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

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

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

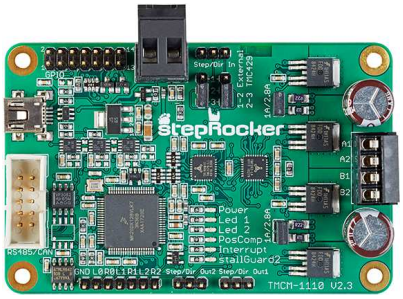
Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

TMCM-1110 TMCL™ Firmware Manual

Firmware Version V1.08 | Document Revision V1.01 • 2017-OCT-10

The TMCM-1110 is a single axis controller/driver module for 2-phase bipolar stepper motors. The TMCM-1110 TMCL firmware allows to control the module using TMCL™ commands, supporting standalone operation as well as direct mode control, making use of the Trinamic TMC429 motion controller and the TMC262 motor driver. Dynamic current control, and quiet, smooth and efficient operation are combined with stallGuard™ and coolStep™ features.



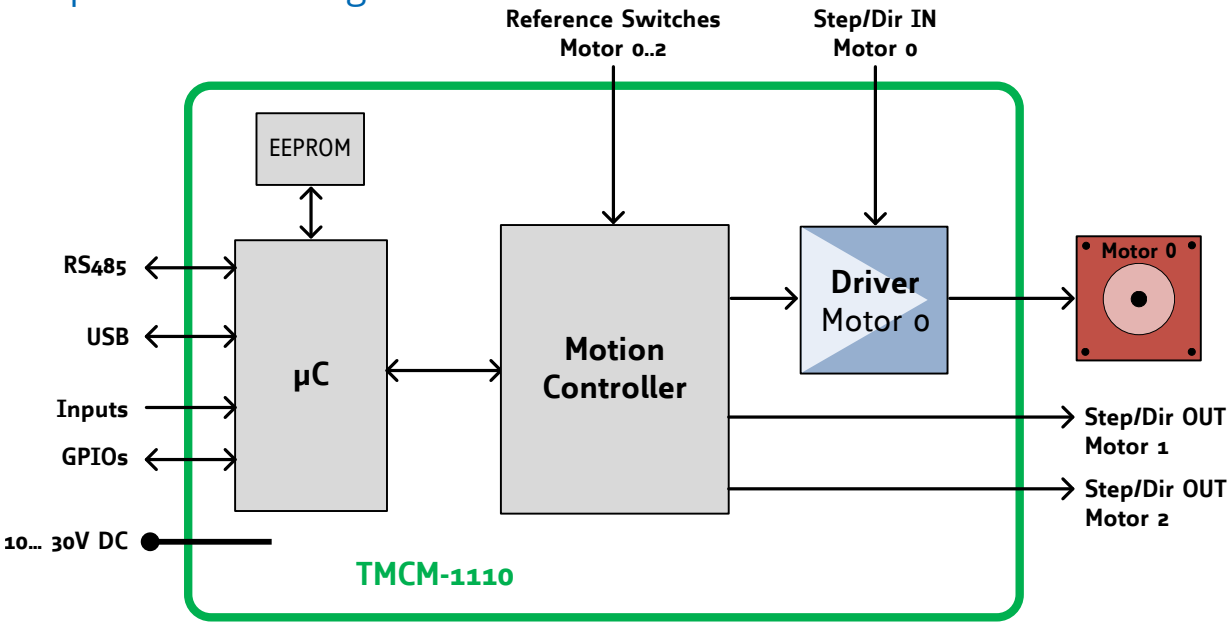
Features

- Single Axis Stepper motor control
- Supply voltage 24V DC
- TMCL™
- USB interface
- RS485 interface
- CAN interface
- coolStep™
- stallGuard2™

Applications

- Laboratory Automation
- Manufacturing
- Robotics
- Factory Automation
- Test & Measurement
- Technology evaluation
- First experiences with stepper motors
- Hobby applications

Simplified Block Diagram



Read entire documentation.

