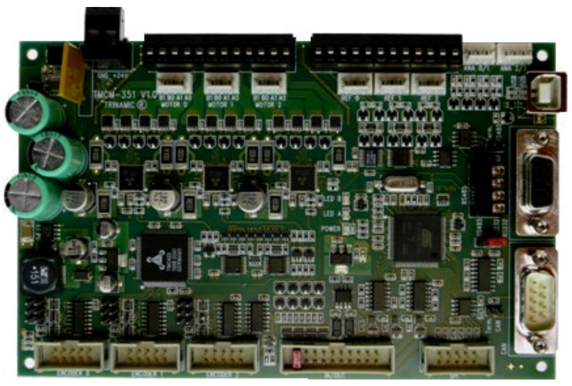
Firmware Version V4.45

TMCL™ FIRMWARE MANUAL



TMCM-351

**3-Axis Stepper
Controller / Driver
2.8 A / 24 V
SPI, RS232, RS485, CAN, and USB
Encoder Interface**

stallGuard™

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

www.trinamic.com

Table of Contents

1 Features

The TMCM-351 is a powerful three axes bipolar stepper motor controller/driver board with optional encoder interface for all three axes and a large number of general purpose digital and analogue input/outputs. Several different serial communication interfaces are available.

Main Characteristics

Electrical data

- Supply voltage: +24V DC nominal (28.5V DC max.)
- Motor current: up to 2.8A RMS per axis (programmable)

Stepper motor data

- two phase bipolar stepper motors with up to 2.8A RMS coil current
- optional incremental encoder interface (a/b/n), accepts differential or single ended input signals

Interfaces

- 2 reference switch inputs per motor axis (6 altogether, internal pull-up resistors, +24V compatible)
- 8 general purpose inputs (+24V compatible)
- 8 general purpose outputs incl. two power outputs (all open-collector)
- 1 shutdown input (enable/disable driver stage in hardware)
- 4 dedicated analogue inputs (programmable 3.3V/10V input range)
- SPI™[1] connector with three chip select signals for I/O extension
- RS232, RS485, CAN and USB serial communication interfaces

Features

- High-efficient operation, low power-dissipation (TMC249 stepper driver with external MOSFETs)
- Dynamic current control
- Integrated Protection
- On the fly alteration of motor parameters (e.g. position, velocity, acceleration)
- Motion profile calculation in real-time (TMC429 motion controller)
- Each axis individually and independently programmable
- Supports up to 64 microsteps per fullstep
- Integrated stallGuard™ for motor stall detection (e.g. elimination of end switches)
- Closed-loop operation with TMCL possible (when using the optional incremental encoder interface)

Software

- TMCL™ remote (direct mode) or stand-alone operation (memory for 2048 TMCL commands)
- Fully supported by TMCL-IDE (PC based integrated development environment)
- Optional CANopen firmware

[1] SPI™ is a trademark of Motorola

2 Putting the TMCM-351 into Operation

Here you can find basic information for putting your module into operation. The text contains two simple examples (with and without encoder) for a TMCL program and a short description of operating the module in direct mode.

THE THINGS YOU NEED

- TMCM-351
- Interface (RS232, RS485, USB or CAN) suitable to your TMCM-351 version with cables
- Nominal supply voltage +24V DC (+7…+28.5V DC) for your module
- Up to three stepper motors which fit to your module, for example QSH-5718 or QSH-6018.
- TMCL-IDE program and PC
- Encoder optional

PRECAUTIONS

- Do not connect or disconnect the motor while powered!
- Do not mix up connections or short-circuit pins.
- Avoid bounding I/O wires with motor power wires as this may cause noise picked up from the motor supply.
- Do not exceed the maximum power supply of 28.5V DC.
- Start with power supply OFF!

2.1 Starting up

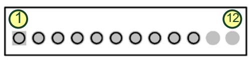
1. **Connect the motors**

For the three motors there are two connector options:

- one detachable screw connector (for prototyping, smaller series)
- three separate crimp connectors (for higher volume series)

For this example we choose the screw connector. You will find further information about the crimp connectors in the hardware manual.

Please connect the motors with the screw connector as follows:

	Pin	Label	Description
	1	Motor_0_B-	Motor 0, coil B
	2	Motor_0_B+	Motor 0, coil B
	3	Motor_0_A-	Motor 0, coil A
	4	Motor_0_A+	Motor 0, coil A
	5	Motor_1_B-	Motor 1, coil B
	6	Motor_1_B+	Motor 1, coil B
	7	Motor_1_A-	Motor 1, coil A
	8	Motor_1_A+	Motor 1, coil A
	9	Motor_2_B-	Motor 2, coil B
	10	Motor_2_B+	Motor 2, coil B
	11	Motor_2_A-	Motor 2, coil A
	12	Motor_2_A+	Motor 2, coil A

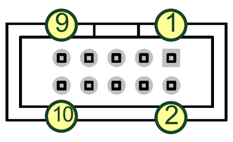
2. Connect the encoder

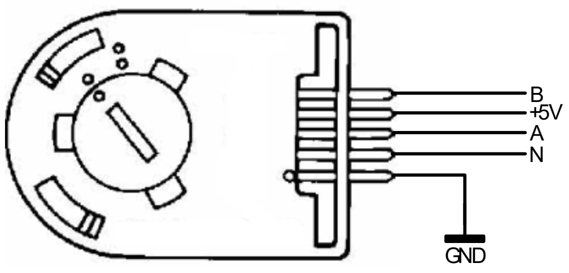
For boards with assembled encoder option three connectors (one encoder interface connector per axis) will be available. A standard 2.54mm pitch two row header is used for connecting an encoder. Differential and single ended incremental encoders with/without zero/index channel are supported.

Please connect the encoders as follows:

- Single ended encoder: GND to pin 1 and/or 2
  +5V to pin 7 and/or 8
  A to pin 5
  N to pin 3
  B to pin 9

- Differential encoder: GND to pin 1 and/or 2
  +5V to pin 7 and/or 8
  A+ to pin 5, A- to pin 6
  N+ to pin 3, N- to pin 4
  B+ to pin 9, B- to pin 10

	Pin	Label	Pin	Label
	1	GND	2	GND
	3	Encoder_0/1/2_N+	4	Encoder_0/1/2_N-
	5	Encoder_0/1/2_A+	6	Encoder_0/1/2_A-
	7	+5V output	8	+5V output
	9	Encoder_0/1/2_B+	10	Encoder_0/1/2_B-

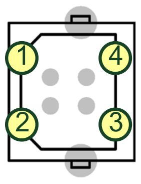


Example for single ended encoder

3. Connect the interface

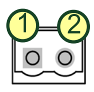
In this case we choose the USB interface for serial communication. A standard USB type B connector is used for this purpose. USB is one out of four different interfaces available for communication with the board. You can refer to the hardware manual for further information about the pinning of the other interfaces.

Please connect the USB interface with the enclosed cable as follows:

	Pin	Label	Description
	1	+5V	Board is self-powered – just use to detect availability of attached host system (e.g. PC)
	2	USB-	Differential USB bus
	3	USB+	Differential USB bus
	4	GND	System / module ground

4. **Connect the power supply:**
   Attention: Do not exceed the maximum power supply of 28.5 V DC!

   Please connect the power supply as follows:

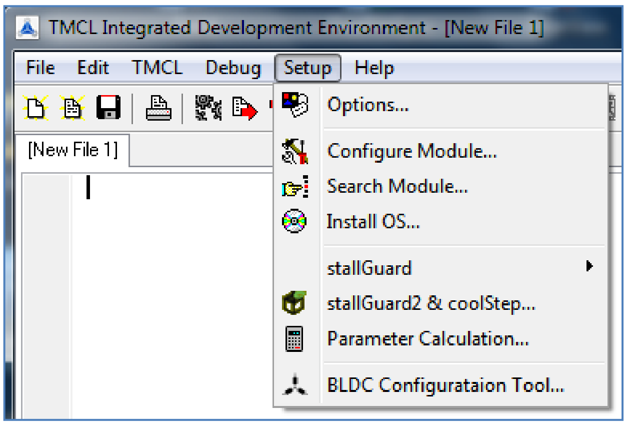
	Pin	Label	Description
	1	GND	Module ground (power supply and signal ground)
	2	VDD	Power supply input, nom. +24V DC (+7 .. +28.5V DC)

5. **Switch on the power supply**
   The green LED for power should glow now. This indicates that the on-board +5V supply is available.

