TMCL
Reference and Programming Manual

Version: 2.22
May 15th, 2008

Trinamic Motion Control GmbH & Co. KG
Sternstraße 67
D – 20357 Hamburg, Germany
http://www.trinamic.com

Version

Version	Date	Author	Comment
2.11	22-Dec-04	OK	Corrections for SCO/GCO/CCO, Code protection feature
2.12	21-Feb-05	OK	ASCII interface included
2.13	22-Mar-05	OK	Minor error corrections
2.14	3-Jun-05	OK	New features of TMCL V3.23 included
2.15	11-Nov-05	OK	TMCM-109 and TMCM-111 added
2.16	20-Dec-05	OK	Features of version TMCL V3.26 and V3.27 included
2.17	22-Jun-06	OK	Error with CCO command corrected
2.18	20-Nov-06	OK	GAP/GGP/GIO/SIO command descriptions extended
2.20	23-Aug-07	OK	Command 138 added, Chapter 8 added
2.21	5-Dec-07	OK	TMCL Debugger added
2.22	5-Jun-08	OK	MVP COORD with TMCM-61x and TMCM-34x added

Contents

1 Introduction

The Trinamic Motion Control Language (TMCL) provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM module to form programs that run stand alone on a 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). So, TMCL forms a powerful language that can either be used to control a module directly from a host (direct mode) or to program applications that run on a stand-alone module (program mode or stand-alone mode).
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 stand-alone TMCL applications using the TMCL IDE (IDE means "Integrated Development Environment").
There is also a set of configuration variables for every 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. It also describes how to use the TMCL IDE.

2 Basic TMCL Concepts

2.1 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 or RS485 interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. So it then consists of nine bytes. This is not the case when communicating via the CAN bus as address and checksum are included in the CAN standard in do not have to be supplied by the user.
When using a module with IIC interface the first byte (address byte) is left out because IIC has its own addressing scheme. So for IIC the telegram consists of eight bytes, starting with the command byte.
So the binary command format when using RS232 or RS485 is as follows:

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

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

2.1.1 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
```

2.2 The reply format

Every time a command has been sent to a module, the module sends a reply. When using RS232 or RS485 the format of the reply 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. When using IIC bus the first byte (reply address) is left out. **Do not send the next command before you have received the reply!**

2.2.1 Status codes

The reply contains a status code. This 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

2.3 Stand-alone applications

When the module which is used is equipped with an EEPROM to store TMCL applications, the TMCL-IDE can be used to develop stand-alone TMCL applications that can be downloaded into the EEPROM of the module and then run on the module. The TMCL IDE contains an editor and a "TMCL assembler" where the commands can be entered using their mnemonic format and then assembled to their binary representations. This code can then be downloaded into the module to be executed there. The TMCL IDE is described in detail in chapter 7 of this manual.

2.4 TMCL command overview

In this section an overview of the TMCL commands is given. All commands are described in detail in chapter 3. Some TMCL programming techniques are given in chapter 8.

2.4.1 Motion commands

These commands control the motion of the motors. They are the most important commands and can be used in direct mode or in stand-alone 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

2.4.2 Parameter commands

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

2.4.3 I/O port commands

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

Mnemonic	Command number	Meaning
SIO	14	Set output
GIO	15	Get input
SAC	29	Access to external SPI device

2.4.4 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 stand-alone 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

2.4.5 Calculation commands

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

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

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.

2.5 The ASCII interface

Since TMCL V3.21 there is also an ASCII interface that can be used to communicate with the module and to send some commands as text strings. The ASCII command line interface is entered by sending the binary command 139 (enter ASCII mode). The commands are then entered as in the TMCL IDE, but not all commands can be entered in ASCII mode. Only those commands that can be used in direct mode can also be entered in ASCII mode.
To leave the ASCII mode and re-enter the binary mode enter the command "BIN".

2.5.1 Format of the command line

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

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

These command would all 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.

2.5.2 Format of a reply

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

2.5.3 Commands that can be 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, SAC, SCO, GCO, CCO, UF0, UF1, UF2, UF3, UF4, UF5, UF6, UF7.

There are also special commands that 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.

2.5.4 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 5.1). Bit 0 determines the startup mode: if this bit is set, the module starts up in ASCII mode, else it will start up in binary mode (default). Bit 4 and Bit 5 determine how the characters that are entered are echoed back. Normally, both bits are set to zero. In this case every character that is entered is echoed back when the module is addressed. Character can also be erased using the backspace character (press the backspace key in a terminal program). When bit 4 is set and bit 5 is clear the characters that are entered are not echoed back immediately but the entire line will be echoed back after the <CR> character has been sent. When bit 5 is set and bit 4 is clear there will be no echo, only the reply will be sent. This may be useful in RS485 systems.

3 TMCL Command Dictionary

This chapter describes all TMCL commands. The commands are sorted by their command numbers. For every command the mnemonic with its syntax and the binary representation are given.
Ranges for the parameters are also given. They sometimes differ between the types of the modules. If this is the case, value ranges are given for every module.
Last but not least some examples are given at the end of every command description. In the examples for the binary representation always RS232/RS485 communication (9 byte) format is used with module address 1 and host address 2. When using CAN bus communication just leave out the first and the last byte (as mentioned in chapter 2.1).

3.1 ROR – Rotate Right

Description: This instruction starts rotation in "right" direction, i.e. increasing the position counter.

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

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR <motor number>, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
1	(don't care)	<motor number>	<velocity>

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Motor number	Velocity
TMCM-300/301/302/303/310	0..2	0..2047
TMCM-100	always 0	0..8191

Example:

Rotate right, motor #2, velocity = 350

Mnemonic: ROR 2, 350

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$01	$00	$02	$00	$00	$01	$5e	$62

Note for TMCM-300/301/302/303/310 modules:
Adjust the axis parameter "pulse divisor" (no. 154, see Chapter 5), if the speed value is very low (<50) or above the upper limit (2047). See TMC 428 datasheet (p.24) for calculation of physical units.

Note for TMCM-100 modules:
Adjust the axis parameter "pre-divider" if the speed is not within the desired range.

Note:
With some modules it is possible to use stall detection. Please see section 6.4 for details.

3.2 ROL – Rotate Left

Description: This instruction starts rotation in "left" direction, i.e. decreasing the position counter.

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

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL <motor number>, <velocity>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
2	(don't care)	<motor number>	<velocity>

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Motor number	Velocity
TMCM-300/301/302/303/310	0..2	0..2047
TMCM-100	always 0	0..8191

Example:

Rotate left, motor #1, velocity = 1200

Mnemonic: ROL 1, 1200

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$02	$00	$01	$00	$00	$04	$b0	$b8

Note for TMCM-300/301/302/303/310 modules:
Adjust the axis parameter "pulse divisor" (no. 154, see Chapter 5), if the speed value is very low (<50) or above the upper limit (2047). See TMC 428 datasheet (p.24) for calculation of physical units.

Note for TMCM-100 modules:
Adjust the axis parameter "pre-divider" if the speed is not within the desired range.

Note:
With some modules it is possible to use stall detection. Please see section 6.4 for details.

3.3 MST – Motor Stop

Description: This instruction stops the motor.

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

Related commands: ROL, ROR, SAP, GAP

Mnemonic: MST <motor number>

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Motor number
TMCM-300/301/302/303/310	0..2
TMCM-100	always 0

Example: Stop motor #1

Mnemonic: MST 1

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$03	$00	$01	$00	$00	$00	$00	$05

Note:
With some modules it is possible to use stall detection. Please see section 6.4 for details.

3.4 MVP – Move to Position

Description: A movement towards the specified position is started, with automatic generation of acceleration- and deceleration ramps. The maximum velocity and acceleration are defined by axis parameters #4 and #5.
Three operation types are available:
- Moving to an absolute position in the range from - 8388608 to +8388608 (-2^{23} to $+2^{23}$).
- Starting a relative movement by means of an offset to the actual position. In this case, the resulting new position value must not exceed the above mentioned limits, too.
- Moving one or more motors to a (previously stored) coordinate, see SCO (section 3.25) for details. When moving more than one axis the module will try to interpolate: The velocities will be calculated so that all motors reach their target positions at the same time. It is important that the maximum accelerations (axis parameter #5) and the ramp and pulse dividers (axis parameters #153 and #154) of all axes are set to the same values as otherwise interpolation will not work correctly. With TMCM-100 modules there is no interpolation as it controls only one axis.

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

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

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
4	0 ABS – absolute	<motor number>	<position>
	1 REL – relative	<motor number>	<offset>
	2 COORD – coordinate	0 motor 0 1 motor 1 2 motor 2 TMCM-301, 302, 303, 310: 3 (not allowed) 4 motors 0&1 5 motors 1&2 6 motors 0&2 7 motors 0,1,2 TMCM-6xx, 34x: see below	<coordinate number (0..20) (Version >=3.17: 0..56)>

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges: The <motor number> parameter depends on the module type: always 0 on single axis modules, 0..2 on three axis modules and 0..5 on six axis modules. With MVP COORD, the motor number is interpreted differently.

Examples:
Move motor #1 to (absolute) position 90000
Mnemonic: MVP ABS, 1, 9000
Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum

Value (hex)	$01	$04	$00	$01	$00	$01	$5f	$90	$f6

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

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$04	$01	$00	$ff	$ff	$fc	$18	$18

Move motor #2 to previously stored coordinate #8
Mnemonic: MVP COORD, 2, 8
Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$04	$02	$02	$00	$00	$00	$08	$11

Note:
With some modules it is possible to use stall detection. Please see section 6.4 for details.

MVP COORD on TMCM-6xx and TMCM-34x modules
On these modules, there are some more options for the MVP COORD command. For this reason the "motor" parameter with the MVP COORD command is interpreted as follows on those modules:

- Moving only one motor: set the "Motor" parameter to the motor number (0..5).
- Moving multiple motors without interpolation: Set bit 7 of the "Motor" parameter. Now the bits 0..5 of the "Motor" parameter define which motors are to be started. Each of these bits stands for one motor.
- Moving multiple motors using interpolation: Set bit 6 of the "Motor" parameter. . Now the bits 0..5 of the "Motor" parameter define which motors are to be moved using interpolation. Each of these bits stands for one motor. It is not possible to start a group of more than three motors using interpolation. However, it is possible to start one group of three motors right after starting a group of other three motors.

On the TMCM-610/611/612 modules the interpolation feature is not available in firmware versions prior to 6.31.
On the TMCM-341/342/343 modules the interpolation feature is available in all firmware versions that have been released for these modules.

Examples:

- MVP COORD, $47, 2 moves motors 0, 1 and 2 to coordinate 2 using interpolation.
- MVP COORD, $87, 5 moves motors 0, 1 and 2 to coordinate 5 without using interpolation.

3.5 SAP – Set Axis Parameter

Description: Most parameters of a TMCM module can be adjusted individually for each axis. Although these parameters vary widely in their formats (1 to 24 bits, signed or unsigned) and physical locations (TMC428, TMC453, controller RAM, controller EEPROM), they all can be set by this function. See chapter 4 for a complete list of all axis parameters. See STAP (section 3.7) for permanent storage of a modified value.

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

Related commands: GAP, STAP, RSAP , AAP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Parameter number	Motor number	Value
TMCM-300/301/302/303/310	s. chapter 4	0..2	s. chapter 4
TMCM-100	s. chapter 4	always 0	s. chapter 4

Example:
set the absolute maximum current of motor #1 to 200mA

Mnemonic: SAP 6, 1, 200
Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$05	$06	$01	$00	$00	$00	$c8	$d5

Note:
The parameter numbers are different in TMCL firmware versions before 2.18. Please upgrade to a newer firmware version if you have such an old version.

3.6 GAP – Get Axis Parameter

Description: Most parameters of a TMCM module can be adjusted individually for each axis. Although these parameters vary widely in their formats (1 to 24 bits, signed or unsigned) and physical locations (TMC428, TMC453, controller RAM, controller EEPROM), they all can be read by this function. In stand-alone 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. See chapter 4 for a complete list of all parameters.

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

Related commands: SAP, STAP, AAP, RSAP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Parameter number	Motor number
TMCM-300/301/302/303/310	s. chapter 4	0..2
TMCM-100	s. chapter 4	always 0

Example: get the actual position of motor #2
Mnemonic: GAP 2, 1

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$06	$01	$02	$00	$00	$00	$00	$0a

Reply:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Host-address	Target-address	Status	Instruction	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$02	$01	$64	$06	$00	$00	$02	$c7	$36

⇨ **status=no error, position=711**

3.7 STAP – Store Axis Parameter

Description: Axis parameters are located in RAM memory, so modifications are lost at power down. This instruction enables permanent storing. Most parameters are automatically restored after power up (see axis parameter list in chapter 4).

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

Related commands: SAP, RSAP, GAP, AAP

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

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
7	<parameter number>	<motor number>	(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)

Parameter ranges:

Parameter number	Motor number	Value
s. chapter 4	0..2	s. chapter 4
s. chapter 4	always 0	s. chapter 4

Example: store the maximum speed of motor #1
Mnemonic: STAP 4, 1

Binary:

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

Note: The STAP command will not have any effect when the configuration EEPROM is locked (s. 5.1, "EEPROM lock flag" and section 7.6.2.5). In direct mode, the error code 5 (configuration EEPROM locked, see also section 2.2.1) will be returned in this case.

3.8 RSAP – Restore Axis Parameter

Description: For all configuration-related axis parameters, non-volatile memory locations are provided. By default, most parameters are automatically restored after power up (see axis parameter list in chapter 4). A single parameter that has been changed before can be reset by this instruction.

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

Relate commands: SAP, STAP, GAP, AAP

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

Binary representation:

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

Reply structure in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example: restore the maximum current of motor #1
Mnemonic: RSAP 6, 1

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$08	$06	$01	$00	$00	$00	$00	$10

3.9 SGP – Set Global Parameter

Description: Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organised 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. See chapter 0 for a complete parameter list.

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	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$09	$42	$00	$00	$00	$00	$03	$4f

3.10 GGP – Get Global Parameter

Description: All global parameters can be read with this function. In stand-alone mode, the result is copied to the accumulator register for further processing purposes such as conditional jumps. In direct mode, the result is only output in the "value" field of the reply, without affecting the accumulator. See chapter 0 for a complete parameter list.

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	(see chapter 6)	<bank number> see chapter 6	(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	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$0a	$42	$00	$00	$00	$00	$00	$4d

Reply:

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

⇨ **Status=no error, Value=1**

3.11 STGP – Store Global Parameter

Description: Some global parameters are located in RAM memory, so modifications are lost at power down. This instruction enables permanent storing. Most parameters are automatically restored after power up (see the list of global parameters in chapter 0).

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	(see chapter 6)	<bank number> (see chapter 6)	(don't care)

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example: store the serial address of the target device
Mnemonic: STGP 42, 0

Binary:

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

Note: The STGP command will not have any effect when the configuration EEPROM is locked (s. 5.1, "EEPROM lock flag" and section 7.6.2.5). In direct mode, the error code 5 (configuration EEPROM locked, see also section 2.2.1) will be returned in this case.

3.12 RSGP – Restore Global Parameter

Description: This instruction recovers the (permanently) stores the value of a RAM-located parameter. Please see chapter 0 for a list of available parameters.

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

Related commands: SGP, STGP, GGP, AGP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example: restore the serial address of the device
Mnemonic: RSGP 66, 0

Binary:

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

3.13 RFS – Reference Search

Description: A build-in reference point search algorithm can be started (and stopped). The reference search algorithm provides switching point calibration and three switch modes. The status of the reference search can also be queried to see if it has already finished. (In a TMCL program it is better to use the WAIT command to wait for the end of a reference search.) Please see the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet your needs (chapter 4). The reference search can be started or stopped, or the actual status of the reference search can be checked.

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

Related commands: WAIT

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

Binary representation:

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

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

STATUS	VALUE
100 – OK	(don't care)

When using type 2 (STATUS):

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

Example: start reference search of motor #1
Mnemonic: RFS START, 1

Binary:

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

Note:
With some modules it is possible to use stall detection instead of a reference switch. Please see section 6.4 for details.

3.14 SIO – Set Output

Description: This command sets the status of a digital output either to low (0) or to high (1). Please see the list of the outputs in this section.

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

Related commands: GIO, WAIT

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

Binary representation:

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

Reply structure:

STATUS	VALUE
100 – OK	(don't care)

Example: turn the LED on the TMCM-300 base board off (bank 2, line 7)
Mnemonic: SIO 7, 2, 1

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$0e	$07	$02	$00	$00	$00	$01	$19

Overview of available output ports:
With the TMCM-100/301/302/303/310/610 and TMCM-300 modules, only bank 2 is available and contains 8 output lines:

Type	I/O line	Range
0	DOUT0	0/1
1	DOUT1	0/1
2	DOUT2	0/1
3	DOUT3	0/1
4	DOUT4	0/1
5	DOUT5	0/1
6	DOUT6	0/1
7	DOUT7	0/1

Please see the manuals of the individual modules for detailed electrical description of the output ports of a module.

Extension in TMCL version 3.23:
On the TMCM-301/302/303/310 and 61x modules it is now also possible to switch the pull-up resistors on the inputs on and off. When using the inputs as digital inputs it is often useful to have the pull-up resistors switched on, but when using the inputs as analogue inputs the pull-up resistors must be switched of. For this purpose, set the bank parameter to 0 and the type parameter to the number of the desired input (please see the GIO command for a list of the input numbers). Set the value parameter to 0 to switch off and to 1 to switch on a pull-up resistor. The pull-up resistors are switched **off** by default.

Extension in TMCL version 3.30 and 6.22:
On the TMCM-3xx, TMCM-34x and TMCM-61x modules it is now possible to address all eight output lines with one single SIO command. This can be done by setting 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: SIO 255, 2, 255 sets all output pins high.
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
```

