FAULHABER

Firmware TMCL MCST 3601

# Instruction Manual

EN

W E CREATE MOTION

# Imprint

Version:
1st edition, 01.10.2014



This technical manual has been prepared with care. FAULHABER PRECISTEP SA cannot accept any liability for any errors in this technical manual or for the consequences of such errors. Equally, no liability can be accepted for direct or consequential damages resulting from improper use of the equipment.

The relevant regulations regarding safety engineering and interference suppression as well as the requirements specified in this technical manual are to be noted and followed when using the software.

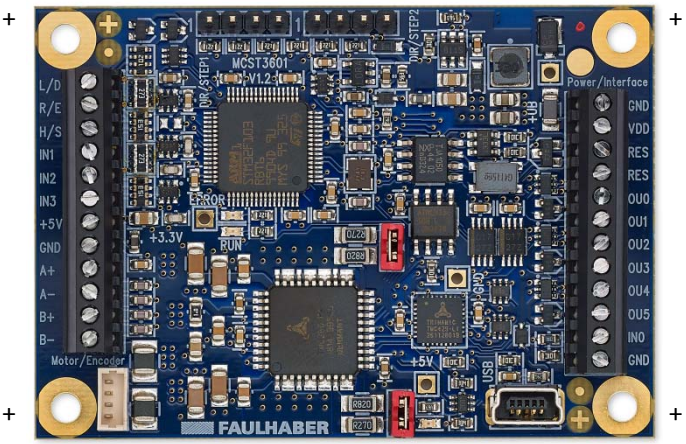
Subject to change without notice.

The respective current version of this technical manual is available on FAULHABER's internet site:
www.faulhaber.com

FAULHABER

## Firmware Version V1.33

# TMCL™ FIRMWARE MANUAL




## MCST3601

**1-Axis Stepper Controller / Driver**
**3-axes controller**
**Master / Slave operation**
**Up-to 1 A / 36 V**
**Incremental encoder input**
**GPIOs**

### UNIQUE FEATURES:

- **Compatible with the whole PRECIstep® stepper motor range**
- **Compact and fully programmable**
- **ASIC design**

**POWERED BY:** TRINAMIC MOTION CONTROL

# Table of Contents

# 1 Features

The MCST3601 is a single axis controller/driver module for 2-phase bipolar stepper motors. It supports supply voltages up-to 36V DC and motor currents up-to 1A RMS (different motor current settings selectable in software and via two jumpers). The TMCL™ firmware allows for both, standalone operation and direct mode. The module can be configured as master (controller + driver) controlling up-to two external drivers in addition to the on-board one or as slave (driver only) with step/direction/enable inputs.

#### 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

### Interfaces

- USB device interface (on-board mini-USB connector)
  6x open drain outputs (24V compatible)
- REF_L / REF_R / HOME switch inputs (24V compatible with programmable pull-ups)
- 1x S/D input for the on-board driver (on-board motion controller can be deactivated)
- 2x Step / direction output for two separate external drivers (in addition to the on-board)
- 1x encoder input for incremental A/B/I encoder
- 3x general purpose digital inputs (24V compatible)
- 1x analog input (0 .. 10V)

*Please note: not all functions are available at the same time as connector pins are shared*

### Software

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

### Electrical and mechanical data

- Supply voltage: +24 V DC nominal (9… 36 V DC)
- Motor current: up to 1 A RMS / 1.5 A peak (programmable)
- Board size: 68mm + 47.5mm

# 2 Overview

The software running on the microprocessor of the MCST3601 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.

The firmware is related to the standard TMCL™ firmware 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 an energy efficient high current high precision microstepping driver IC for bipolar stepper motors.

All commands and parameters available with this unit are explained on the following pages.

# 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.

The MCST3601 is able to control up to three motors. In this chapter it is explained how to start with one motor (motor number 0), only. If you want to use the module for controlling more motors, refer to the Hardware Manual, please. There you will find information about extensions.

THINGS YOU NEED

- MCST3601 with appropriate stepper motor
- Power supply with nominal supply voltage of +24V DC (+9… +36V DC) for your module
- PC with USB interface
- TMCL-IDE program (can be downloaded free of charge from www.trinamic.com. Please refer to the TMCL-IDE User Manual, too)
- Appropriate cables – at least for power supply, communication and motor

**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 +36V DC!***
***Do not connect or disconnect the motor while powered on!***
***START WITH POWER SUPPLY OFF!***

Figure 3.1: MCST3601 connectors

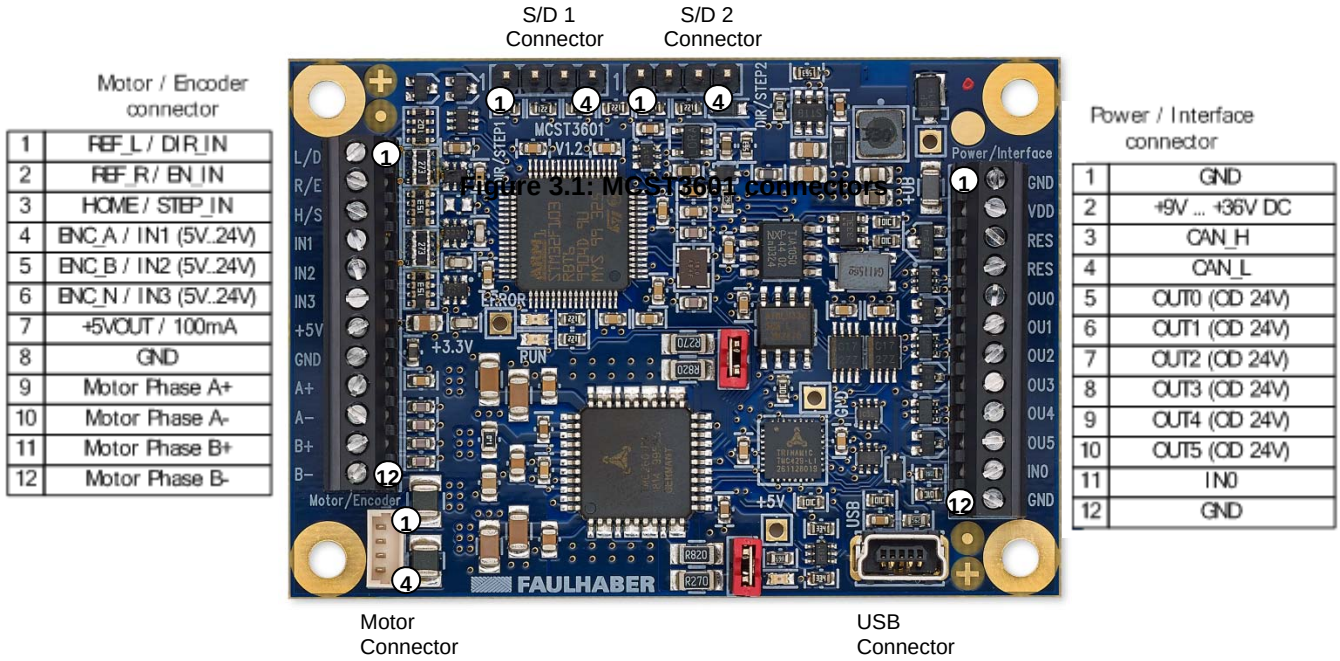



**Figure 3.2: MCST3601 connectors**

## 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 connection between PC and the USB interface of the MCST3601 for communication.

#### 3.1.1.1 Communication

##### 3.1.1.1.1 USB

*Before using the USB interface the device driver has to be installed.*

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

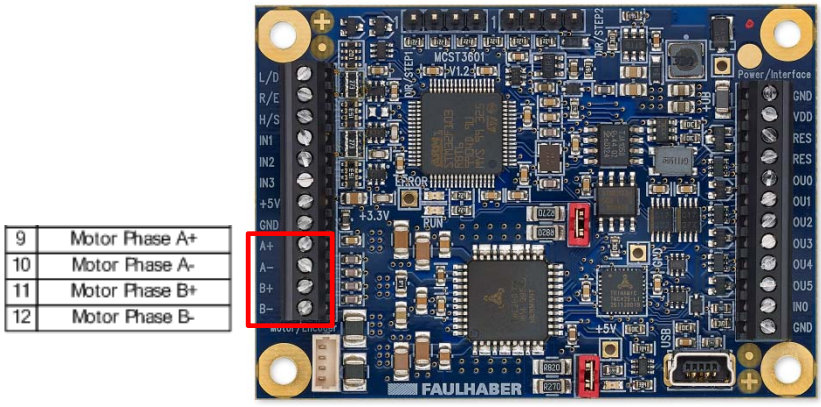
#### 3.1.1.2 Motor

The MCST3601 controls and drives one 2-phase stepper motor, directly (a second and third one via additional external driver). Connect one coil of the motor to the terminal marked *A+* and *A-* and the other coil to the connector marked *B+* and *B-*.