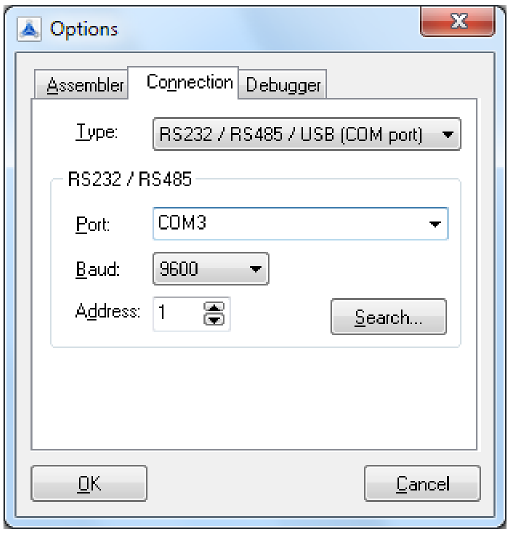
   If this does not occur, switch power OFF and check your connections as well as the power supply.

6. **Start the TMCL-IDE software development environment**
   The TMCL-IDE is on hand on the TechLibCD and on www.trinamic.com.

   Installing the TMCL-IDE:
   - Make sure the COM port you intend to use is not blocked by another program.
   - Open TMCL-IDE by clicking ***TMCL.exe***.
   - Choose ***Setup*** and ***Options*** and thereafter the ***Connection tab***.



   - Choose ***COM port*** and ***type*** with the parameters shown below (baud rate 9600). Click *OK*.



Type: RS232 / RS485 / USB (COM port)

RS232 / RS485

Port: COM3

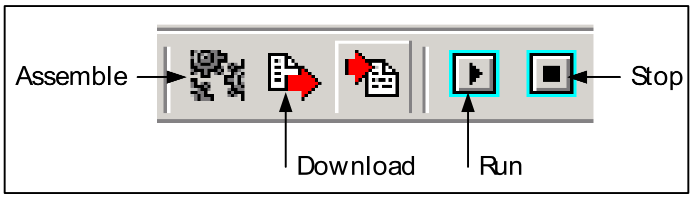
Baud: 9600

Address: 1

2.2 Testing with a Simple TMCL Program

2.2.1 Testing without Encoder

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

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

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



1. Click on Icon ***Assemble*** to convert the TMCL into machine code.
2. Then download the program to the TMCM-351 module via the icon ***Download***.
3. Press icon ***Run***. The desired program will be executed.
4. Click ***Stop*** button to stop the program.

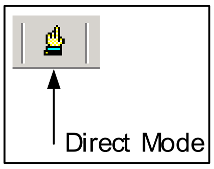
2.2.2 Testing with Encoder

The motor rotates between two positions and stops if itis obstructed. The position is then corrected so that the motor always reaches the correct target positions.
The encoder multiplier and the microstep resolution must be set so that the resolution of the encoder position and the motor position match with each other.
The values here are for an encoder with 2000 steps per rotation and a motor with 200 full steps per rotation. The setting of 64 microsteps then results in a motor resolution of 12800 microsteps per rotation and the encoder multiplier of 68672 (=>6.4) also results in an encoder resolution of 12800 steps per rotation.

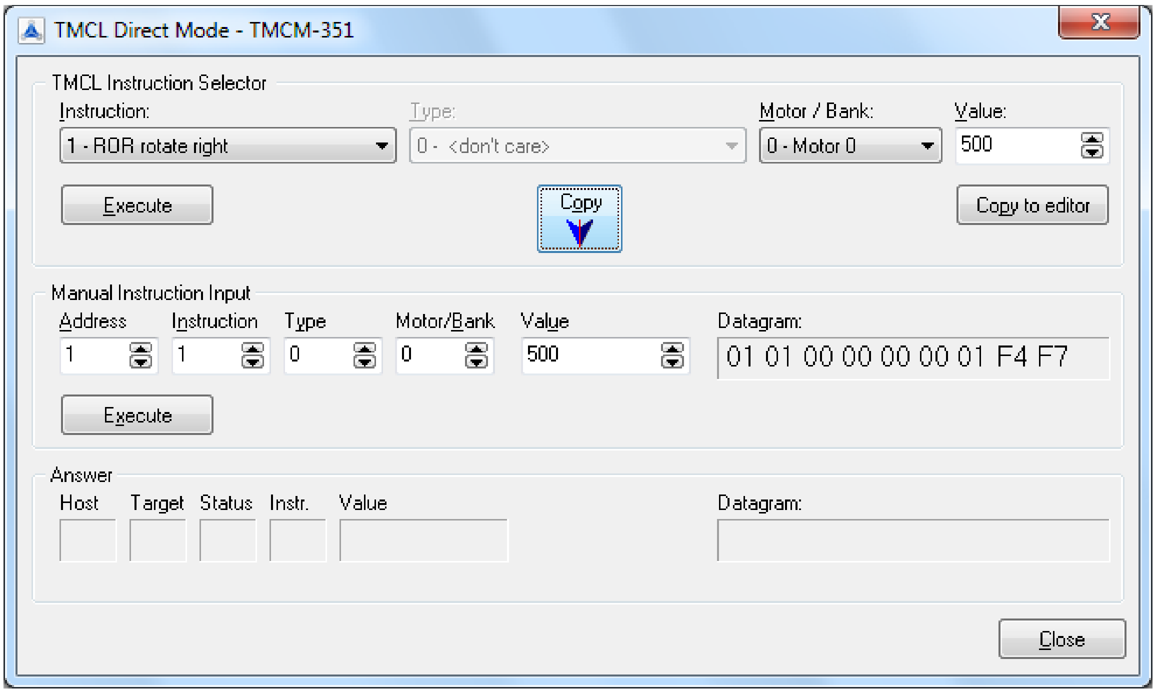
```
// Encoder demo program for all modules with encoder interface

     MST 0                         //Ensure that the motor is not moving
        CSUB WaitUntilStanding
        SAP 210, 0, 68672          //Encoder multiplier (6.4)
        SAP 209, 0, 0              //reset encoder position
        SAP 0, 0, 0                //reset the motor
        SAP 1, 0, 0                //position registers
        SAP 140, 0, 6              //Microstep resolution (64)
        SAP 5, 0, 50               //Acceleration
        SAP 212, 0, 250            //use  automatic  deviation  check  to  stop
                                   motor
                                   //when it is obstructed

Loop:   MVP ABS, 0, 128000         //Rotate 10 revolutions
        CSUB WaitUntilRunning      //Wait until the motor is running
        CSUB WaitUntilStanding     //Wait until the motor has stopped
        GAP 8, 0                   //Check if the end position has been reached
        JC NZ, PosReached1         //Jump if yes
        GAP 209, 0                 //if not: copy encoder position to...
        AAP 0, 0                   //...target position and...
        AAP 1, 0                   //...actual position
        WAIT TICKS, 0, 50          //wait 0.5sec
        JA Loop                    //continue the move

PosReached1:                       //End position has been reached
        WAIT TICKS, 0, 200         //Wait 2sec
Rst2:   MVP ABS, 0, 0              //Move 10 revolutions back
        CSUB WaitUntilRunning      //Wait until the motor is running
        CSUB WaitUntilStanding     //Wait until the motor has stopped
        GAP 8, 0                   //Check if the end position has been reached
        JC NZ, PosReached2         //Jump if yes
        GAP 209, 0                 //if not: copy encoder position to...
        AAP 0, 0                   //...target position and...
        AAP 1, 0                   //...actual position
        WAIT TICKS, 0, 50          //wait 0.5sec
        JA Rst2                    //continue the move

PosReached2:                       //The other end position has been reached
        WAIT TICKS, 0, 200         //Wait 2sec
        JA Loop                    //Start again

WaitUntilRunning:                  //Subroutine that waits until the motor is
                                   running
       GAP 3, 0
       COMP 0
       JC EQ, WaitUntilRunning
       RSUB

WaitUntilStanding:                 //Subroutine that waits until the motor has
                                   stooped
       GAP 3, 0
       COMP 0
       JC NE, WaitUntilStanding
       RSUB
```

2.3 Operating the Module in Direct Mode

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



2. If the communication is established the TMCM-351 is automatically detected. ***If the module is not detected, please check all points above (cables, interface, power supply, COM port, baud rate).***
3. Issue a command by choosing ***instruction***, ***type*** (if necessary), ***motor***, and ***value*** and click *Execute* to send it to the module.


