



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

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

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



Contact us

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

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

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



Firmware Version V4.45

TMCL™ FIRMWARE MANUAL



TMCM-1180 PD86-1180

1-Axis Stepper
Controller / Driver
5.5A RMS/ 24 or 48V DC
USB, RS232, RS485, and CAN



stallGuard²

TRINAMIC Motion Control GmbH & Co. KG
Hamburg, Germany

www.trinamic.com



Table of Contents

1	Features.....	4
2	Overview.....	5
3	Putting the PD86-1180 into Operation.....	6
3.1	Starting up.....	6
3.2	Testing with a Simple TMCL Program.....	9
3.3	Operating the Module in Direct Mode.....	10
4	TMCL and TMCL-IDE.....	11
4.1	Binary Command Format.....	11
4.2	Reply Format.....	12
4.2.1	Status Codes.....	13
4.3	Standalone Applications.....	13
4.4	TMCL Command Overview.....	14
4.4.1	TMCL Commands.....	14
4.4.2	Commands Listed According to Subject Area.....	15
4.5	The ASCII Interface.....	19
4.5.1	Format of the Command Line.....	19
4.5.2	Format of a Reply.....	19
4.5.3	Configuring the ASCII Interface.....	20
4.6	Commands.....	21
4.6.1	ROR (rotate right).....	21
4.6.2	ROL (rotate left).....	22
4.6.3	MST (motor stop).....	23
4.6.4	MVP (move to position).....	24
4.6.5	SAP (set axis parameter).....	26
4.6.6	GAP (get axis parameter).....	27
4.6.7	STAP (store axis parameter).....	28
4.6.8	RSAP (restore axis parameter).....	29
4.6.9	SGP (set global parameter).....	30
4.6.10	GGP (get global parameter).....	31
4.6.11	STGP (store global parameter).....	32
4.6.12	RSGP (restore global parameter).....	33
4.6.13	RFS (reference search).....	34
4.6.14	SIO (set input / output).....	35
4.6.15	GIO (get input/output).....	37
4.6.16	CALC (calculate).....	39
4.6.17	COMP (compare).....	40
4.6.18	JC (jump conditional).....	41
4.6.19	JA (jump always).....	42
4.6.20	CSUB (call subroutine).....	43
4.6.21	RSUB (return from subroutine).....	44
4.6.22	WAIT (wait for an event to occur).....	45
4.6.23	STOP (stop TMCL program execution).....	46
4.6.24	SCO (set coordinate).....	47
4.6.25	GCO (get coordinate).....	48
4.6.26	CCO (capture coordinate).....	49
4.6.27	ACO (accu to coordinate; valid from TMCL version 4.18 on).....	50
4.6.28	CALCX (calculate using the X register).....	51
4.6.29	AAP (accumulator to axis parameter).....	52
4.6.30	AGP (accumulator to global parameter).....	53
4.6.31	CLE (clear error flags).....	54
4.6.32	VECT (set interrupt vector).....	55
4.6.33	EI (enable interrupt).....	56
4.6.34	DI (disable interrupt).....	57
4.6.35	RETI (return from interrupt).....	58
4.6.36	Customer Specific TMCL Command Extension (UF0.. UF7 / User Function).....	58
4.6.37	Request Target Position Reached Event.....	59

4.6.38	BIN (return to binary mode)	59
4.6.39	TMCL Control Functions	60
5	Axis Parameters	62
5.1	coolStep Related Parameters	68
6	Global Parameters	70
6.1	Bank 0	70
6.2	Bank 1	72
6.3	Bank 2	73
6.4	Bank 3	73
7	Hints and Tips	74
7.1	Reference Search	74
7.2	Changing the Prescaler Value of an Encoder	75
7.3	stallGuard2	76
7.4	Using the RS485 interface	76
8	TMCL Programming Techniques and Structure	77
8.1	Initialization	77
8.2	Main Loop	77
8.3	Using Symbolic Constants	77
8.4	Using Variables	78
8.5	Using Subroutines	78
8.6	Mixing Direct Mode and Standalone Mode	79
9	Life Support Policy	80
10	Revision History	81
10.1	Firmware Revision	81
10.2	Document Revision	81
11	References	82

1 Features

The PD86-1180 is a full mechatronic solution with state of the arte feature set. It is highly integrated and offers a convenient handling. The PD86-1180 consists of a NEMA 34 (flange size 86mm) stepper motor, controller/driver electronics and integrated encoder.

The TMCM-1180 is an intelligent stepper motor controller/driver module featuring the new outstanding coolStep™ technology for sensorless load dependent current control. This allows energy efficient motor operation. With the advanced stallGuard2™ feature the load of the motor can be detected with high resolution. The module is designed to be mounted directly on an 86mm flange QMot stepper motor.

