TMCM-1278 TMCL™ Firmware Manual

Firmware Version V1.15 | Document Revision V1.04 • 2020-NOV-25

The TMCM-1278 is a single axis controller/driver module for 2-phase bipolar stepper motors with with coil currents of up to 9A RMS. The TMCM-1278 TMCL firmware allows to control the module using TMCL™ commands, supporting standalone operation as well as direct mode control, making use of the Trinamic TMC5160 motion controller and motor driver. Dynamic current control, and quiet, smooth and efficient operation are combined with StealthChop™, DcStep™, StallGuard™ and CoolStep™ features.

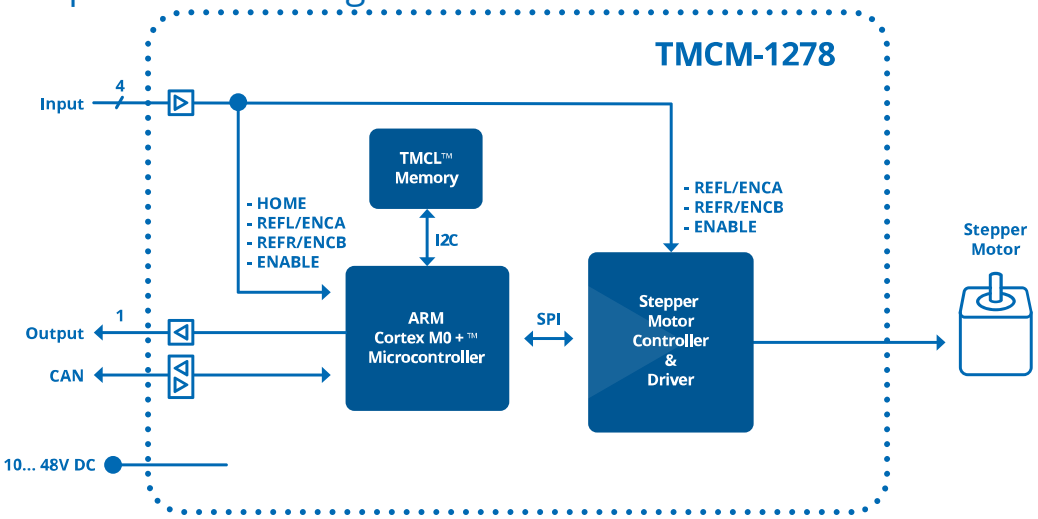
Features

- Single Axis Stepper motor control
- Supply voltage 10…48V DC
- TMCL™
- Motion Contoller
- CAN
- SixPoint™ ramp motion controller
- StealthChop™ silent PWM mode
- SpreadCycle™ smart mixed decay
- StallGuard2™ load detection
- CoolStep™ automatic current scaling
- DcStep™

Applications

- Laboratory Automation
- Manufacturing
- Semiconductor Handling
- Robotics
- Factory Automation
- CNC
- Life Science
- Biotechnology
- Liquid Handling

Simplified Block Diagram



Read entire documentation.

TRINAMIC
MOTION CONTROL
Now part of Maxim Integrated

Contents

1 Features

The TMCM-1278 is a single axis controller/driver module for 2-phase bipolar stepper motors with state of the art feature set. It has been designed for coil currents up to 9A RMS and up to 48V DC supply voltage. Three digital inputs can be configured as home and endstop switches or incremental encoder inputs. With its high energy efficiency from TRINAMIC's CoolStep™, technology cost for power consumption is kept down. The TMCL firmware allows for both standalone and direct mode operation.

Main characteristics

- Motion controller & stepper motor driver:
  - Hardware motion profile calculation in real-time
  - On the fly alteration of motion parameters (e.g. position, velocity, acceleration)
  - High performance microcontroller for overall system control and communication protocol handling
  - 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
  - StealthChop™ feature for quiet operation and smooth motion
  - DcStep™ feature for load dependent speed control
- Interfaces
  - CAN
  - Low-active enable pin
  - Three I/O modes:
    * Incremental Encoder and Home Switch
    * Home and Endstop Switches
    * One analog and two digital inputs

Software

TMCL: remote controlled operation alone or during step/direction mode. PC-based application development software TMCL-IDE available for free.

Electrical data

- Supply voltage: 10…48V DC (+24V or +48V nominal).
- Motor current: up to 9A RMS / 12.7A peak (programmable).

Please see also the separate Hardware Manual.

1.1 StallGuard2

StallGuard2 is a high-precision sensorless load measurement using the back EMF of the coils. It can be used for stall detection as well as other uses at loads below those which stall the motor. The StallGuard2 measurement value changes linearly over a wide range of load, velocity, and current settings. At maximum motor load, the value reaches zero or is near zero. This is the most energy-efficient point of operation for the motor.

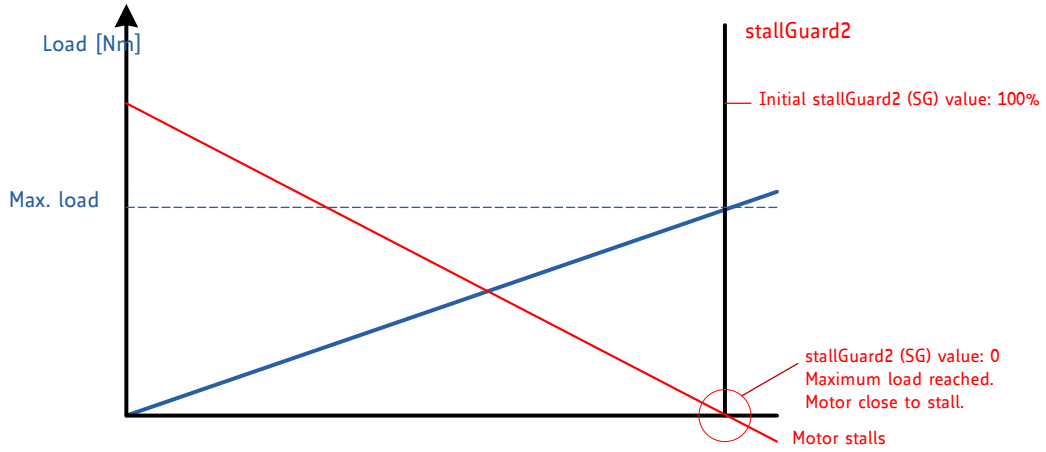


Figure 1: StallGuard2 Load Measurement as a Function of Load

1.2 CoolStep

CoolStep is a load-adaptive automatic current scaling based on the load measurement via StallGuard2 adapting the required current to the load. Energy consumption can be reduced by as much as 75%. CoolStep allows substantial energy savings, especially for motors which see varying loads or operate at a high duty cycle. Because a stepper motor application needs to work with a torque reserve of 30% to 50%, even a constant-load application allows significant energy savings because CoolStep automatically enables torque reserve when required. Reducing power consumption keeps the ystem cooler, increases motor life, and allows cost reduction.

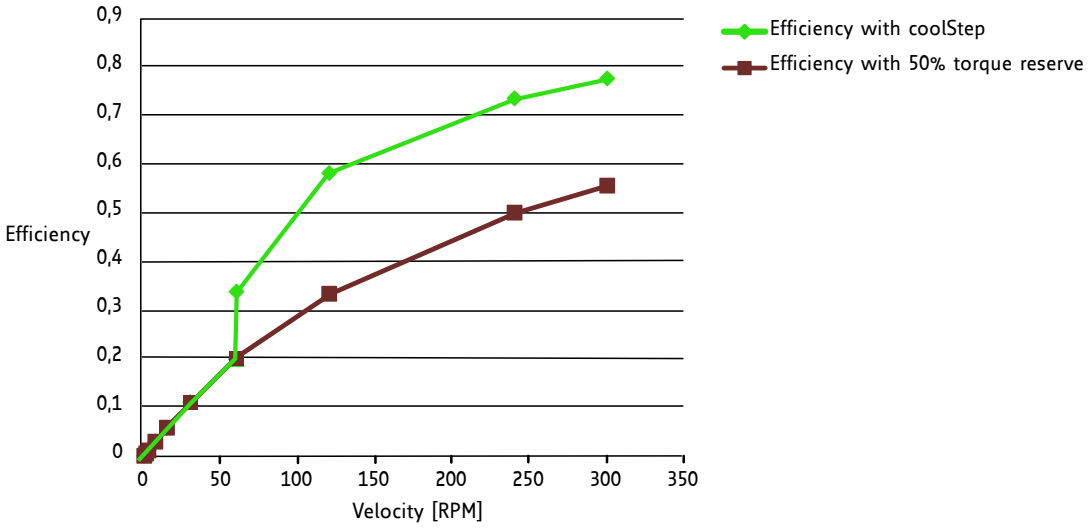


Figure 2: Energy Efficiency Example with CoolStep

1.3 SixPoint Motion Controller

TRINAMIC's SixPoint motion controller is a new type of ramp generator which offers faster machine operation compared to the classical linear acceleration ramps. The SixPoint ramp generator allows adapting the acceleration ramps to the torque curves of a stepper motor. It uses two different acceleration settings for the acceleration phase and also tow different deceleration settings for the deceleration phase. Start and stop speeds greater than zero can also be used.

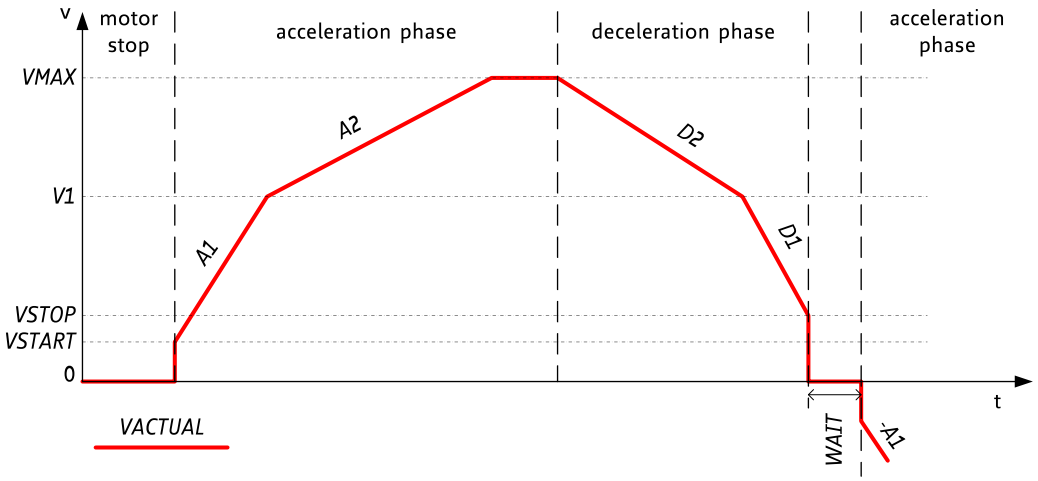


Figure 3: Typical motion profile with TRINAMIC's SixPoint motion controller

A six point ramp begins using the start speed V_{START} (which can also be zero). Then, the acceleration value A1 will be used to accelerate the motor to the speed V1. When the speed V1 has been reached, the motor will be further accelerated using the acceleration value A2 until it has reached the speed V_{MAX}. The deceleration phase begins using the deceleration value D2. After reaching the speed V1 again the deceleration value D2 will be used to declerate to the stop speed V_{STOP} (which can also be zero).

2 First Steps with TMCL

In this chapter you can find some hints for your first steps with the TMCM-1278 and TMCL. You may skip this chapter if you are already familiar with TMCL and the TMCL-IDE.

Things that you will need

- Your TMCM-1278 Module.
- A CAN adapter.
- A power supply (24V DC) for your TMCM-1278 module.
- The TMCL-IDE 3.x already installed on your PC
- A two-phase bipolar stepper motor.

2.1 Basic Setup

First of all, you will need a PC with Windows (at least Windows 7) and the TMCL-IDE 3.x installed on it. If you do not have the TMCL-IDE installed on your PC then please download it from the TMCL-IDE product page of Trinamic's website (http://www.trinamic.com) and install it on your PC.

Please also ensure that your TMCM-1278 is properly connected to your power supply and that the stepper motor is properly connected to the module. Please see the TMCM-1278 hardware manual for instructions on how to do this. **Do not connect or disconnect a stepper motor to or from the module while the module is powered!**
Then, please start up the TMCL-IDE. After that you can connect your TMCM-1278 via CAN and switch on the power supply for the module (while the TMCL-IDE is running on your PC). The module will be recognized by the TMCL-IDE, and necessary driver registrations in Windows will automatically done by the TMCL-IDE.

2.2 Using the TMCL Direct Mode

At first try to use some TMCL commands in direct mode. In the TMCL-IDE a tree view showing the TMCM-1278 and all tools available for it is displayed. Click on the Direct Mode entry of the tool tree. Now, the Direct Mode tool will pop up.

In the Direct Mode tool you can choose a TMCL command, enter the necessary parameters and execute the command. For example, choose the command ROL (rotate left). Then choose the appropriate motor (motor 0 if your motor is connected to the motor 0 connector). Now, enter the desired speed. Try entering 51200 (pps) as the value and then click the Execute button. The motor will now run. Choose the MST (motor stop) command and click Execute again to stop the motor.

2.3 Changing Axis Parameters

Next you can try changing some settings (also called axis parameters) using the SAP command in direct mode. Choose the SAP command. Then choose the parameter type and the motor number. Last, enter the desired value and click execute to execute the command which then changes the desired parameter. The following table points out the most important axis parameters. Please see chapter 4 for a complete list of all axis parameters.

<table>
<tr><th colspan="5">Most important axis parameters</th></tr>
<tr><th>Number</th><th>Axis Parameter</th><th>Description</th><th>Range [Units]</th><th>Access</th></tr>
<tr><td>4</td><td>Maximum positioning speed</td><td>The maximum speed used for positioning ramps.</td><td>0...7999774 [pps]</td><td>RW</td></tr>
<tr><td>5</td><td>Maximum acceleration</td><td>Maximum acceleration in positioning ramps. Acceleration and deceleration value in velocity mode.</td><td>0...7629278 [pps^2]</td><td>RW</td></tr>
<tr><td>6</td><td>Maximum current</td><td>Motor current used when motor is running. The maximum value is 255 which means 100% of the maximum current of the module.
The current can be adjusted in 32 steps:
<table>
<tr><td>0...7</td><td>79...87</td><td>160...167</td><td>240...247</td></tr>
<tr><td>8...15</td><td>88...95</td><td>168...175</td><td>248...255</td></tr>
<tr><td>16...23</td><td>96...103</td><td>176...183</td><td></td></tr>
<tr><td>24...31</td><td>104...111</td><td>184...191</td><td></td></tr>
<tr><td>32...39</td><td>112...119</td><td>192...199</td><td></td></tr>
<tr><td>40...47</td><td>120...127</td><td>200...207</td><td></td></tr>
<tr><td>48...55</td><td>128...135</td><td>208...215</td><td></td></tr>
<tr><td>56...63</td><td>136...143</td><td>216...223</td><td></td></tr>
<tr><td>64...71</td><td>144...151</td><td>224...231</td><td></td></tr>
<tr><td>72...79</td><td>152...159</td><td>232...239</td><td></td></tr>
</table>
The most important setting, as too high values can cause motor damage.</td><td>0...255</td><td>RW</td></tr>
<tr><td>7</td><td>Standby current</td><td>The current used when the motor is not running. The maximum value is 255 which means 100% of the maximum current of the module. This value should be as low as possible so that the motor can cool down when it is not moving.</td><td>0...255</td><td>RW</td></tr>
</table>

Table 1: Most important Axis Parameters

2.4 Testing with a simple TMCL Program

Now, test the TMCL stand alone mode with a simple TMCL program. To type in, assemble and download the program, you will need the TMCL creator. This is also a tool that can be found in the tool tree of the TMCL-IDE. Click the TMCL creator entry to open the TMCL creator. In the TMCL creator, type in the following little TMCL program:

```
ROL 0, 51200              //Rotate motor 0 with speed 10000
WAIT TICKS, 0, 500
MST 0
ROR 0, 51200               //Rotate motor 0 with 50000
WAIT TICKS, 0, 500
MST 0
```


```
7
    SAP 4, 0, 51200             //Set max. Velocity
9   SAP 5, 0, 51200             //Set max. Acceleration
Loop:
11  MVP ABS, 0, 512000          //Move to Position 512000
    WAIT POS, 0, 0              //Wait until position reached
13  MVP ABS, 0, -512000         //Move to Position -512000
    WAIT POS, 0, 0              //Wait until position reached
15  JA Loop                     //Infinite Loop
```

After you have done that, take the following steps:

1. Click the Assemble icon (or choose Assemble from the TMCL menu) in the TMCL creator to assemble the program.
2. Click the Download icon (or choose Download from the TMCL menu) in the TMCL creator to download the program to the module.
3. Click the Run icon (or choose Run from the TMCL menu) in the TMCL creator to run the program on the module.