TMCL Direct Mode - TMCM-351

TMCL Instruction Selector

Instruction: 1 - ROR rotate right
Type: 0 - <don't care>
Motor / Bank: 0 - Motor 0
Value: 500

Manual Instruction Input

Address	Instruction	Type	Motor/Bank	Value	Datagram:
1	1	0	0	500	01 01 00 00 00 00 01 F4 F7

Answer

Host	Target	Status	Instr.	Value	Datagram:

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.

You will find a description of all TMCL commands in the following chapters.

3 Overview

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-351 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 – normally – untouched throughout the whole lifetime, the firmware can be updated by the user. New versions can be downloaded free of charge from the TRINAMIC website (http://www.trinamic.com).
The firmware shipped with this module is related to the standard TMCL firmware [TMCL] shipped with most of TRINAMIC modules with regard to protocol and commands. Corresponding, the module is based on the TMC428/429 stepper motor controller and the TMC249 power driver and supports the standard TMCL with a special range of values. Further you can order the module with encoder option, realized with the TMC423.

4 TMCL and TMCL-IDE

The TMCM-351module supports TMCL direct mode (binary commands or ASCII interface) and standalone TMCL program execution. You can store up to 2048 TMCL instructions on it. In direct mode and most cases the TMCL communication over RS485, RS232, USB and CAN follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the module. The TMCL interpreter on it will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over RS485/RS232/USB/CAN to the bus master. Only then should the master transfer the next command. Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.

The Trinamic Motion Control Language (TMCL) provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM-351 to form programs that run stand-alone 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

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes.
When a command is to be sent via RS232, RS485 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In this case it consists of nine bytes.
This is different when communicating takes place via the CAN bus. Address and checksum are included in the CAN standard and do not have to be supplied by the user.

The binary command format for RS232, RS485, and USB is as follows:

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

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

- When using the CAN bus, just leave out the first byte (module address) and the last byte (checksum).

4.1.1 Checksum Calculation

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

- in C:

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

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

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

- in Delphi:

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

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

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

4.2 Reply Format

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

The reply format for RS485, RS232, and USB is as follows:

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

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

4.2.1 Status Codes

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

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

4.3 Standalone Applications

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

4.4 TMCL Command Overview

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

4.4.1 TMCL Commands

The following TMCL commands are currently supported:

Command	Number	Parameter	Description
ROR	1	<motor number>, <velocity>	Rotate right with specified velocity
ROL	2	<motor number>, <velocity>	Rotate left with specified velocity
MST	3	<motor number>	Stop motor movement
MVP	4	ABS|REL|COORD, <motor number>, <position|offset>	Move to position (absolute or relative)
SAP	5	<parameter>, <motor number>, <value>	Set axis parameter (motion control specific settings)
GAP	6	<parameter>, <motor number>	Get axis parameter (read out motion control specific settings)
STAP	7	<parameter>, <motor number>	Store axis parameter permanently (non volatile)
RSAP	8	<parameter>, <motor number>	Restore axis parameter
SGP	9	<parameter>, <bank number>, value	Set global parameter (module specific settings e.g. communication settings or TMCL user variables)
GGP	10	<parameter>, <bank number>	Get global parameter (read out module specific settings e.g. communication settings or TMCL user variables)
STGP	11	<parameter>, <bank number>	Store global parameter (TMCL user variables only)
RSGP	12	<parameter>, <bank number>	Restore global parameter (TMCL user variable only)
RFS	13	START|STOP|STATUS, <motor number>	Reference search
SIO	14	<port number>, <bank number>, <value>	Set digital output to specified value
GIO	15	<port number>, <bank number>	Get value of analogue/digital input
CALC	19	<operation>, <value>	Process accumulator & value
COMP	20	<value>	Compare accumulator <-> value
JC	21	<condition>, <jump address>	Jump conditional
JA	22	<jump address>	Jump absolute
CSUB	23	<subroutine address>	Call subroutine
RSUB	24		Return from subroutine
EI	25	<interrupt number>	Enable interrupt
DI	26	<interrupt number>	Disable interrupt
WAIT	27	<condition>, <motor number>, <ticks>	Wait with further program execution
STOP	28		Stop program execution
SAC	29	<bus number>, <number of bites>, <send data>	SPI bus access
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 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
SAC	29	Access to external SPI device

4.4.2.4 SPI Bus Access Command

Mnemonic	Command number	Meaning
SAC	29	SPI bus access

4.4.2.5 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
CLE	36	Clear error flags
CSUB	23	Call subroutine
RSUB	24	Return from subroutine
WAIT	27	Wait for a specified event
STOP	28	End of a TMCL program

4.4.2.6 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 stand-alone 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 (stand-alone mode), a host can still send commands like GAP, GGP or GIO 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.

4.4.2.7 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

4.4.2.7.1 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.

4.4.2.7.2 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.

4.4.2.7.3 Interrupt Vectors:

The following table shows all interrupt vectors for the three motors 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 0
16	stallGuard 1
17	stallGuard 2
21	Deviation 0
22	Deviation 1
23	Deviation 2
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
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

4.4.2.7.4 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 for further information about that.

4.4.2.7.5 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.

4.4.2.8 ASCII Commands

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

4.4.2.9 TMCL Control Commands

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

4.5 The ASCII Interface

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

PROCEED AS FOLLOWS

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

4.5.1 Command Line Format

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

EXAMPLES FOR VALID COMMAND LINES

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

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

4.5.2 Format of a Reply

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

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

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

4.5.3 Commands Used in ASCII Mode

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

SPECIAL COMMANDS WHICH ARE ONLY AVAILABLE IN ASCII MODE

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

4.5.4 Configuring the ASCII Interface

The module can be configured so that it starts up either in binary mode or in ASCII mode. ***Global parameter 67 is used for this purpose*** (please see also chapter 6).

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

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

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

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

4.6 Commands

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

4.6.1 ROR (rotate right)

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

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

The module is based on the TMC429 stepper motor controller and the TMC249 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR <motor>, <velocity>

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Rotate right motor 2, velocity = 350
Mnemonic: ROR 2, 350

Binary:

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

4.6.2 ROL (rotate left)

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

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

The module is based on the TMC429 stepper motor controller and the TMC249 power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL <motor>, <velocity>

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Rotate left motor 0, velocity = 1200
Mnemonic: ROL 0, 1200

Binary:

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

4.6.3 MST (motor stop)

The motor will be instructed to stop.

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

Related commands: ROL, ROR, SAP, GAP

Mnemonic: MST <motor>

Binary representation:

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

4.6.4 MVP (move to position)

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

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

THREE 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.
- Moving the motor to a (previously stored) coordinate (refer to SCO for details).

