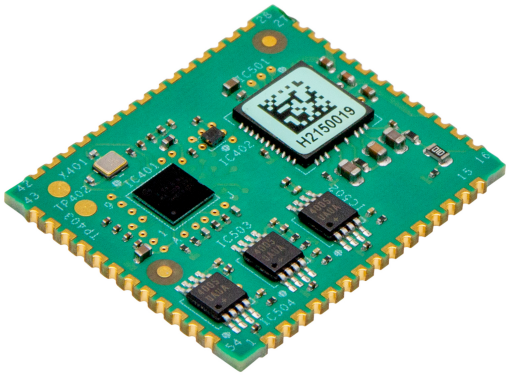
TMCM-1690 TMCL™ Firmware Manual

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

The TMCM-1690 is a single axis FOC servo controller gate driver module for 3-phase BLDC and DC motors with up to 1.5A gate drive current and +60V (+48 V nominal) supply. It offers UART (RS232-/RS485-ready), CAN, and EtherCAT® interfaces with TMCL, CANopen, or CANopen-over-EtherCAT protocol support for communication. TMCM-1690 supports incremental encoders, digital hall sensors, and absolute encoder as position feedback.



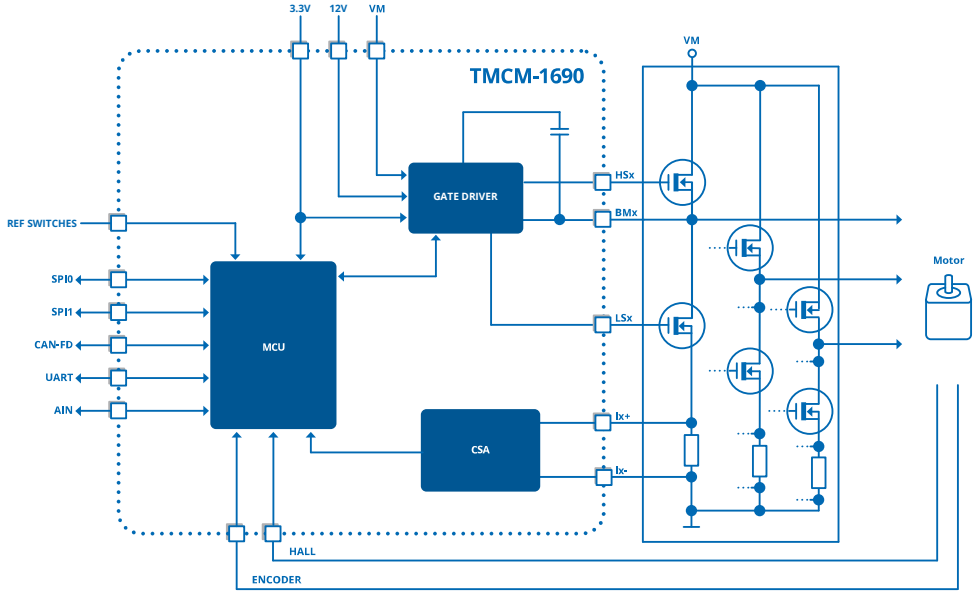
Features

- Supply voltage +10V to +60V DC
- FOC servo controller gate driver module for BLDC and DC motors
- 0.5A/1.0A/1.5A gate drive current
- Up to 120kHz PWM frequency
- Onboard current-sense amplifiers
- Supports UART (RS232/RS485-ready), CAN, and EtherCAT™ interface
- Supports incremental encoders (ABN), digital HALL sensors, absolute SPI encoders
- Reference switch inputs
- Compact size 27mm x 22.5mm

Applications

- Robotics
- Laboratory Automation
- Manufacturing
- Factory Automation
- Servo Drives
- Motorized Tables and Chairs
- Industrial BLDC and DC Motor Drives

Simplified Block Diagram



ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

Read entire documentation.

Contents

1 Features **5**

2 First Steps with TMCL **6**
2.1 Basic Setup . . . 6
2.2 Using the TMCL Direct Mode . . . 6
2.3 Changing Axis Parameters . . . 6
2.4 Testing with a Simple TMCL Program . . . 7

3 TMCL and the TMCL-IDE — An Introduction **9**
3.1 Binary Command Format . . . 9
3.1.1 Checksum Calculation . . . 10
3.2 Reply Format . . . 10
3.2.1 Status Codes . . . 11
3.3 Standalone Applications . . . 11
3.4 TMCL Command Overview . . . 13
3.5 TMCL Commands by Subject . . . 15
3.5.1 Motion Commands . . . 15
3.5.2 Parameter Commands . . . 15
3.5.3 Branch Commands . . . 16
3.5.4 I/O Port Commands . . . 16
3.5.5 Calculation Commands . . . 17
3.5.6 Interrupt Processing Commands . . . 17
3.5.7 New TMCL Commands . . . 20
3.6 Detailed TMCL Command Descriptions . . . 21
3.6.1 ROR (Rotate Right) . . . 21
3.6.2 ROL (Rotate Left) . . . 22
3.6.3 MST (Motor Stop) . . . 23
3.6.4 MVP (Move to Position) . . . 24
3.6.5 SAP (Set Axis Parameter) . . . 27
3.6.6 GAP (Get Axis Parameter) . . . 28
3.6.7 STAP (Store Axis Parameter) . . . 29
3.6.8 RSAP (Restore Axis Parameter) . . . 30
3.6.9 SGP (Set Global Parameter) . . . 31
3.6.10 GGP (Get Global Parameter) . . . 33
3.6.11 STGP (Store Global Parameter) . . . 35
3.6.12 RSGP (Restore Global Parameter) . . . 36
3.6.13 SIO (Set Output) . . . 37
3.6.14 GIO (Get Input) . . . 39
3.6.15 CALC (Calculate) . . . 42
3.6.16 COMP (Compare) . . . 44
3.6.17 JC (Jump Conditional) . . . 45
3.6.18 JA (Jump Always) . . . 47
3.6.19 CSUB (Call Subroutine) . . . 48
3.6.20 RSUB (Return from Subroutine) . . . 49
3.6.21 WAIT (Wait for an Event to Occur) . . . 50
3.6.22 STOP (Stop TMCL Program Execution – End of TMCL Program) . . . 52
3.6.23 SCO (Set Coordinate) . . . 53
3.6.24 GCO (Get Coordinate) . . . 54
3.6.25 CCO (Capture Coordinate) . . . 56
3.6.26 ACO (Accumulator to Coordinate) . . . 57
3.6.27 CALCX (Calculate Using the X Register) . . . 58
3.6.28 AAP (Accumulator to Axis Parameter) . . . 60

3.6.29 AGP (Accumulator to Global Parameter) . . . 61
3.6.30 CLE (Clear Error Flags) . . . 62
3.6.31 EI (Enable Interrupt) . . . 64
3.6.32 DI (Disable Interrupt) . . . 65
3.6.33 VECT (Define Interrupt Vector) . . . 66
3.6.34 RETI (Return from Interrupt) . . . 68
3.6.35 CALCVV (Calculate Using Two User Variables) . . . 69
3.6.36 CALCVA (Calculate Using a User Variable and the Accumulator Register) . . . 71
3.6.37 CALCAV (Calculate Using the Accumulator Register and a User Variable) . . . 73
3.6.38 CALCVX (Calculate Using a User Variable and the X Register) . . . 75
3.6.39 CALCXV (Calculate Using the X Register and a User Variable) . . . 77
3.6.40 CALCV (Calculate Using a User Variable and a Direct Value) . . . 79
3.6.41 RST (Restart) . . . 81
3.6.42 DJNZ (Decrement and Jump if not Zero) . . . 82
3.6.43 CALL (Conditional Subroutine Call) . . . 83
3.6.44 MVPA (Move to Position Specified by Accumulator Register) . . . 85
3.6.45 ROLA (Rotate Left Using the Accumulator Register) . . . 87
3.6.46 RORA (Rotate Right Using the Accumulator Register) . . . 88
3.6.47 SIV (Set Indexed Variable) . . . 89
3.6.48 GIV (Get Indexed Variable) . . . 90
3.6.49 AIV (Accumulator to Indexed Variable) . . . 91
3.6.50 Customer Specific Command Extensions (UF0...UF7 – User Functions) . . . 92
3.6.51 TMCL Control Commands . . . 93

4 Axis Parameters 95

5 Global Parameters 112
5.1 Bank 0 . . . 112
5.2 Bank 2 . . . 114

6 Motor Regulation 115
6.1 Structure of Motor Control Regulation Modes . . . 115
6.2 Current Regulation . . . 115
6.2.1 Structure of the Current Regulator . . . 116
6.3 Velocity Regulation . . . 117
6.3.1 Structure of the Velocity Regulator . . . 117
6.4 Velocity Ramp Generator . . . 118
6.5 Position Regulation . . . 118
6.5.1 Structure of the Position Regulator . . . 118
6.5.2 Correlation of Axis Parameters 105 and 109, the Target Position and the Position End Flag . . . 119

7 Ramps 120
7.1 Linear Ramp . . . 120
7.2 Sine Shaped Ramp . . . 120

8 Module Specific Functions 122
8.1 Filters . . . 122
8.1.1 Biquad Filter (for Future Use) . . . 123
8.2 Mechanical brake . . . 123
8.3 Brake Chopper . . . 124
8.3.1 PWM Braking . . . 124
8.3.2 Resistive/Shunt Braking . . . 124
8.4 IIT . . . 125
8.5 Lower Velocity PI . . . 126

9 TMCL Programming Techniques and Structure 127
9.1 Initialization . . . 127
9.2 Main Loop . . . 127
9.3 Using Symbolic Constants . . . 127
9.4 Using Variables . . . 128
9.5 Using Subroutines . . . 129
9.6 Combining Direct Mode and Standalone Mode . . . 129
9.7 Make the TMCL Program Start Automatically . . . 130

10 Figures Index 131

11 Tables Index 132

12 Supplemental Directives 133
12.1 Producer Information . . . 133
12.2 Copyright . . . 133
12.3 Trademark Designations and Symbols . . . 133
12.4 Target User . . . 133
12.5 Disclaimer: Life Support Systems . . . 133
12.6 Disclaimer: Intended Use . . . 133
12.7 Collateral Documents & Tools . . . 134

13 Revision History 135
13.1 Firmware Revision . . . 135
13.2 Document Revision . . . 135

1 Features

The TMCM-1690 is a controller/driver module for BLDC motors. It is highly integrated, offers a convenient handling, and can be used in many decentralized applications. The module is designed for phase currents up to 1.5A gate drive current and 60V DC (48V nominal) supply voltage. All motors are controlled using field oriented control (FOC), using either encoder (ABN or absolute) feedback or hall sensor feedback. The TMCL firmware allows for both standalone and direct mode operation.

Main characteristics

- Motion and motor controller for single and 3-phase motors:
  - Field oriented control
  - On-the-fly modification of motion parameters (example: position, velocity, acceleration).
  - High performance microcontroller for overall system control and communication protocol handling.
  - High-efficient operation, low power dissipation.
  - Supports incremental encoders, absolute encoders, and digital hall sensors.
- Interfaces
  - UART (RS485-ready or RS232 with external transceiver)
  - CAN interface with external transceiver.
  - Two digital inputs, one analog input, and six additional digital inputs/outputs.

Software

TMCL remote controlled operation through UART (RS485-ready/RS232) or CAN interface and/or standalone operation through TMCL programming. PC-based application development software TMCL-IDE is available for free.

Electrical data

- Supply voltage: 10V to 60V (48V nominal).
- Gate drive current: up to 1.5A (programmable).

Also see the separate hardware manual.

2 First Steps with TMCL

This chapter provides some hints for the first steps with the TMCM-1690 and TMCL. Skip this chapter if already familiar with TMCL and the TMCL-IDE.