Also try out the debugging functions in the TMCL creator:

1. Click on the Bug icon to start the debugger.
2. Click the Animate button to see the single steps of the program.
3. You can at any time pause the program, set or reset breakpoints and resume program execution.
4. To end the debug mode click the Bug icon again.

3 TMCL and the TMCL-IDE — An Introduction

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-1278 consists of two parts, a boot loader and the firmware itself. Whereas the boot loader is installed during production and testing at TRINAMIC and remains untouched throughout the whole lifetime, the firmware can be updated by the user. New versions can be downloaded free of charge from the TRINAMIC website (http://www.trinamic.com).
The TMCM-1278 supports TMCL direct mode (binary commands). It also implements standalone TMCL program execution. This makes it possible to write TMCL programs using the TMCL-IDE and store them in the memory of the module.

In direct mode the TMCL communication over RS-232, RS-485, CAN, and USB follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCM-1278. 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 the interface to the bus master. Only then should the master transfer the next command.

Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.
The Trinamic Motion Control Language [TMCL] provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM module to form programs that run standalone on the module. For this purpose there are not only motion control commands but also commands to control the program structure (like conditional jumps, compare and calculating).

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

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

3.1 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 RS-232, RS-485, RS-422 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In these cases it consists of nine bytes.

The binary command format with RS-232, RS-485, RS-422 and USB is as follows:

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

Table 2: TMCL Command Format

Info	The checksum is calculated by accumulating all the other bytes using an 8-bit addition.
Note	When using the CAN interface, leave out the address byte and the checksum byte. With CAN, the CAN-ID is used as the module address and the checksum is not needed because CAN bus uses hardware CRC checking.

3.1.1 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 which show how to do this:

Checksum calculation 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
```

Checksum calculation 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
```

3.2 Reply Format

Every time a command has been sent to a module, the module sends a reply. The reply format with RS-232, RS-485, RS-422 and USB is as follows:

TMCL Reply Format	
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

Table 3: TMCL Reply Format

Info	The checksum is also calculated by adding up all the other bytes using an 8-bit addition. Do not send the next command before having received the reply!

Note	When using CAN interface, the reply does not contain an address byte and a checksum byte. With CAN, the CAN-ID is used as the reply address and the checksum is not needed because the CAN bus uses hardware CRC checking.

3.2.1 Status Codes

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

TMCL Status Codes	
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

Table 4: TMCL Status Codes

3.3 Standalone Applications

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

3.4 TMCL Command Overview

This section gives a short overview of all TMCL commands.

Overview of all 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)
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 variables 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 analog/digital input
CALC	19	<operation>, <value>	Aithmetical operation between accumulator and direct value
COMP	20	<value>	Compare accumulator with 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

Command	Number	Parameter	Description
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>	Arithmetical operation between accumulator and X-register
AAP	34	<parameter>, <motor number>	Accumulator to axis parameter
AGP	35	<parameter>, <bank number>	Accumulator to global parameter
CLE	36	<flag>	Clear an error flag
VECT	37	<interrupt number>, <address>	Define interrupt vector
RETI	38		Return from interrupt
ACO	39	<coordinate number>, <motor number>	Accu to coordinate
CALCVV	40	<operation>, <user variable 1>, <user variable 2>	Arithmetical operation between two user variables
CALCVA	41	<operation>, <user variable>	Arithmetical operation between user variable and accumulator
CALCAV	42	<operation>, <user variable>	Arithmetical operation between accumulator and user variable
CALCVX	43	<operation>, <user variable>	Arithmetical operation between user variable and X register
CALCXV	44	<operation>, <user variable>	Arithmetical operation between X register and user variable
CALCV	45	<operation>, <value>	Arithmetical operation between user variable and direct value
MVPA	46	ABS|REL|COORD, <motor number>	Move to position specified by accumulator
RST	48	<jump address>	Restart the program from the given address
DJNZ	49	<user variable>, <jump address>	Decrement and jump if not zero
ROLA	50	<motor number>	Rotate left, velocity specified by accumulator
RORA	51	<motor number>	Rotate right, velocity specified by accumulator
SIV	55	<value>	Set indexed variable
GIV	56		Get indexed variable

Command	Number	Parameter	Description
AIV	57		Accumulator to indexed variable

Table 5: Overview of all TMCL Commands

3.5 TMCL Commands by Subject

3.5.1 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.

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

Table 6: Motion Commands

3.5.2 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.

Parameter Commands		
Mnemonic	Command number	Meaning
SAP	5	Set axis parameter
GAP	6	Get axis parameter
SGP	9	Set global parameter
GGP	10	Get global parameter
STGP	11	Store global parameter
RSGP	12	Restore global parameter

Table 7: Parameter Commands

3.5.3 Branch Commands

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

Branch Commands		
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

Table 8: Branch Commands

3.5.4 I/O Port Commands

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

I/O Port Commands		
Mnemonic	Command number	Meaning
SIO	14	Set output
GIO	15	Get input

Table 9: I/O Port Commands

3.5.5 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.

Calculation Commands		
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

Table 10: Calculation Commands

For calculating purposes there is an accumulator (also called 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.

Please see also chapter 3.5.7 for more calculation commands.

3.5.6 Interrupt Processing Commands

TMCL also contains functions for a simple way of interrupt processing. Using interrupts, many tasks can be programmed in an easier way.
The following commands are use to define and handle interrupts:

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

Table 11: Interrupt Processing Commands

3.5.6.1 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 include file Interrupts.inc in order to have symbolic constants for the interrupt numbers. Table 12 show all interrupts that are available on the TMCM-1278.

Interrupt Vectors	
Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached 0
15	StallGuard axis 0
21	Deviation axis 0
27	Left stop switch 0
28	Right stop switch 0
39	Input change 0
40	Input change 1
41	Input change 2
255	Global interrupts

Table 12: Interrupt Vectors

3.5.6.2 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 (i.e. accumulator register, X register, flags) will be saved automatically.
There is no interrupt nesting, i.e. all other interrupts are disabled while an interrupt handling routine is being executed.
On return from an interrupt handling routine (RETI command), the context of the normal program will automatically be restored and the execution of the normal program will be continued.

3.5.6.3 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 (chapter 3.6.7) for further information about that.

3.5.6.4 Using Interrupts in TMCL

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

- Define an interrupt handling routine using the VECT command.
- If necessary, configure the interrupt using an SGP <type>, 3, <value> command.
- Enable the interrupt using an EI <interrupt> command.
- Globally enable interrupts using an EI 255 command.
- An interrupt handling routine must always end with a RETI command.
- Do not allow the normal program flow to run into an interrupt handling routine.