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

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

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

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
4	0 ABS – absolute	<motor> 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

4.6.5 SAP (set axis parameter)

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

Please use command STAP (store axis parameter) in order to store any setting permanently.

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

Related commands: GAP, STAP, RSAP, AAP

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
5	<parameter number>	<motor> 0… 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 0 to 1.4 A

Because of the current unit $I_{RMS} = < value > \times \frac{2.8A}{255}$ the 200mA setting has the <value> 128 (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, 128

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	$12

4.6.6 GAP (get axis parameter)

Most parameters of the TMCM-351 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>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
6	<parameter number>	<motor> 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 1
Mnemonic: GAP 6, 1

Binary:

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

Reply:

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

4.6.7 STAP (store axis parameter)

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

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>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
7	<parameter number>	<motor> 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.
The STAP command will not have any effect when the configuration EEPROM is locked (refer to 6). In direct mode, the error code 5 will be returned in this case.

Example:

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

Binary:

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

4.6.8 RSAP (restore axis parameter)

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

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

Relate commands: SAP, STAP, GAP, and AAP

Mnemonic: RSAP <parameter number>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
8	<parameter number>	<motor> 0… 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 3
Mnemonic: RSAP 6, 0

Binary:

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

4.6.9 SGP (set global parameter)

With this command most of the module specific parameters not directly related to motion control can be specified and the TMCL user variables can be changed. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in *banks* to allow a larger total number for future products. Currently, only bank 0 and 1 are used for global parameters, and bank 2 is used for user variables. 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.
For a table with parameters and bank numbers which can be used together with this command please refer to chapter 6.

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

Related commands: GGP, STGP, RSGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

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

Binary:

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

4.6.10 GGP (get global parameter)

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

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

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

Related commands: SGP, STGP, RSGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

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

Binary:

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

Reply:

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

⇨ **Status = no error, value = 1**

4.6.11 STGP (store global parameter)

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

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

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

Related commands: SGP, GGP, RSGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

Store the user variable #42
Mnemonic: STGP 42, 2

Binary:

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

4.6.12 RSGP (restore global parameter)

With this command the contents of a TMCL user variable can be restored from the EEPROM. For all configuration-related axis parameters, non-volatile memory locations are provided. By default, most parameters are automatically restored after power up. A single parameter that has been changed before can be reset by this instruction.

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

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

Relate commands: SGP, STGP, GGP, and AGP

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

Binary representation:

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

Reply structure in direct mode:

STATUS	VALUE
100 – OK	don’t care

Example:

Restore the user variable #42

Mnemonic: RSGP 42, 2

Binary:

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

4.6.13 RFS (reference search)

The TMCM-351 module 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. The reference search can be started, stopped, and the actual status of the reference search can be checked.

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

Related commands: WAIT

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

Binary representation:

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

4.6.14 SIO (set output)

This command can be used as follows:

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

Related commands: GIO, WAIT

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

Binary representation:

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

Reply structure:

STATUS	VALUE
100 – OK	don't care

Example:

Set OUT_7 to high (bank 2, output 7)
Mnemonic: SIO 7, 2, 1

Binary:

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

CONNECTORS

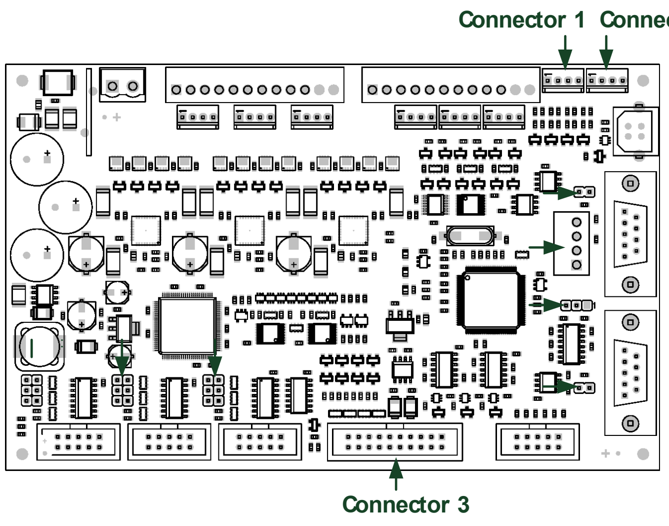
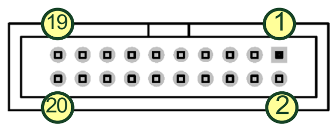


Figure 4.1 Connectors of TMCM-351

I/O PORTS USED FOR SIO AND COMMAND



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

ADDRESSING ALL SIX OUTPUT LINES WITH ONE SIO COMMAND:

Proceed as follows:

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

Example:

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

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

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

ADJUSTING THE ANALOGUE INPUT:

It is possible to adjust the analogue input with the following commands:

```
SIO 8, 0, 0 for 3,3V input range
SIO 8, 0, 1 for 10V input range
```

COMMAND FOR SWITCHING THE PULL-UP RESISTORS FOR ADDITIONAL DIGITAL INPUTS

SIO can be used to switch the pull-up resistors for all digital inputs on and off. Bank 0 is used for this purpose. Every pull-up resistor can be switched individually by setting the related bit using the bitmask.

Pin (connector 3)	Input	Bit	Command	Range
11	IN_0	0	SIO 0, 0,<bitmask>	0… 255
12	IN_1	1		
13	IN_2	2		
14	IN_3	3		
15	IN_4	4		
16	IN_5	5		
17	ADIN_0	6		
18	ADIN_1	7		

4.6.15 GIO (get input/output)

With this command the status of the two available general purpose inputs of the module can be read out. The function reads a digital or analogue input port. Digital lines will read 0 and 1, while the ADC channels deliver their 10 bit result in the range of 0… 1023.

GIO in Standalone Mode

In standalone mode the requested value is copied to the *accumulator* (accu) for further processing purposes such as conditioned jumps.

GIO in Direct Mode

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

Internal function: The specified line is read.

Related commands: SIO, WAIT

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	<status of the port>

Example:

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

Binary:

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

Reply:

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

⇨ value: 506

CONNECTORS

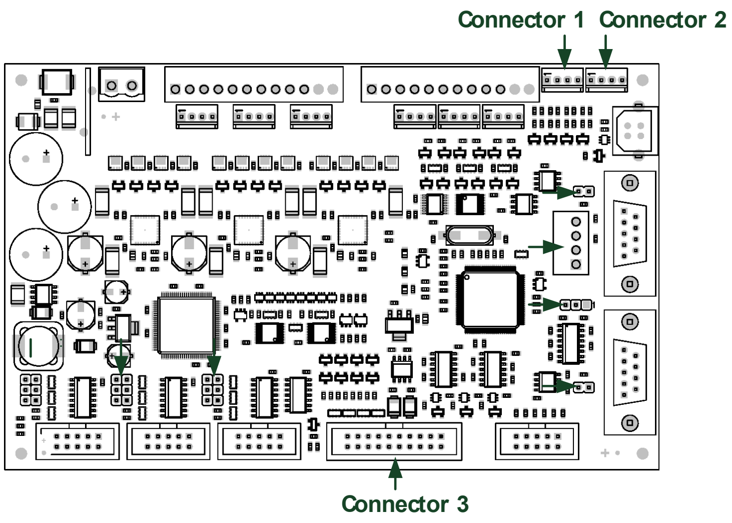
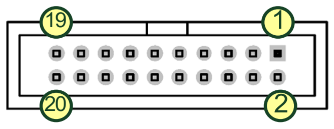


Figure 4.2 Connectors of TMCM-351

4.6.15.1 I/O Bank 0 – Digital Inputs:

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

	Pin	I/O port	Command	Range
	**11**	IN_0	GIO 0, 0	1/0
	12	IN_1	GIO 1, 0	1/0
	13	IN_2	GIO 2, 0	1/0
	14	IN_3	GIO 3, 0	1/0
	15	IN_4	GIO 4, 0	1/0
	16	IN_5	GIO 5, 0	1/0
	17	ADIN_6	GIO 6, 0	1/0
	18	ADIN_7	GIO 7, 0	1/0

READING ALL DIGITAL INPUTS WITH ONE GIO COMMAND:

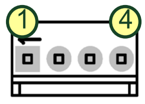
- Set the type parameter to 255 and the bank parameter to 0.
- In this case the status of all digital input lines will be read to the lower eight bits of the accumulator.

