> <tr><td>7</td><td>search home switch in negative direction, ignore end switches</td></tr> <tr><td>8</td><td>search home switch in positive direction, ignore end switches</td></tr> </table> <p>Additional functions:</p> <ul style="list-style-type: none"> - Add 128 to a mode value for inverting the home switch polarity (can be used with mode 5... 8). - Add 64 to a mode value for exchanging left and right stop switches (can be used with mode 1... 4). 	1	search left stop switch only	2	search right stop switch, then search left stop switch	3	search right stop switch, then search left stop switch from both sides	4	search left stop switch from both sides	5	search home switch in negative direction, reverse the direction when left stop switch reached	6	search home switch in positive direction, reverse the direction when right stop switch reached	7	search home switch in negative direction, ignore end switches	8	search home switch in positive direction, ignore end switches
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 negative direction, ignore end switches																	
8	search home switch in positive direction, ignore end switches																	
194	Reference search speed	For the reference search this value directly specifies the search speed.																
195	Reference switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.																
196	Reference switch distance	This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3).																

5.3.1 Reference Search Modes (Axis Parameter 193)

The following figures explain the actual motor movement during the different reference search modes in more detail. A linear stage with two end points and a moving slider is taken as example.

SAP 193, 0, 1

Search left stop switch, only

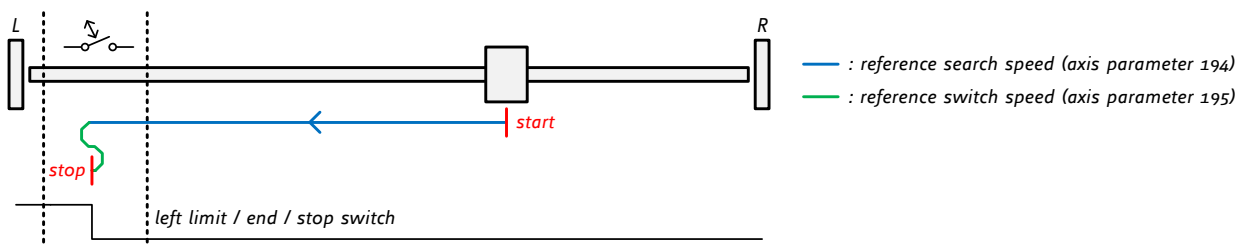


Figure 5.2: Search left stop switch, only

SAP 193, 0, 2

Search right stop switch, then search left stop switch

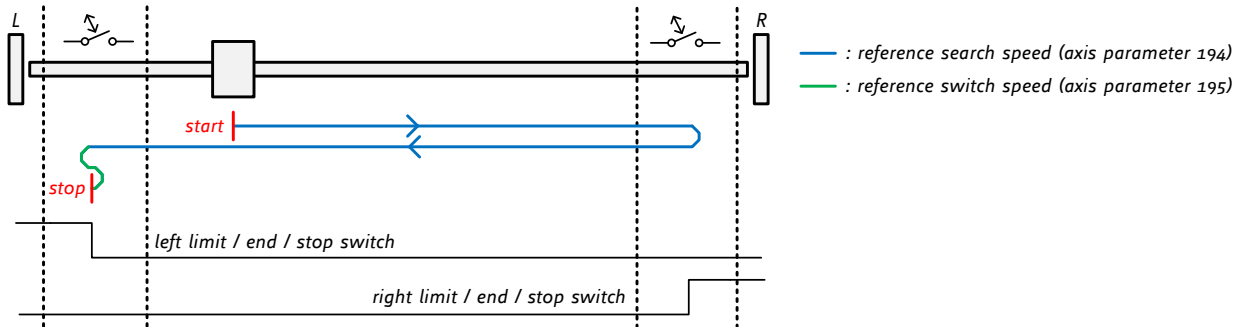


Figure 5.3: Search right stop switch, then search left stop switch

SAP 193, 0, 3

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

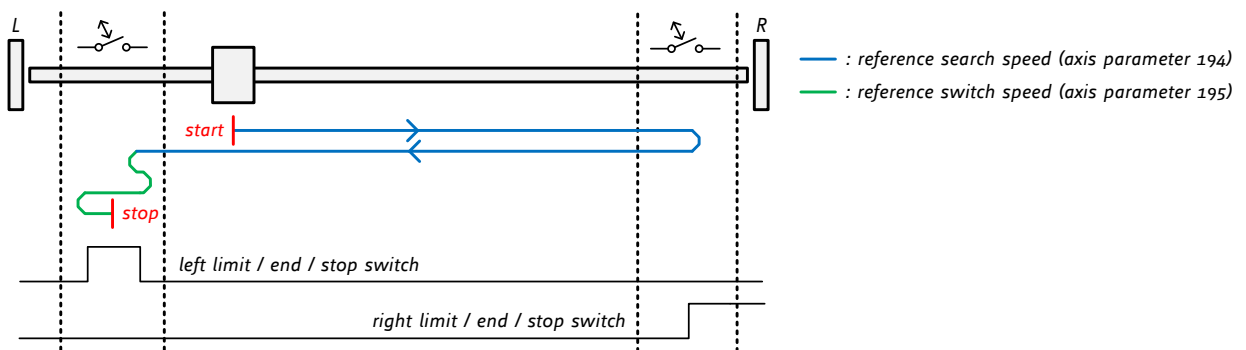


Figure 5.4: Search right stop switch, then search left stop switch from both sides

SAP 193, 0, 4

Search left stop switch from both sides

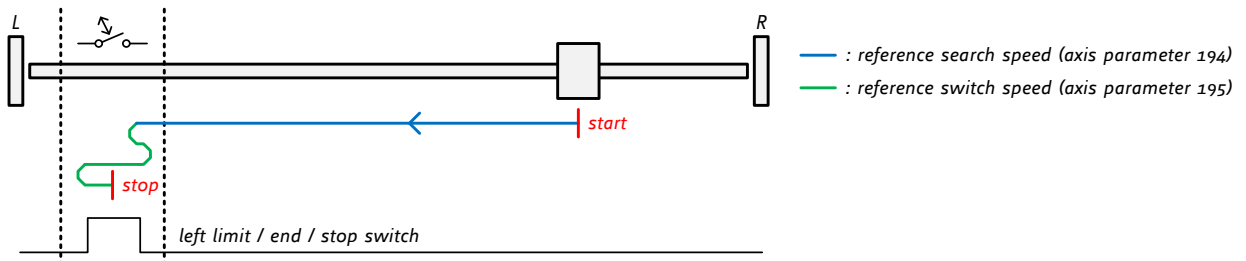


Figure 5.5: Search left stop switch from both sides

SAP 193, 0, 5

Search home switch in negative direction, reverse direction in case left stop switch is hit

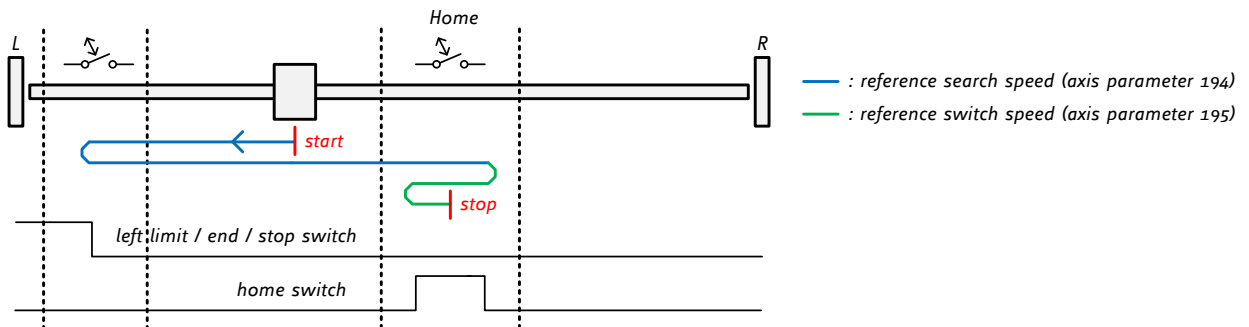


Figure 5.6: Search home switch in negative direction, reverse direction in case left stop switch is hit

SAP 193, 0, 6

Search home switch in positive direction, reverse direction in case right stop switch is hit