TRINAMIC
MOTION CONTROL

Contents

1 Features

The TMCM-1110 is a single axis controller/driver module for 2-phase bipolar stepper motors with state of the art feature set. It is highly integrated, offers a convenient handling and can be used in many decentralized applications. The module has been designed for coil currents up to 2.8A RMS and 24V DC supply voltage. It is also equipped with end switch inputs as well as some general purpose digital inputs and outputs and one analogue input. With its high energy efficiency from TRINAMIC's coolStep™ technology cost for power consumption is kept down. The TMCL firmware allows for both standalone and direct mode operation.

Main characteristics

- Motion controller & stepper motor driver:
  - Hardware motion profile calculation in real-time.
  - On the fly alteration of motion parameters (e.g. position, velocity, acceleration).
  - High performance microcontroller for overall system control and communication protocol handling.
  - Up to 256 microsteps per full step.
  - High-efficient operation, low power dissipation.
  - Dynamic current control.
  - Integrated protection.
  - stallGuard2™ feature for stall detection.
  - coolStep™ feature for reduced power consumption and heat dissipation.
- Interfaces
  - USB interface.
  - RS485 bus.
  - CAN bus.
  - Additional digital inputs and outputs.
  - One analogue input.
  - End switch inputs.
  - Step/direction input and output.

Software

TMCL remote controlled operation via USB, RS485 or CAN interface and/or stand-alone operation via TMCL programming. PC-based application development software TMCL-IDE available for free.

Electrical data

- Supply voltage: +12V and +24V nominal (10...27V DC supply range).
- Motor current: up to 2.8A RMS / 3.9A peak (programmable).

Please see also the separate Hardware Manual.

1.1 stallGuard2

stallGuard2 is a high-precision sensorless load measurement using the back EMF of the coils. It can be used for stall detection as well as other uses at loads below those which stall the motor. The stallGuard2 measurement value changes linearly over a wide range of load, velocity, and current settings. At maximum motor load, the value reaches zero or is near zero. This is the most energy-efficient point of operation for the motor.

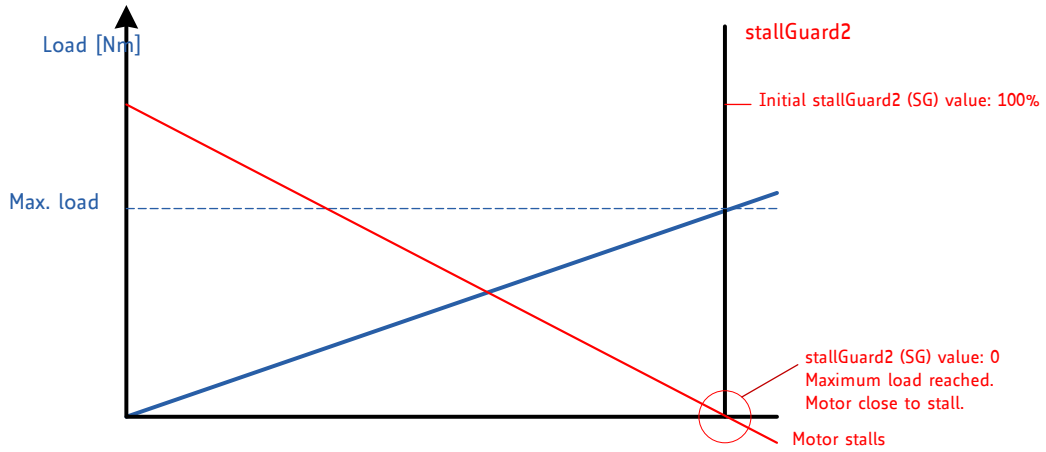


Figure 1: stallGuard2 Load Measurement as a Function of Load

1.2 coolStep

coolStep is a load-adaptive automatic current scaling based on the load measurement via stallGuard2 adapting the required current to the load. Energy consumption can be reduced by as much as 75%. coolStep allows substantial energy savings, especially for motors which see varying loads or operate at a high duty cycle. Because a stepper motor application needs to work with a torque reserve of 30% to 50%, even a constant-load application allows significant energy savings because coolStep automatically enables torque reserve when required. Reducing power consumption keeps the ystem cooler, increases motor life, and allows cost reduction.