The following example shows the use of a timer interrupt:

```
    VECT 0, Timer0Irq  //define the interrupt vector
    SGP 0, 3, 1000     //configure the interrupt: set its period to 1000ms
    EI 0               //enable this interrupt
    EI 255             //globally switch on interrupt processing

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

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

In the example above, the interrupt numbers are being 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 as follows:

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

3.5.7 New TMCL Commands

In order to make several operations easier, the following new commands have been introduced from firmware version 1.14 on. Using these new commands many taks can be programmed in an easier way. This can save some code, thus making a TMCL program shorter, faster and easier to understand.

Please note that these commands are not available on TMCM-1278 modules with firmware versions before 1.14. So please make sure that at least firmware version 1.14 is installed before using them.

New TMCL Commands		
Mnemonic	Command number	Meaning
CALCVV	40	Calculate using two user variables
CALCVA	41	Calculate using a user variable and the accumulator
CALCAV	42	Calculate using the accumulator and a user variable
CALCVX	43	Calculate using a user variable and the X register
CALCXV	44	Calculate using the X register and a user variable

Mnemonic	Command number	Meaning
CALCV	45	Calculate using a user variable and a direct value
MVPA	46	Move to position specified by accumulator
RST	48	Restart the program
DJNZ	49	Decrement and jump if not zero
CALL	80	Conditional subroutine call
ROLA	50	Rotate left using the accumulator
RORA	51	Rotate right using the accumulator
SIV	55	Set indexed variable
GIV	56	Get indexed variable
AIV	57	Accu to indexed variable

Table 13: New TMCL Commands

3.6 Detailed TMCL Command Descriptions

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

3.6.1 ROR (Rotate Right)

The motor is instructed to rotate with a specified velocity in right direction (increasing the position counter). The velocity is given in microsteps per second (pulse per second [pps]).

Internal function:

- First, velocity mode is selected.
- Then, the velocity value is transferred to axis parameter #2 (target velocity).

Related commands: ROL, MST, SAP, GAP.

Mnemonic: ROR <axis>, <velocity>

Binary Representation			
Instruction	Type	Motor/Bank	Value
1	0	0	-2147483648...2147583647

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Rotate right motor 0, velocity 51200.
Mnemonic: ROR 0, 51200.

Binary Form of ROR 0, 51200	
Field	Value
Target address	01_h
Instruction number	01_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	$C8_h$
Value (Byte 0)	00_h
Checksum	CA_h

3.6.2 ROL (Rotate Left)

The motor is instructed to rotate with a specified velocity in left direction (decreasing the position counter). The velocity is given in microsteps per second (pulse per second [pps]).

Internal function:

- First, velocity mode is selected.
- Then, the velocity value is transferred to axis parameter #2 (target velocity).

Related commands: ROR, MST, SAP, GAP.

Mnemonic: ROL <axis>, <velocity>

Binary Representation			
Instruction	Type	Motor/Bank	Value
2	0	0	-2147483648...2147583647

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Rotate left motor 0, velocity 51200.
Mnemonic: ROL 0, 51200.

Binary Form of ROL 0, 51200	
Field	Value
Target address	01_h
Instruction number	02_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	$C8_h$
Value (Byte 0)	00_h
Checksum	CB_h

3.6.3 MST (Motor Stop)

The motor is instructed to stop with a soft stop.

Internal function: The velocity mode is selected. Then, the target speed (axis parameter #0) is set to zero.
Related commands: ROR, ROL, SAP, GAP.

Mnemonic: MST <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
3	0	0	0

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Stop motor 0.
Mnemonic: MST 0.

Binary Form of MST 0	
Field	Value
Target address	01_h
Instruction number	03_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	04_h

3.6.4 MVP (Move to Position)

With this command the motor will be instructed to move to a specified relative or absolute position. 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 as well as other ramp parameters are defined by the appropriate axis parameters. For a list of these parameters please refer to section 4.
The range of the MVP command is 32 bit signed (-2147483648...2147483647). Positioning can be interrupted using MST, ROL or ROR commands.

Three operation types are available:

- Moving to an absolute position in the range from -2147483648...2147483647 ($-2^{31}...2^{31}-1$).
- Starting a relative movement by means of an offset to the actual position. In this case, the new resulting position value must not exceed the above mentioned limits, too.
- Moving the motor to a (previously stored) coordinate (refer to SCO for details).

Note The distance between the actual position and the new position must not be more than 2147483647 ($2^{31}-1$) microsteps . Otherwise the motor will run in the opposite direction in order to take the shorter distance (caused by 32 bit overflow).

Internal function: A new position value is transferred to the axis parameter #0 (target position).
Related commands: SAP, GAP, SCO, GCO, CCO, ACO, MST.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
4	0 – ABS – absolute	0	<position>
	1 – REL – relative	0	<offset>
	2 – COORD – coordinate	0...255	<coordinate number (0..20)>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Move motor 0 to position 90000.
Mnemonic: MVP ABS, 0, 90000

Binary Form of MVP ABS, 0, 90000	
Field	Value
Target address	01_h
Instruction number	04_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	01_h
Value (Byte 1)	$5F_h$
Value (Byte 0)	90_h
Checksum	$F5_h$

Example
Move motor 0 from current position 10000 steps backward.
Mnemonic: MVP REL, 0, -10000

Binary Form of MVP REL, 0, -10000	
Field	Value
Target address	01_h
Instruction number	04_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	FF_h
Value (Byte 2)	FF_h
Value (Byte 1)	$D8_h$
Value (Byte 0)	$F0_h$
Checksum	CC_h

Example
Move motor 0 to stored coordinate #8.

Mnemonic: MVP COORD, 0, 8

Binary Form of MVP COORD, 0, 8	
Field	Value
Target address	01_h
Instruction number	04_h
Type	02_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	08_h
Checksum	$0F_h$

Note Before moving to a stored coordinate, the coordinate has to be set using an SCO, CCO or ACO command.

3.6.5 SAP (Set Axis Parameter)

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

Info	For a table with parameters and values which can be used together with this command please refer to section 4.

Internal function: The specified value is written to the axis parameter specified by the parameter number.
Related commands: GAP, AAP.

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

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
5	see chapter 4	0	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example Set the maximum positioning speed for motor 0 to 51200 pps.
Mnemonic: SAP 4, 0, 51200.

Binary Form of SAP 4, 0, 51200	
Field	Value
Target address	01_h
Instruction number	05_h
Type	04_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	$C8_h$
Value (Byte 0)	00_h
Checksum	$D2_h$

3.6.6 GAP (Get Axis Parameter)

Most motion / driver related parameters of the TMCM-1278 can be adjusted using e.g. the SAP command. With the GAP parameter they can be read out. In standalone mode the requested value is also transferred to the accumulator register for further processing purposes (such as conditional jumps). In direct mode the value read is only output in the value field of the reply, without affecting the accumulator.

Info For a table with parameters and values that can be used together with this command please refer to section 4.

Internal function: The specified value gets copied to the accumulator.
Related commands: SAP, AAP.

Mnemonic: GAP <parameter number>, <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
6	see chapter 4	0	<value>

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

Example
Get the actual position of motor 0.
Mnemonic: GAP 1, 0.

Binary Form of GAP 1, 0	
Field	Value
Target address	01_h
Instruction number	06_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	08_h

3.6.7 SGP (Set Global Parameter)

With this command most of the module specific parameters not directly related to motion control can be specified and the TMCL user variables can be changed. Global parameters are related to the host interface, peripherals or application specific variables. The different groups of these parameters are organized in banks to allow a larger total number for future products. Currently, bank 0 is used for global parameters, and bank 2 is used for user variables. Bank 3 is used for interrupt configuration.
All module settings in bank 0 will automatically be stored in non-volatile memory (EEPROM).

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

Internal function: The specified value will be copied to the global parameter specified by the type and bank number. Most parameters of bank 0 will automatically be stored in non-volatile memory.
Related commands: GGP, AGP.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
9	see chapter 5	0/2/3	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

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

Binary Form of SGP 66, 0, 3	
Field	Value
Target address	01_h
Instruction number	09_h
Type	42_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	03_h
Checksum	$4F_h$

3.6.8 GGP (Get Global Parameter)

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

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

Internal function: The global parameter specified by the type and bank number will be copied to the accumulator register.
Related commands: SGP, AGP.

Mnemonic: GGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
10	see chapter 5	0/2/3	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

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

Binary Form of GGP 66, 0	
Field	Value
Target address	01_h
Instruction number	$0A_h$
Type	42_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$4D_h$

3.6.9 STGP (Store Global Parameter)

This command is used to store TMCL global parameters permanently in the EEPROM of the module. This command is mainly needed to store the TMCL user variables (located in bank 2) in the EEPROM of the module, as most other global parameters (located in bank 0) are stored automatically when being modified. The contents of the user variables can either be automatically or manually restored at power on.

Info	For a table with parameters and values which can be used together with this command please refer to dection 5.3.

Internal function: The global parameter specified by the type and bank number will be stored in the EEPROM.

Related commands: SGP, AGP, GGP, RSGP.

Mnemonic: STGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
11	see chapter 5.3	2	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Store user variable #42.
Mnemonic: STGP 42, 2.

Binary Form of STGP 42, 2	
Field	Value
Target address	01_h
Instruction number	$0B_h$
Type	$2A_h$
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	38_h

3.6.10 RSGP (Restore Global Parameter)

With this command the contents of a TMCL user variable can be restored from the EEPROM. By default, all user variables are automatically restored after power up. A user variable that has been changed before can be reset to the stored value by this instruction.

Info For a table with parameters and values which can be used together with this command please refer to section 5.3.

Internal function: The global parameter specified by the type and bank number will be restored from the EEPROM.

Related commands: SGP, AGP, GGP, STGP.

Mnemonic: RSGP <parameter number>, <bank>

Binary Representation			
Instruction	Type	Motor/Bank	Value
12	see chapter 5.3	2	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Restore user variable #42.
Mnemonic: RSGP 42, 2.

Binary Form of RSGP 42, 2	
Field	Value
Target address	01_h
Instruction number	$0C_h$
Type	$2A_h$
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	39_h

3.6.11 RFS (Reference Search)

The TMCM-1278 has a built-in reference search algorithm. The reference search algorithm provides different refrence search modes. This command starts or stops the built-in reference search algorithm. The status of the reference search can also be queried to see if it already has finished. (In a TMCL program it mostly is better to use the WAIT RFS 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 (please see chapter 4).

Internal function: The internal reference search state machine is started or stoped, or its state is queried.

Related commands: SAP, GAP, WAIT.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
13	0 START — start reference search 1 STOP — stop reference search 2 STATUS — get status	0	0 (don't care)

Reply in Direct Mode (RFS START or RFS STOP)	
Status	Value
100 - OK	0 (don't care)

Reply in Direct Mode (RFS STATUS)		
Status	Value	
100 - OK	0	no ref. search active
	other values	reference search active

Example
Start reference search of motor 0.
Mnemonic: RFS START, 0.

Binary Form of RFS START	
Field	Value
Target address	01_h
Instruction number	$0D_h$
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$0E_h$

3.6.12 SIO (Set Output)

This command sets the states of the general purpose digital outputs.

Internal function: The state of the output line specified by the type parameter is set according to the value passed to this command.

Related commands: GIO.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
14	<port number>	<bank number> (2)	0/1

Reply in Direct Mode	
Status	Value
100 - OK	0 (don't care)

Example
Set output 0 (bank 2) to high.
Mnemonic: SIO 0, 2, 1.

Binary Form of SIO 0, 2, 1	
Field	Value
Target address	01_h
Instruction number	$0E_h$
Type	00_h
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	01_h
Checksum	12_h

Bank 2 – Digital Outputs
The following output lines can be set by the SIO commands) using bank 2.

Digital Outputs in Bank 2		
Port	Command	Range
OUT0	SIO 0, 2, <value>	0/1

Special case: SIO 255, 2, <x> can be used to change all general purpose digital output lines simulaneously. The value <x> will then be interpreted as a bit vector where each bit represents one of the digital outputs. The value <x> can also be -1. In this case, the value will be taken from the accumulator register. The following program can be used to copy the states of the input lines to the output lines:

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

3.6.13 GIO (Get Input)

With this command the status of the available general purpose outputs of the module can be read. The function reads a digital or an analog input port. Digital lines will read as 0 or 1, while the ADC channels deliver their 12 bit result in the range of 0...4095. In standalone mode the requested value is copied to the accumulator register for further processing purposes such as conditional 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 state of the i/o line specified by the type parameter and the bank parameter is read.

Related commands: SIO.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
15	<port number>	<bank number> (0/1/2)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	status of the port

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

Binary Form of GIO 0, 1	
Field	Value
Target address	01_h
Instruction number	$0F_h$
Type	00_h
Motor/Bank	01_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	11_h

Reply (Status=no error, Value=302)	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	$0F_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	01_h
Value (Byte 0)	$2E_h$
Checksum	$A5_h$

Bank 0 – Digital Inputs
The analog input lines can be read as digital or analog inputs at the same time. The digital input states can be accessed in bank 0.

Digital Inputs in Bank 0		
Port	Command	Range
IN0	GIO 0, 0	0/1
IN1	GIO 1, 0	0/1
IN2	GIO 2, 0	0/1

Special case: GIO 255, 0 reads all general purpose inputs simulataneously and puts the result into the the accumulator register. The result is a bit vector where each bit represents one input.

Bank 1 – Analog Inputs
The analog input lines can be read back as digital or analog inputs at the same time. The analog values can be accessed in bank 1.

Analog Inputs in Bank 1		
Port	Command	Range / Units
IN0	GIO 0, 1	0…4095
Voltage	GIO 8, 1	[1/10V]
Temperature	GIO 9, 1	[°C]

Bank 2 – States of the Digital Outputs
The states of the output lines (that have been set by SIO commands) can be read back using bank 2.

Digital Outputs in Bank 2		
Port	Command	Range
OUT0	GIO 0, 2	0/1

3.6.14 CALC (Calculate)

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

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

Mnemonic: CALC <operation>, <operand>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
19	0 ADD – add to accumulator 1 SUB – subtract from accumulator 2 MUL – multiply accumulator by 3 DIV – divide accumulator by 4 MOD – modulo divide accumulator by 5 AND – logical and accumulator with 6 OR – logical or accumulator with 7 XOR – logical exor accumulator with 8 NOT – logical invert accumulator 9 LOAD – load operand into accumulator	0 (don't care)	<operand>

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Multiply accumulator by -5000.
Mnemonic: CALC MUL, -5000

Binary Form of CALC MUL, -5000	
Field	Value
Target address	01_h
Instruction number	13_h
Type	02_h
Motor/Bank	00_h
Value (Byte 3)	FF_h
Value (Byte 2)	FF_h
Value (Byte 1)	EC_h
Value (Byte 0)	78_h
Checksum	78_h

Reply (Status=no error, value=-5000:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	13_h
Value (Byte 3)	FF_h
Value (Byte 2)	FF_h
Value (Byte 1)	EC_h
Value (Byte 0)	78_h
Checksum	DC_h

3.6.15 COMP (Compare)

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

Internal function: The accumulator register is compared with the sepcified value. The internal arithmetic status flags are set according to the result of the comparison. These can then control e.g. a conditional jump.
Related commands: JC, GAP, GGP, GIO, CALC, CALCX.

Mnemonic: COMP <operand>

Binary Representation			
Instruction	Type	Motor/Bank	Value
20	0 (don't care)	0 (don't care)	<operand>

Example
Jump to the address given by the label when the position of motor #0 is greater than or equal to 1000.

```
GAP 1, 0 //get actual position of motor 0
COMP 1000   //compare actual value with 1000
JC GE, Label   //jump to Lable if greter or equal to 1000
```

Binary Form of COMP 1000	
Field	Value
Target address	01_h
Instruction number	14_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	03_h
Value (Byte 0)	$E8_h$
Checksum	00_h

3.6.16 JC (Jump conditional)

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

Internal function: The TMCL program counter is set to the value passed to this command if the status flags are in the appropriate states.

Related commands: JA, COMP, WAIT, CLE.

Mnemonic: JC <condition>, <label>

Binary Representation			
Instruction	Type	Motor/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	0 (don't care)	<jump address>

Example
Jump to the address given by the label when the position of motor #0 is greater than or equal to 1000.

```
GAP 1, 0  //get actual position of motor 0
COMP 1000   //compare actual value with 1000
JC GE, Label  //jump to Lable if greter or equal to 1000
...
Label: ROL 0, 1000
```

Binary form of JC GE, Label assuming Label at address 10	
Field	Value
Target address	01_h
Instruction number	15_h
Type	05_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	$0A_h$
Checksum	25_h

3.6.17 JA (Jump always)

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

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

Related commands: JC, WAIT, CSUB.

Mnemonic: JA <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
22	0 (don't care)	0 (don't care)	<jump address>

Example
An infinite loop in TMCL:

```
Loop:
    MVP ABS, 0, 51200
    WAIT POS, 0, 0
    MVP ABS, 0, 0
    WAIT POS, 0, 0
    JA Loop
```

Binary form of the JA Loop command when the label Loop is at address 10:

Binary Form of JA Loop (assuming Loop at address 10)	
Field	Value
Target address	01_h
Instruction number	16_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	$0A_h$
Checksum	21_h

3.6.18 CSUB (Call Subroutine)

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

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

Related commands: RSUB, JA.

Mnemonic: CSUB <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
23	0 (don't care)	0 (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
    CSUB SubW   //Save program counter and jump to label SubW
    JA Loop

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

Binary form of CSUB SubW (assuming SubW at address 100)	
Field	Value
Target address	01_h
Instruction number	17_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	64_h
Checksum	$7C_h$

3.6.19 RSUB (Return from Subroutine)

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

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

Related commands: CSUB.

Mnemonic: RSUB

Binary Representation			
Instruction	Type	Motor/Bank	Value
24	0 (don't care)	0 (don't care)	0 (don't care)

Example
Please see the CSUB example (section 3.6.18).
Binary form:

Binary Form of RSUB	
Field	Value
Target address	01_h
Instruction number	18_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	19_h

3.6.20 WAIT (Wait for an Event to occur)

This instruction interrupts the execution of the TMCL program until the specified condition is met. *This command is intended for standalone operation only.*
There are five different wait conditions that can be used:

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

Special case for the <ticks> parameter: When this parameter is set to -1 the contents of the accumulator register will be taken for this value. So for example WAIT TICKS, 0, -1 will wait as long as specified by the value store in the accumulator. *The accumulator must not contain a negative value when using this option.*

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 will be held at the address of this WAIT command until the condition is met or the timeout has expired.

Related commands: JC, CLE.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
27	0 TICKS – timer ticks	0 (don't care)	<no. of ticks to wait[1]>
	1 POS – target position reached	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout
	2 REFSW – reference switch	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout
	3 LIMSW – limit switch	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout
	4 RFS – reference search completed	<motor number>	<no. of ticks for timeout[1]> 0 for no timeout

Example

[1] one tick is 10 milliseconds

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

Binary Form of WAIT POS, 0, 0	
Field	Value
Target address	01_h
Instruction number	$1B_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$1D_h$

3.6.21 STOP (Stop TMCL Program Execution – End of TMCL Program)

This command stops the execution of a TMCL program. *It is intended for use in standalone operation only.*

Internal function: Execution of a TMCL program in standalone mode will be stopped.

Related commands: none.

Mnemonic: STOP

Binary Representation			
Instruction	Type	Motor/Bank	Value
28	0 (don't care)	0 (don't care)	0 (don't care)

Example
Mnemonic: STOP

Binary Form of STOP	
Field	Value
Target address	01_h
Instruction number	$1C_h$
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$1D_h$

3.6.22 SCO (Set Coordinate)

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

Note Coordinate #0 is always stored in RAM only.

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

Related commands: GCO, CCO, ACO, MVP COORD.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
30	<coordinate number> 0...20	<motor number> 0	<position> $-2^{31} \dots 2^{31} - 1$

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

Binary Form of SCO 1, 0, 1000	
Field	Value
Target address	01_h
Instruction number	$1E_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	03_h
Value (Byte 0)	$E8_h$
Checksum	$0B_h$

Two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate to the EEPROM. These functions can be accessed using the following special forms of the SCO command:

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

3.6.23 GCO (Get Coordinate)

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

Note Coordinate #0 is always stored in RAM only.

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

Related commands: SCO, CCO, ACO, MVP COORD.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
31	<coordinate number> 0...20	<motor number> 0	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

Example
Get coordinate #1 of motor #0.
Mnemonic: GCO 1, 0

Binary Form of GCO 1, 0	
Field	Value
Target address	01_h
Instruction number	$1F_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	21_h

Two special functions of this command have been introduced that make it possible to copy all coordinates or one selected coordinate from the EEPROM to the RAM.
These functions can be accessed using the following special forms of the GCO command:

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

3.6.24 CCO (Capture Coordinate)

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

Note	Coordinate #0 is always stored in RAM only.

Internal function: the actual position of the selected motor is copied to selected coordinate array entry.

Related commands: SCO, GCO, ACO, MVP COORD.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
32	<coordinate number> 0…20	<motor number> 0	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	value read by this command

Example
Store current position of motor #0 to coordinate array entry #3.

Mnemonic: CCO 3, 0

Binary Form of CCO 3, 0	
Field	Value
Target address	01_h
Instruction number	20_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	22_h

3.6.25 ACO (Accu to Coordinate)

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

Note	Coordinate #0 is always stored in RAM only.

Internal function: the actual position of the selected motor is copied to selected coordinate array entry.

Related commands: SCO, GCO, CO, MVP COORD.

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
39	<coordinate number> 0...20	<motor number> 0	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Copy the actual value of the accumulator to coordinate #1 of motor #0.

Mnemonic: ACO 1, 0

Binary Form of ACO 1, 0	
Field	Value
Target address	01_h
Instruction number	27_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	29_h

3.6.26 CALCX (Calculate using the X Register)

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

Related commands: CALC, COMP, JC, AAP, AGP, GAP, GGP, GIO.

Mnemonic: CALCX <operation>

Binary Representation			
Instruction	Type	Motor/Bank	Value
33	0 ADD – add X register to accumulator 1 SUB – subtract X register from accumulator 2 MUL – multiply accumulator by X register 3 DIV – divide accumulator by X register 4 MOD – modulo divide accumulator by X register 5 AND – logical and accumulator with X register 6 OR – logical or accumulator with X register 7 XOR – logical exor accumulator with X register 8 NOT – logical invert X register 9 LOAD – copy accumulator to X register 10 SWAP – swap accumulator and X register	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Multiply accumulator and X register.
Mnemonic: CALCX MUL

Binary Form of CALCX MUL	
Field	Value
Target address	01_h
Instruction number	21_h
Type	02_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	24_h

3.6.27 AAP (Accu to Axis Parameter)

The content of the accumulator register is transferred to the specified axis parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction. *This command is mainly intended for use in standalone mode.*

Info	For a table with parameters and values which can be used together with this command please refer to section 4.

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

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
34	see chapter 4	0	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Position motor #0 by a potentiometer connected to analog input #0:

```
Start:
    GIO 0,1       //get value of analog 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 Form of AAP 0, 0	
Field	Value
Target address	01_h
Instruction number	22_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	23_h

3.6.28 AGP (Accu to Global Parameter)

The content of the accumulator register is transferred to the specified global parameter. For practical usage, the accumulator has to be loaded e.g. by a preceding GAP instruction. The accumulator may have been modified by the CALC or CALCX (calculate) instruction. *This command is mainly intended for use in standalone mode.*

Info	For an overview of parameter and bank indices that can be used with this command please see section 5.