3.15 GIO – Get Input / Output

Description: This function reads a digital or analogue input port. So, digital lines will read 0 and 1, while the ADC channels deliver their 10 bit result in the range of 0…1023. In stand-alone mode the requested value is copied to the "accumulator" (accu) for further processing purposes such as conditioned jumps. In direct mode the value is only output in the "value" field of the reply, without affecting the accumulator. The actual status of a digital output line can also be read.

Internal function: The specified line is read.

Related commands: SIO, WAIT

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	<status of the port>

Example: get the analogue value of ADC channel 3
Mnemonic: GIO 3, 1

Binary:

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

Reply:

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

⇨ value: 506

Overview of available input ports:
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.
Note: When using a digital input it is often useful to switch on its pull-up resistor. Please see the SIO command on how to do this.

Type	I/O line	Range
0	ADIN0	0/1
1	ADIN1	0/1
2	ADIN2	0/1
3	ADIN3	0/1
4	ADIN4	0/1
5	ADIN5	0/1
6	ADIN6	0/1
7	ADIN7	0/1
8	OPTO1	0/1
9	OPTO2	0/1
10	SHUTDOWN	0/1

I/O bank 1 – analogue inputs (not on all modules). The ADIN lines can be read as digital or analogue inputs at the same time. The digital states can be accessed in bank 0.
Note: When using an input in analogue mode its pull-up resistor must be switched off. Please see the SIO command on how to do this.

Type	I/O line	Range
0	ADIN0	0…1023
1	ADIN1	0…1023
2	ADIN2	0…1023
3	ADIN3	0…1023
4	ADIN4	0…1023
5	ADIN5	0…1023
6	ADIN6	0…1023
7	ADIN7	0…1023

I/O bank 2: here, the status of the digital outputs (see section 3.14) can be read back.

Note: Not all I/O lines are available on every module. Please see the documentation of the individual module for details. There may also be more I/O lines on newer modules.

Extension in TMCL V3.30 and 6.22:
On the TMCM-30x, TMCM-34x and TMCM-61x modules it is now possible to read all eight digital inputs with only one GIO command. This can be done by setting the type parameter to 255 and the bank parameter to 0. In this case the status of all eight digital input lines will be read to the lower eight bits of the accumulator. So the following program can be used to represent the states of the eight input lines on the eight output lines:

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

3.16 CALC – Calculate

Description: 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 (in most cases) 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 <op>, <value>
where <op> is ADD, SUB, MUL, DIV, MOD, AND, OR, XOR, NOT or LOAD

Binary representation:

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

Example: multiply accu by –5000
Mnemonic: CALC MUL, -5000

Binary:

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

3.17 COMP – Compare

Description: 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 intend for use in stand-alone operation only and must not be used indirect mode.

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

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

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 #2 is greater than or equal to 1000.

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

Binary format of the COMP 1000 command:

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

Note that the host address and the reply is only used to transfer this instruction to the TMCL program memory during downloading of a TMCL program. It does not make sense to use this command in direct mode.

3.18 JC – Jump Conditional

Description: 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, e.g. by the COMP instruction. For stand-alone 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>
where <condition>=ZE|NZ|EQ|NE|GT|GE|LT|LE|ETO|EAL|EDV|EPO

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
21	0 ZE - zero 1 NZ - not zero 2 EQ - equal 3 NE - not equal 4 GT - greater 5 GE - greater/equal 6 LT - lower 7 LE - lower/equal 8 ETO - time out error 9 EAL – external alarm 10 EDV – deviation error 11 EPO – position error 12 ESD – shutdown error	(don't care)	<jump address>

EDV and EPO: TMCM-100 only.
EAL and ESD: not available with TMCM-300.

Example: Jump to address given by the label when the position of motor #2 is greater than or equal to 1000.

```
GAP 1, 2, 0          //get axis parameter, type: no. 1 (actual position), motor: 2, value:0 (don't care)
COMP 1000            //compare actual value to 1000
JC GE, Label         //jump, type: 5 greater/equal
…
…
Label: ROL 0, 1000
```

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

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

Note that the host address and the reply is only used to transfer this instruction to the TMCL program memory. See the host-only control functions for details. It is not possible to use this command in direct mode.

3.19 JA – Jump Always

Description: Jump to a fixed address in the TMCL program memory. This command is intended for standalone operation only and not to be used in direct mode.

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

Related commands: JC, WAIT, CSUB

Mnemonic: JA <Label>

Binary representation:

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

Example: An infinite loop in TMCL

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

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

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

Note that the host address and the reply is only used to transfer this instruction to the TMCL program memory. This command can not be used in direct mode.

3.20 CSUB – Call Subroutine

Description: Calls a subroutine in the TMCL program memory. This command is intended for standalone operation only and must not be used in direct mode.

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

Related commands: RSUB, JA

Mnemonic: CSUB <Label>

Binary representation:

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

Example: Call a subroutine

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

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

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

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

Note that the host address and the reply is only used to transfer this instruction to the TMCL program memory. This command can not be used in direct mode.

3.21 RSUB – Return from Subroutine

Description: Return from a subroutine to the command after the CSUB command that called it. This command is intended for use in stand-alone mode only and must not be used in direct mode.

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

Related command: CSUB

Mnemonic: RSUB

Binary representation:

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

Example: please see the CSUB example (section 3.20).

Binary format of RSUB:

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

Note that the host address and the reply is only used to transfer this instruction to the TMCL program memory. This command can not be used in direct mode.

3.22 WAIT – Wait for an event to occur

Description: Pause the execution of the TMCL program until the specified condition is met. This command is intended for stand-alone operation only and must not be used in direct mode. There are 5 different wait conditions that can be used:

- TICKS: wait until the number of timer ticks specified by the <ticks> parameter has been reached. The <motor number> parameter is ignored in this case (set it to zero).
- 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 trigerred. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- LIMSW: wait until a limit switch of the motor specified by the <motor> parameter has been triggered. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.
- RFS: wait until the reference search of the motor specified by the <motor> field has been reached. An optional timeout value (0 for no timeout) must be specified by the <ticks> parameter.

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

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

Related commands: JC, CLE

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

Binary representation:

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

***one tick is 10 milliseconds (in standard firmware)**

Parameter ranges:

Single axis modules: <motor number> must always be 0.
Three axis modules: <motor number> can be 0..2
Six axis modules: <motor number> can be 0..5

Example: wait for motor #1 to reach its target position, without timeout
Mnemonic: WAIT POS, 1, 0

Binary:

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

Note that the host address and the reply is only used to transfer this instruction to the TMCL program memory. This command is not to be used in direct mode.

3.23 STOP – Stop TMCL program execution

Description: Stops executing a TMCL stand-alone program. This command should be placed at the end of every TMCL program. It is intended for stand-alone operation only and must not be used in direct mode.

Internal function: TMCL instruction fetching is stopped.

Related commands: none

Mnemonic: STOP

Binary representation:

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

Example:
Mnemonic: STOP

Binary:

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

The host address and the reply is only used to transfer this instruction to the TMCL program memory. Do not use this command in direct mode.

3.24 SAC – SPI Bus Access

Description: Allows access to external SPI devices connected to the SPI bus of the module. Connection of external SPI devices differs between the different module types. Please contact TRINAMIC for further details and see also the hardware manual of your module.

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>

In version 3.23 of TMCL this command has been extended so that not only direct values but also the contents of the accumulator register can be sent. In stand-alone mode the received data is also stored in the accumulator. Most modules have three chip select outputs (SPI_SEL0, SPI_SEL1, 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 is sent instead. But 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). The bus numbers are as follows (please note the gap in the bus numbers; do not use 1 or 129!):

Bus number	Chip select output
0	SPI_SEL0, output direct value
1	Do not use
2	SPI_SEL1, output direct value
3	SPI_SEL2, output direct value
128	SPI_SEL0, output contents of accumulator
129	Do not use
130	SPI_SLE1, output contents of accumulator
131	SPI_SEL2, output contents of accumulator

3.25 SCO – Set Coordinate

Description: Up to 20 position values (coordinates) can be stored for every axis for use with the MVP COORD command (s. section 3.4). This command sets a coordinate to a specified value.
Note: the coordinate number 0 is only stored in RAM, all others are also stored in the EEPROM.

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

Related commands: GCO, CCO, MVP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Motor number
TMCM-300/301/302/303/310	0..2
TMCM-1xx	always 0
TMCM-610/611/612	0..5

Example: set coordinate #1 of motor #2 to 1000
Mnemonic: SCO 1, 2, 1000

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	\$01	\$1e	\$01	\$02	\$00	\$00	\$03	\$e8	\$0d

3.26 GCO – Get Coordinate

Description: Read a previously stored coordinate. In stand-alone 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. Note: the coordinate number 0 is stored in RAM only, all others are also stored in the EEPROM.

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

Related commands: SCO, CCO, MVP

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Motor number
TMCM-300/301/302/303/310	0..2
TMCM-1xx	always 0
TMCM-610/611/612	0..5

Example: get motor #2 value of coordinate 1
Mnemonic: GCO 1, 2

Binary:

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

Reply:

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

⇨ **Value: 0**

3.27 CCO – Capture Coordinate

Description: The actual positions of the specified axes are copied to the selected coordinate variables.
Note: the coordinate number 0 is stored in RAM only, all others are also stored in the EEPROM.

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 <coodinate number>, <motor number>

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
32	<coordinate number> (0…20)	TMCM-3xx: 0 motor 0 1 motor 1 2 motor 2 3 (not allowed) 4 motors 0&1 5 motors 1&2 6 motors 0&2 7 motors 0,1,2 TMCM-6xx: 0..5 (motor number) TMCM-1xx: always 0 TMCM-34x: 0..2 (motor 0..2)	(don't care)

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges: The <motor number> value must always be 0 on TMCM-100 modules.

