

TMCM-1290 TMCL Firmware Manual

Firmware Version V1.01 | Rev 0: 02/24

The TMCM-1290 is a single axis controller/driver module for 2-phase bipolar stepper motors. The TMCM-1290 TMCL firmware allows to control the module using TMCL™ commands, supporting standalone operation as well as direct mode control, making use of the Trinamic TMC5240 motion controller and motor driver. Dynamic current control, and quiet, smooth, and efficient operation are combined with StealthChop, DcStep, StallGuard2, and CoolStep features.



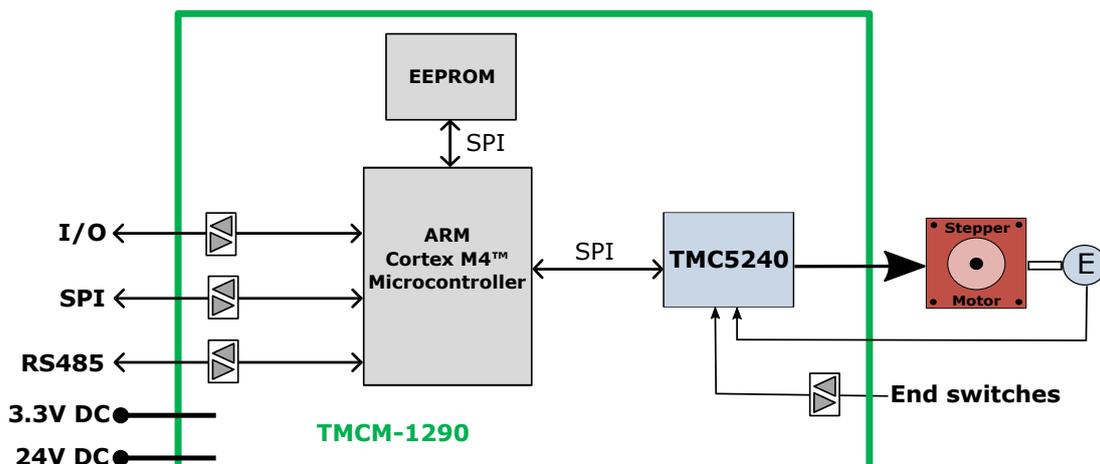
Features

- Single axis stepper motor control
- Supply voltage 24V DC
- TMCL™
- Host interface: RS485 or UART (TTL), SPI
- Additional inputs and outputs
- EightPoint ramps
- CoolStep
- StallGuard2
- StealthChop
- ABN encoder interface

Applications

- Laboratory Automation
- Manufacturing
- Semiconductor Handling
- Robotics
- Factory Automation
- Test and Measurement
- Life Science
- Biotechnology
- Liquid Handling

Simplified Block Diagram



Contents

1	Features	5
1.1	StallGuard2	6
1.2	CoolStep	6
1.3	EightPoint Motion Controller	7
2	First Steps with TMCL	8
2.1	Basic Setup	8
2.2	Using the TMCL Direct Mode	8
2.3	Changing Axis Parameters	8
2.4	Testing with a Simple TMCL Program	9
3	TMCL and the TMCL-IDE — An Introduction	11
3.1	Binary Command Format	11
3.1.1	Checksum Calculation	12
3.2	Reply Format	13
3.2.1	Status Codes	13
3.3	Standalone Applications	14
3.4	TMCL Command Overview	15
3.5	TMCL Commands by Subject	17
3.5.1	Motion Commands	17
3.5.2	Parameter Commands	17
3.5.3	Branch Commands	18
3.5.4	I/O Port Commands	18
3.5.5	Calculation Commands	19
3.5.6	Interrupt Processing Commands	19
3.5.7	New TMCL Commands	21
3.6	Detailed TMCL Command Descriptions	23
3.6.1	ROR (Rotate Right)	23
3.6.2	ROL (Rotate Left)	24
3.6.3	MST (Motor Stop)	25
3.6.4	MVP (Move to Position)	26
3.6.5	SAP (Set Axis Parameter)	29
3.6.6	GAP (Get Axis Parameter)	30
3.6.7	SGP (Set Global Parameter)	31
3.6.8	GGP (Get Global Parameter)	33
3.6.9	STGP (Store Global Parameter)	35
3.6.10	RSGP (Restore Global Parameter)	36
3.6.11	RFS (Reference Search)	37
3.6.12	SIO (Set Output)	39
3.6.13	GIO (Get Input)	41
3.6.14	CALC (Calculate)	44
3.6.15	COMP (Compare)	46
3.6.16	JC (Jump Conditional)	47
3.6.17	JA (Jump Always)	49
3.6.18	CSUB (Call Subroutine)	50
3.6.19	RSUB (Return from Subroutine)	51
3.6.20	WAIT (Wait for an Event to Occur)	52
3.6.21	STOP (Stop TMCL Program Execution – End of TMCL Program)	54
3.6.22	SCO (Set Coordinate)	55
3.6.23	GCO (Get Coordinate)	56
3.6.24	CCO (Capture Coordinate)	58
3.6.25	ACO (Accumulator to Coordinate)	59

3.6.26	CALCX (Calculate Using the X Register)	60
3.6.27	AAP (Accumulator to Axis Parameter)	62
3.6.28	AGP (Accumulator to Global Parameter)	63
3.6.29	CLE (Clear Error Flags)	64
3.6.30	EI (Enable Interrupt)	66
3.6.31	DI (Disable Interrupt)	67
3.6.32	VECT (Define Interrupt Vector)	68
3.6.33	RETI (Return from Interrupt)	70
3.6.34	CALCVV (Calculate Using Two User Variables)	71
3.6.35	CALCVA (Calculate Using a User Variable and the Accumulator Register)	73
3.6.36	CALCAV (Calculate Using the Accumulator Register and a User Variable)	75
3.6.37	CALCVX (Calculate Using a User Variable and the X Register)	77
3.6.38	CALCXV (Calculate Using the X Register and a User Variable)	79
3.6.39	CALCV (Calculate Using a User Variable and a Direct Value)	81
3.6.40	RST (Restart)	83
3.6.41	DJNZ (Decrement and Jump if not Zero)	84
3.6.42	CALL (Conditional Subroutine Call)	85
3.6.43	MVPA (Move to Position Specified by Accumulator Register)	87
3.6.44	ROLA (Rotate Left Using the Accumulator Register)	89
3.6.45	RORA (Rotate Right Using the Accumulator Register)	90
3.6.46	SIV (Set Indexed Variable)	91
3.6.47	GIV (Get Indexed Variable)	92
3.6.48	AIV (Accumulator to Indexed Variable)	93
3.6.49	Customer Specific Command Extensions (UF0... UF7 – User Functions)	94
3.6.50	Request Target Position Reached Event	95
3.6.51	TMCL Control Commands	97
4	Axis Parameters	99
5	Global Parameters	112
5.1	Bank 0	112
5.2	Bank 1	115
5.3	Bank 2	115
5.4	Bank 3	115
6	Module Specific Hints	117
6.1	TMCL over SPI	117
6.2	Motor Current Settings	118
6.3	General Purpose Inputs and Outputs	118
6.4	Position Compare Outputs	119
6.5	Target Reached Outputs	119
6.6	Motion Start Inputs	119
6.7	Synchronized Start of Motion Using RS485 Only	120
7	Hints and Tips	121
7.1	Reference Search	121
7.1.1	Mode 1	122
7.1.2	Mode 2	122
7.1.3	Mode 3	122
7.1.4	Mode 4	123
7.1.5	Mode 5	123
7.1.6	Mode 6	124
7.1.7	Mode 7	124
7.1.8	Mode 8	125
7.1.9	Mode 9	125

7.1.10	Mode 10	125
7.2	Using Encoders	127
7.3	StallGuard2	127
7.4	CoolStep	128
7.5	Velocity and Acceleration Calculation	131
7.6	EightPoint Ramp	132
7.7	StealthChop	133
7.8	Freewheeling	133
7.9	Secondary Serial Address	135
8	TMCL Programming Techniques and Structure	136
8.1	Initialization	136
8.2	Main Loop	136
8.3	Using Symbolic Constants	136
8.4	Using Variables	137
8.5	Using Subroutines	138
8.6	Combining Direct Mode and Standalone Mode	138
8.7	Make the TMCL Program Start Automatically	139
9	Figures Index	140
10	Tables Index	141
11	Supplemental Directives	142
11.1	Producer Information	142
11.2	Copyright	142
11.3	Trademark Designations and Symbols	142
11.4	Target User	142
11.5	Disclaimer: Life Support Systems	142
11.6	Disclaimer: Intended Use	142
11.7	Collateral Documents & Tools	143
12	Revision History	144
12.1	Firmware Revision	144
12.2	Document Revision	144

1 Features

The TMCM-1290 is a single axis controller/driver module for 2-phase bipolar stepper motors with state-of-the-art feature set. It is highly integrated and can be used in many decentralized applications. The module has been designed for coil currents of up to $2.1A_{RMS}$ and 24V DC supply voltage. With its high energy efficiency from Trinamic's CoolStep technology, cost for power consumption is kept down. The TMCL firmware allows for standalone operation and direct mode control.

Main characteristics

- Motion controller and stepper motor driver:
 - Hardware motion profile calculation in real-time.
 - On the fly alteration of motion parameters (for example, 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
 - RS485 bus.
 - ABN encoder interface.
 - Three general-purpose digital inputs/outputs.
 - One dedicated analog input.
 - Two stop switch inputs.

Software

TMCL remote controlled operation through RS485 interface and/or standalone operation through TMCL programming. PC-based application development software TMCL-IDE available for free.

Electrical data

- Supply voltage: +24V DC nominal (10V...30V DC supply range).
- Motor current: up to $2.1A_{RMS}/3A_{peak}$ (programmable).

Refer 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.

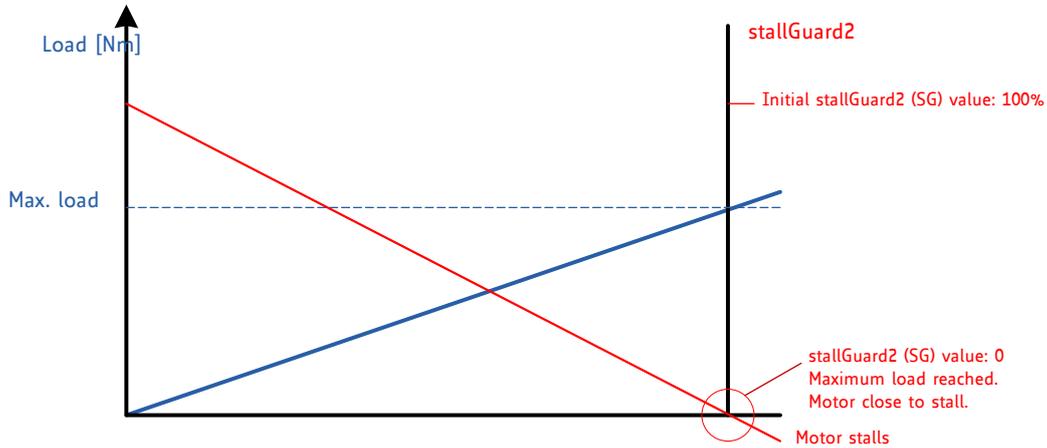


Figure 1: StallGuard2 Load Measurement as a Function of Load

1.2 CoolStep

CoolStep is a load-adaptive automatic current scaling based on the load measurement through 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 system cooler, increases motor life, and allows cost reduction.