Requirements

- A TMCM-1690 module
- A CAN interface supported by the TMCL-IDE or a UART (RS232/RS485-ready with transceiver) interface connected to the PC.
- A power supply (24V DC) for the TMCM-1690 module.
- The TMCL-IDE (3.x or higher) already installed on the PC.
- A BLDC 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-1690 is properly connected to the power supply and the BLDC motor is properly connected to the module. See the TMCM-1690 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-1690 through CAN or UART (RS232-/RS485-ready) interface and switch on the power supply for the module (while the TMCL-IDE is running on the PC). When the module is connected properly, it is recognized by the TMCL-IDE and is ready for use.

2.2 Using the TMCL Direct Mode

The TMCL-IDE displays a tree view showing the TMCM-1690 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 500 rpm 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.

Most Important Axis Parameters				
Number	Axis Parameter	Description	Range [Units]	Access
0	Motor type	Select the motor type. 0 - three-phase BLDC 1 - single-phase DC	0 ... 1	RWEX
2	Motor pole pairs	Number of motor pole pairs.	1 ... 255	RWEX
13	Commutation mode	Select a commutation mode that fits best to the motor's sensors. 0 - disabled 1 - open loop 2 - digital hall (foc) 3 - digital hall (block) 4 - ABN encoder 5 - ABS encoder	0 ... 5	RWEX
16	Maximum current	Maximum allowed absolute motor current. This value can be temporarily exceeded marginally due to the operation of the current regulator.	0 ... 10000 [mA]	RWEX
32	Open loop current	Motor current for controlled commutation. This parameter is used in commutation mode 1.	0 ... 10000 [mA]	RWEX
94	Maximum velocity	Maximum absolute velocity for velocity and positioning mode.	0 ... 200000 [rpm]	RWEX
95	Acceleration	Acceleration parameter for ROL, ROR, and the velocity ramp of MVP.	0 ... 200000 [rpm/s]	RWEX

Table 1: Most 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 uisng 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:

```
SAP 24, 0, 1868    //set ADC offsets
SAP 25, 0, 1866
SAP 26, 0, 1866

SAP 107, 0, 1000    //set position reached distance
SAP 108, 0, 500     //set position reached velocity [rpm]
SAP 110, 0, 500     //set position P
SAP 95, 0, 10000    //set acceleration [rpm/s]

SAP 13, 0, 5       //switch to abs encoder mode

ROR 0, 200         //rotate motor
WAIT TICKS, 0, 300
```

```
15 PositionTest:
       MVP ABS, 0, 0
17     WAIT POS, 0, 0
       MVP ABS, 0, 655360
19     WAIT POS, 0, 0
       JA PositionTest
```

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 donwload 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-1690 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-1690 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-1690. 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 is an example on how to do this:

Checksum calculation in C:

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

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

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

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

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)
STAP	7	<parameter>, <motor number>, <value>	Store axis parameter (store motion control specific settings)
RSAP	8	<parameter>, <motor number>	Restore axis parameter (restore 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)
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

Command	Number	Parameter	Description
DI	26	<interrupt number>	Disable interrupt
WAIT	27	<condition>, <motor number>, <ticks>	Wait with further program execution
STOP	28		Stop program execution
SCO	30	<coordinate number>, <motor number>, <position>	Set coordinate
GCO	31	<coordinate number>, <motor number>	Get coordinate
CCO	32	<coordinate number>, <motor number>	Capture coordinate
CALCX	33	<operation>	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

Command	Number	Parameter	Description
RORA	51	<motor number>	Rotate right, velocity specified by accumulator
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
STAP	7	Store axis parameter
RSAP	8	Restore 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 use to define and handle interrupts:

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

Table 11: Interrupt Processing Commands

3.5.6.1 Interrupt Types

There are many different interrupts in TMCL, like timer interrupts, stop switch interrupts, position reached interrupts, and input pin change interrupts. Each of these interrupts has its own interrupt vector. Each interrupt vector is identified by its interrupt number. Use the TMCL include file Interrupts.inc to have symbolic constants for the interrupt numbers. Table 12 shows all interrupts available on the TMCM-1690.

Interrupt Vectors	
Interrupt number	Interrupt type
0	Timer 0
1	Timer 1
2	Timer 2
3	Target position reached 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 3.6.9) 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:

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

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

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

In the example above, the interrupt numbers are being used directly. To make the program better readable use the provided include file Interrupts.inc. This file defines symbolic constants for all interrupt numbers that can be used in all interrupt commands. The beginning of the program above then looks as follows:

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

3.5.7 New TMCL Commands

To make several operations easier, the following new commands have been introduced from firmware version 1.07 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-1690 modules with firmware versions before 1.07. So, make sure that at least firmware version 1.07 is installed before using them.

New TMCL Commands		
Mnemonic	Command Number	Meaning
CALCVV	40	Calculate using two user variables
CALCVA	41	Calculate using a user variable and the accumulator
CALCAV	42	Calculate using the accumulator and a user variable
CALCVX	43	Calculate using a user variable and the X register
CALCXV	44	Calculate using the X register and a user variable
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 rounds per minute (rpm).

Internal function: Velocity mode is selected. Then, the velocity value is transferred to the target velocity.

Related commands: ROL, MST, SAP, GAP

Mnemonic: ROR <axis>, <velocity>

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

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

Example
Rotate right motor 0, velocity 500.
Mnemonic: ROR 0, 500.

Binary Form of ROR 0, 500	
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)	01_h
Value (Byte 0)	$F4_h$
Checksum	$F7_h$

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 rounds per minute (rpm).

Internal function: Velocity mode is selected. Then, the velocity value is transferred to the target velocity.

Related commands: ROR, MST, SAP, GAP

Mnemonic: ROL <axis>, <velocity>

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

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

Example
Rotate left motor 0, velocity 500.
Mnemonic: ROL 0, 500.

Binary Form of ROL 0, 500	
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)	01_h
Value (Byte 0)	$F4_h$
Checksum	$F8_h$

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

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

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 velocity for motor #0 to 200 rpm.
Mnemonic: SAP 94, 0, 200.

Binary Form of SAP 94, 0, 200	
Field	Value
Target address	01_h
Instruction number	05_h
Type	$5E_h$
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	00_h
Value (Byte 1)	00_h
Value (Byte 0)	$C0_h$
Checksum	$F9_h$

3.6.6 GAP (Get Axis Parameter)

Most motion/driver related parameters of the TMCM-1690 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 106, 0.

Binary Form of GAP 106, 0	
Field	Value
Target address	01_h
Instruction number	06_h
Type	$6A_h$
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	$3B_h$

3.6.7 STAP (Store Axis Parameter)

This command is used to store TMCL axis parameters permanently in the EEPROM of the module. This command is mainly needed to store the default configuration of the module. 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, refer to section 4.

Internal function: The axis parameter specified by the type and axis number is stored in the EEPROM.

Related commands: SAP, AAP, GAP, RSAP

Mnemonic: STAP <parameter number>, <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
7	See chapter 4	0	0 (Don't care)

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

Example
Store axis parameter #6 of motor #0.
Mnemonic: STAP 6, 0.

Binary Form of STAP 6, 0	
Field	Value
Target address	01_h
Instruction number	07_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)	00_h
Checksum	$0E_h$

3.6.8 RSAP (Restore Axis Parameter)

With this command, the contents of an axis parameter can be restored from the EEPROM. By default, all axis parameters are automatically restored after power up. An axis parameter 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, refer to section 4.

Internal function: The axis parameter specified by the type and axis number is restored from the EEPROM.

Related commands: SAP, AAP, GAP, RSAP

Mnemonic: RSAP <parameter number>, <axis>

Binary Representation			
Instruction	Type	Motor/Bank	Value
8	See chapter 4	0	0 (Don't care)

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

Example
Restore axis parameter #6 of motor #0.
Mnemonic: RSAP 6, 0.

Binary Form of RSAP 6, 0	
Field	Value
Target address	01_h
Instruction number	08_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)	00_h
Checksum	$0F_h$

3.6.9 SGP (Set Global Parameter)

With this command, most of the module specific parameters not directly related to motion control can be specified and the TMCL user variables can be changed. Global parameters are related to the host interface, peripherals, or application specific variables. The different groups of these parameters are organized in banks to allow a larger total number for future products. Currently, 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	$4F_h$

3.6.10 GGP (Get Global Parameter)

All global parameters can be read with this function. Global parameters are related to the host interface, peripherals, or application specific variables. The different groups of these parameters are organized in banks to allow a larger total number for future products. Currently, 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.

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	$4D_h$

3.6.11 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 5.2.

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 5.2	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

3.6.12 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 5.2.

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 5.2	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

3.6.13 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 2 (bank 2) to high.
Mnemonic: SIO 2, 2, 1.

Binary Form of SIO 2, 2, 1	
Field	Value
Target address	01_h
Instruction number	$0E_h$
Type	02_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
GPIO_2	SIO 2, 2, <value>	0/1
GPIO_3	SIO 3, 2, <value>	0/1
GPIO_4	SIO 4, 2, <value>	0/1
GPIO_5	SIO 5, 2, <value>	0/1
GPIO_6	SIO 6, 2, <value>	0/1
GPIO_7	SIO 7, 2, <value>	0/1

3.6.14 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
REF_R	GIO 0, 0	0/1
REF_L	GIO 1, 0	0/1

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
ACU_U	GIO 0, 1	0…4095
ADC_V	GIO 1, 1	0…4095
ADC_W	GIO 2, 1	0…4095
ADC_VOLTAGE	GIO 3, 1	0…4095
ADC_TEMP	GIO 4, 1	0…4095
ADC_AIN	GIO 5, 1	0…4095

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
GPIO_2	GIO 2, 2	0/1
GPIO_3	GIO 3, 2	0/1
GPIO_4	GIO 4, 2	0/1
GPIO_5	GIO 5, 2	0/1
GPIO_6	GIO 6, 2	0/1
GPIO_7	GIO 7, 2	0/1

3.6.15 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

3.6.16 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.

```
GAP 106, 0  //get actual position of motor 0
COMP 1000   //compare actual value with 1000
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

3.6.17 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 1 NZ - not zero 2 EQ - equal 3 NE - not equal 4 GT - greater 5 GE - greater/equal 6 LT - lower 7 LE - lower/equal 8 ETO - time out error 9 EAL - external alarm 10 EDV - deviation error 11 EPO - position error	0 (don't care)	<jump address>

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

```
GAP 106, 0  //get actual position of motor 0
COMP 1000   //compare actual value with 1000
JC GE, Label  //jump to Label if greater or equal to 1000
...
Label: ROL 0, 1000
```

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

3.6.18 JA (Jump Always)

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

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

Related commands: JC, WAIT, CSUB

Mnemonic: JA <label>

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

Example
An infinite loop in TMCL:

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

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

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

3.6.19 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:
    MVP ABS, 0, 10000
    CSUB SubW   //Save program counter and jump to label SubW
    MVP ABS, 0, 0
    CSUB SubW   //Save program counter and jump to label SubW
    JA Loop

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

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

3.6.20 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 3.6.19).
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

3.6.21 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.

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

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

[1] one tick is 10 milliseconds

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	$1D_h$

3.6.22 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	$1D_h$

3.6.23 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.

3.6.24 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.

3.6.25 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

3.6.26 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

3.6.27 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

3.6.28 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:

```
Start:
    GIO 0,1       //get value of analog input line 0
    CALC MUL, 4   //multiply by 4
    AAP 105,0      //transfer result to target position of motor 0
    JA Start      //jump back to start
```

Binary Form of AAP 105, 0	
Field	Value
Target address	01_h
Instruction number	22_h
Type	69_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	55_h

3.6.29 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

3.6.30 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

3.6.31 EI (Enable Interrupt)

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

Info See table 12 for a list of interrupts that can be used on the TMCM-1690 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

3.6.32 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-1690 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	$1A_h$

3.6.33 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-1690 module.

Related commands: EI, DI, RETI

Mnemonic: VECT <interrupt number>, <label>

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

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

Example
Define interrupt vector for timer #0 interrupt:

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

Binary Form of VECT (Assuming Label is at 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

3.6.34 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

3.6.35 CALCVV (Calculate Using Two User Variables)

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

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

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

Binary representation

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

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

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

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

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

3.6.36 CALCVA (Calculate Using a User Variable and the Accumulator Register)

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

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

Mnemonic: CALCVA <operation>, <var>

Binary representation

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

Reply in Direct Mode	
Status	Value
100 - OK	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

3.6.37 CALCAV (Calculate Using the Accumulator Register and a User Variable)

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

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

Mnemonic: CALCAV <operation>, <var>

Binary representation

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

Reply in Direct Mode	
Status	Value
100 - OK	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

3.6.38 CALCVX (Calculate Using a User Variable and the X Register)

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

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

Mnemonic: CALCVX <operation>, <var>

Binary representation

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

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

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

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

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

3.6.39 CALCXV (Calculate Using the X Register and a User Variable)

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

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

Mnemonic: CALCXV <operation>, <var>

Binary representation

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

Reply in Direct Mode	
Status	Value
100 - OK	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

3.6.40 CALCV (Calculate Using a User Variable and a Direct Value)

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

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

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

Binary representation

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

Reply in Direct Mode	
Status	Value
100 - OK	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	$2F_h$

3.6.41 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 rotuines. *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:

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

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

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

3.6.42 DJNZ (Decrement and Jump if not Zero)

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

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

Related commands: JC, WAIT, CSUB

Mnemonic: DJNZ <var>, <label>

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

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

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

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

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

3.6.43 CALL (Conditional Subroutine Call)

The CALL command calls a subroutine in the TMCL program, but only if the specifed condition is met. Otherwise, the program execution 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 1 NZ - not zero 2 EQ - equal 3 NE - not equal 4 GT - greater 5 GE - greater/equal 6 LT - lower 7 LE - lower/equal 8 ETO - time out error 9 EAL - external alarm 10 EDV - deviation error 11 EPO - position error	0 (don't care)	<jump address>

Example
Call a subroutine if a condition is met:

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

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

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

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

3.6.44 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).

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	$2F_h$

3.6.45 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 rounds per minute (rpm).

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

3.6.46 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 rounds per minute (rpm).

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

3.6.47 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	$3B_h$

3.6.48 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

3.6.49 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	$3A_h$

3.6.50 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

3.6.51 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 9.6). The following table lists all TMCL control commands.

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

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

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

Table 14: TMCL Control Commands

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-1690 module are controlled by axis parameters. Axis parameters can be modified or read using SAP, GAP, and AAP commands. Some axis parameters can also be stored to or restored from the EEPROM using STAP and RSAP commands. This chapter describes all axis parameters that can be used on the TMCM-1690 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
E	STAP, RSAP	Parameter can be stored in the EEPROM

Table 15: Meaning of the Letters in the Access Column

Axis 0 Parameters of the TMCM-1690 Module					
Number	Axis Parameter	Description	Range [Units]	Default	Access
0	Motor type	Select the motor type: 0 - three phase BLDC 1 - single phase DC	0 ... 1	0	RWEX
1	Motor family	Select the motor family: 0 - Rotary 1 - Linear	0 ... 1	0	RWEX
2	Motor pole pairs	Number of motor poles pairs.	0 ... 255	4	RWEX
3	Motor pole pair distance	Distance between motor poles for linear motor.	1 ... 65535 [μm]	10	RWEX
4	Motor nominal current	Motor nominal current.	0 ... 10000 [mA]	3470	RWEX
5	Motor peak current	Motor peak current.	0 ... 10000 [mA]	10000	RWEX
6	Motor line to line resistance	Motor line to line resistance.	1 ... 65535 [m Ω]	720	RWEX
7	Motor line to line inductance	Motor line to line inductance.	1 ... 65535 [μH]	1200	RWEX
8	Motor torque constant	Motor torque constant.	1 ... 65535 [mNm/A]	30	RWEX
9	Motor inertia	Motor inertia.	1 ... 65535 [gcm^2]	480	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
11	Motor ramp type	Type of ramp for motor (linear and sine). 0 - linear 1 - sine	0 ... 1	0	RWE
12	Motor direction	Set motor direction. 0 - Rotate clockwise. 1 - Rotate counter-clockwise.	0 ... 1	0	RWEX
13	Commutation mode	Select a commutation mode that fits best to the motor's sensors. 0 - disabled 1 - open loop 2 - digital hall (foc) 3 - digital hall (block) 4 - abn encoder 5 - absolute encoder	0 ... 5	0	RWEX
14	Motor peripherals	Sensors, gearbox and brakes attached to the motor $0x01$ - Motor side encoder (ABN, ABS, ABN2) $0x02$ - Hall sensor $0x04$ - Gearbox $0x08$ - Load side encoder (ABS, ABN2) $0x10$ - Mechanical brake $0x20$ - Brake chopper	0 ... 63	34	RWEX
15	Current sensor selection	Select current measurement type to commutate the motor. 0 - bottom shunt	0 ... 0	0	R
16	Max current	Max. allowed absolute motor current. This value can be temporarily exceeded marginally due to the operation of the current regulator.	0 ... 10000 [mA]	4000	RWEX
17	PWM scheme	PWM scheme. 0 - center aligned 1 - down counting	0 ... 1	0	RWEX
18	PWM frequency	Frequency of the PWM used for commutation. 0 - 20kHz pwm frequeny 1 - 40kHz pwm frequeny 2 - 60kHz pwm frequeny 3 - 80kHz pwm frequeny 4 - 100kHz pwm frequeny 5 - 120kHz pwm frequeny	0 ... 5	0	RWEX
19	PWM dead time	PWM dead (break before make) time.	0 ... 100 [ms]	1	RWEX
21	ADC_u_raw	Raw ADC measurement of the phase_A shunt	0 ... 4095	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
22	ADC_v_raw	Raw ADC measurement of the phase_B shunt	0 ... 4095	0	R
23	ADC_w_raw	Raw ADC measurement of the phase_C shunt	0 ... 4095	0	R
24	ADC_u_offset	Manually set/get the bottom-shunt phase_A offset.	0 ... 4095	2047	RWEX
25	ADC_v_offset	Manually set/get the bottom-shunt phase_B offset.	0 ... 4095	2047	RWEX
26	ADC_w_offset	Manually set/get the bottom-shunt phase_C offset.	0 ... 4095	2047	RWEX
27	ADC_u	Calculated current measurement for phase_A shunt and used offset	−32768 ... 32767	0	R
28	ADC_v	Calculated current measurement for phase_B shunt and used offset	−32768 ... 32767	0	R
29	ADC_w	Calculated current of phase_C from phase_A and phase_B measurements	−32768 ... 32767	0	R
31	Open loop commutation angle	Actual controlled angle value.	−32768 ... 32767	0	R
32	Open loop current	Motor current for controlled commutation. This parameter is used in commutation mode 1.	0 ... 10000 [mA]	2000	RWEX
33	Open loop velocity	Actual open loop velocity value.	−2147483648 ... 2147483647	0	R
34	Open loop position	The open loop position counter.	−2147483648 ... 2147483647	0	RW
36	Digital hall commutation angle	Actual digital hall angle value.	−32768 ... 32767	0	R
37	Digital hall/open loop commutation angle diff	Actual digital hall and open loop angle difference.	−32768 ... 32767	0	R
38	Digital hall velocity	Actual digital hall velocity value.	−2147483648 ... 2147483647	0	R
39	Digital hall position	The actual digital hall position counter.	−2147483648 ... 2147483647	0	RW

Number	Axis Parameter	Description	Range [Units]	Default	Access