Example: store current position of all axes to coordinate 3
Mnemonic: CCO 3, 7

Binary:

Byte Index	0	1	2	3	4	5	6	7	8
Function	Target-address	Instruction Number	Type	Motor/ Bank	Operand Byte3	Operand Byte2	Operand Byte1	Operand Byte0	Checksum
Value (hex)	$01	$20	$03	$07	$00	$00	$00	$00	$2b

3.28 CALCX – Calculate using the X register

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

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

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

Binary representation:

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

Example: multiply accu by X-register
Mnemonic: CALCX MUL

Binary:

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

3.29 AAP – Accumulator to Axis Parameter

Description: 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.
See chapter 4 for a complete list of axis parameters.

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

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Parameter ranges:

Module	Parameter number	Motor number
TMCM-300/301/302/303/310	s. Table	0..2
TMCM-100	s. Table	always 0

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

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

Binary format of the AAP 0,0 command:

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

3.30 AGP – Accumulator to Global Parameter

Description: 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 stand-alone application.

See chapter 6 for a complete list of global parameters.

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

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

Binary representation:

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

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

Binary:

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

3.31 CLE – Clear Error Flags

Description: This command clears the internal error flags. It is intended for use in stand-alone 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 (modules with encoder feedback only, e.g. TMCM-100)
- EPO: clear the position error flag (modules with encoder feedback only, e.g. TMCM-100)

Related commands: JC

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

Binary representation:

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

Example: Reset the timeout flag
Mnemonic: CLE ETO

Binary:

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

3.32 User definable commands (UF0..UF7)

Description: The user definable functions UF0 through UF7 are predefined, "empty" functions for user specific purposes. Contact TRINAMIC for customer specific programming of these functions.

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

Related commands: none

Mnemonic: UF0 .. UF7

Binary representation:

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

Reply in direct mode:

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

3.33 Request target position reached event

Description: 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 specified motors have reached their target positions. This command can only be used in direct mode (in stand alone mode, this is covered by the WAIT command) and hence does not have a mnemonic. Its opcode is 138.
This command is avialable since firmware version 3.26 (on the TMCM-301/302/303/310/110/109/111/112 modules) resp. firmware version 6.24 (on the TMCM-610/611/612/101/102 modules).

Internal function: Send an additional reply when one or more motors have reached their target positions. Only usable in direct mode.

Mnemonic: ---

Binary representation:

INSTRUCTION NO.	TYPE	MOT/BANK	VALUE
138	(don't care)	(don't care)	Motor bit mask

The contents of the "value" field is a bit mask where every bit stands for one motor: bit 0 = motor 0, bit 1 = motor 1, and so on. The additional reply will be sent when all motors for which the bits in the bit mask are set have reached their target positions.

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

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

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

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

3.34 BIN – Return to Binary Mode

Description: This command can only be used in ASCII mode. It quits the ASCII mode and returns to binary mode. It is available in version 3.21 or higher.

Related Commands: none

Mnemonic: BIN

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

3.35 TMCL Control Functions

Description: The following functions are for host control purposes only and are not allowed for stand-alone mode. **In most cases, there is no need for the customer to use one of those** (except command 139). They are mentioned here only for reasons of completeness. These commands have no mnemonics, as they can not be used in TMCL programs.
They are to be used only by the TMCL IDE to communicate with the module, for example to download a TMCL application into the module. The only control commands that could be useful for a user host application is "get firmware revision" (command 136, please note the special reply format of this command, described at the end of this section) and 129 (run application).
All other functions can be achieved by using the appropriate functions of the TMCL IDE.

Instruction	Description	Type	Mot/Bank	Value
128 – stop application	a running TMCL standalone application is stopped	(don't care)	(don't care)	(don't care)
129 – run application	TMCL execution is started (or continued)	0 - run from current address 1 - run from specified address	(don't care)	(don't care) starting address
130 – step application	only the next command of a TMCL application is executed	(don't care)	(don't care)	(don't care)
131 – reset application	the program counter is set to zero, and the standalone application is stopped (when running or stepped)	(don't care)	(don't care)	(don't care)
132 – start download mode	target command execution is stopped and all following commands are transferred to the TMCL memory	(don't care)	(don't care)	starting address of the application
133 – quit download mode	target command execution is resumed	(don't care)	(don't care)	(don't care)
134 – read TMCL memory	the specified program memory location is read	(don't care)	(don't care)	<memory address>
135 – get application status	one of these values is returned: 0 – stop 1 –run 2 – step 3 - reset	(don't care)	(don't care)	(don't care)
136 – get firmware version	return the module type and firmware revision either as a string or in binary format	0 – string 1 – binary	(don't care)	(don't care)
137 – restore factory settings	reset all settings stored in the EEPROM to their factory defaults This command does not send back a reply.	(don't care)	(don't care)	must be 1234
138 – reserved				
139 – enter ASCII mode	Enter ASCII command line (see chapter 2.5)	(don't care)	(don't care)	(don't care)

Special reply format of command 136:

- Type set to 0: reply as a string:

Byte index	Contents
1	Host Address
2..9	Version string (8 characters, e.g. "303V2.48"

There is no checksum in this reply format! To get also the last byte when using the CAN bus interface, just send this command in an eight byte frame instead of a seven byte frame. Then, eight bytes will be sent back, so you will get all characters of the version string.

- Type set to 1: version number in binary format: Here, the normal reply format is used. The version number is output in the "value" field of the reply in the following way:

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

4 Axis Parameters

The following sections describe all axis parameters that can be used with the SAP, GAP, AAP, STAP and RSAP commands. Please note that some parameters are different with different module types. The letters under "access" mean: R = readable (GAP), W = writable (SAP), E = automatically restored from EEPROM after reset or power-on.

4.1 Basic axis parameters (all TMCL stepper motor modules except the TMCM-100 module and the Monopack 2)

The parameters described in the following table are those which are needed very often. Please note that the **TMCM-100** module uses a different parameter set (see chapter 4.3), but all other TMCL stepper motor modules use these parameters.

Number	Axis Parameter	Description	Range	Access
0	target (next) position	The desired position in position mode (see ramp mode, no. 138).	$\pm 2^{23}$	RW
1	actual position	The current position of the motor. Should only be overwritten for reference point setting.	$\pm 2^{23}$	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	RW
3	actual speed	The current rotation speed. Should never be overwritten.	±2047	R
4	maximum positioning speed	Should not exceed the physically highest possible value. Adjust the pulse divisor (no. 154), if the speed value is very low (<50) or above the upper limit. See TMC 428 datasheet (p.24) for calculation of physical units.	0…2047	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 TMC 428 datasheet (p.24) for calculation of physical units.	0… 2047	RWE
6	absolute max. current	The most important motor setting, since too high values might cause motor damage! Note that on the TMCM-300 the phase current can not be reduced down to zero due to the Allegro A3972 driver hardware. On the TMCM-300, 303, 310, 110, 610, 611 and 612 the maximum value is 1500 (which means 1.5A). On all other modules the maximum value is 255 (which means 100% of the maximum current of the module).	0…1500 resp. 0..255	RWE
7	standby current	The current limit two seconds after the motor has stopped. The value range of this parameter is the same as with parameter 6.	0..1500 resp. 0..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 TMC 428 data sheet for the different switch modes. Default is two switch mode: 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

Number	Axis Parameter	Description	Range	Access
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
14	steprate prescaler	Currently not used, see parameters no. 153 and 154		

4.2 Advanced axis parameters (all TMCL stepper motor modules except the TMCM-100)

These parameters are only needed if the desired needs can not be met by setting the basic parameters listed in section 4.1. Some of these parameters influence the TMC428 directly so that advanced understanding of the TMC428 chip is needed. There are even some parameters that should only be changed if recommended by TRINAMIC. Please note that these paramters are not available on the TMCM-100 module.

Number	Parameter	Description	Range	Access
130	minimum speed	Should always be set 1 to ensure exact reaching of the target position. Do not change!	0… 2047	RWE
135	actual acceleration	The current acceleration (read only).	0… 2047	R
136	acceleration threshold	specifies the threshold between low and high acceleration values for the parameters 144&145. **Normally not needed.**	0… 2047	RWE
137	acceleration divisor	A ramping parameter, can be adjusted in special cases, automatically calculated by setting the maximum acceleration (e.g. during normal initialisation). See the TMC428 data sheet for details. **Normally no need to change.**	0…13	RWE
138	ramp mode	In version 2.16 and later: 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
139	interrupt flags	Must not be modified. See the TMC 428 datasheet for details.	16bits	RW

Number	Parameter	Description	Range	Access
140	microstep resolution	0 – full step *) 1 – half step *) 2 – 4 microsteps 3 – 8 microsteps 4 – 16 microsteps 5 - 32 microsteps 6 – 64 microsteps Note that modifying this parameter will affect the rotation speed in the same relation. Note that modifying this parameter will affect the rotation speed in the same relation. Even if the module is specified for 16 microsteps only, switching to 32 or 64 microsteps still brings an enhancement in resolution and smoothness. The position counter will use the full resolution, but, however, the motor will resolve a maximum of 24 different microsteps only for the 32 or 64 microstep units. *) Please note that the fullstep setting as well as the half step setting are not optimized for use without an adapted microstepping table. These settings just step through the microstep table in steps of 64 respectively 32. To get real full stepping use axis parameter 211 or load an adapted microstepping table.	0…6	RWE
141	ref. switch tolerance	For three-switch mode: a position range, where an additional switch (connected to the REFL input) won't cause motor stop. See section 6.1 for details.	0…4095	RW
142	snapshot position	For referencing purposes, the exact position at hitting of the reference switch can be captured in this parameter. A dummy value has to be written first to prepare caption.	$\pm 2^{23}$	RW
143	max. current at rest	In contrast to the standby current, this current limit becomes immediately active when the motor speed reaches zero. The value represents a fraction of the absolute maximum current: 0 – no change of current at rest (default, 100%) 1..7 – 12.5%..87.5% See the TMC428 datasheet for details. **Normally not used, use parameters 6 and 7 instead!**	0…7	RWE
144	max. current at low accel.	An optional current reduction factor, see parameters 136 and 143 for details. **Normally not used, use parameters 6 and 7 instead!**	0…7	RWE
145	max. current at high accel.	An optional current reduction factor, see parameters 136 and 143 for details. **Normally not used, use parameters 6 and 7 instead!**	0…7	RWE
146	acceleration factor	A ramping parameter, can be adjusted in special cases, automatically calculated by setting the maximum acceleration (e.g. during normal initialisation). See the TMC428 data sheet for details. **Normally no need to change.**	0…128	RWE
147	ref. switch disable flag	If set, the reference switch (left switch) won't cause the motor to stop. See parameters 12 and 13.	0/1	RWE
148	limit switch disable flag	If set, the limit switch (right switch) won't cause the motor to stop. See parameters 12 and 13.	0/1	RWE
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

Number	Parameter	Description	Range	Access
150	(reserved)	**Do not change!**	0/1	R
151	position latch flag	Indicates that a position snapshot has been completed (see parameter 142).	0/1	R
152	interrupt mask	Must not be modified. See the TMC 428 datasheet for details.	(16bits)	R
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	referencing mode	1 – Only the left reference switch is searched. 2 – The right switch is searched, then the left switch is searched. 3 – Three-switch-mode: the right switch is searched first, then the reference switch will be searched. Please see chapter 6.1 for details on reference search.	1/2/3	RWE
194	referencing search speed	For the reference search this value specifies the search speed as a fraction of the maximum velocity: 0 – full speed 1 – half of the maximum speed 2 – a quarter of the maximum speed 3 – 1/8 of the maximum speed (etc.) On the TMCM-34x modules the speed is given directly as a value between 0..2047.	0…8 M-34x: 0..2047	RWE
195	referencing switch speed	Similar to parameter no. 194, the speed for the switching point calibration can be selected. On the TMCM-34x modules the speed is given directly as a value between 0..2047.	0..8 M-34x: 0..2047	RWE
196	(reserved)	**Do not change!**		
197	(reserved)	Not used	0…2^{15}	RWE
198	driver off time	A special adjustment of the motor driver A3972 **(TMCM-300 only)**. Low values may cause more mechanical vibrations, while the higher ones lead to acoustic noise of the drivers. The default value of 20 is a good compromise for most applications. See the Allegro A3972 datasheet for details.	0…31	RWE
200	fast decay time	A special adjustment of the motor driver A3972 **(TMCM-300 only)**, with less influence than the driver off time (no. 198) in most cases. Low values generally reduce driver noise. See the Allegro A3972 datasheet for details.	0…15	RWE
203	mixed decay threshold	If the actual velocity is above this threshold, mixed decay will be used **(all modules except the TMCM-300)**. Since V3.13, this can also be set to –1 which turns 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 (TMCM-301 / 303 / 310 / 11x and 61x only).	0..65535 0 = never	RWE
205	Stall Detection Threshold	Stall detection threshold. Only usable on modules equipped with TMC246 or TMC249 motor drivers. Set it to 0 for no stall detection or to a value between 1 (low threshold) and 7 (high threshold). The motor will be stopped if the load value exceeds the stall detection threshold. Switch off mixed decay to get usable results.	0..7	RWE

Number	Parameter	Description	Range	Access
206	Actual Load Value	Readout of the actual load value used for stall detection. Only usable on modules equipped with TMC246 or TMC249 motor drivers. On other modules this value is undefined.	0..7	R
208	Driver Error Flags	TMC236 Error Flags		R
209	Encoder Position (V3.21 or higher)	The value of an encoder register of a TMCM-323 module connected to a TMCM-30x module can be read or written. Please see the TMCM-323 manual for details.		RW
210	Encoder Pre-scaler (V3.21 or higher)	Pre-scaler for an encoder connected to a TMCM-323 module. Please see the TMCM-323 manual for details. This value can not be read back! This setting is also available on the TMCM-611, TMCM-101 and TMCM-102 modules. Please see the manuals of these modules for the meaning of these parameter on a specific module.		W
211	Fullstep Threshold (V3.26 or higher)	When exceeding this speed the driver will switch to real full step mode. To disable this feature set this parameter to zero or to a value greater than 2047. 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. Only on TMCM-101, TMCM-102 and TMCM-611 modules.	0..65535	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. Only on TMCM-610, TMCM-611, TMCM-612 and TMCM-34x modules.	0..255	RW

4.3 Axis parameters on the TMCM-100 and on the Monopack 2

The axis parameters on the TMCM-100 module and on the Monopack 2 with TMCL differ from those on the other modules. There are no "advanced" axis parameters on the TMCM-100 module.