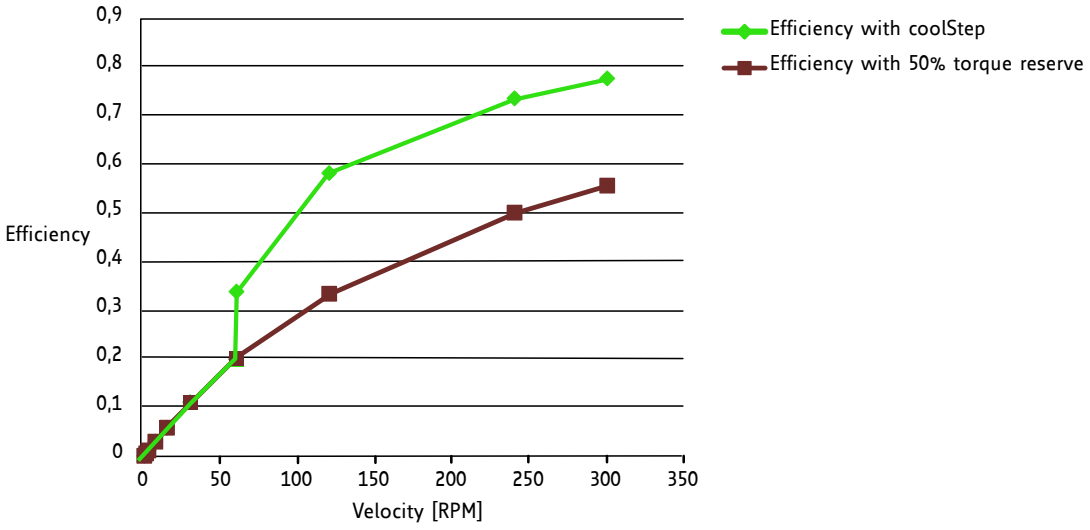


Figure 2: Energy Efficiency Example with coolStep

2 First Steps with TMCL

In this chapter you can find some hints for your first steps with the TMCM-1110 and TMCL. You may skip this chapter if you are already familiar with TMCL and the TMCL-IDE.

Things that you will need

- Your TMCM-1110 module.
- A USB cable.
- A power supply (24V DC) for your TMCM-1110 module.
- The TMCL-IDE 3.x already installed on your PC
- A two-phase bipolar stepper motor.

2.1 Basic Setup

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

Please also ensure that your TMCM-1110 is properly connected to your power supply and that the stepper motor is properly connected to the module. Please see the TMCM-1110 hardware manual for instructions on how to do this. **Do not connect or disconnect a stepper motor to or from the module while the module is powered!**
Then, please start up the TMCL-IDE. After that you can connect your TMCM-1110 via USB and switch on the power supply for the module (while the TMCL-IDE is running on your PC). The module will be recognized by the TMCL-IDE, and necessary driver registrations in Windows will automatically done by the TMCL-IDE.

2.2 Using the TMCL Direct Mode

At first try to use some TMCL commands in direct mode. In the TMCL-IDE a tree view showing the TMCM-1110 and all tools available for it is displayed. Click on the Direct Mode entry of the tool tree. Now, the Direct Mode tool will pop up.