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

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

Binary Representation			
Instruction	Type	Motor/Bank	Value
35	<parameter number>	0/2/3 <bank number>	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Copy accumulator to user variable #42:
Mnemonic: AGP 42, 2

Binary Form of AGP 42, 2	
Field	Value
Target address	01_h
Instruction number	23_h
Type	$2A_h$
Motor/Bank	02_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	50_h

3.6.29 CLE (Clear Error Flags)

This command clears the internal error flags. It is mainly intended for use in standalone mode.
The following error flags can be cleared by this command (determined by the <flag> parameter):

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

Related commands: JC, WAIT.

Mnemonic: CLE <flags>

Binary Representation			
Instruction	Type	Motor/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	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Reset the timeout flag.
Mnemonic: CLE ETO

Binary Form of CLE ETO	
Field	Value
Target address	01_h
Instruction number	24_h
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	26_h

3.6.30 EI (Enable Interrupt)

The EI command enables an interrupt. It needs the interrupt number as parameter. Interrupt number 255 globally enables interrupt processing. *This command is mainly intended for use in standalone mode.*

Info	Please see table 12 for a list of interrupts that can be used on the TMCM-1278 module.

Related commands: DI, VECT, RETI.

Mnemonic: EI <interrupt number>

Binary Representation			
Instruction	Type	Motor/Bank	Value
25	<interrupt number>	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Globally enable interrupt processing:
Mnemonic: EI 255

Binary form of EI 255	
Field	Value
Target address	01_h
Instruction number	19_h
Type	FF_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	19_h

3.6.31 DI (Disable Interrupt)

The DI command disables an interrupt. It needs the interrupt number as parameter. Interrupt number 255 globally disables interrupt processing. *This command is mainly intended for use in standalone mode.*

Info	Please see table 12 for a list of interrupts that can be used on the TMCM-1278 module.

Related commands: EI, VECT, RETI.

Mnemonic: DI <interrupt number>

Binary Representation			
Instruction	Type	Motor/Bank	Value
26	<interrupt number>	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Globally disable interrupt processing:
Mnemonic: DI 255

Binary Form of DI 255	
Field	Value
Target address	01_h
Instruction number	$1A_h$
Type	FF_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$1A_h$

3.6.32 VECT (Define Interrupt Vector)

The VECT command defines an interrupt vector. It takes an interrupt number and a label (just like with JA, JC and CSUB commands) as parameters. The label must be the entry point of the interrupt handling routine for this interrupts. Interrupt vectors can also be re-defined. *This command is intended for use in standalone mode only.*

Info	Please see table 12 for a list of interrupts that can be used on the TMCM-1278 module.

Related commands: EI, DI, RETI.

Mnemonic: VECT <interrupt number>, <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
37	<interrupt number>	0 (don't care)	<label>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Define interrupt vector for timer #0 interrupt:

```
    VECT 0, Timer0Irq
    ...
Loop:
    ...
    JA Loop
    ...
Timer0Irq:
    SIO 0, 2, 1
    RETI
```

Binary form of VECT (assuming label is at 50)	
Field	Value
Target address	01_h
Instruction number	25_h
Type	FF_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	32_h
Checksum	58_h

3.6.33 RETI (Return from Interrupt)

This command terminates an interrupt handling routine. Normal program flow will be continued then. *This command is intended for use in standalone mode only.*

An interrupt routine must always end with a RETI command. Do not allow the normal program flow to run into an interrupt routine.

Internal function: The saved registers (accumulator, X registers, flags and program counter) are copied back so that normal program flow will continue.

Related commands: EI, DI, VECT.

Mnemonic: RETI

Binary Representation			
Instruction	Type	Motor/Bank	Value
38	<interrupt number>	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Return from an interrup handling routine.
Mnemonic: RETI

Binary Form of RETI	
Field	Value
Target address	01_h
Instruction number	26_h
Type	FF_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	27_h

3.6.34 CALCVV (Calculate using two User Variables)

The CALCVV instruction directly uses the contents of two user variables for an arithmetic operation, storing the result in the first user variable. This eliminates the need for using the accumulator register and/or X register for such purposes. The parameters of this command are the arithmetic function, the index of the first user variable (0...255) and the index of the second user variable (0...255). *This command is mainly intended for use in standalone mode.*

Related commands: CALCVA, CALCAV, CALCVX, CALCXV, CALCV.

Mnemonic: CALCVV <operation>, <var1>, <var2>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
40	0 ADD – add <var2> to <var1> 1 SUB – subtract <var2> from <var1> 2 MUL – multiply <var2> with <var1> 3 DIV – divide <var2> by <var1> 4 MOD – modulo divide <var2> by <var1> 5 AND – logical and <var2> with <var1> 6 OR – logical or <var2> with <var1> 7 XOR – logical exor <var2> with <var1> 8 NOT – copy logical inverted <var2> to <var1> 9 LOAD – copy <var2> to <var1> 10 SWAP – swap contents of <var1> and <var2> 11 COMP – compare <var1> with <var2>	0 <var1> (0...255)	<var2> (0...255)

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Subtract user variable #42 from user variable #65.
Mnemonic: CALCVV SUB, 65, 42

Binary Form of CALCVV SUB, 65, 42	
Field	Value
Target address	01_h
Instruction number	28_h
Type	01_h
Motor/Bank	41_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	$2A_h$
Checksum	95_h

Reply (Status=no error, value=0:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	28_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$8F_h$

3.6.35 CALCVA (Calculate using a User Variable and the Accumulator Register)

The CALCVA instruction directly modifies a user variable using an arithmetical operation and the contents of the accumulator register. The parameters of this command are the arithmetic function and the index of a user variable (0...255). *This command is mainly intended for use in standalone mode.*

Related commands: CALCV, CALCAV, CALCVX, CALCXV, CALCVV.

Mnemonic: CALCVA <operation>, <var>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
41	0 ADD – add accumulator to <var> 1 SUB – subtract accumulator from <var> 2 MUL – multiply <var> with accumulator 3 DIV – divide <var> by accumulator 4 MOD – modulo divide <var> by accumulator 5 AND – logical and <var> with accumulator 6 OR – logical or <var> with accumulator 7 XOR – logical exor <var> with accumulator 8 NOT – copy logical inverted accumulator to <var> 9 LOAD – copy accumulator to <var> 10 SWAP – swap contents of <var> and accumulator 11 COMP – compare <var> with accumulator	0 <var> (0...255)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Subtract accumulator from user variable #27.
Mnemonic: CALCVA SUB, 27

Binary Form of CALCVA SUB, 27	
Field	Value
Target address	01_h
Instruction number	29_h
Type	01_h
Motor/Bank	$1B_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	46_h

Reply (Status=no error, value=0:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	29_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	90_h

3.6.36 CALCAV (Calculate using the Accumulator Register and a User Variable)

The CALCAV instruction modifies the accumulator register using an arithmetical operation and the contents of a user variable. The parameters of this command are the arithmetic function and the index of a user variable (0...255). *This command is mainly intended for use in standalone mode.*

Related commands: CALCV, CALCAV, CALCVX, CALCXV, CALCVV.

Mnemonic: CALCAV <operation>, <var>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
42	0 ADD – add <var> to accumulator 1 SUB – subtract <var> from accumulator 2 MUL – multiply accumulator with <var> 3 DIV – divide accumulator by <var> 4 MOD – modulo divide accumulator by <var> 5 AND – logical and accumulator with <var> 6 OR – logical or accumulator with <var> 7 XOR – logical exor accumulator with <var> 8 NOT – copy logical inverted <var> to accumulator 9 LOAD – copy <var> to accumulator 10 SWAP – swap contents of <var> and accumulator 11 COMP – compare accumulator with <var>	0 <var> (0...255)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Subtract user variable #27 from accumulator.
Mnemonic: CALCXV SUB, 27

Binary Form of CALCXV SUB, 27	
Field	Value
Target address	01_h
Instruction number	$2A_h$
Type	01_h
Motor/Bank	$1B_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	47_h

Reply (Status=no error, value=0:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	$2A_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	91_h

3.6.37 CALCVX (Calculate using a User Variable and the X Register)

The CALCVX instruction directly modifies a user variable using an arithmetical operation and the contents of the X register. The parameters of this command are the arithmetic function and the index of a user variable (0...255). *This command is mainly intended for use in standalone mode.*

Related commands: CALCV, CALCAV, CALCVA, CALCXV, CALCVV.

Mnemonic: CALCVX <operation>, <var>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
43	0 ADD – add X register to <var> 1 SUB – subtract X register from <var> 2 MUL – multiply <var> with X register 3 DIV – divide <var> by X register 4 MOD – modulo divide <var> by X register 5 AND – logical and <var> with X register 6 OR – logical or <var> with X register 7 XOR – logical exor <var> with X register 8 NOT – copy logical inverted X register to <var> 9 LOAD – copy X register to <var> 10 SWAP – swap contents of <var> and X register 11 COMP – compare <var> with X register	0 <var> (0...255)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Subtract X register from user variable #27.
Mnemonic: CALCVX SUB, 27

Binary Form of CALCVX SUB, 27	
Field	Value
Target address	01_h
Instruction number	$2B_h$
Type	01_h
Motor/Bank	$1B_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	48_h

Reply (Status=no error, value=0:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	$2B_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	92_h

3.6.38 CALCXV (Calculate using the X Register and a User Variable)

The CALCXV instruction modifies the X register using an arithmetical operation and the contents of a user variable. The parameters of this command are the arithmetic function and the index of a user variable (0...255). *This command is mainly intended for use in standalone mode.*

Related commands: CALCV, CALCAV, CALCVA, CALCVX, CALCVV.

Mnemonic: CALCXV <operation>, <var>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
44	0 ADD – add <var> to X register 1 SUB – subtract <var> from X register 2 MUL – multiply X register with <var> 3 DIV – divide X register by <var> 4 MOD – modulo divide X register by <var> 5 AND – logical and X register with <var> 6 OR – logical or X register with <var> 7 XOR – logical exor X register with <var> 8 NOT – copy logical inverted <var> to X register 9 LOAD – copy <var> to X register 10 SWAP – swap contents of <var> and X register 11 COMP – compare X register with <var>	0 <var> (0...255)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Subtract user variable #27 from X register.
Mnemonic: CALCXV SUB, 27

Binary Form of CALCXV SUB, 27	
Field	Value
Target address	01_h
Instruction number	$2C_h$
Type	01_h
Motor/Bank	$1B_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	49_h

Reply (Status=no error, value=0:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	$2C_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	93_h

3.6.39 CALCV (Calculate using a User Variable and a Direct Value)

The CALCV directly modifies a user variable using an arithmetical operation and a direct value. This eliminates the need of using the accumulator register for such a purpose and thus can make the program shorter and faster. The parameters of this command are the arithmetic function, the index of a user variable (0…255) and a direct value. *This command is mainly intended for use in standalone mode.*

Related commands: CALCVA, CALCAV, CALCVX, CALCXV, CALCVV.

Mnemonic: CALCV <operation>, <var>, <value>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
45	0 ADD – add <value> to <var> 1 SUB – subtract <value> from <var> 2 MUL – multiply <var> with <value> 3 DIV – divide <var> by <value> 4 MOD – modulo divide <var> by <value> 5 AND – logical and <var> with <value> 6 OR – logical or <var> with <value> 7 XOR – logical exor <var> with <value> 8 NOT – logical invert <var> (<value> ignored) 9 LOAD – copy <value> to <var> 11 COMP – compare <var> with <value>	0 <var> (0…255)	<value>

Reply in Direct Mode	
Status	Value
100 - OK	the operand (don't care)

Example
Subtract 5000 from user variable #27.
Mnemonic: CALCV SUB, 27, 5000

Binary Form of CALCV SUB, 27, 5000	
Field	Value
Target address	01_h
Instruction number	$2D_h$
Type	01_h
Motor/Bank	$1B_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	13_h
Value (Byte 0)	88_h
Checksum	$E5_h$

Reply (Status=no error, value=5000:	
Field	Value
Host address	02_h
Target address	01_h
Status	64_h
Instruction	$2D_h$
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	13_h
Value (Byte 0)	88_h
Checksum	$2F_h$

3.6.40 RST (Restart)

Stop the program, reset the TMCL interpreter and then re-start the program at the given label. This command can be used to re-start the TMCL program from anywhere in the program, also out of subroutines or interrupt rotuines. *This command is intended for standalone operation only.*

Internal function: The TMCL interpreter is reset (the subroutine stack, the interrupt stack and the registers are cleared) and then the program counter is set to the value passed to this command.

Related commands: JA, CSUB, STOP.

Mnemonic: RST <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
48	0 (don't care)	0 (don't care)	<restart address>

Example
Restart the program from a label, out of a subroutine:

```
Entry:
    MVP ABS, 0, 51200
    CSUB Subroutine
    ...
    ...
Subroutine:
    RST Entry
    RSUB
```

Binary form of the RST Entry command when the label Entry is at address 10:

Binary Form of RST Entry (assuming Entry at address 10)	
Field	Value
Target address	01_h
Instruction number	30_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	$0A_h$
Checksum	$3A_h$

3.6.41 DJNZ (Decrement and Jump if not Zero)

Decrement a given user variable and jump to the given address if the user variable is greater than zero. This command can for example be used to easily program a counting loop, using any user variable as the loop counter. *This command is intended for standalone operation only.*

Internal function: The user variable passed to this command is decremented. If it is not zero then the TMCL program counter is set to the value passed to this command.

Related commands: JC, WAIT, CSUB.

Mnemonic: DJNZ <var>, <label>

Binary Representation			
Instruction	Type	Motor/Bank	Value
49	<user variable> (0…255)	0 (don't care)	<jump address>

Example
A counting loop in TMCL, using user variable #42:

```
    SGP 42, 2, 100
Loop:
    MVP ABS, REL, 51200
    WAIT POS, 0, 0
    WAIT TICKS, 0, 500
    DJNZ 42, Loop
```

Binary form of the DJNZ 42, Loop command when the label Loop is at address 1:

Binary Form of DJNZ Loop (assuming Loop at address 1)	
Field	Value
Target address	01_h
Instruction number	31_h
Type	64_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	01_h
Checksum	97_h

3.6.42 CALL (Conditional Subroutine Call)

The CALL command calls a subroutine in the TMCL program, but only if the specifed condition is met. Otherwise the program execution will be continued with the next command following the CALL command. The conditions refer to the result of a preceding comparison or assignment. *This command is intended for standalone operation only.*

Internal function: When the condition is met the actual TMCL program counter value will be saved to an internal stack. Afterwards the program counter will be overwritten with the address supplied to this command. 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, JC.

