

# Motor Control Devices

## User's Manual

Version 1.9



Version: 1.9  
Last revision: April 5, 2019  
Printed in Switzerland

© Copyright 2002-2019 FiveCo Sàrl. All rights reserved.  
The contents of this manual may be modified by FiveCo without any warning.

---

#### Trademarks

Windows® is a registered trademark of Microsoft Corporation.

Ethernet® is a registered trademark of Xerox Corporation.

Java® is a registered trademark of Sun Microsystems.

#### Warning

This device is not intended to be used in medical, life-support or space products.

Any failure of this device that may cause serious consequences should be prevented through the implementation of backup systems. The user agrees that protection against consequences resulting from device system failure is the user's responsibility. Changes or modifications to this device not explicitly approved by FiveCo will void the user's authority to operate this device.

#### Support

Web page: <http://www.fiveco.ch/motor-controllers-products.html>

e-mail: [support@fiveco.ch](mailto:support@fiveco.ch)

## Revision history




Revision	Date	Author	Note	Firmware version	Applet version	FSoft-MotorCtrl version
1.0	08.06.2016	KB	- First revision based on FMod-IPECMOT 48/10 v3.4 and FMod-I2CDCMOT DB 48/1.5 v2.4 user's manual	FMod-IPECMOT 48/10 T1 Since 5.0 FMod-I2CSTEPMOT SLP 35/1 & 35/0.1 Since 3.2 FMod-I2CDCMOT SLP 48/1 Since 2.2 FMod-I2CDCMOT DB 48/1.5 Since 3.14 FMod-I2C485ECMOT DB 48/10 Since 3.6	4.0	2.14
1.1	23.06.2016	KB	- Correction made on thermal limitation of the controllers	Same as above	4.0	2.14
1.2	09.09.2016	KB	- Correct <b>CURRENTMAX</b> for FMod-I2CSTEPMOT 35/0.1 - Fmod-IPECMOT 48/10 T2 final release	FMod-I2CSTEPMOT SLP 35/1 & 35/0.1 Since 3.2 FMod-IPECMOT 48/10 T2 Since 3.1	4.0	2.14
1.3	29.05.2017	XG	- IOState 0x56 + IOCfg 0x55	7.0	4.1	3.01
1.4	05.07.2017	XG	Options.12 Dead zone linear Optiona.13 IO on Address bus	7.2	4.1	3.02
1.5	21.07.2017	PHD	- FeedForward registers - FeedForward explanation	7.2	4.1	3.01
1.6	14.08.2017	KB	- I2C and RS485 address selection improved explanation as well as I/Os - J2 instead of J1 for FMod-I2CSTEPMOT and FMod-I2CDCMOT	-	4.1	3.02
1.7	04.10.2017	XG	- Warnings.26 Index state	7.5	4.1	3.05
1.8	22.12.2017	KB	- Improved explanation on Enable pin for FMod-IPECMOT 48/10 T1 & T2	-	4.1	3.06
1.9	05.04.2019	XG	- TrackPosition for multi synchro - Encoder I failure detection - Communication Watchdog		4.1	3.10



## Table of Contents



1. FiveCo's motor controllers product line.....	6
2. FMod-IPECMOT 48/10 T1 & T2.....	9
Operating conditions .....	9
Overview.....	10
Quick start for FMod-IPECMOT 48/10 T1 & T2.....	15
Hardware .....	18
Java Applet.....	25
3. FMod-I2C485ECMOT DB 48/10.....	28
Operating conditions .....	28
Overview.....	28
Hardware .....	32
4. FMod-I2CDCMOT DB 48/1.5 & SLP 48/1 .....	39
Operating conditions .....	39
Overview.....	40
Hardware .....	45
5. FMod-I2CSTEPMOT SLP 35/1 & 35/0.1 .....	48
Operating conditions .....	48
Overview.....	49
Hardware .....	52
Position and phases synchronisation .....	54
6. Ethernet Interface.....	57
General.....	57
TCP-HTTP port (TCP # 80).....	58
Control ports (TCP # 8010) & (UDP #7010).....	58
Easy IP address config (UDP # 7010).....	60
Checksum calculation .....	62
7. I2C Interface.....	64
Description .....	64
Protocol .....	64
Sequence.....	65
Write Sequence (1 byte and 4 bytes).....	67
Read Sequence (1 byte and 4 bytes).....	68
8. RS485 Interface .....	69
Physical layer – RS485.....	69
Registers Access Protocol .....	70
9. Configuration software : FSoft-MotorCtrl.....	73
Overview.....	73
10. Motion Control modes.....	79
Motor regulation parameters.....	79
List of regulation modes.....	81
Brake mode.....	81
Driver Open mode.....	82

Open Loop mode.....	82
Wait mode.....	82
Speed Control mode.....	83
Position Control mode.....	88
Standby mode.....	93
11. Auto-tuning.....	94
What does auto-tuning consist of? .....	94
12. Limit switches.....	96
13. Homing (position reference).....	97
List of Homing methods.....	98
Homing method 0: Actual position is correct, don't alter .....	100
Homing method 1: Actual position is correct, set new INPUT.....	100
Homing method 2: Move backward (-) to the first index .....	100
Homing method 3: Move forward (+) to the first index.....	100
Homing method 4: Move backward (-) to the mechanical limit.....	101
Homing method 5: Move forward (+) to the mechanical limit.....	101
Homing method 6: Move backward (-) to a mechanical limit and index .....	102
Homing method 7: Move forward (+) to a mechanical limit and index .....	102
Homing method 8: Move backward (-) to limit switch 1 .....	103
Homing method 9: Move forward (+) to Limit Switch 1 .....	103
Homing method 10: Move backward (-) to Limit Switch 1 and index.....	104
Homing method 11: Move forward (+) to Limit Switch 1 and index .....	104
Homing method 12: Move of Start Input .....	105
Homing method 13: Move of Start Input to the Limit Switch 1 .....	106
14. Loops management.....	107
Overview.....	107
Loops configuration .....	108
Loops mode.....	108
Loops Options and Status.....	110
Peak current management.....	110
Using Loops mode example .....	111
15. Register management.....	112
Memory organization.....	112
Full description of registers.....	113

## I. FiveCo's motor controllers product line

Model	FMod-IPECMOT T2 48/10	FMod-IPECMOT T1 48/10	FMod-I2C485ECMOT DB 48/10
			
Dimension (LxBxH) [mm]	120x110x34 (DIN rail)	120x110x34 (DIN rail)	80x56x25
Motor type	DC brushed DC brushless	DC brushed DC brushless	DC brushed DC brushless
Communication bus	Ethernet TCP/IP (1ms)	Ethernet TCP/IP (2ms)	I2C RS485
Power supply (input)	DC [9-48V], max 10A	DC [15-48V], max 15A	DC [15-48V], max 15A
Logic supply (input)	Internally generated	Internally generated	Internally generated
Encoder	5 V 2 channels quadrature incremental with differential output + index (A/B 500kHz = 2M p/s)	5 V 2 channels quadrature incremental with differential output + index (A/B 250kHz = 1M p/s)	5 V 2 channels quadrature incremental with differential output + index (A/B 250kHz = 1M p/s)
Limits / Inputs	2 limits + 2 IOs	2 limits + 1IO	2 limits
Phases output	PWM 125kHz or 62kHz 4 quadrants management Thermal protection 10 A max Current measurement Variable self filter 125uH @5A, 75uH @10A	PWM 69kHz or 35kHz 4 quadrants management Thermal protection 10 A continuous 15 A max	PWM 69kHz or 35kHz 4 quadrants management Thermal protection 10 A continuous 15 A max
Motion control	32 bit PID Auto-tuning . Brake mode . Free mode . Open loop mode . Speed control mode . Position control mode	32 bit PID Auto-tuning . Brake mode . Free mode . Open loop mode . Speed control mode . Position control mode	32 bit PID Auto-tuning . Brake mode . Free mode . Open loop mode . Speed control mode . Position control mode
Standby mode	-	-	-
Extra feature	EC motor's Hall sensor can be used as encoders <b>Dual encoder management</b> Active dissipation Phases short-circuited @ power-down	EC motor's Hall sensor can be used as encoders	EC motor's Hall sensor can be used as encoders

Model	FMod-I2CDCMOT DB 48/1.5	FMod-I2CDCMOT SLP 48/1
		
Dimension (LxBxH) [mm]	48x34x23	26x28.5x6
Motor type	DC brushed	DC brushed
Communication bus	I2C	I2C
Power supply (input)	DC [10-48V], max 2A	DC [10-48V], max 2A
Logic supply (input)	DC [5V], max 50mA	DC [5V], max 50mA
Encoder	5 V 2 channels quadrature incremental (A/B 250kHz = 1M p/s)	5 V 2 channels quadrature incremental (A/B 500kHz = 2M p/s)
Limits / Inputs	2 limits	2 limits + 2 IOs
Phases output	PWM 69kHz or 35kHz 4 quadrants management Thermal protection 1.5 A continuous 2.0 A max	PWM 69kHz or 35kHz 4 quadrants management Thermal protection 1 A continuous 2.0 A max
Motion control	32 bit PID Auto-tuning . Brake mode . Free mode . Open loop mode . Speed control mode . Position control mode	32 bit PID Auto-tuning . Brake mode . Free mode . Open loop mode . Speed control mode . Position control mode
Standby mode	-	Max 1 uA @ 5V logic <b>(50nA at 25°C)</b> Max 1 uA @ motor supply <b>(50nA at 25°C)</b>
Extra feature	-	Low power mode

Model	FMod-I2CSTEPMOT SLP 35/I	FMod-I2CSTEPMOT SLP 35/0.I
		
Dimension (LxBxH) [mm]	39.8x22x6	39.8x22x6
Motor type	2 phases stepper bipolar	2 phases stepper bipolar
Communication bus	I2C	I2C
Power supply (input)	DC [9-35V], max 2A	DC [9-35V], max 150mA
Logic supply (input)	DC [5V], max 50mA	DC [5V], max 50mA
Encoder	-	-
Limits / Inputs	2 limits	2 limits
Phases output	PWM 50kHz Max 8'192 full steps/s Resolution: 1/4 step Thermal protection 1 A continuous 1.5 A max	PWM 50kHz Max 8'192 full steps/s Resolution: 1/4 step Thermal protection 150mA max and continuous
Motion control	. Open mode . Speed control mode . Position control mode	. Open mode . Speed control mode . Position control mode
Standby mode	Max 1 uA @ 5V logic <b>(50nA at 25°C)</b> Max 1 uA @ motor supply <b>(50nA at 25°C)</b>	Max 1 uA @ 5V logic <b>(50nA at 25°C)</b> Max 1 uA @ motor supply <b>(50nA at 25°C)</b>
Extra feature	Low power mode	Low power mode



## 2. FMod-IPECMOT 48/10 T1 & T2

### Operating conditions

#### FMod-IPECMOT 48/10 T2

▪ Operating temperature	0 – 70 °C
▪ Supply voltage Vcc	9-48 VDC
▪ Supply current	max ±10 A
▪ Power consumption	77mA when idle @ Vcc 12V 42mA when idle @ Vcc 24V 30mA when idle @ Vcc 48V
▪ Input capacity (Power +,-)	~3000uF, ~10mOhm ESR
▪ 5V out (encodeur + hall + limits)	5.0 V (max 300 mA)
▪ Max A&B encoder signal	500 kHz = 2Mio pulses/s (CPRx4)
▪ Output voltage	90% Vcc @ 62.5kHz PWM 83% Vcc @ 125kHz PWM
▪ Max. output current	10 A
▪ Continuous <b>OUTPUT</b> current (Pwr Mosfet thermal limitation)	Brushed and brushless motor 8A (Tamb. 25°C, vertically mounted!)

#### FMod-IPECMOT 48/10 T1

▪ Operating temperature	0 – 70 °C
▪ Supply voltage Vcc	15-48 VDC
▪ Supply current	max ±10 A
▪ Power consumption	85mA when idle @ Vcc 15 V 65mA when idle @ Vcc 24 V 45mA when idle @ Vcc 48 V
▪ Input capacity (Power +,-)	~3000uF, ~10mOhm ESR
▪ 5V out (encodeur + hall + limits)	5.0 V (max 150 mA)
▪ Max A&B encoder signal	250 kHz = 1Mio pulses/s (CPRx4)
▪ Output voltage	92% Vcc @ 35kHz PWM 85% Vcc @ 69kHz PWM
▪ Max. output current	15 A
▪ Continuous <b>OUTPUT</b> current (Pwr Mosfet thermal limitation)	Brushed Motor 10A (Tamb. 25°C) Brushless Motor 8A (Tamb. 25°C) (Vertically mounted!)

## Overview

---

### Applications

---

The FMod-IPECMOT 48/10 T1 & T2 are motion control and driver devices (up to 480W) for DC motors, either with brushes or brushless (EC) with hall sensors. It is particularly interesting not only because of its small size and quality of the regulation system but also because of its communication protocol (Ethernet: TCP/IP-HTTP) which gives it an easy interfacing capability.

Both cards can be accessed through a TCP connection (socket) from any computer or through a simple web page using a standard browser, allowing for the easy setup of the controller.

The device connections are described on the following pages.

For a "Quick start", go to page 15.

### Type 2 updates

---

Since the FMod-IPECMOT 48/10 T2 is an upgrade made on the T1, they share the same scope of applications. The T2 main updates made compared to the T1 are explained below.

The dissipation of the breaking energy is a security feature that ensures a small increase of the supply voltage even when performing a fast break at high speed and high torque.

The second encoder management is useful for applications where a possible "slippery" transmission exists between the motor output shaft and the final desired movement. The second encoder is to be implemented on the output movement and the first one on the motor.

Moreover, the supply power range has been increased, [15-48V] for T1 and [9-48V] for T2.

Another feature is the implementation of a relay, short-circuiting the phases of the motor when the FMod-IPECMOT 48/10 T2 is powered down, therefore performing a magnetic brake.

Finally a measurement of the current through the motor windings is accessible externally in a new register **CURRENTSENSE** (0x2B).

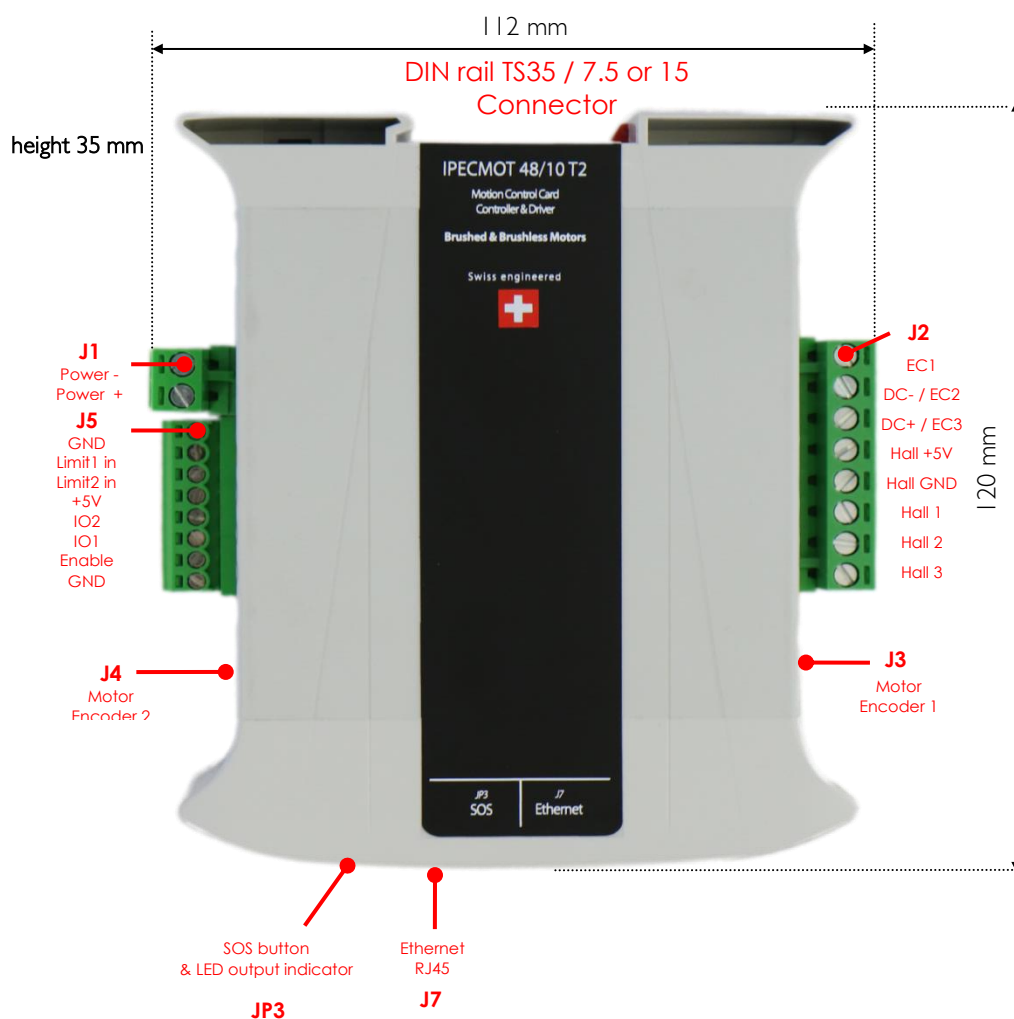
## Software operating principle

The PC software that exchanges data with this device will use a dedicated protocol layer on top of the TCP Layer (see “6. Ethernet Interface” chapter). This protocol is Question & Answer oriented. The PC will send a question, wait for the answer and so on.

To configure the card's parameters, the protocol uses an Internal Register Access routine (see “15. Register management” chapter).

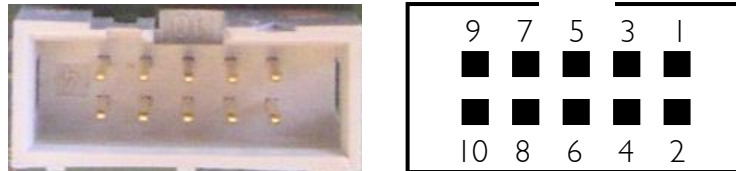
The Java library and the Embarcadero (Borland) C++ code sample and DLL are available from <http://www.fiveco.ch/produit-fmod-ipecmot-4810-t1.html>. It can help programmers get started with their development.

## Hardware description of FMod-IPECMOT 48/10 T2



### Connector J3 and J4 pinning

The below figure shows the pinning from the incremental encoders; the connector reference on the PCB side is : DIN41651 2x5 pins.