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

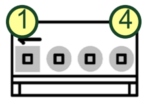
```
Loop: GIO 255, 0
      SIO 255, 2,-1
      JA Loop
```

4.6.15.2 I/O Bank 1 – Analogue Inputs:

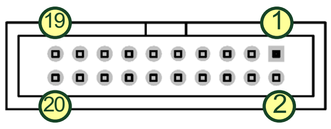
The ADIN lines can be read back as digital or analogue inputs at the same time. The digital states can be accessed in bank 0. The AIN lines can be used as analogue inputs only.



Connector 1



Connector 2

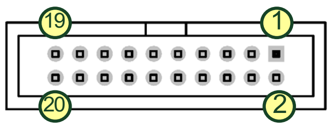


Connector 3

Pin	I/O port	Command	Comment	Range
1	AIN_0	GIO 0, 1	Connector 1	0…1023
3	AIN_1	GIO 1, 1	Connector 1	0…1023
1	AIN_2	GIO 2, 1	Connector 2	0…1023
3	AIN_3	GIO 3, 1	Connector 2	0…1023
-	-	GIO 4, 1	Power supply	0…1023
-	-	GIO 5, 1	Temperature	0…1023
7	ADIN_6	GIO 6, 1	Connector 3	0..1023
8	ADIN_7	GIO 7, 1	Connector 3	0..1023

4.6.15.3 I/O Bank 2 – the States of Digital Outputs

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



Connector 3

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

4.6.16 CALC (calculate)

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

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

Mnemonic: CALC <operation>, <value>

Binary representation:

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

Example:

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

Binary:

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

Reply:

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

Status = no error, value = -5000

4.6.17 COMP (compare)

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

This command is intended for use in standalone operation only.

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

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

Mnemonic: COMP <value>

Binary representation:

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

Example:

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

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

Binary format of the COMP 1000 command:

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

4.6.18 JC (jump conditional)

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

This function is for standalone operation only.

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

Related commands: JA, COMP, WAIT, CLE

Mnemonic: JC <condition>, <label>

Binary representation:

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

Example:

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

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

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

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

4.6.19 JA (jump always)

Jump to a fixed address in the TMCL program memory.

This command is intended for standalone operation only.

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

Related commands: JC, WAIT, CSUB

Mnemonic: JA <Label>

Binary representation:

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

Example: An infinite loop in TMCL

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

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

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

4.6.20 CSUB (call subroutine)

This function calls a subroutine in the TMCL program memory.

This command is intended for standalone operation only.

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

Related commands: RSUB, JA

Mnemonic: CSUB <Label>

Binary representation:

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

Example: Call a subroutine

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

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

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

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

4.6.21 RSUB (return from subroutine)

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

This command is intended for use in standalone mode only.

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

Related command: CSUB

Mnemonic: RSUB

Binary representation:

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

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

Example: Return form subroutine

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

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

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

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

4.6.22 WAIT (wait for an event to occur)

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

This command is intended for standalone operation only.

THERE ARE FIVE DIFFERENT WAIT CONDITIONS THAT CAN BE USED:

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

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

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

Related commands: JC, CLE

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

Binary representation:

INSTRUCTION NO.	TYPE <condition>	MOT/BANK	VALUE
27	0 TICKS - timer ticks*[1]	don't care	<no. of ticks*>
	1 POS - target position reached	<motor> 0… 2	<no. of ticks* for timeout>, 0 for no timeout
	2 REFSW – reference switch	<motor> 0… 2	<no. of ticks* for timeout>, 0 for no timeout
	3 LIMSW – limit switch	<motor> 0… 2	<no. of ticks* for timeout>, 0 for no timeout
	4 RFS – reference search completed	<motor> 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**
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	$01	$1b	$01	$00	$00	$00	$00	$00

4.6.23 STOP (stop TMCL program execution)

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

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

Internal function: TMCL instruction fetching is stopped.

Related commands: none
Mnemonic: STOP

Binary representation:

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

Example:

Mnemonic: STOP

Binary:

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

4.6.24 SAC (SPI bus access)

This command allows access to external SPI devices connected to the SPI bus of the module. Direct values and the contents of the accumulator register can be sent.
In standalone mode the received data is also stored in the accumulator.
The module has three chip select outputs (SPI_SEL0, SPI_SEL1, and SPI_SEL2). The *type* parameter (bus number) determines the chip select output that is to be used. The *motor/bank* parameter determines the number of bytes to be sent (1, 2, 3, or 4). The *value* parameter contains the data to be sent. When bit 7 of the bus number is set, this value is ignored and the contents of the accumulator are sent instead.

Please note that in the TMCL-IDE always all three values have to be specified (when sending the contents of the accumulator the value parameter is a dummy parameter).

Related commands: SIO, GIO

Mnemonic: SAC <bus number>, <number of bytes>, <send data>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
29	<bus number>	<number of bytes>	<send data>

Reply in direct mode:

STATUS	VALUE
100 – Success	<received data>

THE BUS NUMBERS ARE AS FOLLOWS:

Bus number	Chip select output
0	SPI_SEL0, output direct value
2	SPI_SEL1, output direct value
3	SPI_SEL2, output direct value
128	SPI_SEL0, output contents of accumulator
130	SPI_SLE1, output contents of accumulator
131	SPI_SEL2, output contents of accumulator

4.6.25 SCO (set coordinate)

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

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

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

Related commands: GCO, CCO, MVP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

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

Binary:

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

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

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

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

4.6.26 GCO (get coordinate)

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

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

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

Related commands: SCO, CCO, MVP

Mnemonic: GCO <coordinate number>, <motor>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
31	<coordinate number> 0… 20	<motor> 0… 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	Instructio n	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0
Value (hex)	$02	$01	$64	$0a	$00	$00	$00	$00

⇨ Value: 0

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

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

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

4.6.27 CCO (capture coordinate)

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

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

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

Related commands: SCO, GCO, MVP

Mnemonic: CCO <coordinate number>, <motor>

Binary representation:

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

4.6.28 ACO (accu to coordinate)

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

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

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

Related commands: GCO, CCO, MVP COORD, SCO

Mnemonic: ACO <coordinate number>, <motor>

Binary representation:

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

4.6.29 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	don't care	don't care
	1 SUB	subtract X register from accu		
	2 MUL	multiply accu by X register		
	3 DIV	divide accu by X-register		
	4 MOD	modulo divide accu by x-register		
	5 AND	logical and accu with X-register		
	6 OR	logical or accu with X-register		
	7 XOR	logical exor accu with X-register		
	8 NOT	logical invert X-register		
	9 LOAD	load accu to X-register		
	10 SWAP	swap accu with X-register		

Example:

Multiply accu by X-register
Mnemonic: CALCX MUL

Binary:

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

4.6.30 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.

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

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

Mnemonic: AAP <parameter number>, <motor>

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don’t care

Example:

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

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

Binary format of the AAP 0,0 command:

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

4.6.31 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.
For a table with parameters and bank numbers which can be used together with this command please refer to chapter 6.

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

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	don't care

Example:

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

Binary:

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

4.6.32 CLE (clear error flags)

This command clears the internal error flags.

The CLE command is intended for use in standalone mode only and must not be used in direct mode.