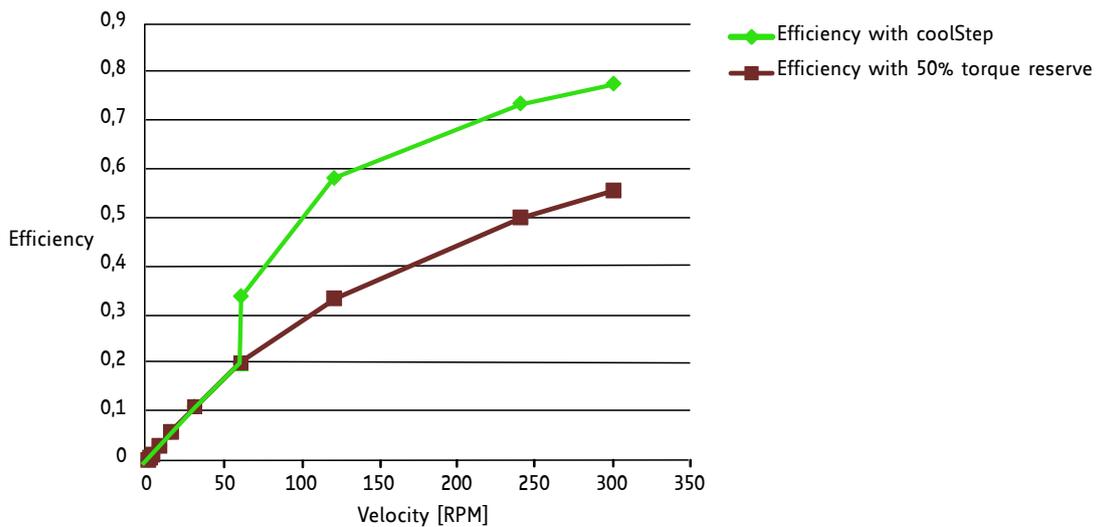


Figure 2: Energy Efficiency Example with CoolStep

1.3 EightPoint Motion Controller

TRINAMIC’s EightPoint motion controller is a new type of ramp generator that offers faster machine operation compared to the classical linear acceleration ramps. It uses three different acceleration settings for the acceleration phase and also three different deceleration settings for the deceleration phase. Start and stop speeds greater than zero can also be used.

This set of three acceleration and deceleration segments can be used in two ways: either for adaptation to the motor torque curve, by using higher acceleration values at lower velocity, or to reduce the jerk (change of acceleration) when transitioning from one acceleration segment to the next. For jerk optimized ramps, typically, A1, D1, AMAX, and DMAX are set to lower values than A2 and D2. The most critical points with regards to jerk are the transition from acceleration to deceleration with no constant velocity segment, as well as the transition from deceleration to acceleration in case of on-the-fly change of target position. To address both, the EightPoint motion profile generator allows to enforce a constant velocity segment based on a minimum segment duration (TVMAX). In case this duration cannot be kept due to insufficient distance, VMAX is automatically reduced for the constant velocity segment.

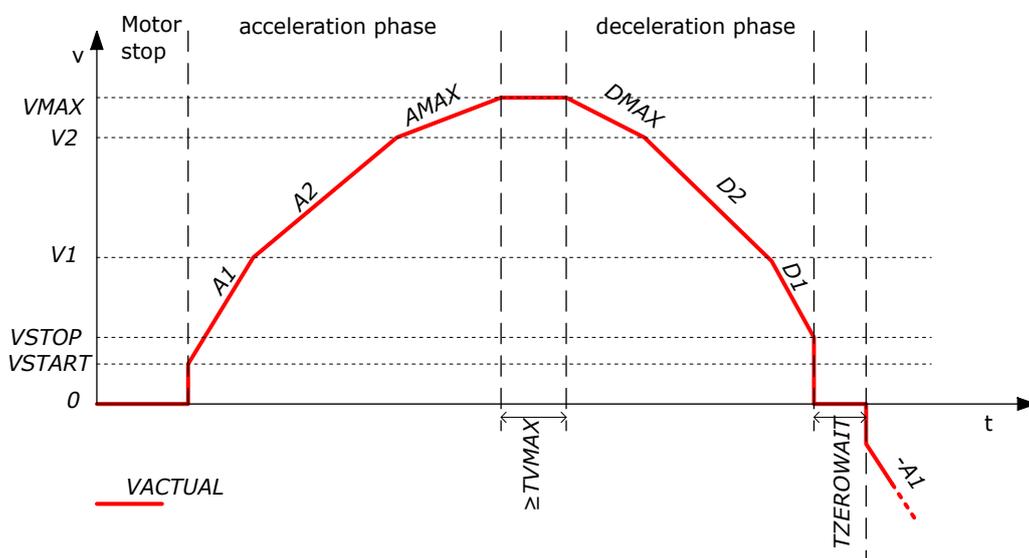


Figure 3: Typical Motion Profile with TRINAMIC EightPoint Motion Controller

An eight point ramp begins using the start speed VSTART (which can also be zero). Then, the acceleration value A1 is used to accelerate the motor to the speed V1. When the speed V1 is reached, the motor is accelerated to the speed V2 using acceleration A2. After reaching the speed V2, the motor is further accelerated using the acceleration value AMAX until it has reached the speed VMAX. The speed is then constant at least as long as set using TVMAX. The deceleration phase begins using the deceleration value DMAX. After reaching the speed V2 again, the deceleration value D2 is used to decelerate to the speed V1. After reaching the speed V1, the deceleration value D1 is used to decelerate to the stop speed VSTOP (which can also be zero). The speed is then zero at least as long as set by TZEROWAIT.

2 First Steps with TMCL

This chapter provides some hints for the first steps with the TMCM-1290 and TMCL. Skip this chapter if already familiar with TMCL and the TMCL-IDE.

Requirements

- Your TMCM-1290 module.
- An RS485 interface.
- A power supply (24V DC) for your TMCM-1290 module.
- The TMCL-IDE 3.x already installed on your PC.
- A two-phase bipolar stepper motor.

2.1 Basic Setup

First of all, the basic setup needs a PC with Windows (at least Windows 7) and the latest TMCL-IDE (> V3.0) installed on it. If the TMCL-IDE is not installed on the PC, download it from the TMCL-IDE product page on the [ADI Trinamic website](#).

Ensure the TMCM-1290 is properly connected to the power supply and the motor is properly connected to the module.

Refer to the TMCM-1290 hardware manual for instructions on how to do this. **Do not connect or disconnect a motor to or from the module while the module is powered!**

Then, start the TMCL-IDE. After that, connect the TMCM-1290 through RS485 and switch on the power supply for the module (while the TMCL-IDE is running on the PC). The TMCL-IDE recognizes the module, and automatically does the necessary driver registrations in Windows.

2.2 Using the TMCL Direct Mode

The TMCL-IDE displays a tree view showing the TMCM-1290 and all tools available for it. Click the *Direct Mode* entry of the tool tree. The *Direct Mode* tool pops up.

In the *Direct Mode* tool, 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, try changing some settings (also called axis parameters) using the SAP command in the direct mode. Choose the SAP command. Then choose the parameter type and 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. See chapter 4 for a complete list of all axis parameters.

Important Axis Parameters																																												
Number	Axis Parameter	Description	Range [Units]	Access																																								
4	Maximum positioning speed	The maximum speed used for positioning ramps.	0...7999774 [pps]	RW																																								
5	Maximum acceleration	Maximum acceleration in positioning ramps. Acceleration and deceleration value in velocity mode.	117...7629278 [pps ²]	RW																																								
6	Maximum current	<p>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:</p> <table border="1"> <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> <p><i>The most important setting, as too high values can cause motor damage.</i></p>	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		0...255	RW
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																																										
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.	0...255	RW																																								

Table 1: Important Axis Parameters

2.4 Testing with a Simple TMCL Program

Now, test the TMCL standalone mode with a simple TMCL program. Type in, assemble and download the program using 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:

```

1  ROL 0, 51200 //Rotate motor 0 with speed 10000
   WAIT TICKS, 0, 500
3  MST 0
   ROR 0, 51200 //Rotate motor 0 with 50000
5  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 that, follow these 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 *Debug* to start the debugger.
2. Click *Animate* to see the single steps of the program.
3. At any time, pause the program, set or reset breakpoints, and resume program execution.
4. To end the debug mode, click *Debug* again.

3 TMCL and the TMCL-IDE — An Introduction

As with most TRINAMIC modules, the software running on the microprocessor of the TMCM-1290 consists of two parts: a bootloader 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. Download new versions free of charge from the [ADI Trinamic software and tools website](#).

The TMCM-1290 supports the 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 client/server relationship. That is, a computer (example, PC/PLC) acting as the interface bus host sends a command to the TMCM-1290. The TMCL interpreter on the module then interprets this command, does the initialization of the motion controller, and reads inputs and writes outputs or whatever is necessary according to the specified command. As soon as this step is done, the module sends a reply back over the interface to the host. The host must not send any next command before the reply for a preceding command is received.

Normally, the module just switches to transmission and occupies the bus for a reply. Otherwise it stays in receive mode. It does not send any data over the interface without receiving a command first. This way, any collision on the bus is 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 that 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 through 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 checksum byte. With CAN, the CAN-ID is used as the module address and the checksum is not needed because the 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 that show how to do this:

Checksum calculation in C:

```

1 unsigned char i, Checksum;
  unsigned char Command[9];
3
  //Set the Command array to the desired command
5 Checksum = Command[0];
  for(i=1; i<8; i++)
7     Checksum+=Command[i];
9
  Command[8]=Checksum; //insert checksum as last byte of the command
  //Now, send it to the module

```

Checksum calculation in Delphi:

```

var
2   i, Checksum: byte;
   Command: array[0..8] of byte;
4
   //Set the Command array to the desired command
6
   //Calculate the Checksum:
8   Checksum:=Command[0];
   for i:=1 to 7 do Checksum:=Checksum+Command[i];
10  Command[8]:=Checksum;
   //Now, send the Command array (9 bytes) to the module

```

3.2 Reply Format

Every time a command is 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 (example, 100 means no error)
1	Command number
4	Value (MSB first!)
1	Checksum

Table 3: TMCL Reply Format

i 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. Use the TMCL-IDE for developing standalone TMCL applications. Download a program into the EEPROM and afterwards it runs on the module. The TMCL-IDE contains an editor and the TMCL assembler, where the commands can be entered using their mnemonic format. They are 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). Example: communication settings or TMCL user variables.
GGP	10	<parameter>, <bank number>	Get global parameter (read out module specific settings). Example: 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

Command	Number	Parameter	Description
WAIT	27	<condition>, <motor number>, <ticks>	Wait with further program execution
STOP	28		Stop program execution
SCO	30	<coordinate number>, <motor number>, <position>	Set coordinate
GCO	31	<coordinate number>, <motor number>	Get coordinate
CCO	32	<coordinate number>, <motor number>	Capture coordinate
CALCX	33	<operation>	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>	Accumulator 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

Command	Number	Parameter	Description
SIV	55	<value>	Set indexed variable
GIV	56		Get indexed variable
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 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 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 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 for calculations within TMCL applications. Although they can 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 accumulator 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 is not 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 (for example, to query the actual position of the motor) without affecting the flow of the TMCL program running on the module.

Also see 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 used 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. Use the TMCL include file `Interrupts.inc` to have symbolic constants for the interrupt numbers. Table 12 shows all interrupts available on the TMCM-1290.

Interrupt Vectors	
Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached 0
9	Position compare 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 is defined for that interrupt, the normal TMCL program flow is interrupted, and the interrupt handling routine is called. Before an interrupt handling routine gets called, the context of the normal program (that is, accumulator register, X register, flags) is saved automatically.

There is no interrupt nesting, that is 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 is automatically restored and the execution of the normal program is continued.

3.5.6.3 Further Configuration of Interrupts

Some interrupts need further configuration (example, the timer interval of a timer interrupt). This can be done using SGP commands with parameter bank 3 (SGP <type> , 3, <value>). See the SGP command (chapter 3.6.7) for further information about this.

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:

```

1  VECT 0, Timer0Irq //define the interrupt vector
   SGP 0, 3, 1000 //configure the interrupt: set its period to 1000ms
3  EI 0 //enable this interrupt
   EI 255 //globally switch on interrupt processing
5
//Main program: toggles output 3, using a WAIT command for the delay
7 Loop:
   SID 3, 2, 1
9   WAIT TICKS, 0, 50
   SID 3, 2, 0
11  WAIT TICKS, 0, 50
   JA Loop
13
//Here is the interrupt handling routine
15 Timer0Irq:
   GIO 0, 2 //check if OUT0 is high
17  JC NZ, Out00ff //jump if not
   SID 0, 2, 1 //switch OUT0 high
19  RETI //end of interrupt
Out00ff:
21  SID 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 that can be used in all interrupt commands. The beginning of the program above then looks as follows:

```

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

3.5.7 New TMCL Commands

To make several operations easier, the following new commands have been introduced from firmware version 1.00 on. Using these new commands, many tasks can be programmed in an easier way. This can save some code, thus making a TMCL program shorter, faster, and easier to understand.

Note that these commands are not available on TMCM-1290 modules with firmware versions before 1.00. So, make sure that at least firmware version 1.00 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

Mnemonic	Command Number	Meaning
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
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: Velocity mode is selected. Then, the velocity value is transferred to the target velocity (axis parameter #2).

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

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 MST command stops the motor using a soft stop.

Internal function: The velocity mode is selected. Then, the target velocity (axis parameter #2) 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 is instructed to move to a specified relative or absolute position. It uses the acceleration/deceleration ramp and the positioning speed programmed into the unit. This command is non-blocking. That is, a reply is 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, see 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 ($-2^{31} \dots 2^{31} - 1$).
- Starting a relative movement by means of an offset to target position (default) or actual position, according to axis parameter #127 (relative positioning option). 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 runs in the opposite direction to take the shorter distance (caused by 32-bit overflow).

Internal function: Position mode is selected and the new position value is transferred to 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 are stored in SRAM and therefore are volatile. That is, information is lost after power off.

Info For a table with parameters and values that can be used together with this command, 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 .

Mnemonic: SAP 4, 0, 51200.

Binary Form of SAP 4, 0, 51200	
Field	Value
Target address	01 _h
Instruction number	05 _h
Type	4 _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-1290 can be adjusted using for example, the SAP command. With the GAP command, these 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 requested value is only returned 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, 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 are automatically stored in the non-volatile memory (EEPROM).

Info

For a table with parameters and values that can be used together with this command, refer to section 5.

Internal function: The specified value is copied to the global parameter specified by the type and bank number. Most parameters of bank 0 are automatically stored in the 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. 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 requested value is only returned in the value field of the reply, without affecting the accumulator.

i Info

For a table with parameters and values that can be used together with this command, refer to section 5.

Internal function: The global parameter specified by the type and bank number is read.

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 that can be used together with this command, see section 5.3.

Internal function: The global parameter specified by the type and bank number is 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 that can be used together with this command, see section 5.3.

Internal function: The global parameter specified by the type and bank number is 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-1290 has a built-in reference search algorithm. The reference search algorithm provides different reference 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.) See the appropriate parameters in the axis parameter table to configure the reference search algorithm to meet needs (see chapter 4).

Internal function: The internal reference search state machine is started or stopped, 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	0	0 (Don't care)
	1 STOP — stop reference search		
	2 STATUS — get status		

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 reference 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 command using bank 2.

Digital Outputs in Bank 2		
Port	Command	Range
GPIO0	SIO 0, 2, <value>	0/1
GPIO1	SIO 1, 2, <value>	0/1
GPIO2	SIO 2, 2, <value>	0/1

Special case: SIO 255, 2, <x> can be used to change all general purpose digital output lines simultaneously. The value <x> is then interpreted as a bit vector, where each of the lower eight bits represents one of the digital outputs. So, the range for <x> is 0...255. The value <x> can also be -1. In this case, the value is taken from the accumulator register.

Pull-up Resistor Control

The SIO command can also be used to program the pull-up resistors of the GPIO0, GPIO1, and GPIO2 inputs. All three pull-up resistors can be switched separately. Use the command SIO 0, 0, <x> to control the pull-up resistor setting. The value <x> is interpreted as a bit vector: bit 0 controls the pull-up resistor of the GPIO0 input, bit 1 controls the pull-up resistor of GPIO1, and bit 2 controls the pull-up resistors of the GPIO2 input.

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 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 returned 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 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 (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
GPIO0	GIO 0, 0	0/1
GPIO1	GIO 1, 0	0/1
GPIO2	GIO 2, 0	0/1

Special case: GIO 255, 0 reads all general purpose inputs simultaneously and puts the result into the lower eight bits of the accumulator register.

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
AIN0	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 (set by SIO commands) can be read back using bank 2.

Digital Outputs in Bank 2		
Port	Command	Range
GPIO0	GIO 0, 2	0/1
GPIO1	GIO 1, 2	0/1
GPIO2	GIO 2, 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 specified value. The internal arithmetic status flags are set according to the result of the comparison. These can then control, for example, 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.

```

1 GAP 1, 0 //get actual position of motor 0
  COMP 1000 //compare actual value with 1000
3 JC GE, Label //jump to Label if greater 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. 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	0 (don't care)	<jump address>
	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		

Example

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

```

1 GAP 1, 0 //get actual position of motor 0
  COMP 1000 //compare actual value with 1000
3 JC GE, Label //jump to Label if greater or equal to 1000
  ...
5 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:

```

1 Loop :
  MVP ABS , 0 , 51200
3  WAIT POS , 0 , 0
  MVP ABS , 0 , 0
5  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 is 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:
2   MVP ABS, 0, 10000
   CSUB SubW //Save program counter and jump to label SubW
4   MVP ABS, 0, 0
   CSUB SubW //Save program counter and jump to label SubW
6   JA Loop

8 SubW:
   WAIT POS, 0, 0
10  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 is 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

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 are taken for this value. So, for example, WAIT TICKS, 0, -1 waits as long as specified by the value stored in the accumulator. *The accumulator must not contain a negative value when using this option.*

The timeout flag (ETO) is set after a timeout limit is reached. Then, use a JC ETO command to check for such errors or clear the error using the CLE command.

Internal function: The TMCL program counter is held 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 is 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 start up (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 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, a 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 returned 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 start up (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 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 start up (with the default setting, the coordinates are stored in RAM only). 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 (Accumulator 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 start up (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 the 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 (Accumulator to Axis Parameter)

The content of the accumulator register is transferred to the specified axis parameter. For practical usage, the accumulator has to be loaded, example, 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 that can be used together with this command, see to section 4.

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

```

1 Start:
  GIO 0,1 //get value of analog input line 0
3  CALC MUL, 4 //multiply by 4
  AAP 0,0 //transfer result to target position of motor 0
5  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 (Accumulator to Global Parameter)

The content of the accumulator register is transferred to the specified global parameter. For practical usage, the accumulator has to be loaded, example, 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, 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.*

i Info

See table 12 for a list of interrupts that can be used on the TMCM-1290 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

See table 12 for a list of interrupts that can be used on the TMCM-1290 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 redefined. *This command is intended for use in standalone mode only.*

Info

See table 12 for a list of interrupts that can be used on the TMCM-1290 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:

```

1   VECT 0, Timer0Irq
   ...
3 Loop:
   ...
5   JA Loop
   ...
7 Timer0Irq:
   SIO 0, 2, 1
9   RETI
    
```

Binary Form of VECT (Assuming Label is at Address 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 is 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 continues.

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 interrupt 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, CALC VX, CALC XV, CALC V

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	Don't care

Example

Subtract user variable #42 from user variable #65.

Mnemonic: CALCVV SUB, 65, 42

Binary Form of CALCVW 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, CALC VX, CALC XV, CALC VV

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	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, CALCWV

Mnemonic: CALCAV <operation>, <var>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
42	0 ADD – add <var> to accumulator	0 <var> (0...255)	0 (don't care)
	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>		

Reply in Direct Mode	
Status	Value
100 - OK	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 CALC VX (Calculate Using a User Variable and the X Register)

The CALC VX 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: CALC V, CALCA V, CALCVA, CALCX V, CALCV V

Mnemonic: CALC VX <operation>, <var>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
43	0 ADD – add X register to <var>	0 <var> (0...255)	0 (don't care)
	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			

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

Example

Subtract X register from user variable #27.

Mnemonic: CALC VX 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, CALC VX, CALCVW

Mnemonic: CALCXV <operation>, <var>

Binary representation

Binary Representation			
Instruction	Type	Motor/Bank	Value
44	0 ADD – add <var> to X register	0 <var> (0...255)	0 (don't care)
	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>		

Reply in Direct Mode	
Status	Value
100 - OK	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, CALC VX, CALC XV, CALC VV

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	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 restart the program at the given label. This command can be used to restart the TMCL program from anywhere in the program, and also out of subroutines or interrupt routines. *This command is intended for standalone operation only.*

Internal function: The TMCL interpreter is reset (the subroutine stack, interrupt stack, and 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:

```

1 Entry :
   MVP ABS, 0, 51200
3   CSUB Subroutine
   ...
5   ...
Subroutine :
7   RST Entry
   RSUB
    
```

Binary form of the RST Entry command when the label Entry is 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:

```

1   SGP 42, 2, 100
2 Loop:
3   MVP ABS, REL, 51200
4   WAIT POS, 0, 0
5   WAIT TICKS, 0, 500
6   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 specified condition is met. Otherwise, the program execution is 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 is saved to an internal stack. Afterwards, the program counter is 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 is 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	0 (don't care)	<jump address>
	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		

Example

Call a subroutine if a condition is met:

```

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

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

```

14 RunRight:
16     RORA 0
17     RSUB
18
19 MotorStop:
20     GAP 2, 0
21     JC ZE, MotorIsStopped
22     MST 0
23 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 is instructed to move to a specified relative or absolute position. The contents of the accumulator register are used as the target position. This command is non-blocking, which means that a reply is 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, see 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 runs in the opposite direction to take the shorter distance (caused by 32 bit overflow).