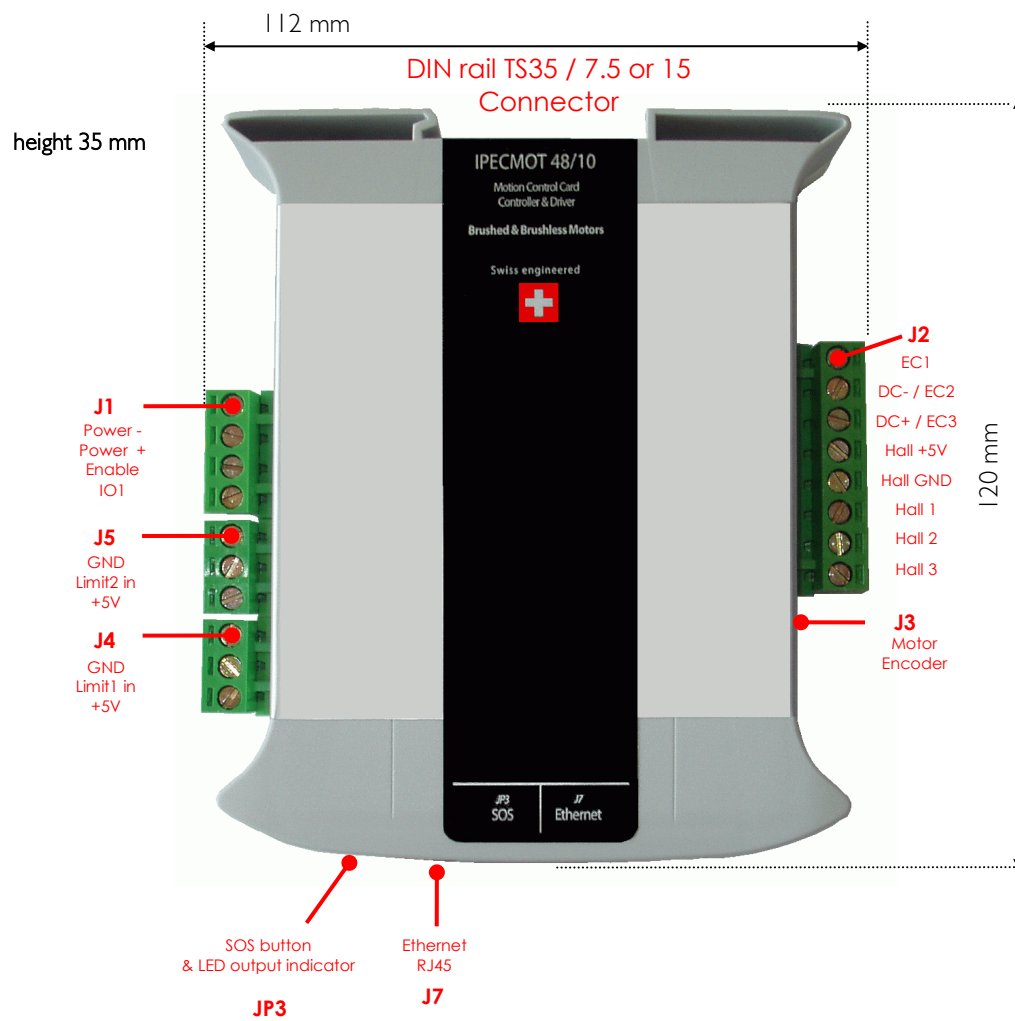


- 1 Not connected
- 2 5V (<150mA) (out)
- 3 GND (out)
- 4 Not connected
- 5 Encoder channel A inverted (in)
- 6 Encoder channel A (in)
- 7 Encoder channel B inverted (in)
- 8 Encoder channel B (in)
- 9 Encoder Index inverted (in)
- 10 Encoder Index (in)

#### Notes:

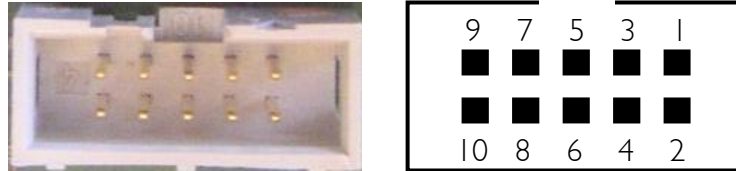
- The device's logic power supply is internally generated by the card from the DC supply. It can be used externally to power limits, hall sensors and encoder to a maximum of 300mA in total.

## Hardware description of FMod-IPECMOT 48/10 T1



### Connector J3 pinning

The below figure shows the pinning from the incremental encoder; the connector reference on the PCB side is : DIN41651 2x5 pins.



- 1 Not connected
- 2 5V (<150mA) (out)
- 3 GND (out)
- 4 Not connected
- 5 Encoder channel A inverted (in)
- 6 Encoder channel A (in)
- 7 Encoder channel B inverted (in)
- 8 Encoder channel B (in)
- 9 Encoder Index inverted (in)
- 10 Encoder Index (in)

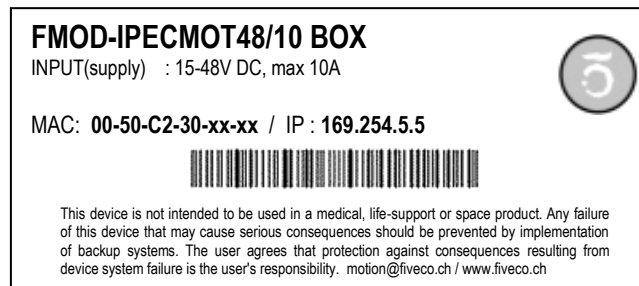
#### Notes:

- The device's logic power supply is internally generated by the card from the DC supply. It can be used externally to power limits, hall sensors and encoder to a maximum of 200mA in total.

## Quick start for FMod-IPECMOT 48/10 T1 & T2

This section is intended to help users quickly plug the device into their system and establish a connection between the computer and the device. Detailed information about hardware and software is provided further in this document.

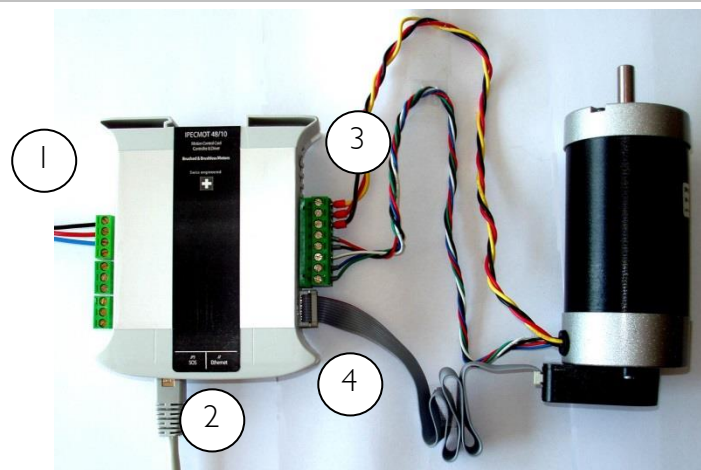
You will find the device's factory communication settings on the box label.



The MAC address is the 48-bit unique identifier on Ethernet networks. The IP address can be modified. Refer to the “Plug and Play” and “Change IP address” chapters to complete the first configuration of the controller.

**Note:** *If the device has already been configured and the IP address has been changed to an unknown value, you can retrieve an SOS IP address (the one on label) by pressing the “SOS button” during the normal operation of the device.*

### Plug and Play



1. Connect the DC power ([15-48V] for T1 or [9-48V] for T2) and Enable pin to the device. (The Enable pin can be connected to the “Power +” pin, if not needed). Although it should be used as an emergency switch to stop the motor.
2. Connect the device to a computer using a RJ45 **cross-wired** cable (direct link), or with a straight cable to an Ethernet switch.

3. Connect the motor to the device. For brushless motors, see “How to connect hall sensors” chapter.
4. Connect the 10-pin encoder.
5. Download the free windows application “FSoft-MOTORCTRL.exe” from <http://www.fiveco.ch/produit-fmod-ipecmot-4810-t1.html> to your hard disk.
6. Deactivate your computer’s firewall software or configure it to accept TCP/IP connections and broadcast messages from “FSoft-MOTORCTRL.exe”.
7. Run “FSoft-MOTORCTRL.exe”; if the IP address of the controller needs to be modified refer to the “Change IP address” next section to configure the FMod-IPECMOT 48/10 T1 or T2 IP address to a valid IP on your network.
8. Click on “Communication→Scan Network” and select your FMod-IPECMOT 48/10 T1 or T2 (using the MAC address).
9. Set the regulation mode to “Open Loop” and send a new input (-65536...+65536) using the track bar. The interface should update the position smoothly, the connection should not get disconnected; if this is the case, the IP address might not be well configured.
10. The motor should be moving. Select the “Auto-tuning” menu and follow the instructions.
11. Set the regulation mode for your application (speed or position) and move the input track bar to send the new input.



## Change IP address

---

To easily change the factory IP address, the user can use the "FSoft-MOTORCTRL.exe" software available at <http://www.fiveco.ch/produit-fmod-ipecmot-4810-t1.html> under the "support" section.

1. Connect your new device to your PC network.
2. Start the FSoft-MotorCtrl application.
3. Click on "Communication → Easy change IP".
4. The software will scan the network and display a list of all FiveCo devices found.
5. Select the MAC address corresponding to your new device.
6. If you have more than one network adapter on your PC, the software will ask you to select the one which is connected to the same network as the FMod-IPECMOT 48/10.
7. The software will suggest a new IP address without the last byte. Choose a new IP (**one that is not already used on your network!!**) and click the "Change IP address" button.

All done! The device has a new address and subnet mask (the same as your PC). These are automatically saved to EEPROM.

You can now connect to the device with the Win32 software or open its web page by typing the new IP address into a web browser.

Please note:

The IP address won't be changed if a TCP connection exists with the device.

## Hardware

---

### Power supply

---

[15-48V] for T1 or [9-48V] for T2 with the peak current equal to the nominal current of the motor on connector J1 (min 1A). E.g. for a motor with a constant of 24V and 2A, choose a power supply of 24V 2A. Don't worry about the motor start current you will see on the datasheet, it is electronically limited by the device.

**WARNING: When braking (deceleration), the 4Q device regenerates the inertial energy to electricity, so the voltage (J1 Power+ and Power-) could be higher than the power supply.**

For FMod-IPECMOT 48/10 T1, the device will accept overrides if below 50V, or if between 51-53V inside the clamping diode, it will temporarily accept 3-5A. If your system does not accept overrides, you need to add an external dissipation system for example.

For FMod-IPECMOT 48/10 T2, active dissipation hardware is included on-board. Refer to the next chapter "Dissipation" for more information.

**The power supply has to be able to manage reverse power when braking (power generation) on connector J1 Power+ and Power-.**

For FMod-IPECMOT 48/10 T1

- > Below 12.0V, the driver is automatically set to Brake mode.
- > Over 56V, the driver is automatically set to DriverOpen mode.

For FMod-IPECMOT 48/10 T2

- > Below 8.0V, the driver is automatically set to Brake mode.
- > Over 56V, the driver is automatically set to DriverOpen mode.

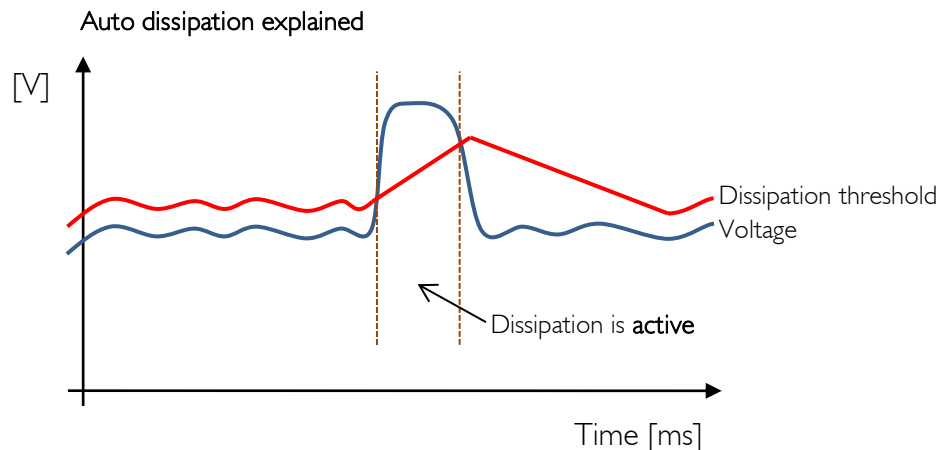
### Dissipation

---

Active dissipation is a feature of the FMod-IPECMOT 48/10 T2. It ensures a constant current dissipation of 3A, for any power supply voltage, through power transistors inside the case.

The dissipation is activated when the voltage measured by the controller exceeds a certain threshold. This threshold could be set by the user with the register **DISSIPATIONVOLTAGE** (0x2F), or use the automatic feature setting the **OPTIONS.16** bit (0x2C).

When the automatic threshold is used, the dissipation voltage threshold stabilizes 3 V above the actual average of the measured input power voltage. When a too abrupt rise on the measured voltage occurs, the dissipation is activated while the threshold is still updated with the current average. The following figure shows this mechanism.



The fixed **DISSIPATIONVOLTAGE** (0x2F) register is not saved; therefore it is not restored at the card power-up (security reasons). When the user uses the fixed dissipation voltage, it has to be sent to the card after each power-up.

An estimated temperature of the dissipation power transistors are accessible through a new register: **DISSIPTEMPERATURE** (0x60). When the dissipation temperature is above 170°C, the dissipation feature is disabled to prevent damaging the controller. When the temperature of the dissipation transistors drops below 170°C, the dissipation feature is automatically re-enabled.

See register description **OPTIONS** (0x2C), **DISSIPATIONVOLTAGE** (0x2F) and **DISSIPTEMPERATURE** (0x60) for more information.

## Motor type

The device drives DC brushed motors as well as brushless motors with hall sensors, (0.1-10A, 3-48V).

FMod-IPECMOT 48/10 T1 & T2 automatically detects whether the motor is brushed or brushless.

If speed control or positioning is needed, use A&B (&Index) channel quadrature incremental encoders with a differential line driver (EIA/TIA/RS 422). FMod-IPECMOT 48/10 T2 is compliant with non-differential encoder lines. Each logical change of A or B channels will define a pulse.

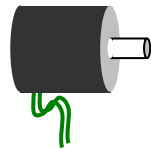
If the encoder is defined in cycles (or counts) per revolution (CPR), you can multiply the cycles by 4 to obtain the result in pulses (PPR).

For example: an encoder of 500 CPR represents 2000 PPR in the controller.

With brushless motors only, it is possible to configure the device to use the hall sensors (6 states/cycle) for speed control and positioning, but the resolution is poor.

### Brushed motor (DC)

---

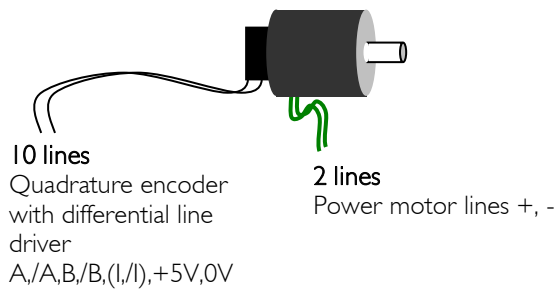


2 lines  
Power motor lines +, -

Open Loop only, no regulation because no feedback!

### Brushed motor (DC) with encoder

---

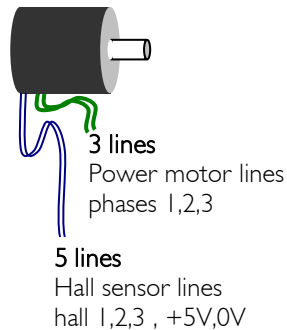


Full regulation: speed or positioning.

(\*) For FMod-IPECMOT 48/10 T2, also compliant with non-differential encoder lines

### Brushless motor (EC)

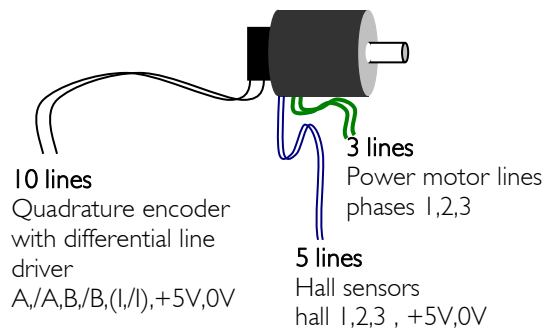
---



Full regulation: speed or positioning with hall sensors as encoders.  
Poor regulation due to the few states of hall sensors.  
(see **OPTIONS** register (0x2C) to set this feature)

### Brushless motor (EC) with encoder

---



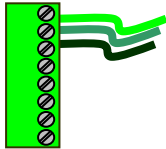
Full regulation: speed or positioning

(\*) For FMod-IPECMOT 48/10 T2, also compliant with non-differential encoder lines

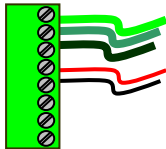
## How to connect hall sensors

Different motor manufacturers do not have the same enumeration for hall sensors 1,2,3 and motor phases (windings) 1,2,3.

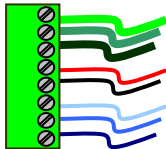
Here is a method to find the corresponding connection between the brushless motor and the device.



Connect phases (windings) 1,2,3 (U,V,W=A,B,C) to the FMod-IPECMOT 48/10 motor connector J2, through pins EC1, EC2, EC3.

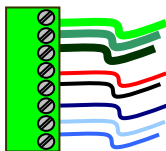


Connect the +5V and GND hall sensors to the J2 connector.



Connect the hall sensor lines 1,2,3 to the J2 connector.

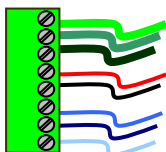
Power the device, connect the Ethernet cable, set the Enable pin to high, set the **REGULATIONMODE** register to "Open Loop", set **INPUT** to 48'000 (~3/4 full PWM).  
If the motor turns correctly, you have done it!



If the motor does not turn, set **REGULATIONMODE** to "Driver Open", and switch the hall sensor lines: Hall1 → Hall2, Hall2 → Hall3, Hall3 → Hall1.

Set **REGULATIONMODE** to "Open Loop", set **INPUT** to 48'000.

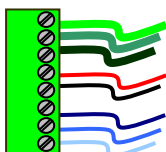
If the motor turns correctly, you have done it!



If the motor does not turn, do the same manipulation one more time: switch the hall sensor lines: Hall1 → Hall2, Hall2 → Hall3, Hall3 → Hall1 ...

Set **REGULATIONMODE** to "Open Loop", set **INPUT** to 48'000.

If the motor turns correctly, you have done it!



If it still isn't working, unlucky!

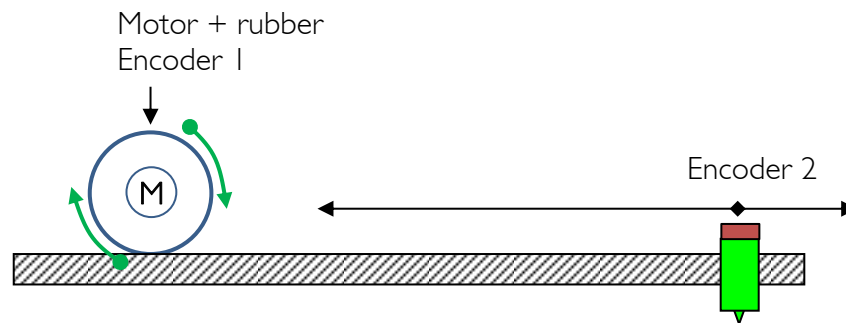
Swap lines Hall 1 ↔ Hall 2

Test it and if it still doesn't work, switch the hall sensor lines: Hall1 → Hall2, Hall2 → Hall3, Hall3 → Hall 1, test again, and switch one more time.

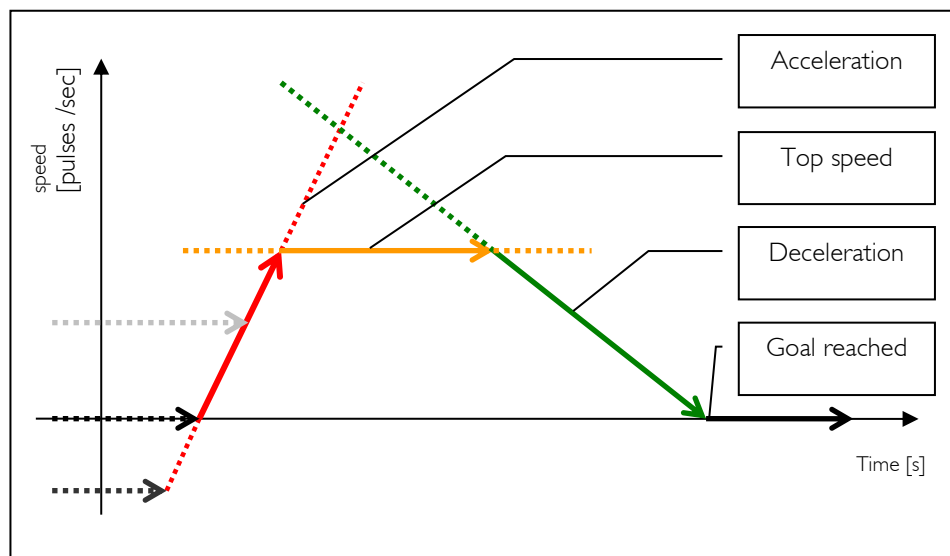
If it still doesn't work, the motor or device is most likely damaged.