40	Digital hall sector offset	Hall sensor sector offset. 0 - 0° sector offset 1 - 60° sector offset 2 - 120° sector offset 3 - 180° sector offset 4 - 240° sector offset 5 - 300° sector offset	0 ... 5	0	RWEX
41	Digital hall direction	Hall sensor direction. 0 - standard 1 - inverted	0 ... 1	0	RWEX
42	Digital hall interpolation	Hall sensor interpolation. 0 - off 1 - on	0 ... 1	0	RWEX
43	Digital hall phi_e offset	Offset for electrical angle hall_phi_e of hall sensor.	−32768 ... 32767	0	RWEX
44	Digital hall inputs	Raw hall sensor inputs.	0 ... 7	0	R
45	Digital hall auto configuration trigger	Hall sensor auto configuration trigger. 0 - Standby 1 - Direction estimation 2 - Sector offset estimation	0 ... 2	0	RW
46	ABN encoder commutation angle	Actual ABN encoder angle value.	−32768 ... 32767	0	R
47	ABN encoder/open loop commutation angle difference	Actual ABN encoder and open loop angle difference.	−32768 ... 32767	0	R
48	ABN encoder velocity	Actual ABN encoder velocity value.	−2147483648 ... 2147483647	0	R
49	ABN encoder position	The actual ABN encoder position counter.	−2147483648 ... 2147483647	0	RW
50	ABN encoder steps	ABN encoder steps per full motor rotation.	0 ... 16777215	4096	RWEX
51	ABN encoder direction	Set the ABN encoder direction in a way that ROR increases position counter. 0 - standard 1 - inverted	0 ... 1	0	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
52	ABN encoder init mode	Select an ABN encoder init mode that fits best to the motor's sensors. 0 - estimate offset 1 - estimate offset (shake) 2 - use offset 3 - use hall	0 ... 3	0	RWEX
53	ABN encoder init state	Actual state of ABN encoder initialization. 0 - nothing to do 1 - start_init 2 - wait_init_time 3 - estimate_offset	0 ... 3	0	R
54	ABN encoder init delay	Duration for encodersine initialization sequence. This parameter should be set in a way that the motor has stopped mechanical oscillations after the specified time.	0 ... 10000 [ms]	1000	RWEX
55	ABN encoder init velocity	Init velocity for ABN encoder initialization with encoder N-channel.	−200000 ... 200000 [rpm]	200	RWEX
56	ABN encoder offset	This value represents the internal commutation offset. (0...max. encoder steps per rotation).	0 ... 65535	0	RWE
57	Clear on null	Clear the position counter on ABN encoder N channel. 0 - do not clear position counter at next N channel event 1 - set position counter to zero at next N channel event	0 ... 1	0	RW
58	Clear once	Clear the position counter on encoder N channel. 0 - clear position counter always at an N channel event 1 - set position counter to zero only once	0 ... 1	0	RW
59	ABN encoder inputs	Raw ABN encoder inputs. Enc_A = Bit_0, Enc_B = Bit_1, Enc_N = bit_2 Bit_x = 1 - Encoder channel signal is high Bit_x = 0 - Encoder channel signal is low	0 ... 7	0	R
60	ABN encoder value	Raw ABN encoder counter value.	0 ... 16777215	0	R
61	ABN encoder side	Select side of the gearbox the enocder is mounted on: 0 - disabled 1 - motor side	0 ... 1	0	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
62	ABN encoder 2 velocity	Actual ABN encoder 2 velocity value.	−2147483648 ...2147483647	0	R
63	ABN encoder 2 position	The actual ABN encoder 2 position counter.	−2147483648 ...2147483647	0	RW
64	ABN encoder 2 steps	ABN encoder 2 steps per full motor rotation.	0 ...16777215	4096	RWEX
65	ABN encoder 2 direction	Set the ABN encoder 2 direction in a way that ROR increases position counter. 0 - standard 1 - inverted	0 ...1	0	RWEX
66	ABN encoder 2 inputs	Raw abn encoder 2 inputs. Enc_A = Bit_0, Enc_B = Bit_1, Enc_N = bit_2 Bit_x = 1 - Encoder channel signal is high Bit_x = 0 - Encoder channel signal is low	0 ...7	0	R
67	ABN Encoder 2 Side	Select side of the gearbox the enocder is mounted on: 0 - disabled 1 - motor side 2 - load side	0 ...2	0	RWEX
68	Absolute encoder commutation angle	Actual absolute encoder angle value.	−32768 ...32767	0	R
69	Absolute encoder/open loop commutation angle diff	Actual absolute encoder and open loop angle difference.	−32768 ...32767	0	R
70	Absolute encoder velocity	Actual absolute encoder velocity value.	−2147483648 ...2147483647	0	R
71	Absolute encoder position	The actual absolute encoder counter.	−2147483648 ...2147483647	0	RW
72	Absolute encoder type	Select the used absolute encoder: 0 - disabled 1 - AM4096 2 - ADA4573	0 ...2	0	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
73	Absolute encoder init	Select the used absolute encoder init mode: 0 - estimate offset 1 - estimate offset (shake) 2 - use offset	0 ... 2	0	RWEX
74	Absolute encoder direction	Set the absolute encoder direction in a way that ROR increases position counter. 0 - standard 1 - inverted	0 ... 1	0	RWEX
75	Absolute encoder offset	This value represents the internal commutation offset. (0...max. encoder steps per rotation).	0 ... 65535	0	RWE
76	Absolute encoder side	Select side of the gearbox the enocder is mounted on: 0 - disabled 1 - motor side 2 - load side	0 ... 2	0	RWEX
77	Target torque	Get desired target current or set target current to activate current regulation mode. (+= turn motor in right direction; -= turn motor in left direction)	−10000 ... 10000 [mA]	0	RW
78	Actual torque	The actual motor current.	−2147483648 ... 2147483647 [mA]	0	R
79	Target flux	Get desired target flux or set target flux to activate current regulation mode.	−10000 ... 10000 [mA]	0	RW
80	Actual flux	The actual motor flux.	−2147483648 ... 2147483647 [mA]	0	R
82	Torque offset	The desired torque offset.	−4700 ... 4700 [mA]	0	RW
83	Torque P	P parameter for current PID regulator.	0 ... 32767	300	RWEX
84	Torque I	I parameter for current PID regulator.	0 ... 32767	600	RWEX
85	Torque PI error sum	Sum of errors of current PI regulator.	−2147483648 ... 2147483647	0	R
86	Flux PI error sum	Sum of errors of flux PI regulator.	−2147483648 ... 2147483647	0	R
87	Torque PI error	Error of torque PI regulator.	−2147483648 ... 2147483647	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
88	Flux PI error	Error of flux PI regulator.	−2147483648 ...2147483647	0	R
90	Velocity sensor selection	Select a commutation mode that fits best to the motor's sensors. 0 - same as commutation 1 - digital hall 2 - abn encoder 3 - abn encoder 2 4 - abs encoder	0 ...4	0	RWEX
91	Velocity unit selection	Select mechanical or electrical velocity unit. 0 - mechanical rpm 1 - electrical rpm	0 ...1	0	RWEX
92	Target velocity	The desired target velocity.	−200000 ...200000 [rpm]	0	RW
93	Actual velocity	The actual velocity of the motor.	−2147483648 ...2147483647 [rpm]	0	R
94	Max velocity	Max. absolute velocity for velocity and positioning mode.	0 ...200000 [rpm]	4000	RWEX
95	Acceleration	Acceleration parameter for ROL, ROR, and the velocity ramp of MVP.	0 ...200000 [rpm/s]	2000	RWEX
96	Motor halted velocity	If the actual velocity is below this value, the motor halted flag is set.	0 ...200000 [rpm]	10	RWEX
98	Velocity offset	The desired velocity offset.	−200000 ...200000 [rpm]	0	RW
99	Velocity P	P parameter for velocity PID regulator.	0 ...32767	600	RWEX
100	Velocity I	I parameter for velocity PID regulator	0 ...32767	300	RWEX
101	Velocity PI error sum	Sum of errors of velocity PI regulator.	−2147483648 ...2147483647	0	R
102	Velocity PI error	Error of velocity PI regulator.	−2147483648 ...2147483647	0	R
104	Position sensor selection	Select a commutation mode that fits best to the motor's sensors. 0 - same as commutation 1 - digital hall 2 - abn encoder 3 - abn encoder 2 4 - abs encoder	0 ...4	0	RWEX
105	Target position	The target position of a currently executed ramp.	−2147483648 ...2147483647	0	RW
106	Actual position	The actual position counter.	−2147483648 ...2147483647	0	RW

Number	Axis Parameter	Description	Range [Units]	Default	Access
107	Position reached distance	Maximum distance at which the position end flag is set.	0 ... 100000	50	RWEX
108	Position reached velocity	Max. velocity at which end position flag can be set. Prevents issuing of end position flag when the target is passed at high velocity.	0 ... 200000 [rpm]	500	RWEX
109	Position reached flag	This flag is set when actual positon and velocity matches target position window. 0 - Position window not reached 1 - Position window reached	0 ... 1	0	R
110	Position P	P parameter for position PID regulator.	0 ... 32767	20	RWEX
111	Position PI error	Error of position PI regulator.	−2147483648 ... 2147483647	0	R
113	Gearbox transmission type	Select the transmission type of the Gearbox. 0 - Rotary to rotary 1 - Rotary to linear	0 ... 1	0	RWEX
114	Gearbox input displacement	The input displacement to the gearbox.	1 ... 2147483647	1	RWEX
115	Gearbox output displacement	The output displacement to the gearbox.	1 ... 2147483647	1	RWEX
116	Gearbox invert direction	Gearbox inverts the direction of motion: 0 - Direction not inverted 1 - Direction inverted	0 ... 1	0	RWEX
118	Thermal winding time constant (1)	Thermal winding time constant for the used motor. Used for IIt monitoring.	1000 ... 60000 [ms]	30000	RWEX
119	IIt limit (1)	An actual IIt sum that exceeds this limit leads to trigger the IIt_exceed_flag_1.	0 ... 54000000	10300000	RWEX
120	IIt sum (1)	Actual sum of the IIt monitor_1.	0 ... 4294967295	0	R
121	Thermal winding time constant (2)	Thermal winding time constant for the used motor. Used for IIt monitoring.	1000 ... 60000 [ms]	30000	RWEX
122	IIt limit (2)	An actual IIt sum that exceeds this limit leads to trigger the IIt_exceed_flag_1.	0 ... 54000000	10300000	RWEX
123	IIt sum (2)	Actual sum of the IIt monitor_2.	0 ... 4294967295	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
124	Clear IIt exceeded flags	Clear the flags that indicates that the IIt sum has exceeded the IIt limit.	0 ... 0	0	RW
126	Velocity window	A velocity window error occurs, if the difference between ramp velocity and actual velocity is bigger than this window.	0 ... 65535	500	RWEX
127	Clear velocity window error	Clear the flag that indicates that the velocity window has been left.	0 ... 0	0	RW
128	Position window	A position window error occurs, if the difference between ramp position and actual velocity is bigger than this window.	0 ... 2147483647	1638400	RWEX
129	Clear position window error	Clear the flag that indicates that the position window has been left.	0 ... 0	0	RW
131	Ramp acceleration	The actual acceleration used for measuring velocity and position.	−200000 ... 200000 [rpm/s]	0	R
132	Ramp velocity	The actual velocity of the velocity ramp used for positioning and velocity mode.	−200000 ... 200000 [rpm]	0	R
133	Enable velocity ramp	An activated ramp allows a defined acceleration for velocity and position mode. 0 - Deactivate velocity ramp generator. 1 - Activate velocity ramp generator.	0 ... 1	1	RWEX
134	Enable velocity feed forward	An activated velocity feed forward allows to use the velocity computed by the position ramp generator to be used as feed forward value for the velocity controller. 0 - Deactivate velocity feed forward. 1 - Activate velocity feed forward.	0 ... 1	1	RWEX
135	Ramp position	The actual position of the position ramp used for positioning mode.	−2147483648 ... 2147483647	0	R
137	Homing mode	0 - HOMING_OFF 1 - Single ended HARD_STOP_CW 2 - Single ended HARD_STOP_CCW 3 - Double ended HARD_STOP_CW 4 - Double ended HARD_STOP_CCW	0 ... 4	0	RWX