Electrical data

- Supply voltage: +24V DC or +48V DC nominal
- Motor current: up to 5.5A RMS (programmable)

PANdrive motor

- Two phase bipolar stepper motor with up to 5.5A RMS nom. coil current
- Holding torque: 7Nm

Encoder

- Integrated sensOstep™ magnetic encoder (max. 256 increments per rotation) e.g. for step-loss detection under all operating conditions and positioning

Integrated motion controller

- Motion profile calculation in real-time (TMC428/429 motion controller)
- On the fly alteration of motor parameters (e.g. position, velocity, acceleration)
- High performance microcontroller for overall system control and serial communication protocol handling

Bipolar stepper motor driver

- Up to 256 microsteps per full step
- High-efficient operation, low power dissipation
- Dynamic current control
- Integrated protection
- stallGuard2 feature for stall detection
- coolStep feature for reduced power consumption and heat dissipation

Interfaces

- inputs for stop switches (left and right) and home switch
- general purpose inputs and 2 general purpose outputs
- USB, RS232, RS485 and CAN (2.0B up to 1Mbit/s) communication interfaces

Safety features

- Shutdown input. The driver will be disabled in hardware as long as this pin is left open or shorted to ground
- Separate supply voltage inputs for driver and digital logic – driver supply voltage may be switched off externally while supply for digital logic and therefore digital logic remains active

Software

- Available with TMCL or CANopen
- Standalone TMCL operation or remote controlled operation
- Program memory (non volatile) for up to 2048 TMCL commands
- PC-based application development software TMCL-IDE available for free
- CANopen: CiA 301 + CiA 402 (homing mode, profile position mode and velocity mode) supported

Please refer to separate Hardware Manual for further information.

2 Overview

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-1180 consists of two parts, a boot loader and the firmware itself. Whereas the boot loader is installed during production and testing at TRINAMIC and remains normally untouched throughout the whole lifetime, the firmware can be updated by the user. New versions can be downloaded free of charge from the TRINAMIC website (<http://www.trinamic.com>).

The firmware shipped with this module is related to the standard TMCL firmware shipped with most of TRINAMIC modules with regard to protocol and commands. Corresponding, this module is based on the TMC428/429 stepper motor controller and the TMC262A-PC power driver and supports the standard TMCL with a special range of values.

The TMC262A-PC is a new energy efficient high current high precision microstepping driver IC for bipolar stepper motors and offers TRINAMICs patented coolStep feature with its special commands. Please mind this technical innovation.

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

3 Putting the PD86-1180 into Operation

Here you can find basic information for putting your PANdrive into operation. The further text contains a simple example for a TMCL program and a short description of operating the module in direct mode.

The things you need:

- PD83-1180
- Interface (RS232, RS485, USB or CAN) suitable to your PANdrive with cables
- Nominal supply voltage +24V DC (+24 or +48V DC) for your module
- TMCL-IDE program and PC
- External encoder optional. The PANdrive™ has an integrated sensOstep encoder.

Precautions:

- Do not connect or disconnect the PD86-1180 while powered!
- Do not connect or disconnect the motor while powered!
- Do not exceed the maximum power supply of 55V DC.
- Start with power supply OFF!