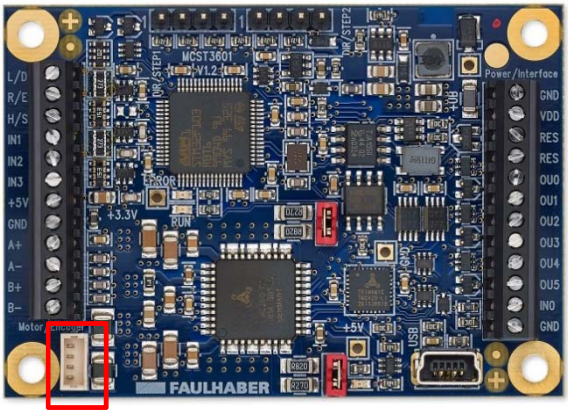
*Before connecting a motor please make sure which cable belongs to which coil. Wrong connections may lead to damage of the driver chips or the motor!*

*The MCST3601 offers two connection options for connecting the motor. Please use only one option at the same time!*

*Motor connection option 1 (using the screw terminals):*



*Motor connection option 2 (using the on-board Molex PicoBlade™ 4pin 1.25mm pitch connector):*




| | Pin | Label | Direction | Description |
|---|---|---|---|---|
|  | 1 | Motor Phase A+ | Output | Motor driver output, coil A |
| | 2 | Motor Phase A- | Output | Motor driver output, coil A |
| | 3 | Motor Phase B+ | Output | Motor driver output, coil B |
| | 6 | Motor Phase B- | Output | Motor driver output, coil B |



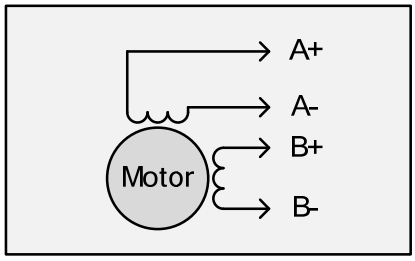
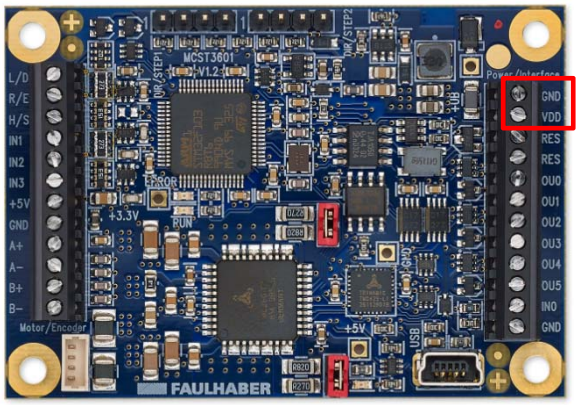



**Figure 3.3: Motor connection**

#### 3.1.1.3 Power Supply

Connect the power supply with the power supply terminals (see Figure 3.1), but, start with power supply OFF.

*Take care of the polarity, wrong polarity can destroy the board!*
*Do not exceed the maximum power supply of +36V DC!*




Power / Interface connector

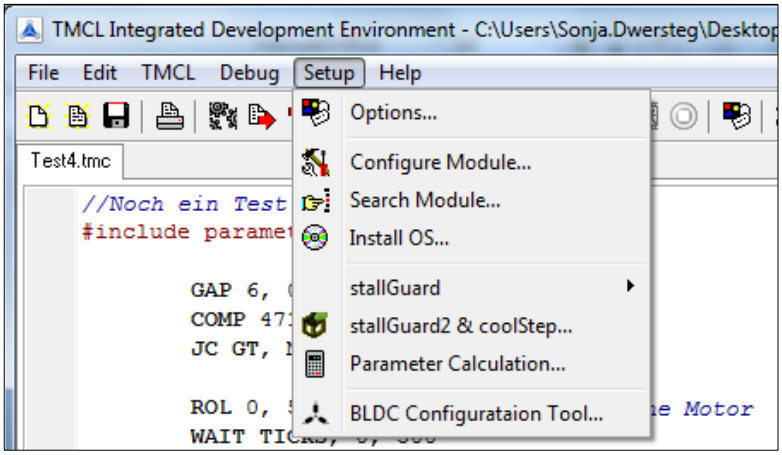
| | |
|---|---|
| 1 | GND |
| 2 | +9V ... +36V DC |

### 3.1.2 Start the TMCL-IDE Software Development Environment

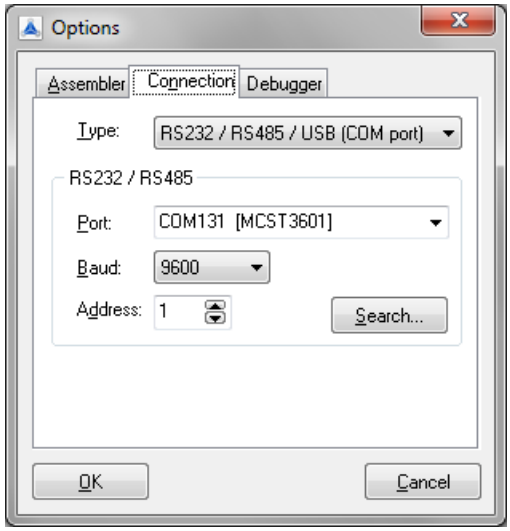
The TMCL-IDE is available on www.trinamic.com.

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***.



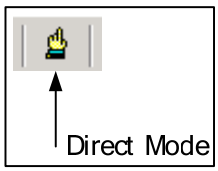

For USB choose ***COM port*** and ***Type*** with the parameters shown below. Click *OK*.




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

### 3.1.3 Using TMCL™ Direct Mode

Start TMCL™ ***Direct Mode***.

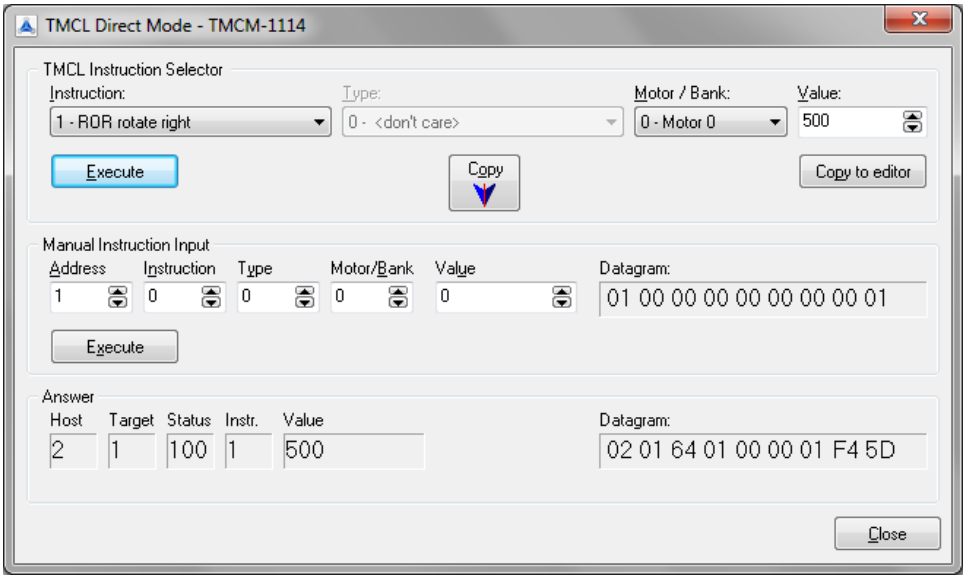



If the communication is established the MCST3601 is automatically detected (using the latest TMCL-IDE).

***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.

ATTENTION
*As the MCST3601 is able to control up to three motors the motor numbers for the three motors are 0, 1, and 2. If only one motor is connected the motor number is always 0.*




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

### 3.1.4 Important Motor Settings

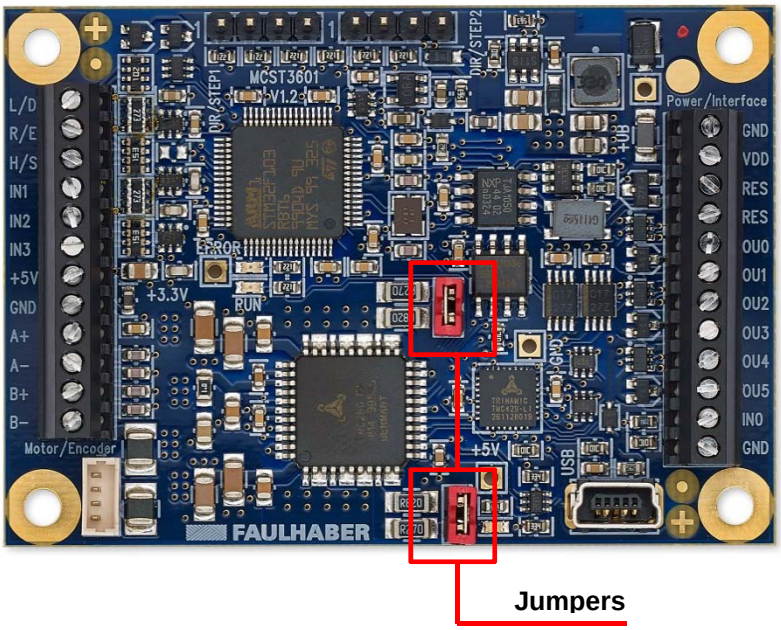
There are some axis parameters which have to be adjusted right in the beginning after installing your module. Please set the upper limiting values for the speed (axis parameter 4), the acceleration (axis parameter 5), and the current (axis parameter 6). Further set the standby current (axis parameter 7) and choose your microstep resolution with axis parameter 140.

Use the *SAP (Set Axis Parameter)* command for adjusting these values. The SAP command is described in paragraph 4.5.5. You can use the TMCM-IDE direct mode to easily configure your module.

**ATTENTION**

The most important motor setting is the *absolute maximum motor current* setting, since too high values might cause motor damage! In addition to the settings in the software please also select the correct settings of the two on-board jumpers for motor current range selection.