Internal function: Position mode is selected and the value stored in the accumulator register is copied to 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: Velocity mode is selected. Then, the velocity value is transferred to the target velocity (axis parameter #2).

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

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 is ignored. *This command is mainly intended for use in standalone mode.*

Internal function: The direct value supplied to this command is 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 is ignored. *This command is mainly intended for use in standalone mode.*

Internal function: The user variable specified by the x register is 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 is ignored. *This command is mainly intended for use in standalone mode.*

Internal function: The accumulator is 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 are implemented in C by Trinamic. Contact the sales department of Trinamic Motion Control GmbH & Co KG for 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 is executed (like all other commands also), and one additional reply that is 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 TMC1290, it only makes sense to have bit 0 set. So, always set this parameter to 1 with the TMC1290 module. With the type field set to 0, a position reached message is generated only for the next MVP command that follows this command. With type set to 1, a position reached message is 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 Reaches 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 that are called TMCL control commands. These commands can only be used in direct mode and not in a standalone program. So, they only have opcodes, but no mnemonics. Most of these commands are only used by the TMCL-IDE (to implement, example, 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 (also see section 8.6). 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	0 (don't care)
		1 – from specific address	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

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-1290 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-1290 module.

There are different parameter access types, like read only or read/write. Table 15 shows the different parameter access types used in the axis parameter tables.

Meaning of the Letters in the Access Column		
Access Type	Command	Description
R	GAP	Parameter readable
W	SAP, AAP	Parameter writable

Table 15: Meaning of the Letters in the Access Column

All Axis Parameters of the TMCM-1290 Module				
Number	Axis Parameter	Description	Range [Units]	Access
0	Target position	The desired target position in position mode.	-2147483648 ...2147483647 [μsteps]	RW
1	Actual position	The actual position of the motor. Stop the motor before overwriting it. Should normally only be overwritten for reference position setting.	-2147483648 ...2147483647 [μsteps]	RW
2	Target speed	The desired speed in velocity mode. Not valid in position mode.	-7999774 ...7999774 [pps]	RW
3	Actual speed	The actual speed of the motor.	-7999774 ...7999774 [pps]	R
4	Maximum positioning speed	The maximum speed used for positioning ramps.	0...7999774 [pps]	RW
5	Maximum acceleration	Maximum acceleration in positioning ramps. Acceleration and deceleration value in velocity mode.	117...7629278 [pps ²]	RW

Number	Axis Parameter	Description	Range [Units]	Access																																								
6	Maximum current	<p>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:</p> <table border="1"> <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> <p><i>The most important setting, as too high values can cause motor damage.</i></p>	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		0...255	RW
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																																										
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. Also see 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).	117...7629278 [pps ²]	RW																																								

Number	Axis Parameter	Description	Range [Units]	Access
16	Velocity V1	First acceleration/deceleration phase target velocity (in position mode only). Setting this value to 0 turns off the first acceleration/deceleration phase. In this case, only maximum acceleration (axis parameter 5) and maximum deceleration (axis parameter 17) are used.	0...1000000 [pps]	RW
17	Maximum deceleration	Maximum deceleration in positioning ramps. Used to decelerate from maximum positioning speed (axis parameter 4) to velocity V1.	117...7629278 [pps ²]	RW
18	Deceleration D1	Deceleration between V1 and VSTOP (in positioning mode only).	117...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). Do not set lower than 10.	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 0s to 2s. This setting avoids excess acceleration, example, from VSTOP to -VSTART.	0...65535 [0.000032s]	RW
22	Speed threshold for CoolStep/fullstep	Speed threshold for deactivating 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 to 1.	0/1	RW
29	Measured speed	Speed measured by the motor driver.	0...7999774 [pps]	R

Number	Axis Parameter	Description	Range [Units]	Access								
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								
34	Acceleration A2	Second acceleration between V1 and V2 (in position mode only).	117...7629278 [pps ²]	RW								
35	Velocity V2	Second acceleration/deceleration phase target velocity (in position mode only).	0...1000000 [pps]	RW								
36	Deceleration D2	Deceleration between V2 and V1 (in positioning mode only).	117...7629278 [pps ²]	RW								
37	Waiting time at V _{Max}	Minimum time for constant velocity segments.	0...65535 []	RW								
38	Virtual stop left	Position of the left virtual stop.	-2147483648 ...2147483647 [μsteps]	RW								
39	Virtual stop right	Position of the right virtual stop.	-2147483648 ...2147483647 [μsteps]	RW								
40	Virtual stop flags	Shows if the motor/encoder is at one of the virtual stops: <table border="1" style="margin-left: 20px;"> <tr> <td>0</td> <td>not at virtual stop</td> </tr> <tr> <td>1</td> <td>at left virtual stop</td> </tr> <tr> <td>2</td> <td>at right virtual stop</td> </tr> </table>	0	not at virtual stop	1	at left virtual stop	2	at right virtual stop	0...2	R		
0	not at virtual stop											
1	at left virtual stop											
2	at right virtual stop											
41	Virtual stop enable	Enables the virtual stops: <table border="1" style="margin-left: 20px;"> <tr> <td>0</td> <td>no virtual stop</td> </tr> <tr> <td>1</td> <td>only left virtual stop enabled</td> </tr> <tr> <td>2</td> <td>only right virtual stop enabled</td> </tr> <tr> <td>3</td> <td>both virtual stops enabled</td> </tr> </table>	0	no virtual stop	1	only left virtual stop enabled	2	only right virtual stop enabled	3	both virtual stops enabled	0...3	RW
0	no virtual stop											
1	only left virtual stop enabled											
2	only right virtual stop enabled											
3	both virtual stops enabled											
42	Virtual stop encoder mode	0 — use actual motor position for virtual stops 1 — use actual encoder position for virtual stops	0/1	RW								

Number	Axis Parameter	Description	Range [Units]	Access								
43	Compare position	First position where the TMCL position interrupt and the DIAG1 output as well as the position compare outputs (if configured) are triggered.	-2147483648 ...2147483647 [μsteps]	RW								
44	Compare repeat	This parameter defines a relative distance. If set to a value greater than 1, the position compare interrupt is raised every time a multiple of this number of microsteps are made. Each time the position interrupt is triggered, this value is added to or subtracted from the compare position (depending on the direction of movement) so that the position interrupt, DIAG1 output and position compare outputs can be triggered repeatedly during a motion.	0...16777215 [μsteps]	RW								
45	Position compare output mask	Using this bit vector, one or more general purpose outputs can be declared to be additional position compare outputs. These are then triggered by the position compare event defined by axis parameters #43 and #44. Each bit is used for one of the general purpose outputs. Each bit that is set to 1 defines the corresponding output to be a position compare output. <table border="1" data-bbox="523 1010 715 1196"> <thead> <tr> <th>Bit</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>GPIO0</td> </tr> <tr> <td>1</td> <td>GPIO1</td> </tr> <tr> <td>2</td> <td>GPIO2</td> </tr> </tbody> </table>	Bit	Output	0	GPIO0	1	GPIO1	2	GPIO2	0...7	RW
Bit	Output											
0	GPIO0											
1	GPIO1											
2	GPIO2											
46	Position compare pulse length	Defines the length of the position compare output pulses in milliseconds.	0...255 [ms]	RW								
47	Target reached output mask	This bit vector defines one or more general purpose outputs to be target reached outputs. A target reached output is high as long as the actual position (axis parameter #1) and the target position (axis parameter #0) are equal. Otherwise it is low. Each bit that is set to 1 defines the corresponding output to be a target reached output. <table border="1" data-bbox="523 1585 715 1771"> <thead> <tr> <th>Bit</th> <th>Output</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>GPIO0</td> </tr> <tr> <td>1</td> <td>GPIO1</td> </tr> <tr> <td>2</td> <td>GPIO2</td> </tr> </tbody> </table>	Bit	Output	0	GPIO0	1	GPIO1	2	GPIO2	0...7	RW
Bit	Output											
0	GPIO0											
1	GPIO1											
2	GPIO2											

Number	Axis Parameter	Description	Range [Units]	Access																		
48	Motion start input mask	<p>Using this bit vector, one or more general purpose inputs can be declared to be motion start inputs. If at least one input is defined as motion start input, each motion commanded by a ROR, ROL, or MVP command does not start until a low-to-high transition on at least one of the motion start inputs is detected. If more than one input is declared as motion start inputs, their signals are internally ORed. Each bit that is set to 1 defines the corresponding input to be a motion start input.</p> <table border="1"> <thead> <tr> <th>Bit</th> <th>Input</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>GPIO0</td> </tr> <tr> <td>1</td> <td>GPIO1</td> </tr> <tr> <td>2</td> <td>GPIO2</td> </tr> </tbody> </table>	Bit	Input	0	GPIO0	1	GPIO1	2	GPIO2	0..7	RW										
Bit	Input																					
0	GPIO0																					
1	GPIO1																					
2	GPIO2																					
127	Relative positioning option	<p>Start position for MVP REL command:</p> <table border="1"> <tbody> <tr> <td>0</td> <td>last target position</td> </tr> <tr> <td>1</td> <td>actual position</td> </tr> </tbody> </table>	0	last target position	1	actual position	0/1	RW														
0	last target position																					
1	actual position																					
140	Microstep resolution	<p>Microstep resolutions per full step:</p> <table border="1"> <tbody> <tr> <td>0</td> <td>fullstep</td> </tr> <tr> <td>1</td> <td>halfstep</td> </tr> <tr> <td>2</td> <td>4 microsteps</td> </tr> <tr> <td>3</td> <td>8 microsteps</td> </tr> <tr> <td>4</td> <td>16 microsteps</td> </tr> <tr> <td>5</td> <td>32 microsteps</td> </tr> <tr> <td>6</td> <td>64 microsteps</td> </tr> <tr> <td>7</td> <td>128 microsteps</td> </tr> <tr> <td>8</td> <td>256 microsteps</td> </tr> </tbody> </table>	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
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																					
160	Step interpolation enable	<p>The selected microstep resolution becomes extrapolated to 256 microsteps for smoothest motor operation also with low microstep resolution settings. This is useful in some legacy applications when low microstep settings are required. In new applications, always consider using the highest microstep resolution (256 microsteps) directly.</p> <p>0 - step interpolation off 1 - step interpolation on</p>	0/1	RW																		
162	Chopper blank time	<p>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.</p>	0..3	RW																		