## Dual encoder management

This is a feature of the FMod-IPECMOT 48/10 T2 that is really useful when the transmission between the motor and the final movement can be slippery. The dual encoder management is enabled by setting the **OPTIONS.10** bit to '1'. The second encoder can be inverted setting the **OPTIONS.11** bit to '1'.



As in the example above, the 1<sup>st</sup> encoder is located on the motor while the 2<sup>nd</sup> one is on the final output movement. When reading the **SPEED** (0x28) register, it is expressed in pulse/sec of the 1<sup>st</sup> encoder, while the **POSITION** (0x26) register is expressed in pulses of the 2<sup>nd</sup> encoder. Let's look at the trapezoidal speed trajectory made by the motor in position control mode.



Speed profile in the "Position Control mode"

The Acceleration and the Top speed phases are calculated with respect to the encoder 1. However the speed consign during the Deceleration phase is calculated with respect to the encoder 2, since the position is updated with the 2<sup>nd</sup> encoder. Therefore a relation has to be made between the encoder 2 "space" and the encoder 1, since the speed consign is calculated with the 2<sup>nd</sup> encoder but **applied** to the 1<sup>st</sup> encoder. The register **ENCODERSRATIO** (0x44) makes the link between the encoder 1 and 2.

The **ENCODERSRATIO** (0x44) is the ratio between the encoder 1 and the encoder 2. Which means that if an increase of 1'000 pulses on the 1<sup>st</sup> encoder changes the 2<sup>nd</sup> encoder of 500 pulses, the ratio is 2. Refer to the register (0x44) description under the "Register management" chapter to have more information on how to write this value on the controller.

The **AUTO-TUNING** (0x39) function can easily estimate this value when both encoders are connected and the **OPTIONS.10** (use encoder 2) bit is set to '1'. If the **AUTO-TUNING** cannot be used, the user can set the value of **ENCODERSRATIO** manually; it does not need to be the exact ratio ( $\pm 10\%$  is completely suitable) since the regulation during the deceleration phase will compensate for the error in the ratio.

The best situation is when both encoders have a good resolution, typically > 1000 CPR, and the ratio is close to unity.

## Enable pin

---

This feature is for security purposes and stops the output power (connector J1).

For the FMod-IPECMOT 48/10 T1, the voltage ranging between [4.2V; 48V] on the **Enable pin** allows the driver to work normally (regulation on motor).

If the voltage on the **Enable pin** drops below 1.6V for FMod-IPECMOT 48/10 T1 or 4V for FMod-IPECMOT 48/10 T2, the driver (motor) is set to "Brake" mode.

When the **Enable pin** goes back to high [4.2V; 48V], the user needs to change the mode from "Brake" back to the required one.

## Limit Switch: stop, reference, stopper type

---

Both Limit1 and Limit2 are inputs lines with a Schmitt trigger (logic low : [0-1V], logic high [4-5V]).

5 VDC sensors can be powered from the connector (J4 and J5 for T1 and J5 for T2). Open collector (NPN,PNP) and open drain (N,P) output stage sensors can be used.

Micro switches or reed sensors can also be used, but take a special care to the rebounds of the limit signal (typ < 10ms).

**WARNING:** Both Limit Switch sensors (L1 & L2) must be of the same type (NPN,N or PNP,P) because of the internal pull-up or pull-down resistor selection.

## Inputs and outputs

---

Only 1 I/O is present on the FMod-IPECMOT 48/10 T1 (connector J1), 2 I/Os are present on the FMod-IPECMOT 48/10 T2 (connector J5). The voltage range is [0;5V] for inputs and outputs.

Refer to registers description *IOCFG* and *IOSTATE* to have more informations on the usage of the I/Os.

## LED state

---

- Green: PWM duty cycle [0-75%]
- Yellow: PWM duty cycle [75-99%]
- Red: PWM at 100% (possible current limitation)



## Java Applet

---

*The Java Applet stays accessible to allow an handy access to parameters, though users should use the configuration software: FSoft-MotorCtrl. Refer to the chapter 9 to have more information.*

*Note also that v4.0 of the applet is described here.*

A specific Java Applet is provided with this device in order to control the parameters without having to write any specific software.

When you use a browser to load the HTTP web page loaded on the device, a Java Applet (included with the on-board web page) begins a communication between the device and the browser software (PC), and periodically refreshes all the useful registers (read and write sequences).

## Overview

---

To connect to the http server included on the device, simply open your web browser (not working with Google Chrome explorer) and type the IP address of the device. Here is an example with default address:

***“http:// 169.254.5.5”***

The applet is downloaded from the device to your computer and runs as a local process. You need to use a web browser with an up to date version of Java (1.8 minimum).

Please note that on a MS Windows<sup>®</sup>-based computer, a delay of several seconds can occur when you download the Java Applet due to an OS NetBios issue.

## State panel

The state panel shows the status of the main information related to the device. The values of the registers are displayed.

The screenshot displays the 'State panel' software interface. At the top, there are two red buttons: 'STOP !' and 'FREE !'. The version number 'Version 4.0 (17.06.2016)' is shown in the top right corner. Below these are two tabs: 'Device State' (selected) and 'Communication parameters'. The 'Device State' tab contains a list of parameters:

Type / Version:	FMod-IPECMOT 48/10 T1	HW1.8 / FW5.8
Regulation type:	Driver Stop	
Voltage:	42.227 V	
Temperature:	32.0 °C	
Input value:	0	
Position:	0	
Speed:	0	
Current limitation:	5.000 A	
Dissipation voltage:	NA	
IO State (1/2):	NA	
Current measured:	NA	

To the right of these parameters is a 'WARNINGS' panel. It has a title bar 'WARNINGS' and a sub-header 'ACTUAL BEFORE'. The list of warnings includes:

- Enable not act.
- Under voltage
- Over voltage
- Input not reached
- Output saturated
- Homing / no home
- Over current, stop integrator
- Limit 1 (home)
- Limit 2
- Over temperature
- Peak current
- PWR dissipation
- Output error

Below the warnings list is a 'Clear' button. At the bottom of the interface, there is a section for 'Change a register value:' with a dropdown menu showing '0x08 WARNING', a text input field containing '00000000', and 'Set' and 'Save' buttons. The status bar at the very bottom shows: 'TCP com state: Connected', 'MAC:00-50-C2-30-83-81', 'Name: IPECMOT 48/10', '77 pwr up', '6870435 s', and '© FiveCo Innovative Engineering 2002-2016'.

The upper part displays current state of the system with the Warning register bit at the right. The Clear button permits to clear warning bits fast.

The lower part allows access to all writable registers of the device. When you choose a register in the list, its current value is displayed in the white text area. You can change this value and use the Set button to update the register content. The Save button provoke the backup of the current registers state to User EEPROM memory.

## Main parameters panel

---

This panel is used to change the device's main parameters:

- **IP address:** the new one will be valid only when all users are disconnected, including this applet. You have so to close the browser to apply the change. Then, you will have to open a new browser with the new IP address in the address bar to access the applet again!
- **Subnet mask:** useful only for special UDP directed broadcast.

The screenshot shows a web interface for configuring a device. At the top, there are two red buttons labeled "STOP !" and "FREE !", and the text "Version 4.0 (17.06.2016)". Below this, there are two tabs: "Device State" and "Communication parameters". The "Communication parameters" tab is active, showing two rows of input fields. The first row is for "IP Address:" with four fields containing the values 192, 168, 16, and 136. The second row is for "Subnet mask:" with four fields containing the values 255, 255, 255, and 0. Below the input fields, there is a red warning message: "Warning: Use values within your computer subnet! Close this page to apply changes." Underneath the warning is a blue button labeled "Change and save". At the bottom of the interface, there is a status bar with the following information: "TCP com state: Connected", "MAC:00-50-C2-30-83-81", "Name: IPECMOT 48/10", "77 pwr up", and "6871166 s". At the very bottom, there is a copyright notice: "© FiveCo Innovative Engineering 2002-2016".

Device State	Communication parameters
IP Address:	192 168 16 136
Subnet mask:	255 255 255 0

Warning: Use values within your computer subnet! Close this page to apply changes.

Change and save

TCP com state: Connected    MAC:00-50-C2-30-83-81    Name: IPECMOT 48/10    77 pwr up    6871166 s

© FiveCo Innovative Engineering 2002-2016

### 3. FMod-I2C485ECMOT DB 48/10

#### Operating conditions

▪ Operating temperature	-20...+85°C
▪ Supply voltage Vcc	15-48 VDC
▪ Supply current	max ±10 A
▪ Power consumption	78mA when idle @ Vcc 15 V 52mA when idle @ Vcc 24 V 44mA when idle @ Vcc 48 V
▪ Input capacity (Power +,-)	~2000uF, ~15mOhm ESR
▪ Limit-switch voltage supply	5.0 V (max 2 x 50 mA )
▪ Max A&B encoder signal	250 kHz = 1Mio pulses/s (CPRx4)
▪ Output voltage	92% Vcc @ 35kHz PWM 85% Vcc @ 69kHz PWM
▪ Max. output current	15 A
▪ Continuous <b>OUTPUT</b> current (Pwr Mosfet thermal limitation)	Brushed Motor 10A (Tamb. 25°C) Brushless Motor 8A (Tamb. 25°C)

#### Overview

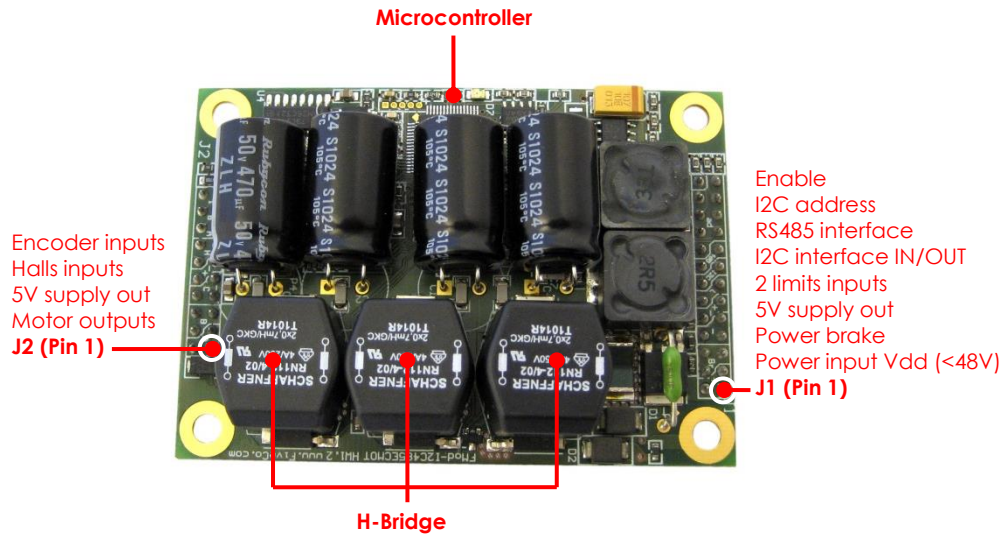
##### Applications

The FMod-I2C485ECMOT DB 48/10 is a motion control and driver device for DC motors, either with brushes or brushless (EC) with hall sensors. It is particularly interesting because of its small size and quality of the regulation system.

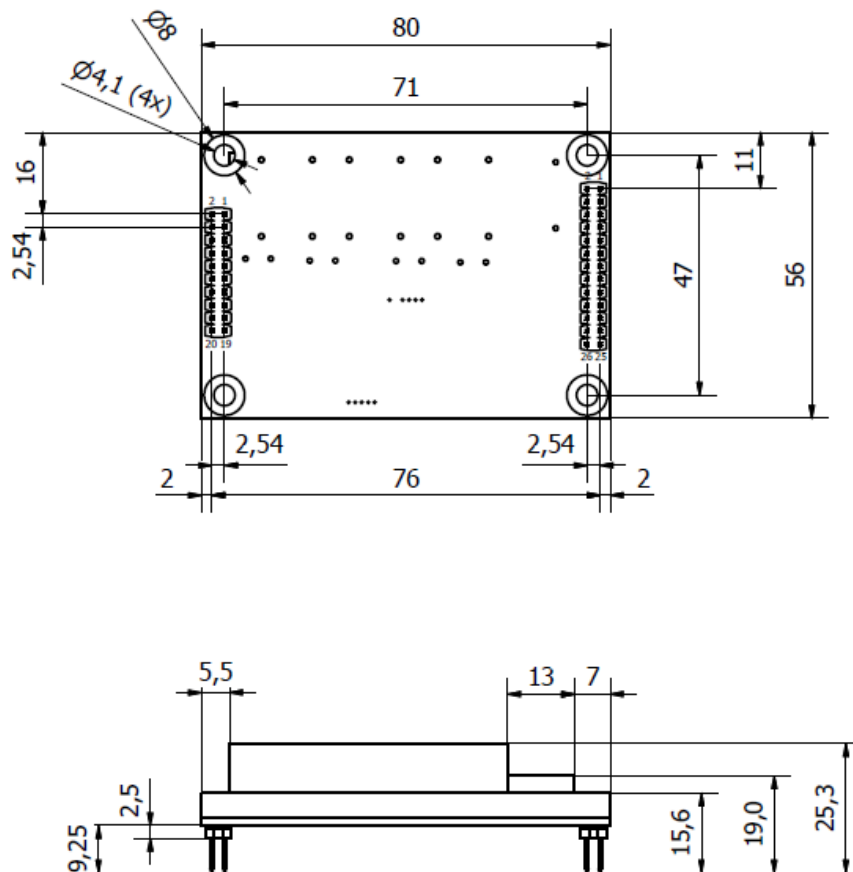
It can be accessed through an I<sup>2</sup>C or an RS485 connection. Access through I<sup>2</sup>C can be made by the user or with the help of another FiveCo module, FMod-TCP DB or FMod-TCP Box 2 which are bridges between TCP/IP and I<sup>2</sup>C. Access through RS485 is easily made with a serial adapter USB-RS485 connected to the computer or from a motherboard. Refer to the chapter "*I<sup>2</sup>C Interface*" and "*RS485 Interface*" to have more information on how to communicate with the board.

The device connections and dimensions are described on the following pages.

Hardware description



Physical Dimensions [mm]



**Connector J1 pinning**

The Pad 0 near pin 1 of J1 should be connected to the shield

- 1 Power + (in)
- 2 Power + (in)
- 3 Power + (in)
- 4 Power Brake (out)
- 5 Power Brake (out)
- 6 Power Brake (out)
- 7 Power – (in)
- 8 Power – (in)
- 9 Power – (in)
- 10 Logic GND (in)
- 11 Logic +5v (out)
- 12 Limit 1 or home (in)
- 13 Limit 2 (in)
- 14 SDA (I2C data, in/out)
- 15 SCL (I2C clock, in/out)
- 16 RS485+ (in/out)
- 17 RS485- (in/out)
- 18 I2C/RS485 Address selection bit 0 (in)
- 19 Address bit 1 (in)
- 20 Address bit 2 (in)
- 21 Address bit 3 (in)
- 22 Address bit 4 (in)
- 23 Address bit 5 (in)
- 24 Address bit 6 (in) or IO 1 (in/out) see **OPTIONS.13**
- 25 RS485 Address bit 7 (in) or IO 2 (in/out)
- 26 Enable (in)

## Notes:

- The device's logic power supply is internally generated by the card from the DC supply. It can be used externally to power limits, hall sensors and encoder to a maximum of 300mA in total on connectors J1 and J2. The DC power supply (15-48v) for the motor has to be connected to pins 1,2,3 and 7,8,9.
- When the dissipation is activated, the 3 Power Brake pins (4,5 and 6) are tied to GND through a transistor. Therefore if a power dissipating element (e.g. a power resistor) is connected between Power + and Power Brake pins, it dissipates the extra energy.

**Connector J2 pinning**

- 1 EC1 (out)
- 2 EC1 (out)
- 3 EC1 (out)
- 4 EC2 / DC- (out)
- 5 EC2 / DC- (out)
- 6 EC2 / DC- (out)
- 7 EC3 / DC+ (out)
- 8 EC3 / DC+ (out)
- 9 EC3 / DC+ (out)
- 10 Logic +5v (out)
- 11 Logic GND (in)
- 12 Hall 1 (in)
- 13 Hall 2 (in)
- 14 Hall 3 (in)
- 15 Encoder channel A inverted (in)
- 16 Encoder channel A (in)
- 17 Encoder channel B inverted (in)
- 18 Encoder channel B (in)
- 19 Encoder Index inverted (in)
- 20 Encoder Index (in)

**Notes:**

- The device's logic power supply is internally generated by the card from the DC supply. It can be used externally to power limits, hall sensors and encoder to a maximum of 300mA in total on connectors J1 and J2.

## Hardware

---

### Power supply

---

[15-48V] with the peak current equal to the nominal current of the motor on connector J1 (min 1A). E.g. for a motor with a constant of 24V and 2A, choose a power supply of 24V 2A. Don't worry about the motor start current you will see on the datasheet, it is electronically limited by the device.

**WARNING: When braking (deceleration), the 4Q device regenerates the inertial energy to electricity, so the voltage between Power+ and Power- pins could be higher than the power supply. Dissipation of the braking energy can be made through the "Power brake" pins. Refer to the next chapter "Dissipation" to have more information on this feature.**

If no power dissipating element is connected between "Power Brake" and "Power+", the device will accept overrides if below 50V, or if between 51-53V inside the clamping diode, it will temporarily accept 3-5A.

If your system does not accept overrides, you need to add a dissipating element between Power Brake and Power+ pins.

**The power supply has to be able to manage reverse power when braking (power generation) on connector J1 Power+ and Power-.**

- > Below 12.0V, the driver is automatically set to Brake mode.
- > Over 56V, the driver is automatically set to DriverOpen mode.

### Dissipation

---

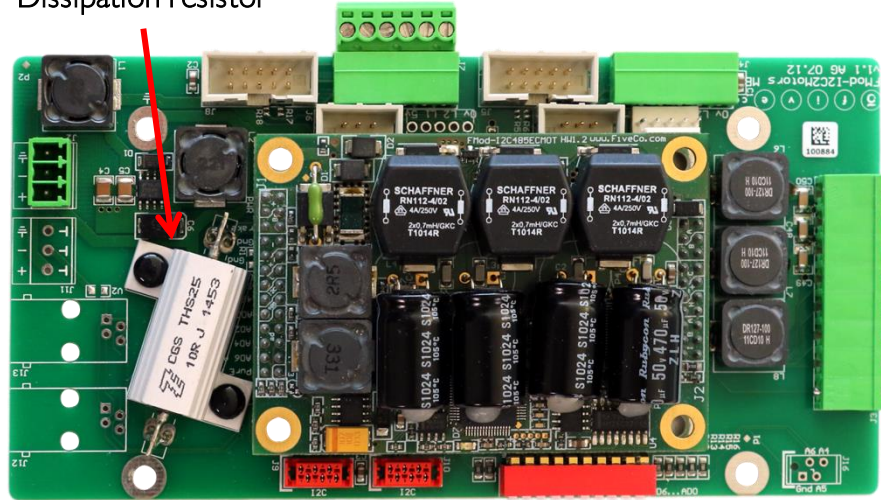
When the dissipation is activated, the 3 Power Brake pins (4,5 and 6) are tied to GND through a transistor. Therefore if a power dissipating element (e.g. a power resistor) is connected between Power + and Power Brake pins, it dissipates the extra energy.

The dissipation is obviously dependent on the resistor value, if the user wants to dissipate 2 A at 24V, the value of the power resistor should be 12 ohms.

In the figure below, a setup with an FMod-I2C485ECMOT DB 48/10 with its dissipation resistor connected to the PowerBrake pins.