Number	Axis Parameter	Description	Range [Units]	Default	Access
138	Homing state	0 - NOT_HOMED 1 - HOMED 2 - START 3 - MOVE_TO_END_POSITION 4 - CHECK_NEGATIVE_STOP 5 - CHECK_POSITIVE_STOP 252 - STOP_ERROR 253 - STOP_TORQUE 254 - STOP_VELOCITY 255 - STOP_POSITION	0 ... 255	0	RW
139	Homing fast velocity	The velocity is used to drive to the position limit at the end of the homing function.	0 ... 65535	1000	RWEX
140	Homing slow velocity	The velocity is used during homing to search the hard stops or N-channels.	0 ... 65535	500	RWEX
141	Homing current threshold	Current threshold value to detect the hard stops.	0 ... 10000 [mA]	2000	RWEX
142	Position offset CW	Absolute position offset between clockwise hardstop/N-channel and max position limit.	0 ... 2147483647	0	RWEX
143	Position offset CCW	Absolute position offset between counterclockwise hardstop/N-channel and min position limit.	0 ... 2147483647	0	RWEX
144	Min position limit	Min position limit for positioning tasks.	−2147483648 ... 2147483647	−2147483648	RWEX
145	Max position limit	Max position limit for positioning tasks.	−2147483648 ... 2147483647	2147483647	RWEX
146	teach position limit	Teach position limit 0. Do not teach position 1. Teach left position 2. Teach Right position 3. Rescale to 16 bit	0 ... 3	0	RW
148	Target torque filter type	Target-torque filter type. 0 - disabled 1 - average 2 - biquad	0 ... 2	1	RWEX
149	Target torque simple filter size	Target-torque simple-filter size, value entered is the order of the filter n, where sample size = 2^n	1 ... 8	1	RWEX
150	Target torque biquad filter aCoeff_1	Target-torque biquad-filter aCoeff_1.	−2147483648 ... 2147483647	0	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
151	Target torque biquad filter aCoeff_2	Target-torque biquad-filter aCoeff_2.	−2147483648 ...2147483647	0	RWEX
152	Target torque biquad filter bCoeff_0	Target-torque biquad-filter bCoeff_0.	−2147483648 ...2147483647	0	RWEX
153	Target torque biquad filter bCoeff_1	Target-torque biquad-filter bCoeff_1.	−2147483648 ...2147483647	0	RWEX
154	Target torque biquad filter bCoeff_2	Target-torque biquad-filter bCoeff_2.	−2147483648 ...2147483647	0	RWEX
156	Actual current filter type	Actual-current filter type. 0 - disabled 1 - average 2 - biquad	0 ...2	1	RWEX
157	Actual current simple filter size	Actual-torque simple-filter size, value entered is the order of the filter n, where sample size = 2^n	1 ...8	1	RWEX
158	Actual current biquad filter aCoeff_1	Actual-current biquad-filter aCoeff_1.	−2147483648 ...2147483647	0	RWEX
159	Actual current biquad filter aCoeff_2	Actual-current biquad-filter aCoeff_2.	−2147483648 ...2147483647	0	RWEX
160	Actual current biquad filter bCoeff_0	Actual-current biquad-filter bCoeff_0.	−2147483648 ...2147483647	0	RWEX
161	Actual current biquad filter bCoeff_1	Actual-current biquad-filter bCoeff_1.	−2147483648 ...2147483647	0	RWEX
162	Actual current biquad filter bCoeff_2	Actual-current biquad-filter bCoeff_2.	−2147483648 ...2147483647	0	RWEX
164	Target velocity filter type	Target-velocity filter type. 0 - disabled 1 - average 2 - biquad	0 ...2	1	RWEX
165	Target velocity simple filter size	Target-velocity simple-filter size, value entered is the order of the filter n, where sample size = 2^n	1 ...8	1	RWEX
166	Target velocity biquad filter aCoeff_1	Target-velocity biquad-filter aCoeff_1.	−2147483648 ...2147483647	0	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
167	Target velocity biquad filter aCoeff_2	Target-velocity biquad-filter aCoeff_2.	−2147483648 ...2147483647	0	RWEX
168	Target velocity biquad filter bCoeff_0	Target-velocity biquad-filter bCoeff_0.	−2147483648 ...2147483647	0	RWEX
169	Target velocity biquad filter bCoeff_1	Target-velocity biquad-filter bCoeff_1.	−2147483648 ...2147483647	0	RWEX
170	Target velocity biquad filter bCoeff_2	Target-velocity biquad-filter bCoeff_2.	−2147483648 ...2147483647	0	RWEX
172	Actual velocity filter type	Actual-velocity filter type. 0 - disabled 1 - average 2 - biquad	0 ...2	1	RWEX
173	Actual velocity simple filter size	Actual-velocity simple-filter size, value entered is the order of the filter n, where sample size = 2^n	1 ...8	1	RWEX
174	Actual velocity biquad filter aCoeff_1	Actual-velocity biquad-filter aCoeff_1.	−2147483648 ...2147483647	0	RWEX
175	Actual velocity biquad filter aCoeff_2	Actual-velocity biquad-filter aCoeff_2.	−2147483648 ...2147483647	0	RWEX
176	Actual velocity biquad filter bCoeff_0	Actual-velocity biquad-filter bCoeff_0.	−2147483648 ...2147483647	0	RWEX
177	Actual velocity biquad filter bCoeff_1	Actual-velocity biquad-filter bCoeff_1.	−2147483648 ...2147483647	0	RWEX
178	Actual velocity biquad filter bCoeff_2	Actual-velocity biquad-filter bCoeff_2.	−2147483648 ...2147483647	0	RWEX
180	Target position filter type	Target position filter type. 0 - disabled 1 - average 2 - biquad	0 ...2	1	RWEX
181	Target position simple filter size	Target-position simple-filter size, value entered is the order of the filter n, where sample size = 2^n	1 ...8	1	RWEX
182	Target position biquad filter aCoeff_1	Target-position biquad-filter aCoeff_1.	−2147483648 ...2147483647	0	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
183	Target position biquad filter aCoeff_2	Target-position biquad-filter aCoeff_2.	−2147483648 ...2147483647	0	RWEX
184	Target position biquad filter bCoeff_0	Target-position biquad-filter bCoeff_0.	−2147483648 ...2147483647	0	RWEX
185	Target position biquad filter bCoeff_1	Target-position biquad-filter bCoeff_1.	−2147483648 ...2147483647	0	RWEX
186	Target position biquad filter bCoeff_2	Target-position biquad-filter bCoeff_2.	−2147483648 ...2147483647	0	RWEX
188	Release brake	Controls the external brake of the module. 0 - brake PWM deactivated. 1 - brake PWM activated.	0 ...1	0	RW
189	Brake releasing duty cycle	Controls the duty cycle of the first PWM phase for releasing the brake.	0 ...100 [%]	75	RWEX
190	Brake holding duty cycle	Controls the duty cycle of the second PWM phase to hold the brake.	0 ...100 [%]	11	RWEX
191	Brake releasing duration	Controls the duration the brake PWM uses the first duty cycle.	0 ...65535 [ms]	1000	RWEX
192	Enable brake output	Enables the brake functionality. 0 - brake functionality enabled 1 - brake functionality disabled	0 ...1	0	RWEX
193	Invert brake output	Inverts the brake output. 0 - brake output inverted 1 - brake output normal	0 ...1	0	RWEX
194	Brake supply voltage	Brake supply voltage.	0 ...65535 [0.1V]	240	RWEX
195	Brake resistance	Brake resistance.	0 ...65535 [m Ω]	440	RWEX
197	Enable brake chopper	Enable brake chopper functionality. 0 - deactivate brake chopper. 1 - activate brake chopper.	0 ...1	0	RWE
198	Brake chopper voltage limit	If the brake chopper is enabled and supply voltage exceeds this value, the brake chopper output is activated.	50 ...1000 [0.1V]	300	RWE
199	Brake chopper hysteresis	An activated brake chopper is disabled if the actual supply voltage is lower than (limit voltage-hysteresis).	0 ...50 [0.1V]	5	RWE

Number	Axis Parameter	Description	Range [Units]	Default	Access
200	Brake chopper type	Type of brake chopper. 0 - PWM braking 1 - shunt braking	0 ... 1 [0.1V]	0	RWEX
201	Brake chopper active	A value unequal to zero indicates an active brake chopper.	0 ... 1	0	R
202	Brake chopper supply voltage	Brake supply voltage.	0 ... 65535 [0.1V]	240	RWEX
203	Brake chopper resistance	Brake resistance.	0 ... 65535 [m Ω]	22000	RWEX
205	Reference switch enable	REF_L = Bit_1, REF_R = Bit_0 Bit_x = 1 - Reference switch functionality is enabled. Bit_x = 0 - Reference switch functionality is disabled.	0 ... 3	0	RWEX
206	Reference switch polarity	REF_L = Bit_1, REF_R = Bit_0 Bit_x = 1 - Reference switch is high active. Bit_x = 0 - Reference switch is low active.	0 ... 3	0	RWEX
207	Right reference switch active	0 - right reference switch deactivated. 1 - right reference switch activated.	0 ... 1	0	R
208	Left reference switch active	0 - left reference switch deactivated. 1 - left reference switch activated.	0 ... 1	0	R
210	Status flags	Actual status flags. 0x00000001 - OVERCURRENT 0x00000002 - UNDERVOLTAGE 0x00000004 - OVERVOLTAGE 0x00000008 - OVERTEMPERATURE 0x00000010 - MOTORHALTED 0x00000020 - HALLERROR 0x00000040 - DRIVER_ERROR 0x00000080 - INIT_ERROR 0x00000100 - STOP_MODE 0x00000200 - VELOCITY_MODE 0x00000400 - POSITION_MODE 0x00000800 - TORQUE_MODE 0x00001000 - VELOCITY_WINDOW 0x00002000 - POSITION_WINDOW 0x00004000 - POSITION_END 0x00008000 - MODULE_INITIALIZED 0x04000000 - MODULE_ENABLED 0x00020000 - IIT_EXCEEDED (1) 0x00040000 - IIT_EXCEEDED(2)	0 ... 4294967295	0	R
211	Supply voltage	The actual supply voltage.	0 ... 1000 [0.1V]	0	R
212	Supply voltage threshold	The supply voltage threshold.	0 ... 1000 [0.1V]	480	RWEX

Number	Axis Parameter	Description	Range [Units]	Default	Access
213	Driver temperature	The actual temperature of the motor driver.	−20 ... 150 [° C]	0	R
214	Driver temperature threshold	The temperature threshold at which the motor driver is shut down.	−20 ... 136 [° C]	0	RWEX
216	ABN encoder 2 angle	ABN encoder 2 angle value.	−32768 ... 32767	0	R
217	ABN encoder linear resolution	Linear resolution of the encoder per increment.	1 ... 65535 [nm]	500	RWE
220	Switch Over velocity for PI parameter	P parameter for lower velocity PID regulator.	0 ... 32767	0	RWEX
221	Velocity P for lower velocity	P parameter for lower velocity PID regulator.	0 ... 32767	0	RWEX
222	Velocity I for lower velocity	I parameter for lower velocity PID regulator.	0 ... 32767	0	RWEX
223	Linear maximum speed	Max. absolute velocity for velocity and positioning mode.	0 ... 2147483647 [μm/s]	2147483647	RW
224	Linear acceleration	Acceleration parameter for ROL, ROR, and the velocity ramp of MVP.	0 ... 2147483647 [μm/s^2]	2147483647	RWEX
225	Linear target velocity	Target velocity.	−2147483648 ... 2147483647 [μm/s]	0	RW
226	Linear actual velocity	Actual velocity.	−2147483648 ... 2147483647 [μm/s]	0	R
227	Linear target position	Target position.	−2147483648 ... 2147483647 [μm]	0	RW
228	Linear actual position	Actual multi-turn position for positioning.	−2147483648 ... 2147483647 [μm]	0	RW
229	Linear ramp velocity	The actual velocity of the velocity ramp used for positioning and velocity mode.	−2147483648 ... 2147483647 [μm/s]	0	R
230	Linear ramp position	The actual position of the position ramp used for positioning mode.	−2147483648 ... 2147483647 [μm]	0	R
238	Driver status register	Information regarding the current status of the driver.	0 ... 65535	0	R

Number	Axis Parameter	Description	Range [Units]	Default	Access
239	Clear driver error flag	Check/clear the driver error flag	0 ... 1	0	RW
240	Main loops	Main loops per second.	0 ... 4294967295 [1/s]	0	R
241	PWM loops	Torque loops per second.	0 ... 4294967295 [1/s]	0	R
242	Torque loops	Torque loops per second.	0 ... 4294967295 [1/s]	0	R
243	Velocity loops	Velocity loops per second.	0 ... 4294967295 [1/s]	0	R
244	debug value 0	Free used debugging value.	−2147483648 ... 2147483647	0	RW
245	debug value 1	Free used debugging value.	−2147483648 ... 2147483647	0	RW