Number	Parameter	Description	Range	Access
0	Target position	Read: The target position of a currently executed ramp. Write: Same function as a MVP ABS command.	-8388608.. +8388607	RW
1	Actual position	Read: The actual position of the motor. Write: Change the position and encoder counter without moving the motor.	-8388608.. +8388607	RW
2	Target velocity	Write: value >0: same function as ROR value <0: same function as ROL value =0: same function as MST Read: not possible	-8191.. +8191	W
3	Actual velocity	The actual velocity of the motor. Write access not possible.	-8191.. +8191	R
4	Max. positioning velocity	The maximum velocity used when executing a ramp to a position. Do not set to zero!	1..8191	RWE
5	Max. acceleration	The maximum acceleration used to accelerate or	1..8191	RWE

Number	Parameter	Description	Range	Access
		decelerate the motor. Do not set to zero!		
6	Current at constant rotation	Maximum current when moving with constant velocity (255 => 100%).	0 .. 255	RWE
7	Current at standby	Current when the motor is standing (255 => 100%).	0..255	RWE
8	Position reached flag	Reads 1 when the actual position equals the target position. Write access not possible.	0/1	R
9	Reference switch status	The logical state of the reference switch (connected to the SYNC_IN pin).	0/1	R
10	Right stop switch status	The logical state of the right stop switch. Write access not possible.	0/1	R
11	Left stop switch status	The logical state of the left stop switch. Write access not possible.	0/1	R
12	Stop switch disable	Deactivates the function of both stop switches when set to 1.	0/1	RWE
13	Stop switch disable	Same function as parameter #12.	0/1	RWE
14	Step rate prescaler	Prescaler for the step rate, determines the maximum step frequency.	0..15	RWE
15	Bow	The bow parameter of the ramp function. Do not set to zero!	1..8191	RWE
16	Microstep resolution	The number of microsteps to be used. 1..64: 1..64 microsteps 65: 100 microsteps 66: 202 microsteps 67: 406 microsteps	1..67	RWE
17	Microstep waveform	The microstep waveform to be used.	-127..+127	RWE
18	Step/direction mode	Activates the step/direction input when set to 1 (the module then works as a step / direction sequencer).	0/1	RWE
19	Step pulse length	The length of the step pulses at the step / direction output.	0..3	RWE
20	Phases	Set to 2 for two phase motors (default) or to 3 for 3 phase motors (currently not usable).	2/3	RWE
21	Current at acceleration	Current when the motor is accelerating or decelerating (255 => 100%).	0..255	RWE
22	Reference search mode (s. section 6.2)	0: Separate stop and reference switches. 1: Same switch for stop and reference point. 2: Circular mode: Only one reference switch, search the switch from both sides.	0/1/2	RWE
23	Reference search velocity	Velocity used for reference searching. If the velocity is negative the right switch will be searched instead of the left switch.	-8191.. +8191	RWE
24	Stop Switch Deceleration	Deceleration when touching a stop switch. A value of 0 means a hard stop.	0..8191	RWE
25	Encoder Position	Actual position of the encoder (read only).		R
26	Encoder Configuration	Bit 0: polarity of the encoder N signal. Bit 1: the next N signal clears the encoder position counter. Bit 4: set this bit to copy the actual encoder value to the position register (bit resets automatically). Bit 6: direction of the encoder signals (1: A->B, 0: B->A).		RWE
27	Encoder Predivider	Predivider for the incremental encoder.	0..255	RWE

Number	Parameter	Description	Range	Access
28	Encoder Multiplier	Multiplier for the incremental encoder.	1..255	RWE
29	Maximum deviation	Maximum deviation between motor and incremental encoder.	0..65535	RWE
30	Deviation action	0: Disabled 1: Alarm, but no stop 2: Soft Stop 3: Hard Stop	0/1/2/3	RWE
31	Correction delay	Automatic correction after deviation alarm: 0: Disabled. >0: 1/100s until correction starts.	0..65535	RWE
32	Correction retries	Number of retries when automatic position correction is done. 0: Disabled, >0: number of retries.	0..255	RWE
33	Correction tolerance	Tolerance around the target position.	0..65535	RWE
34	Correction velocity	Velocity used for automatic position correction.	1..8191	RWE

5 Global Parameters

The global parameters apply for all types of TMCM modules. They are grouped into 3 banks: bank 0 (global configuration of the module), bank 1 (user C variables) and bank 2 (user TMCL variables). The letters under "access" mean: R = readable (GGP), W = writeable (SGP), E = automatically restored from EEPROM after reset or power-on. Use SGP and GGP (see sections 3.9 and 3.10) commands to write and read global parameters.

5.1 Bank 0

These parameters affect the global configuration of the module. The parameters 0..38 exist only on TMCM-300 / 301 / 302 / 303 / 310 / 610 modules and must normally not be set by the user. They should never be changed on TMCM-300, TMCM-302, TMCM-303, TMCM-310 and TMCM-610 modules. On TMCM-301 modules these parameters can be changed to adapt the module to specific motor drivers. It is best to set these parameters by using the appropriate functions of the TMCL DIE and not by entering many SGP commands (the TMCL IDE does this automatically). The parameters 0..38 are only mentioned here in short form, for completeness. They are not available on TMCM-100 modules.

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 SMGP register
7..22	Driver chain configuration long words 0..15
23..38	Microstep table long word 0..15

An STGP 23, 0 command will store the entire microstep table, and an STGP 7, 0 command will store the entire driver chain configuration table. **Use the appropriate functions of the TMCL IDE to change these parameters interactively, if really necessary! Take extreme care when doing this, as wrong configurations here may cause damage to the motor drivers! The TMCM-301 modules is the only device where changes may be necessary (when using it with other motor drivers than the TMC236/TMC239 chips).**

The following parameters with the numbers from 64 on configure things like the serial address of the module RS232 / RS485 baud rate or 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, so that an SGP command on such a parameter will always store it permanently (no extra STGP command needed).
Take care when changing these parameters, and use the appropriate functions of the TMCL IDE to do it in an interactive way!

Number	Global parameter	Description	Range	Access
64	EEPROM magic	Setting this parameter to a different value as $E4 will cause re-initialisation 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

Number	Global parameter	Description	Range	Access
65	RS232 and 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 **Caution:** Not supported by Windows! 7 – 115200 baud **Caution:** 115200 does not work with most host PCs, as the baud rate error on the modules is too high with this baud rate (-3.5% baud rate error).	0…7	RWE
66	Serial address	The module (target) address for RS-232 and RS-485.	0…255	RWE
67	ASCII mode (available since version 3.21)	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	Reserved	(currently not used, do not change!)		RWE
69	CAN bit rate	1 – 10kBit/s 2 – 20kBit/s 3 – 50kBit/s 4 – 100kBit/s 5 – 125kBit/s 6 – 250kBit/s (default) 7 – 500kBit/s 8 – 1000kBit/s (not supported by TMCM-30x/110/111/112)	1..7	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
72	System error mask	(currently not used, do not change!)		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
74	Encoder interface (available since version 3.21)	Determines if a TMCM-323 is connected to the external SPI interface and to which SPI_SEL line it is connected: 0 – No TMCM-323 connected 1 – Connected to SPI_SEL0 2 – Connected to SPI_SEL1 3 – Connected to SPI_SEL2 Please see TMCM-323 manual for details!		RWE
75	Telegram pause time	Pause time before the reply via RS232 or RS485 will be 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 or IIC 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
78	Poll interval	(currently not used, do not change!)	---	---
79	Port function mask	(currently not used, do not change!)	---	---

Number	Global parameter	Description	Range	Access
80	Shutdown pin functionality	Select the functionality of the SHUTDOWN pin (not with TMCM-300). 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 Note: When a user tries to switch off the protection against disassembling, the program will be erased first! So, when changing this value from 1 or 3 to 0 or 2, the TMCL program will be erased.	0,1,2,3	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
131	Application error flags	(currently not used)	---	---
132	Tick timer	A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value.		RW

5.2 Bank 1

The global parameter bank 1 contains a set of variables that are mainly intended for use in customer specific extensions to the firmware. So, together with the user definable commands (see section 3.32) these variables form the interface between extensions to the firmware (written in C) and a TMCL application. Although they could also be used as general purpose variables in TMCL programs, it is much better to use the variables in bank 2 (see section 5.3) for this purpose.
These parameters are not available on the TMCM-34x modules.

Number	Global parameter	Description	Range	Access
0	C application state	The main state machine variable of the example user C code.	0…255	RW
1	(not used)		0…255	RW
2	C application state timer	A universal timer, supposed for state timing purposes.	0…255	RW
3	C application general purpose variable "unsigned char #0"		0…255	RWE
4	C application general purpose variable "unsigned char #1"		0…255	RWE
5	C application general purpose variable "unsigned char #2"		0…255	RWE
6	C application general purpose variable "unsigned int #0"		0…2^{16}	RWE
7	C application general purpose variable "unsigned int #1"		0…2^{16}	RWE
8	C application general purpose variable "unsigned int #2"		0…2^{16}	RWE
9	C application general purpose variable "signed long #0"		-2^{31}…$+2^{31}$	RWE
10	C application general purpose variable "signed long #1"		-2^{31}…$+2^{31}$	RWE
11	C application general purpose variable "signed long #2"		-2^{31}…$+2^{31}$	RWE

5.3 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. By default, 20 variables are available, but the number may be increased in future firmware modification versions. From firmware version 3.17 on, 56 user variables are available.

Number	Global parameter	Description	Range	Access
0	General purpose variable #0	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
1	General purpose variable #1	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
2	General purpose variable #2	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
3	General purpose variable #3	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
4	General purpose variable #4	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
5	General purpose variable #5	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
6	General purpose variable #6	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
7	General purpose variable #7	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
8	General purpose variable #8	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
9	General purpose variable #9	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
10	General purpose variable #10	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
11	General purpose variable #11	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
12	General purpose variable #12	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
13	General purpose variable #13	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
14	General purpose variable #14	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
15	General purpose variable #15	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
16	General purpose variable #16	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
17	General purpose variable #17	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
18	General purpose variable #18	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
19	General purpose variable #19	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE
20..55	General purpose variables #20..#55 (TMCL Version 3.17 or higher)	for use in TMCL applications	$-2^{31}...+2^{31}$	RWE

6 Hints and Tips

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

6.1 Reference search with TMCM-3xx / 10x / 11x / 61x modules

The built-in reference search features switching point calibration and support of one or two reference switches. The internal operation is based on three individual state machines (one per axis) 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.

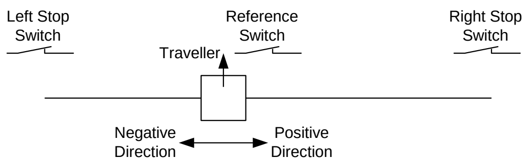


Figure 6.1: Definition of the switches

- 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 Figure 6.2 the connection of the left and the right limit switch is shown. Figure 6.3 shows the connection of three switches as left and right limit switch and a reference switch for the reference point. 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.

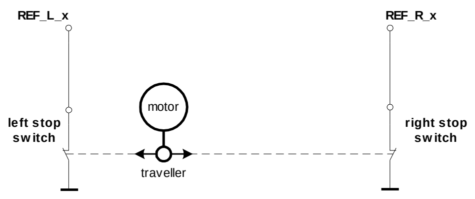


Figure 6.2: Two limit switches

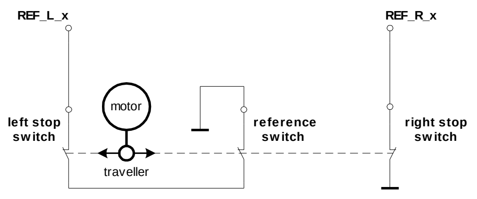


Figure 6.3: Limit switches with extra reference switch

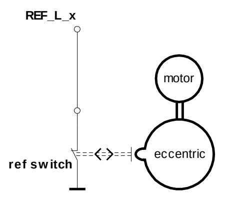


Figure 6.4: Circular system

6.2 Reference search with TMCM-100 modules

The behaviour of the reference search depends on the reference search mode setting (axis parameter #22, section 4.3) and is as follows:

- Mode 0: Linear mode, reference switch is also end switch: A move into the reference switch and then out of the reference switch is executed. The zero position is then set to the beginning of the switch.
- Mode 1: Linear mode, separate reference switch and end switch: First, the left or the right stop switch (specified by the sign of the reference search velocity; value>0: left switch, value<0: right switch) is searched. After that, the reference switch is searched at first from one, then from the other side. The zero position is then set to the middle of the reference switch.