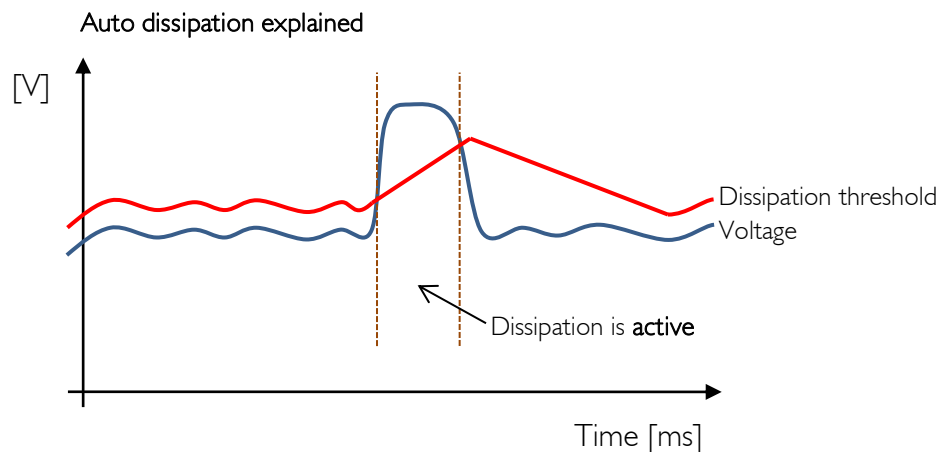


Dissipation resistor



The dissipation is activated when the voltage measured by the controller exceeds a certain threshold. This threshold could be set by the user with the register **DISSIPATIONVOLTAGE** (0x2F), or use the automatic feature setting the **OPTIONS.16** bit (0x2C).

When the automatic threshold is used, the dissipation voltage threshold stabilizes 3 V above the actual average of the measured voltage. When a too abrupt rise on the measured voltage occurs, the dissipation is activated while the threshold is still updated with the current average. The following figure shows this mechanism.



#### Warning:

If a fixed dissipation voltage is used, it will not stop dissipating until the measured voltage goes below the specified threshold.

See register description **OPTIONS** (0x2C) and **DISSIPATIONVOLTAGE** (0x2F) for more information.

## Motor type

---

The device drives DC brushed motors as well as brushless motors with hall sensors, (0.1-10A, 3-48V).

FMod-I2C485ECMOT DB 48/10 automatically detects whether the motor is brushed or brushless.

If speed control or positioning is needed, use A&B (&Index) channel quadrature incremental encoders with a differential line driver (EIA/TIA/RS 422). The controller is also compliant with non-differential encoder lines. Each logical change of A or B channels will define a pulse.

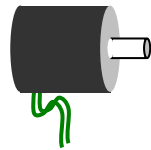
If the encoder is defined in cycles (or counts) per revolution (CPR), you can multiply the cycles by 4 to obtain the result in pulses (PPR).

For example: an encoder of 500 CPR represents 2000 PPR in the controller.

With brushless motors only, it is possible to configure the device to use the hall sensors (6 states/cycle) for speed control and positioning, but the resolution is poor.

### Brushed motor (DC)

---

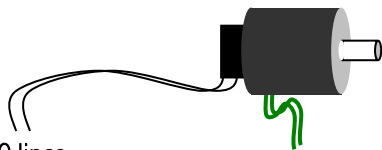


2 lines  
Power motor lines +, -

Open Loop only, no regulation because no feedback!

### Brushed motor (DC) with encoder

---



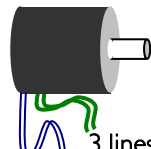
10 lines  
Quadrature encoder  
with differential line  
driver  
A<sub>v</sub>/A<sub>b</sub>/B<sub>v</sub>/B<sub>b</sub>(I,I), +5V, 0V  
Also compliant with  
non-differential encoder  
lines

2 lines  
Power motor lines +, -

Full regulation: speed or positioning.

### Brushless motor (EC)

---



3 lines  
Power motor lines  
phases 1,2,3

5 lines  
Hall sensor lines  
hall 1,2,3 , +5V,0V

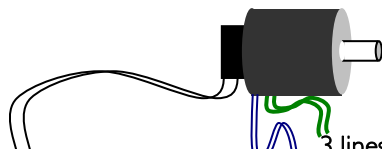
Full regulation: speed or positioning with hall sensors as encoders.

Poor regulation due to the few states of hall sensors.

(see **OPTIONS** register (0x2C) to set this feature)

### Brushless motor (EC) with encoder

---



10 lines  
Quadrature encoder  
with differential line  
driver  
A<sub>v</sub>/A<sub>b</sub>/B<sub>v</sub>/B<sub>b</sub>(I,I), +5V, 0V  
Also compliant with  
non-differential encoder  
lines

3 lines  
Power motor lines  
phases 1,2,3

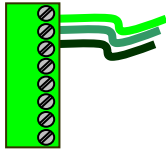
5 lines  
Hall sensors  
hall 1,2,3 , +5V,0V

Full regulation: speed or positioning

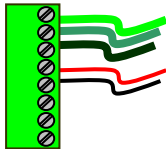
## How to connect hall sensors

Different motor manufacturers do not have the same enumeration for hall sensors 1,2,3 and motor phases (windings) 1,2,3.

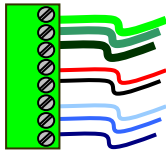
Here is a method to find the corresponding connection between the brushless motor and the device.



Connect phases (windings) 1,2,3 (U,V,W=A,B,C) to the FMod-I2C485ECMOT DB 48/10 motor connector J2, through pins EC1, EC2, EC3.



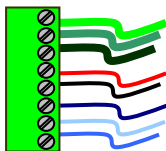
Connect the +5V and GND hall sensors to the J2 connector (Logic +5V and Logic GND pins).



Connect the hall sensor lines 1,2,3 to the J2 connector.

Power the device, connect through RS485 or I<sup>2</sup>C to the board, set the Enable pin (connector J1) to high, set the **REGULATIONMODE** register to "Open Loop", set **INPUT** to 48'000 (~3/4 full PWM).

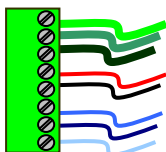
If the motor turns correctly, you have done it!



If the motor does not turn, set **REGULATIONMODE** to "Driver Open", and switch the hall sensor lines: Hall1 → Hall2, Hall2 → Hall3, Hall3 → Hall1.

Set **REGULATIONMODE** to "Open Loop", set **INPUT** to 48'000.

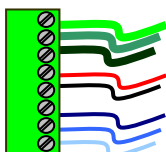
If the motor turns correctly, you have done it!



If the motor does not turn, do the same manipulation one more time: switch the hall sensor lines: Hall1 → Hall2, Hall2 → Hall3, Hall3 → Hall1 ...

Set **REGULATIONMODE** to "Open Loop", set **INPUT** to 48'000.

If the motor turns correctly, you have done it!



If it still isn't working, unlucky!

Swap lines Hall 1 ↔ Hall 2

Test it and if it still doesn't work, switch the hall sensor lines: Hall1 → Hall2, Hall2 → Hall3, Hall3 → Hall 1, test again, and switch one more time.

If it still doesn't work, the motor or device is most likely damaged.

## Enable pin

---

This feature is for security purposes and stops the output power (connector J1).

The voltage ranging between [4.2V; 48V] on the **Enable pin** allows the driver to work normally (regulation on motor).

If the voltage on the **Enable pin** drops below 4V, the driver (motor) is set to "Brake" mode.

When the **Enable pin** goes back to high [4.2V; 48V], the user needs to change the mode from "Brake" back to the required one.

## Limit Switch: stop, reference, stopper type

---

Both Limit1 and Limit2 are inputs lines with a Schmitt trigger (logic low : [0-1V], logic high [4-5V]).

5 VDC sensors can be powered from the connector J1, pins Logic +5v and Logic GND. Open collector (NPN,PNP) and open drain (N,P) output stage sensors can be used.

Micro switches or reed sensors can also be used, but take a special care to the rebounds of the limit signal (typ <10ms).

**WARNING:** Both Limit Switch sensors (L1 & L2) must be of the same type (NPN,N or PNP,P) because of the internal pull-up or pull-down resistor selection.

## I2C/RS485 address selection and I/Os

---

The I2C/RS485 address selection is made by hardware connection on connector J1; 7 bits for I2C address and 8 bits for RS485 address. Each line needs to be connected to +5v logic (1) or logic ground (0). Do not leave any pin of address floating, except when I/Os are used on the 2 pins I2C/RS485 Address selection [6-7] (J1, pin 24-25).

The range for I2C address is [0x08;0x77] in hexadecimal or [8;120] in decimal. If the address set in hardware is out of this range, the default address is 0x55 (85).

The range for RS485 address is [0x00;0xFF] in hexadecimal or [0;255] in decimal.

As example, if you want to set one FMod-I2C485ECMOT DB 48/10 to address 40 (0x28), convert 40 (decimal) in binary code (b'00101000'); this bits are the values to be set to the corresponding pins of J1 "I2C/RS485 Address selection →7-0".

The I2C and RS485 addresses can be different if the hardware address is out of the I2C address range. For example if the address is set to 202 (0xCA), the I2C address will be 85 (0x55 default address) and the RS485 address will be 202 (0xCA).

When the **OPTIONS.13** bit is set, the 2 most significant bits are used as I/Os (I2C/RS485 Address selection [7;6]). If the I/Os are used, the range for I2C address is [0x08;0x3F] in hexadecimal or [8;63] in decimal. The range for RS485 address in that situation is [0x00;0x3F] in hexadecimal or [0;63] in decimal.

The voltage range is [0;5V] for inputs and outputs. Refer to registers description **OPTIONS** (bit 13), **IOCFG** and **IOSTATE** to have more informations on the usage of the I/Os.

## 4. FMod-I2CDCMOT DB 48/1.5 & SLP 48/1

### Operating conditions

#### FMod-I2CDCMOT DB 48/1.5

▪ Operating temperature	-20...+85°C
▪ Supply voltage Vcc	10-48 VDC
▪ Supply current	max ±1.5 A
▪ Logic consumption supply	max 50mA @ 5V logic 20mA when idle @ 5V logic
▪ Input capacity (Power +,-)	~220uF, ~50mOhm ESR
▪ Max A&B encoder signal	250 kHz = 1Mio pulses/s (CPRx4)
▪ Output voltage	Vcc - 0.7 × Output current
▪ Max. output current	2 A
▪ Continuous <b>OUTPUT</b> current (Pwr Mosfet thermal limitation)	1.5 A (-20...+60°C) 1.3 A (+60°C...+85°C)

#### FMod-I2CDCMOT SLP 48/1

▪ Operating temperature	-20...+85°C
▪ Supply voltage Vcc	10-48 VDC
▪ Supply current	max ±1 A
▪ Logic consumption supply	max 50mA @ 5V logic <1 µA when standby @ 5V logic <b>(50nA at 25°C)</b>
▪ Input capacity (Power +,-)	~15uF, ~10mOhm ESR
▪ Max A&B encoder signal	500 kHz = 2Mio pulses/s (CPRx4)
▪ Output voltage	Vcc - 0.7 × Output current
▪ Max. output current	2 A
▪ Continuous <b>OUTPUT</b> current (Pwr Mosfet thermal limitation)	1.0 A (-20...+85°C)

## Overview

---

### Applications

---

The FMod-I2CDCMOT DB 48/1.5 and FMod-I2CDCMOT SLP 48/1 are motion control and driver boards for DC motors (with brushes). They are particularly interesting because of their small size, the quality of its regulation and the power that can be delivered by this card; 70W continuous for the DB version and 48W continuous for the SLP version. With its world known I2C data-bus communication device, more than 100 I2C devices can be connected to the same bus. Access through I<sup>2</sup>C can be made by the user or with the help of another FiveCo module, FMod-TCP DB or FMod-TCP Box 2 which are bridges between TCP/IP and I<sup>2</sup>C. Refer to the chapter "**I2C Interface**" to have more information on how to communicate with the board.

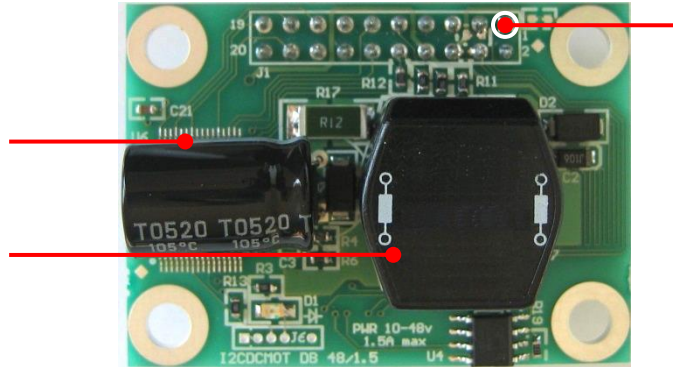
The daughter board version (FMod-I2CDCMOT DB 48/1.5), dedicated to production, can be easily plugged on a motherboard without any cable, just with its 20 pins (2.54mm space) connector.

The SLP version ("SLP" stands for Soldered Low Power) can be easily soldered to a motherboard without any cables through its 22+2 plated holes on board edge (1.27mm spacing). Another way is to solder a 1.27 mm spacing male connector to the board, making it easily pluggable to a dedicated motherboard.

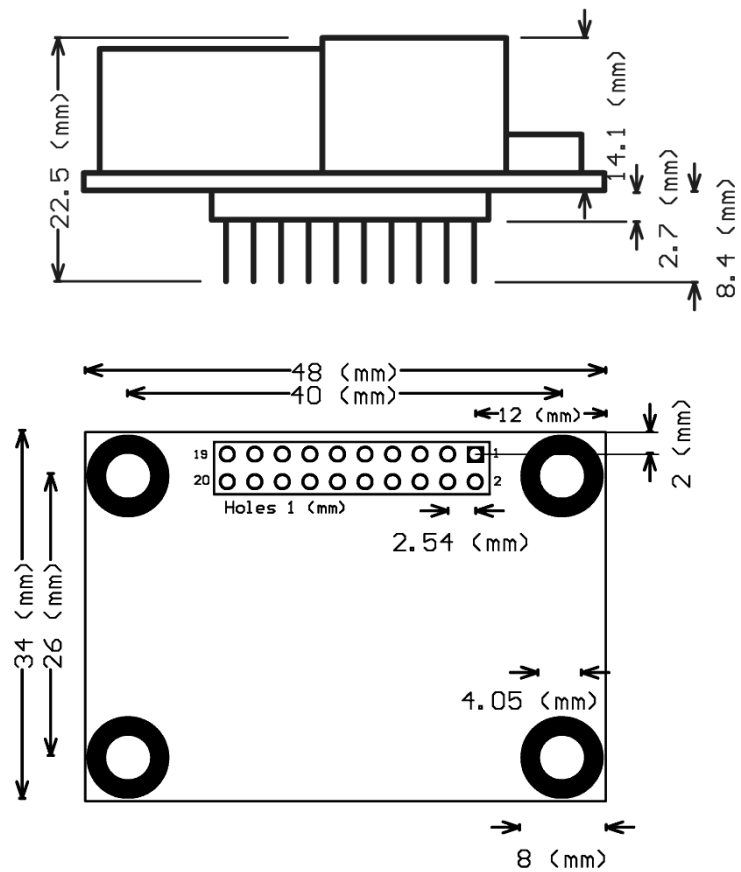
Compared to the FMod-I2CDCMOT DB 48/1.5, the FMod-I2CDCMOT SLP 48/1 offers a smaller power output (48W) but adds the low power mode, 1μA maximum current consumption on logic 5V and power supply. At ambient temperature, this value drops as low as 50nA on each supply.



## Hardware description of FMod-I2CDCMOT DB 48/1.5



### Physical Dimensions [mm]



**Connector J1 pinning**

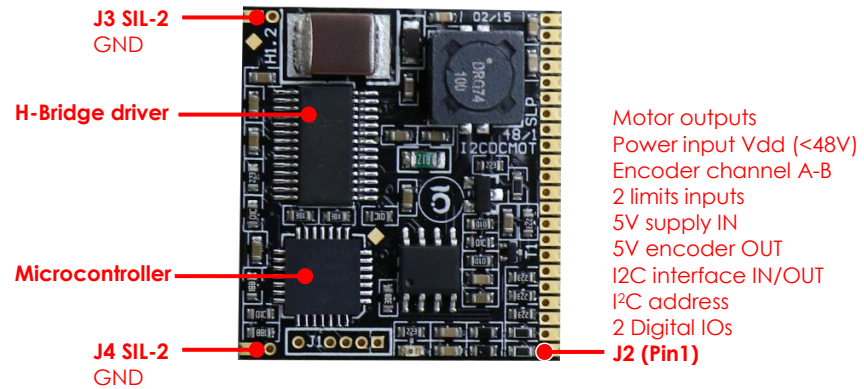
The Pad 0 near pin 1 of J1 should be connected to the shield

- 1 Motor + (out)
- 2 Motor – (out)
- 3 Encoder channel A (in)
- 4 Encoder channel B (in)
- 5 Power + (in)
- 6 Power – (in)
- 7 Logic +5v (in)
- 8 Logic GND (in)
- 9 Limit 1 or home (in)
- 10 Limit 2 (in)
- 11 SDA (I2C data, in/out)
- 12 SCL (I2C clock, in/out)
- 13 I2C Address selection bit 0 (in)
- 14 Address bit 1 (in)
- 15 Address bit 2 (in)
- 16 Address bit 3 (in)
- 17 Address bit 4 (in)
- 18 Address bit 5 (in) or IO 1 (in/out) see **OPTIONS.13**
- 19 Address bit 6 (in) or IO 2 (in/out)
- 20 #Reset logic (in or floating), reset is active low, internally pull-up

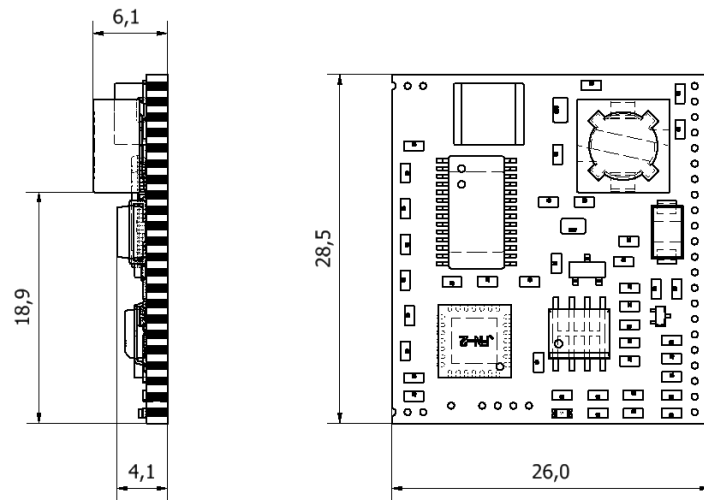
**Notes:**

- The device's logic power supply (electronics and processor) is on pin 7 (Logic +5V) and 8 (Logic GND) and must be applied externally. A second power supply (10-48v) for the motor has to be connected to pins 5 and 6.

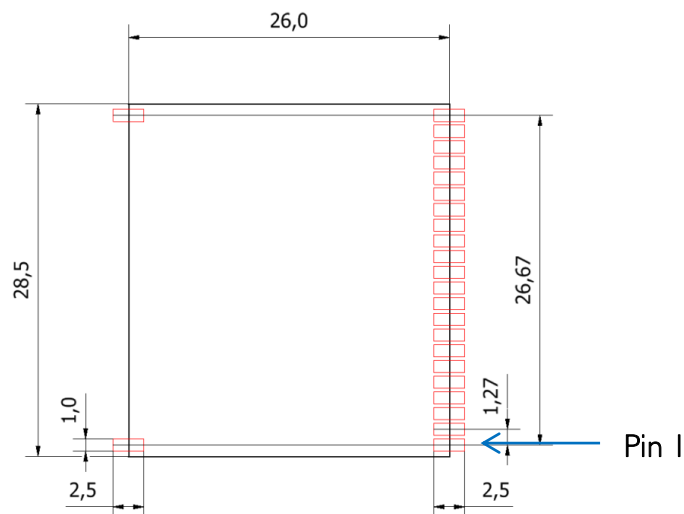
Hardware description of FMod-I2CDCMOT SLP 48/I



Physical Dimensions [mm]



Footprint



In red are the recommended pads footprints on the motherboard.