Number	Axis Parameter	Description	Range [Units]	Access
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" disables 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 sets/returns the hysteresis end setting (hysteresis end value after a number of decrements). For "classic const. off time" chopper mode, this parameter sets/returns 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 sets/returns the hysteresis start setting (note that this value is an offset to the hysteresis end value). For "classic const. off time" chopper mode, this parameter sets/returns 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 5 μ s to 20 μ s fits. Off time setting for constant t Off chopper: $N_{CLK} = 12 + 32 \times t_{OFF}$ (Minimum is 64 clocks) Setting this parameter to zero completely disables all driver transistors and the motor can free-wheel.	0...15	RW
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: $([AP172] + 1) \times 32$ Upper StallGuard threshold: $([AP172] + [AP170] + 1) \times 32$	0...15	RW

Number	Axis Parameter	Description	Range [Units]	Access						
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						
178	Global scaler	Global scaling of the motor current. This value is multiplied with the current scaling to adapt the driver to a certain motor type. It should be chosen before tuning other parameters as it also influences the chopper hysteresis. This setting is intended for fine tuning the maximum motor current. The following settings are possible: <table border="1" data-bbox="510 1433 1085 1590"> <tbody> <tr> <td>0</td> <td>Full scale (100% of maximum)</td> </tr> <tr> <td>1...31</td> <td>Do not use</td> </tr> <tr> <td>32...255</td> <td>32/256...255/256 of maximum</td> </tr> </tbody> </table>	0	Full scale (100% of maximum)	1...31	Do not use	32...255	32/256...255/256 of maximum	0...255	RW
0	Full scale (100% of maximum)									
1...31	Do not use									
32...255	32/256...255/256 of maximum									

Number	Axis Parameter	Description	Range [Units]	Access								
179	Full scale current range	<p>The absolute maximum current of the motor driver depends on the externally connected resistor and this setting. This setting allows a basic adaptation of the driver current sensing to the motor current range. Select the lowest fitting range for best current precision. With a resistor of 12 kΩ the following settings are possible:</p> <table border="1"> <tr> <td>0</td> <td>$0.7A_{RMS}/1A_{peak}$</td> </tr> <tr> <td>1</td> <td>$1.4A_{RMS}/2A_{peak}$</td> </tr> <tr> <td>2</td> <td>$2.1A_{RMS}/3A_{peak}$</td> </tr> <tr> <td>3</td> <td>$2.1A_{RMS}/3A_{peak}$</td> </tr> </table>	0	$0.7A_{RMS}/1A_{peak}$	1	$1.4A_{RMS}/2A_{peak}$	2	$2.1A_{RMS}/3A_{peak}$	3	$2.1A_{RMS}/3A_{peak}$	0...3	RW
0	$0.7A_{RMS}/1A_{peak}$											
1	$1.4A_{RMS}/2A_{peak}$											
2	$2.1A_{RMS}/3A_{peak}$											
3	$2.1A_{RMS}/3A_{peak}$											
180	SmartEnergy actual current	<p>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</p>	0...31	R								
181	Stop on stall	Below this speed, motor is not stopped. Above this speed, motor stops 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								
185	Passive fast decay time	<p>Passive fast decay allows dampening of motor mid-range resonances. This parameter controls the duration of the fast decay phase inserted after bridge polarity change.</p> <p>0: off 1...15: time in units of 128 clock cycles.</p>	0...15	RW								
186	PWM threshold speed	The StealthChop feature is switched off when the actual velocity is higher than this value. It is 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								

Number	Axis Parameter	Description	Range [Units]	Access
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 - $f_{PWM} = 15.625\text{kHz}$ 1 - $f_{PWM} = 23.426\text{kHz}$ 2 - $f_{PWM} = 31.250\text{kHz}$ 3 - $f_{PWM} = 39.024\text{kHz}$	0...3	RW
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

Number	Axis Parameter	Description	Range [Units]	Access																				
193	Reference search mode	<table border="1"> <tr> <td>1</td> <td>Search left stop switch only.</td> </tr> <tr> <td>2</td> <td>Search right stop switch, then search left stop switch.</td> </tr> <tr> <td>3</td> <td>Search right stop switch, then search left stop switch from both sides.</td> </tr> <tr> <td>4</td> <td>Search left stop switch from both sides.</td> </tr> <tr> <td>5</td> <td>Search home switch in negative direction, reverse the direction when left stop switch reached.</td> </tr> <tr> <td>6</td> <td>Search home switch in positive direction, reverse the direction when right stop switch reached.</td> </tr> <tr> <td>7</td> <td>Search home switch in positive direction, ignore end switches.</td> </tr> <tr> <td>8</td> <td>Search home switch in negative direction, ignore end switches.</td> </tr> <tr> <td>9</td> <td>Search encoder index pulse in positive direction.</td> </tr> <tr> <td>10</td> <td>Search encoder index pulse in negative direction.</td> </tr> </table> <p>Additional functions:</p> <ul style="list-style-type: none"> • 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	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 positive direction, ignore end switches.	8	Search home switch in negative direction, ignore end switches.	9	Search encoder index pulse in positive direction.	10	Search encoder index pulse in negative direction.	1..10	RW
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 positive direction, ignore end switches.																							
8	Search home switch in negative direction, ignore end switches.																							
9	Search encoder index pulse in positive direction.																							
10	Search encoder index pulse in negative direction.																							
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 [μ steps]	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 [μ steps]	R																				

Number	Axis Parameter	Description	Range [Units]	Access														
201	Encoder mode	<p>A combination of the following values:</p> <table border="1"> <tr> <td>Bit 0</td> <td>Required channel A polarity for null channel event. (0: low, 1: high)</td> </tr> <tr> <td>Bit 1</td> <td>Required channel B polarity for null channel event. (0: low, 1: high)</td> </tr> <tr> <td>Bit 2</td> <td>Null channel polarity. (0:low, 1: high)</td> </tr> <tr> <td>Bit 3</td> <td>Ignore channel A and B polarity for null channel event.</td> </tr> <tr> <td>Bit 4</td> <td>Clear/latch encoder at every null channel event.</td> </tr> <tr> <td>Bit 5</td> <td>Clear/latch encoder only at next null channel event.</td> </tr> <tr> <td>Bit 8</td> <td>0: Latch encoder and actual position on N event. 1: Also clear encoder counter on N event.</td> </tr> </table>	Bit 0	Required channel A polarity for null channel event. (0: low, 1: high)	Bit 1	Required channel B polarity for null channel event. (0: low, 1: high)	Bit 2	Null channel polarity. (0:low, 1: high)	Bit 3	Ignore channel A and B polarity for null channel event.	Bit 4	Clear/latch encoder at every null channel event.	Bit 5	Clear/latch encoder only at next null channel event.	Bit 8	0: Latch encoder and actual position on N event. 1: Also clear encoder counter on N event.	0...511	RW
Bit 0	Required channel A polarity for null channel event. (0: low, 1: high)																	
Bit 1	Required channel B polarity for null channel event. (0: low, 1: high)																	
Bit 2	Null channel polarity. (0:low, 1: high)																	
Bit 3	Ignore channel A and B polarity for null channel event.																	
Bit 4	Clear/latch encoder at every null channel event.																	
Bit 5	Clear/latch encoder only at next null channel event.																	
Bit 8	0: Latch encoder and actual position on N event. 1: Also clear encoder counter on N event.																	
202	Motor fullstep resolution	The fullstep resolution of the stepper motor used with the drive. The default value is 200.	0...32768 [full steps]	RW														
204	Freewheeling mode	<p>Stand still option when the standby current (parameter 7) is set to zero and StealthChop is active.</p> <table border="1"> <tr> <td>0</td> <td>normal operation</td> </tr> <tr> <td>1</td> <td>freewheeling</td> </tr> <tr> <td>2</td> <td>coil shorted using low side drivers</td> </tr> <tr> <td>3</td> <td>coil shorted using high side drivers</td> </tr> </table>	0	normal operation	1	freewheeling	2	coil shorted using low side drivers	3	coil shorted using high side drivers	0...3	RW						
0	normal operation																	
1	freewheeling																	
2	coil shorted using low side drivers																	
3	coil shorted using high side drivers																	
206	Actual load value	Readout of the actual load value used for stall detection (StallGuard2).	0...1023	R														
207	Extended error flags	<p>A combination of the following values:</p> <table border="1"> <tr> <td>1</td> <td>StallGuard error</td> </tr> <tr> <td>2</td> <td>deviation error</td> </tr> </table> <p>These error flags are cleared automatically when this parameter is read out or when a motion command is executed.</p>	1	StallGuard error	2	deviation error	0...3	R										
1	StallGuard error																	
2	deviation error																	

Number	Axis Parameter	Description	Range [Units]	Access																
208	Motor driver error flags	A combination of the following values: <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">Bit 0</td> <td>StallGuard2 status (1: stall detected)</td> </tr> <tr> <td>Bit 1</td> <td>Overtemperature (1: driver is shut down due to overtemperature)</td> </tr> <tr> <td>Bit 2</td> <td>Overtemperature pre-warning (1: temperature threshold is exceeded)</td> </tr> <tr> <td>Bit 3</td> <td>Short to ground A (1: short condition detected, driver currently shut down)</td> </tr> <tr> <td>Bit 4</td> <td>Short to ground B (1: short condition detected, driver currently shut down)</td> </tr> <tr> <td>Bit 5</td> <td>Open load A (1: no chopper event has happened during the last period with constant coil polarity)</td> </tr> <tr> <td>Bit 6</td> <td>Open load B (1: no chopper event has happened during the last period with constant coil polarity)</td> </tr> <tr> <td>Bit 7</td> <td>Stand still (1: no step pulse occurred during the last 2²⁰ clock cycles)</td> </tr> </table>	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
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)																			
209	Encoder position	Encoder counter value.	-2147483648 ...2147483647 [μsteps]	RW																
210	Encoder resolution	Encoder counts per round. A negative value reverses the encoder counting direction. The minimum/maximum values also depend on the motor fullstep and microstep resolution.	-2147483648 ...2147483647 [cpr]	RW																
212	Maximum encoder deviation	When the actual position (parameter #1) and the encoder position (parameter #209) differ more than set, here the motor is stopped. Setting this parameter to zero turns off this feature.	0 ...2147483647 [encoder steps]	RW																
214	Power down delay	Standstill period before the current is ramped down to standby current. The standard value is 0, which means that the current is immediately ramped down to standby current using the power down ramp (see parameter #31) after the motor has stopped. The delay time is given in units of 10ms, which means that, for example, a value of 200 results in a delay time of 2000ms.	0...417 [10ms]	RW																
251	Reverse shaft	Reverse the rotation direction of the motor shaft.	0/1	RW																