Mnemonic: CALL <condition>, <label>

Binary Representation			
Instruction	Type	Motor/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	0 (don't care)	<jump address>

Example
Call a subroutine if a condition is met:

```
Loop:
    GIO 0, 1              //read analog value
    CALC SUB, 512         //subtract 512
    COMP 0                //compare with zero
    CALL LT, RunLeft      //Call routine "RunLeft" if accu<0
    CALL ZE, MotorStop    //Call routine "MotosStop" if accu=0
    CALL GT, RunRight     //Call routine "RunRight" if accu>0
    JA Loop

RunLeft:
    CALC MUL, -1
    ROLA 0
    RSUB
```

```
14
RunRight:
16    RORA 0
      RSUB
18
MotorStop:
20    GAP 2, 0
      JC ZE, MotorIsStopped
22    MST 0
MotorIsStopped:
24    RSUB
```

Binary form of CALL LT, RunLeft (assuming RunLeft at address 100)	
Field	Value
Target address	01_h
Instruction number	50_h
Type	06_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	64_h
Checksum	BB_h

3.6.43 MVPA (Move to Position specified by Accumulator Register)

With this command the motor will be instructed to move to a specified relative or absolute position. The contents of the accumulator register will be used as the target position. This command is non-blocking which means that 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 as well as other ramp parameters are defined by the appropriate axis parameters. For a list of these parameters please refer to section 4.
Positioning can be interrupted using MST, ROL or ROR commands.

Three operation types are available:

- Moving to an absolute position specified by the accumulator register contents.
- Starting a relative movement by means of an offset to the actual position.
- Moving the motor to a (previously stored) coordinate (refer to SCO for details).

Note The distance between the actual position and the new position must not be more than 2147483647 ($2^{31} - 1$) microsteps. Otherwise the motor will run in the opposite direction in order to take the shorter distance (caused by 32 bit overflow).

Internal function: The value stored in the accumulator register is copied to the axis parameter #0 (target position).

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

Mnemonic: MVPA <ABS|REL|COORD>, <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
46	0 – ABS – absolute	0	0 (don't care)
	1 – REL – relative	0	0 (don't care)
	2 – COORD – coordinate	0…255	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Move motor 0 to position specified by accumulator.
Mnemonic: MVPA ABS, 0

Binary Form of MVPA ABS, 0	
Field	Value
Target address	01_h
Instruction number	$2E_h$
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$2F_h$

3.6.44 ROLA (Rotate Left using the Accumulator Register)

Rotate in left direction (decreasing the position counter) using the velocity specified by the contents of the accumulator register. The velocity is given in microsteps per second (pulse per second [pps]).

Internal function:

- First, velocity mode is selected.
- Then, the content of the accumulator is copied to axis parameter #2 (target velocity).

Related commands: RORA, MST, SAP, GAP.

Mnemonic: ROLA <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
50	0 (don't care)	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Rotate left motor 0, velocity specified by accumulator.
Mnemonic: ROLA 0.

Binary Form of ROLA 0	
Field	Value
Target address	01_h
Instruction number	32_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	33_h

3.6.45 RORA (Rotate Right using the Accumulator Register)

Rotate in right direction (increasing the position counter) using the velocity specified by the contents of the accumulator register. The velocity is given in microsteps per second (pulse per second [pps]).

Internal function:

- First, velocity mode is selected.
- Then, the content of the accumulator is copied to axis parameter #2 (target velocity).

Related commands: ROLA, MST, SAP, GAP.

Mnemonic: ROLA <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
51	0 (don't care)	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Rotate right motor 0, velocity specified by accumulator.
Mnemonic: RORA 0.

Binary Form of RORA 0	
Field	Value
Target address	01_h
Instruction number	33_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	33_h

3.6.46 SIV (Set Indexed Variable)

This command copies a direct value to a TMCL user variable. The index of the user variable (0…255) is specified by the content of the X register. Therefore the value in the X register must not be lower than zero or greater than 255. Otherwise this command will be ignored. *This command is mainly intended for use in standalone mode.*

Internal function: The direct value supplied to this command will be copied to the user variable specified by the X register.

Related commands: AIV, GIV.

Mnemonic: SIV

Binary Representation			
Instruction	Type	Motor/Bank	Value
55	0 (don't care)	0 (don't care)	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Copy the value 3 to the user variable indexed by the X register.
Mnemonic: SIV 3.

Binary Form of SIV 3	
Field	Value
Target address	01_h
Instruction number	37_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	03_h
Checksum	$3B_h$

3.6.47 GIV (Get Indexed Variable)

This command reads a TMCL user variable and copies its content to the accumulator register. The index of the user variable (0...255) is specified by the X register. Therefore the content of the X register must not be lower than zero or greater than 255. Otherwise this command will be ignored. *This command is mainly intended for use in standalone mode.*

Internal function: The user variable specified by the x register will be copied to the accumulator register.

Related commands: SIV, AIV.

Mnemonic: GIV

Binary Representation			
Instruction	Type	Motor/Bank	Value
55	0 (don't care)	0 (don't care)	0 (don't care)

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Read the user variable indexed by the X register.
Mnemonic: GIV.

Binary Form of GIV	
Field	Value
Target address	01_h
Instruction number	38_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	03_h
Checksum	39_h

3.6.48 AIV (Accumulator to Indexed Variable)

This command copies the content of the accumulator to a TMCL user variable. The index of the user variable (0...255) is specified by the content of the X register. Therefore the value in the X register must not be lower than zero or greater than 255. Otherwise this command will be ignored. *This command is mainly intended for use in standalone mode.*

Internal function: The accumulator will be copied to the user variable specified by the X register.

Related commands: SIV, GIV.

Mnemonic: AIV

Binary Representation			
Instruction	Type	Motor/Bank	Value
55	0 (don't care)	0 (don't care)	<value>

Reply in Direct Mode	
Status	Value
100 - OK	don't care

Example
Copy the accumulator to the user variable indexed by the X register.
Mnemonic: AIV.

Binary Form of AIV	
Field	Value
Target address	01_h
Instruction number	39_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	00_h
Checksum	$3A_h$

3.6.49 Customer specific Command Extensions (UF0...UF7 – User Functions)

These commands are used for customer specific extensions of TMCL. They will be implemented in C by Trinamic. Please contact the sales department of Trinamic Motion Control GmbH & Co KG if you need a customized TMCL firmware.

Related commands: none.

Mnemonic: UF0...UF7

Binary Representation			
Instruction	Type	Motor/Bank	Value
64...71	<user defined>	0 <user defined>	0 <user defined>

Reply in Direct Mode	
Status	Value
100 - OK	user defined

3.6.50 Request Target Position reached Event

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

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

Related commands: none.

Binary Representation			
Instruction	Type	Motor/Bank	Value
138	0/1	0 (don't care)	always 1

With command 138 the value field is a bit vector. It shows for which motors one would like to have a position reached message. The value field contains a bit mask where every bit stands for one motor. For one motor modules like the TMCM-1278 it only makes sense to have bit 0 set. So, always set this parameter to 1 with the TMCM-1278 module. With the type field set to 0, only for the next MVP command that follows this command a position reached message will be generated. With type set to 1 a position reached message will be generated for every MVP command that follows this command. It is recommended to use the latter option.

Example
Get a target position reached message for each MVP command that follows.

Binary Form for this example	
Field	Value
Target address	01_h
Instruction number	$8A_h$
Type	01_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	01_h
Checksum	$8D_h$

Reply in Direct Mode	
Field	Value
Target address	01_h
Host address	02_h
Status	64_h (100)
Command	$8A_h$ (138)
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	Motor bit mask
Checksum	depends also on motor bit mask

Additional Reply after Motor has reached Target Position	
Field	Value
Target address	01_h
Host address	02_h
Status	80_h (128)
Command	$8A_h$ (138)
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	Motor bit mask
Checksum	depends also on motor bit mask

3.6.51 TMCL Control Commands

There is a set of TMCL commands which are called TMCL control commands. These commands can only be used in direct mode and not in a standalone program. For this reason they only have opcodes, but no mnemonics. Most of these commands are only used by the TMCL-IDE (in order to implement e.g. the debugging functions in the TMCL creator). Some of them are also interesting for use in custom host applications, for example to start a TMCL routine on a module, when combining direct mode and standalone mode (please see also section 7.6. The following table lists all TMCL control commands.

The motor/bank parameter is not used by any of these functions and thus is not listed in the table. It should always be set to 0 with these commands.

TMCL Control Commands			
Instruction	Description	Type	Value
128 – stop application	stop a running TMCL application	0 (don't care)	0 (don't care)
129 – run application	start or continue TMCL program execution	0 – from current address 1 – from specific address	0 (don't care) starting address
130 – step application	execute only the next TMCL command	0 (don't care)	0 (don't care)
131 – reset application	Stop a running TMCL program. Reset program counter and stack pointer to zero. Reset accumulator and X register to zero. Reset all flags.	0 (don't care)	0 (don't care)
132 – enter download mode	All following commands (except control commands) are not executed but stored in the TMCL memory.	0 (don't care)	start address for download
133 – exit download mode	End the download mode. All following commands are executed normally again.	0 (don't care)	0 (don't care)
134 – read program memory	Return contents of the specified program memory location (special reply format).	0 (don't care)	address of memory location

Instruction	Description	Type	Value
135 – get application status	Return information about the current status, depending on the type field.	0 - return mode, wait flag, memory pointer 1 - return mode, wait flag, program counter 2 - return accumulator 3 - return X register	0 (don't care)
136 – get firmware version	Return firmware version in string format (special reply) or binary format).	0 - string format 1 - binary format	0 (don't care)
137 – restore factory settings	Reset all settings in the EEPROM to their factory defaults. This command does not send a reply.	0 (don't care)	set to 1234
255 – software reset	Restart the CPU of the module (like a power cycle). The reply of this command might not always get through.	0 (don't care)	set to 1234

Table 14: TMCL Control Commands

Especially the commands 128, 129, 131, 136 and 255 are interesting for use in custom host applications. The other control commands are mainly being used by the TMCL-IDE.

4 Axis Parameters

Most motor controller features of the TMCM-1278 module are controlled by axis parameters. Axis parameters can be modified or read using SAP, GAP and AAP commands. This chapter describes all axis parameters that can be used on the TMCM-1278 module.

<table>
<tr><th colspan="5">All Axis Parameters of the TMCM-1278 Module</th></tr>
<tr><th>Number</th><th>Axis Parameter</th><th>Description</th><th>Range [Units]</th><th>Access</th></tr>
<tr><td>0</td><td>Target position</td><td>The desired target position in position mode</td><td>-2147483648 ...2147483647 [µsteps]</td><td>RW</td></tr>
<tr><td>1</td><td>Actual position</td><td>The actual position of the motor. Stop the motor before overwriting it. Should normally only be overwritten for reference position setting.</td><td>-2147483648 ...2147483647 [µsteps]</td><td>RW</td></tr>
<tr><td>2</td><td>Target speed</td><td>The desired speed in velocity mode. Not valid in position mode.</td><td>-7999774 ...7999774 [pps]</td><td>RW</td></tr>
<tr><td>3</td><td>Actual speed</td><td>The actual speed of the motor.</td><td>-7999774 ...7999774 [pps]</td><td>R</td></tr>
<tr><td>4</td><td>Maximum positioning speed</td><td>The maximum speed used for positioning ramps.</td><td>0...7999774 [pps]</td><td>RW</td></tr>
<tr><td>5</td><td>Maximum acceleration</td><td>Maximum acceleration in positioning ramps. Acceleration and deceleration value in velocity mode.</td><td>0...7629278 [pps²]</td><td>RW</td></tr>
<tr><td>6</td><td>Maximum current</td><td>Motor current used when motor is running. The maximum value is 255 which means 100% of the maximum current of the module.
The current can be adjusted in 32 steps:
0...7 | 79...87 | 160...167 | 240...247
8...15 | 88...95 | 168...175 | 248...255
16...23 | 96...103 | 176...183
24...31 | 104...111 | 184...191
32...39 | 112...119 | 192...199
40...47 | 120...127 | 200...207
48...55 | 128...135 | 208...215
56...63 | 136...143 | 216...223
64...71 | 144...151 | 224...231
72...79 | 152...159 | 232...239
The most important setting, as too high values can cause motor damage.</td><td>0...255</td><td>RW</td></tr>
</table>

Number	Axis Parameter	Description	Range [Units]	Access
7	Standby current	The current used when the motor is not running. The maximum value is 255 which means 100% of the maximum current of the module. This value should be as low as possible so that the motor can cool down when it is not moving. Please see also parameter 214.	0…255	RW
8	Position reached flag	This flag is always set when target position and actual position are equal.	0/1	R
9	Home switch state	The logical state of the home switch input.	0/1	R
10	Right limit switch state	The logical state of the right limit switch input.	0/1	R
11	Left limit switch state	The logical state of the left limit switch input.	0/1	R
12	Right limit switch disable	Deactivates the stop function of the right limit switch if set to 1.	0/1	RW
13	Left limit switch disable	Deactivates the stop function of the left limit switch if set to 1.	0/1	RW
14	Swap limit switches	Swap the left and right limit switches when set to 1.	0/1	RW
15	Acceleration A1	First acceleration between VSTART and V1 (in position mode only).	0…7629278 [pps^2]	RW
16	Velocity V1	First acceleration / decelaration phase target velocizy (in position mode only). Setting this value to 0 turns off the first acceleration / deceleration phase, maximum acceleration (axis parameter 5) and maximum decleration (axis parameter 17) are used only.	0…1000000 [pps]	RW
17	Maximum deceleration	Maximum deceleration in positioning ramps. Used to decelerate from maximum positiong speed (axis parameter 4) to velocity V1.	0…7629278 [pps^2]	RW
18	Deceleration D1	Deceletation bewteen V1 and VSTOP (in positioning mode only).	0…7629278 [pps^2]	RW
19	Velocity VSTART	Motor start velocity (in position mode only). Do not set VSTART higher than VSTOP.	0…249999 [pps]	RW
20	Velocity VSTOP	Motor stop velocity (in position mode only).	0…249999 [pps]	RW
21	Ramp wait time	Defines the waiting time after ramping down to zero velocity before next movement or direction inversion can start. Time range is 0 to 2 seconds. This setting avoids excess acceleration e.g. from VSTOP to -VSTART.	0…65535 [0.000032s]	RW

Number	Axis Parameter	Description	Range [Units]	Access
22	Speed threshold for CoolStep / fullstep	Speed threshold for de-activating CoolStep or switching to fullstep mode.	0...7999774 [pps]	RW
23	Minimum speed for DcStep	Minimum speed for switching to DcStep	0...7999774 [pps]	RW
24	Right limit switch polarity	Setting this parameter to 1 inverts the logic state of the right limit switch input.	0/1	RW
25	Left limit switch polarity	Setting this parameter to 1 inverts the logic state of the left limit switch input.	0/1	RW
26	Soft stop enable	Use soft stop when motor is stopped by a limit switch.	0/1	RW
27	High speed chopper mode	Switch to other chopper mode when measured speed is higher than axis parameter 22 when set to 1.	0/1	RW
28	High speed fullstep mode	Switch to fullstep mode when measured speed is higher than axis parameter 22 when set ot 1.	0/1	RW
29	Measured speed	Speed measured by the motor driver.	0...7999774 [pps]	R
31	Power down ramp	Controls the number of clock cycles for motor power down after a motion as soon as the motor has stopped and the setting time has expired. The smooth transition avoids a motor jerk upon power down. 0=instant power down, 15=longest possible power down ramp.	0...15 [0.16384s]	RW
32	DcStep time	This setting controls the reference pulse width for DcStep load measurement. It must be optimized for robust operation with maximum motor torque. A higher value allows higher torque and higher velocity, a lower value allows operation down to a lower velocity as set by axis parameter #23.	0...1023	RW
33	DcStep StallGuard	This setting controls stall detection in DcStep mode. Increase for higher sensitivity.	0...255	RW
127	Relative positioning option	Start position for MVP REL command: 0 | last target position 1 | actual position	0/1	RW