*Motor current range selection via two on-board jumpers:*




| Jumper | Description |
|---|---|
| Closed | Max. motor current 1A RMS / 1.5A peak (with VSENSE = 0 (programmable))<br>Max. motor current 0.57A RMS / 0.8A peak (with VSENSE = 1 (programmable)) |
| Open | Max. motor current 0.26A RMS / 0.37A peak (with VSENSE = 0 (programmable))<br>Max. motor current 0.14A RMS / 0.20A peak (with VSENSE = 1 (programmable)) |

**IMPORTANT AXIS PARAMETERS FOR MOTOR SETTING**

<table>
<tr><th>Number</th><th>Axis Parameter</th><th>Description</th><th>Range [Unit]</th></tr>
<tr><td>4</td><td>maximum positioning speed</td><td>Should not exceed the physically highest possible value. Adjust the pulse divisor (axis parameter 154), if the speed value is very low (<50) or above the upper limit. See TMC 429 datasheet for calculation of physical units or use the TMCL-IDE calculation tool.</td><td>0… 2047<br><br>$\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$</td></tr>
<tr><td>5</td><td>maximum acceleration</td><td>The limit for acceleration and deceleration. Changing this parameter requires re-calculation of the acceleration factor and the acceleration divisor. Therefore adjust the ramp divisor (axis parameter 153) carefully in steps of one.<br>See TMC 429 datasheet for calculation of physical units or use the TMCL-IDE calculation tool.</td><td>0… 2047*[1]</td></tr>
<tr><td>6</td><td>absolute max. current<br>(CS / Current Scale)</td><td>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>
<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>
This is the most important adjustment which has to be made according to the selected motor, since too high values might cause motor damage!</td><td>With jumpers set and Vsense = 0 (see parameter 179):<br>$I_{peak} =< value >\times \frac{1.5A}{255}$<br>$I_{RMS} =< value >\times \frac{1A}{255}$<br>With jumpers set and Vsense = 1 (see parameter 179):<br>$I_{peak} =< value >\times \frac{0.8A}{255}$<br>$I_{RMS} =< value >\times \frac{0.57A}{255}$<br>Without jumpers and Vsense = 0 (see parameter 179):<br>$I_{peak} =< value >\times \frac{0.37A}{255}$<br>$I_{RMS} =< value >\times \frac{0.26A}{255}$<br>Without jumpers and Vsense = 1 (see parameter 179):<br>$I_{peak} =< value >\times \frac{0.20A}{255}$<br>$I_{RMS} =< value >\times \frac{0.147A}{255}$</td></tr>
<tr><td>7</td><td>standby current</td><td>The current limit two seconds after the motor has stopped.<br>The conversion between settings and motor current is the same as for axis parameter 6.<br><br>Please note that the value of Vsense (axis parameter 179) and jumper settings are the same for axis parameter 6 and this parameter.</td><td>0… 255<br><br>Same conversion as for axis parameter 6</td></tr>
</table>

| Number | Axis Parameter | Description | | Range [Unit] |
|---|---|---|---|---|
| 140 | microstep resolution | 0 | full step | 0… 8 |
| | | 1 | half step | |
| | | 2 | 4 microsteps | |
| | | 3 | 8 microsteps | |
| | | 4 | 16 microsteps | |
| | | 5 | 32 microsteps | |
| | | 6 | 64 microsteps | |
| | | 7 | 128 microsteps | |
| | | 8 | 256 microsteps | |
| 179 | Vsense | sense resistor voltage based current scaling<br>**0:** Full scale sense resistor voltage is max. 1A RMS / 1.5A peak (with jumper closed) or max. 0.26A RMS / 0.37A peak (with jumper open)<br>**1:** Full scale sense resistor voltage is max. 0.57A RMS / 0.8A peak (with jumper closed) or max. 0.14A RMS / 0.24A peak (with jumper open) | | 0/1 |

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

# 4 TMCL™ and TMCL-IDE

The MCST3601 supports TMCL™ direct mode (binary commands) and standalone TMCL™ program execution. You can store up to 2048 TMCL™ instructions on it.

In direct mode and most cases the TMCL™ communication over USB follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the MCST3601. 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 USB to the bus master. Only then should the master transfer the next command. Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.

The Trinamic Motion Control Language [TMCL™] provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM module to form programs that run standalone on the module. For this purpose there are not only motion control commands but also commands to control the program structure (like conditional jumps, compare and calculating).

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

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

## 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 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.

**The binary command format for 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.

**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 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.
- 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 the 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.

### 4.3.1 Testing with a Simple TMCL™ Program

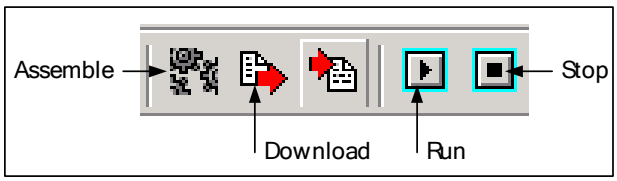
Open the file test2.tmc of the TMCL-IDE. The test program is written for three motors. Change the motor numbers into 0, if only one motor is connected.

Now, the test program looks as follows:

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

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

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




1. Click on Icon ***Assemble*** to convert the TMCL™ into machine code.
2. Then download the program to the MCST3601 module via the icon ***Download***.
3. Press icon ***Run***. The desired program will be executed.
4. Click ***Stop*** button to stop the program.

## 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) |

| Command | Number | Parameter | Description |
|---|---|---|---|
| SAP | 5 | <parameter>, <motor number>, <value> | Set axis parameter (motion control specific settings) |
| GAP | 6 | <parameter>, <motor number> | Get axis parameter (read out motion control specific settings) |
| STAP | 7 | <parameter>, <motor number> | 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

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

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

##### 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).

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

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

#### 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 0 |
| 4 | Target position reached 1 |
| 5 | Target position reached 2 |
| 15 | stallGuard™ axis 0 |
| 21 | Deviation axis 0 |
| 27 | Left stop switch 0 |
| 28 | Right stop switch 0 |
| 29 | Left stop switch 1 |
| 30 | Right stop switch 1 |
| 31 | Left stop switch 2 |
| 32 | Right stop switch 2 |
| 39 | Input change 0 |
| 40 | Input change 1 |
| 41 | Input change 2 |
| 42 | Input change 3 |
| 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 4.5.9) 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, Timer0Irq  //define the interrupt vector
    SGP 0, 3, 1000     //configure the interrupt: set its period to 1000ms
    EI 0               //enable this interrupt
    EI 255             //globally switch on interrupt processing

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

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

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

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

***Please also take a look at the other example programs.***

## 4.5 Commands

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

### 4.5.1 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 TMC262 power driver. This makes possible choosing a velocity between 0 and 2047.

**Related commands:** ROL, MST, SAP, GAP

**Mnemonic:** ROR <motor number>, <velocity>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 1 | don't care | <motor number><br>0… 2 | <velocity><br>0… 2047 |

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

**Example:**

Rotate right motor 0, velocity = 350
*Mnemonic:* ROR 0, 350

*Binary:*

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

### 4.5.2 ROL (rotate left)

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

**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 TMC262 power driver. This makes possible choosing a velocity between 0 and 2047.

**Related commands:** ROR, MST, SAP, GAP

**Mnemonic:** ROL <motor number>, <velocity>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 2 | don't care | <motor number><br>0… 2 | <velocity><br>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 |

### 4.5.3 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 number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 3 | don't care | <motor number><br>0… 2 | 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 |

### 4.5.4 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.

*Attention:*

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

**Two operation types are available:**

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

**Internal function:** A new position value is transferred to the axis parameter #0 *target position*.

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

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 4 | 0 ABS – absolute | <motor number> 0… 2 | <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 |

**Example:**

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.***

### 4.5.5 SAP (set axis parameter)

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

**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 number>, <value>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 5 | <parameter number> | <motor number> 0… 2 | <value> |

**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:**

Set the absolute maximum current of motor to 200mA

Because of the current unit $I_{RMS} =< value >\times \frac{1A}{255}$ *) the 200mA setting has the <value> 51 (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, 0, 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 | $00 | $00 | $00 | $00 | $2f |

**) Other current units are possible because the motor current can be chosen by jumper. Please refer to chapter 5 for further information about the current unit and to the Hardware Manual for information about using jumpers.*

### 4.5.6 GAP (get axis parameter)

Most parameters of the MCST3601 can be adjusted individually for the axis. With this parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditioned jumps). In direct mode the value read is only output in the *value* field of the reply (without affecting the accumulator).

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

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

**Mnemonic:** GAP <parameter number>, <motor number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 6 | <parameter number> | <motor number> 0… 2 | 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:**

Get the maximum current of motor
*Mnemonic:* GAP 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 | $06 | $06 | $00 | $00 | $00 | $00 | $00 |

*Reply:*

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

⇨ **Status = no error, value = 128**

### 4.5.7 STAP (store axis parameter)

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

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

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

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 7 | <parameter number> | <motor number> 0… 2 | 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 |

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

**Example:**

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

*Binary:*

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

***Note: The STAP command will not have any effect when the configuration EEPROM is locked (refer to 7.1). In direct mode, the error code 5 (configuration EEPROM locked, see also section 0) will be returned in this case.***

### 4.5.8 RSAP (restore axis parameter)

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

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

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

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 8 | <parameter number> | <motor number> 0… 2 | don't care |

**Reply structure 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:**

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

*Binary:*

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

### 4.5.9 SGP (set global parameter)

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 other application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, bank 0 and bank 1 are used for global parameters. Bank 2 is used for user variables and bank 3 is used for interrupt configuration.

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

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

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

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

**Binary representation:**

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 0.

**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 |

## 4.5.10 GGP (get global parameter)

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

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

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

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

**Binary representation:**

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

**Reply in direct mode:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 0.

**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**

## 4.5.11 STGP (store global parameter)

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

**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 |

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 0.

**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 |

## 4.5.12 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.

**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 |

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 0.