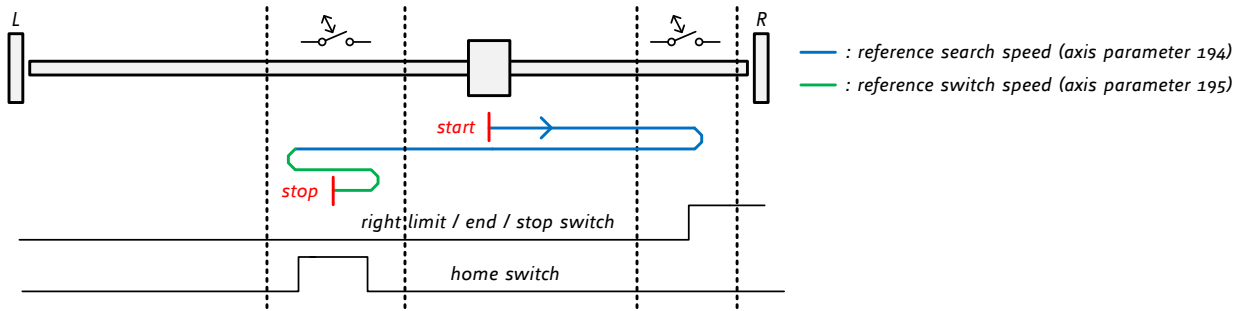


Figure 5.7: Search home switch in positive direction, reverse direction in case right stop switch is hit

SAP 193, 0, 7

Search home switch in negative direction, ignore end switches

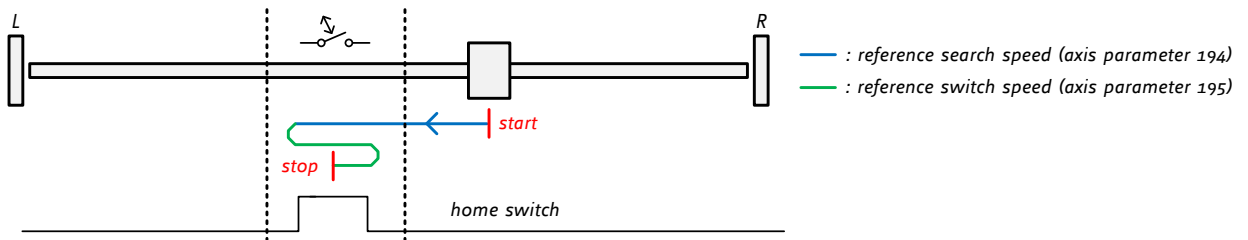


Figure 5.8: Search home switch in negative direction, ignore end switches

SAP 193, 0, 8

Search home switch in positive direction, ignore end switches

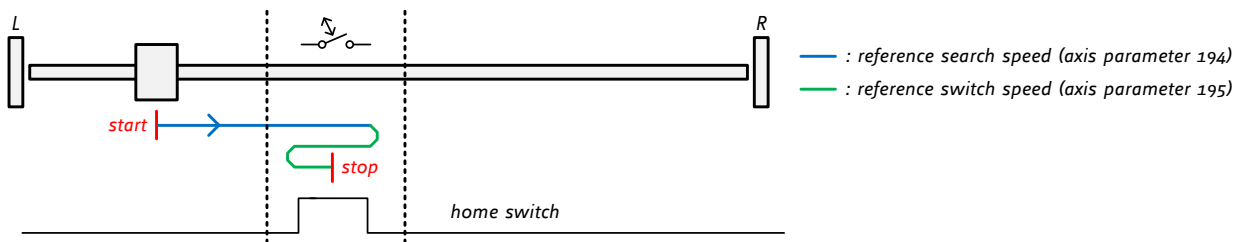


Figure 5.9: Search home switch in positive direction, ignore end switches

5.4 Grouping Motors

For grouping motors, a "group index" can be set for every motor using SAP 213. The following rules apply for the group index:

A motor that has a group index of 0 does not belong to any group at all

Motors that have the same group index belong to the same group

All motors that belong to one group must be controlled by the same TMCM-6110 module.