Number	Axis Parameter	Description	Range [Units]	Access
140	Microstep resolution	Microstep resolutions per full step: 0 fullstep 1 halfstep 2 4 microsteps 3 8 microsteps 4 16 microsteps 5 32 microsteps 6 64 microsteps 7 128 microsteps 8 256 microsteps	0..8	RW
162	Chopper blank time	Selects the comparator blank time. This time needs to safely cover the switching event and the duration of the ringing on the sense resistor. Normally leave at the default value.	0..3	RW
163	Constant TOff mode	Selection of the chopper mode: 0 – spread cycle 1 – classic constant off time	0/1	RW
164	Disable fast decay comperator	See parameter 163. For ”classic const. off time” setting this parameter to ”1” will disable current comparator usage for termination of fast decay cycle.	0/1	RW
165	Chopper hysteresis end / fast decay time	See parameter 163. For ”spread cycle” chopper mode this parameter will set / return the hysteresis end setting (hysteresis end value after a number of decrements). For ”classic const. off time” chopper mode this parameter will set / return the fast decay time.	0…15	RW
166	Chopper hysteresis start / sine wave offset	See parameter 163. For ”spread cycle” chopper mode this parameter will set / return the Hysteresis start setting (please note that this value is an offset to the hysteresis end value). For ”classic const. off time” chopper mode this parameter will set / return the sine wave offset.	0…8	RW
167	Chopper off time (TOff)	The off time setting controls the minimum chopper frequency. An off time within the range of 5μs to 20μs will fit. Off time setting for constant t Off chopper: $N_{CLK} = 12 + 32 * tOFF$ (Minimum is 64 clocks) Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel.	0…15	RW

Number	Axis Parameter	Description	Range [Units]	Access
168	SmartEnergy current minimum (SEIMIN)	Sets the lower motor current limit for CoolStep operation by scaling the maximum current (see axis parameter 6) value. Minimum motor current: 0 - $\frac{1}{2}$ of CS 1 - $\frac{1}{4}$ of CS	0/1	RW
169	SmartEnergy current down step	Sets the number of StallGuard2 readings above the upper threshold necessary for each current decrement of the motor current. Number of StallGuard2 measurements per decrement: Scaling: 0…3: 32, 8, 2, 1 0: slow decrement 3: fast decrement	0…3	RW
170	SmartEnergy hysteresis	Sets the distance between the lower and the upper threshold for StallGuard2 reading. Above the upper threshold the motor current becomes decreased. Hysteresis: $([AP172]+1)*32$ Upper StallGuard threshold: $([AP172]+[AP170]+1)*32$	0…15	RW
171	SmartEnergy current up step	Sets the current increment step. The current becomes incremented for each measured StallGuard2 value below the lower threshold see SmartEnergy hysteresis start). Current increment step size: Scaling: 0…3: 1, 2, 4, 8 0: slow increment 3: fast increment / fast reaction to rising load	0…3	RW
172	SmartEnergy hysteresis start	The lower threshold for the StallGuard2 value (see SmartEnergy current up step). **Setting this to 0 (default) turns off CoolStep.**	0..15	RW
173	StallGuard2 filter enable	Enables the StallGuard2 filter for more precision of the measurement. If set, reduces the measurement frequency to one measurement per four fullsteps. In most cases it is expedient to set the filtered mode before using CoolStep. Use the standard mode for step loss detection. 0 - standard mode 1 - filtered mode	0/1	RW
174	StallGuard2 threshold	This signed value controls StallGuard2 threshold level for stall output and sets the optimum measurement range for readout. A lower value gives a higher sensitivity. Zero is the starting value. A higher value makes StallGuard2 less sensitive and requires more torque to indicate a stall.	-64…+63	RW

Number	Axis Parameter	Description	Range [Units]	Access
180	SmartEnergy actual current	This status value provides the actual motor current setting as controlled by CoolStep. The value goes up to the CS value and down to the portion of CS as specified by SEIMIN. Actual motor current scaling factor: 0…31: 1/32, 2/32, …32/32	0…31	R
181	Stop on stall	Below this speed motor will not be stopped. Above this speed motor will stop in case StallGuard2 load value reaches zero.	0…7999774 [pps]	RW
182	SmartEnergy threshold speed	Above this speed CoolStep becomes enabled.	0…7999774 [pps]	RW
184	Random TOff mode	0 - Chopper off time is fixed 1 - Chopper off time is random	0/1	RW
185	Chopper synchronization	This parameter allows synchronization of the chopper for both phases of a two phase motor in order to avoid the occurrence of a beat, especially at low velocities. 0: chopper sync function chopSync off 1…15: chopper synchronization	0…15	RW
186	PWM threshold speed	The StealthChop feature will be switched off when the actual velocity is higher than this value. It will be switched on when the actual velocity is below this value (and parameter #187 is greater than zero).	0…7999774 [pps]	RW
187	PWM gradient	Velocity dependent gradient for PWM amplitude (StealthChop). Setting this value to 0 turns off StealthChop.	0..15	RW
188	PWM amplitude	Maximum PWM amplitude when switching to StealthChop mode. Do not set too low. Values above 64 recommended.	0..255	RW
189	PWM scale	Actual PWM amplitude scaler (255=maximum voltage). In voltage mode PWM, this value allows to detect a motor stall.	0…255	R
190	PWM mode	Status of StealthChop voltage PWM mode (depending on velocity thresholds). 0 - StealthChop disabled 1 - StealthChop enabled	0/1	R
191	PWM frequency	PWM frequency selection for StealthChop. 0 - f_{PWM} = 15.625kHz 1 - f_{PWM} = 23.426kHz 2 - f_{PWM} = 31.250kHz 3 - f_{PWM} = 39.024kHz	0…3	RW

Number	Axis Parameter	Description	Range [Units]	Access
192	PWM autoscale	PWM automatic amplitude scaling for StealthChop. 0 - User defined PWM amplitude. The current settings do not have any influence. 1 - Enable automatic current control.	0…1	RW
193	Reference search mode	1 Search left stop switch only. 4 Search left stop switch from both sides. 7 Search home switch in positive direction, ignore end switches. 8 Search home switch in negative direction, ignore end switches. Additional functions: • Add 128 to a mode value for inverting the home switch (can be used with mode 5…8). • Add 64 to a mode for searching the right instead of the left reference switch (can be used with mode 1…4).	1..8	RW
194	Reference search speed	This value specifies the speed for roughly searching the reference switch.	0…7999774 [pps]	RW
195	Reference switch speed	This parameter specifies the speed for searching the switching point. It should be slower than parameter 194.	0…7999774 [pps]	RW
196	End switch distance	This parameter provides the distance between the end switches after executing the RFS command (with reference search mode 2 or 3).	-2147483648 …2147483647 [μsteps]	R
197	Last reference position	This parameter contains the last position value before the position counter is set to zero during reference search.	-2147483648 …2147483647 [μsteps]	R
201	Encoder mode	A combination of the following values: Bit 2 Null channel polarity. (0:low, 1: high) Bit 5 Clear encoder at next null channel event. Connect encoder N channel to GPI0. The value read back may also contain other bits. Bit 10 cannot be changed. Other bits are unused.	0..2047	RW
202	Motor full step resolution	Full step resolution of the motor (Default: 200).	0…65535 [$\frac{fullsteps}{round}$]	RW

Number	Axis Parameter	Description	Range [Units]	Access
204	Freewheeling mode	Stand still option when the standby current (parameter 7) is set to zero. 0 normal operation 1 freewheeling 2 coil shorted using low side drivers 3 coil shorted using high side drivers	0…3	RW
206	Actual load value	Readout of the actual load value used for stall detection (StallGuard2).	0…1023	R
207	Extended error flags	A combination of the following values: 1 StallGuard error 2 deviation error These error flags are cleared automatically when this parameter has been read out or when a motion command has been executed.	0…3	R
208	Motor driver error flags	A combination of the following values: Bit 0 StallGuard2 status (1: stall detected) Bit 1 Overtemperature (1: driver is shut down due to overtemperature) Bit 2 Overtemperature pre-warning (1: temperature threshold is exceeded) Bit 3 Short to ground A (1: short condition detected, driver currently shut down) Bit 4 Short to ground B (1: short condition detected, driver currently shut down) Bit 5 Open load A (1: no chopper event has happened during the last period with constant coil polarity) Bit 6 Open load B (1: no chopper event has happened during the last period with constant coil polarity) Bit 7 Stand still (1: no step pulse occurred during the last 2^{20} clock cycles)	0…255	R
209	Encoder position	Encoder counter value.	-2147483648 …2147483647 [μsteps]	RW
210	Encoder resolution	Encoder counts per round.	0…65535	RW

Number	Axis Parameter	Description	Range [Units]	Access
212	Maximum encoder deviation	When the actual position (parameter 1) and the encoder position (parameter 209) differ more than set here the motor will be stopped. This function is switched off when the maximum deviation is set to zero.	0…65535 [encoder steps]	RW
214	Power down delay	Standstill period before the current will be ramped down to standby current. The standard value is 200 (which means 2000ms).	0…417 [10ms]	RW
255	Unit mode	Units of velocity and acceleration/deceleration settings: 0 - the internal units of the TMC5130 are used directly 1 - the units are pps for velocity and pps^2 for acceleration/deceleration **Default value is 1 (pps units).**	0/1	RW

Table 15: All TMCM-1278 Axis Parameters

5 Global Parameters

The following sections describe all global parameters that can be used with the SGP, GGP, AGP, STGP and RSGP commands. Global parameters are grouped into banks:

- Bank 0: Global configuration of the module.
- Bank 1: Not used.
- Bank 2: TMCL user variables.
- Bank 3: TMCL interrupt configuration.

5.1 Bank 0

Parameters with numbers from 64 on configure all settings that affect the overall behaviour of a module. These are things like the serial address, the RS485 baud rate or the CAN bit rate (where appropriate). 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 automatically stored in the EEPROM.

Note

- An SGP command on such a parameter will always store it permanently and no extra STGP command is needed.
- Take care when changing these parameters, and use the appropriate functions of the TMCL-IDE to do it in an interactive way.
- Some configurations of the interface (for example baud rates that are not supported by the PC) may leed to the fact that the module cannot be reached any more. In such a case please see the TMCM-1278 Hardware Manual on how to reset all parameters to factory default settings.
- Some settings (especially interface bit rate settings) do not take effect immediately. For those settings, power cycle the module after changing them to make the changes take effect.

There are different parameter access types, like read only or read/write. Table 16 shows the different parameter access types used in the global parameter tables.

Meaning of the Letters in the Access Column		
Access type	Command	Description
R	GGP	Parameter readable
W	SGP, AGP	Parameter writable
E	STGP, RSGP	Parameter can be stored in the EEPROM
A	SGP	Automatically stored in the EEPROM

Table 16: Meaning of the Letters in the Access Column

All Global Parameters of the TMCM-1278 Module in Bank 0				
Number	Global Parameter	Description	Range [Units]	Access
69	CAN bit rate	2 20kBit/s 3 50kBit/s 4 100kBit/s 5 125kBit/s 6 250kBit/s 7 500kBit/s 8 1000kBit/s (Default)	2…8	RWA
70	CAN reply ID	The CAN ID for replies from the board (default: 2).	0…2047	RWA
71	CAN ID	The module (target) address for CAN (default: 1).	0…2047	RWA
77	Auto start mode	0 - Do not start TMCL application after power up (default). 1 - Start TMCL application automatically after power up.	0/1	RWA
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 **When switching off the protection against disassembling (changing this parameter from 1 or 3 to 0 or 2, the program will be erased first!**	0/1/2/3	RWA
82	CAN heartbeat	Heartbeat for CAN interface. If this time limit is up and no further command is received the motor will be stopped. Setting this parameter to 0 (default) turns off the CAN heartbeat function.	0…65535 [ms]	RWA
83	CAN secondary address	Second CAN ID for the module. Switched off when set to zero.	0…2047	RWA
84	Coordinate storage	0 - coordinates are stored in RAM only (but can be copied explicitly between RAM and EEPROM) 1 - coordinates are always also stored in the EEPROM	0/1	RWA
85	Do not restore user variables	Determines if TMCL user variables are to be restored from the EEPROM automatically on startup. 0 - user variables are restored (default) 1 - user variables are not restored	0/1	RWA

Number	Global Parameter	Description	Range [Units]	Access
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	Contains the address of the currently executed TMCL command.		R
132	TMCL tick timer	A 32 bit counter that gets incremented by one every millisecond. It can also be reset to any start value.	0...2147483647	RW
133	Random number	Returns a random number. The seed value can be set by writing to this parameter.	0...2147483647	RW
255	Suppress reply	The reply in direct mode will be suppressed when this parameter is set to 1. This parameter cannot be stored to EEPROM and will be reset to 0 on startup. The reply will not be suppressed for GAP, GGP and GIO commands.	0/1	RW

Table 17: All Global Parameters of the TMCM-1278 Module in Bank 0

5.2 Bank 1

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

5.3 Bank 2

Bank 2 contains general purpose 32 bit variables for use in TMCL applications. They are located in RAM and the first 56 variables can also be stored permanently in the EEPROM. After booting, their values are automatically restored to the RAM. Up to 256 user variables are available. Please see table 16 for an explanation of the different parameter access types.

User Variables in Bank 2				
Number	Global Parameter	Description	Range [Units]	Access
0...55	user variables #0...#55	TMCL user variables	-2147483648 ... 2147483647	RWE
56...255	user variables #56...#255	TMCL user variables	-2147483648 ... 2147483647	RWE

Table 18: User Variables in Bank 2

5.4 Bank 3

Bank 3 contains interrupt parameters. Some interrupts need configuration (e.g. the timer interval of a timer interrupt). This can be done using the SGP commands with parameter bank 3 (SGP <type>, 3, <value>). **The priority of an interrupt depends on its number. Interrupts with a lower number have a higher priority.**
Table 19 shows all interrupt parameters that can be set. Please see table 16 for an explanation of the parameter access types.

Interrupt Parameters in Bank 3				
Number	Global Parameter	Description	Range [Units]	Access
0	Timer 0 period (ms)	Time between two interrupts	0...4294967295 [ms]	RW
1	Timer 1 period (ms)	Time between two interrupts	0...4294967295 [ms]	RW
2	Timer 2 period (ms)	Time between two interrupts	0...4294967295 [ms]	RW
27	Stop left 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0...3	RW
28	Stop right 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0...3	RW
39	Input 0 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0...3	RW
40	Input 1 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0...3	RW
41	Input 2 trigger transition	0=off, 1=low-high, 2=high-low, 3=both	0...3	RW

Table 19: Interrupt Parameters in Bank 3

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 search algorithm. You will also find basic information about StallGuard2™ and CoolStep™ in this chapter.