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

2.3 Changing Axis Parameters

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

Most important axis parameters				
Number	Axis Parameter	Description	Range [Units]	Access
4	Maximum positioning speed	The maximum speed used for positioning ramps.	1...2047 [int]	RW
5	Maximum acceleration	Maximum acceleration in positioning ramps. Acceleration and deceleration value in velocity mode.	1...2047 [int]	RW
6	Maximum current	Motor current used when motor is running. The maximum value is 255 which means 100% of the maximum current of the module. *The most important setting, as too high values can cause motor damage.*	0...255	RW
7	Standby current	The current used when the motor is not running. The maximum value is 255 which means 100% of the maximum current of the module. This value should be as low as possible so that the motor can cool down when it is not moving. Please see also parameter 214.	0...255	RW

Table 1: Most important Axis Parameters

2.4 Testing with a simple TMCL Program

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

```
    ROL 0, 1000                 //Rotate motor 0 with speed 1000
    WAIT TICKS, 0, 500
    MST 0
    ROR 0, 100                  //Rotate motor 0 with 1000
    WAIT TICKS, 0, 500
    MST 0

    SAP 4, 0, 100                //Set max. Velocity
    SAP 5, 0, 100                //Set max. Acceleration
Loop:
    MVP ABS, 0, 512000           //Move to Position 512000
    WAIT POS, 0, 0               //Wait until position reached
    MVP ABS, 0, -512000          //Move to Position -512000
    WAIT POS, 0, 0               //Wait until position reached
    JA Loop                      //Infinite Loop
```

After you have done that, take the following steps:

1. Click the Assemble icon (or choose Assemble from the TMCL menu) in the TMCL creator to assemble the program.
2. Click the Download icon (or choose Download from the TMCL menu) in the TMCL creator to 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 on the Bug icon to start the debugger.
2. Click the Animate button to see the single steps of the program.
3. You can at any time pause the program, set or reset breakpoints and resume program execution.
4. To end the debug mode click the Bug icon again.

3 TMCL and the TMCL-IDE — An Introduction

As with most TRINAMIC modules the software running on the microprocessor of the TMCM-1110 consists of two parts, a boot loader and the firmware itself. Whereas the boot loader is installed during production and testing at TRINAMIC and remains untouched throughout the whole lifetime, the firmware can be updated by the user. New versions can be downloaded free of charge from the TRINAMIC website (http://www.trinamic.com).
The TMCM-1110 supports TMCL direct mode (binary commands). It also implements standalone TMCL program execution. This makes it possible to write TMCL programs using the TMCL-IDE and store them in the memory of the module.
In direct mode the TMCL communication over RS-232, RS-485, CAN and USB follows a strict master/slave relationship. That is, a host computer (e.g. PC/PLC) acting as the interface bus master will send a command to the TMCM-1110. The TMCL interpreter on the module will then interpret this command, do the initialization of the motion controller, read inputs and write outputs or whatever is necessary according to the specified command. As soon as this step has been done, the module will send a reply back over the interface to the bus master. Only then should the master transfer the next command.
Normally, the module will just switch to transmission and occupy the bus for a reply, otherwise it will stay in receive mode. It will not send any data over the interface without receiving a command first. This way, any collision on the bus will be avoided when there are more than two nodes connected to a single bus.
The Trinamic Motion Control Language [TMCL] provides a set of structured motion control commands. Every motion control command can be given by a host computer or can be stored in an EEPROM on the TMCM module to form programs that run standalone on the module. For this purpose there are not only motion control commands but also commands to control the program structure (like conditional jumps, compare and calculating).
Every command has a binary representation and a mnemonic. The binary format is used to send commands from the host to a module in direct mode, whereas the mnemonic format is used for easy usage of the commands when developing standalone TMCL applications using the TMCL-IDE (IDE means Integrated Development Environment).
There is also a set of configuration variables for the axis and for global parameters which allow individual configuration of nearly every function of a module. This manual gives a detailed description of all TMCL commands and their usage.