This method allows a great flexibility for grouping motors together. For example, it is possible to form two groups of three motors, or three groups of two motors, or one group of four motors or any other combinations.

For example, to form one group of motors 1, 3 and 5 and one group of motors 0 and 2 and motor 4 not belonging to any group, set the group indices of motor 1, 3, and 5 to 1, the group indices of motor 0 and 2 to 2 and the group index of motor 4 to 0 by issuing the following commands:

```
SAP 213, 1, 1
```

```
SAP 213, 3, 1
```

```
SAP 213, 5, 1
```

```
SAP 213, 0, 2
```

```
SAP 213, 2, 2
```

```
SAP 213, 4, 0
```

The groups are applied to the commands ROL, ROR, MST, MVP ABS and MVP REL. If one of these commands is issued for one motor that belongs to a group, all the other motors belonging to that group will also get the same command. In the current TMCL version (6.08), grouping does not apply to all the other commands. So, acceleration, velocity and other parameters have to be set separately for every motor. This might change in future versions.

The group settings also apply for deviation errors or motor stall: When a motor that belongs to a group is stopped because of a stall or a deviation error, all other motors belonging to this group will also be stopped.

To remove a motor from a group just set its group index to zero.

5.5 Calculation: Velocity and Acceleration vs. Microstep- and Fullstep-Frequency

The values of the axis parameters, sent to the TMC429 do not have typical motor values, like rotations per second as velocity. But these values can be calculated from the TMC429 parameters, as shown in this document.

TMC429 VELOCITY PARAMETERS

TMC429 velocity parameters	Related TMC6110 axis parameters	Range TMC429 and TMC6110																		
Velocity	<table border="1"> <tr><td>Axis parameter 2</td><td>target (next) speed</td></tr> <tr><td>Axis parameter 3</td><td>actual speed</td></tr> <tr><td>Axis parameter 4</td><td>maximum positioning speed</td></tr> <tr><td>Axis parameter 13</td><td>minimum speed</td></tr> <tr><td>Axis parameter 194</td><td>referencing search speed</td></tr> <tr><td>Axis parameter 195</td><td>referencing switch speed</td></tr> </table>	Axis parameter 2	target (next) speed	Axis parameter 3	actual speed	Axis parameter 4	maximum positioning speed	Axis parameter 13	minimum speed	Axis parameter 194	referencing search speed	Axis parameter 195	referencing switch speed	0... 2047						
Axis parameter 2	target (next) speed																			
Axis parameter 3	actual speed																			
Axis parameter 4	maximum positioning speed																			
Axis parameter 13	minimum speed																			
Axis parameter 194	referencing search speed																			
Axis parameter 195	referencing switch speed																			
a_max / maximum acceleration	Axis parameter 5	0... 2047																		
μsrs / microstep resolution <i>microsteps per fullstep = 2^{μsrs}</i>	Axis parameter 140 offers the following settings: <table border="1"> <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> <i>Default: 8</i>	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																			
ramp_div / ramp divisor	Axis parameter 153: divider for the acceleration. The higher the value is, the less is the maximum acceleration <i>Default: 7</i>	0... 13																		
pulse_div / pulse divisor	Axis parameter 154: divider for the velocity. Increasing this value by one divides velocity by two. Decreasing this value by one doubles velocity. <i>Default: 3</i>	0... 13																		
f_{CLK} / clock frequency	---	16MHz																		

5.5.1 Microstep Frequency

The microstep frequency of the stepper motor is calculated with

$$\mu sf [Hz] = \frac{f_{CLK} [Hz] \cdot velocity}{2^{pulse_div} \cdot 2048 \cdot 32} \quad \mu sf: \text{microstep-frequency}$$

5.5.2 Fullstep Frequency

To calculate the fullstep frequency from the microstep frequency, the microstep frequency must be divided by the number of microsteps per fullstep.

$$fsf[Hz] = \frac{\mu sf[Hz]}{2^{\mu srs}} \quad fsf: \text{fullstep-frequency}$$

The change in the pulse rate per time unit (a : pulse frequency change per second) is given by

$$a = \frac{f_{CLK}^2 \cdot a_{max}}{2^{pulse_div+ramp_div+29}}$$

This results in acceleration in fullsteps of:

$$af = \frac{a}{2^{\mu srs}} \quad af: \text{acceleration in fullsteps}$$

Example:

Signal	Value
f_{CLK}	16 MHz
velocity	1000
a_max	1000
pulse_div	1
ramp_div	1
μsrs	6

$$\mu sf = \frac{16MHz \cdot 1000}{2^1 \cdot 2048 \cdot 32} = \underline{\underline{12207031Hz}}$$

$$fsf[Hz] = \frac{12207031}{2^6} = \underline{\underline{1907.34Hz}}$$

$$a = \frac{(16MHz)^2 \cdot 1000}{2^{1+1+29}} = \underline{\underline{119.21 \frac{MHz}{s}}}$$

$$af = \frac{119.21 \frac{MHz}{s}}{2^6} = \underline{\underline{1.863 \frac{MHz}{s}}}$$

5.5.2.1 Calculation of Number of Rotations:

A stepper motor has e.g. 72 fullsteps per rotation.

$$RPS = \frac{fsf}{fullstepsper\ rotation} = \frac{1907.34}{72} = 26.49$$

$$RPM = \frac{fsf \cdot 60}{fullstepsper\ rotation} = \frac{1907.34 \cdot 60}{72} = 1589.46$$

6 Global Parameters

GLOBAL PARAMETERS ARE GROUPED INTO 4 BANKS:

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

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

6.1 Bank 0

PARAMETERS 0... 38

The first parameters 0... 38 are only mentioned here for completeness. They are used for the internal handling of the TMCL-IDE and serve for loading microstep and driver tables. Normally these parameters remain untouched.

If you want to use them for loading your specific values with your PC software please contact TRINAMIC and ask how to do this. Otherwise you might cause damage on the motor driver!

Number	Parameter
0	datagram low word (read only)
1	datagram high word (read only)
2	cover datagram position
3	cover datagram length
4	cover datagram contents
5	reference switch states (read only)
6	TMC429 SMGP register
7... 22	driver chain configuration long words 0... 15
23... 38	microstep table long word 0... 15

PARAMETERS 64... 132

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

Number	Global parameter	Description	Range	Acc.																								
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*)	<table border="1"> <tr> <td>0</td> <td>9600 baud</td> <td><i>Default</i></td> </tr> <tr> <td>1</td> <td>14400 baud</td> <td></td> </tr> <tr> <td>2</td> <td>19200 baud</td> <td></td> </tr> <tr> <td>3</td> <td>28800 baud</td> <td></td> </tr> <tr> <td>4</td> <td>38400 baud</td> <td></td> </tr> <tr> <td>5</td> <td>57600 baud</td> <td></td> </tr> <tr> <td>6</td> <td>76800 baud</td> <td><i>Not supported by Windows!</i></td> </tr> <tr> <td>7</td> <td>115200 baud</td> <td></td> </tr> </table> <p><i>Please note: values above 7 are not supported by the hardware!</i></p>	0	9600 baud	<i>Default</i>	1	14400 baud		2	19200 baud		3	28800 baud		4	38400 baud		5	57600 baud		6	76800 baud	<i>Not supported by Windows!</i>	7	115200 baud		0... 11	RWE
0	9600 baud	<i>Default</i>																										
1	14400 baud																											
2	19200 baud																											
3	28800 baud																											
4	38400 baud																											
5	57600 baud																											
6	76800 baud	<i>Not supported by Windows!</i>																										
7	115200 baud																											
66	Serial address	The module RS485 (target) address (default: 1) <i>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.</i>	0... 255	RWE																								
67	ASCII mode	Configure the TMCL ASCII interface: Bit 0: 0 – start up in binary (normal) mode 1 – start up in ASCII mode Bits 4 and 5: 00 – echo back each character 01 – echo back complete command 10 – do not send echo, only send command reply		RWE																								
68	Serial heartbeat	Serial heartbeat for the RS485 interface. If this time limit is reached and no further command received the motor will be stopped. 0: parameter is disabled	[ms]	RWE																								
69	CAN bit rate	<table border="1"> <tr> <td>2</td> <td>20kBit/s</td> <td></td> </tr> <tr> <td>3</td> <td>50kBit/s</td> <td></td> </tr> <tr> <td>4</td> <td>100kBit/s</td> <td></td> </tr> <tr> <td>5</td> <td>125kBit/s</td> <td></td> </tr> <tr> <td>6</td> <td>250kBit/s</td> <td></td> </tr> <tr> <td>7</td> <td>500kBit/s</td> <td></td> </tr> <tr> <td>8</td> <td>1000kBit/s</td> <td><i>Default</i></td> </tr> </table>	2	20kBit/s		3	50kBit/s		4	100kBit/s		5	125kBit/s		6	250kBit/s		7	500kBit/s		8	1000kBit/s	<i>Default</i>	2... 8	RWE			
2	20kBit/s																											
3	50kBit/s																											
4	100kBit/s																											
5	125kBit/s																											
6	250kBit/s																											
7	500kBit/s																											
8	1000kBit/s	<i>Default</i>																										
70	CAN reply ID	The CAN ID for replies from the board (default: 2)	0... 7ff	RWE																								
71	CAN ID	The module (target) address for CAN (default: 1)	0... 7ff	RWE																								
73	configuration EEPROM lock flag	Write: 1234 to lock the EEPROM, 4321 to unlock it. Read: 1=EEPROM locked, 0=EEPROM unlocked.	0/1	RWE																								
75	Telegram pause time	Pause time before the reply via RS485 is sent. For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin). For CAN interface this parameter has no effect!	0... 255	RWE																								
76	Serial host address	Host address used in the reply telegrams sent back via RS485.	0... 255	RWE																								
77	Auto start mode	0: do not start TMCL application after power up (default). 1: start TMCL application automatically after power up.	0/1	RWE																								