6.1 Reference Search

The built-in reference search features switching point calibration and support for a home switch and/or one or two end switches. The internal operation is based on a state machine that can be started, stopped and monitored (instruction RFS, opcode 13). The settings of the automatic stop functions corresponding to the end switches (axis parameters 12 and 13) do not influence the reference search.

Notes:

- Until the reference switch is found for the first time, the searching speed set by axis parameter 194 is used.
- After hitting the reference switch, the motor slowly moves until the switch is released. Finally the switch is re-entered in the other direction, setting the reference point to the center of the two switching points. The speed used for this calibration is defined by axis parameter 195.

Axis parameter 193 defines the reference search mode to be used. Choose one of the reference search modes shown in table 20 and in the following subsections:

Reference Search Modes	
Value	Description
1	search left stop switch only
2	search right stop switch, then search left stop switch
3	search right stop switch, then search left stop switch from both sides
4	search left stop switch from both sides
5	search home switch in negative direction, reverse the direction when left stop switch reached
6	search home switch in positive direction, reverse the direction when right stop switch reached
7	search home switch in negative direction, ignore end switches
8	search home switch in positive direction, ignore end switches
9	search encoder null channel in positive direction
10	search encoder null channel in negative direction

Table 20: Reference Search Modes

The drawings in the following subsections show how each reference search mode works. A linear stage with two end points and a moving slider is used as example.

6.1.1 Mode 1

Reference search mode 1 only searches the left end switch. Select this mode by setting axis parameter #193 to 1. Figure 4 illustrates this.

Add 64 to the mode number (i.e. set axis parameter #193 to 65) to search the right end switch instead of the left end switch.

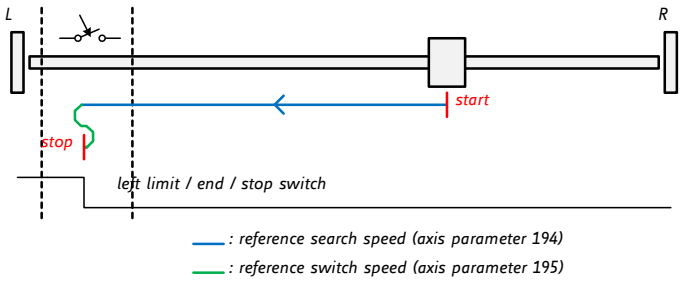


Figure 4: Reference Search Mode 1

6.1.2 Mode 2

Reference search mode 2 first searches the right end switch and then the left end switch. The left end switch is then used as the zero point. Figure 5 illustrates this. Select this mode by setting axis parameter #193 to 2. After the reference search has finished, axis parameter #196 contains the distance between the two reference switches in microsteps.

Add 64 to the mode number (i.e. set axis parameter #193 to 66) to search the left end switch first and then use the right end switch as the zero point.

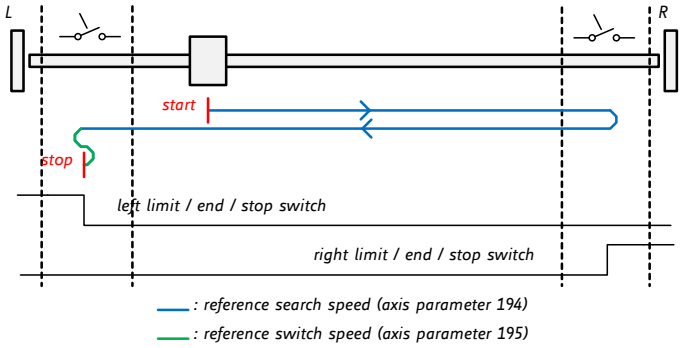


Figure 5: Reference Search Mode 2

6.1.3 Mode 3

Reference search mode 3 first searches the right end switch and then the left end switch. The left end switch is then searched from both sides, to find the middle of the left end switch. This is then used as the zero point. Figure 6 illustrates this. Select this mode by setting axis parameter #193 to 3. After the reference search has finished, axis parameter #196 contains the distance between the right end switch and the middle of the left end switch in microsteps.

Add 64 to the mode number (i.e. set axis parameter #193 to 67) to search the left end switch first and then use the middle of the right end switch as the zero point.

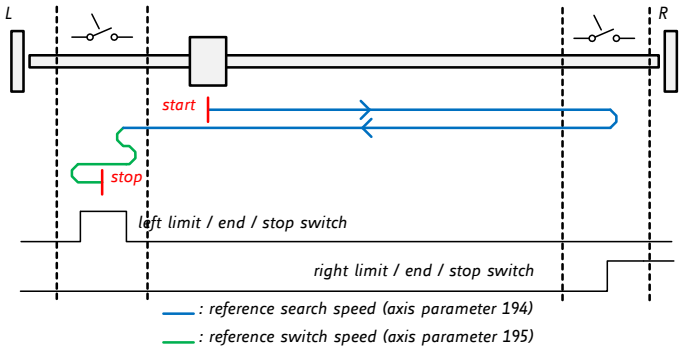


Figure 6: Reference Search Mode 3

6.1.4 Mode 4

Reference search mode 4 searches the left end switch only, but from both sides so that the middle of the switch will be found and used as the zero point. This is shown in figure 7.

Add 64 to the mode number (i.e. set axis parameter #193 to 68) to search the right end switch instead.

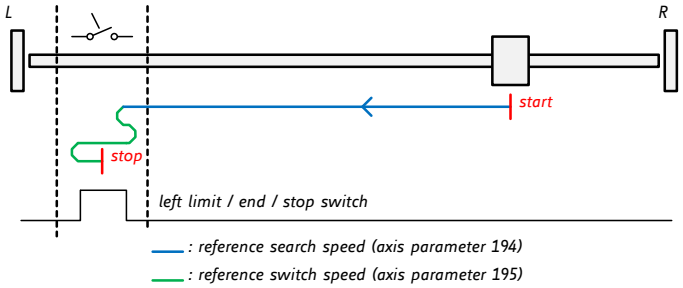


Figure 7: Reference Search Mode 4

6.1.5 Mode 5

Reference search mode 5 searches the home switch in negative direction. The search direction will be reversed if the left limit switch is reached. This is shown in figure 8.
Add 128 to the mode number (i.e. set axis parameter #193 to 133) to reverse the polarity of the home switch input.

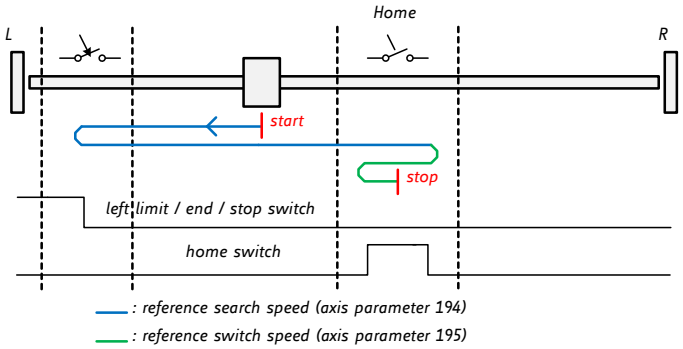


Figure 8: Reference Search Mode 5

6.1.6 Mode 6

Reference search mode 6 searches the home switch in positive direction. The search direction will be reversed if the right limit switch is reached. This is shown in figure 9.
Add 128 to the mode number (i.e. set axis parameter #193 to 134) to reverse the polarity of the home switch input.

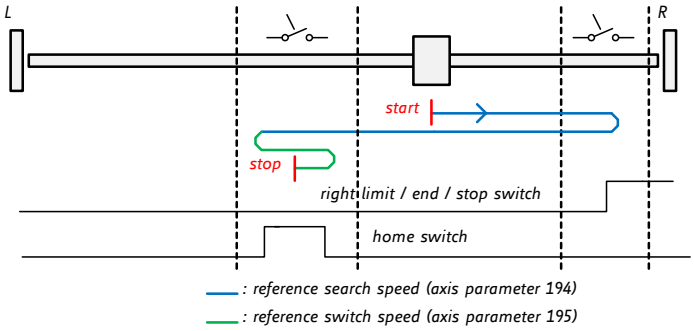


Figure 9: Reference Search Mode 6

6.1.7 Mode 7

Reference search mode 7 searches the home switch in negative direction, ignoring the limit switch inputs. It is recommended mainly for use with a circular axis. The exact middle of the switch will be found and used as the zero point. Figure 10 illustrates this.
Add 128 to the mode number (i.e. set axis parameter #193 to 135) to reverse the polarity of the home switch input.

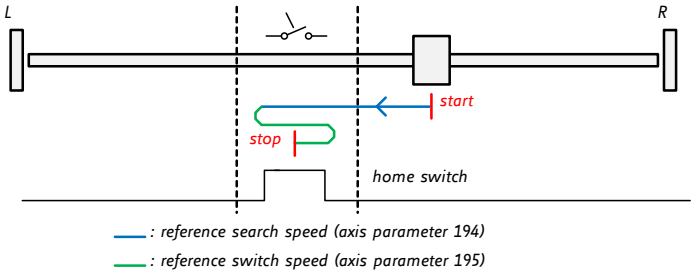


Figure 10: Reference Search Mode 7

6.1.8 Mode 8

Reference search mode 8 searches the home switch in positive direction, ignoring the limit switch inputs. It is recommended mainly for use with a circular axis. The exact middle of the switch will be found and used as the zero point. Figure 11 illustrates this.
Add 128 to the mode number (i.e. set axis parameter #193 to 136) to reverse the polarity of the home switch input.

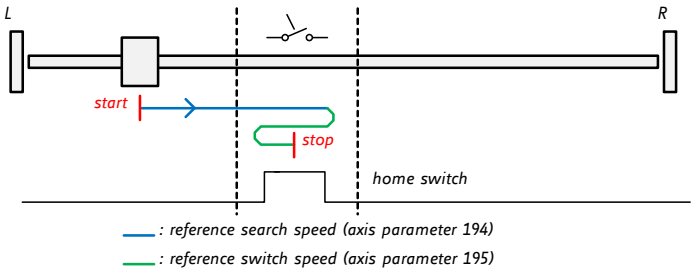


Figure 11: Reference Search Mode 8

6.1.9 Mode 9

Reference search mode 9 searches the null channel (also called index pulse) of an encoder in positive direction. The encoder resolution and the encoder null channel polarity have to be set correctly as otherwise this reference search method cannot work.

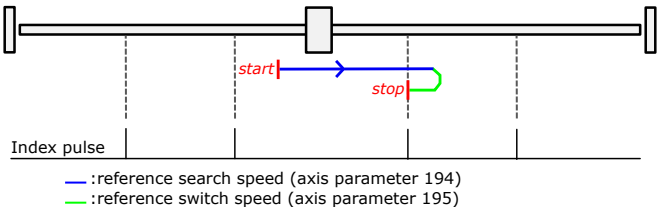


Figure 12: Reference Search Mode 9

6.1.10 Mode 10

Reference search mode 10 searches the null channel (also called index pulse) of an encoder in negative direction. The encoder resolution and the encoder null channel polarity have to be set correctly as otherwise this reference search method cannot work.

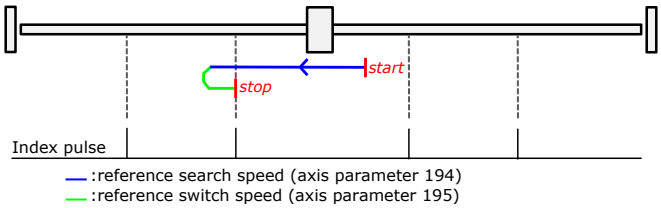


Figure 13: Reference Search Mode 10

6.2 Using Encoders

With the TMCM-1278 it is also possible to use encoders. Please note that the TMCM-1278 is an open-loop system, and hence encoders can only be used to check if the motor has really reached its target position or really follows the position counter. With the TMCM-1278 the encoder cannot be used for a servo-like closed-loop operation (but there are a number of other Trinamic modules which can do real closed-loop operation).
Consider the following things for using encoders with the TMCM-1278:

- Set the motor full step resolution using parameter #202 (for 1.8° motors this is 200 which is also the default value of this parameter).
- Set the encoder resolution (as encoder counts per round) using axis parameter #210.
- When parameters #202 and #210 are set to the right values the module will automatically convert the encoder resolution to the motor microstep resolution so that the encoder postion is the same as the motor position.
- The encoder position can be read using axis parameter #209. This parameter is also writable, e.g. for setting a new origin. After a reference search this parameter is also automatically set to 0.
- The motor can also be stopped automatically when it cannot follow anymore (due to overload or obstruction). Axis parameter #212 controls this function.
- Some special encoder functions (like clear when null channel has been found) can be controlled using axis parameter #201.

6.3 StallGuard2

The module is equipped with motor driver chips that feature load measurement. This load measurement can be used for stall detection. StallGuard2 delivers a sensorless load measurement of the motor as well as a stall detection signal. The measured value changes linear with the load on the motor in a wide range of load, velocity and current settings. At maximum motor load the StallGuard value goes to zero. This corresponds to a load angle of 90° between the magnetic field of the stator and magnets in the rotor. This also is the most energy efficient point of operation for the motor.

Stall detection means that the motor will be stopped automatically when the load gets too high. This function is configured mainly using axis parameters #174 and #181.

Stall detection can for example be used for finding the reference point without the need for reference switches. A short routine written in TMCL is needed to use StallGuard for reference searching.

6.4 CoolStep

This section gives an overview of the CoolStep related parameters. Please bear in mind that the figure only shows one example for a drive. There are parameters which concern the configuration of the current. Other parameters are there for velocity regulation and for time adjustment.

Figure 14 shows all the adjustment points for CoolStep. It is necessary to identify and configure the thresholds for current (I6, I7 and I183) and velocity (V182). Furthermore the StallGuard2 feature has to be adjusted (SG170). It can also be enabled if needed (SG181).

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

In this chapter only basic axis parameters are mentioned which concern CoolStep and StallGuard2. The complete list of axis parameters in chapter 4 contains further parameters which offer more configuration options.