**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 |

## 4.5.13 RFS (reference search)

The MCST3601 has a built-in reference search algorithm which can be used. The reference search algorithm provides switching point calibration and three switch modes. The status of the reference search can also be queried to see if it has already finished. (In a TMCL™ program it is better to use the WAIT command to wait for the end of a reference search.) Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (chapter 5). The reference search can be started, stopped, and the actual status of the reference search can be checked.

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

**Related commands:** WAIT

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 13 | 0 START – start ref. search<br>1 STOP – abort ref. search<br>2 STATUS – get status | <motor number><br>0… 2 | 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 | ref. search active |
| | other values | no ref. search active |

**Example:**

Start reference search of motor 0
*Mnemonic:* RFS START, 0

*Binary:*

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

***With this module it is possible to use stall detection instead of a reference search. Please refer to section 6.1 for details.***

## 4.5.14 SIO (set output)

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

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

**Related commands:** GIO, WAIT

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

**Binary representation:**

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

**Reply structure:**

| STATUS | VALUE |
|---|---|
| 100 – OK | don't care |

**Example:**

Set OUT1 to high (bank 2, output 1) -> the output 1 will be pulled low actively.
*Mnemonic:* SIO 1, 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 | $01 | $02 | $00 | $00 | $00 | $01 |

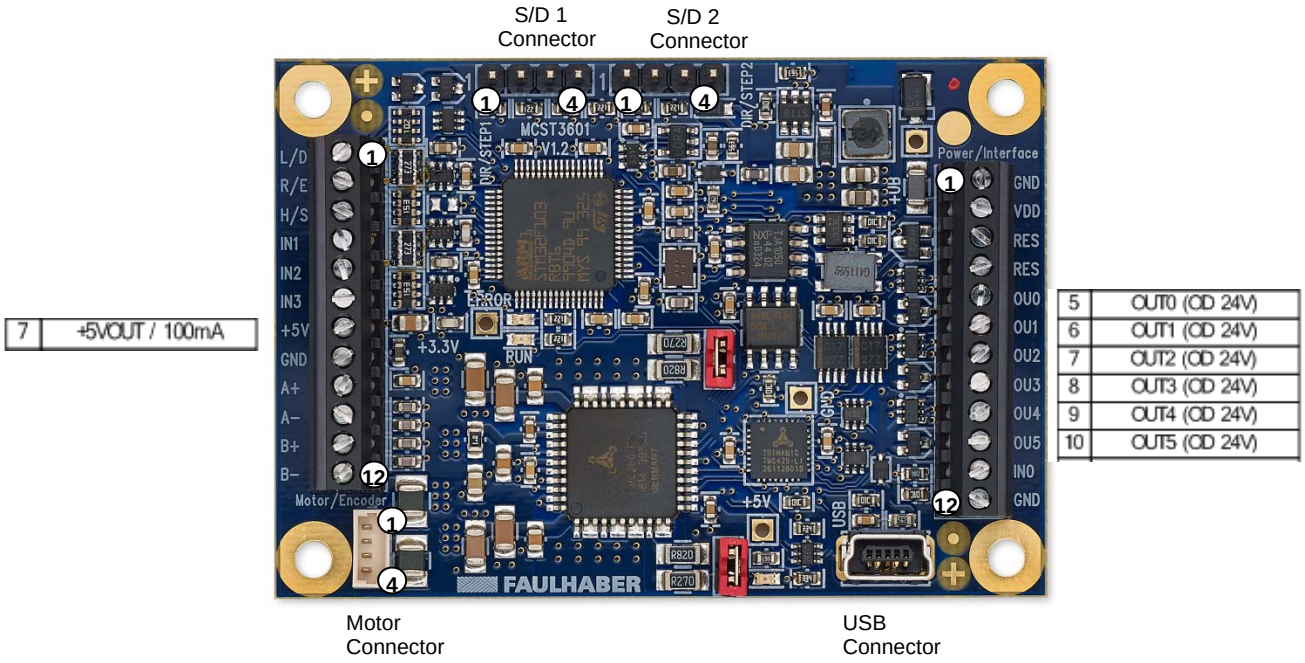



**Figure 4.1: Programmable general purpose outputs on MCST3601**

Bank 2 is used for setting the status of the general purpose digital output (OD = open drain output) either to low (low = 0, output pin floating) or to high (high = 1, output pin pulled low).

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 5 | OUT0 (OD 24V) | SIO 0, 2, <n> | 1/0 |
| 6 | OUT1 (OD 24V) | SIO 1, 2, <n> | 1/0 |
| 7 | OUT2 (OD 24V) | SIO 2, 2, <n> | 1/0 |
| 8 | OUT3 (OD 24V) | SIO 3, 2, <n> | 1/0 |
| 9 | OUT4 (OD 24V) | SIO 4, 2, <n> | 1/0 |
| 10 | OUT5 (OD 24V) | SIO 5, 2, <n> | 1/0 |

The SIO command can be used to switch on (value = 1) or off (value = 0) the +5V supply output for external circuits (Pin 7 of the Motor / Encoder connector). This +5V output might be used to supply an external encoder. As default setting this output is switched on – delivering +5V from the internal DC/DC converter.

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 7 | +5VOUT / 100mA | SIO 6, 2, <n> | 1/0 |

## 4.5.15 GIO (get input/output)

With this command the status of the available general purpose inputs of the module can be read out. The function reads a digital or analogue input port. Digital lines will 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 IN0
*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

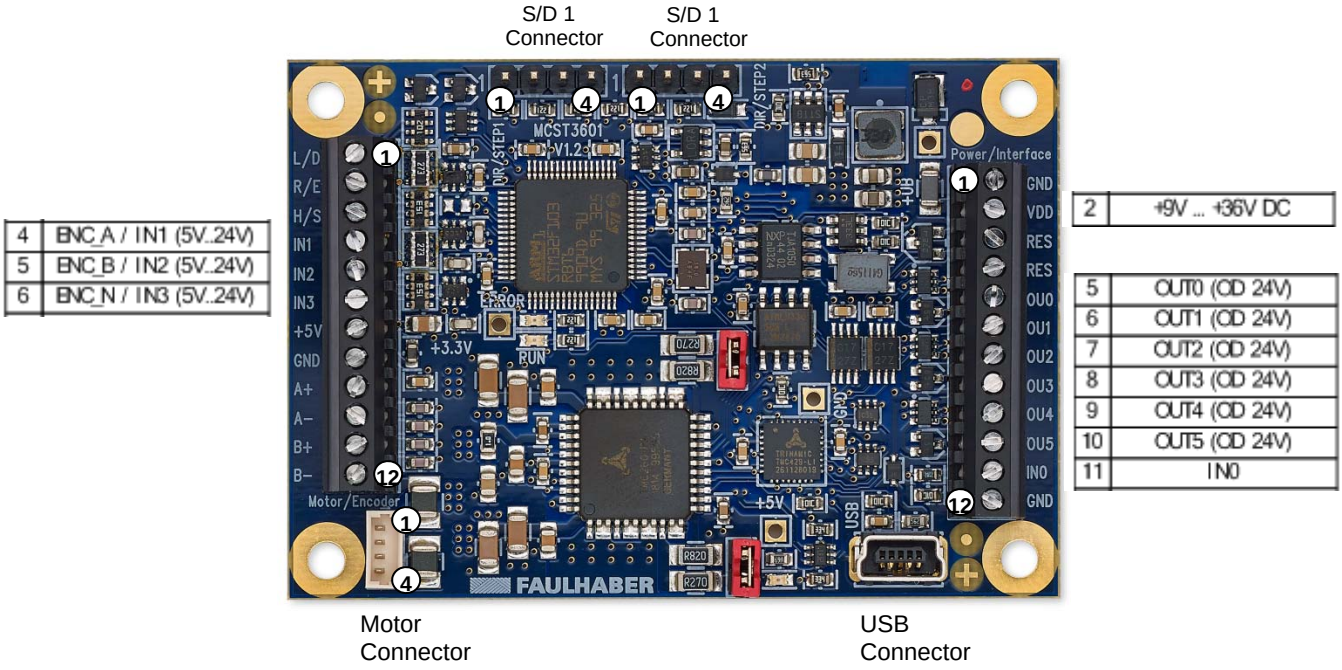



**Figure 4.2: Programmable general purpose inputs and outputs on MCST3601**

### 4.5.15.1 I/O bank 0 – digital inputs

***The voltage at the IN0..IN3 inputs can be read back as digital value.***

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 11 | IN0 | GIO 0, 0 | 0/1 |
| 4 | IN1 | GIO 1, 0 | 0/1 |
| 5 | IN2 | GIO 2, 0 | 0/1 |
| 6 | IN3 | GIO 3, 0 | 0/1 |

### 4.5.15.2 I/O bank 1 – analog inputs

***The voltage at the IN0 input can be read back as analog value via bank 1.***

| Pin | I/O port | Command | Range / Unit |
|---|---|---|---|
| 11 | IN0 | GIO 0, 1 | Read back value in the range of 0..4095 for an input voltage in the range of approx. 0..10.3V DC |
| 2 | Voltage | GIO 8, 1 | Read back supply voltage in x100mV, e.g. a value of 240 means 24.0V DC supply voltage. |

### 4.5.15.3 I/O bank 2 – the states of digital outputs

***The states of the open drain outputs OUT0..OUT5 (that have been set by SIO commands) can be read back using bank 2.***

| Pin | I/O port | Command | Range |
|---|---|---|---|
| 5 | OUT0 | GIO 0, 2 | 1/0 |
| 6 | OUT1 | GIO 1, 2 | 1/0 |
| 7 | OUT2 | GIO 2, 2 | 1/0 |
| 8 | OUT3 | GIO 3, 2 | 1/0 |
| 9 | OUT4 | GIO 4, 2 | 1/0 |
| 10 | OUT5 | GIO 5, 2 | 1/0 |