3.1 Starting up

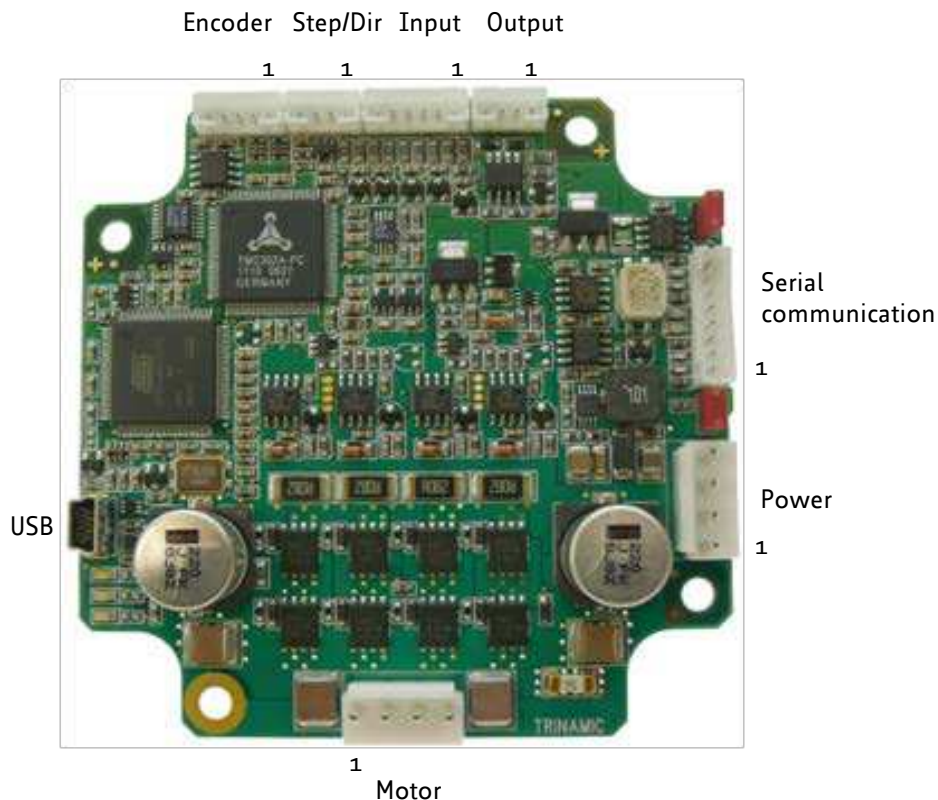


Figure 3.1 Overview connectors

1. **Connect the interface**

a) **Connect the RS232, the RS485, or the CAN interface**

A 2mm pitch 8 pin JST B8B-PH-K connector is used for serial communication. With this connector the module supports RS232, RS485, and CAN communication.


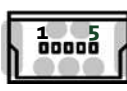
	Pin	Label	Description
	1	RS232_TxD	RS232 transmit data
	2	RS232_RxD	RS232 receive data
	3	GND	Module ground (system and signal ground)
	4	CAN_H	CAN_H bus line (dominant high)
	5	CAN_L	CAN_L bus line (dominant low)
	6	GND	Module ground (system and signal ground)
	7	RS485+	RS485 non-inverted bus signal
	8	RS485-	RS485 inverted bus signal

Figure 3.2: RS232, RS485, and CAN connector

b) **Connect the USB interface**

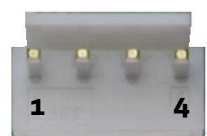
A 5-pin mini-USB connector is available on the board.

Download and install the file *TMCM-1180.inf* (www.trinamic.com).

	Pin	Label	Description
	1	VBUS	+5V power
	2	D-	Data -
	3	D+	Data +
	4	ID	Not connected
	5	GND	ground

2. **Connect the power supply**

A 4-pin JST B04P-VL connector is used for power supply.

	Pin	Label	Description
	1	+U _{Driver}	Module + driver stage power supply input (nom. +48V DC)
	2	+U _{Logic}	(Optional) separate digital logic power supply input (nom. +48V DC)
	3	GND	Module ground (power supply and signal ground)
	4	GND	Module ground (power supply and signal ground)

3. **Switch ON the power supply**

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

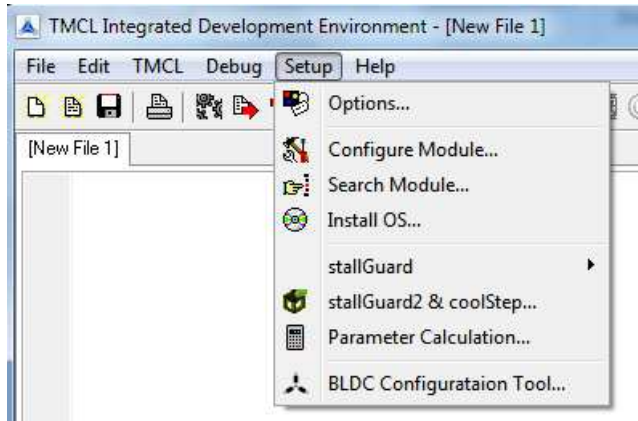
If this does not occur, switch power OFF and check your connections as well as the power supply.

4. Start the TMCL-IDE software development environment

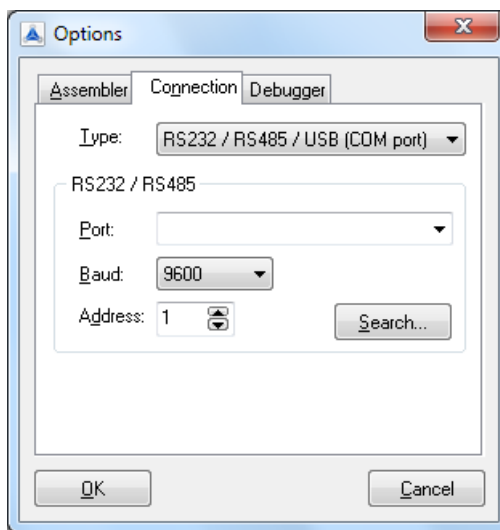
The TMCL-IDE is available on the TechLibCD and on www.trinamic.com.