THE FOLLOWING ERROR FLAGS CAN BE CLEARED BY THIS COMMAND (DETERMINED BY THE <FLAG> PARAMETER):

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

Related commands: JC

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
36	0 – (ALL) all flags 1 – (ETO) timeout flag 2 – (EAL) alarm flag 3 – (EDV) deviation flag 4 – (EPO) position flag 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

4.6.33 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
16	stallGuard axis 1
17	stallGuard axis 2
21	Deviation 0
22	Deviation 1
23	Deviation 2
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
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

Example:

Define interrupt vector at target position 500
VECT 3, 500

Binary format of VECT:

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

4.6.34 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
16	stallGuard axis 1
17	stallGuard axis 2
21	Deviation 0
22	Deviation 1
23	Deviation 2
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
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

Examples:

Enable interrupts globally
EI, 255

Binary format of EI:

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

Enable interrupt when target position reached
EI, 3

Binary format of EI:

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

4.6.35 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
16	stallGuard axis 1
17	stallGuard axis 2
21	Deviation 0
22	Deviation 1
23	Deviation 2
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
43	Input change 4
44	Input change 5
45	Input change 6
46	Input change 7
255	Global interrupts

Examples:

Disable interrupts globally
DI, 255

Binary format of DI:

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

Disable interrupt when target position reached
DI, 3

Binary format of DI:

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

4.6.36 RETI (return from interrupt)

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

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

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

Related commands: EI, DI, VECT

Mnemonic: RETI

Binary representation:

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

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

Binary format of RETI:

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

4.6.37 Customer Specific TMCL Command Extension (user function)

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

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

Related commands: none

Mnemonic: UF0… UF7 <parameter number>

Binary representation:

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

Reply in direct mode:

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

4.6.38 Request Target Position Reached Event

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

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

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

Mnemonic: ---

Binary representation:

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

The *value* field contains a bit mask where every bit stands for one motor:
- bit 0 = motor 0
- bit 1 = motor 1
- bit 2 = motor 2

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

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

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

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

4.6.39 BIN (return to binary mode)

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

Related Commands: none

Mnemonic: BIN

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

4.6.40 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 NO.	TYPE	COMMAND	DESCRIPTION
136	0 – string 1 – binary	Get 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 This command does not send back a reply. *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. 351V.442)

There is no checksum in this reply format!

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

Byte index in value field	Contents
1	01
2	5F
3	Type number, low byte
4	Type number, high byte

5 Axis Parameters

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

MEANING OF THE LETTERS IN COLUMN *ACCESS*:

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

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

Number	Axis Parameter	Description	Range	Acc.
0	Target (next) position	The desired position in position mode (see ramp mode, no. 138).	-2^{31}… 2^{31}-1 [µsteps]	RW
1	Actual position	The current position of the motor. Should only be overwritten for reference point setting.	-2^{31}… 2^{31}-1 [µsteps]	RW
2	Target (next) speed	The desired speed in velocity mode (see ramp mode, no. 138). In position mode, this parameter is set by hardware: to the maximum speed during acceleration, and to zero during deceleration and rest.	±2047 $\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$	RW
3	Actual speed	The current rotation speed.	±2047 $\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$	RW
4	Maximum positioning speed	Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC429 datasheet for calculation of physical units.	0… 2047 $\left[\frac{16\text{MHz}}{65536}\cdot 2^{\text{PD}}\frac{\mu\text{steps}}{\text{sec}}\right]$	RWE
5	Maximum acceleration	The limit for acceleration (and deceleration). Changing this parameter requires re-calculation of the acceleration factor (no. 146) and the acceleration divisor (no. 137), which is done automatically. See TMC429 datasheet for calculation of physical units.	0… 2047*[1]	RWE
6	Absolute max. current (CS / Current Scale)	The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0… 255 and can be adjusted in 32 steps. 0… 7 ‖ 79…87 ‖ 160… 167 ‖ 240… 247 8… 15 ‖ 88… 95 ‖ 168… 175 ‖ 248… 255 16… 23 ‖ 96… 103 ‖ 176… 183 24… 31 ‖ 104… 111 ‖ 184… 191 32… 39 ‖ 112… 119 ‖ 192… 199 40… 47 ‖ 120… 127 ‖ 200… 207 48… 55 ‖ 128… 135 ‖ 208… 215 56… 63 ‖ 136… 143 ‖ 216… 223 64… 71 ‖ 144… 151 ‖ 224… 231 72… 79 ‖ 152… 159 ‖ 232… 239 *The most important motor setting, since too high values might cause motor damage!*	0… 255 $I_{peak} = < value > \times \frac{4A}{255}$ $I_{RMS} = < value > \times \frac{2.8A}{255}$	RWE

Number	Axis Parameter	Description	Range	Acc.
7	Standby current	The current limit two seconds after the motor has stopped.	0… 255 $I_{peak} = < value > \times \frac{4A}{255}$ $I_{RMS} = < value > \times \frac{2.8A}{255}$	RWE
8	Target pos. reached	Indicates that the actual position equals the target position.	0/1	R
9	Ref. switch status	The logical state of the reference (left) switch. See the TMC429 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.	0/1	R
10	Right limit switch status	The logical state of the (right) limit switch.	0/1	R
11	Left limit switch status	The logical state of the left limit switch (in three switch mode)	0/1	R
12	Right limit switch disable	If set, deactivates the stop function of the right switch	0/1	RWE
13	Left limit switch disable	Deactivates the stop function of the left switch resp. reference switch if set.	0/1	RWE
130	Minimum speed	Should always be set 1 to ensure exact reaching of the target position. Do not change!	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. 0: position mode. Steps are generated, when the parameters actual position and target position differ. Trapezoidal speed ramps are provided. 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. 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*) 1: half step*) 2: 4 microsteps 3: 8 microsteps 4: 16 microsteps 5: 32 microsteps**) 6: 64 microsteps**) Modifying this parameter will affect the rotation speed in the same relation: *) Full-step and half-step settings are not optimized for use without an adapted microstepping table. These settings step through the microstep table in steps of 64 resp. 32. To get real full stepping use axis parameter 211 or load an adapted microstepping table. **) If the module is specified for 16 microsteps only, switching to 32 or 64 microsteps brings an enhancement in resolution and smoothness. The position counter will use the full resolution, but, the motor will resolve a maximum of 24 different microsteps for the 32 or 64 microstep units.	0… 6	RWE

Number	Axis Parameter	Description	Range	Acc.
141	Reference switch tolerance	For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop.	0… 4095 [µsteps]	RW
149	Soft stop flag	If cleared, the motor will stop immediately (disregarding motor limits), when the reference or limit switch is hit.	0/1	RWE
153	Ramp divisor	The exponent of the scaling factor for the ramp generator- should be de/incremented carefully (in steps of one).	0… 13	RWE
154	Pulse divisor	The exponent of the scaling factor for the pulse (step) generator – should be de/incremented carefully (in steps of one).	0… 13	RWE
193	Reference search mode	1 search left stop switch only 2 search rightstop switch, then search left stop switch 3 search right stop switch, then search left stop switch from both sides Please see chapter 7.1 for details on reference search.	1/2/3	RWE
194	Reference search speed	For the reference search this value directly specifies the search speed.	0… 2047	RWE
195	Reference switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected.	0… 2047	RWE
196	Reference switch distance	This parameter provides the distance between the end switches after executing the RFS command (mode 2 or 3).	0… 2.147.483.647	R
200	Boost current	Current used for acceleration and deceleration phases. If set to 0 the same current as set by axis parameter 6 will be used.	0… 255 $I_{peak} =< value >\times \frac{4A}{255}$ $I_{RMS} =< value >\times \frac{2.8A}{255}$	RWE
203	Mixed decay threshold	If the actual velocity is above this threshold, mixed decay will be used. Set this parameter to -1 to turn on mixed decay permanently – also in the rising part of the microstep wave. This can be used to fix microstep errors.	0… 2048 or -1	RWE
204	Freewheeling	Time after which the power to the motor will be cut when its velocity has reached zero.	0… 65535 0 = never [msec]	RWE
205	Stall detection threshold	The motor will be stopped if the load value exceeds the stall detection threshold. 0 no stall detection 1… 7 Stall detection threshold setting: 1 (low threshold) up to 7 (high threshold). *Switch off mixed decay to get usable results.*	0… 7	RWE
206	Actual load value	Readout of the actual load value used for stall detection.	0… 7	R
207	Extended error flags	Bit 0: motor has been stopped due to encoder deviation error. Bit 1: motor has been stopped due to motor stall. These two flags are cleared with the next movement command.	0… 3	R