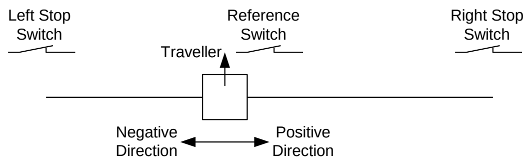


Figure 6.5: A linear drive (use mode 1)

- Mode 2: Circular mode: The reference switch (connected to the reference switch input) is searched at first from one and then form the other side. The zero position is then set to the middle of the reference switch. There are no end points.

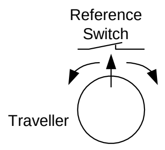


Figure 6.6: A circular drive (use mode 2)

The velocity of the reference search is specified by axis parameter #23 (section 4.3). Start the reference search with a RFS START command. The reference search can be aborted by a RFS STOP command. To query if the reference search is still running in direct mode, use a RFS STATUS command. In a stand-alone TMCL program, use WAIT RFS to wait until a reference search has finished.
The reference switch always has to be connected to the "SYNC IN" pin of the module.

6.3 Using an incremental encoder with TMCM-100 modules

Using an incremental encoder allows exact position control, as it feeds back the steps of the motor into the module. The module can then detect deviations and can also try to correct such deviations automatically. The deviation detection and the automatic position correction after a deviation has occurred are easy to use features that automatically detect and correct any errors during a movement.

6.3.1 Setting the resolution

The resolution of the encoder (pulses per revolution) must match the resolution of the motor to make deviation detection and automatic position correction function correctly. If the resolutions do not match, they can be adapted by changing one or more of the following parameters using GAP commands:

- Microstep resolution (axis parameter #16, section 4.3): This parameter changes the resolution of the motor.
- Encoder pre-divider (axis parameter #27, section 4.3): This can be used if the resolution of the encoder is higher than the resolution of the motor. When for example the pre-divider is set to 4, only every 4th pulse of the encoder will be used to increment or decrement the encoder counter register.

- Encoder multiplier (axis parameter #28, section 4.3): This can be used if the resolution of the encoder is lower than the resolution of the motor. When for example the multiplier is set to 5, every encoder pulse will increment or decrement the encoder counter register by 5.

To find the right combination of the parameters you can just let the motor run for example 10000 steps (using an MVP command in direct mode, with deviation detection and automatic position correction switched off) and then watch the encoder counter register (using GAP 25, 0). Before doing that, use a SAP 1,0,0 command to set all position registers to zero.

6.3.2 Deviation detection

This function can be configured using axis parameters #29, #30 and #31. An error flag (EDV, see also section 3.18 and 3.31) will be set when the maximum deviation between motor and encoder is exceeded. Optionally the motor can also be stopped then. Set the maximum deviation using parameter #29. The motor can also be stopped immediately or softly when in case of deviation (parameter #30). Furthermore, the automatic position correction (see) can be started n/10 sec (n=1..65535, parameter #31) after a deviation has been detected.

6.3.3 Position correction

Automatic position correction can be done at the end of each ramp or when a deviation has been detected. Automatic position correction can only be used in conjunction with an incremental encoder which has to be configured correctly first.
When this function is turned on (by setting axis parameter #32 to a value greater than zero), the module checks if the position counter of the incremental encoder matches the desired end position at the end of every ramp (a tolerance window around the end position can be specified by axis parameter #33). If this is not the case, the module will try to correct the position of the motor using the velocity specified by axis parameter #34. The maximum number of retries after each ramp can also be configured by setting axis parameter #32. The EPO flag (see also section 3.18 and 3.31) will be set and the position correction will be aborted if this number is exceeded.

6.4 Stall Detection (TMCL Version 3.06 or higher)

The modules TMCM-303, TMCM-310 and TMCM-610 can be equipped with TMC246 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 also be used with a TMCM-301 module together with a TMCM-035 module that is equipped with a TMC249 chip.
Stall detection can also be used for finding the reference point. You can do this by using 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.

6.5 Fixing microstep errors (TMCL V3.13 or higher)

Due to the "zero crossing problem" of the TMC236/TMC246 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 (waveform table, see sections 7.6.2.4 and 7.6.2.5 on how to do that).
Use SAP 203, <m>, -1 to turn on this feature (where <m> stands for the motor number).

6.6 Using the RS485 interface

For using the modules with RS485 interface they must be equipped with at least TMCL version 3.16 (except for TMCM-110 modules).
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.

7 The TMCL IDE

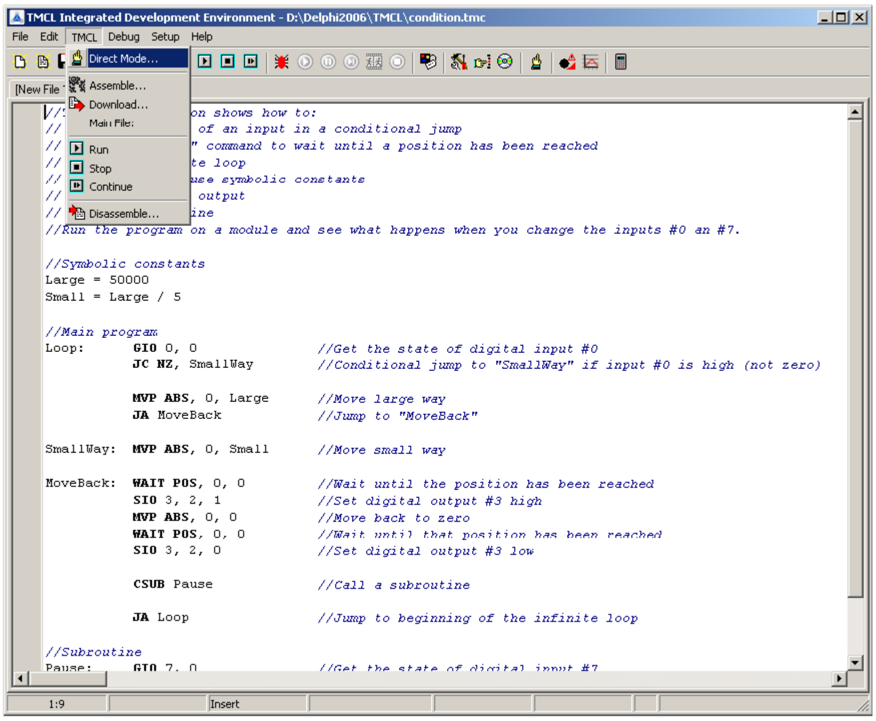
The TMCL IDE is an integrated development environment mainly for developing stand-alone TMCL applications, but it also includes a function for using the TMCL commands in direct mode. The TMCL IDE is a PC application running under Windows 95/98/NT/2000/XP (Windows 3.x is not supported) that includes

- a text editor for writing and modifying TMCL programs,
- a TMCL assembler for translating TMCL programs from mnemonic to binary format and downloading them into a module,
- a TMCL disassembler for getting a TMCL application out of a module and translating it back to mnemonic format,
- a dialogue which allows setting the configuration of a module in an easy, interactive way,
- a dialogue for entering and executing TMCL commands in direct mode,
- a function for updating the firmware of a module.

and many more powerful features.
Please be sure to always use the latest version of the TMCL IDE a its functionality is being extended and improved constantly.

7.1 Installation

To install the TMCL IDE to your computer, just copy the file "TMCL.EXE" to your hard disk. Then, just double click the file to start the program.



```
//                 on shows how to:
//                  of an input in a conditional jump
//                 " command to wait until a position has been reached
//                 te loop
//                 use symbolic constants
//                  output
//                 ine
//Run the program on a module and see what happens when you change the inputs #0 an #7.

//Symbolic constants
Large = 50000
Small = Large / 5

//Main program
Loop:      GIO 0, 0              //Get the state of digital input #0
           JC NZ, SmallWay       //Conditional jump to "SmallWay" if input #0 is high (not zero)

           MVP ABS, 0, Large     //Move large way
           JA MoveBack           //Jump to "MoveBack"

SmallWay:  MVP ABS, 0, Small     //Move small way

MoveBack:  WAIT POS, 0, 0        //Wait until the position has been reached
           SIO 3, 2, 1           //Set digital output #3 high
           MVP ABS, 0, 0         //Move back to zero
           WAIT POS, 0, 0        //Wait until that position has been reached
           SIO 3, 2, 0           //Set digital output #3 low

           CSUB Pause            //Call a subroutine

           JA Loop               //Jump to beginning of the infinite loop

//Subroutine
Pause:     GIO 7, 0              //Get the state of digital input #7
```

Fig. 7.1: The TMCL IDE main window

After the first start of the program you should select the "Options..." function from the "Setup" menu (see section 7.6.1.2) and set up the COM port where your TMCM module is connected to. You should not need to change any other settings.

7.2 Getting started

First, try out the TMCL commands in direct mode. Use the "Direct Mode" function for this purpose (see section 7.5.2). After you have successfully tried out the direct mode, have a look at the TMCL sample programs supplied on the software CD. Load a program, assemble it, download it into a module and run it. Try to understand the TMCL sample programs. After that you are ready to write your own TMCL programs. Read the chapters 2, 3 and 8 to learn more about the TMCL programming language.

7.3 The integrated editor

The text editor of the TMCL-IDE mainly provides the functionality of a standard Windows text editor. An additional function to the Windows standard is Ctrl-Y to delete a line. Some functions of the editor can be found in the "Edit" menu. It is also possible to edit multiple documents. They are then shown in a workbook style.

7.4 The "File" menu

The file menu provides functions to load and save files. Some functions can also be found in the tool bar below the menu bar.

7.4.1 New

This function opens a new editor page, so a new file can be created.

7.4.2 Open

After selecting the "Open" function a file selection dialog will be shown. Here you can select a file to be opened. Then, a new editor page opens and the selected file will be loaded into that editor page.

7.4.3 Save, Save as

Select one of these functions to save the contents of the currently selected editor page into a file. The "Save as" function allows to save a file using a new name.

7.4.4 Save all

Select this function to save all files that are currently loaded into the editor and have been changed.

7.4.5 Close

The "Close" function closes the actual editor page. This function can also be selected from the context menu of the editor (click the right mouse button in the editor window to open the context menu).

7.4.6 Exit

Use this function to close the TMCL IDE. The same function can be achieved by closing the main window.

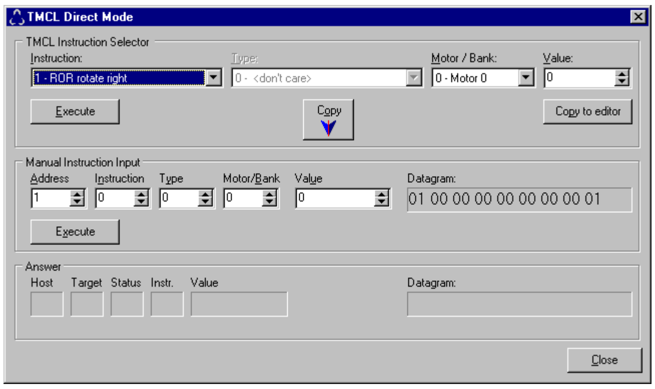
7.5 The "TMCL" menu

7.5.1 Basics

The "TMCL" menu contains all functions needed for assembling, downloading and disassembling TMCL programs. It also contains the functions to run and stop a TMCL program on a module and to use TMCL commands in direct mode. Assembling a TMCL program always takes place from the editor. So, a before assembling a TMCL file it must be loaded into the editor.

7.5.2 Direct mode

Select the "Direct Mode" function in the "TMCL" menu to open the "Direct Mode" dialog (Fig. 7.2). The TMCL IDE then first checks the type of the module that is connected (TMCM-3xx or TMCM-100), as some menus in the direct mode dialogue differ between TMCM-3xx or TMCM-100 modules. If the type of the module could not be detected (e.g. because there is no module connected), a little dialogue pops up where you are prompted to select your module type. Do this by clicking the appropriate button.
By using the direct mode you can send commands to a module that are executed immediately. In the "TMCL Instruction Selector" area you can select a command and its parameters. Click the "Execute" button in this area to send the command to the module. The response is then displayed in the "Answer" section. By clicking the "Copy to editor" button the command mnemonic will be copied to the TMCL editor.
You can also enter the instruction numbers directly in the "Manual Instruction Input" area an execute them by clicking the appropriate "Execute" button (but this option is needed only very seldom). The "Copy" button in the "TMCL Instruction Selector" area copies the instruction bytes to the "Manual Instruction Input" area.


TMCL Direct Mode

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

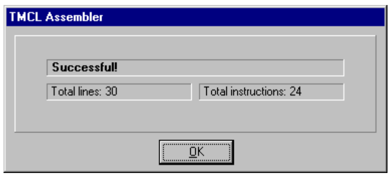
Manual Instruction Input
Address: 1
Instruction: 0
Type: 0
Motor/Bank: 0
Value: 0
Datagram: 01 00 00 00 00 00 00 00 01

Answer
Host / Target / Status / Instr. / Value
Datagram:

Fig. 7.2: The Direct Mode dialog

7.5.3 Assemble a TMCL program

Selecting the "Assemble" function in the "TMCL" menu assembles a file. If more than one files are open and no main file has been defined the currently selected file will be assembled. If a main file has been selected (see section 7.5.5) the main file will always be assembled, regardless of the currently selected editor file.
The progress of the assembly is displayed.


TMCL Assembler

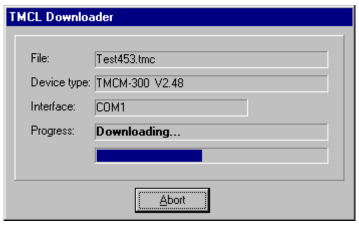
Successfull!
Total lines: 30
Total instructions: 24

Fig. 7.3: The progress of the assembly

If an error occurs, the line containing the error will be highlighted and the assembly will be aborted. The error message will be displayed in the assembler progress dialog.