## 4.5.16 CALC (calculate)

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

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

**Mnemonic:** CALC <operation>, <operand>

**Binary representation:**

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

## 4.5.17 COMP (compare)

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

***The host address and the reply are used only to take the instruction to the TMCL™ program memory while the program downloads.***

**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 <comparison 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 |

## 4.5.18 JC (jump conditional)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program downloads.***

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

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

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 21 | 0 ZE - zero<br>1 NZ - not zero<br>2 EQ - equal<br>3 NE - not equal<br>4 GT - greater<br>5 GE - greater/equal<br>6 LT - lower<br>7 LE - lower/equal<br>8 ETO - time out error<br>9 EAL - external alarm<br>10 EDV - deviation error<br>11 EPO - position 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 |

## 4.5.19 JA (jump always)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program downloads.***

**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 |

## 4.5.20 CSUB (call subroutine)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program downloads.***

**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 |

## 4.5.21 RSUB (return from subroutine)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. This command cannot be used in direct mode.***

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

**Related command:** CSUB

**Mnemonic:** RSUB

**Binary representation:**

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

**Example:** please see the CSUB example (section 4.5.20).

*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 |

## 4.5.22 WAIT (wait for an event to occur)

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

***The host address and the reply are only used to take the instruction to the TMCL™ program memory while the program loads down. This command cannot be used in direct mode.***

**There are five different wait conditions that can be used:**

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

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

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

**Related commands:** JC, CLE

**Mnemonic:** WAIT <condition>, <motor number>, <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 number> 0… 2 | <no. of ticks* for timeout>, 0 for no timeout |
| | 2 REFSW – reference switch | <motor number> 0… 2 | <no. of ticks* for timeout>, 0 for no timeout |
| | 3 LIMSW – limit switch | <motor number> 0… 2 | <no. of ticks* for timeout>, 0 for no timeout |
| | 4 RFS – reference search completed | <motor number> 0… 2 | <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 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Function** | Target-address | Instruction Number | Type | Motor/ Bank | Operand Byte3 | Operand Byte2 | Operand Byte1 | Operand Byte0 | Checksum |
| **Value (hex)** | $01 | $1b | $01 | $00 | $00 | $00 | $00 | $00 | $1d |

## 4.5.23 STOP (stop TMCL™ program execution)

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

***End standalone TMCL™ programs with the STOP command. 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 |

## 4.5.24 SCO (set coordinate)

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

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

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

**Related commands:** GCO, CCO, MVP

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

**Binary representation:**

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

**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. |

## 4.5.25 GCO (get coordinate)

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

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

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

**Related commands:** SCO, CCO, MVP

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 31 | <coordinate number><br>0… 20 | <motor number><br>0… 2 | 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. |

## 4.5.26 CCO (capture coordinate)

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

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

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

**Related commands:** SCO, GCO, MVP

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 32 | <coordinate number><br>0… 20 | <motor number><br>0… 2 | 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 |

## 4.5.27 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 number>

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 39 | <coordinate number><br>0… 20 | <motor number><br>0… 2 | 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 |

## 4.5.28 CALCX (calculate using the X register)

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

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

**Mnemonic:** CALCX <operation>

**Binary representation:**

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

## 4.5.29 AAP (accumulator to axis parameter)

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

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

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

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 34 | <parameter number> | <motor number><br>0… 2 | <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 |

## 4.5.30 AGP (accumulator to global parameter)

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

For a table with parameters and bank numbers which can be used together with this command please refer to chapter 0.

**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 |

## 4.5.31 CLE (clear error flags)

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

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

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

**Related commands:** JC

**Mnemonic:** CLE <flags>

**Binary representation:**

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

**Example:**

Reset the timeout flag
*Mnemonic:* CLE ETO

*Binary:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **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 |

## 4.5.32 VECT (set interrupt vector)

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

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

**Related commands:** EI, DI, RETI

**Mnemonic:** VECT <interrupt number>, <label>

**Binary representation:**

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

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

| Interrupt number | Interrupt type |
|---|---|
| 0 | Timer 0 |
| 1 | Timer 1 |
| 2 | Timer 2 |
| 3 | Target position reached 0 |
| 4 | Target position reached 1 |
| 5 | Target position reached 2 |
| 15 | stallGuard™ axis 0 |
| 21 | Deviation axis 0 |
| 27 | Left stop switch 0 |
| 28 | Right stop switch 0 |
| 29 | Left stop switch 1 |
| 30 | Right stop switch 1 |
| 31 | Left stop switch 2 |
| 32 | Right stop switch 2 |
| 39 | Input change 0 |
| 40 | Input change 1 |
| 41 | Input change 2 |
| 42 | Input change 3 |
| 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 |

## 4.5.33 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 0 |
| 4 | Target position reached 1 |
| 5 | Target position reached 2 |
| 15 | stallGuard™ axis 0 |
| 21 | Deviation axis 0 |
| 27 | Left stop switch 0 |
| 28 | Right stop switch 0 |

| Interrupt number | Interrupt type |
|---|---|
| 29 | Left stop switch 1 |
| 30 | Right stop switch 1 |
| 31 | Left stop switch 2 |
| 32 | Right stop switch 2 |
| 39 | Input change 0 |
| 40 | Input change 1 |
| 41 | Input change 2 |
| 42 | Input change 3 |
| 255 | Global interrupts |
| 29 | Left stop switch 1 |

**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 |

## 4.5.34 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 0 |
| 4 | Target position reached 1 |
| 5 | Target position reached 2 |
| 15 | stallGuard™ axis 0 |
| 21 | Deviation axis 0 |
| 27 | Left stop switch 0 |
| 28 | Right stop switch 0 |

| Interrupt number | Interrupt type |
|---|---|
| 29 | Left stop switch 1 |
| 30 | Right stop switch 1 |
| 31 | Left stop switch 2 |
| 32 | Right stop switch 2 |
| 39 | Input change 0 |
| 40 | Input change 1 |
| 41 | Input change 2 |
| 42 | Input change 3 |
| 255 | Global interrupts |
| 29 | Left stop switch 1 |

**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 |

## 4.5.35 RETI (return from interrupt)

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

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

**Related commands:** EI, DI, VECT

**Mnemonic:** RETI

**Binary representation:**

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

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

*Binary format of RETI:*

| Byte Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **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 |

## 4.5.36 Customer Specific TMCL™ Command Extension (UF0… UF7 - User Function)

The user definable functions UF0… UF7 are predefined functions without topic for user specific purposes. 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 |

## 4.5.37 Request Target Position Reached Event

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

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

**Mnemonic: ---**

**Binary representation:**

| INSTRUCTION NO. | TYPE | MOT/BANK | VALUE |
|---|---|---|---|
| 138 | 0/1 | (don't care) | motor bit mask |

With command 138 the value field is a bit vector. It shows for which motors one would like to have a position reached message. The *value* field contains a bit mask where every bit stands for one motor.

**MOTOR BIT MASK**

| Bit | Selected motor |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |

**VALUES FOR TYPE**

| Value | Description |
|---|---|
| 0 | Position reached messages only for the next MVP command. |
| 1 | Position reached event message for every MVP command. |

**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 |

The additional reply will be sent when at least the first motor has reached its target position. The MCST3601 can control up to three motors.

**Additional reply in direct mode (after a motor has reached its target position):**

| 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 |

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

**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. 3601V133) |

*There is no checksum in this reply format!*

- **Type set to 1 - version number in binary format:**
  Please use the normal reply format. The version number is output in the *value* field of the reply in the following way:

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

# 5 Custom specific functions

In contrast to current standard TMCL functionality the stepper motor connected to axis 0 of the MCST3601 module will automatically be moved to the next fullstep position after power-up.

# 6 Axis Parameters

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

**ATTENTION**

The following axis parameters are only available for axis 0, because the module has only one driver IC:

#6, #7
#140
#160… #184
#204… #254

**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). | −2.147.483.648… +2.147.483.647 [µsteps] | RW |
| 1 | actual position | The current position of the motor. Should only be overwritten for reference point setting. | −2.147.483.648… +2.147.483.647 [µsteps] | RW |
| 2 | target (next) speed | The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 3 | actual speed | The current rotation speed. | ±2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 4 | maximum positioning speed | Should not exceed the physically highest possible value. Adjust the pulse divisor (axis parameter 154), if the speed value is very low (<50) or above the upper limit. See TMC 429 datasheet for calculation of physical units or use the TMCL-IDE calculation tool. | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{PD} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RWE |