246	debug value 2	Free used debugging value.	−2147483648 ... 2147483647	0	RW
247	debug value 3	Free used debugging value.	−2147483648 ... 2147483647	0	RW
248	debug value 4	Free used debugging value.	−2147483648 ... 2147483647	0	RW
249	debug value 5	Free used debugging value.	−2147483648 ... 2147483647	0	RW
250	debug value 6	Free used debugging value.	−2147483648 ... 2147483647	0	RW
251	debug value 7	Free used debugging value.	−2147483648 ... 2147483647	0	RW
252	debug value 8	Free used debugging value.	−2147483648 ... 2147483647	0	RW
253	debug value 9	Free used debugging value.	−2147483648 ... 2147483647	0	RW
254	Reinit bldc regulation	Reinit bldc regulation.	0 ... 1	0	W
255	Enable driver	Enables the motor driver (enabled by default): 0 - driver disabled 1 - driver enabled	0 ... 1	1	RW

Table 16: All TMCM-1690 Axis 0 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 2: TMCL user variables.

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-1690 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-1690 Module in Bank 0					
Number	Global Parameter	Description	Range [Units]	Default	Access
65	Serial baud rate	RS485/RS232 baud rate: 0 - 9600 [kBit/s] 1 - 14400 [kBit/s] 2 - 19200 [kBit/s] 3 - 28800 [kBit/s] 4 - 38400 [kBit/s] 5 - 57600 [kBit/s] 6 - 76800 [kBit/s] 7 - 115200 [kBit/s]	0 ... 7	7	RW
66	Serial address	The module (target) address for RS485, RS232, and virtual COM port.	1 ... 255	1	RWX
69	CAN bit rate	CAN bit rate: 2 - 20 [kBit/s] 3 - 50 [kBit/s] 4 - 100 [kBit/s] 5 - 125 [kBit/s] 6 - 250 [kBit/s] 7 - 500 [kBit/s] 8 - 1000 [kBit/s]	2 ... 8	8	RW
70	CAN send ID	The CAN send ID of the module.	0 ... 255	2	RWX
71	CAN receive ID	The CAN receive ID of the module.	0 ... 255	1	RWX
75	Telegram pause time	Pause time before the reply through RS485/RS232 is sent.	0 ... 255 [ms]	0	RWX
76	Serial host address	Host address used in the reply telegrams sent back via RS485/RS232.	1 ... 255	2	RWX
77	Auto start mode	Use automatic TMCL application start after power up. 0 - do not start TMCL application after power up 1 - start TMCL application automatically after power up	0 ... 1	1	RWX
78	IO direction mask	IO direction for selecting GPIOs as inputs or outputs	0 ... 511	0	RWX
79	IO output mask	Mask for digital IOs	0 ... 511	0	RWX
128	Application status	Actual TMCL application status. 0 - stop 1 - run 2 - step 3 - reset	0 ... 3	0	R
130	Program counter	TMCL program counter.	0 ... 4294967295	0	R

Number	Global Parameter	Description	Range [Units]	Default	Access
132	Tick timer	TMCL tick timer.	0 ...2147483647	0	RW

Table 18: All Global Parameters of the TMCM-1690 Module in Bank 0

5.2 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	RWE

Table 19: User Variables in Bank 2

6 Motor Regulation

6.1 Structure of Motor Control Regulation Modes

The TMCM-1690 supports a current, velocity, and position PI regulation mode for motor control in different application areas. Figure 1 shows these regulation modes. Individual modes are explained in the following subsections.

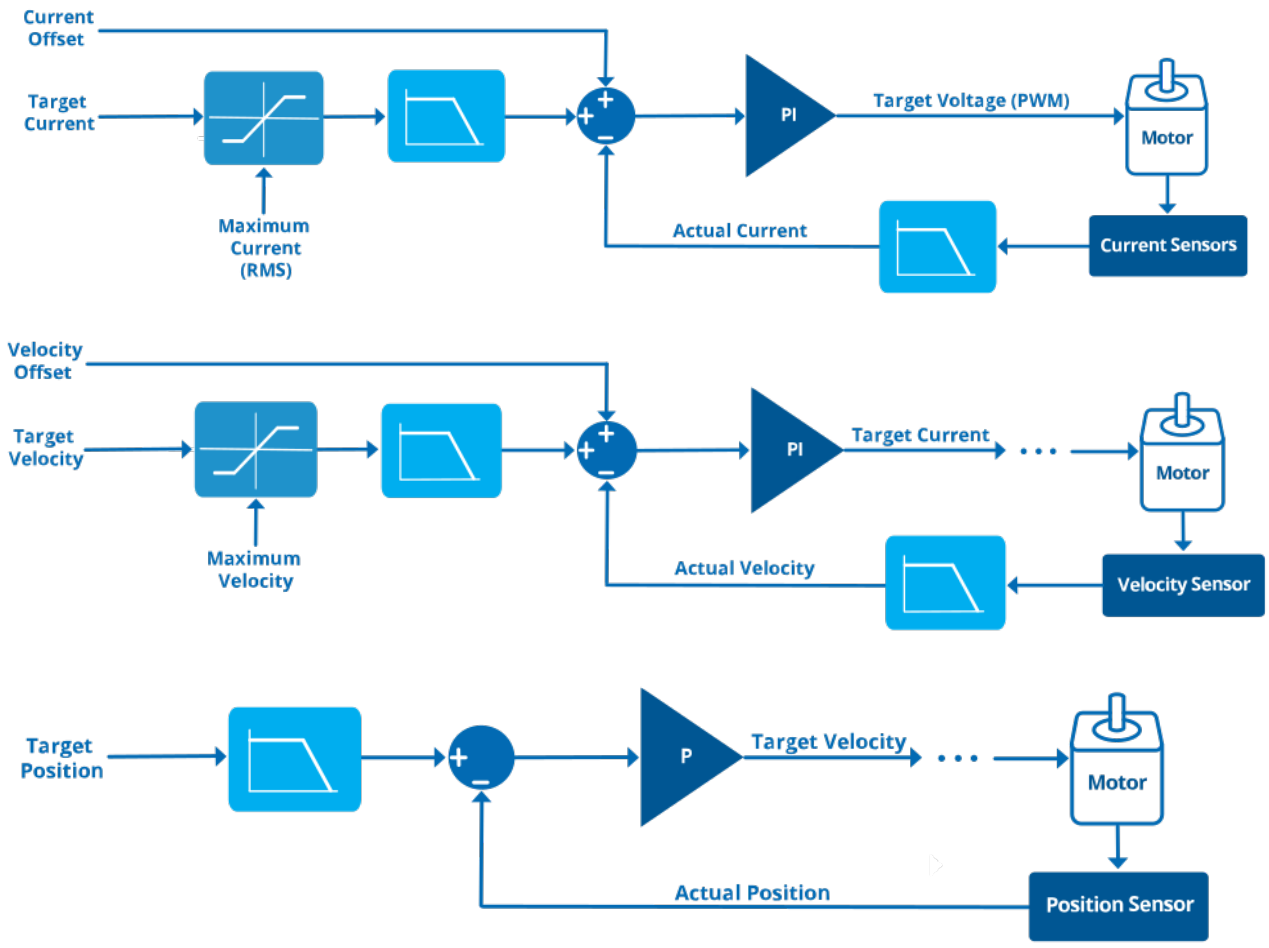


Figure 1: Motion Regulation

It can be seen that the target current signal is first limited by maximum current. There is an option for having a filter (average or biquad) to this signal. This is followed by having an impact of P and I parameters to the signal and the target voltage (PWM) is fed to the motor. There is a feedback coming from the actual current forming a closed loop. In the same fashion, the velocity and position regulation are performed.

6.2 Current Regulation

The current regulation mode uses a field-oriented current (FOC) and flux regulator to adjust a desired motor current. The current loop is running typically at 20KHz and the PWM loop is variable and can be set using AP 18. The PWM frequency ranges from 20KHz to 120KHz. The target current can be set by axis parameter 77. The maximal target current is limited by axis parameter 16. The current regulation uses

two basic parameters: the P and I parameters.

6.2.1 Structure of the Current Regulator

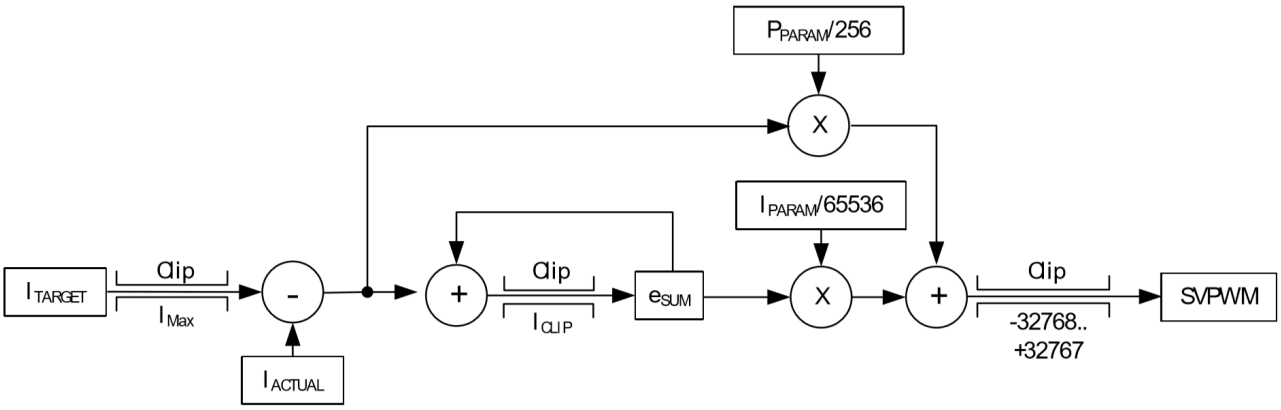


Figure 2: Current Regulation (See Parameter Descriptions in Table 20)

Current Regulation Parameters	
Parameter	Description
I_{ACTUAL}	Actual current (GAP 78)
I_{TARGET}	Target current (SAP 77)
I_{Max}	Max. target current (SAP 16)
e_{SUM}	Error sum for integral calculation
P_{PARAM}	Current P parameter (SAP 83)
I_{PARAM}	Current I parameter (SAP 84)

Table 20: Current Regulation Parameters

6.2.1.1 Parametrizing the Current Regulator Set

Parametrizing the current regulator set is preferably done with the PI tuning tool using auto tuning. Further details can be seen in the TMCL-IDE documentation under the section PI tuning. However, it can be done manually as well without the tool. To parameterize the current regulator set properly, do as follows:

1. Set the P parameter and the I parameter to zero.
2. Start the motor by using a low target flux (example, 1000mA).
3. Modify the current P parameter. Start from a low value and go to a higher value, until the actual current nearly reaches 70% of the desired target current.
4. Do the same with the current I parameter.

For all tests, set the motor current limitation to a realistic value, so that the power supply does not become overloaded during acceleration phases. If the power supply reaches current limitation, the unit may reset or undetermined regulation results may occur.

6.3 Velocity Regulation

Based on the current regulation, the motor velocity can be controlled by the velocity PI regulator. The velocity regulation runs in the velocity loop clocked at 2KHz.

6.3.1 Structure of the Velocity Regulator

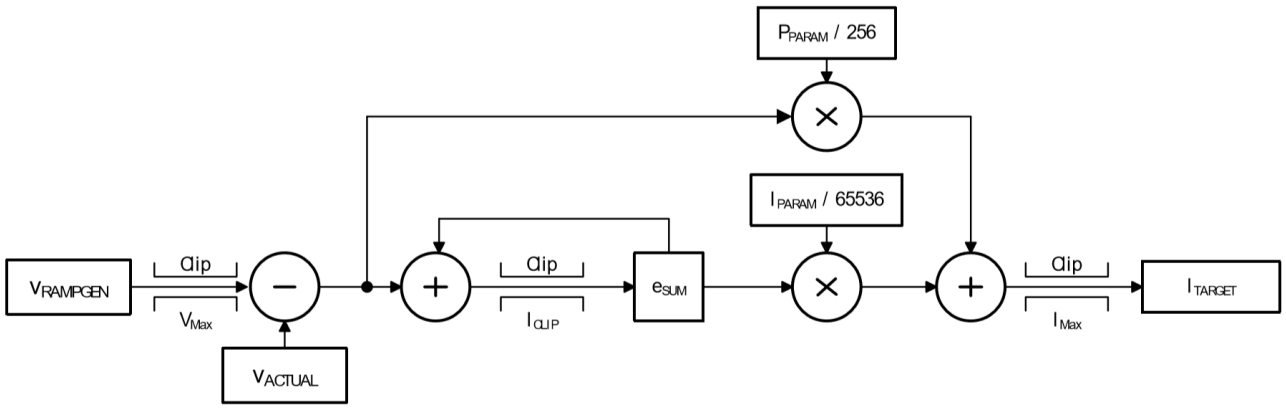


Figure 3: Velocity Regulation (See Parameter Descriptions in Table 21)

Velocity Regulation Parameters	
Parameter	Description
V_{ACTUAL}	Actual motor velocity (GAP 93)
$V_{RAMPGEN}$	Target velocity of ramp generator (SAP 92)
V_{Max}	Max. target velocity (SAP 94)
e_{SUM}	Error sum for integral calculation
P_{PARAM}	Velocity P parameter (SAP 99)
I_{PARAM}	Velocity I parameter (SAP 100)
I_{Max}	Max. target current (SAP 16)

Table 21: Velocity Regulation Parameters

6.3.1.1 Parametrizing the Velocity Regulator Set

Parametrizing the velocity regulator set is preferably done with the PI tuning tool using auto tuning. Further details can be seen in the TMCL-IDE documentation under the section PI tuning. However, it can be done manually as well without the tool. To parameterize the velocity regulator set properly, do as follows:

1. Set the velocity I parameter to zero.
2. Start the motor by using a medium target velocity (example, 2000 rpm).
3. Modify the current P parameter.
   (a) Start from a low value and go to a higher value, until the actual motor speed reaches 80% or 90% of the target velocity.
   (b) The lasting 10% or 20% speed difference can be reduced by slowly increasing the velocity I parameter.

6.4 Velocity Ramp Generator

For a controlled startup of the motor's velocity, a velocity ramp generator can be activated/deactivated by axis parameter 133. The ramp generator uses the maximal allowed motor velocity (axis parameter 94), the acceleration (axis parameter 95), and the desired target velocity (axis parameter 92) to calculate a ramp generator velocity for the following velocity PI regulator.

6.5 Position Regulation

Based on current and velocity regulators, the TMCM-1690 supports a positioning mode based on encoder (ABN or absolute) or hall sensor position. During positioning, the velocity ramp generator can be activated to enable motor positioning with controlled acceleration or it can be disabled to support motor positioning with maximum allowed speed.

The position regulation uses only one basic parameter: the P regulation parameter

6.5.1 Structure of the Position Regulator

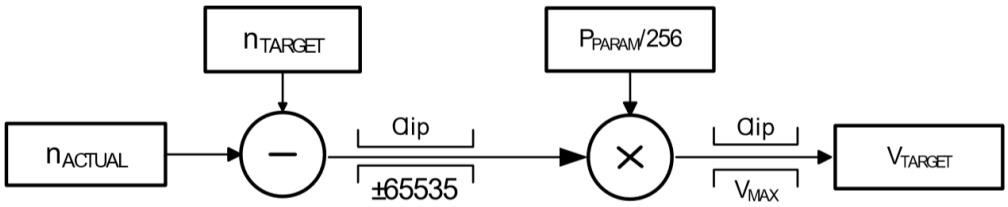


Figure 4: Positioning Regulation (See Parameter Descriptions in Table 22)

Position Regulation Parameters	
Parameter	Description
n_{ACTUAL}	Actual motor position (GAP 106)
n_{TARGET}	Target motor position (SAP 105)
P_{PARAM}	Position P parameter (SAP 110)
V_{MAX}	Max. allowed velocity (SAP 94)
V_{TARGET}	New target velocity for the ramp generator

Table 22: Position Regulation Parameters

6.5.1.1 Parametrizing the Position Regulation

Based on the velocity regulator, only the position regulator P has to be parameterized.
Parametrizing the position regulator is preferably done with the PI tuning tool using auto tuning. Further details can be seen in the TMCL-IDE documentation under the section PI tuning. However, it can be done manually as well without the tool. To parameterize the position regulator, do as follows:

1. Disable the velocity ramp generator and set position P parameter to zero.
2. Choose a target position and increase the position P parameter until the motor reaches the target position approximately.

3. Switch on the velocity ramp generator. Based on the maximum positioning velocity (axis parameter 94) and the acceleration value (axis parameter 95), the ramp generator automatically calculates the slow down point, that is, the point at which the velocity has to be reduced to stop at the desired target position.
4. Reaching the target position is signaled by setting the position end flag.

To minimize the time until this flag becomes set, the positioning tolerance MVP target reached distance can be chosen with axis parameter 107.

Since the motor typically is assumed not to signal target reached when the target is just passed in a short moment at a high velocity, additionally, the maximum target reached velocity (MVP target reached velocity) can be defined by axis parameter 108.

A value of zero for axis parameter 108 is the most universal, since it implies that the motor stands still at the target. But when a fast rising of the position end flag is desired, a higher value for the MVP target reached velocity parameter saves a lot of time. The best value should be tried out in the actual application.

6.5.2 Correlation of Axis Parameters 105 and 109, the Target Position and the Position End Flag

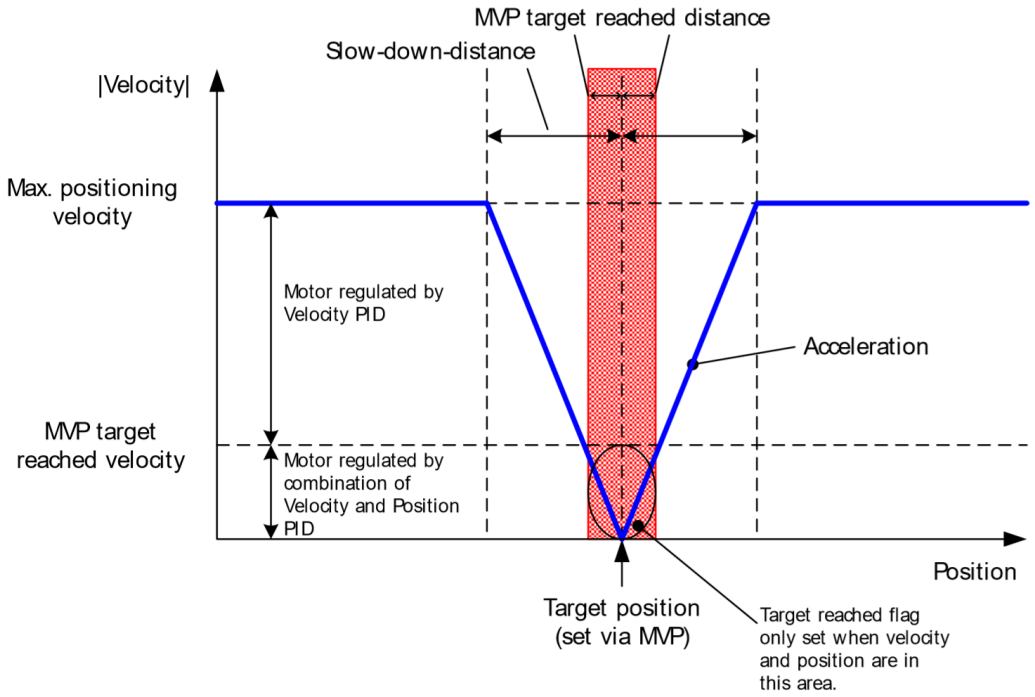


Figure 5: Positioning Algorithm

Depending on motor and mechanics, a low oscillation is normal. This can be reduced to at least +/-1 steps. Without oscillation the regulation cannot keep the position!

7 Ramps

TMCM-1690 supports two forms of ramps for motion control:

- Linear ramp
- Sine shaped ramp

7.1 Linear Ramp

A linear ramp or a trapezoidal ramp predicts the acceleration rates using a constant gradient. This constant gradient results in constant increase in the velocity. The acceleration and velocity are limited by the AP 95 (maximum acceleration) and 94 (maximum velocity) respectively. This ramp can be used to perform positioning tasks as well. There are some resonances in the system but for many systems they can be ignored. The linear ramp is selected by setting 0 to AP 11

7.2 Sine Shaped Ramp