7.5.4 Downloading the program

After a TMCL program has been successfully assembled, it can be downloaded to the module. Make sure that the module is connected to a COM port and that the port is selected in the "Options" dialogue. By selecting the "Download" function the program will then be downloaded into the module. The download progress is shown in a special window (Fig. 7.4). Downloading can be aborted by clicking the "Abort" button.

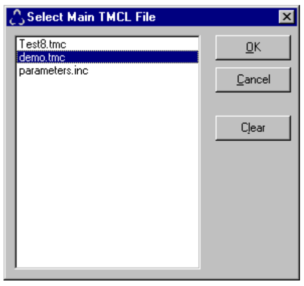

TMCL Downloader

File: Test453.tmc
Device type: TMCM-300 V2.48
Interface: COM1
Progress: Downloading...

Fig. 7.4: The program download progress

7.5.5 The "Main file" function

By using this function you can select a main file. This file is then always used when assembling, regardless of which file is currently selected in the editor. Click on the "Clear" button in the main file selection dialog to switch off this function and to always assemble the currently selected editor file again.


Select Main TMCL File

Test8.tmc
demo.tmc
parameters.inc

Fig. 7.5: The "Main file" dialog

7.5.6 The "Start" function

This functions starts the TMCL program which is currently in the TMCL memory of the module just by sending a reset and a run command to the module. The module then starts executing the program form the first command on.

7.5.7 The "Stop" function

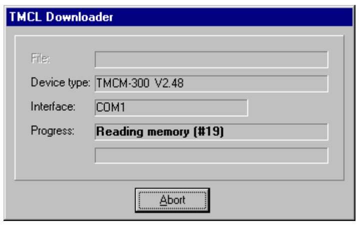
This functions sends a stop command to the module and so stops the execution of a TMCL program.

7.5.8 The "Continue" function

This function allows to continue the run of a TMCL program that has previously been stopped. It does this just by sending a run command to the module (without sending a reset command before). The module then continues executing the program from the next command before it had been stopped.

7.5.9 Disassembling a TMCL program

The "Disassemble" function reads out the TMCL memory of a module and disassembles its contents. The result is then written into a new editor page. So this function allows to check the program which is currently in the TMCL program memory of a module. The progress of downloading the program from the module and the disassembly is shown in special windows. It can be aborted by clicking the "Abort" button.



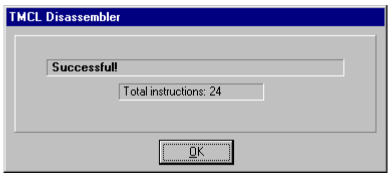
TMCL Downloader

File:

Device type: TMCM-300 V2.48

Interface: COM1

Progress: Reading memory (#19)



TMCL Disassembler

Successful!

Total instructions: 24

Fig. 7.6: Progress of disassembling a TMCL program

7.6 The "Setup" menu

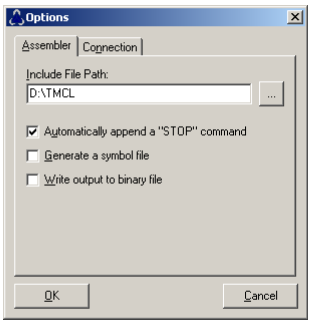
7.6.1 Options

This function provides the "Options" dialogue which allows to set up all global options of the TMCL IDE.

7.6.1.1 Assembler options

The "Assembler" page in the "Options" dialogue provides the assembler options. There are the following options:

- Include file path: The path where the assembler searches for include files included by the *#include* directive.
- Automatically append a "STOP" command: If this option is ticked, the assembler automatically appends a "STOP" command at the end of every TMCL program (if the last command in the program is not already a "STOP" command).
- Generate a symbol file: The assembler generates a text file that contains the address of every label. This can be useful to start the program from other addresses than 0.
- Write output to binary file: The assembler writes its output also to a binary file. For every command eight bytes are written (the command with checksum, but without a device address).



Options

Assembler | Connection

Include File Path: D:\TMCL

☑ Automatically append a "STOP" command

☐ Generate a symbol file

☐ Write output to binary file

Fig. 7.7: Assembler options

7.6.1.2 Connection options

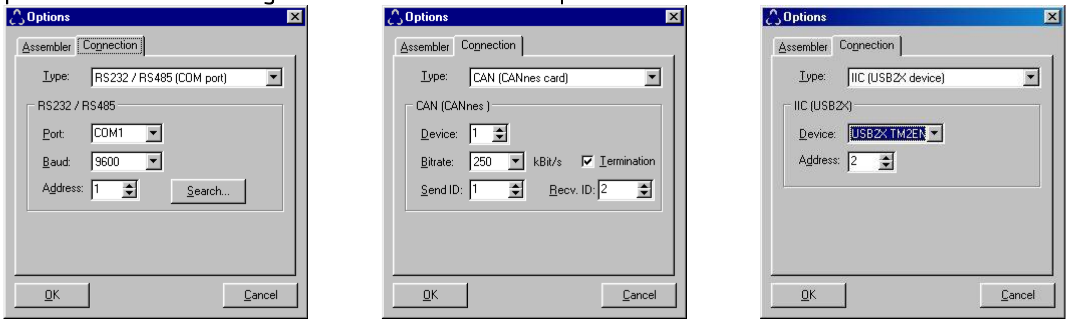
Here the interface type and the port can be selected that are to be used to communicate with a module. First, select the connection type. In most cases this will be "RS232 / RS485". If you have a Trinamic CANnes card or USB2X module and you would like to use the CAN or IIC interface of a module just select the appropriate connection type. The connection parameters that are displayed below the connection type change according to the selected interface.

When you have selected "RS232 / RS485" the COM port that is to be used, the baud rate and the module address can be set. The factory default baud rate on a TMCM module is 9600. The factory default module address is 1.

If you do not know the address of the connected module, just click the "Search" button. The TMCL IDE then tries to find the module address (please see section 7.6.3 for details).

N.B.: Versions prior to 1.18 also provided the option "XOR checksum". This has now been removed, and it must not be switched on in these older versions of the TMCL IDE.

Please note that RS485 adapters where the data direction is controlled by the RTS line of the COM port do not work with Windows 95, Windows 98 or Windows ME (due to a bug in these operating systems). They can only be used with Windows NT4, Windows 2000 or Windows XP, and also with these operating systems there may be the problem that switching back to "receive" takes place too late.



Options

Assembler | Connection

Type: RS232 / RS485 (COM port)

RS232 / RS485

Port: COM1

Baud: 9600

Address: 1

Search...

OK | Cancel

Options

Assembler | Connection

Type: CAN (CANnes card)

CAN (CANnes)

Device: 1

Bitrate: 250 kBit/s ☑ Termination

Send ID: 1 Recv. ID: 2

OK | Cancel

Options

Assembler | Connection

Type: IIC (USB2X device)

IIC (USB2X)

Device: USB2X TM2EN

Address: 2

OK | Cancel

Fig. 7.8: Some connection options

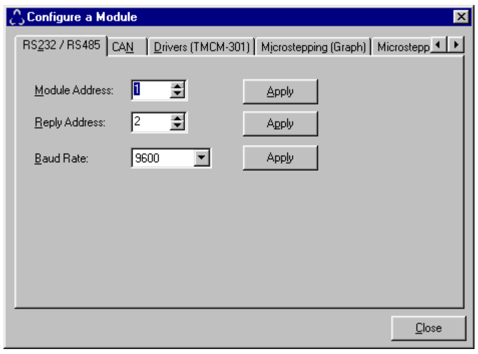
When the interface type "CANnes card" is selected, the CAN parameters are displayed and can be changed. "Device" selects which CANnes card to use (if you have more than one CANnes card installed). The factory default of the CAN bitrate on TMCM modules is normally 250kBit/s. "Termination" turns the termination resistor on the CANnes card on or off. "Send ID" is the CAN ID to use when sending CAN datagrams to the module. "Recv. ID" is the ID that the module uses to send back data to the PC.
Select "IIC (USB2X device)" when you wish to use the IIC interface of a Trinamic USB2X device. Here, only the device name of the USB2X device and the IIC address of the module can be set. The factory default on a new module is normally 2.
Select "CAN (USB2X device)" when you wish to use the CAN interface of a Trinamic USB2X device. The device name of the USB2X device and the CAN send ID, receive ID and bit rate can be set. Please note that the USB2X device must have firmware version 1.01 or higher to make use of this feature!
Select "RS485 (USB2X device)" when you wish to use the RS485 interface of a Trinamic USB2X device. The device name of the USB2X, the bit rate and the module address can be set. Please note that the USB2X device must have at least firmware version 2.05 to make use of this feature! This kind of RS485 interface is mainly useful for updating modules that only have RS485 interface (e.g. TMCM-110/RS485).
When a TMCM-610 module is connected via USB the option "USB direct" can be used. The TMCL-IDE can communicate with this module directly via USB. The device name of the TMCM-610 module can be selected here.

7.6.2 Configure

This function provides a dialogue for changing the configuration (global parameters) of a module.

7.6.2.1 RS232/RS485

Here, you can change the send address, the receive addresses and the baud rate of the serial interface of a module. Just set the desired value and then click on the appropriate "Apply" button to send the new setting to the module. Be careful, as most changes take effect immediately.



Configure a Module

RS232 / RS485 | CAN | Drivers (TMCM-301) | Microstepping (Graph) | Microstepp

Module Address: 1 Apply

Reply Address: 2 Apply

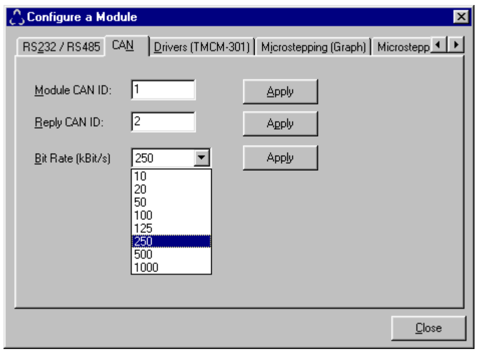
Baud Rate: 9600 Apply

Close

Fig. 7.9: RS232 / RS485 settings

7.6.2.2 CAN

This page provides the settings of the CAN interface of a module. The usage is the same as with the RS232/RS485 page. These settings do not have any effect on a TMCM-300 module as it does not have a CAN interface.



Module CAN ID: 1

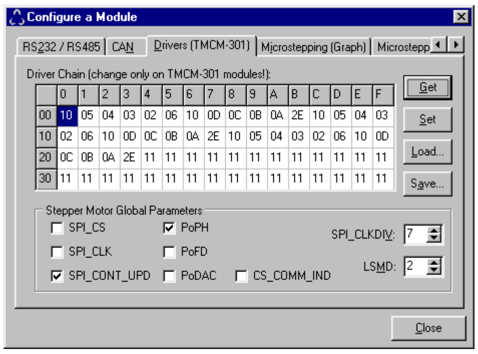
Reply CAN ID: 2

Bit Rate (kBit/s): 250

Fig. 7.10: CAN settings

7.6.2.3 Drivers

This page allows to change the settings for the SPI driver chain. See the TMC428 datasheet for details. These settings must not be changed on modules other than TMCM-301! On TMCM-301 modules they only need changing when motor drivers other than TMC-236 or TMC-239 are to be used.
When programming user specific driver configuration chain tables please disable the automatic mixed decay handling for TMC236 drivers, the automatic full step switching and the freewheeling function first (axis parameters 203, 204 and 211). This is needed because these functions modifie the driver configuration table.



Driver Chain (change only on TMCM-301 modules!):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	10	05	04	03	02	06	10	0D	0C	0B	0A	2E	10	05	04	03
10	02	06	10	0D	0C	0B	0A	2E	10	05	04	03	02	06	10	0D
20	0C	0B	0A	2E	11	11	11	11	11	11	11	11	11	11	11	11
30	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11

Stepper Motor Global Parameters

☐ SPI_CS ☑ PoPH SPI_CLKDIV: 7

☐ SPI_CLK ☐ PoFD

☑ SPI_CONT_UPD ☐ PoDAC ☐ CS_COMM_IND LSMD: 2

Fig. 7.11: The driver chain configuration

7.6.2.4 Microstepping (graphical view)

Here, the microstepping wave of the module can be changed to enhance the microstepping behaviour of a stepper motor. The microstepping wave function can be changed between a triangular wave, a sine wave or a trapezoid wave or something between that by changing the sigma value. Clicking the "Apply" button programs the new wave table into the module. Just let the motor run and try out the best value.
The microstepping wave does not have any effect on the TMCM-302 module as it only provides step/direction output.

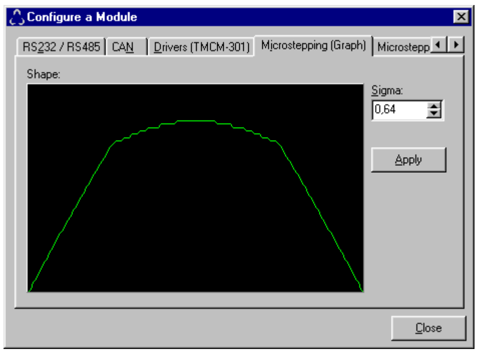
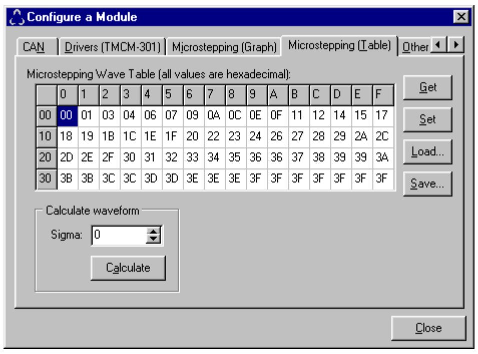


Fig. 7.12: Microstep wave form setting (graphical view)

7.6.2.5 Microstepping (table view)

The table view allows a finer adjusting of the microstep table. Clicking the "Get" button reads the actual microstep table from the module. You can then edit the values. All values are hexadecimal numbers between 0 and 3F. The "Set" button programs the table that can be seen on this screen into the module. Use the "Load" and "Save" buttons to load a table from a file or store a table into the file.
When clicking the "Calculate" button a waveform will be calculated in the same manner as in the graphical view of the microstepping function (section 7.6.2.4). This can be used as a basis of a table that can then be fine tuned manually.