<table>
<tr><th>Number</th><th>Axis Parameter</th><th>Description</th><th>Range [Unit]</th><th>Acc.</th></tr>
<tr><td>5</td><td>maximum acceleration</td><td>The limit for acceleration and deceleration. Changing this parameter requires re-calculation of the acceleration factor and the acceleration divisor. Therefore adjust the ramp divisor (axis parameter 153) carefully in steps of one.<br>See TMC 429 datasheet for calculation of physical units or use the TMCL-IDE calculation tool.</td><td>0… 2047*[1]</td><td>RWE</td></tr>
<tr><td>6</td><td>absolute max. current<br>(CS / Current Scale)</td><td>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>
<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>
The unit of the current is adequate to the chosen motor current (with or without jumper).<br>The most important motor setting, since too high values might cause motor damage!</td><td>0… 255<br><br>With jumpers set and Vsense = 0 (see parameter 179):<br>$I_{peak} =< value >\times \frac{1.5A}{255}$<br>$I_{RMS} =< value >\times \frac{1A}{255}$<br>With jumpers set and Vsense = 1 (see parameter 179):<br>$I_{peak} =< value >\times \frac{0.8A}{255}$<br>$I_{RMS} =< value >\times \frac{0.57A}{255}$<br>Without jumpers and Vsense = 0 (see parameter 179):<br>$I_{peak} =< value >\times \frac{0.37A}{255}$<br>$I_{RMS} =< value >\times \frac{0.26A}{255}$<br>Without jumpers and Vsense = 1 (see parameter 179):<br>$I_{peak} =< value >\times \frac{0.20A}{255}$<br>$I_{RMS} =< value >\times \frac{0.147A}{255}$</td><td>RWE</td></tr>
<tr><td>7</td><td>standby current</td><td>The current limit two seconds after the motor has stopped.<br>The unit of the current is adequate to the chosen motor current (with or without jumper).</td><td>0… 255<br><br>Same conversion as for axis parameter 6</td><td>RWE</td></tr>
<tr><td>8</td><td>target pos. reached</td><td>Indicates that the actual position equals the target position.</td><td>0/1</td><td>R</td></tr>
<tr><td>9</td><td>ref. switch status</td><td>The logical state of the reference (left) switch.<br>See the TMC 429 data sheet for the different switch modes. The default has two switch modes: the left switch as the reference switch, the right switch as a limit (stop) switch.</td><td>0/1</td><td>R</td></tr>
<tr><td>10</td><td>right limit switch status</td><td>The logical state of the (right) limit switch.</td><td>0/1</td><td>R</td></tr>
</table>

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 11 | left limit switch status | The logical state of the left limit switch (in three switch mode) | 0/1 | R |
| 130 | minimum speed | Should always be set 1 to ensure exact reaching of the target position. Do not change! | 0… 2047 $\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RWE |
| 135 | actual acceleration | The current acceleration (read only). | 0… 2047*[1] | R |
| 138 | ramp mode | Automatically set when using ROR, ROL, MST and MVP.<br>0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided.<br>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.<br>For special purposes, the soft mode (value 1) with exponential decrease of speed can be selected. | 0/1/2 | RWE |
| 140 | microstep resolution | 0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps | 0… 8 | 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.<br>0 – step interpolation off<br>1 – step interpolation on | 0/1 | RW |
| 161 | double step enable | Every edge of the cycle releases a step/microstep. *It does not make sense to activate this parameter for internal use.*<br>Double step enable can be used with Step/Dir interface.<br>0 – double step off<br>1 – double step on | 0/1 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 162 | chopper blank time | Selects the comparator *blank time*. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. | 0… 3 | RW |
| 163 | chopper mode | Selection of the chopper mode:<br>0 – spread cycle<br>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.<br>0 – fast decrement<br>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.<br>-3… -1 \| negative hysteresis end setting<br>0 \| zero hysteresis end setting<br>1… 12 \| positive hysteresis end setting | -3… 12 | RW |
| 166 | chopper hysteresis start | Hysteresis start setting. Please remark, that this value is an offset to the hysteresis end value. | 0… 8 | RW |
| 167 | chopper off time | The off time setting controls the minimum chopper frequency. An off time within the range of 5µs to 20µs will fit.<br><br>Off time setting for constant $t_{Off}$ chopper:<br>$N_{CLK}$= 12 + 32*$t_{OFF}$ (Minimum is 64 clocks)<br><br>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.<br>minimum motor current:<br>0 – 1/2 of CS<br>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.<br>Number of stallGuard2™ measurements per decrement:<br>Scaling: 0… 3: 32, 8, 2, 1<br>0: slow decrement<br>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.<br>Hysteresis:<br>(smartEnergy hysteresis value + 1) * 32<br>Upper stallGuard2™ threshold:<br>(smartEnergy hysteresis start + smartEnergy hysteresis + 1) * 32 | 0… 15 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 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).<br>current increment step size:<br>Scaling: 0… 3: 1, 2, 4, 8<br>0: slow increment<br>3: fast increment / fast reaction to rising load | 1… 3 | RW |
| 172 | smartEnergy hysteresis start | The lower threshold for the stallGuard2™ value (see smart Energy current up step). | 0… 15 | RW |
| 173 | stallGuard2™ filter enable | Enables the stallGuard2™ filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps.<br>*In most cases it is expedient to set the filtered mode before using coolStep™.*<br>*Use the standard mode for step loss detection.*<br>0 – standard mode<br>1 – filtered mode | 0/1 | RW |
| 174 | stallGuard2™ threshold | This signed value controls stallGuard2™ *threshold* level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes stallGuard2™ less sensitive and requires more torque to indicate a stall.<br>0 Indifferent value<br>1… 63 less sensitivity<br>-1 -64 higher sensitivity | -64… 63 | RW |
| 175 | slope control high side | Determines the slope of the motor driver outputs.<br>*Set to 2 or 3 for this module or rather use the default value.*<br>0: lowest slope<br>3: fastest slope | 0… 3 | RW |
| 176 | slope control low side | Determines the slope of the motor driver outputs.<br>*Set identical to slope control high side.* | 0… 3 | RW |
| 177 | short protection disable | 0: Short to GND protection is on<br>1: Short to GND protection is disabled<br>*Use default value!* | 0/1 | RW |
| 178 | short detection timer | 0: 3.2µs<br>1: 1.6µs<br>2: 1.2µs<br>3: 0.8µs<br>*Use default value!* | 0… 3 | RW |
| 179 | Vsense | sense resistor voltage based current scaling<br>**0:** Full scale sense resistor voltage is max. 1A RMS / 1.5A peak (with jumper closed) or max. 0.26A RMS / 0.37A peak (with jumper open)<br>**1:** Full scale sense resistor voltage is max. 0.57A RMS / 0.8A peak (with jumper closed) or max. 0.14A RMS / 0.24A peak (with jumper open) | 0/1 | RW |

| Number | Axis Parameter | Description | Range [Unit] | Acc. |
|---|---|---|---|---|
| 180 | smartEnergy actual current | This status value provides the *actual motor current* setting as controlled by coolStep™. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN.<br>actual motor current scaling factor:<br>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<br>$\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 182 | smartEnergy threshold speed | Above this speed coolStep™ becomes enabled. | 0… 2047<br>$\left[\frac{16\text{MHz}}{65536} \cdot 2^{\text{PD}} \frac{\mu\text{steps}}{\text{sec}}\right]$ | RW |
| 183 | smartEnergy slow run current | Sets the motor current which is used below the threshold speed.<br>The unit of the current is adequate to the chosen motor current (with or without jumper). | 0… 255<br>Same conversion as for axis parameter 6 | RW |
| 193 | ref. search mode | 1 search left stop switch only<br>2 search rightstop switch, then search left stop switch<br>3 search right stop switch, then search left stop switch from both sides<br>4 search left stop switch from both sides<br>5 search home switch in negative direction, reverse the direction when left stop switch reached<br>6 search home switch in positive direction, reverse the direction when right stop switch reached<br>7 search home switch in positive direction, ignore end switches<br>8 search home switch in negative direction, ignore end switches<br>*Adding 128 to these values reverses the polarity of the home switch input.* | 1… 8 | RWE |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. | 0… 2047 | RWE |
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. | 0… 2047 | RWE |
| 196 | distance end switches | This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3). | 0… 2147483647 | R |
| 204 | freewheeling | Time after which the power to the motor will be cut when its velocity has reached zero. | 0… 65535<br>0 = never<br>[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 | TMC262 driver error flags | Bit 0: stallGuard2™ status (1:threshold reached)<br>Bit 1: Overtemperature (1: driver is sht down due to overtemperature)<br>Bit 2: Pre-warning overtemperature (1: treshold is exceeded)<br>Bit 3: Short to ground A (1: short condition deteted, driver currently shut down)<br>Bit 4: Short to ground B (1: short condition detected, driver currently shut down)<br>Bit 5: Open load A (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 6: Open load B (1: no chopper event has happened during the last period with constant coil polarity)<br>Bit 7: Stand still (1: no step impulse occurred on the step input during the last 2^20 clock cycles)<br>*Please refer to the TMC262 Datasheet for more information.* | 0… 255 | R |
| 209 | encoder position | The value of an encoder register can be read out or written. | [encoder steps] | RW |
| 210 | Encoder prescaler | Prescaler for the encoder. | See paragraph 6.2.1. | RWE |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. | 0… 65535<br>[encoder steps] | RWE |
| 214 | power down delay | Standstill period before the current is changed down to standby current. The standard value is 200 (value equates 2000msec). | 1… 65535<br>[10msec] | RWE |
| 254 | Step/dir mode | 0: Normal mode. Step/dir mode off.<br>1: Step/dir mode with automatic current reduction in case of standstill. If current reduction in standstill is not desired, choose the same value for the axis parameters #6 and #7. | 0/1 | RWE |

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

## 6.1 Reference Search

The built-in reference search features switching point calibration and support of one or two reference switches. The internal operation is based on a state machine that can be started, stopped and monitored (instruction RFS, no. 13). The reference switch is connected in series with the left limit switch. The differentiation between the left limit switch and the home switch is made through software. Switches with open contacts (normally closed) are used. The analogue input AIN_0 of the module can be used as home switch.