### Connector J2 pinning

- 1 Digital IO 1 (in/out)
- 2 Digital IO 2 (in/out)
- 3 I2C Address selection bit 0 (in)
- 4 I2C Address selection bit 1 (in)
- 5 I2C Address selection bit 2 (in)
- 6 I2C Address selection bit 3 (in)
- 7 I2C Address selection bit 4 (in)
- 8 I2C Address selection bit 5 (in)
- 9 I2C Address selection bit 6 (in)
- 10 SDA (I2C data, in/out)
- 11 SCL (I2C clock, in/out)
- 12 Encoder +5V (out)
- 13 Logic +5V (in)
- 14 Logic GND (in)
- 15 Limit 2 (in)
- 16 Limit 1 or home (in)
- 17 Encoder channel B (in)
- 18 Encoder channel A (in)
- 19 Power – (in)
- 20 Power + (in)
- 21 Motor – (out)
- 22 Motor + (out)

#### Notes:

- The device's logic power supply (electronics and processor) is on pin 13 (Logic +5V) and 14 (Logic GND) and must be applied externally. A second power supply (10-48v) for the motor has to be connected to pins 20 and 19.
- The encoder's power supply is an output of the board and must be connected to pins 12 (Encoder +5V) and 14 (Logic GND). It is an output of the board since it is disabled in ultra-low power mode; therefore it must not be tied with the general Logic +5V.

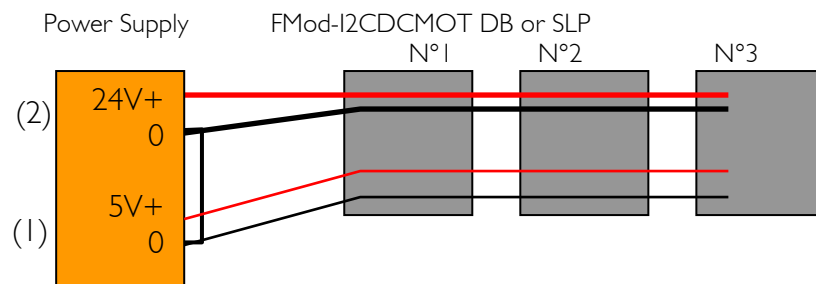
## Hardware

### Power supply

1. VCC (+5V) used for the electronics and processor must be applied on the Logic 5V and Logic GND pins. (See details on chapter "Connector J2 pinning", under their respective Hardware descriptions).
2. 10-48 V with the peak current equal to the nominal current of the motor on the power inputs.

E.g. for a motor with a constant of 24V and 1A, choose a power supply of 24V 1A. Don't worry about the motor start current you will see on the datasheet, it is electronically limited by the device.

!!! Both power supplies must be connected to the same ground. !!!



**WARNING:** When braking (deceleration), the 4Q device regenerates the inertial energy to electricity, so the voltage between Power+ and Power- pins could be higher than the power supply. The device will accept overrides if below 50V, or if between 51-53V inside the clamping diode, it will temporarily accept 3-5A. If your system does not accept overrides, you need to add an external dissipation system.

Therefore the power supply must manage backward power when braking (power generation), e.g. over voltage protection.

Under 9.0 V driver is automatically put in Brake-Mode.

Over 54 V driver is automatically put in DriverOpen-Mode.

## Motor type

---

DC brushed motors (current 0.1-1.5A for DB version, current 0.1-1A for SLP version, voltage 10-48V).

With A&B channel quadrature encoders. If the encoder has differential lines, use only the non-inverted outputs A and B.

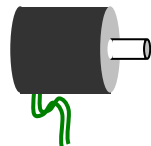
Each logic change of A or B channel defines a pulse.

If the encoder is defined in cycles (or counts) per revolution (CPR), you can multiply by 4 the cycles to obtain the result in pulses (PPR).

Example: with an encoder of 500 CPR, it represents 2000 PPR in the controller.

## Brushed motor (DC)

---

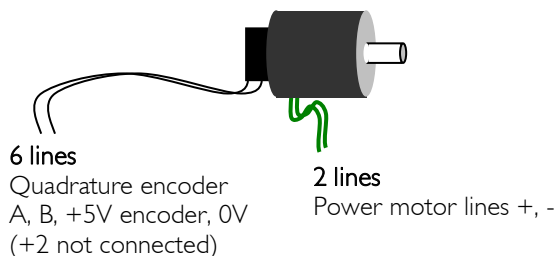


2 lines  
Power motor lines +, -

Open Loop only, no regulation because no feedback!

## Brushed motor (DC) with encoder

---



6 lines  
Quadrature encoder  
A, B, +5V encoder, 0V  
(+2 not connected)

2 lines  
Power motor lines +, -

Full regulation: speed or positioning.

## Limit-Switch: stop, reference, stopper type

---

Both Limit1 and Limit2 are inputs lines with a Schmitt trigger (logic low : [0-1V], logic high [4-5V]).

5 VDC sensors must be used; for example, open collector (NPN,PNP) and open drain (N,P) output stage sensors.

Micro switches or reed sensors can also be used, but take a special care to the rebounds of the limit signal (typ < 10ms).

### WARNING:

Both Limit-switch sensors (L1 & L2) must be of the same type (NPN,N or PNP,P), because of the internal pull-up or pull-down resistor selection.

## I2C address selection and I/Os

---

The I2C address selection is made by hardware connection on connector J1 for DB version and J2 for SLP version; 7 bits for I2C address. Each line needs to be connected to +5v logic (1) or logic ground (0). Do not leave any pin of address floating, except when I/Os are used on the 2 pins I2C Address selection [5-6] (only for FMod-I2CDCMOT DB 48/1.5).

The range for I2C address is [0x08;0x77] in hexadecimal or [8;120] in decimal. If the address set in hardware is out of this range, the default address is 0x55 (85).

As example, if you want to set one FMod-I2CDCMOT DB 48/1.5 to address 40 (0x28), convert 40 (decimal) in binary code (b'00101000'); the 7 least significant bits (→0101000) are the values to be set to the corresponding pins of J1 (DB) or J2 (SLP) "I2C Address selection →6-0".

Only for FMod-I2CDCMOT DB 48/1.5, when the **OPTIONS.13** bit is set, the 2 most significant bits of the address are used as I/Os (I2C Address selection [6;5]). If the I/Os are used, the range for I2C address is [0x08;0x1F] in hexadecimal or [8;31] in decimal.

The voltage range is [0;5V] for inputs and outputs. Refer to registers description **OPTIONS** (bit 13), **IOCFG** and **IOSTATE** to have more informations on the usage of the I/Os.

## 5. FMod-I2CSTEPMOT SLP 35/I & 35/0. I

### Operating conditions

#### FMod-I2CSTEPMOT SLP 35/I

▪ Operating temperature	-20...+85°C
▪ Supply voltage Vcc	9-35 VDC
▪ Supply current	max ±1 A
▪ Logic consumption supply	max 50mA @ 5V logic <1 µA when standby @ 5V logic <b>(50nA at 25°C)</b>
▪ Input capacity (Power +,-)	~15µF, ~10mOhm ESR
▪ Output voltage	Vcc
▪ Max. output current	1.5 A
▪ Continuous <b>OUTPUT</b> current (Pwr Mosfet thermal limitation)	1.0 A (-20...+85°C)

#### FMod-I2CSTEPMOT SLP 35/0. I

▪ Operating temperature	-20...+85°C
▪ Supply voltage Vcc	9-35 VDC
▪ Supply current	max ±150 mA
▪ Logic consumption supply	max 50mA @ 5V logic <1 µA when standby @ 5V logic <b>(50nA at 25°C)</b>
▪ Input capacity (Power +,-)	~15µF, ~10mOhm ESR
▪ Output voltage	Vcc
▪ Max. output current	150 mA
▪ Continuous <b>OUTPUT</b> current (Pwr Mosfet thermal limitation)	150 mA (-20...+85°C)



## Overview

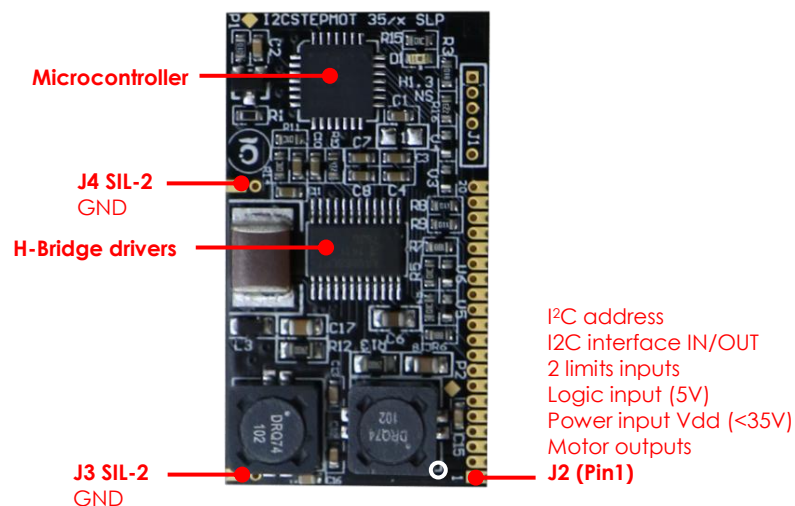
### Applications

The FMod-I2CSTEPMOT SLP 35/I & 35/O.I are motion control and driver boards for 2 phases stepper motors. They are low cost and very small control device with 1/4 step motion control for position and speed control using trapezoidal trajectory profile. Ultra low power consumption in stand-by mode, typically less than 1  $\mu$ A makes them ideal for portable and compact applications.

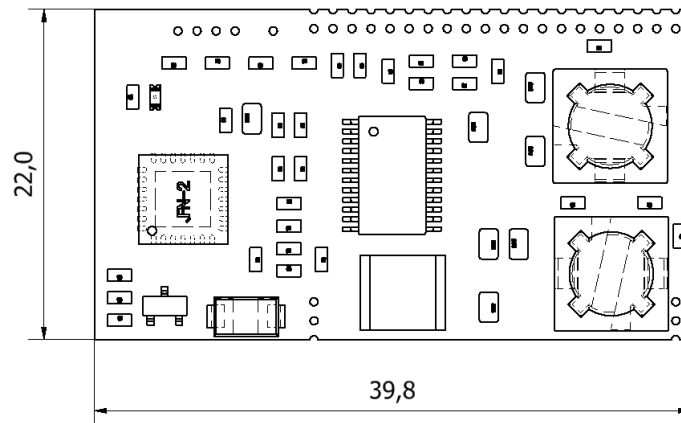
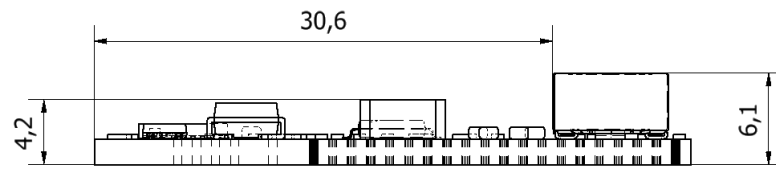
With its world known I2C data-bus communication device, more than 100 I2C devices can be connected to the same bus. Access through I<sup>2</sup>C can be made by the user or with the help of another FiveCo module, FMod-TCP DB or FMod-TCP Box 2 which are bridges between TCP/IP and I<sup>2</sup>C. Refer to the chapter "**I2C Interface**" to have more information on how to communicate with the board.

The boards are dedicated to production; they can be easily soldered to a motherboard without any cables through its 20+2 plated holes on board edge (1.27mm spacing). Another way is to solder a 1.27 mm spacing male connector to the board, making it easily pluggable to a dedicated motherboard.

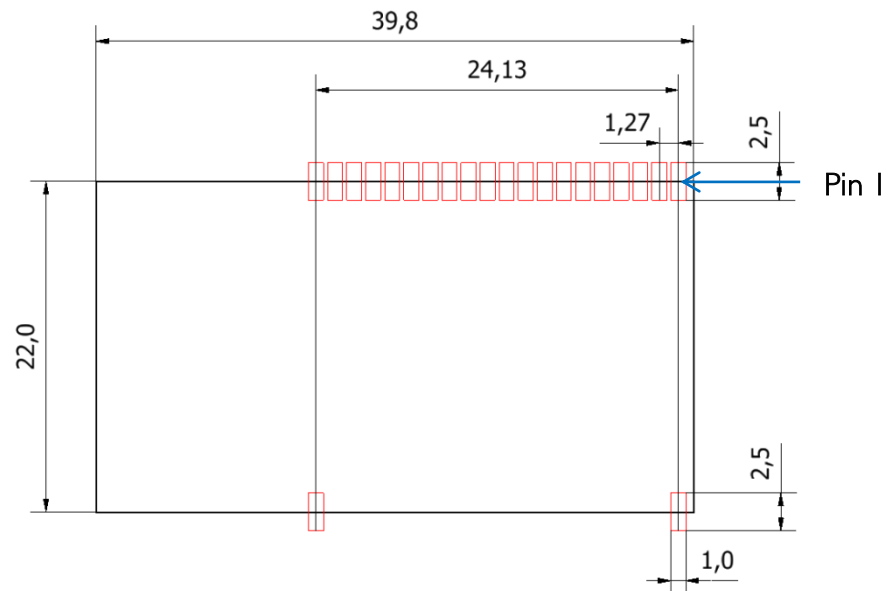
### Hardware description of FMod-I2CSTEPMOT SLP 35/I & 35/O.I



## Physical Dimensions [mm]



## Footprint



In red are the recommended pads footprint on the motherboard.

**Connector J2 pinning**

- 1 Motor phase 1 + (out)
- 2 Motor phase 1 – (out)
- 3 Motor phase 2 – (out)
- 4 Motor phase 2 + (out)
- 5 Power + (in)
- 6 Power – (in)
- 7 Logic +5V (in)
- 8 Logic GND (in)
- 9 Limit 1 or home (in)
- 10 Limit 2 (in)
- 11 SDA (I2C data, in/out)
- 12 SCL (I2C clock, in/out)
- 13 I2C Address selection bit 0 (in)
- 14 I2C Address selection bit 1 (in)
- 15 I2C Address selection bit 2 (in)
- 16 I2C Address selection bit 3 (in)
- 17 I2C Address selection bit 4 (in)
- 18 I2C Address selection bit 5 (in)
- 19 I2C Address selection bit 6 (in)
- 20 #Reset logic (in or floating), reset is active low, internally pull-up

## Notes:

- The device's logic power supply (electronics and processor) is on pin 7 (Logic +5V) and 8 (Logic GND) and must be applied externally. A second power supply (10-35V) for the motor has to be connected to pins 5 and 6.

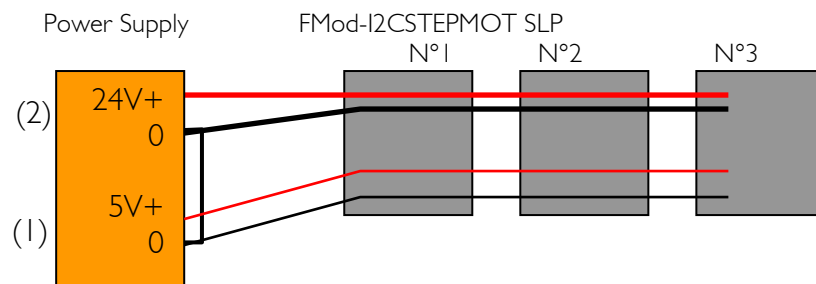
## Hardware

### Power supply

1. VCC (+5V) used for the electronics and processor must be applied on the Logic 5V and Logic GND pins. (See details on chapter "Connector J2 pinning", under the Hardware description).
2. 10-35 V with the peak current equal to the nominal current of the motor on the power inputs.

E.g. for a motor with a constant of 24V and 1A, choose a power supply of 24V 1A. Although a higher power supply current limit is not a problem. Don't worry about the motor start current you will see on the datasheet, it is electronically limited by the device.

!!! Both power supplies must be connected to the same ground. !!!



**WARNING:** When braking (deceleration), the 4Q device regenerates the inertial energy to electricity, so the voltage between Power+ and Power- pins could be higher than the power supply. The device will accept overrides if below 36V, or if between 36-38V inside the clamping diode, it will temporarily accept 3-5A. If your system does not accept overrides, you need to add an external dissipation system.

Therefore the power supply must manage backward power when braking (power generation), e.g. over voltage protection.

Under 9.0 V driver is automatically put in DriverOpen-Mode.

Over 38 V driver is automatically put in DriverOpen-Mode.

## Motor type

---

2 phases stepper motors:

- Nominal current [5mA-150mA] for 35/0.1 version
- Nominal current [0.1-1A] for 35/1 version,
- Nominal voltage [2V-35V]

Often the stepper motor manufacturers state that the nominal voltage of the motor is the nominal current \* phase resistance. This is true only if the stepper motor is driven directly by the power supply (referred as voltage driven). Here we use current chopper drive, therefore the nominal voltage does not have to be respected.

If an application requires low speed of the motor, better choose a high nominal voltage, current consumption will be lower.

Instead if an application requires high speed of the motor, better choose a low nominal voltage, but current consumption will be higher. With a lower nominal voltage, the counter-electromotive force is lower and more voltage is left for the motor torque at high speed.

## Limit-Switch: stop, reference, stopper type

---

Both Limit1 and Limit2 are inputs lines with a Schmitt trigger (logic low : [0-1V], logic high [4-5V]).

5 VDC sensors must be used; for example, open collector (NPN,PNP) and open drain (N,P) output stage sensors.

Micro switches or reed sensors can also be used, but take a special care to the rebounds of the limit signal (typ <10ms).

### WARNING:

Both Limit-switch sensors (L1 & L2) must be of the same type (NPN,N or PNP,P), because of the internal pull-up or pull-down resistor selection.

## I2C address selection

---

The I2C address selection is made by hardware connection on connector J2; 7 bits for I2C address. Each line needs to be connected to +5v logic (1) or logic ground (0). Do not leave any pin of address floating!

The range for I2C address is [0x08;0x77] in hexadecimal or [8;120] in decimal. If the address set in hardware is out of this range, the default address is 0x55 (85).

As example, if you want to set one FMod-I2CSTEPMOT SLP 35/1 to address 40 (0x28), convert 40 (decimal) in binary code (b'00101000'); the 7 least significant bits (→0101000) are the values to be set to the corresponding pins of J2 "I2C Address selection →6-0".

## Position and phases synchronisation

---

### Overview

---

With **OPTIONS.19** and **OPTIONS.20** bits (0x2C), the user can decide to synchronize the position with the actual phases of the stepper motor. This means that the user has the possibility to know exactly on which phases the motor is currently driven.

If the user wants to send its motor to positions that are magnetically stable, which means that only one phase has current flowing through its winding. You could see in documents from the motor manufacturer that the term “full-step” is used when both phases are activated, but in this document “full-step” refers to position that are magnetically stable as explained above. The advantage is that even if the card is powered down, the motor will not move until the holding torque of the motor is not countered.

To achieve this, the bit **OPTIONS.19** must be set to '1' which activates the synchronisation between the **POSITION** (0x26) register and the phases. Then the synchronisation can be made on 1 full-step or 4 full-steps (the 4 full-steps synchronisation will be explained later); setting the bit **OPTIONS.20** to '1' => 1 full-step synchronisation, setting the bit **OPTIONS.20** to '0' => 4 full-steps synchronisations. In this example, the 1 full-step synchronisation is chosen.

With this configuration, if the **INPUT** (0x21) is a multiple of 256  $\mu$ pulses (1  $\mu$ pulse = 1/256 of a full-step), the motor will be moving with  $\frac{1}{4}$  step increment towards its goal position where it will stop at a magnetically stable position.

If the user does not want to check if the **INPUT** (0x21) is a multiple of 256 or if the **INPUT** is computed by another microcontroller, bit **OPTIONS.21** can be used to automatically make the motor stop on full step (lowest byte of the **INPUT** is masked, hence equals to 0x00). Activate this feature by setting bit **OPTIONS.21** to '1'.