Microstepping Wave Table (all values are hexadecimal):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	01	03	04	06	07	09	0A	0C	0E	0F	11	12	14	15	17
10	18	19	1B	1C	1E	1F	20	22	23	24	26	27	28	29	2A	2C
20	2D	2E	2F	30	31	32	33	34	35	36	36	37	38	39	39	3A
30	3B	3B	3C	3C	3D	3D	3E	3E	3E	3F	3F	3F	3F	3F	3F	3F

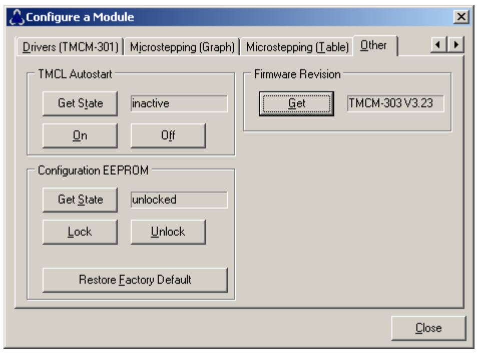
Calculate waveform
Sigma: 0
Calculate

Fig. 7.13: Microstep wave form setting (table view)

7.6.2.6 Other

This page provides the following functions:

- Firmware revision: Click the "Get" button in the "Firmware Revision" section to read the firmware revision of a module. The result will be displayed beside the "Get" button.
- Configuration EEPROM: Here you can lock or unlock the configuration EEPROM. When the configuration EEPROM is locked, STAP and STGP commands do not have any effect so that the settings stored in the configuration EEPROM can not be changed by accident. The buttons in the "Configuration EEPROM" section provide the following functionality:
  - Button "Get State": Click this button to see if the configuration EEPROM is locked. The state will then be displayed beside the button.
  - "Lock": Click this button to lock the configuration EEPROM.
  - "Unlock": Click here to unlock the configuration EEPROM.
  - "Restore Factory Default": Clicking this button will restore **all** settings stored in the configuration EEPROM to their factory defaults and then reset the module. Please use this function with extreme care. If this function should not work, try the "Clear EEPROM" function in the OS installation dialogue (see section 7.6.4).
- TMCL Autostart: The TMCL program auto start option can be turned on or off by clicking the appropriate button. Click the "Get state" button to see whether auto start is turned on or off on a module.



TMCL Autostart: inactive

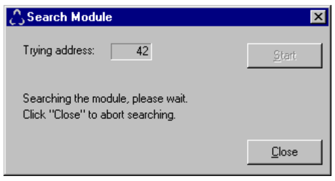
Firmware Revision: TMCM-303 V3.23

Configuration EEPROM: unlocked

Fig. 7.14: Other module settings

7.6.3 Search

If you do not know the RS232/RS485 address of a module you can use this function to find out: This function provides the address search dialogue. Here, just click the "Start" button. Then, every address (1..255) will be tried out.



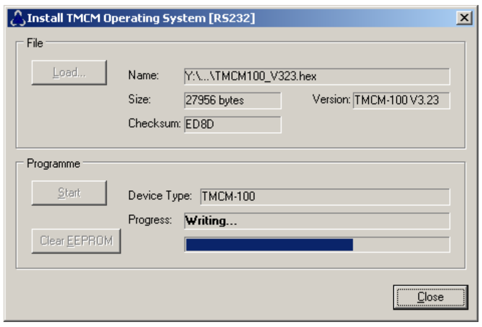
Trying address: 42

Searching the module, please wait.
Click "Close" to abort searching.

Fig. 7.15: Searching a module address

7.6.4 Install OS

By using this function you can update the firmware of a module. First, load the new firmware file (which must be in Intel-HEX format) by clicking the "Load" button. The file is then checked if it is a TMCL firmware file, and its device type and version number will be displayed. Then, click the "Start" button to program the new firmware into the module. Please make sure that there will be no power cut or cut of the serial connection during the programming process. The program checks if the device type in the firmware file and the device type of the module are identical. An error message will be displayed if this is should not be the case. If everything is okay, the new firmware will be programmed into the module and verified afterwards. The programming progress is shown by the status bar.



File

Name: Y:\...\TMCM100_V323.hex

Size: 27956 bytes

Version: TMCM-100 V3.23

Checksum: ED8D

Programme

Device Type: TMCM-100

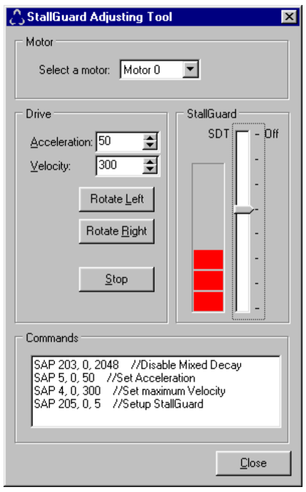
Progress: Writing...

Fig. 7.16: Firmware update in progress

If you click the "Clear EEPROM" button, all settings stored in the EEPROM of the module will be erased. If you then power the module off and on again, all settings will be restored to their factory defaults.
Attention: Before these functions can be used, TMCM-300 modules have to be prepared in the following way: Connect the pins 1 and 2 (the pins near the LED) with a jumper, then power it on. The module is now in boot

mode. After you have finished updating the firmware of such a module, power it off, remove the jumper and power it on again. With other modules (TMCM-301/302/303/310/100/110/610) there is no need to put on a jumper.
Please note that with most of the modules this function is only available when the RS232 interface of the module is used. With the TMCM-110(/SG)-CAN module the firmware can be updated via the CAN interface, and with the TMCM-610 module also the USB interface can be used to upgrade the firmware. The TMCM-103, TMCM-116 and TMCM-34x modules also support firmware upgrading via CAN (at 1000kBit/s).

7.6.5 StallGuard adjusting tool

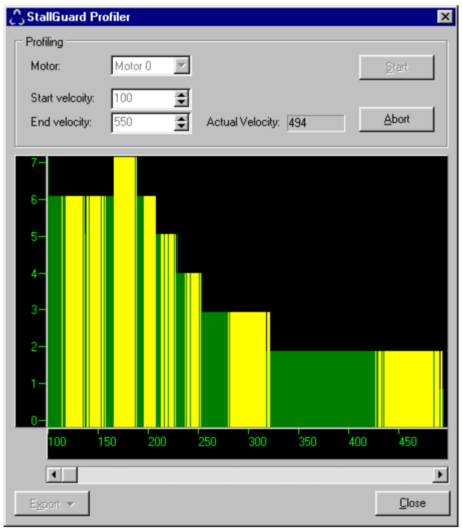


```
SAP 203, 0, 2048    //Disable Mixed Decay
SAP 5, 0, 50    //Set Acceleration
SAP 4, 0, 300    //Set maximum Velocity
SAP 205, 0, 5    //Setup StallGuard
```

The StallGuard adjusting tool helps to find the necessary motor parameters when StallGuard is to be used. This function can only be used when a module is connected that features StallGuard. This is checked when the StallGuard adjusting tool is selected in the "Setup" menu. After this has been successfully checked the StallGuard adjusting tool is displayed.
First, select the axis that is to be used in the "Motor" area.
Now you can enter a velocity and an acceleration value in the "Drive" area and then click "Rotate Left" or "Rotate Right". Clicking one of these button will send the necessary commands to the module so that the motor starts running. The red bar in the "StallGuard" area on the right side of the windows displays the actual load value. Use the slider to set the StallGuard threshold value. If the load value reaches this value the motor stops. Clicking the "Stop" button also stops the motor.
All commands necessary to set the values entered in this dialogue are displayed in the "Commands" area at the bottom of the window. There, they can be selected, copied and pasted into the TMCL editor.

7.6.6 StallGuard profiler

The StallGuard profiler is a utility that helps you find the best parameters for using stall detection. It scans through given velocities and shows which velocities are the best ones. Similar to the StallGuard adjusting tool it can only be used together with a module that supports StallGuard. This is checked right after the StallGuard profiler has been selected in the "Setup" menu. After this has been successfully checked the StallGuard profiler window will be shown.



First, select the axis that is to be used. Then, enter the "Start velocity" and the "End velocity". The start velocity is used at the beginning at the profile recording. The recording ends when the end velocity has been reached. Start velocity and end velocity must not be equal. After you have entered these parameters, click the "Start" button to start the StallGuard profile recording. Depending on the range between start and end velocity this can take several minutes, as the load value for every velocity value is measured ten times. The "Actual velocity" value shows the velocity that is currently being tested and so tells you the progress of the profile recording. You can also abort a profile recording by clicking the "Abort" button.
The result can also be exported to Excel or to a text file by using the "Export" button.

Fig. 7.17: The StallGuard Profiler

7.6.6.1 The result of the StallGuard profiler

The result is shown as a graphic in the StallGuard profiler window. After the profile recording has finished you can scroll through the profile graphic using the scroll bar below it. The scale on the vertical axis shows the load value:

a higher value means a higher load. The scale on the horizontal axis is the velocity scale. The colour of each line shows the standard deviation of the ten load values that have been measured for the velocity at that point. This is an indicator for the vibration of the motor at the given velocity. There are three colours used:

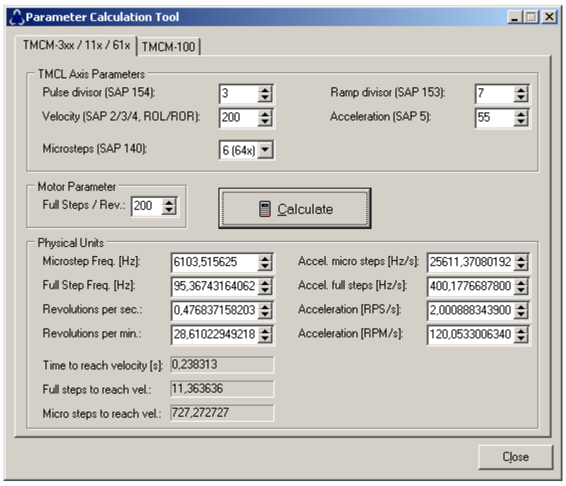
- Green: The standard deviation is very low or zero. This means that there is effectively no vibration at this velocity.
- Yellow: This colour means that there might be some low vibration at this velocity.
- Red: The red colour means that there is high vibration at that velocity.

7.6.6.2 Interpreting the result

In order to make effective use of the StallGuard feature you should choose a velocity where the load value is as low as possible and where the colour is green. The very best velocity values are those where the load value is zero (areas that do not show any green, yellow or red line). Velocities shown in yellow can also be used, but with care as they might cause problems (maybe the motor stops even if it is not stalled).
Velocities shown in red should not be chosen. Because of vibration the load value is often unpredictable and so not usable to produce good results when using stall detection.
As it is very seldom that exactly the same result is produced when recording a profile with the same parameters a second time, always two or more profiles should be recorded and compared against each other.

7.6.7 Parameter calculation tool

The parameter calculation tool helps you to calculate the velocity and acceleration parameters in TMCL. Parameters can be converted from physical units like rpm or rps into the internal units of TMCL and vice versa. There are actually two parameter calculation tools: One for the TMCM-3xx, TMCM-6xx, TMCM-101/109/110/111 modules (which are TMC428 based) and one for the TMCM-100 module (which is TMC453 based). You can choose the tool by selecting the appropriate tab page in the parameter calculation dialogue. Always be sure to use the right one.



TMCL Axis Parameters

Pulse divisor (SAP 154):	3	Ramp divisor (SAP 153):	7
Velocity (SAP 2/3/4, ROL/ROR):	200	Acceleration (SAP 5):	55
Microsteps (SAP 140):	6 (64x)		

Motor Parameter

Full Steps / Rev.: 200

Physical Units

Microstep Freq. [Hz]:	6103,515625	Accel. micro steps [Hz/s]:	25611,37080192
Full Step Freq. [Hz]:	95,36743164062	Accel. full steps [Hz/s]:	400,1776687800
Revolutions per sec.:	0,476837158203	Acceleration [RPS/s]:	2,000888343900
Revolutions per min.:	28,61022949218	Acceleration [RPM/s]:	120,0533006340
Time to reach velocity [s]:	0,238313		
Full steps to reach vel.:	11,363636		
Micro steps to reach vel.:	727,272727		

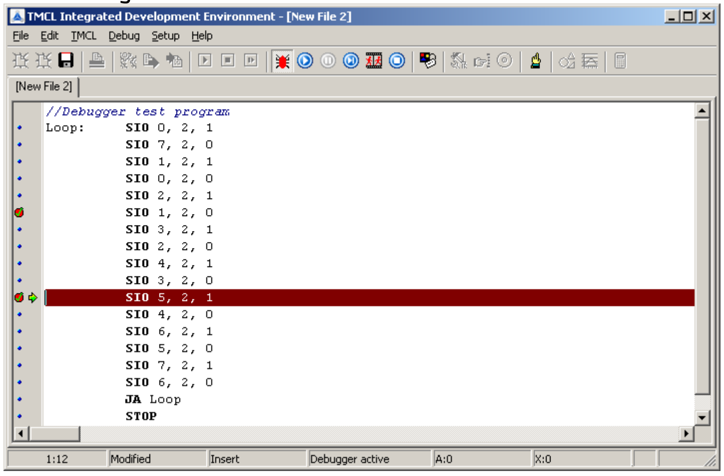
To use the calculation tool, just fill in the values that are known and then click the "Calculate" button. After changing any parameter always the "Calculate" button again. When a parameter in the TMCL section has been changed the physical units will be re-calculated with the next click on the "Calculate" button. When a paramater in the physical units section has been changed the TMCL parameters will be re-calculated with the next click on the "Calculate" button.