Table 16: All TMCM-1290 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

Global parameters in bank 0 configure all settings that affect the overall behaviour of a module. These are things like the serial address, RS485 baud rate, or CAN bit rate (where appropriate). Change these parameters per 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 always stores 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 not supported by the PC) may lead to a situation in which the module cannot be reached any more. In such a case refer to the TMCM-1290 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 17 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 17: Meaning of the Letters in the Access Column

All Global Parameters of the TMCM-1290 Module in Bank 0						
Number	Global Parameter	Description			Range [Units]	Access
65	RS485 baud rate	0	9600		0...11	RWA
		1	14400			
		2	19200			
		3	28800			
		4	38400			
		5	57600			
		6	76800			
		7	115200	Default		
		8	230400			
		9	250000			
		10	500000			
		11	1000000			
66	Serial address	Module (target) address for RS485.			1...255	RWA
68	Serial heartbeat	Serial heartbeat for RS485 interface. When this time limit is reached and no further command is received by the module, the motor is stopped. Setting this parameter to 0 (default) turns off the serial heartbeat function.			0...65535 [ms]	RWA
76	Serial host address	Host address used in the reply telegrams sent back through RS485.			0...255	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
78	I/O mode	Bit vector that selects input or output mode for each general purpose input/output. A bit that is clear selects input mode and a bit that is set selects output mode. Add up the values of the bits that are to be set.			0...7	RWA
		Bit	Value	Input/Output		
		0	1	GPIO0		
		1	2	GPIO1		
		2	4	GPIO2		

Number	Global Parameter	Description	Range [Units]	Access
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 is erased first!	0/1/2/3	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 and restored on startup	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
87	Serial secondary address	Second module (target) address for RS485. Setting this parameter to 0 switches off the secondary address. See section 7.9.	0...255	RWA
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 is suppressed when this parameter is set to 1. This parameter cannot be stored to EEPROM and is reset to 0 on startup. The reply is not suppressed for GAP, GGP, and GIO commands.	0/1	RW

Table 18: All Global Parameters of the TMC1290 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. See table 17 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	RW

Table 19: User Variables in Bank 2

5.4 Bank 3

Bank 3 contains interrupt parameters. Some interrupts need configuration (example, 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 20 shows all interrupt parameters that can be set. See table 17 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

Number	Global Parameter	Description	Range [Units]	Access
41	Input 2 trigger transition	0 = off, 1 = low-high, 2 = high-low, 3 = both	0...3	RW

Table 20: Interrupt Parameters in Bank 3

6 Module Specific Hints

This section contains some hints specific to the TMCM-1290 module.

6.1 TMCL over SPI

The TMCM-1290 module is also equipped with an SPI. Instead of the RS485 interface, this interface can also be used for controlling the TMCM-1290 module from a host microcontroller using TMCL communication. A TMCL command sent over SPI uses an SPI telegram with a length of eight bytes. The interface uses SPI mode 3. Each SPI datagram consists of the following data:

- The first byte contains the command opcode.
- The second byte contains the type byte.
- The third byte contains the motor/bank byte.
- Bytes 4...7 contain the 32-bit value (with MSB first).
- The last byte contains the checksum over the first seven bytes, which is formed by adding up all these bytes and then taking the LSB of the result as the checksum byte.

When using SPI communication, the RS485_DIR pin is used as SPI_BUSY pin. At the beginning of the SPI datagram, the SPI_BUSY pin is set high. After the command is processed by the module (which can take between 0.1ms and 5ms, depending on the command), the SPI_BUSY pin is set low again. Now the result can be read by sending an SPI datagram containing eight FF_h bytes.

Instead of fetching the result using an SPI datagram of eight FF_h bytes, it is also possible to send the next command. The receive data of the SPI datagram then contains the result of the last command.

Note

It is not allowed to send any SPI data while the SPI_BUSY pin is high. This internally overwrites the result of the last command or even overwrites a command not yet processed. So, before sending any new SPI datagram, always wait until the SPI_BUSY pin has gone low again.

For normal TMCL replies, the eight reply bytes contain the following data:

- The first byte always contains FF_h.
- The second byte contains the status code (which mostly is 100).
- The third byte contains the opcode of the command.
- Bytes 4...7 contain the 32-bit value (with MSB first).
- The last byte contains the checksum over the first seven bytes, which is formed by adding up all these bytes and then taking the LSB of the result as the checksum byte.

For command 134 and command 136 type 0 (both are mostly used only by the TMCL-IDE), the first byte contains the first data byte (but not FF_h), and the other bytes contain the other data bytes. A checksum is not used with such replies.

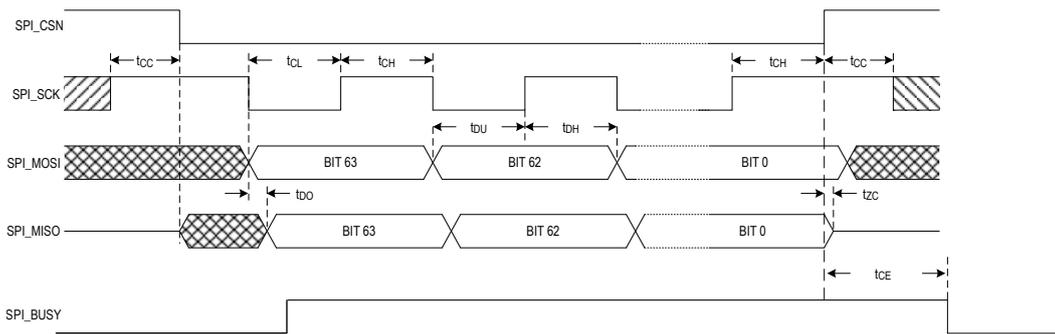


Figure 4: TMCL SPI Timing

- t_{CE}: TMCL command execution time
- t_{CL}: clock low time
- t_{CH}: clock high time
- t_{CC}: SCK valid before or after change of CSN
- t_{DO}: MOSI valid time after SCK falling edge

The TMCL command execution time depends on the command (0.1ms...5ms). For all other timing parameters, see the TMCM-1290 hardware manual.

6.2 Motor Current Settings

The stepper motor controller/driver has three stages of setting the motor current:

- Full scale current range (axis parameter #179) together with the external resistor connected to the IREF pin: this limits the overall maximum current. See the TMCM-1290 hardware manual for more about the external resistor.
- Global scaler (axis parameter #178): this axis parameter scales the full scale current. Use this to fine tune the maximum possible current to best suit the maximum current of the motor.
- Maximum current (axis parameter #6) and standby current (axis parameter #7): these two parameters set the motor current used when the motor is running and when the motor is stopped. They are both derived from the maximum current set by the global scaler.

6.3 General Purpose Inputs and Outputs

The TMCM-1290 module is equipped with three general purpose digital lines. Each of these can either be used as an input or output. This can be selected using global parameter #78. These inputs and outputs are named GPIO0, GPIO1, and GPIO2.

Global parameter #78 is a bit vector, where each bit determines if the corresponding digital line is an input or an output. A bit that is clear means input, and a bit that is set means output.

Table 21 shows the relationship between the bits of global parameter #78 and the digital lines. Just add up the values of all bits to be set (that is, of those digital lines to be used as outputs) and set global parameter #78 to this value.

Selection of Inputs and Outputs		
Bit	Value	Digital I/O line
0	1	GPIO0
1	2	GPIO1
2	4	GPIO2

Table 21: Selection of Inputs and Outputs

For example, setting global parameter #78 to 5 programs GPIO0 and GPIO2 as outputs and GPIO1 as an input.

6.4 Position Compare Outputs

The DIAG1 output and general purpose outputs can be used as position compare outputs. This is useful if some other device (for example a camera or a shutter) has to be triggered when a certain position is reached (for example, once per round). This can be configured using axis parameters #43 and #44. Bear in mind that when axis parameter #44 is set to a value >1, this value is (depending on the direction of motion) added to or subtracted from axis parameter #43 each time the compare position is reached. So, when moving back the same way, the first compare position (axis parameter #43) has to be set accordingly.

The signal on output DIAG1 is only high for one clock cycle each time the compare position is reached. If this time should be too short, either connect a monoflop in between or use the additional position compare outputs that can be configured using axis parameters #45 and #46. When using a general purpose output as a position compare output, be sure that this pin is also configured to output mode (using global parameter #78).

6.5 Target Reached Outputs

General purpose outputs can also be used as target reached outputs. This can be controlled using axis parameter #47. Whenever the target position and the actual position are equal, the target reached output is high. When target position and actual position are not equal, the target reached output is low. So, target reached outputs always have the same state as axis parameter #8. When using a general purpose output as a target reached output, be sure that this pin is also configured to output mode (using global parameter #78).

6.6 Motion Start Inputs

General purpose inputs can also be used as motion start inputs. This can be selected using axis parameter #48. When at least one input is selected as motion start input, each motion commanded by an ROR, ROL, or MVP command is not started until a low-to-high transition is detected on the motion start input. When more than one input is configured as motion start inputs, the signals on these pins are internally ORed, so that a low-high transition on at least one of the inputs starts the motion. This introduces a simple way of synchronizing the start of a motion on multiple TMCM-1290 modules:

- Configure one input as motion start input on each involved module.
- Connect all these pins together to one output pin of the host microcontroller.
- Set this pin low.
- Send the appropriate motion command to each of the modules.

- Set this pin high. The motion then starts simultaneously on all involved modules.

This method works independently of the used interface (RS485, UART, or SPI).

6.7 Synchronized Start of Motion Using RS485 Only

When all TMCM-1290 modules are connected to the host controller using the same RS485 bus, it is also possible to start a motion simultaneously using just TMCL commands through the RS485 bus. For such purposes, the secondary serial address (global parameter #87) can be used. Set this parameter to the same value (>0) on all modules. When now a command is being sent using this address, it reaches all modules at the same time. When sending a motion command (ROR, ROL, MVP, MST) to all modules this way, a motion can also be started or stopped simultaneously. To avoid bus contention, there is no reply for commands sent using the secondary serial address.

7 Hints and Tips

This chapter gives some hints and tips on using the functionality of TMCL, for example, how to use and parameterize the built-in reference search algorithm. This chapter also provides basic information about StallGuard2™ and CoolStep™.

7.1 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 22 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 22: 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.

7.1.1 Mode 1

Reference search mode 1 only searches the left end switch. Select this mode by setting axis parameter #193 to 1. Figure 5 illustrates this.