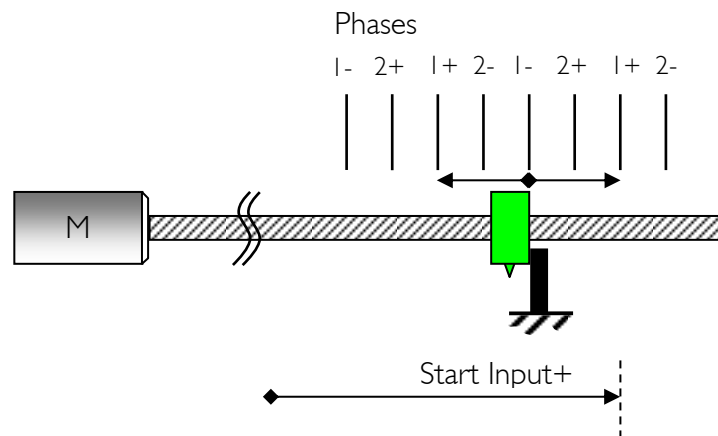
### 4 full-steps synchronisation

---

As explained above, setting the bit **OPTIONS.20** to '0' activates the 4 full-steps synchronisation instead of 1 full-step. This feature is useful when homing is done with a mechanical stopper (e.g. Homing method 12:

Move of Start Input) and the motor has few steps/turn and a small (or no) reduction with a gearhead. There are 4 full-steps (magnetically stable position) in one cycle of a 2 phases stepper motor; phase 2 positive, phase 1 positive, phase 2 negative and phase 1 negative.

The figure below explains the mechanism where a misalignment can appear if the 4 full-steps synchronisation is not used.



In this situation, the mechanical stopper is located close to a full-step (phase 1-). If the user asks the motor to go further the mechanical stopper with no synchronisation on the **POSITION** and the **INPUT**, the motor can possibly stop on the phase 1+, therefore the motor will sometime stay against the mechanical stopper, or go backwards towards the other phase 1+. This can lead to a misalignment of 4 full-steps when performing Homing.

To counter this, when needed by the application, the user can set the bit **OPTIONS.20** to '0', activating the 4 full-steps synchronisation as well as the **OPTIONS.19** set to '1' (position and phase synch enabled).

In the situation depicted above, the user will set the **HOMINGSTARTINPUT** (0x47) to have a rest of 768  $\mu$ pulses when divided by 1024 (**HOMINGSTARTINPUT** = 1024\*n + 768) to be sure that the motor will always stop at the same desired location. The table below gives the corresponding phase in function of the full-step position and the bit **OPTIONS.1** (invert direction).

		Bit <b>OPTIONS.1</b> = 0 (Invert direction = 0)	Bit <b>OPTIONS.1</b> = 1 (Invert direction = 1)
Rest [ $\mu$ pulses]	<b>POSITION</b>	Active phase	Active phase
0	-1024 0 1024 etc.	Phase 2+	Phase 2+
256	-768 256 1280 etc.	Phase 1+	Phase 1-
512	-512 512 1536 etc.	Phase 2-	Phase 2-
768	-256 768 1792 etc.	Phase 1-	Phase 1+

After the Homing is finished, the user can disable the synchronisation if it is not needed anymore.

### Consequence on modifying **POSITION**

---

When the **POSITION** (0x26) register is written to (by the user or after completed Homing), the synchronisation has to be kept; therefore the last 8 bits (1 full-step synch) or the last 10 bits (4 full-steps synch) of **POSITION** will keep their previous value.

For example the motor is actually on the first  $\frac{1}{4}$  step after Phase 2- (with **OPTIONS.I** = 0), it means that its position has the rest, when divided by 1024, of  $512 + 256/4$   $\mu$ pulses = 576  $\mu$ pulses (refer to the table in the previous chapter for more information). Let's assume that the **POSITION** is 92'736  $\mu$ pulses ( $1024*90+576$ ) and the user (or Homing) sets the **POSITION** to be '0'; if the 1 full-step synch is enabled, the **POSITION** will be 64  $\mu$ pulses and if the 4 full-steps synch is enabled, the **POSITION** will be 576  $\mu$ pulses after the register is written to '0'.



## 6. Ethernet Interface

### General

The board includes an Ethernet port (RJ45 – connector J7) which allows access to all the module's parameters (registers) through a TCP or UDP connection.

Here you will find a small comparison table for these two protocols (non exhaustive):

Features	UDP	TCP
<i>Checksum (data integrity)</i>	YES	YES
<i>Multiport (data multiplexing)</i>	YES	YES
<i>Flow control</i>	NO	YES
<i>Acknowledge data</i>	NO	YES

Two ports are available using the **TCP protocol**:

- Port #80            HTTP communication
- Port #8010        Access to the control port

Only one port is accessible through the **UDP protocol**:

- Port #7010        Access to the control port

You will find a detailed description of the different ports in the following pages.

**Note:** *The board allows for up to 4 simultaneous TCP connections. That means for example, that 4 users can connect to the #80 port to view the web page, or 4 users can be connected to the #8010 port and control the I/Os, or even that two can view the page and two can control the I/Os, etc. With the UDP protocol, there are no limitations to the number of users who can connect.*

## TCP-HTTP port (TCP # 80)

---

This port is used to access the web page stored on the module.

The user can simply access that port and ask for a particular page, using a standard web browser (IE, Mozilla, Safari, etc.), and enter the card's (IP) address:

*E.g. Type `http://169.254.5.5` in the address bar of the browser and the "index.htm" page will be loaded.*

## Control ports (TCP # 8010) & (UDP #7010)

---

This port is used to access the registers described in the "Register management" chapter of this manual.

**Note:** *If you are planning to configure all the registers using only the onboard web page, you can skip this section and go to the Java-Applet section. (The Java-Applet also uses this port to read and write the different settings).*

TCP/IP works in big endian: most significant byte first, followed by least significant byte(s). Access is made by sending a packet that follows a simple (6 byte header) protocol.

### Structure of each packet:

1) <i>Function ID (2 bytes)</i>	<i>Code of the function to be executed.</i>
2) <i>Transaction ID (2 bytes)</i>	<i>Number that defines this packet</i>
3) <i>Length of the parameters (2 bytes)</i>	<i>Number of parameters + data bytes</i>
4) <i>Parameters (X byte)</i>	<i>Parameters + data</i>
5) <i>Checksum (2 bytes)</i>	<i>Control sum of the packet bytes</i>

### Function ID

The specific code for each function can be found on the next page of this manual.

### Transaction ID

The user defines the values of the **Transaction ID**. Normally, each packet/transaction (communication request) should have a different ID (even though it is not mandatory). When the FMod-IPECMOT 48/10 receives a command/packet, it sends back an answer (at each request). This answer contains the same **Transaction ID** as the corresponding command previously sent. In that way, the user is able to confirm that each command has been executed.

### Length of the parameters

This 2-byte value corresponds to the length (in bytes) of the next section of the packet (parameters only).

Parameters

This part of the packet contains all the parameters (mainly the data that is sent).

Checksum

This 2-byte value is the checksum of all the bytes of the packet (more information in the following pages).

**READ register value command(s).**

Byte#		Number of bits	Example
0x00	Read (0x0021)	16 bits	0x0021
0x02	TransactionID	16 bits	0x1B34
0x04	Number of registers to read (X)	16 bits	0x0001
0x06	X * register addresses	X * 8 bits	0x02
0x06+X	Checksum	16 bits	0x...

The maximum number of registers that can be read at one time is about 30. The answer sequence should not be greater than 180 bytes. If the number of registers is too high, the FMod-IPECMOT 48/10 will answer only with the value of some of them.

The module **answers** with the following sequence:

Byte#		Number of bits	Example
0x00	Read Answer (0x0023)	16 bits	0x0023
0x02	TransactionID (same as demand)	16 bits	0x1B34
0x04	Number of bytes in answer	16 bits	0x0019
0x06	Register address	8 bits	0x02
...	Register value	8–128 bits (16B)	0x12345
The two previous entries are replicated for each register that needs to be read.			
...	Checksum	16 bits	0x...

### WRITE register value command(s).

Byte#		Number of bits	Example
0x00	Write (0x0022)	16 bits	0x0022
0x02	TransactionID	16 bits	0x1B34
0x04	Number of bytes in command	16 bits	0x0003
0x06	Register Addresses	8 bits	0x02
0x07	Register value	8 – 128 bits	0x1234
The two previous entries are replicated for each register that needs to be write.			
...	Checksum	16 bits	0x...

The max length of this sequence is 180 bytes.

The module **answers** with the following sequence:

Byte#		Number of bits	Example
0x00	Write Answer (0x0024)	16 bits	0x0024
0x02	TransactionID (same as demand)	16 bits	0x1B34
0x04	0x0000	16 bits	0x0000
0x06	Checksum	16 bits	0x...

### Easy IP address config (UDP # 7010)

A really useful feature of the UDP port #7010 is the "Easy IP config".

The user who wants to design their own software can use this feature to go through a "quick start/install" method. Indeed, as this protocol uses a broadcast UDP packet, even if the device is not in the same subnet, it should receive its new IP address and subnet mask.

Procedure:

Send a UDP broadcast message (using a local or direct broadcast IP address) to your network (inside which the FMod-IPECMOT 48/10 is connected) with the following command:

Byte#		Number of bits	Example
0x00	Change IP fct (0x002A)	16 bits	0x002A
0x02	TransactionID	16 bits	0x0000
0x04	Length of params (0x000E)	16 bits	0x000E
0x06	Device Mac address	6 bytes	0x0050C2308101
0x0C	Device new IP address	4 bytes	0xC0A81064
0x10	Device new SubnetMask	4 bytes	0xFFFF0000
0x14	Checksum	16 bits	0x...

If the FMod-IPECMOT 48/10 recognizes its MAC address, it will answer this command with a simple acknowledgement and change its IP address and subnet mask IF NO TCP CONNECTION EXISTS TO THE BOARD.

Byte#		Number of bits	Example
0x00	Change IP fct ack (0x002B)	16 bits	0x002B
0x02	TransactionID	16 bits	0x0000
0x04	Length of params (0x0000)	16 bits	0x0000
0x14	Checksum	16 bits	0x...

## Checksum calculation

---

This checksum is the same as the IP checksum.

**Definition:** The sum of 1's complement (=all bits inverted) all 16-bit words of the whole message (FiveCo packet) except checksum bytes.

**Note:** all values are unsigned!

Sequence:

1. Clear accumulator

**Loop**

- x. Only if the last word is not composed of two bytes, the data byte will be the upper byte (big endian).
2. Compute the 1's complement of each 16-bit word, the result will be 16 bits.
3. Convert the last result from 16 bits to 32 bits, the result will be 32 bits: 0x0000+last result.
4. Add the last result to the 32-bit accumulator.

**Try the Loop**

5. Convert accumulator into two 16-bit words.
6. Add those two 16-bit words, the result will be a 16-bit word.
7. If an overflow occurs with the last addition (Carry), add 1 to the last result.
8. The last result will be the final result.

Example (in hexadecimal):

```

!0x0021 (0xFFDE) → 0x0000FFDE (Read)
+!0x1234 (0xEDCB) → 0x0001EDA9 (TransID)
+!0x0003 (0xFFFF) → 0x0002EDA5 (3 reg to read)
+!0x0A10 (0xF5EF) → 0x0003E394 (reg 0A,10,02)
+!0x02(00) (0xFDFF) → 0x0004E193

```

Note that in this case an end 00 is implicitly used. (02 → 02 00).

```

0x0004 + 0xE193 = 0xE197, (carry=0)
0xE197 + carry = 0xE197

```

**Checksum = 0xE197**

Here is an example of a checksum calculation function in C++:

```
int RetChecksum(Byte* ByteTab, int Size)
{
    // This function returns the calculated checksum
    unsigned int    Sum=0;
    bool            AddHighByte=true;
    unsigned int    ChecksumCalculated;

    for(int i=0;i<Size;i++)
    {
        if(AddHighByte)
        {
            Sum+=((ByteTab[i]<<8)^0xFF00;
            AddHighByte=false;
        }
        else
        {
            Sum+=(ByteTab[i]^0x00FF;
            AddHighByte=true;
        }
    }
    if (AddHighByte==false)
        Sum+= 0xFF;

    ChecksumCalculated = ((Sum>>16) &0xFFFF) + (Sum&0xFFFF);

    ChecksumCalculated = ((ChecksumCalculated>>16) &0xFFFF)
        + (ChecksumCalculated&0xFFFF);

    return ChecksumCalculated;
}
```

This function needs a Byte array (ByteTab) containing the command sequence and the array length (Size) as input. It returns the checksum as an int.

## 7. I2C Interface

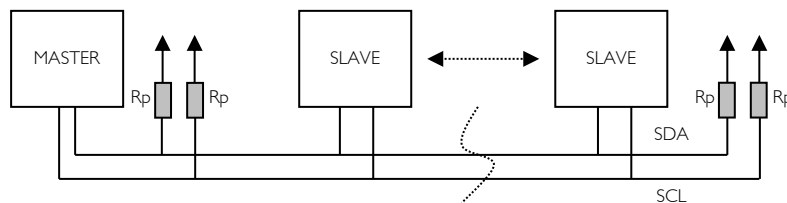
### Description

Registers are written to and read from the devices through the I2C bus; the controllers are all I2C slave device. It is controlled by the I2C clock (SCL), which is driven by the I2C master. Data is transferred into and out of the cards through the I2C data (SDA) line. Either the slave or master device can pull the SDA line down; the I2C protocol determines which device is allowed to pull the SDA line down at any given time.

The maximum speed of the fast I2C interface is **400 kHz**.

#### WARNING:

A **pull up resistor  $R_p$**  (off-card) has to be placed between SDA and VCC (+5V) and between SCL and VCC. This is usually done at the beginning (near the master) and at the end (near the last device of the daisy chain) of the SDA and SCL lines.



( $R_p = 4.7k\Omega$ )

### Protocol

The I2C bus defines several different transmission codes, as follows:

- a start bit
- the slave device 8-bit address
- an (no) acknowledge bit
- an 8-bit message
- a stop bit



## Sequence

---

A typical read or write sequence begins by the master sending a start bit. After the start bit, the master sends the slave device's 8-bit address. The last bit of the address determines if the request will be a read or a write, where a '0' indicates a write and a '1' indicates a read. The slave device acknowledges its address by sending an acknowledge bit back to the master. If the request was a write, the master then transfers the 8-bit register address to which a write should take place. The slave sends an acknowledge bit to indicate that the register address has been received. The master then transfers the data 8 bits at a time, with the slave sending an acknowledge bit after each 8 bits.

The motor controllers use data length of 1 byte up to 16 bytes for their internal registers. The master stops writing by sending a restart or stop bit.

A typical read sequence is executed as follows:

First the master sends the write-mode slave address and 8-bit register address just as in the write request. The master then sends a (re)start bit and the read-mode slave address. It clocks out the register data 8 bits at a time. The master sends an acknowledge bit after each 8-bit transfer. The data transfer is stopped when the master sends a no-acknowledge bit.

## Bus Idle State

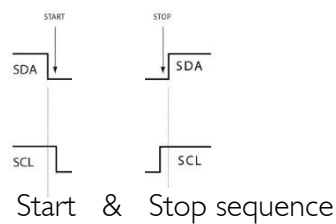
---

The bus is idle when both the data and clock lines are HIGH. Control of the bus is initiated with a Start bit, and the bus is released with a Stop bit. Only the master can generate the start and stop bits.

## Start Bit and Stop Bit

---

The start bit is defined as a HIGH to LOW transition of the data line while the clock line is HIGH. The stop bit is defined as a LOW to HIGH transition of the data line while the clock line is HIGH.



## I2C address selection

---

On each controller with I2C interface, its 7bits of I2C address must be defined in hardware. Each line needs to be connected to +5v logic (1) or logic ground (0). Do not leave any pin of address floating!

As example, if you want to set one FMod-I2CDCMOT DB 48/1.5 to address 40 (0x28), convert 40 (decimal) in binary code (b'00101000'), the 7 least significant bits (→0101000) are the values to be set to the corresponding pins of J1 "I2C Address selection →6-0".

## Slave AddressWrite/Read

---

The 8-bit address of an I2C device consists of 7 bits of address and 1 bit of direction. A '0' in the LSB of the address indicates write-mode, and a '1' indicates read-mode.

If we want to do a **write-sequence** to the address 0x55, the AddressWrite is :

**AddressWrite = 0xAA** [i2c address (0x55)<<1 + direction bit (0)]

and if we want to do a **read-sequence**, the address used is:

**AddressRead = 0xAB** [i2c address (0x55)<<1 + direction bit (1)]

The **I2C address** of the device is the one that is hard-coded through the 7 address pins : [I2C Address selection bit 0,1,2,3,4,5,6,7 (in)]. With 7bits of address, 128 values are possible, but the first (8) ones (0x00-0x07) and last (8) ones (0x78-0x7F) are reserved for I2C protocol specific actions, these values must not be used as an I2C address. If the device is set with one wrong (reserved) address, it will take its default address 0x55 (85decimal, binary'1010101').

Therefore you can define the address you want between 0x08 to 0x77 (decimal 8-119).

## Data Bit Transfer

---

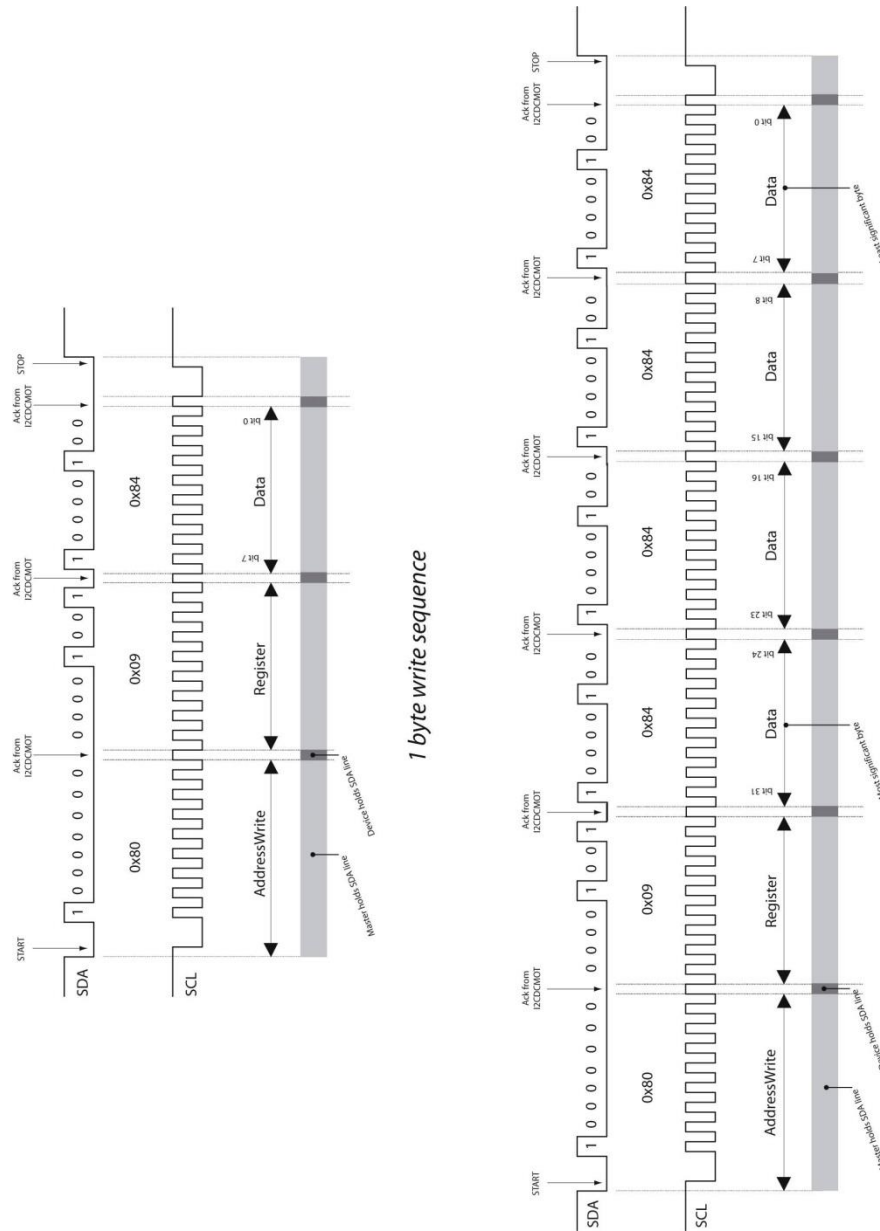
One data bit is transferred during each clock pulse. The I2C clock pulse is provided by the master. The data must be stable during the HIGH period of the I2C clock. It can change only when the I2C clock is LOW. Data is transferred 8 bits at a time, followed by an acknowledge bit.