Hints for reference search:

- The settings of the automatic stop functions corresponding to the switches (axis parameters 12 and 13) have no influence on the reference search.
- Until the reference switch is found for the first time, the searching speed is identical to the maximum positioning speed (axis parameter 4), unless reduced by axis parameter 194.
- After hitting the reference switch, the motor slowly moves until the switch is released. Finally the switch is re-entered in the other direction, setting the reference point to the center of the two switching points. This low calibrating speed is a quarter of the maximum positioning speed by default (axis parameter 195).
- Set one of the values for axis parameter 193 for selecting the reference search mode.

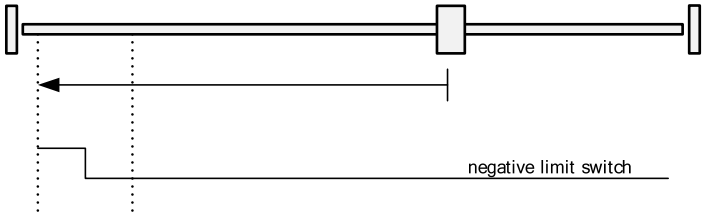
**PARAMETERS NEEDED FOR REFERENCE SEARCH**

| Number | Axis Parameter | Description |
|---|---|---|
| 9 | ref. switch status | The logical state of the reference (left) switch.<br>See the TMC 429 data sheet for the different switch modes. The default has two switch modes: the left switch as the reference switch, the right switch as a limit (stop) 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 |
| 13 | left limit switch disable | Deactivates the stop function of the left switch resp. reference switch if set. |
| 141 | ref. switch tolerance | For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop. |
| 149 | soft stop flag | If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit. |
| 193 | ref. search mode | 1 search left stop switch only<br>2 search right stop switch, then search left stop switch<br>3 search right stop switch, then search left stop switch from both sides<br>4 search left stop switch from both sides<br>5 search home switch in negative direction, reverse the direction when left stop switch reached<br>6 search home switch in positive direction, reverse the direction when right stop switch reached<br>7 search home switch in positive direction, ignore end switches<br>8 search home switch in negative direction, ignore end switches<br>*Adding 128 to these values reverses the polarity of the home switch input.* |
| 194 | referencing search speed | For the reference search this value directly specifies the search speed. |

| | | |
|---|---|---|
| 195 | referencing switch speed | Similar to parameter no. 194, the speed for the switching point calibration can be selected. |
| 196 | distance end switches | This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3). |

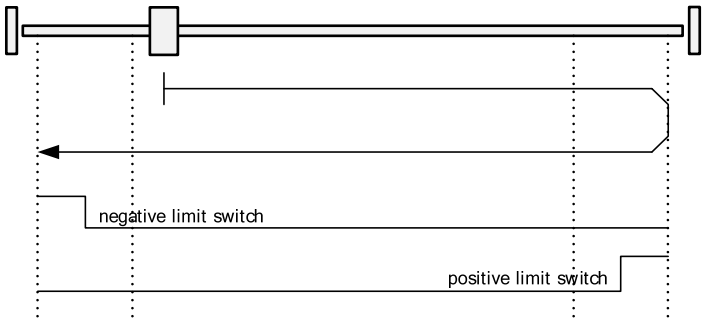
### *6.1.1* Reference Search Modes (Axis Parameter 193)

*SAP 193, 0, 1*



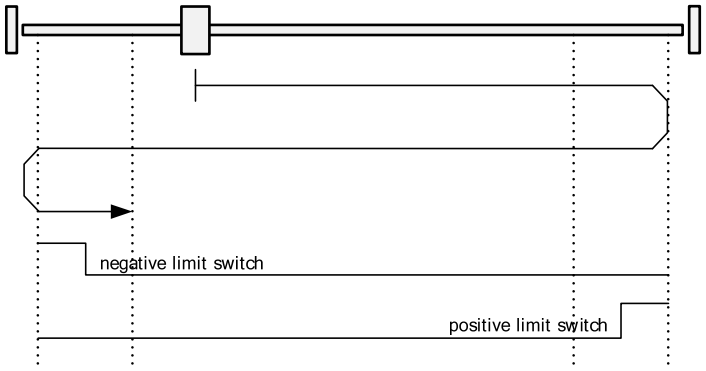

Search left stop switch only.

*SAP 193, 0, 2*



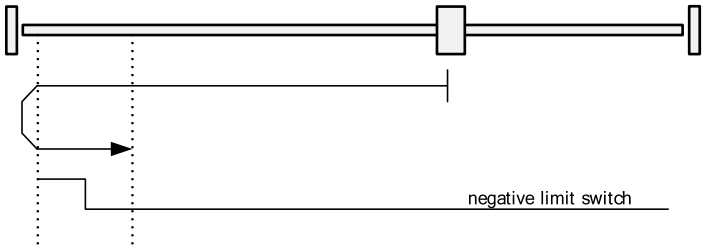

Search right stop switch, then search left stop switch.

*SAP 193, 0, 3*




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

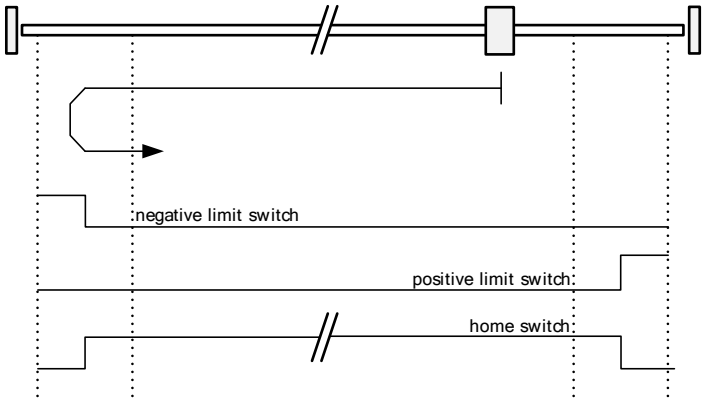
*SAP 193, 0, 4*




Search left stop switch from both sides.

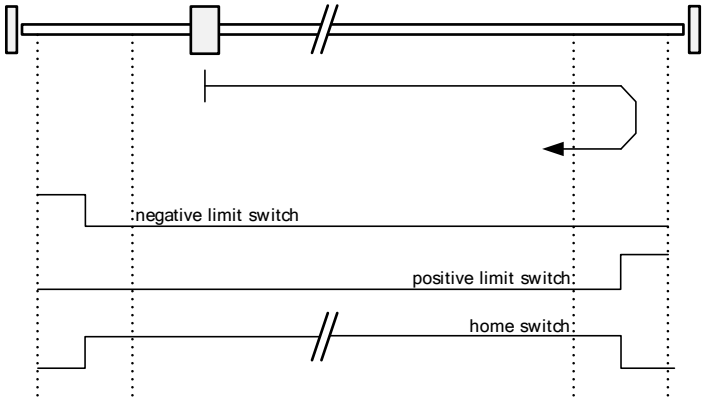
**Figure 6.1: Reference search modes 1-4**

*SAP 193, 0, 5*



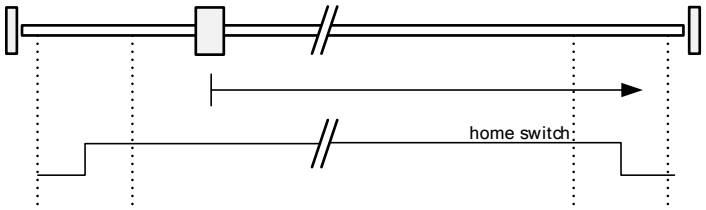

Search home switch in negative direction, reverse the direction when left stop switch reached.

*SAP 193, 0, 6*



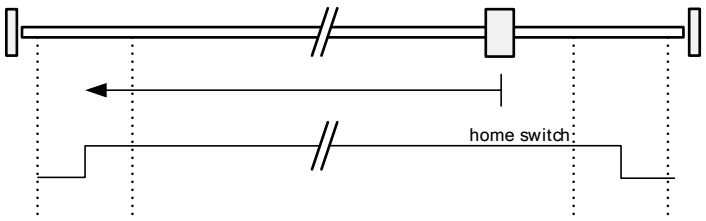

Search home switch in positive direction, reverse the direction when right stop switch reached.

*SAP 193, 0, 7*




Search home switch in positive direction, ignore end switches.

*SAP 193, 0, 8*




Search home switch in negative direction, ignore end switches.

**Figure 6.2: Reference search modes 5-8**

## 6.2 Encoder

The MCST3601 provides an interface for single ended incremental encoders with TTL (5V) outputs. For the operation with encoder please consider the following:

- The encoder counter can be read by software and can be used to monitor the exact position of the motor. This also makes closed loop operation possible.
- The Encoder channel ENC_I is for zeroing the encoder counter. It can be selected as high or as low active, and it is automatically checked in parallel to the Encoder channel A and B inputs for referencing exactly.
- To read out or to change the position value of the encoder, axis parameter 209 is used. To read out the position of your encoder 0 use GAP 209, 0. The position values can also be changed using command SAP 209, 0, <n>, with n = −2.147.483.648… +2.147.483.647
- To change the encoder settings, axis parameter 210 is used. For changing the prescaler of encoder 0 use SAP 210, 0, <p>.
- Automatic motor stop on deviation error is also usable. This can be set using axis parameter 212 (maximum deviation). This function is turned off when the maximum deviation is set to 0.