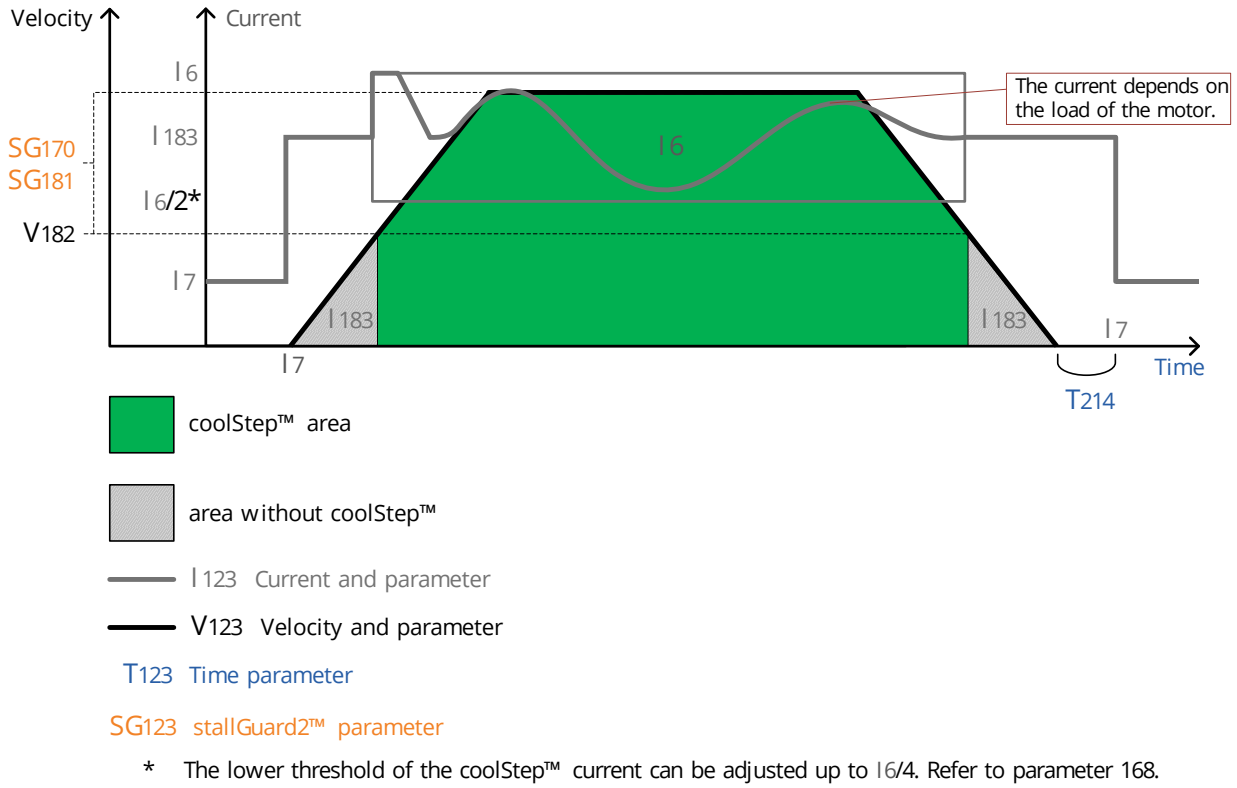


Figure 14: CoolStep Adjustment Points and Thresholds

CoolStep Adjustment Points and Thresholds		
Number	Axis Parameter	Description
I6	Absolute maximum current	The maximum value is 255. This value means 100% of the maximum current of the module. The current adjustment is within the range 0…255 and can be adjusted in 32 steps (0…255 divided by eight; e.g. step 0 = 0…7, step 1 = 8…15 and so on). Too high values may cause motor damage!
I7	Standby current	The current limit two seconds after the motor has stopped.
I168	smartEnergy current minimum	Sets the lower motor current limit for CoolStep operation by scaling the CS (Current Scale, see axis parameter 6) value. Minimum motor current: 0 - 1/2 of CS 1 - 1/4 of CS
I169	smartEnergy current down step	Sets the number of StallGuard2 readings above the upper threshold necessary for each current decrement of the motor current. Number of StallGuard2 measurements per decrement: Scaling: 0…3: 32, 8, 2, 1 0: slow decrement 3: fast decrement
I171	smartEnergy current up step	Sets the current increment step. The current becomes incremented for each measured StallGuard2 value below the lower threshold (see smartEnergy hysteresis start). current increment step size: Scaling: 0…3: 1, 2, 4, 8 0: slow increment 3: fast increment
SG170	smartEnergy hysteresis	Sets the distance between the lower and the upper threshold for StallGuard2 reading. Above the upper threshold the motor current becomes decreased.
SG181	Stop on stall	Below this speed motor will not be stopped. Above this speed motor will stop in case StallGuard2 load value reaches zero.
V182	smartEnergy threshold speed	Above this speed CoolStep becomes enabled.
T214	Power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (which means 2000msec).

Number	Axis Parameter	Description

Table 21: CoolStep Adjustment Points and Thresholds

6.5 Velocity and Acceleration Calculation

When the unit mode (axis parameter #255) is set to 1 (which is also the default value), all velocity parameters on the TMCM-1278 are given in microsteps per second (also called pulse per second or pps). Acceleration and deceleration units are given in pps^2.
When axis parameter #255 is set to 0 the internal units of the ramp generators are directly used. But this is only necessary in very special cases. Normally one should leave axis parameter #255 at 1 and use the pps units.
In order to convert between pps units and units like rounds per second (rps) or rounds per minute (rpm), one has to know the fullstep resolution of the motor (full steps per round) and the microstep resolution setting of the module (axis parameter #140, default setting is 256 microsteps per full step).
So to convert from pps to rps, use the following formula:

$$v_{rps} = \frac{v_{pps}}{r_{fullstep} \cdot r_{microstep}}$$

To convert from rps to rpm, use:

$$v_{rpm} = v_{rps} \cdot 60$$

With the following symbols:

- v_{rps}: velocity in rounds per second
- v_{rpm}: velocity in rounds per minute
- v_{pps}: velocity in pulses (microsteps) per second
- $r_{fullstep}$: fullstep resolution of the motor (with most motors 200 (1.8°))
- $r_{microstep}$: microstep setting of the module (default 256)

So, with a 200 fullsteps motor and a microstep setting of 256 (axis parameter #140 = 8), a velocity of 51200pps will result in 1rps (60rpm).

6.6 SixPoint Ramp

The TMCM-1278 is equipped with a motion controller that supports TRINAMIC's SixPoint ramp technology. Please see also section 1.3 for more information about the SixPoint ramp. The sixPoint ramp can be configured using the following axis parameters:

Parameter Name	TMCL Axis Parameter Number
Start velocity (V_{START})	19
Acceleration A1	15
Velocity V1	16
Acceleration A2	5
Maximum positioning velocity (V_{MAX})	4
Deceleration D2	17
Deceleration D1	18
Stop velocity V_{STOP}	20
Wait time WAIT	21

Table 22: SixPoint Ramp Parameters

Setting the velocity V1 (axis parameter #16) to zero switches off the SixPoint ramp. In this case, a trapezoidal ramp defined by parameters 5, 4, 15, 19, 20 and 21 will be used.

Note The SixPoint ramp will only be used in positioning mode (MVP command). Velocity mode (ROR/ROL commands) will always use a trapezoidal ramp, defined just by the acceleration (axis parameter 5), the speed given with the ROR or ROL command and the start and stop speed (axis parameters 19 and 20). The deceleration parameters will not be used in velocity mode.

6.7 StealthChop™

The TMCM-1278 is equipped with a motor driver that supports TRINAMIC's StealthChop™ technology. Using StealthChop the motor can run at a very low audible noise and also at very low vibrations when running at low speeds. To use StealthChop some parameters need to be specified. The most important parameter is the highest speed at which StealthChop is to be used. Above this speed the motor driver will automatically switch to normal chopper mode. StealthChop is not suitable for running the motor at higher speeds. But this is not a problem as at high speeds the running noise will always be higher than the choppering noise.

The StealthChop™ feature is controlled by the following axis parameters:

- Axis parameter #187 (PWM gradient): setting this parameter to zero (default value) generally switches off StealthChop. So this parameter needs to be set to a value greater than zero to use StealthChop. Mostly it is best to start with a value of 15 (the maximum for this parameter).
- Axis parameter #188 (PWM amplitude): start with the default value. Later, axis parameters #187 and #188 can be fine tuned if really necessary.
- Axis parameter #182 (CoolStep threshold speed): even if CoolStep is not to be used, this parameter has to be set to the maximum speed at which StealthChop is to be used. When the actual speed exceeds this threshold (when CoolStep could be used if enabled) StealthChop will automatically be switched off.
- Axis parameter #186 (PWM threshold speed): this is the maximum speed for StealthChop. Above this speed, StealthChop will automatically be switched off. This means that the actual speed has to be lower than the values of axis parameters #182 and #186 in order to activate StealthChop.
- Axis parameter #22 (Speed threshold for de-activating CoolStep or switching to fullstep mode): The actual speed also has to be lower than specified by this parameter to use StealthChop. But the default value of this parameter is 16777215, so this parameter normally does not need to be set.
- Axis parameter #190 (PWM mode): this read-only parameter shows if StealthChop is currently being used.
- Axis parameter #191 (PWM frequency): this parameter selects the PWM frequency to be used in StealthChop mode. Normally leave at its default value.
- Axis parameter #192 (PWM autoscale): enables automatic current control. This is switched on by default, and mostly there is no need to change this.

The easiest way to get startetd with StealthChop is to set axis parameter #182 and axis parameter #186 to the same speed values (e.g. 5000) and axis parameter #187 to 15. Then, run the motor at different speeds below and above the threshold values to see the difference between StealthChop and normal chopper mode. The following example program shows some typical settings:

```
SAP 182, 0, 5000
SAP 186, 0, 5000
SAP 187, 0, 15
```

Please note that the threshold speeds are not matched exactly. For this example, try to run the motor at a speed slightly below 5000pps and slightly above 5000pps. Watch parameter #190 to see when the module switches between StealthChop mode and normal mode.

6.8 Freewheeling

The motor driver also supports three different freewheeling modes. These can be controlled using axis parameter #204. The following modes can be selected:

- Normal mode: this is the default setting (axis parameter #204 set to 0). When the motor is not running the coil current will just be lowered to the standby current set by axis parameter #7.
- Freewheeling: Settting axis parameter #204 to 1 activates freewheeling mode. When the motor is not running the motor coils will be switched off completely so that the motor can also be moved manually. Please note that in such a case the position will be lost. Either an encoder has to be used or a new reference search will be necessary.
- Coils shorted using low side drivers: Set axis parameter #204 to 2 to activate this mode. When the motor is not running the current will be completely switched off, but the coils will be shorted which will block the motor and thus also hold the position.
- Coils shorted using high side drivers: Setting axis parameter #204 to 3 activates this mode. When the motor is not running the current will be completely switched off, but the coils will be shorted which will block the motor and thus also hold the position.

Please note that modes 1, 2 and 3 can only be used when StealthChop™ is active and the standby current (axis parameter #7) is set to zero. So, StealthChop™ must at least be activated for speed 0 in order to be able to use one of the freewheeling modes. The following example program shows how to activate freewheeling mode #1:

```
SAP 7, 0, 0
SAP 186, 0, 0
SAP 187, 0, 15
SAP 204, 0, 1
```

7 TMCL Programming Techniques and Structure

7.1 Initialization

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

7.2 Main Loop

Embedded systems normally use a main loop that runs infinitely. This is also the case in a TMCL application that is running stand alone. Normally the auto start mode of the module should be turned on. After power up, the module 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, 50000 //define maximum positioning speed
  SAP 5, 0, 10000 //define maximum acceleration

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

7.3 Using Symbolic Constants

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

```
//Define some constants
#include TMCLParam.tmc
MaxSpeed = 50000
MaxAcc = 10000
Position0 = 0
Position1 = 500000

//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
17 JA MainLoop
```

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

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

7.4 Using Variables

The user variables can be used if variables are needed in your program. They can store temporary values. The commands SGP, GGP and AGP as well as STGP and RSGP are used to work with user variables:

- SGP is used to set a variable to a constant value (e.g. during initialization phase).
- GGP is used to read the contents of a user variable and to copy it to the accumulator register for further usage.
- AGP can be used to copy the contents of the accumulator register to a user variable, e.g. to store the result of a calculation.
- The STGP command stores the contents of a user varaible in the EEPROM.
- The RSGP command copies the value stored in the EEPROM back to the user variable.
- Global parameter 85 controls if user variables will be restored from the EEPROM automatically on startup (default setting) or not (user variables will then be initialized with 0 instead).

Please see the following example:

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

5 SGP MyVariable, 2, 1234   //Initialize the variable with the value 1234
  ...
7 ...
  GGP MyVariable, 2    //Copy contents of variable to accumulator register
9 CALC MUL, 2          //Multiply accumulator register with two
  AGP MyVariable, 2    //Store contents of accumulator register to variable
11 ...
  ...
```

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

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

7.5 Using Subroutines

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

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

7.6 Combining Direct Mode and Standalone Mode

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

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

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

Example:

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

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

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

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

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

7.7 Make the TMCL Program start automatically

For stand-alone operation the module has to start the TMCL program in its memory automatically after power-on. In order to achieve this, switch on the Autostart option of the module. This is controlled by global parameter #77. There are different ways to switch on the Autostart option:

- Execute the command SGP 77, 0, 1 in direct mode (using the Direct Mode tool in the TMCL-IDE).
- Use the Global Parameters tool in the TMCL-IDE to set global parameter #77 to 1.
- Use the Autostart entry in the TMCL menu of the TMCL Creator in the TMCL-IDE. Go to the Autostart entry in the TMCL menu and select '"On"'.

8 Figures Index

9 Tables Index

10 Supplemental Directives

10.1 Producer Information

10.2 Copyright

10.3 Trademark Designations and Symbols

Trademark designations and symbols used in this documentation indicate that a product or feature is owned and registered as trademark and/or patent either by TRINAMIC or by other manufacturers, whose products are used or referred to in combination with TRINAMIC's products and TRINAMIC's product documentation.

This TMCL™ Firmware Manual is a non-commercial publication that seeks to provide concise scientific and technical user information to the target user. Thus, trademark designations and symbols are only entered in the Short Spec of this document that introduces the product at a quick glance. The trademark designation /symbol is also entered when the product or feature name occurs for the first time in the document. All trademarks and brand names used are property of their respective owners.

10.4 Target User

The documentation provided here, is for programmers and engineers only, who are equipped with the necessary skills and have been trained to work with this type of product.

The Target User knows how to responsibly make use of this product without causing harm to himself or others, and without causing damage to systems or devices, in which the user incorporates the product.

10.5 Disclaimer: Life Support Systems

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

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

Information given in this document is believed to be accurate and reliable. However, no responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties which may result from its use. Specifications are subject to change without notice.

10.6 Disclaimer: Intended Use

The data specified in this user manual is intended solely for the purpose of product description. No representations or warranties, either express or implied, of merchantability, fitness for a particular purpose

or of any other nature are made hereunder with respect to information/specification or the products to which information refers and no guarantee with respect to compliance to the intended use is given.

In particular, this also applies to the stated possible applications or areas of applications of the product. TRINAMIC products are not designed for and must not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (safety-Critical Applications) without TRINAMIC's specific written consent.

TRINAMIC products are not designed nor intended for use in military or aerospace applications or environments or in automotive applications unless specifically designated for such use by TRINAMIC. TRINAMIC conveys no patent, copyright, mask work right or other trade mark right to this product. TRINAMIC assumes no liability for any patent and/or other trade mark rights of a third party resulting from processing or handling of the product and/or any other use of the product.

10.7 Collateral Documents & Tools

This product documentation is related and/or associated with additional tool kits, firmware and other items, as provided on the product page at: www.trinamic.com.

11 Revision History

11.1 Firmware Revision

Version	Date	Author	Description
V1.14	2019-JUN-07	OK	First release.
V1.15	2019-DEC-09	OK	RORA command fixed.

Table 23: Firmware Revision

11.2 Document Revision

Version	Date	Author	Description
V1.00	2019-JUN-07	OK	First release.
V1.01	2019-DEC-09	OK	Covers firmware V1.15.
V1.02	2020-APR-16	OK	Removed superfluous axis parameter #215. Added missing axis parameter #201.
V1.03	2020-APR-24	OK	New block diagram.
V1.04	2020-NOV-25	OK	Chapter 6.1: corrections.

Table 24: Document Revision

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Trinamic:

TMCM-1278-CABLE TMCM-1278-TMCL