## Acknowledge and No-Acknowledge Bit

---

The master generates the acknowledge clock pulse. The transmitter (which is the master when writing, and the slave when reading) releases the data line, and the receiver indicates an acknowledge bit by pulling the data line low during the acknowledge clock pulse. The no-acknowledge bit is generated when the data line is not pulled down by the receiver during the acknowledge clock pulse. A no-acknowledge bit is used to terminate a read sequence.

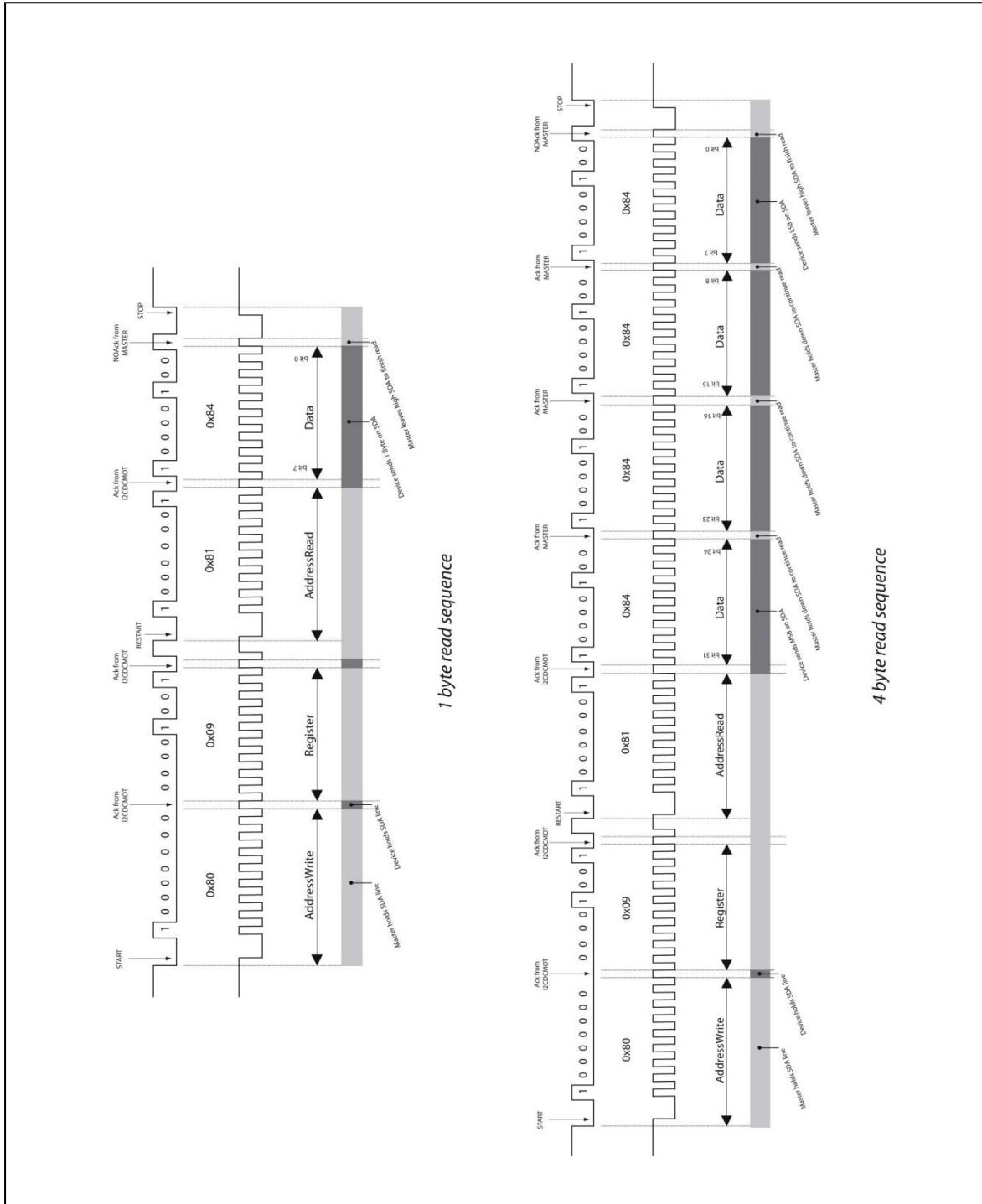
## Write Sequence (1 byte and 4 bytes)



1 byte write sequence

4 byte write sequence

## Read Sequence (1 byte and 4 bytes)



1 byte read sequence

4 byte read sequence

## 8. RS485 Interface

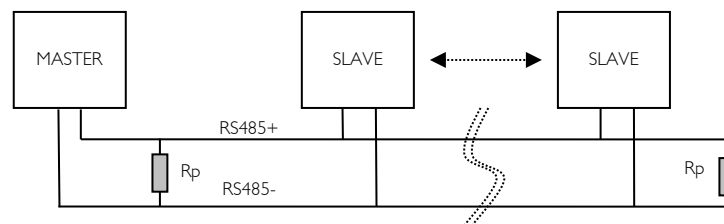
### **Physical layer – RS485**

Registers are written to and read from the devices through the RS485 lines. Standard RS485 electrical specifications apply for our motor controllers. The motor controllers are slave devices and the master always initiates the communication. The register access protocol is described in the next chapter, it is Question & Answer oriented.

The Baud rate of the communication is **115'200 Bd**.

#### **WARNING:**

A **termination resistor  $R_p$**  (off-card) has to be placed between RS485+ and RS485-. This is usually done at the beginning (near the master) and at the end, near the last device of the daisy chain.



**$R_p = 100-200\Omega$ , depends on the Master RS485 driver.**

## Registers Access Protocol

---

### Question & Answer oriented

---

This protocol is based on the principle of a request eventually followed by an answer. The transmitter of the request should wait for the answer before sending another request.

### Packet structure

---

#### Header

The requests and answers are based on the following structure:

1 Byte Destination address	1 Byte Source address	1 Byte [2 -127] Following DATA length	x Byte(s) DATA x ≤127
----------------------------------	-----------------------------	--	-----------------------------

DATA = (Fonction + parameters) × #fonctions + EndOfFrame + Checksum.

The protocol is LSB-type (**least significant byte first**) : when the data of a register are more than 1 byte long; it is the Least significant byte that is sent first.

- The first Byte is the address of the destination of the packet (0x01-0xFF).
- The second Byte is the address of the emitter of the packet. When the communication is made with a computer through a USB-RS485 bridge (COM port), the emitter address is arbitrary and can be taken into account by software (e.g. 0x10).
- The third Byte is the length of the rest of the frame called "Data". Its value is minimum 2 because "DATA" always finishes with 1 Byte "End of Frame" + 1 Byte of "Checksum". In that case, the packet would be empty. The maximum length of "DATA" is 127 bytes.

#### DATA

Contains a **succession of messages** independent from each other.

Each message could be:

- A register read request
- A register read answer
- A register write
- Ask of an acknowledge
- Acknowledge
- An error message (e.g. packet size too large)
- End of frame

List of messages that « DATA » could contain, in **binary** code:

b000x xxxx (0x00-0x1F)	Register read request (l register)  (x xxxx is register length in #Bytes [1-31]) + 1 Byte register index E.g. Register index for <b>POSITION</b> = 0x26
b001x xxxx (0x20-0x3F)	Register read answer (l register)  (x xxxx is register length in #Bytes [1-31]) + 1 Byte register index + n [1-31] register value Bytes
b010x xxxx (0x40-0x5F)	Write register  (x xxxx #Bytes to write [0-31]) + 1 Byte register index + n register value Bytes A Write request can be 0 Bytes, therefore it is a function call. E.g. <b>SAVEUSERPARAMETERS</b> (0x03); Write code = 0x40 0x03
b011x xxxx	Reserved
b100x xxxx	Reserved
b101x xxxx	Reserved
b1100 xxxx	Reserved
B1101 xxxx	Reserved
b1110 0000 (0xE0)	'Type Ext' register index error + 1 byte register index  This answer message indicates that a register could not be accessed. Possible case: Trying to write in a register that is "read-only" The register does not exist The size of the register is not good
b1110 0001 (0xE1)	'Type Ext' Frame ID + 1 byte Frame ID  This message forces the addressee to answer to this acknowledgement request with the same frame ID. The Frame ID (0-255) is an arbitrary value chosen by the emitter of this message.
b1110 0010 (0xE2)	'Type Ext' Frame ID Answer + 1 byte Frame ID  This is the "acknowledgement request" answer containing the same Frame ID than the acknowledgement request.
b1110 0011 (0xE3)	'Type Ext' End of Frame (standard) + 1 byte Checksum  This is ALWAYS the last message of a standard packet, indicating the end of the packet (cancel 0xE5 answers). It is followed by 1 Byte of checksum. The checksum calculation is explained bellow.
b1110 0100 (0xE4)	'Type Ext' Frame error  This message indicates an overflow on the packet size (>127 Bytes) of the answer. E.g. if we asked too many registers to read, the data preceding this message are valid, the following ones were not executed and abandoned. This message is always followed by 'Type Ext' End of Frame + 1 byte Checksum.
b1110 0101 (0xE5)	'Type Ext' End of Frame Multi packet (keep answer) + 1 byte Checksum  It is ALWAYS the last message of a multiple packet, except if this is the

	last one. The answer is kept by the addressee until 0xE6 or abandoned with 0xE3. A pause of 1ms minimum is needed before sending another packet.
b1110 0110 (0xE6)	'Type Ext' End of Frame End of Multi packet + 1 byte Checksum  It is ALWAYS the last message of the last multiple packet. The answers temporarily kept (0xE5) must be transmitted. If 2 packets contain 0xE6, the second one has a completed answer.
b111x xxxx (0xE7-0xFF)	Reserved

**Remark:**

If a packet does not need an answer (e.g. only containing writes without frame ID), a minimum pause of 1ms is needed after sending the last octet, before sending the next packet.

**No answer**

There is no answer to a packet in the following situations:

- Checksum is wrong
- The preceding Byte of the Checksum is not an "End of Frame"
- The size of the packet is larger than the value written in the 3<sup>rd</sup> byte + 3 (Destination address, source address and data length Bytes)

**Packet example**

Read request of 2 registers (4 bytes each): **TYPE** (0x00) and **VERSION** (0x01)

- Destination address = 0x20
- Source address = 0x10
- Data length = 0x06, 2x read request (2 bytes each) + 1 byte EoF + 1 byte Checksum

**Hexadecimal** : 20 10 06 04 00 04 01 E3 **22**

Answer :

- Type = 6.3 (FMod-IPECMOT 48/10 T1)
- Version = HW 1.8, FW 5.14
- Data length = 0x0E, 2x read answer (6 bytes each) + 1 byte EoF + 1 byte Checksum

**Hexadecimal** : 10 20 0E 24 00 03 00 06 00 24 01 0E 05 08 01 E3 **8F**

**Checksum calculation example**

The checksum calculation is made adding all the Bytes in the packet, including the 'End of Frame'.

**Request** :  $0x20+0x10+0x06+0x04+0x00+0x04+0x01+0xE3= 0x0122$

Only the LSB is kept, therefore the Checksum = **0x22**.

**Answer** :

$0x10+0x20+0x0E+0x24+0x00+0x03+0x00+0x06+0x00+0x24+0x01+0x0E+0x05+0x08+0x01+0xE3 = 0x018F$

Only the LSB is kept, therefore the Checksum = **0x8F**.



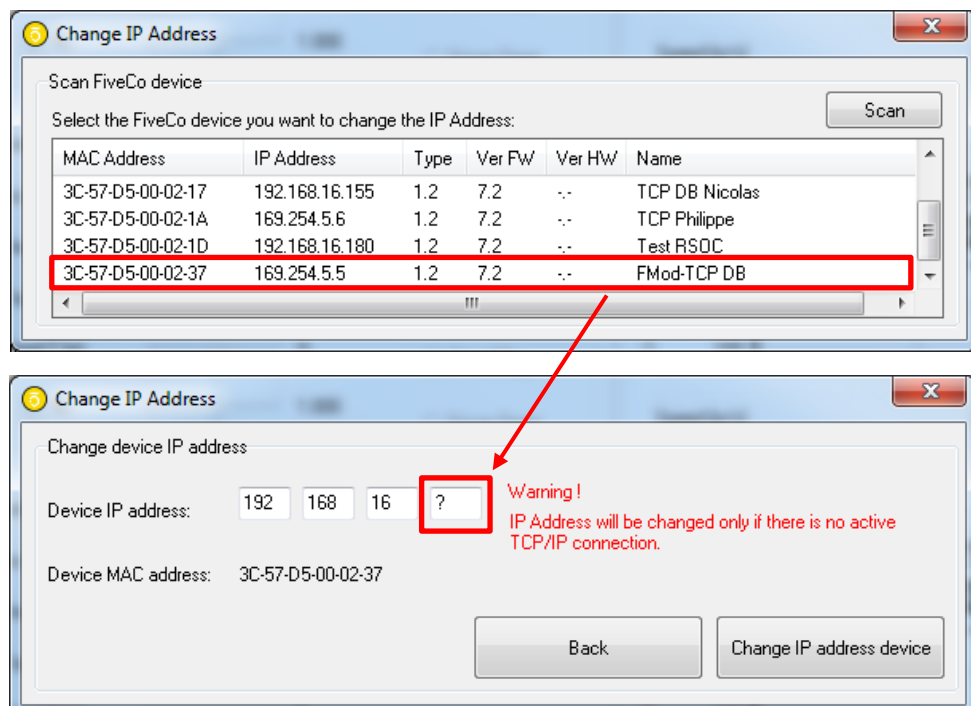
## 9. Configuration software : FSoft-MotorCtrl

A specific software is provided with the FiveCo's motor controllers in order to configure all the parameters as well as a graphical visualization of the trajectory. The software is available at <http://www.fiveco.ch/motor-controllers-products.html> -> left-click on the device you are using -> Support, under the **Softwares** section you can download the latest version of the FSoft-MotorCtrl.

This software is compatible with all the FiveCo's motor controllers connected through Ethernet either with a direct connection for the two FMod-IPECMOT cards or through a FMod-TCP BOX 2 or FMod-TCP DB, which is a bridge between TCP/IP and I<sup>2</sup>C, for the devices without an embedded Ethernet connection.

### Overview

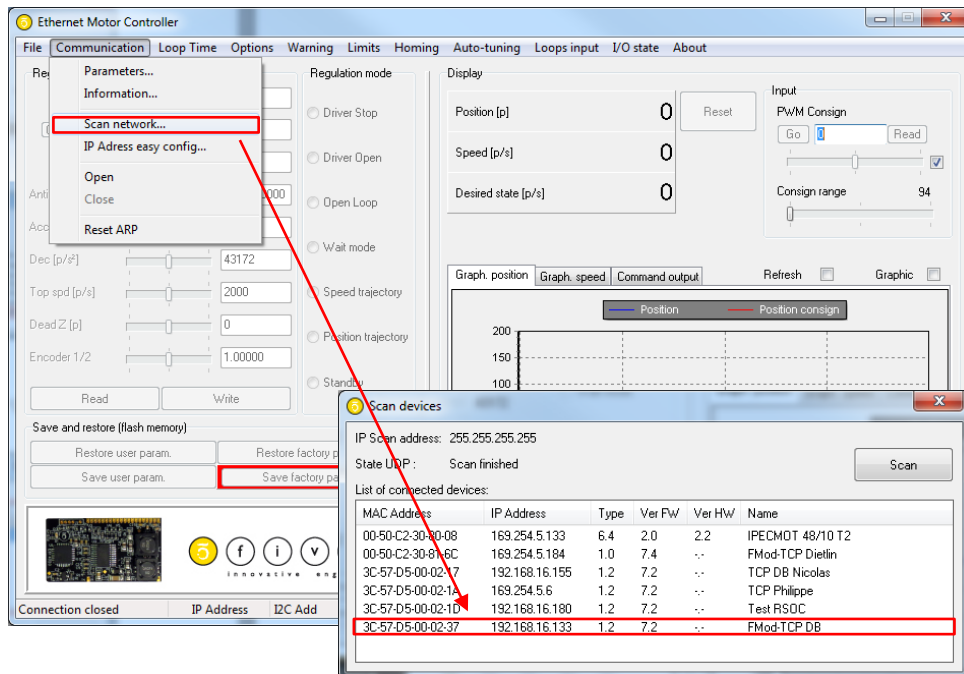
First connect the device to an Ethernet network (direct from the computer or to a local network). Then simply execute the "FSoft\_MOTORCTRL\_vx\_xx.exe" and click on **Communication > IP Address easy config**. A window pops up with the devices connected to your network with their respective MAC Address. If the IP Address is not in your subnet mask range, click on the device and change it manually to suits your need. The "Scan" button allows you to update this window.



In this example, the communication between a FMod-I2CDCMOT SLP 48/I and the computer is made through a FMod-TCP DB. In the above figure, the IP address is set to 192.168.16.133.

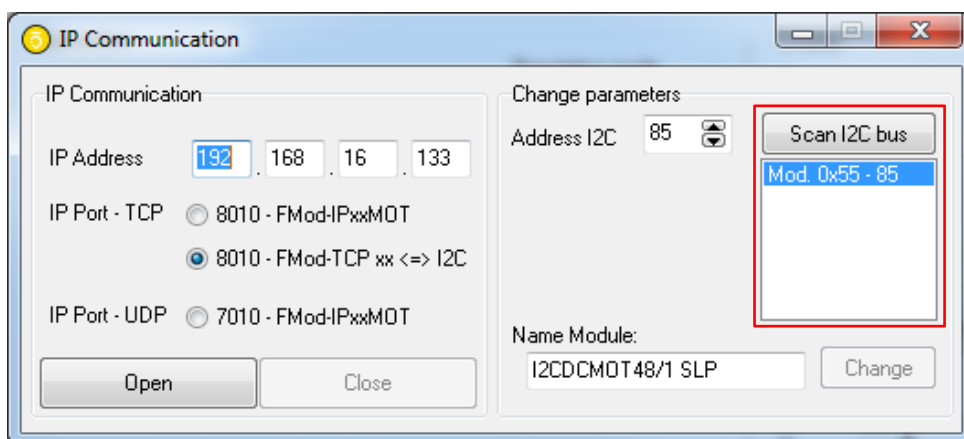
Once the IP address is configured correctly, a connection to the device can be made.

Click on **Communication > Scan network**.



Left-click on the desired device.

If the communication is established through an FMod-TCP DB/BOX 2, the following window will be displayed.