Number	Global parameter	Description	Range	Acc.
79	End switch polarity	0 - normal polarity motor axis 0..5 1 - reverse polarity motor axis 0..2, normal polarity motor axis 3..5 2 - normal polarity motor axis 0..2, reverse polarity motor axis 3..5 3 - reverse polarity motor axis 0..5	0,1,2,3	RWE
81	TMCL code protection	Protect a TMCL program against disassembling or overwriting. 0 - no protection 1 - protection against disassembling 2 - protection against overwriting 3 - protection against disassembling and overwriting <i>If you switch off the protection against disassembling, the program will be erased first! Changing this value from 1 or 3 to 0 or 2, the TMCL program will be wiped off.</i>	0,1,2,3	RWE
82	CAN heartbeat	Heartbeat for CAN interface. If this time limit is up and no further command is noticed the motor will be stopped. 0 - parameter is disabled	[ms]	RWE
83	CAN secondary address	Second CAN ID for the module. Switched off when set to zero.	0... 7ff	RWE
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 RS485 (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 still be used to set parameters of the modules individually. In order to avoid bus collisions, the module will not send 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... 214748364 7	R
255	Disable reply	1 - reply disabled 0 - send reply	0/1	

*) 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.

6.2 Bank 1

The global parameter bank 1 is normally not available. It may be used for customer specific extensions of the firmware. Together with user definable functions (see section 4.6.36) these variables form the interface between extensions of the firmware (written in C) and TMCL applications.

6.3 Bank 2

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

Up to 56 user variables are available.

MEANING OF THE LETTERS IN COLUMN ACCESS:

Access type	Related command(s)	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable
E	STGP, RSGP	Parameter stored permanently in EEPROM

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

6.4 Bank 3

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

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

MEANING OF THE LETTERS IN COLUMN ACCESS:

Access type	Related command	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable
E	SGP, AGP	Parameter stored permanently in EEPROM

Number	Global parameter	Description	Range	Access
0	Timer 0 period (ms)	Time between two interrupts (ms)	32 bit unsigned [ms]	RWE
1	Timer 1 period (ms)	Time between two interrupts (ms)	32 bit unsigned [ms]	RWE
2	Timer 2 period (ms)	Time between two interrupts (ms)	32 bit unsigned [ms]	RWE
27	Stop left 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
28	Stop right 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
29	Stop left 1 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
30	Stop right 1 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
31	Stop left 2 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
32	Stop right 2 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
33	Stop left 3 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
34	Stop right 3 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
35	Stop left 4 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
36	Stop right 4 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
37	Stop left 5 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
38	Stop right 5 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
39	Input 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
40	Input 1 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
41	Input 2 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
42	Input 3 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
43	Input 4 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
44	Input 5 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
45	Input 6 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE
46	Input 7 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0.. 3	RWE

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 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) standalone TMCL programs look like this:

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

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

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. 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

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 stand alone 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.