**PARAMETERS NEEDED FOR USING THE ENCODER**

| Number | Axis Parameter | Description | |
|---|---|---|---|
| 209 | encoder position | The value of an encoder register can be read out or written. | [encoder steps] |
| 210 | Encoder prescaler | Prescaler for the encoder. | See paragraph 6.2.1 |
| 212 | maximum encoder deviation | When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero. | 0… 65535<br><br>[encoder steps] |

### 6.2.1 Changing the Prescaler Value of an Encoder

*The table below shows a prescaler subset which can be selected. Other values between those in the table can be used. The bits 2… 4 must not be used for the prescaler because they are needed to select special encoder functions.*

**TO SELECT A PRESCALER, THE FOLLOWING VALUES CAN BE USED FOR <P>:**

| Value for <p> | Resulting prescaler | SAP command for motor 0 SAP 210, M0, <p> | Resulting steps per rotation for a 400 line (1600 quadrate count) encoder |
|---|---|---|---|
| 64 | 0.125 | SAP 210, 0, 64 | 200 |
| 128 | 0.25 | SAP 210, 0, 128 | 400 |
| 256 | 0.5 | SAP 210, 0, 256 | 800 |
| **512** | **1** | **SAP 210, 0, 512** | **1600** |
| 1024 | 2 | SAP 210, 0, 1024 | 3200 |
| 2048 | 4 | SAP 210, 0, 2048 | 6400 |
| 4096 | 8 | SAP 210, 0, 4096 | 12800 |
| 8192 | 16 | SAP 210, 0, 8192 | 25600 |
| 16384 | 32 | SAP 210, 0, 16384 | 51200 |
| 32768 | 64 | SAP 210, 0, 32768 | 102400 |

**Formula for resulting steps per rotation:**

StepsPerRotation = LinesOfEncoder * 4 * Prescaler

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

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

There are some special functions that can also be configured using these values. To select these functions just add the following values to <p>:

| Bit | Adder for <p> | Command: SAP 210, <motor number>, <p> |
|---|---|---|
| 2 | 4 | If set the encoder will be zeroed with next index channel event. |
| 3 | 8 | If set in combination with bit 2: Encoder will be zeroed with each index channel event. |
| 4 | 16 | Channel Z polarity for encoder clearing: 0 - low<br>1 - high |

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

## 6.3 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 MCST3601 axis parameters | | Range (TMC429 and MCST3601) |
|---|---|---|---|
| ***Velocity*** | Axis parameter 2 | target (next) speed | 0… 2047 |
| | 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 *microsteps per fullstep* $= 2^{\mu srs}$ | Axis parameter 140 offers the following settings:<br>0 full step<br>1 half step<br>2 4 microsteps<br>3 8 microsteps<br>4 16 microsteps<br>5 32 microsteps<br>6 64 microsteps<br>7 128 microsteps<br>8 256 microsteps | | 0… 8 |
| ***ramp_div*** / ramp divisor | Axis parameter 153: divider for the acceleration. The higher the value is, the less is the maximum acceleration<br>Default: 0 | | 0… 13 |
| ***pulse_div*** / pulse divisor | Axis parameter 153: divider for the velocity.<br>Increasing the value by one divides the acceleration into halves, decreasing the value by one doubles the acceleration.<br>Default: 0 | | 0… 13 |
| $f_{CLK}$ / clock frequency | --- | | 16MHz |

### 6.3.1 Microstep Frequency

The microstep frequency of the stepper motor is calculated with

$$\mu sf[Hz] = \frac{f_{CLK}[Hz] \cdot velocity}{2^{pulse\_div} \cdot 2048 \cdot 32} \qquad \textit{µsf: microstep-frequency}$$

### 6.3.2 Fullstep Frequency

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

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

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:

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

**Example:**

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

$$\mu sf = \frac{16\,MHz \cdot 1000}{2^1 \cdot 2048 \cdot 32} = \underline{\underline{122070.31\,Hz}}$$

$$fsf[Hz] = \frac{122070.31}{2^6} = \underline{\underline{1907.34\,Hz}}$$

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

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

#### 6.3.2.1 Calculation of Number of Rotations:

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

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

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

# 7 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.***

## 7.1 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. 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 | Description |
|---|---|---|
| R | GGP | Parameter readable |
| W | SGP, AGP | Parameter writable |
| E | SGP, AGP | Parameter stored permanently in EEPROM |

| Number | Global parameter | Description | | | Range | Access |
|---|---|---|---|---|---|---|
| 64 | EEPROM magic | Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration. | | | 0… 255 | RWE |
| 65 | Serial interface baud rate | 0 | 9600 baud | *Default* | 0… 11 | RWE |
| | | 1 | 14400 baud | | | |
| | | 2 | 19200 baud | | | |
| | | 3 | 28800 baud | | | |
| | | 4 | 38400 baud | | | |
| | | 5 | 57600 baud | | | |
| | | 6 | 76800 baud | *Not supported by Windows!* | | |
| | | 7 | 115200 baud | | | |
| | | 8 | 230400 baud | | | |
| | | 9 | 250000 baud | *Not supported by Windows!* | | |
| | | 10 | 500000 baud | *Not supported by Windows!* | | |
| | | 11 | 1000000 baud | *Not supported by Windows!* | | |
| 66 | serial address | The module (target) address for RS485. | | | 0… 255 | RWE |
| 68 | serial heartbeat | Serial heartbeat for USB interface. If this time limit is up and no further command is noticed the motor will be stopped.<br>0 – parameter is disabled | | | [ms] | RWE |
| 69 | Reserved | | | | | |
| 70 | Reserved | | | | | |
| 71 | Reserved | | | | | |
| 73 | configuration EEPROM lock flag | Write: 1234 to lock the EEPROM, 4321 to unlock it.<br>Read: 1=EEPROM locked, 0=EEPROM unlocked. | | | 0/1 | RWE |
| 76 | serial host address | Host address used in the reply telegrams sent back via USB. | | | 0… 255 | RWE |
| 77 | auto start mode | 0: Do not start TMCL™ application after power up (default).<br>1: Start TMCL™ application automatically after power up. | | | 0/1 | RWE |
| 81 | TMCL™ code protection | Protect a TMCL™ program against disassembling or overwriting.<br>0 – no protection<br>1 – protection against disassembling<br>2 – protection against overwriting<br>3 – protection against disassembling and overwriting<br>***If you switch off the protection against disassembling, the program will be erased first!***<br>***Changing this value from 1 or 3 to 0 or 2, the TMCL™ program will be wiped off.*** | | | 0,1,2,3 | RWE |
| 82 | Reserved | | | | | |
| 83 | Reserved | | | | | |
| 84 | coordinate storage | 0 – coordinates are stored in the RAM only (but can be copied explicitly between RAM and EEPROM)<br>1 – coordinates are always stored in the EEPROM only | | | 0 or 1 | RWE |

| Number | Global parameter | Description | Range | Access |
|---|---|---|---|---|
| 85 | do not restore user variables | 0 – user variables are restored (*default*)<br>1 – user variables are not restored | 0/1 | RWE |
| 86 | step pulse length | Length of step pulse (for Step/Dir interface): $(1 + \mathrm{x})\ \mu s$<br>Default setting: 0 (1µs)<br>*This setting is valid for all three motor axes.* | 0… 15 | RWE |
| 128 | TMCL™ application status | 0 –stop<br>1 – run<br>2 – step<br>3 – reset | 0… 3 | R |
| 129 | download mode | 0 – normal mode<br>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… 2147483647 | R |

## 7.2 Bank 1

The global parameter bank 1 is normally not available. It may be used for customer specific extensions of the firmware. Together with user definable commands (see section 6.3) these variables form the interface between extensions of the firmware (written in C) and TMCL™ applications.

## 7.3 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 | 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… 55 | general purpose variable #0… 55 | for use in TMCL™ applications | $-2^{31}$…$+2^{31}$ | RWE |
| 56… 255 | general purpose variables #56… #255 | for use in TMCL™ applications | $-2^{31}$…$+2^{31}$ | RW |

## 7.4 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 | Left stop switch 0 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 28 | Right stop switch 0 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 29 | Left stop switch 1 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 30 | Right stop switch 1 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 31 | Left stop switch 2 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 32 | Right stop switch 2 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 39 | Input 0 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 40 | Input 1 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 41 | Input 2 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |
| 42 | Input 3 edge type | 0=off, 1=low-high, 2=high-low, 3=both | 0… 3 | RWE |

# 8 TMCL™ Programming Techniques and Structure

## 8.1 Initialization

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

## 8.2 Main Loop

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

*There are exceptions, 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
```

## 8.3 Using Symbolic Constants

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

Example:

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

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

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

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

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

## 8.4 Using Variables

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

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

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

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

Example:

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

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

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

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

## 8.5 Using Subroutines

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

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

## 8.6 Mixing Direct Mode and Standalone Mode

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

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

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

Example:

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

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

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

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

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

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

# 9 Life Support Policy

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

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



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.

# 10 Revision History

## 10.1 Firmware Revision

| Version | Date | Author<br>OK - Olav Kahlbaum | Description |
|---|---|---|---|
| 1.33 | 2014-APR-29 | OK | First version for new hardware version V1.1 |
| | | | |
| | | | |

## 10.2 Document Revision

| Version | Date | Author | Description |
|---|---|---|---|
| 0.90 | 2014-MAY-27 | GE | Initial version based on TMCM-1110 stepRocker TMCL™ firmware manual |
| 0.91 | 2014-JUL-09 | DWI | Suppression of CAN information |

# 11 References

| | |
|---|---|
| [MCST3601] | MCST3601 Hardware Manual |
| [TMC262] | TMC262 Datasheet |
| [TMC429] | TMC429 Datasheet |
| [TMCL-IDE] | TMCL-IDE User Manual |

Please refer to www.trinamic.com.

7000.55002 English, 1st edition, 10.2014


**FAULHABER**
**PRECISTEP SA**

Rue des Gentianes 53
2300 La Chaux-de-Fonds
Switzerland
Tel. +41 32 910 6050
Fax +41 32 910 6059
info@precistep.com
www.faulhaber.com