Number	Axis Parameter	Description	Range	Acc.
208	Driver error flags	Bit 0: Overcurrent bridge A low side Bit 1: Overcurrent bridge B low side Bit 2: Open load bridge A Bit 3: Open load bridge B Bit 4: Overcurrent high side Bit 5: Driver undervoltage Bit 6: Temperature warning Bit 7: Overtemperature	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 chapter 7.2.	RWE
211	Fullstep threshold	When exceeding this speed the driver will switch to real full step mode. To disable this feature set this parameter to zero or to 2048. Setting a full step threshold allows higher motor torque of the motor at higher velocity. When experimenting with this in a given application, try to reduce the motor current in order to be able to reach a higher motor velocity!	0… 2048	RWE
212	Maximum encoder deviation	When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero.	0… 65535 [encoder steps]	RWE
213	Group index	All motors on one module that have the same group index will also get the same commands when a ROL, ROR, MST, MVP or RFS is issued for one of these motors.	0… 255	RW
214	Power down delay	Standstill period before the current is changed down to standby current. The standard value is 200msec.	1… 65535 [10msec]	RWE

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

6 Global Parameters

GLOBAL PARAMETERS ARE GROUPED INTO 4 BANKS:

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

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

6.1 Bank 0

PARAMETERS 0… 38

The first parameters 0…38 are only mentioned here for completeness. They are used for the internal handling of the TMCL-IDE and serve for loading micro step 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	TMC428/429 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 RS232/RS485/USB baud rate or the CAN bit rate. Change these parameters to meet your needs. The best and easiest way to do this is to use the appropriate functions of the TMCL-IDE. The parameters with numbers between 64 and 128 are stored in EEPROM only.

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

MEANING OF THE LETTERS IN COLUMN *ACCESS*:

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

Number	Global parameter	Description	Range	Acc.
64	EEPROM magic	Setting this parameter to a different value as $E4 will cause re-initialization of the axis and global parameters (to factory defaults) after the next power up. This is useful in case of miss-configuration.	0… 255	RWE
65	RS232/RS485*) baud rate	0 – 9600 baud – *Default* 1 – 14400 baud 2 – 19200 baud 3 – 28800 baud 4 – 38400 baud 5 – 57600 baud 6 – 76800 baud – *Not supported by Windows!* 7 – 115200 baud 8 – 230400 baud 9 – 250000 baud – *Not supported by Windows!* 10 – 500000 baud – *Not supported by Windows!* 11 – 1000000 baud – *Not supported by Windows!* *Warning:* *The highest speed for RS232 is 115200 baud limited by the RS232 transceiver.* *The RS232 might work with higher speed but out of specification.*	0… 11	RWE
66	Serial address	The module (target) address for RS232/RS485.	0… 255	RWE
67	ASCII mode	Configure the TMCL ASCII interface: Bit 0: 0 – start up in binary (normal) mode 1 – start up in ASCII mode Bits 4 and 5: 00 – Echo back each character 01 – Echo back complete command 10 – Do not send echo, only send command reply		RWE
68	Serial heartbeat	Serial heartbeat for the RS232/RS485 interface. If this time limit is up and no further command is noticed the motor will be stopped. 0 – parameter is disabled	[ms]	RWE
69	CAN bit rate	2 – 20kBit/s 3 – 50kBit/s 4 – 100kBit/s 5 – 125kBit/s 6 – 250kBit/s 7 – 500kBit/s 8 – 1000kBit/s – *Default*	2… 8	RWE
70	CAN reply ID	The CAN ID for replies from the board (default: 2)	0..7ff	RWE
71	CAN ID	The module (target) address for CAN (default: 1)	0..7ff	RWE
73	Configuration EEPROM lock flag	Write: 1234 to lock the EEPROM, 4321 to unlock it. Read: 1=EEPROM locked, 0=EEPROM unlocked.	0/1	RWE
75	Telegram pause time	Pause time before the reply via RS232 or RS485 is sent. For RS232 set to 0. For RS485 it is often necessary to set it to 15 (for RS485 adapters controlled by the RTS pin). For CAN interface this parameter has no effect!	0… 255	RWE
76	Serial host address	Host address used in the reply telegrams sent back via RS232 or RS485.	0… 255	RWE
77	Auto start mode	0: Do not start TMCL application after power up (default). 1: Start TMCL application automatically after power up.	0/1	RWE

Number	Global parameter	Description	Range	Acc.
80	Shutdown pin functionality	Select the functionality of the SHUTDOWN pin 0 – no function 1 – high active 2 – low active	0… 2	RWE
81	TMCL code protection	Protect a TMCL program against disassembling or overwriting. 0 – no protection 1 – protection against disassembling 2 – protection against overwriting 3 – protection against disassembling and overwriting *If you switch off the protection against disassembling, the program will be erased first!* *Changing this value from 1 or 3 to 0 or 2, the TMCL program will be wiped off.*	0,1,2,3	RWE
82	CAN heartbeat	Heartbeat for CAN interface. If this time limit is up and no further command is noticed the motor will be stopped. 0 – parameter is disabled	[ms]	RWE
83	CAN secondary address	Second CAN ID for the module. Switched off when set to zero.	0… 7ff	RWE
84	Coordinate storage	0 – coordinates are stored in the RAM only (but can be copied explicitly between RAM and EEPROM) 1 – coordinates are always stored in the EEPROM only	0 or 1	RWE
85	Do not store user variables	0 – user variables are restored (*default*) 1 – user variables are not restored	0/1	RWE
87	Serial secondary address	Second module (target) address for RS232 / RS485.	0… 255	RWE
128	TMCL application status	0 –stop 1 – run 2 – step 3 – reset	0..3	R
129	Download mode	0 – normal mode 1 – download mode	0/1	R
130	TMCL program counter	The index of the currently executed TMCL instruction.		R
132	tick timer	A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value.		RW
133	random number	Choose a random number.	0…2147483647	R

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

6.2 Bank 1

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

6.3 Bank 2

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

Up to 56 user variables are available.

MEANING OF THE LETTERS IN COLUMN *ACCESS*:

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

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

6.4 Bank 3

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

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

MEANING OF THE LETTERS IN COLUMN *ACCESS*:

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

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

7 Hints and Tips

This chapter gives some hints and tips on using the functionality of TMCL, for example how to use and parameterize the built-in reference point search algorithm.

7.1 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 settings of the automatic stop functions corresponding to the switches (axis parameters 12 and 13) have no influence on the reference search.

Definition of the switches

- Selecting the referencing mode (axis parameter 193): in modes 1 and 2, the motor will start by moving *left* (negative position counts). In mode 3 (three-switch mode), the right stop switch is searched first to distinguish the left stop switch from the reference switch by the order of activation when moving left (reference switch and left limit switch share the same electrical function).
- 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 right until the switch is released. Finally the switch is re-entered in left 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).
- In the drawings shown here the connection of the left and the right limit switch can be seen. Also the connection of three switches as left and right limit switch and a reference switch for the reference point are shown. The reference switch is connected in series with the left limit switch. The differentiation between the left limit switch and the reference switch is made through software. Switches with open contacts (normally closed) are used.
- In circular systems there are no end points and thus only one reference switch is used for finding the reference point.