Installing the TMCL-IDE:

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



- For RS232 and RS485 choose **COM port** and **type** with the parameters shown below (baud rate 9600). Click OK.



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

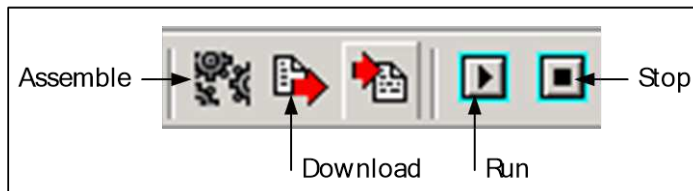
3.2 Testing with a Simple TMCL Program

Open the file test2.tmc. Change the *motor number 2* in the second paragraph in *motor number 0* (because there is only one motor involved). Now your test program looks as follows:

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

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

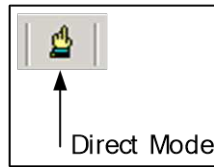
    SAP 4, 0, 500            //Set max. Velocity
    SAP 5, 0, 50            //Set max. Acceleration
Loop: MVP ABS, 0, 10000     //Move to Position 10000
    WAIT POS, 0, 0          //Wait until position reached
    MVP ABS, 0, -10000     //Move to Position -10000
    WAIT POS, 0, 0          //Wait until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



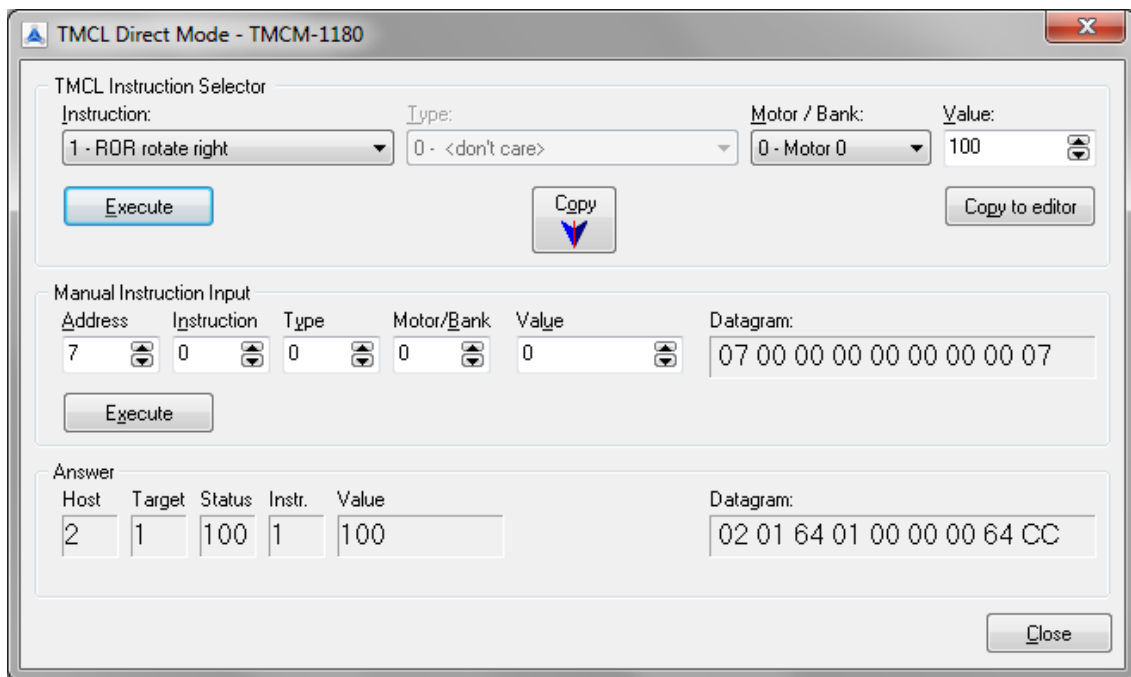
1. Click on Icon **Assemble** to convert the TMCL into machine code.
2. Then download the program to the TMC-1180 module via the icon **Download**.
3. Press icon **Run**. The desired program will be executed.
4. Click **Stop** button to stop the program.

3.3 Operating the Module in Direct Mode

1. Start TMCL *Direct Mode*.



2. If the communication is established the PD86-1180 is automatically detected. **If the module is not detected, please check all points above (cables, interface, power supply, COM port, baud rate).**
3. Issue a command by choosing **Instruction**, **Type** (if necessary), **Motor**, and **Value** and click **Execute** to send it to the module.