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

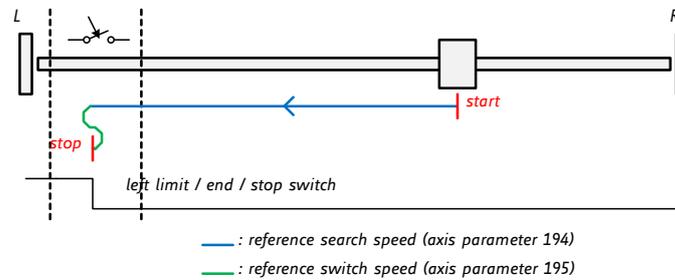


Figure 5: Reference Search Mode 1

7.1.2 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 6 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 (that is, set axis parameter #193 to 66) to search the left end switch first and then use the right end switch as the zero point.

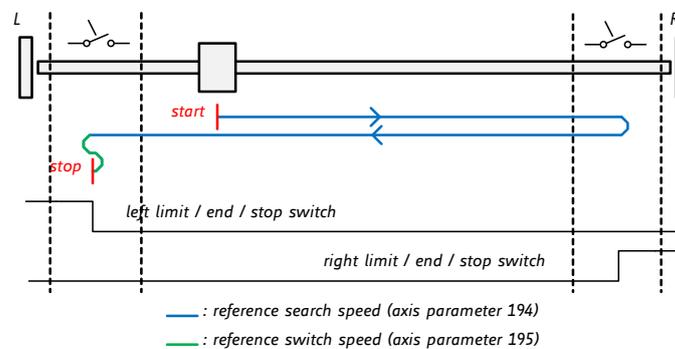


Figure 6: Reference Search Mode 2

7.1.3 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 7 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 (that is, 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.

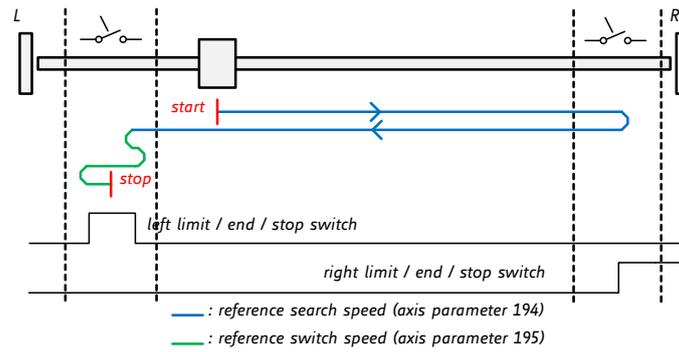


Figure 7: Reference Search Mode 3

7.1.4 Mode 4

Reference search mode 4 searches the left end switch only, but from both sides so that the middle of the switch is found and used as the zero point. This is shown in figure 8.

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

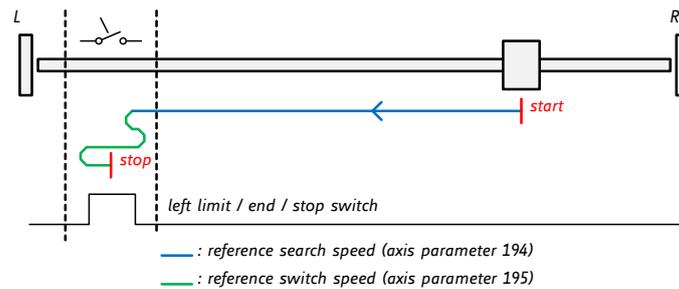


Figure 8: Reference Search Mode 4

7.1.5 Mode 5

Reference search mode 5 searches the home switch in negative direction. The search direction is reversed if the left limit switch is reached. This is shown in figure 9.

Add 128 to the mode number (that is, set axis parameter #193 to 133) to reverse the polarity of the home switch input.

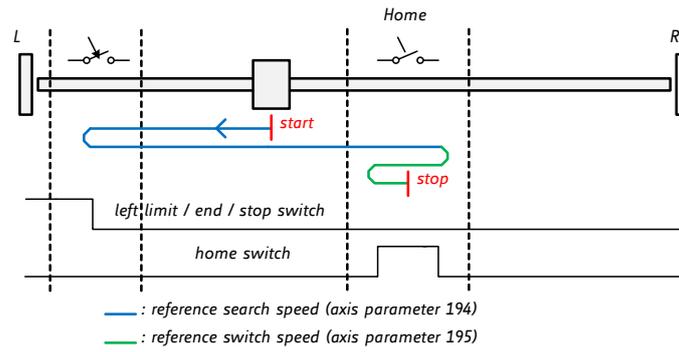


Figure 9: Reference Search Mode 5

7.1.6 Mode 6

Reference search mode 6 searches the home switch in positive direction. The search direction is reversed if the right limit switch is reached. This is shown in figure 10.

Add 128 to the mode number (that is, set axis parameter #193 to 134) to reverse the polarity of the home switch input.

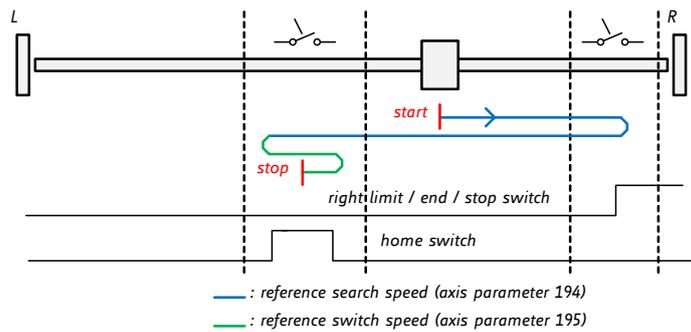


Figure 10: Reference Search Mode 6

7.1.7 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 is found and used as the zero point. Figure 11 illustrates this.

Add 128 to the mode number (that is, set axis parameter #193 to 135) to reverse the polarity of the home switch input.

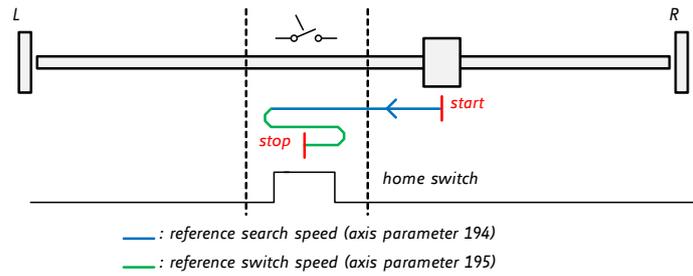


Figure 11: Reference Search Mode 7

7.1.8 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 is found and used as the zero point. Figure 12 illustrates this.

Add 128 to the mode number (that is, set axis parameter #193 to 136) to reverse the polarity of the home switch input.

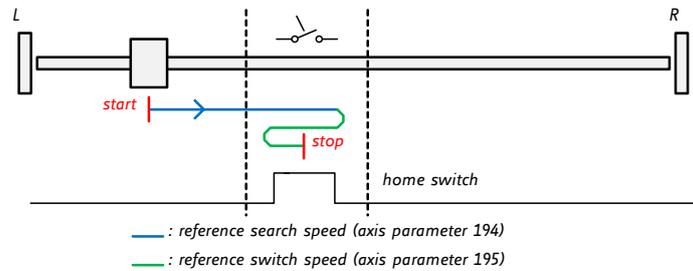


Figure 12: Reference Search Mode 8

7.1.9 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.

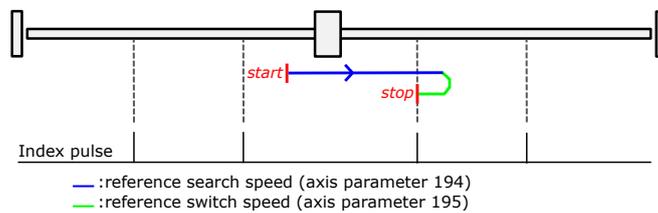


Figure 13: Reference Search Mode 9

7.1.10 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.

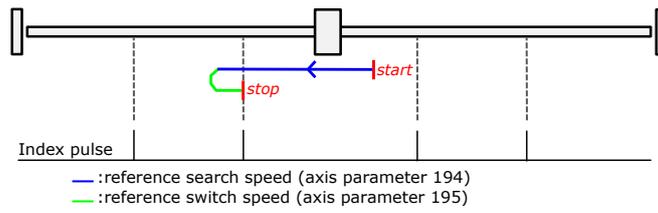


Figure 14: Reference Search Mode 10

7.2 Using Encoders

With the TMCM-1290, it is also possible to use encoders. Note that the TMCM-1290 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-1290, the encoder cannot be used for a servo-like closed-loop operation (but there are a number of other Trinamic modules that can do real closed-loop operation). Consider the following things for using encoders with the TMCM-1290:

- 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 automatically converts the encoder resolution to the motor microstep resolution so that the encoder position is the same as the motor position.
- The encoder position can be read using axis parameter #209. This parameter is also writable, example, 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 is found) can be controlled using axis parameter #201.

7.3 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 is 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.

7.4 CoolStep

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

Figure 15 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 that concern CoolStep and StallGuard2. The complete list of axis parameters in chapter 4 contains further parameters that offer more configuration options.