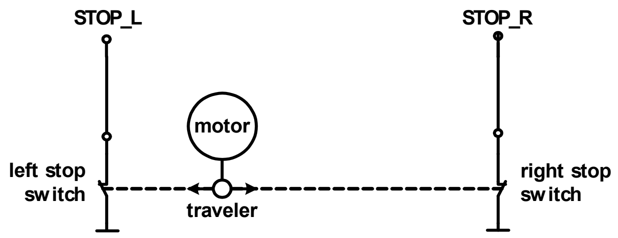


Figure 7.1 Left and right limit switches

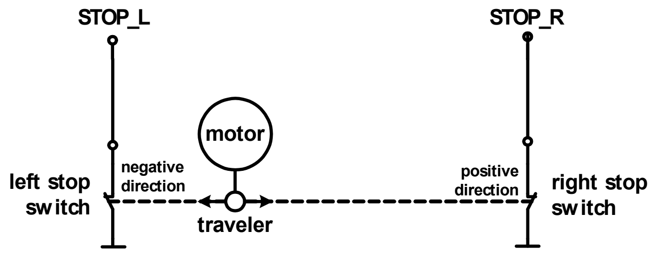


Figure 7.2 Limit switches and reference switch

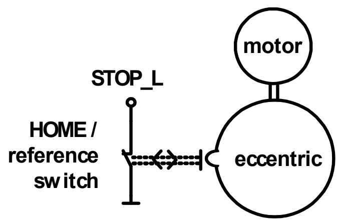


Figure 7.3 One reference switch

7.2 Changing the Prescaler Value of an Encoder

For changing the prescaler value of an encoder, axis parameter 210 is used:

- To change the prescaler of encoder 0 use SAP 210, 0, <p>.
- To change the prescaler of encoder 1 use SAP 210, 1, <p>.
- To change the prescaler of encoder 2 use SAP 210, 2, <p>.

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

Value for <p>	Resulting prescaler	SAP command for motor 0 SAP 210, 0, <p>	Resulting steps per rotation for a 400 line (1600 quadrate count) encoder
64	0.125	SAP 210, M0, 64	200
128	0.25	SAP 210, M0, 128	400
256	0.5	SAP 210, M0, 256	800
512	1	SAP 210, M0, 512	1600
800	1.5625	SAP 210, M0, 800	2500
66144	1.6	SAP 210, M0, 66144	2560
1024	2	SAP 210, M0, 1024	3200
1600	3.125	SAP 210, M0, 1600	5000
67104	3.2	SAP 210, M0, 67104	5120
2048	4	SAP 210, M0, 2048	6400
3200	6.25	SAP 210, M0, 3200	10000
68672	6.4	SAP 210, M0, 68672	10240
4096	8	SAP 210, M0, 4096	12800
6400	12.5	SAP 210, M0, 6400	20000
71808	12.8	SAP 210, M0, 71808	20480
8192	16	SAP 210, M0, 8192	25600
78432	25.6	SAP 210, M0, 78432	40960
16384	32	SAP 210, M0, 16384	51200
32768	64	SAP 210, M0, 32768	102400

FORMULA FOR RESULTING STEPS PER ROTATION:

StepsPerRotation = LinesOfEncoder * 4 * 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>:

Add to <p>	Special function
16	Null channel is active high (default: null channel is active low)
8	Hold encoder value only when null channel is triggered (default: always hold encoder value)
4	Clear encoder value when null channel is triggered (default: do not clear on null channel)
2	Trigger null channel at every N signal (default: only at next N signal)
1	Add when rotating CCW, subtract when rotating CW (default: add on CW, subtract on CCW)

Example:

For a prescaler value of 4 with an active high null channel use a p-value of 2048 + 16 = 2064

7.3 Stall Detection

The TMCM-351 is equipped with three TMC249 motor driver chips. These chips feature load measurement that can be used for stall detection. Stall detection means that the motor will be stopped when the load gets too high. It is controlled by axis parameter 205. If this parameter is set to a value between 1 and 7 the stall detection will be activated. Setting it to 0 means that stall detection is turned off. A greater value means a higher threshold. This also depends on the motor and on the velocity. There is no stall detection while the motor is being accelerated or decelerated.

STALL DETECTION CAN BE USED FOR FINDING THE REFERENCE POINT. THEREFORE, USE THE FOLLOWING TMCL CODE:

```
      SAP 205, 0, 5   //Turn on Stall Detection (use other threshold if needed)
      ROL 0, 500      //Let the motor run (or use ROR or other velocity)
Loop: GAP 3, 0
      COMP 0
      JC NE, Loop     //Wait until the motor has stopped
      SAP 1, 0, 0     //Set this position as the zero position
```

Do not use RFS in this case.
Mixed decay should be switched off when stallGuard operational in order to get usable results.

7.4 Fixing Microstep Errors

Due to the *zero crossing problem* of the TMC249 stepper motor drivers, microstep errors may occur with some motors as the minimum motor current that can be reached is slightly higher than zero (depending on the inductivity, resistance and supply voltage of the motor).
This can be solved by setting the *mixed decay threshold* parameter (axis parameter number 203) to the value –1. This switches on mixed decay permanently, in every part of the microstepping waveform. Now the minimum reachable motor current is always near zero which gives better microstepping results.
A further optimization is possible by adapting the motor current shape. (For further information about TMCL-IDE please refer to the TMCL reference and programming manual.)

Use *SAP 203, <motor number>, -1* to turn on this feature.

7.5 Using the RS485 Interface

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

For RS232 set the telegram pause time to zero for maximum data throughput

8 Life Support Policy

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

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

9 Revision History

9.1 Firmware Revision

Version	Date	Author	Description
4.17	2009-FEB-28	OK	First version supporting all TMCL features
4.28	2010-AUG-09	OK	RFS start resets deviation flags, too. Thus, a reference search is stopped if an encoder deviation is detected.
4.29	2010-OKT-31	OK	Sign error in CANopen version corrected.
4.30	2010-DEC-16	OK	TMCL firmware updates for other modules.
4.31	2011-APR-01	OK	System control improved: every 5ms.
4.32	2011-JUN-06	OK	TMCL firmware updates for other modules.
4.33	2011-JUL-27	OK	Soft stop in case of contouring error enabled.
4.34	2011-SEP-09	OK	TMCL firmware updates for other modules.
4.35	2011-SEP-18	OK	EEPROM readout process updated.
4.36	2011-DEC-01	OK	TMCL firmware updates for other modules.
4.37	2012-JAN-06	OK	Axis parameter 200 (boost current) added Positioning range enlarged: 32 Bit.
4.38	2012-MAR-15	OK	Reference search adapted to 32 Bit range.
4.39	2012-APR-26	OK	TMCL firmware updates for other modules.
4.40	2012-JUN-15	OK	TMCL firmware updates for other modules.
4.41	2012-SEP-21	OK	Global parameter 87 (secondary address for RS232/RS485) added. Reference search: the last position before setting the counter to zero can be read out with axis parameter 197.
4.42	2012-NOV-16	OK	Axis parameter 130 (min. current) updated.
4.43	2013-FEB-20	OK	Not deployed.
4.44	2013-OKT-15	OK	Not deployed.
4.45	21.01.2014	OK	Improved USB connection. Improved command *request target position reached*.

9.2 Document Revision

Version	Date	Author	Description
1.00	2009-MAY-29	SD	Initial version
1.01	2009-JUN-26	OK	Description of axis parameter 194 corrected
1.02	2009-JUL-31	SD	SIO and GIO commands corrected, minor changes
1.03	2010-SEP-25	SD	SIO completed (adjusting the input range), global parameter 69 (bank 0) corrected
1.04	2012-NOV-05	SD	Global Parameter 65 updated.
1.05	2012-DEC-17	SD	Interrupt description added. Several axis parameters and global parameters updated resp. added. Design changes.
1.06	2014-MAY-16	SD	Firmware revision updated.

10 References

[TMCM-351]	TMCM-351 Hardware Manual
[TMCL-IDE]	TMCL-IDE User Manual

(see http://www.trinamic.com)