In contrast to linear shaped ramps, the sine shape ramps follow a constant increase in the acceleration as it is one order higher. This results in a smoother response with far less resonances in the system. Sine shaped ramp is selected by setting 1 to AP 11. Figure 6 below gives an impression of how s shaped ramp looks like and how it is different than linear ramp. In the figure, desired position Pt is reached with the maximum acceleration and maximum velocity.

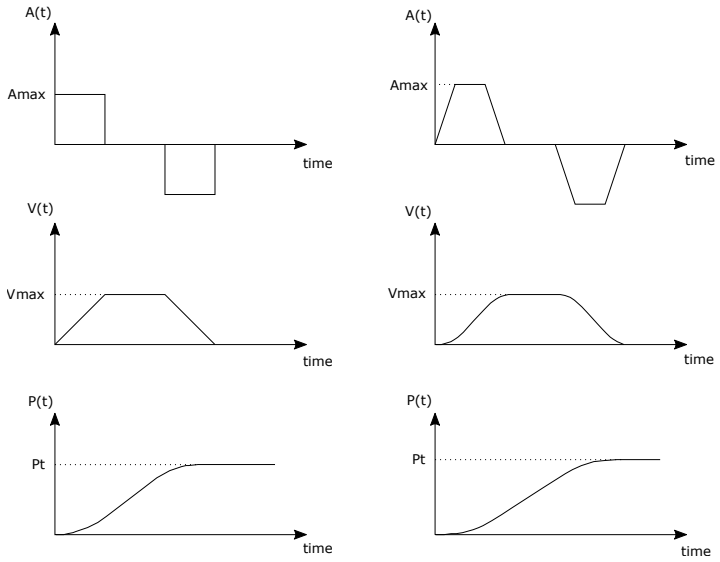


Figure 6: Linear Shaped Ramp (Left) S Shaped Ramp (Right)

It can be seen that s shaped ramps provide smoother operation as the acceleration is changed linearly compared to abrupt change of acceleration as in the linear ramp. This results from controling a finite amount of jerk at the time when the ramp is accelerating or deccelerating
After the ramp type is selected, the motor can be turned in both velocity and position mode. For turning the motor in velocity mode, the target velocity (AP 92) is set to the desired velocity. The ramp tries to reach its desired value with fastest acceleration limited by the parameter maximum acceleration (AP 95).

For turning the motor with position mode, the target position is set with the parameter AP 105. The ramp is calculated that it tries to reach its desired position with the fastest accelration (which is also limited by (AP 95)) and fastest velocity (limited by maximum velocity (AP 94)).
TMCM-1690 supports all three modes (torque, velocitye and position mode) with both linear and sine shaped ramps. The parameters such as maximum velocity, maximum acceleration, target velocity, and target position can be changed on the fly, and the ramp is adjusted accordingly.

8 Module Specific Functions

8.1 Filters

This module supports software filters for five signals, as shown in Figure 7. Each signal can be filtered using either a biquad filter or an averaging filter. Of course, there is the option to not filter at all. The averaging filter averages over the last number of samples, in exponents of 2, up to 2^8. The biquad filter, on the other hand, can be configured as a lowpass filter of the 1st and 2nd orders, and also as resonance-anitresonance filter. As the equation determining the biquad filter is the same, only a different set of coefficients is required to alter its behaviour. See section 8.1.1 for more details.

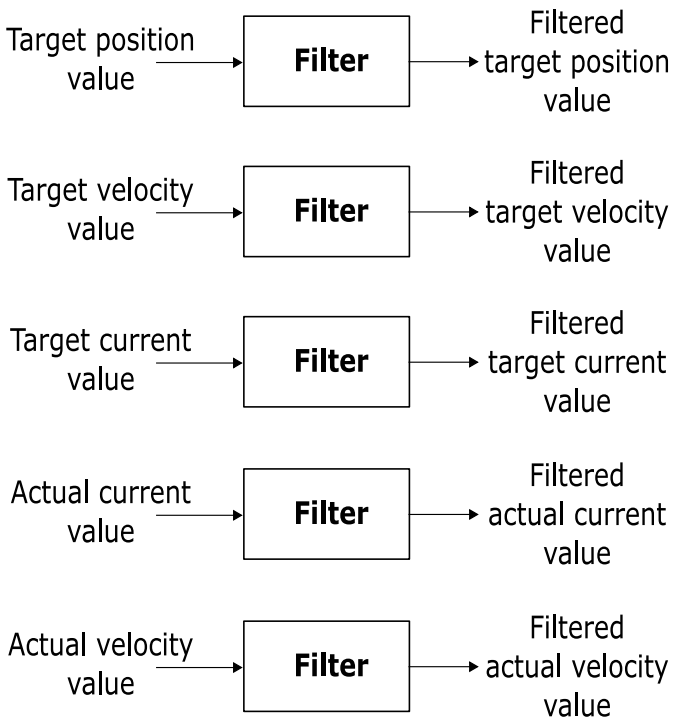


Figure 7: Filter Blocks

The filter for the target position value is intended to smoothen the position input to the control structure. It is evaluated at every tenth PWM cycle. The filter for the target velocity value is intended to smoothen the velocity input from an external master or the position controller. It is also evaluated at every tenth PWM cycle. The filter for the target torque value is intended to smoothen the torque input from an external master or the velocity controller. It is also evaluated at every tenth PWM cycle. The filter for the actual current value is intended to smoothen the measured actual current value. It is evaluated at every PWM cycle. The filter for the actual velocity value is intended to smoothen the measured actual velocity value. It is evaluated at every tenth PWM cycle. It can be used as a low-pass filter for bandwidth limitation and noise suppression. Moreover, it can be designed to suppress a resonance or anti-resonance. Same statements are correct for all the filters.
There are a total of seven axis-paramters for each signal. To set the type of filter for target torque, for example, AP 148 can be set to values 0, 1, or 2. Where 0 disables the filter, 1 enables the averaging filter, and 2 enables the biquad filter. AP 149 sets the sample size for the averaging filter. APs 150 to 154

correspond to the biquad filter coefficients in Q3.29 format. A similar layout of filter settings is followed by all five signals. See the axis-parameters table.

8.1.1 Biquad Filter (for Future Use)

The biquad filters implemented inside of TMCM-1690 use standard biquad filters (standard IIR filter of second order, Wikipedia Article) in the following structure.

$$\text{Y(n)} = \text{X(n)} \times \text{b\_0} + \text{X(n-1)} \times \text{b\_1} + \text{X(n-2)} \times \text{b\_2} + \text{Y(n-1)} \times \text{a\_1} + \text{Y(n-2)} \times \text{a\_2} \tag{1}$$

In this equation, X(n) is the actual input sample, while Y(n-1) is the filter output of the last cycle. All coefficients are S32 values and are normalized to a Q3.29 format. Take care of correct parametrization of the filter. There is no built-in plausibility or stability check. Biquad state variables are reset when parameters are changed. The TRINAMIC IDE supports parametrization with the *Control Settings* tool.
A standard biquad filter has the following transfer function in the Laplace-Domain:

$$\text{G(s)} = \frac{\text{b\_2\_cont} \times s^2 + \text{b\_1\_cont} \times \text{s} + \text{b\_0\_cont}}{\text{a\_2\_cont} \times s^2 + \text{a\_1\_cont} \times \text{s} + \text{a\_0\_cont}} \tag{2}$$

The transfer function needs to be transformed to time discrete domain by Z-Transformation and coefficients need to be normalized. This is done by the following equations.

$$b\_2\_z = (b\_0\_cont \times T^2 + 2 \times b\_1\_cont \times T + 4 \times b\_2\_cont)/(T^2 - 2 \times a\_1\_cont \times T + 4 \times a\_2\_cont) \tag{3}$$
$$b\_1\_z = (2 \times b\_0\_cont \times T^2 - 8 \times b\_2\_cont)/(T^2 - 2 \times a\_1\_cont \times T + 4 \times a\_2\_cont) \tag{4}$$
$$b\_0\_z = (b\_0\_cont \times T^2 - 2 \times b\_1\_cont \times T + 4 \times b\_2\_cont)/(T^2 - 2 \times a\_1\_cont \times T + 4 \times a\_2\_cont) \tag{5}$$
$$a\_2\_z = (T^2 + 2 \times a\_1\_cont \times T + 4 \times a\_2\_cont)/(T^2 - 2 \times a\_1\_cont \times T + 4 \times a\_2\_cont) \tag{6}$$
$$a\_1\_z = (2 \times T^2 - 8 \times a\_2\_cont)/(T^2 - 2 \times a\_1\_cont \times T + 4 \times a\_2\_cont) \tag{7}$$
$$b\_0 = round(b\_0\_z \times 2^{29}) \tag{8}$$
$$b\_1 = round(b\_1\_z \times 2^{29}) \tag{9}$$
$$b\_2 = round(b\_2\_z \times 2^{29}) \tag{10}$$
$$a\_1 = round(-a\_1\_z \times 2^{29}) \tag{11}$$
$$a\_2 = round(-a\_2\_z \times 2^{29}) \tag{12}$$

while T is the sampling time according to PWM_MAX_COUNT $\times$ 10 ns and variables with index z are auxiliary variables.
A standard second order lowpass filter with given cutoff frequency ω_c and damping factor D has the following transfer function in the Laplace-Domain:

$$G_{LP}(s) = \frac{1}{\frac{1}{\omega_c^2} \times s^2 + \frac{2\,D}{\omega_c} \times s + 1} \tag{13}$$

Determine filter coefficients with the equations above by comparing coefficients of both transfer functions.

8.2 Mechanical brake

The TMCM-1690 also supports a mechanical brake that can have a seperate power source. As the brake is operated by MOSFET on the baseboard, there is an upper limit to the current it can handle. Thus, provide the brake's electrical resistance and supply voltage to set the upper limit of PWM duty cycle.

For setup, first provide the supply voltage (AP 194) and brake's resistance (195). Then provide the duty cycles that the brake has on on release (AP 189) and as hold (AP 190), while also providing release duration (AP 191). Setting AP 192 enables the brake to be used. Now, the brake can be released by setting AP 188. This causes the brake line to output a PWM signal with release duty cycle for the release duration. After the release duration, the PWM signal goes to holding duty cycle. See Figure 8.

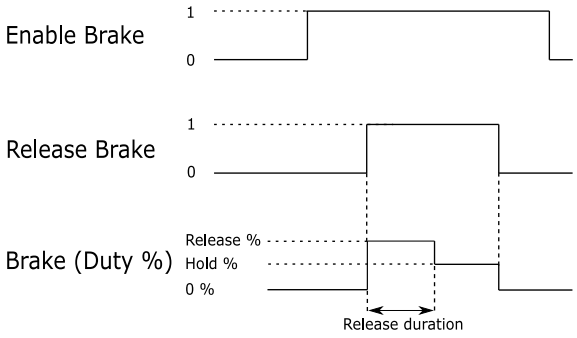


Figure 8: Mechanical Brake Settings

8.3 Brake Chopper

A servo system feeds back energy to the power supply line during deceleration and load control. The energy can lead to a voltage rise on the power supply system if it is not dissipated. The voltage overshoot of a system without brake chopper depends on the motor deceleration time, kinetic energy, and the servo module buffer capacity. The brake chopper dissipates this energy from the system, and thus avoids system damage. TMCM-1690 supports two kinds of brake chopper

- PWM braking
- Resistive/Shunt braking

8.3.1 PWM Braking

In PWM braking, the motor coils are used to perform the dissipation of the energy from the system. This is done by applying 50 % duty cycle on all three phases of the motor. For enabling PWM braking, AP 200 is set to 0. Additionally, the voltage limits (AP 198) is selected such that the supply voltage is larger than it. After that the brake is enabled by setting AP 197. This results in the extra regenerative energy produced due to slowing down dissipated through the coils of the motor.

8.3.2 Resistive/Shunt Braking

TMCM-1690 provides a continuous motor voltage monitoring as well as brake chopper output. The brake resistor is connected between the supply voltage and brake output. For setup, first provide the supply voltage (AP202) and brake's resistance (AP 203). Shunt braking is selected by setting AP 200 to 1. The voltage limits and hysteresis are selected using the AP 198 and 199, respectively. Lastly, the brake is enabled by setting AP 197. For a full speed ramp stop, the brake resistor should be able to dissipate the complete kinetic energy fed back during deceleration phase. The following figure shows an example of brake chopper schematic

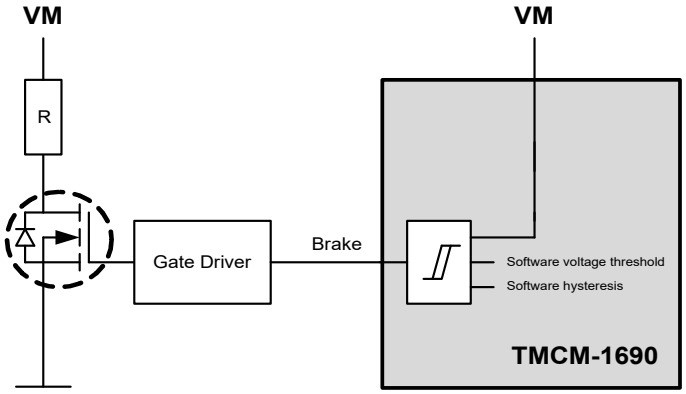


Figure 9: Brake Chopper Example Schematic

8.4 IIT

The IIT monitor provides a check on the energy consumption. The actual current is being monitored, and these values are being squared and summed up periodically over the configured winding time using a 1ms cycle. If one of the limits gets exceeded during this time, the motor is stopped and the IIT error flag is set. The IIT error flag can be reset by writing any value to axis-parameter 124. There are two IIT windows (see Figure 10). The first one directly uses the actual current, and the second one uses the actual current divided by $\sqrt{2}$ (less power over longer time). Axis-parameters 119 and 122 are limits for the two windows. Axis-parameters 120 and 123 show the actual integration sums.

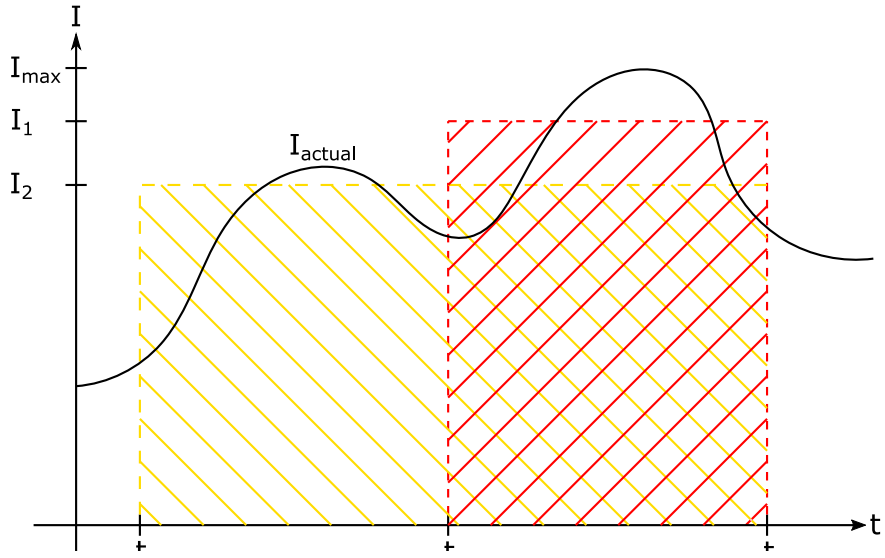


Figure 10: IIt Monitor Windows

8.5 Lower Velocity PI

The TMCM-1690 supports having different PI parameters for lower velocities. This provides a better control and smoother operation. It is possible to have a stronger set of PI parameters for the lower velocities and a relatively weaker set for the higher velocities. It is good to have the set of both PI parameters and the velocity to perform the switch (to get the parameters, PI tuning tool can be used). The velocity at which the switch is performed can be set using the axis parameter 220 (default value is 0). After that, P and I parameters can be set using the axis parameters 221 and 222, respectively. These are the PI parameters for the velocities less than the switch over velocity. Parameters for higher velocities are set using the axis parameters 99 and 100 for P and I, respectively.

9 TMCL Programming Techniques and Structure

9.1 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.

9.2 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:

```
  //Initialization
  SAP 94, 0, 5000 //define maximum positioning speed
  SAP 95, 0, 1000 //define maximum acceleration

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

9.3 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:

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

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

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

```
17   JA MainLoop
```

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.

9.4 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 varaible in the EEPROM.
- The RSGP command copies the value stored in the EEPROM back to the user variable.
- Global parameter 85 controls if user variables 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.

9.5 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.

9.6 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
Func1:  JA Func1Start
Func2:  JA Func2Start
Func3:  JA Func3Start

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

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

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

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

9.7 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"'.

10 Figures Index

1 Motion Regulation 115
2 Current Regulation 116
3 Velocity Regulation 117
4 Positioning Regulation 118
5 Positioning Algorithm 119
6 Linear Shaped Ramp (Left) S Shaped Ramp (Right) 120
7 Filter Blocks 122
8 Mechanical Brake Settings 124
9 Brake Chopper Example Schematic . 125
10 IIt Monitor Windows 125

11 Tables Index

1 Most important Axis Parameters . . . 7
2 TMCL Command Format 10
3 TMCL Reply Format 11
4 TMCL Status Codes 11
5 Overview of All TMCL Commands . . 15
6 Motion Commands 15
7 Parameter Commands 16
8 Branch Commands 16
9 I/O Port Commands 17
10 Calculation Commands 17
11 Interrupt Processing Commands . . . 18
12 Interrupt Vectors 18
13 New TMCL Commands 20
14 TMCL Control Commands 94
15 Meaning of the Letters in the Access Column 95
16 All TMCM-1690 Axis 0 Parameters . . 111
17 Meaning of the Letters in the Access Column 112
18 All Global Parameters of the TMCM-1690 Module in Bank 0 114
19 User Variables in Bank 2 114
20 Current Regulation Parameters 116
21 Velocity Regulation Parameters 117
22 Position Regulation Parameters . . . 118
23 Firmware Revision 135
24 Document Revision 135

12 Supplemental Directives

12.1 Producer Information

12.2 Copyright

12.3 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.

12.4 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.

12.5 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.

12.6 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.

12.7 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.

13 Revision History

13.1 Firmware Revision

Version	Date	Description
1.00	06/23	First release
1.01	07/23	New release: • Fixed on-the-fly change of motion parameters • Fixed issue with the block hall

Table 23: Firmware Revision

13.2 Document Revision

Version	Date	Description
0	01/24	First release

Table 24: Document Revision

Mouser Electronics

Authorized Distributor

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

Analog Devices Inc.:

TMCM-1690-COE TMCM-1690-CAN-T TMCM-1690-COE-T TMCM-1690-CANOPEN