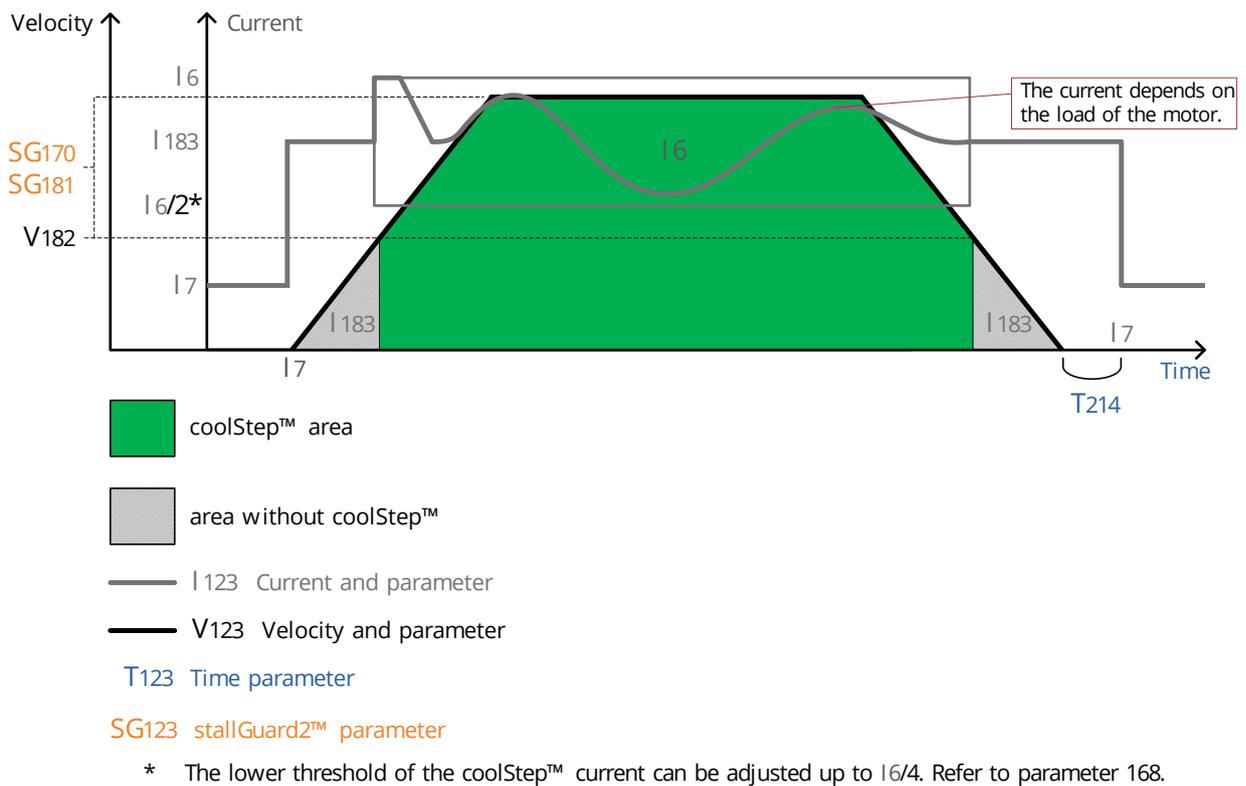


Figure 15: CoolStep Adjustment Points and Thresholds

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; example, 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 upper threshold for StallGuard2 reading. Above the upper threshold, the motor current becomes decreased.
SG181	Stop on stall	Below this speed, motor is not stopped. Above this speed motor stops in case StallGuard2 load value reaches zero.
V182	smartEnergy threshold speed	Above this speed, CoolStep becomes enabled.

Number	Axis Parameter	Description
T214	Power down delay	Standstill period before the current is changed down to standby current. The standard value is 200 (which means 2000ms).

Table 23: CoolStep Adjustment Points and Thresholds

7.5 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-1290 are given in microsteps per second (also called pulse per second or pps). Acceleration and deceleration units are given in pps².

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.

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:

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

To convert from rps to rpm, use:

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

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 results in 1rps (60rpm).

7.6 EightPoint Ramp

The TMCM-1290 is equipped with a motion controller that supports TRINAMIC's EightPoint ramp technology. See also section 1.3 for more information about the EightPoint ramp. The EightPoint ramp can be configured using the following axis parameters:

Parameter Name	TMCL Axis Parameter Number
Start velocity (VSTART)	19
Acceleration A1	15
Velocity V1	16
Acceleration A2	34
Acceleration V2	35
Maximum acceleration (AMAX)	5
Maximum positioning velocity (VMAX)	4
Maximum deceleration (DMAX)	17
Deceleration D2	36
Deceleration D1	18
Stop velocity (VSTOP)	20
Ramp wait time (TZEROWAIT)	21
Waiting time at V_{MAX} (TVMAX)	37

Table 24: EightPoint Ramp Parameters

Setting the velocity V1 (axis parameter #16) to zero switches off the EightPoint ramp. In this case, a trapezoidal ramp defined by parameters 5, 4, 17, 19, 20, and 21 is used.

Note

The EightPoint ramp is only used in positioning mode (MVP command). Velocity mode (ROR/ROL commands) always uses a trapezoidal ramp, defined just by the maximum 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 are not used in velocity mode.

7.7 StealthChop

The TMCM-1290 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 automatically switches 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 is always higher than the chopping 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 is automatically switched off.
- Axis parameter #186 (PWM threshold speed): This is the maximum speed for StealthChop. Above this speed, StealthChop is automatically switched off. This means that the actual speed has to be lower than the values of axis parameters #182 and #186 to activate StealthChop.
- Axis parameter #22 (speed threshold for deactivating 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 started with StealthChop is to set axis parameter #182 and axis parameter #186 to the same speed values (example: 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
```

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.

7.8 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 is just lowered to the standby current set by axis parameter #7.
- Freewheeling: Setting axis parameter #204 to 1 activates freewheeling mode. When the motor is not running, the motor coils are switched off completely so that the motor can also be moved manually. Note that in such a case the position is lost. Either an encoder has to be used or a new reference search is 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 is completely switched off, but the coils are shorted, which blocks the motor and thus also holds 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 is completely switched off, but the coils are shorted, which blocks the motor and thus also holds the position.

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 to be able to use one of the freewheeling modes. The following example program shows how to activate freewheeling mode #1:

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

7.9 Secondary Serial Address

Using the secondary serial address (global parameter #87) is an easy to use method to synchronize multiple TMCL modules controlled by one RS485 host. To use this method, set the secondary serial address of all modules connected to the bus to the same value (but not to 0 or to the value used for global parameter #66). All the serial addresses (global parameter #66) must be set to different values. Each command sent using the secondary address is then executed by all modules at the same time, and there is no bus contention as no replies are being sent. Using the normal serial address each module can still be addressed separately and also still sends back a reply.

8 TMCL Programming Techniques and Structure

8.1 Initialization

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

8.2 Main Loop

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

There are exceptions to this, for example, when TMCL routines are called from a host in direct mode.

So, most (but not all) standalone TMCL programs look like this:

```

1 //Initialization
2 SAP 4, 0, 50000 //define maximum positioning speed
3 SAP 5, 0, 10000 //define maximum acceleration
4
5 MainLoop:
6 //do something, in this example just running between two positions
7 MVP ABS, 0, 5000
8 WAIT POS, 0, 0
9 MVP ABS, 0, 0
10 WAIT POS, 0, 0
11 JA MainLoop //end of the main loop => run infinitely

```

8.3 Using Symbolic Constants

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

```

1 //Define some constants
2 #include TMCLParam.tmc
3 MaxSpeed = 50000
4 MaxAcc = 10000
5 Position0 = 0
6 Position1 = 500000
7
8 //Initialization
9 SAP APMaxPositioningSpeed, Motor0, MaxSpeed
10 SAP APMaxAcceleration, Motor0, MaxAcc
11
12 MainLoop:
13 MVP ABS, Motor0, Position1
14 WAIT POS, Motor0, 0
15 MVP ABS, Motor0, Position0
16 WAIT POS, Motor0, 0

```

17 JA MainLoop

Go through 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. Change the definition of the constant and do not change all occurrences of it in the program.

8.4 Using Variables

The user variables can be used if variables are needed in the 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 (example, 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, for example, to store the result of a calculation.
- The STGP command stores the contents of a user variable 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 are restored from the EEPROM automatically on startup (default setting) or not (user variables are then initialized with 0 instead).

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.

8.5 Using Subroutines

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

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

8.6 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, example, 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 (example, in its main loop) and so the TMCL program can react on such changes. Vice versa, a TMCL program can change a user variable 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 do 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
2 Func1:  JA Func1Start
  Func2:  JA Func2Start
4 Func3:  JA Func3Start

6 Func1Start:
  MVP ABS, 0, 1000
8  WAIT POS, 0, 0
  MVP ABS, 0, 0
10 WAIT POS, 0, 0
  STOP

12 Func2Start:
14  ROL 0, 500
  WAIT TICKS, 0, 100
16  MST 0
  STOP

18 Func3Start:
20  ROR 0, 1000
  WAIT TICKS, 0, 700
22  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. See the addresses of the TMCL labels (needed for the run commands) using the "Generate symbol file function" of the TMCL-IDE.

8.7 Make the TMCL Program Start Automatically

For standalone operation, the module has to start the TMCL program in its memory automatically after power-on. 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".

9 Figures Index

1	StallGuard2 Load Measurement as a Function of Load	6	8	Reference Search Mode 4	123
2	Energy Efficiency Example with CoolStep	6	9	Reference Search Mode 5	124
3	Typical Motion Profile with TRINAMIC EightPoint Motion Controller	7	10	Reference Search Mode 6	124
4	TMCL SPI Timing	118	11	Reference Search Mode 7	125
5	Reference Search Mode 1	122	12	Reference Search Mode 8	125
6	Reference Search Mode 2	122	13	Reference Search Mode 9	125
7	Reference Search Mode 3	123	14	Reference Search Mode 10	126
			15	CoolStep Adjustment Points and Thresholds	128

10 Tables Index

1	Important Axis Parameters	9	16	All TMCM-1290 Axis Parameters	111
2	TMCL Command Format	12	17	Meaning of the Letters in the Access Column	112
3	TMCL Reply Format	13	18	All Global Parameters of the TMCM-1290 Module in Bank 0	114
4	TMCL Status Codes	13	19	User Variables in Bank 2	115
5	Overview of All TMCL Commands	17	20	Interrupt Parameters in Bank 3	116
6	Motion Commands	17	21	Selection of Inputs and Outputs	119
7	Parameter Commands	18	22	Reference Search Modes	121
8	Branch Commands	18	23	CoolStep Adjustment Points and Thresholds	130
9	I/O Port Commands	18	24	EightPoint Ramp Parameters	132
10	Calculation Commands	19	25	Firmware Revision	144
11	Interrupt Processing Commands	19	26	Document Revision	144
12	Interrupt Vectors	20			
13	New TMCL Commands	22			
14	TMCL Control Commands	98			
15	Meaning of the Letters in the Access Column	99			

11 Supplemental Directives

11.1 Producer Information

11.2 Copyright

ADI Trinamic/Trinamic Motion Control GmbH & Co. KG owns the content of this user manual in its entirety, including but not limited to pictures, logos, trademarks, and resources.

Redistribution of sources or derived formats (for example, Portable Document Format or Hypertext Markup Language) must retain the above copyright notice, and the complete data sheet, user manual, and documentation of this product including associated application notes; and a reference to other available product-related documentation.

11.3 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 ADI Trinamic or by other manufacturers, whose products are used or referred to in combination with ADI Trinamic's products and ADI 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.

11.4 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.

11.5 Disclaimer: Life Support Systems

ADI Trinamic/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 ADI Trinamic/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.

11.6 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 ADI Trinamic's/Trinamic Motion Control GmbH & Co. KG specific written consent.

ADI Trinamic/Trinamic Motion Control GmbH & Co. KG 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 ADI Trinamic/Trinamic Motion Control GmbH & Co. KG.

ADI Trinamic/Trinamic Motion Control GmbH & Co. KG conveys no patent, copyright, mask work right or other trade mark right to this product. ADI Trinamic/Trinamic Motion Control GmbH & Co. KG 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.

11.7 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.analog.com.

12 Revision History

12.1 Firmware Revision

Version	Date	Description
V1.00	2022-DEC-21	Initial version
V1.01	2024-JAN-30	Higher RS485 baud rate accuracy

Table 25: Firmware Revision

12.2 Document Revision

Version	Date	Description
0	01/24	Initial version

Table 26: Document Revision