Examples:

- ROR rotate right, motor 0, value 100 -> Click *Execute*. The first motor is rotating now.
- MST motor stop, motor 0 -> Click *Execute*. The first motor stops now.

Note

Chapter 5 (axis parameters) includes a diagram which shows important coolStep related axis parameters and their functions.

4 TMCL and TMCL-IDE

The TMCM-1180 supports TMCL direct mode (binary commands or ASCII interface) and standalone TMCL program execution. You can store up to 2048 TMCL instructions on it.

In direct mode and most cases the TMCL communication over RS485, RS232, USB or CAN follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCL-1180. The TMCL interpreter on the module will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over RS485/RS232/USB/CAN to the bus master. Only then should the master transfer the next command. Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.

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

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

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

4.1 Binary Command Format

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes. When a command is to be sent via RS232, RS485 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In this case it consists of nine bytes.

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

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

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

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

CHECKSUM CALCULATION

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

in C:

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

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

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

in Delphi:

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

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

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

4.2 Reply Format

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

The reply format for RS485/RS232/USB is as follows:

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

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

4.2.1 Status Codes

The reply contains a status code.

The status code can have one of the following values:

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

4.3 Standalone Applications

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

4.4 TMCL Command Overview

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

4.4.1 TMCL Commands

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

4.4.2 Commands Listed According to Subject Area

4.4.2.1 Motion Commands

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

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

4.4.2.2 Parameter Commands

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

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

4.4.2.3 Control Commands

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

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

4.4.2.4 I/O Port Commands

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

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

4.4.2.5 Calculation Commands

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

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

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

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

4.4.2.6 Interrupt Commands

Due to some customer requests, interrupt processing has been introduced in the TMCL firmware s.

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

4.4.2.6.1 Interrupt Types

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

4.4.2.6.2 Interrupt Processing

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

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

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

4.4.2.6.3 Interrupt Vectors

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

Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	(Target) position reached
15	Stall (stallGuard2)
21	Deviation
27	Stop left
28	Stop right
39	IN_0 change
40	IN_1 change

4.4.2.6.4 Further Configuration of Interrupts

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

4.4.2.6.5 Using Interrupts in TMCL

For using an interrupt proceed as follows:

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

EXAMPLE FOR A TIMER INTERRUPT:

```

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

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

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

```

In the example above, the interrupt numbers are used directly. To make the program better readable use the provided include file *Interrupts.inc*. This file defines symbolic constants for all interrupt numbers

which can be used in all interrupt commands. The beginning of the program above then looks like the following:

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

Please also take a look at the other example programs.

4.4.2.7 ASCII Commands

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

4.5 The ASCII Interface

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

THE FOLLOWING COMMANDS CAN BE USED IN ASCII MODE:

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

Only direct mode commands can be entered in ASCII mode!

SPECIAL COMMANDS WHICH ARE ONLY AVAILABLE IN ASCII MODE:

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

ENTERING AND LEAVING ASCII MODE:

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

4.5.1 Format of the Command Line

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

EXAMPLES FOR VALID COMMAND LINES:

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

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

4.5.2 Format of a Reply

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

The reply consists of:

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

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

4.5.3 Configuring the ASCII Interface

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

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

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

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

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

4.6 Commands

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

4.6.1 ROR (rotate right)

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

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

The module is based on the TMC428/429 stepper motor controller and the TMC262A-PC power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR 0, <velocity>

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

Rotate right, velocity = 350

Mnemonic: ROR 0, 350

Binary:

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

4.6.2 ROL (rotate left)

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

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

The module is based on the TMC428/429 stepper motor controller and the TMC262A-PC power driver. This makes possible choosing a velocity between 0 and 2047.

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL 0, <velocity>

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 - OK	(don't care)

Example:

Rotate left, velocity = 1200

Mnemonic: ROL 0, 1200

Binary:

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

4.6.3 MST (motor stop)

With this command the motor will be instructed to stop.

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

Related commands: ROL, ROR, SAP, GAP

Mnemonic: MST 0

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

Stop motor

Mnemonic: MST 0

Binary:

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

4.6.4 MVP(move to position)

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

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

THREE OPERATION TYPES ARE AVAILABLE:

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

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

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

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

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

Binary representation:

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

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

Reply in direct mode:

STATUS	VALUE
100 – OK	(don't care)

Example:

Move motor to (absolute) position 90000

Mnemonic: MVP ABS, 0, 9000

Binary:

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

Example:

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

Binary:

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

Example:

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

Binary:

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

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