© TRINAMIC Motion Control GmbH & Co. KG 2011 - 2014

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

Specifications are subject to change without notice.

All trademarks used are property of their respective owners.



9 Revision History

9.1 Firmware Revision

Version	Date	Author	Description
1.10	2011-AUG-22	OK	First version supporting all features
1.11	2011-SEP-05	OK	WAIT command updated. Inverting home switch during reference search corrected.
1.12	2011-DEC-20	OK	STAP/RSAP 140 corrected.
1.13	2012-JAN-17	OK	Adjustments related to hardware version 1.2. Full functionality preserved with hardware version 1.1. USB bus powered operation: abandonment of TMCL-program (resp. no automatic start) in case of undervoltage. If the voltage is sufficient: re-initialization resp. auto start. Parameter 200 (boost current) new.
1.14	2012-FEB-29	OK	coolStep updated. Positioning range enlarged: 32 Bit. Pin change interrupts for additional I/Os added. Heartbeat function added.
1.15	2012-MAR-12	OK	Global parameter 133 (random number) new.
1.16	2012-MAR-28	OK	Global parameter 84 (coordinate storage) new.
1.17	2012-APR-26	OK	Automatic current reduction (axis 3-5) updated.
1.18	2012-JUN-18	OK	GIO 255,2 und SIO 255, 2, x implementiert.
1.19	2012-JUN-26	OK	Reference switch inputs can be inverted.
1.20	2012-OKT-04	OK	Global parameter 87 (secondary address for RS232/RS485) added. Reference search: the last position before setting the counter to zero can be read out with axis parameter 197.
1.21	2012-NOV-16	OK	Axis parameter 130 (min. current) updated.
1.22	2013-FEB-20	OK	Axis parameter 193 updated.
1.23-1.28	2014-FEB-28	OK	Not deployed.
1.29, 1.30	2014-APR-23	OK	Not deployed.
1.31	2014-APR-23	OK	Improved command <i>request target position reached</i> .

9.2 Document Revision

Version	Date	Author	Description
0.90	2011-AUG-24	SD	Preliminary version
1.00	2011-SEP-12	SD	<ul style="list-style-type: none"> - Start up new. - Axis parameter commands actualized. - New subchapters for special axis parameters added. - Chapter with TMCL programming techniques added.
1.01	2011-NOV-10	SD	<ul style="list-style-type: none"> - Information about USB interface corrected (3.1.1.2.2) - TMCL control function 137 (restore to factory defaults, 4.6.39) added.
1.02	2011-NOV-14	SD	Information about dip switch (3.1.1.2.3) added.
1.03	2012-JUL-31	SD	Design changes
1.04	2012-DEC-19	SD	Interrupts corrected. Global parameter 87 added. Design changes.
1.05	2013-JUL-22	SD	Connector types updated and pinning added.
1.06	2014-FEB-24	SD	<ul style="list-style-type: none"> - Axis parameter 9 corrected. - Examples for MVP COORD added. - Information about grouping motors added.
1.07	2014-MAY-16	SD	<ul style="list-style-type: none"> - Axis parameter 193 (reference search modes) updated. - Firmware Revision updated.

Version	Date	Author	Description
1.08	2014-DEC-10	GE	- Chapter 5.3 <i>Reference Search</i> corrected and updated. - Several minor additions for clarification
1.09	2015-FEB-24	GE	- Chapter 5.5, description and default values for axis parameter <i>pulse_div</i> und <i>ramp_div</i> corrected.

10 References

[TMC6110]	TMC6110 Hardware Manual
[TMC260]	TMC260 Datasheet
[TMC429]	TMC429 Datasheet
[TMCL-IDE]	TMCL-IDE User Manual

Please refer to www.trinamic.com.