3.1 Binary Command Format

Every command has a mnemonic and a binary representation. When commands are sent from a host to a module, the binary format has to be used. Every command consists of a one-byte command field, a one-byte type field, a one-byte motor/bank field and a four-byte value field. So the binary representation of a command always has seven bytes. When a command is to be sent via RS-232, RS-485, RS-422 or USB interface, it has to be enclosed by an address byte at the beginning and a checksum byte at the end. In these cases it consists of nine bytes.

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

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

Table 2: TMCL Command Format

Info	The checksum is calculated by accumulating all the other bytes using an 8-bit addition.

Note	When using the CAN interface, leave out the address byte and the checksum byte. With CAN, the CAN-ID is used as the module address and the checksum is not needed because CAN bus uses hardware CRC checking.

3.1.1 Checksum Calculation

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

Checksum calculation in C:

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

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

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

Checksum calculation in Delphi:

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

    //Set the Command array to the desired command

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


3.2 Reply Format

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

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

Table 3: TMCL Reply Format

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

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

3.2.1 Status Codes

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

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

Table 4: TMCL Status Codes

3.3 Standalone Applications

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

3.4 TMCL Command Overview

This sections gives a short overview of all TMCL commands.

3.4.1 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 e.g. communication settings or TMCL user variables)
GGP	10	<parameter>, <bank number>	Get global parameter (read out module specific settings e.g. communication settings or TMCL user variables)
STGP	11	<parameter>, <bank number>	Store global parameter (TMCL user variables only)
RSGP	12	<parameter>, <bank number>	Restore global parameter (TMCL user variables only)
RFS	13	<START|STOP|STATUS>, <motor number>	Reference search
SIO	14	<port number>, <bank number>, <value>	Set digital output to specified value
GIO	15	<port number>, <bank number>	Get value of analog/digital input
CALC	19	<operation>, <value>	Process accumulator and 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

Command	Number	Parameter	Description
RSUB	24		Return from subroutine
EI	25	<interrupt number>	Enable interrupt
DI	26	<interrupt number>	Disable interrupt
WAIT	27	<condition>, <motor number>, <ticks>	Wait with further program execution
STOP	28		Stop program execution
SCO	30	<coordinate number>, <motor number>, <position>	Set coordinate
GCO	31	<coordinate number>, <motor number>	Get coordinate
CCO	32	<coordinate number>, <motor number>	Capture coordinate
CALCX	33	<operation>	Process accumulator and X-register
AAP	34	<parameter>, <motor number>	Accumulator to axis parameter
AGP	35	<parameter>, <bank number>	Accumulator to global parameter
CLE	36	<flag>	Clear an error flag
VECT	37	<interrupt number>, <address>	Define interrupt vector
RETI	38		Return from interrupt
ACO	39	<coordinate number>, <motor number>	Accu to coordinate

Table 5: Overview of all TMCL Commands

3.5 TMCL Commands by Subject

3.5.1 Motion Commands

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

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

Table 6: Motion Commands

3.5.2 Parameter Commands

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

Parameter Commands		
Mnemonic	Command number	Meaning
SAP	5	Set axis parameter
GAP	6	Get axis parameter
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 in standalone mode.

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

Table 9: I/O Port Commands

3.5.5 Calculation Commands

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

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

Table 10: Calculation Commands

For calculating purposes there is an accumulator (also called accu or A register) and an X register. When executed in a TMCL program (in standalone mode), all TMCL commands that read a value store the result

in the accumulator. The X register can be used as an additional memory when doing calculations. It can be loaded from the accumulator.
When a command that reads a value is executed in direct mode the accumulator will not be affected. This means that while a TMCL program is running on the module (standalone mode), a host can still send commands like GAP and GGP to the module (e.g. to query the actual position of the motor) without affecting the flow of the TMCL program running on the module.