7.7 The TMCL debugger

The TMCL debugger makes source level debugging of TMCL programs possible. The TMCL program still runs on the module, so true in-system debugging can be done. In order to use the debugger the module must have at least firmware version 3.34 (TMCM-100, 110,111,3xx) resp. 6.28 (TMCM-101,102,6xx). Upgrade your module if needed.

7.7.1 Starting the debugger

Before starting the debugger you will first have to make sure that the module is connected to the PC and that the program in the editor of the TMCL-IDE is the same as the program on your module. This can be done either by assembling the program that currently is in the editor and afterwards downloading it to the module or by disassembling the program that is currently stored on the module.
After these two preconditions have been verified you can start the debugger either by selecting the function "Debugger active" in the "Debug" menu or by clicking the debugger icon on the tool bar. After the debugger has been successfully started, the debugger functions will be enabled and most other functions of the TMCL-IDE will be disabled (it is now also not possible to change the program in the editor).
You can exit the debugger by clicking "Debugger active" in the "Debug" menu or the debugger icon on the tool bar once again. Then, all debugger functions will be disabled and all other functions of the TMCL-IDE will be enabled again.


TMCL Integrated Development Environment - [New File 2]
File Edit TMCL Debug Setup Help
[New File 2]

```
//Debugger test program
Loop:      SIO 0, 2, 1
           SIO 7, 2, 0
           SIO 1, 2, 1
           SIO 0, 2, 0
           SIO 2, 2, 1
           SIO 1, 2, 0
           SIO 3, 2, 1
           SIO 2, 2, 0
           SIO 4, 2, 1
           SIO 3, 2, 0
           SIO 5, 2, 1
           SIO 4, 2, 0
           SIO 6, 2, 1
           SIO 5, 2, 0
           SIO 7, 2, 1
           SIO 6, 2, 0
           JA Loop
           STOP
```

```
1:12    Modified    Insert    Debugger active    A:0    X:0
```

Figure 7.1: A TMCL program with two breakpoints, standing in the second breakpoint.

7.7.2 Breakpoints

Breakpoints can be set or removed by clicking the apropriate line on the left breakpoint bar of the editor. A blue bullet is displayed in every line where a breakpoint can be set. When a breakpoint is set a red bullet with a green tick is displayed in that line.
When a program is run either by the "Run" or by the "Animate" function of the debugger it will be stopped when a breakpoint is reached. It can then be continued by clicking "Run", "Animate" or "Step" again. It can also be reset by clicking "Stop / Reset", so that it can be restarted form the beginning again.

7.7.3 The "Run / Continue" function

Choose the "Run / Continue" function from the "Debug" menu (or click the "Run" icon or press F9) to start the program resp. continue its execution. The program will be stopped either when its end is reached, it a breakpoint is reached or when the "Pause" function in the "Debug" menu or the "Pause" icon is selected. In the latter two cases the actual position in the program will be shown in the editor by a green arrow on the left side and the program execution can be continued by using either the "Run / Continue" function, the "Step" function or the "Animate" function. The contents of the accumulator and the X register are also shown on the status bar while the program is paused.

If you wish to restart the program for the very beginning, select the "Stop/Reset" function before cicking "Run / Continue".

7.7.4 The "Pause" function

When a program is running (either started by the "Run / Continue" or by the "Animate" function), program execution can be interrupted at any time by selecting the "Pause" function from the "Debug" menu or clicking the "Pause" icon on the tool bar or pressing F2. Program execution can be continued by clicking "Run", "Animate" or "Step" again.
While a program is paused the actual position in the program is shown by a green arrow in the editor. The contents of the accumulator and the X register are shown on the status bar of the TMCL-IDE.

7.7.5 The "Step" function

Use this function for a step-by-step execution of the TMCL program. Every time the "Step" function is selected (either by selecting "Step" in the debug menu, clicking the "Step" icon on the tool bar or pressing F7) the next command in the TMCL program is executed. The actual position is shown by a green arrow in the editor. The contents of the accumulator and the X register are also shown on the status bar.

7.7.6 The "Animate" function

This function automatically executes the TMCL program step-by-step, so that the flow of the program can be seen. The actual position in the program is shown and updated after every command, and the contents of the accumulator and the X register are also shown on the status bar and updated after every command.
The program will be paused when running into a breakpoint or when the "Pause" function is selected. Program execution can then be continued either by the "Run / Continue", the "Animate" or the "Step" function.

7.7.7 The "Stop / Reset" function

Selecting "Stop / Reset" from the "Debug" menu (or clicking the "Stop / Reset" button or pressing Ctrl+F2) stops program execution immediately and resets the program. This means that starting the program again using "Run / Continue", "Animate" or "Step" will start the program form the beginning.
This function can also be used when the program execution is paused (either by a breakpoint or by the "Pause" function) to reset it and ensure that the program can be started again from the beginning.

7.7.8 The "Direct Mode" function in the debugger

While in the debugger, the "Direct Mode" can be used at any time when the program is not running to inspect or change parameters. Use this with care, as changing parameters out of the normal program flow can lead to unexpected behaviour of the TMCL program.

7.8 The syntax of TMCL in the TMCL assembler

Here, the syntax of the TMCL commands used by the TMCL assembler is given. Please see the chapter 3 for an explanation of the functionality of the TMCL commands. The command mnemonics given there are used in the TMCL assembler. Please see also the sample program files and chapter 3 and 8 to learn more about TMCL programming.

7.8.1 Assembler directives

Assembler directives always start with a # sign. The only directive is *#include* to include a file. The name of that file must be given after the *#include* directive. If that file is already in the editor, it will be taken from there. Otherwise it will be loaded from file, using the include file path that can be set in the "Options" dialogue.
Example:
#include test.tmc

7.8.2 Symbolic constants

Symbolic constants are defined using the syntax <Name>=<Value>. A name must always start with a letter or the sign _ and may then contain any combination of letters, numbers and the sign _. A value must always be a decimal, hexadecimal or binary number or a constant expression (see section 7.8.3). Hexadecimal numbers start with a $ sign, binary numbers start with a % sign. Examples:

Speed=1000
Speed2=Speed/2
Mask=$FF
BinaryValue=%1010101

7.8.3 Constant expressions

Wherever a numerical value is needed, it can also be calculated during assembly. For this purpose constant expressions can be used. A constant expression is just a formula that evaluates to a constant value. The syntax is very similar to BASIC or other programming languages. Please note that the calculation takes place during assembly and not during execution of the TMCL program on the module.
Internally, the assembler uses floating point arithmetic to evaluate a constant expression, but as TMCL commands only take integer values, the result of a constant expression will always be rounded to an integer value when used as an argument to a TMCL command. Here is a list of functions and operators that can be used in constant expressions:

- Functions:

Name	Function
SIN	Sinus
COS	Cosinus
TAN	Tangens
ASIN	Arcus Sinus
ACOS	Arcus Cosinus
ATAN	Arcus Tangens
LOG	Logarithm Base 10
LN	Logarithm Base e
EXP	Power to Base e
SQRT	Square root
ABS	Absolute value
INT	Integer (truncate)
ROUND	Integer (Round)
SIGN	Returns -1 if argument<1, 0 if argument=0 1 if argument>0
DEG	Converts from radiant to degrees
RAD	Converts from degrees to radiant

- Operators

Symbol	Meaning
()	Parenthesis
^	Power
*	Multiplication
/	Division
+	Addition
-	Subtraction

Symbolic constants, floating point numbers, integer numbers, hexadecimal numbers and binary numbers can also be used in constant expressions. Here are some examples of constant expressions used wherever constant values can be placed:

```
ROL 0, 7+9*8
Speed2=Speed*SIN(0.5)
MVP ABS, 0, 3*1000
Sin90=Sin(Rad(90))
```

7.8.4 Labels

Labels have the form <Name>:. There are the same rules for label names as for symbolic constants.
Example (the label has the name "Loop"):

```
Loop:   MVP ABS, 0, 1000
        WAIT POS, 0, 0
        MVP ABS, 0, 0
        WAIT POS, 0, 0
        JA Loop
```

7.8.5 Comments

Comments always start with // (like in C++). The rest of the line is then ignored.

7.8.6 TMCL Commands

Here is a list of all command mnemonics that are recognized by the assembler. Please see chapter 3 of this manual for a detailed explanation of every command.

```
ROR <n1>, <n2>
ROL <n1>, <n2>
MST <n1>
MVP <mvp_opt>, <n1>, <n2>
SAP <n1>, <n2>, <n3>
GAP <n1>, <n2>
STAP <n1>, <n2>
RSAP <n1>, <n2>
SGP <n1>, <n2>, <n3>
GGP <n1>, <n2>
STGP <n1>, <n2>
RSGP <n1>, <n2>
RFS <rfs_opt>, <n1>
SIO <n1>, <n2>, <n3>
GIO <n1>, <n2>
CALC <op1>, <n2>, <n3>
CALCX <op2>, <n2>, <n3>
COMP <n1>
JC <cc>, <Label>
JA <Label>
CSUB <Label>
RSUB
WAIT <Event>, <n1>, <n2>
STOP
SAC <n1>, <n2>, <n3>
SCO <n1>, <n2>, <n3>
GCO <n1>, <n2>
CCO <n1>, <n2>
AAP <n1>, <n2>
AGP <n1>, <n2>
CLE <Flag>
```

with:
<n1>, <n2>, <n3>: Any numerical value, constant expression or symbolic constant
<mvp_opt>: An option for MVP: ABS, REL or COORD.
<rfs_opt>: An option for RFS: START, STOP or STATUS.
<cc>: A condition code: ZE, NZ, EQ, NE, GT, GE, LT, LE, ETO, EAL, EDV, EPO.
<Event>: A wait event. This can be TICKS, POS, LIMSW, REFSW or RFS.
<op1>: An operator for the CALC command: ADD,SUB,MUL,DIV,MOD,AND,OR,NOT,LOAD
<op2>: An operator for the CALCX command: ADD,SUB,MUL,DIV,MOD,AND,OR,NOT,LOAD,SWAP
<Label>: A label defined somewhere else in the program.
<Flag>: An error flag code: ALL, ETO, EAL, EDV or EPO.

8 TMCL Programming Techniques

8.1 General structure of a TMCL program

8.1.1 Initialization

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

8.1.2 Main loop

Embedded systems normally use a main loop that runs infinitely. This is also the case in a TMCL application that is running stand alone. Normally it also the auto start mode of the module should be turned on. After power up, the module then starts the TMCL program, which first does all necessary initializations and then enters the main loop, which does all necessary tasks end never ends (only when the module is powered off or reset). There are exceptions to this, e.g. when TMCL routines are called from a host in direct mode.
So most (but not all) stand alone TMCL programs look like this:

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

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

8.2 Using symbolic constants

To make a program better readable and understandable, symbolic constants should be used for all important numerical values that are used in a program (this is not only true for TMCL programs). Since version 1.68 of the TMCL-IDE we also provide an include file with symbolic names for all important axis parameters and global paramters.
So we can make out example look a little bit nicer:

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

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

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

Just have a look at the file "TMCLParam.tmc" provided with the TMCL-IDE. It contains symbolic constants that define alll important parameter numbers.
Using constants for other values also makes it easier to change them when they are used more thant once in a program. You will then just have to change the definition of the constant and not all occurances of the constant in the program.

8.3 Using variables

The "user variables" (chapter 5.3) can be used if variables are needed in your program. These can store temporary values. The commands SGP, GGP and AGP are used to work with user variables. SGP is used to set a variable to a constant value (e.g. during initialization phase). GGP is used to read the contents of a user variable and to copy it to the accumulator register for further usage. The AGP command can be used to copy the contents of the accumulator register to a user variable, e.g . to store the result of a calculation. Here are some examples

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

8.4 Using subroutines

The CSUB and RSUB commands provide a mechanism for using subroutines. The CSUB command branches to the given label. When an RSUB command is executed control goes back to the command that follows the CSUB command that called the subroutine.
This mechanism can also be nested. From a subroutine called by a CSUB command other subroutines can be called. In the current version of TMCL eight levels of nested subroutine calls are allowed.

8.5 Mixing direct mode and stand alone mode

Direct mode and stand alone mode can also be mixed. When a TMCL program is being executed in stand alone mode, direct mode commands are also processed (and they do not disturb the flow of the program running in stand alone mode). So it is also possible to query e.g. the actual position of the motor in direct mode while a TMCL program is running.
Communication between a program running in stand alone mode and a host can be done using the TMCL user variables. The host can then change the value of a user variable (using a direct mode SGP command) which is regularly polled by the TMCL program (e.g. in its main loop) and so the TMCL program can react on such changes. Vice versa, a TMCL program can change a user variable that is polled by the host (using a direct mode GGP command).
A TMCL program can be started by the host using the run command in direct mode (please see chapter 3.35). This way, also a set of TMCL routines can be defined that are called by a host. In this case it is recommended to place JA commands at the beginning of the TMCL program that jump to the specific routines. This assures that the entry addresses of the routines will not change even when the TMCL routines are change (so when changing the TMCL routines the host program does not have to be changed). Here is an example:

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

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

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

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

This example provides three very simple TMCL routines. They can be called from a host by issuing a run command with address 0 to call the first function, or a run command with address 1 to call the second function, or a run command with address 2 to call the third function.
You can see the addresses of the TMCL labels (that are needed for the run commands) by using the "Generate symbol file" function of the TMCL IDE (please see chapter 7.6.1.1).