3.5.6 Interrupt Processing Commands

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

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

Table 11: Interrupt Processing Commands

3.5.6.1 Interrupt Types

There are many different interrupts in TMCL, like timer interrupts, stop switch interrupts, position reached interrupts, and input pin change interrupts. Each of these interrupts has its own interrupt vector. Each interrupt vector is identified by its interrupt number. Please use the TMCL include file Interrupts.inc in order to have symbolic constants for the interrupt numbers. Table 12 show all interrupts that are available on the TMCM-1110.

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

Interrupt number	Interrupt type
43	Input change 4
44	Input change 5
255	Global interrupts

Table 12: Interrupt Vectors

3.5.6.2 Interrupt Processing

When an interrupt occurs and this interrupt is enabled and a valid interrupt vector has been defined for that interrupt, the normal TMCL program flow will be interrupted and the interrupt handling routine will be called. Before an interrupt handling routine gets called, the context of the normal program (i.e. accumulator register, X register, flags) will be saved automatically.
There is no interrupt nesting, i.e. all other interrupts are disabled while an interrupt handling routine is being executed.
On return from an interrupt handling routine (RETI command), the context of the normal program will automatically be restored and the execution of the normal program will be continued.

3.5.6.3 Further Configuration of Interrupts

Some interrupts need further configuration (e.g. the timer interval of a timer interrupt). This can be done using SGP commands with parameter bank 3 (SGP <type> , 3, <value>). Please refer to the SGP command (chapter 3.6.9) for further information about that.

3.5.6.4 Using Interrupts in TMCL

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

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

The following example shows the use of a timer interrupt:

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

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

//Here is the interrupt handling routine
```


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

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

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

3.6 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 internal units ([int]) of the TMC429 motion controller used on the TMCM-1110 module. Please see also section 6.1.

Internal function:

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

Related commands: ROL, MST, SAP, GAP.

Mnemonic: ROR <axis>, <velocity>

Binary Representation			
Instruction	Type	Motor/Bank	Value
2	0	0	-2047...2047

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

Example
Rotate right motor 0, velocity 1000.
Mnemonic: ROR 0, 1000.

Binary Form of ROR 0, 1000	
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)	03_h
Value (Byte 0)	$E8_h$
Checksum	ED_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 internal units ([int]) of the TMC429 motion controller used on the TMCM-1110 module. Please see also section 6.1.

Internal function:

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

Related commands: ROR, MST, SAP, GAP.

Mnemonic: ROL <axis>, <velocity>

Binary Representation			
Instruction	Type	Motor/Bank	Value
2	0	0	-2047...2047

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

Example
Rotate left motor 0, velocity 1000.
Mnemonic: ROL 0, 1000.

Binary Form of ROL 0, 1000	
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)	03_h
Value (Byte 0)	$E8_h$
Checksum	EE_h

3.6.3 MST (Motor Stop)

The motor is instructed to stop with a soft stop.

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

Mnemonic: MST <axis>

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

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

Example
Stop motor 0.
Mnemonic: MST 0.

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

3.6.4 MVP (Move to Position)

With this command the motor will be instructed to move to a specified relative or absolute position. It will use the acceleration/deceleration ramp and the positioning speed programmed into the unit. This command is non-blocking - that is, a reply will be sent immediately after command interpretation and initialization of the motion controller. Further commands may follow without waiting for the motor reaching its end position. The maximum velocity and acceleration as well as other ramp parameters are defined by the appropriate axis parameters. For a list of these parameters please refer to section 4.
The range of the MVP command is 32 bit signed (-2147483648...2147483647). Positioning can be interrupted using MST, ROL or ROR commands.

Three operation types are available:

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

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

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

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

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

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

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

Binary Form of MVP ABS, 0, 90000	
Field	Value
Target address	01_h
Instruction number	04_h
Type	00_h
Motor/Bank	00_h
Value (Byte 3)	00_h
Value (Byte 2)	10_h
Value (Byte 1)	$5F_h$
Value (Byte 0)	90_h
Checksum	$F5_h$

Example
Move motor 0 from current position 10000 microsteps 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