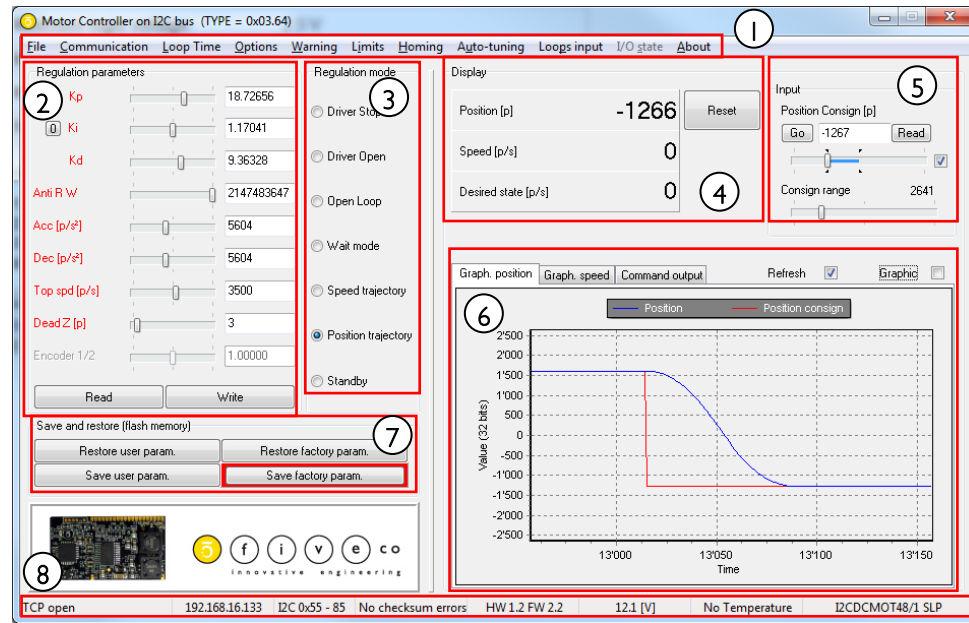


In this example, the device I<sup>2</sup>C address is the default one (0x55) and it is an I2CDCMOT 48/1 SLP motor controller. Click on the "Open" button to start the communication between the computer and the device.

This tab can be accessed afterwards under Communication->Parameters. The name of the module can be changed as well as the communication parameters. If the connection is made directly with the motor controller (e.g.

FMod-IPECMOT 48/10 T1 & T2) select the first check button “8010 – Fmod-IPxxMOT” to access through TCP or “7010 – Fmod-IPxxMOT” to access through UDP, if the connection is made through a FMod-TCP DB/BOX 2 select “8010 – Fmod-TCP xx ↔ I2C”.

The following figure shows the main window of the FSoft-MotorCtrl.



## I. Tab menu

- File : Allow the user to save the motor controller configuration as well as import parameters from previously saved file.
- Communication : The user can change the communication parameters. Under **Information**, the rate at which the device is questioned can be changed. Under **Parameters**, the name of the device can be changed.
- Loop Time : Allow to change the frequency at which the regulation is completed. For more information concerning Loop Time, see the description of the register **LOOPTIME** (0x2D).
- Options : Read and write the **OPTIONS** register. For more information see the description of the register **OPTIONS** (0x2C).
- Warning : Visualize and clear the **WARNING** register. For more information see the description of the register **WARNING** (0x08).
- Limits : Allow the user to configure the behaviour of the device when Limits inputs are reached. Limit2 cannot be used for homing purpose. Refer to chapter 12, 13 and registers related to Limits configuration (**LIMIT1SETUP** (0x50) → **LIMIT1XINPUT** (0x53), **LIMIT2SETUP** (0x58) →

- LIMIT2XINPUT** (0x5B); **HOMINGOPTIONS** (0x48) for more information.
- g. Homing : Homing configuration tab. Do not forget to save the user parameters in the main window before launching the homing since **ACCELERATION**, **TOPSPEED** and **CURRENTMAX** saved value will be restored when homing is completed. For more information on Homing, see chapter 13 and the registers related to homing (**HOMING** (0x49), **STOPHOMING** (0x4A), **HOMINGOPTIONS** (0x48), **HOMINGPOSITION** (0x4B), **HOMINGINPUT** (0x4B) and **HOMINGSTARTINPUT** (0x47)).
  - h. Auto-tuning : More information in chapter 11 and **AUTO-TUNING** (0x39) register.
  - i. Loops input : More information in chapter 14 and the register related to Loops configuration (**ENHANCEDINPUTS** (0x4D), **LOOPSCONFIG** (0x4E) and **ENHANCEDPARAMS** (0x4F)).
  - j. I/O state : Shows the state of the 2 I/Os as well as the limits. More information in the **IOSTATE** register (0x56).
  - k. About : Version of the software.

## 2. Parameters

---

This panel allows the user to manually set the parameters of the motor controller.

- **KP** (0x33)
- **KI** (0x34)
- **KD** (0x35)
- **ANTIRESETWINDUP** (0x36)
- **ACCELERATION** (0x40)
- **DECELERATION** (0x41)
- **TOPSPEED** (0x42)
- **DEADZONE** (0x43)
- **ENCODERSRATIO** (0x44)

More information on each individual parameter in the corresponding registers description at the end of this manual and at chapter 10 "Motion Control modes".

The trackbars can be used to change the value of each parameter, as well as the field next to them. When a value is entered in one field, press enter to send that only value to the controller. Instead when the "Write" and "Read" button are used, it writes/read all the values to/from the controller.

## 3. Regulation mode

---

Manually set the regulation mode in which the controller is operating. The regulation mode is continuously read from the controller, hence it can change by itself (e.g. when an over/under voltage is detected, standby, etc.).

Refer to chapter 10 “Motion Control modes” to have more information on the regulation modes.

#### 4. Trajectory values

---

Numerically displays the actual position and the actual speed. For DC brushed and brushless motor controllers, also shows the desired state, which is the speed that the PID regulator is trying to reach.

For FMod-I2CSTEPMOT, displays the position in full steps with 1  $\mu$ pulse = 1/256 of a full step.

“Reset” button writes 0 to the **POSITION** (0x26) register.

#### 5. Input

---

Allow the user to set the input which could be either the position goal, the speed goal or the PWM consign sent to the controller depending on the actual regulation mode. The consign can be change by entering a value in the field and then press “Go” to send it or “Enter” key.

Another choice is to use the two trackbars; the bottom one changes the range of the trackbar above it. You do not need to press the “Go” button to send the command if the box on the right is checked, it automatically sends the command when changed. Uncheck this box and you will be able to move the consign trackbar without sending its value.

#### 6. Graphical visualization

---

3 different graphics are available. One displays the position trajectory of the motor, the second one the speed trajectory and the third one the PWM output of the controller. For the “Command output” tab, the range is +/-100% of the maximum PWM that the controller can deliver.

Two checkboxes are available on the top right side of the graphical visualization.

“Refresh”: when unchecked, it stops continuously reading the registers from the controller, hence no position update, no regulation mode update, etc.

“Graphic”: when unchecked it stops the graphical visualization, but still updates the position, regulation mode, etc.

#### 7. Save parameters

---

It allows the user to save the current parameters of the controller in the EEPROM memory of the device. Thus at power-up, it will restore back the parameters previously saved. For FMod-I2CDCMOT SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1, the “Save/Restore user param.” is exactly the same as “Save/Restore factory param.”, it points to the same EEPROM data.

On the other controllers, user and factory parameters are different. The registers saved/restored when using those functions are described under the **SAVEUSERPARAMETERS** (0x03) description at the end of this manual.

## 8. Status bar

---

The communication status, IP address, I<sup>2</sup>C address, hardware and firmware versions, voltage, temperature and device name are shown in the status bar.

## 10. Motion Control modes

### General parameters

---

The motion controller drives DC motor by controlling the motor input voltage.

The system works as follow:

- Desired setpoint of the motion is set in the **INPUT** register.
- Depending on the regulation mode, the motion controller generates the **COMMAND** register that is a percent of PWM (equivalent to a percent of voltage in the system)
- The **COMMAND** is sended to the voltage driver that powers the motor

The main **INPUT** (0x21) (PWM/Speed/Position) is software-limited with configurable **INPUTMIN** (0x24) and **INPUTMAX** (0x25) values.

The motor voltage and motor current are software-limited with configurable **OUTPUT VOLTAGE MAX** and **CURRENT MAX** values

### Motor regulation parameters

---

Speed and position regulation of DC motor is based on a PID algorithm and a feed forward prediction technique.

#### Feed forward

---

##### VFFOFFSET (0x61)

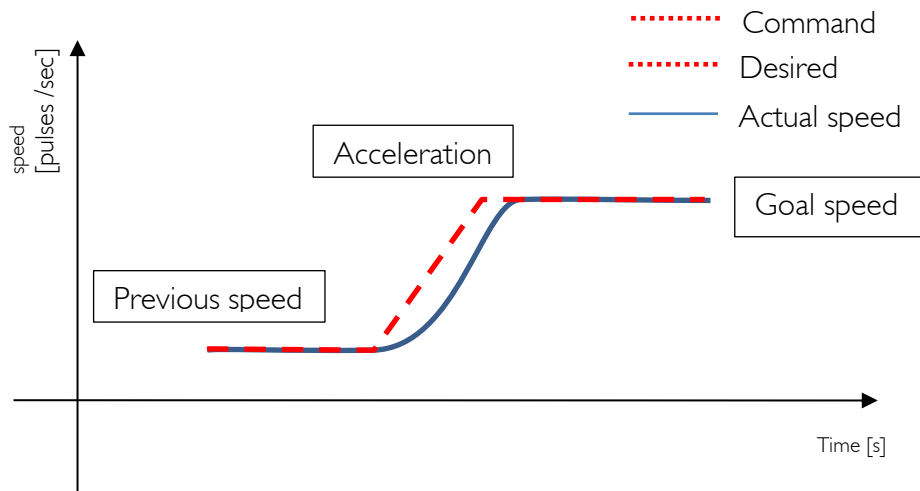
A constant value is added to the **COMMAND** in order to compensate de friction losses of the motor. By default, **VFFOFFSET** is set to -1000.

##### KVFF (0x62)

For ideal motor (without frictions, internal resistance and inertia), the speed of the motor is proportional to its supply voltage. Therefore, the motor can be controlled by adding a velocity feedforward term to the command. The velocity feedforward term is the result of the multiplication of the gain **KVFF** with the desired speed. The ideal **KVFF** is the value for which **SPEED** = **DESIRED** when the motor turns at a constant speed with no load and when **VFFOFFSET** compensates motor friction losses.

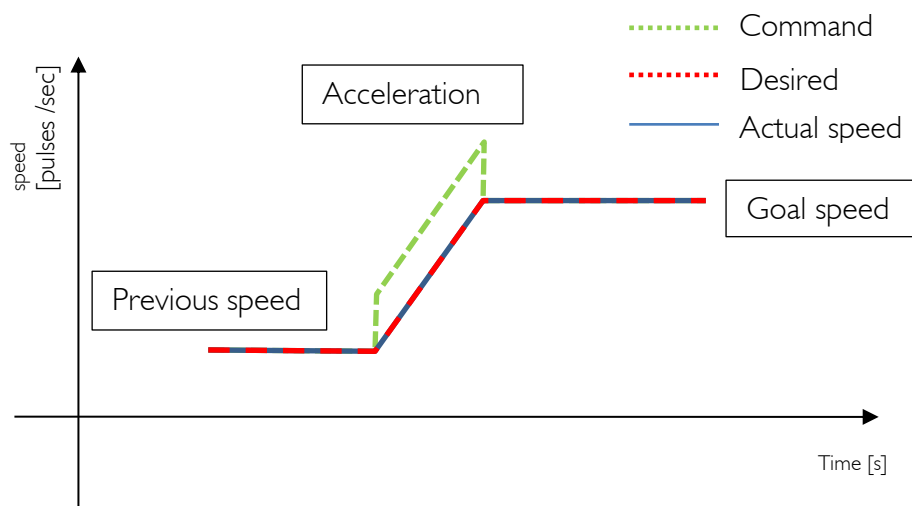
**KAFF (0X63)**

The torque produced by the acceleration of the motor (inertia) increase the current that flows through the motor. A part of this additional power is dissipated in the internal resistance of the motor and leads to a delay between the desired speed and the actual speed during acceleration phases.



*Speed profile without acceleration gain*

This delay can be reduced thanks to feed forward technic by adding an acceleration feedforward term to the command. Since the additional voltage needed is proportional to the desired acceleration, the acceleration feedforward term is the result of the multiplication of the gain **KAFF** with the desired acceleration.



*Speed profile in "Speed Control mode" with acceleration gain*



**KDFF (0x64)**

In position control mode, acceleration and deceleration parameters of the trajectory generator are differentiated. Therefore, a deceleration feed forward term is added to the command while the velocity is decreasing in position control mode. The deceleration feed forward term is the result of the multiplication of the gain **KDFF** with the desired deceleration. Generally,  $KDFF = [0.5 - 1] \cdot KAFF$ .

**KVFF**, **KAFF**, **KDFF** and **VFFOFFSET** depend mainly on the type of motor, voltage and encoder resolution.

(\*) **KVFF**, **KAFF**, **KDFF** and **VFFOFFSET** are obviously not implemented on **FMod-I2CSTEPMOT 35/1 & 35/0.1** since stepper motors don't need feedforward regulation.

**PID controller**

**KP**, **KI**, **KD** depend mainly on the type of motor, voltage and encoder resolution. The more the encoder has pulses per revolution, the smaller the **KP**, **KI**, **KD** values are.

(\*) **KP**, **KI**, and **KD** are obviously not implemented on **FMod-I2CSTEPMOT 35/1 & 35/0.1** since no feedback on the motor position is needed.

**List of regulation modes**

- Brake
- Driver Open
- Open Loop
- Wait
- Speed Control (Torque mode)
- Position Control
- Standby

**Brake mode**

**REGULATIONMODE** = 0x00.

**(\*) For brushed and brushless motor controllers**

Brakes the motor with **CURRENTMAX** register value. After 1-2 seconds, the motor is fully short-circuited.

**(\*) For stepper motor controllers**

Perform a high frequency switching on the phases of the motor, making it equivalent to a magnetic brake.

## Driver Open mode

**REGULATIONMODE** = 0x01.

Switches OFF the H-bridge power transistors.  
(Internal protection diodes still work.)

## Open Loop mode

**REGULATIONMODE** = 0x02.

(\*) *Not implemented in FMod-I2CSTEPMOT SLP 35/x, equivalent to Driver Open mode.*

**INPUT** register is copied to **COMMAND** register which is then converted without regulation to PWM output.

Maximum **INPUT** is 0x0000FFFF (65535). Represents 100% of forward PWM.

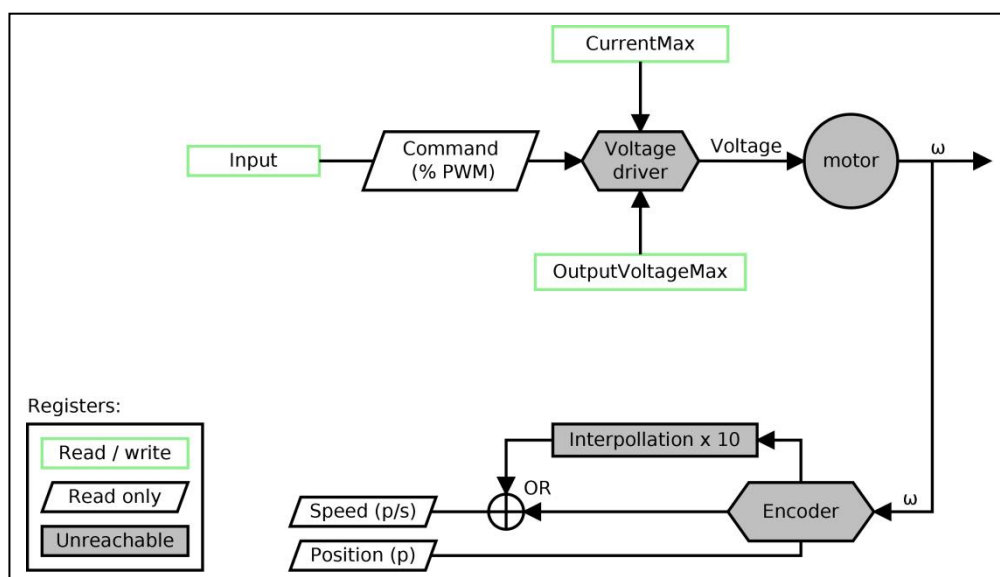
**INPUT** (0) represents 0% of PWM.

Minimum **INPUT** is 0xFFFF0001 (-65535). Represents 100% of backward PWM.

Use this mode if a custom regulation is made on another CPU.

With PWM 9 bits, the 7 least significant bits are unused.

When PWM 10 bits is selected, the 6 least significant bits are unused.



## Wait mode

---

**REGULATIONMODE** = 0x03.

**(\*) For brushed and brushless motor controllers**

This mode leaves the PWM and current output unchanged, and halts PID regulation).

**(\*) For stepper motor controllers**

This mode leaves the phases actuation speed and the current output unchanged.

All registers can be updated with no influence on the outputs.

This mode can be used when several registers need to be changed at the same time.

## Speed Control mode

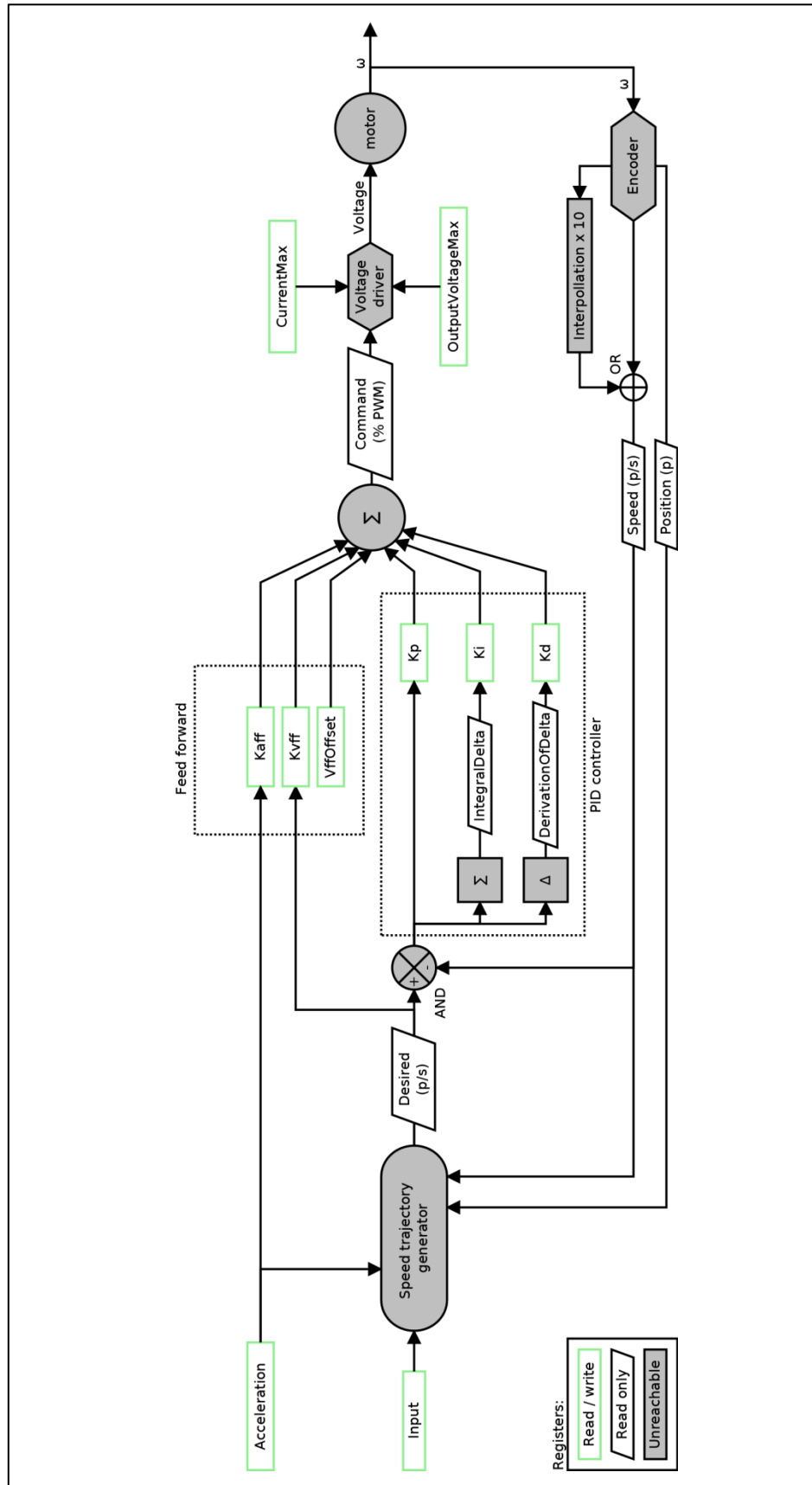
---

**REGULATIONMODE** = 0x04.

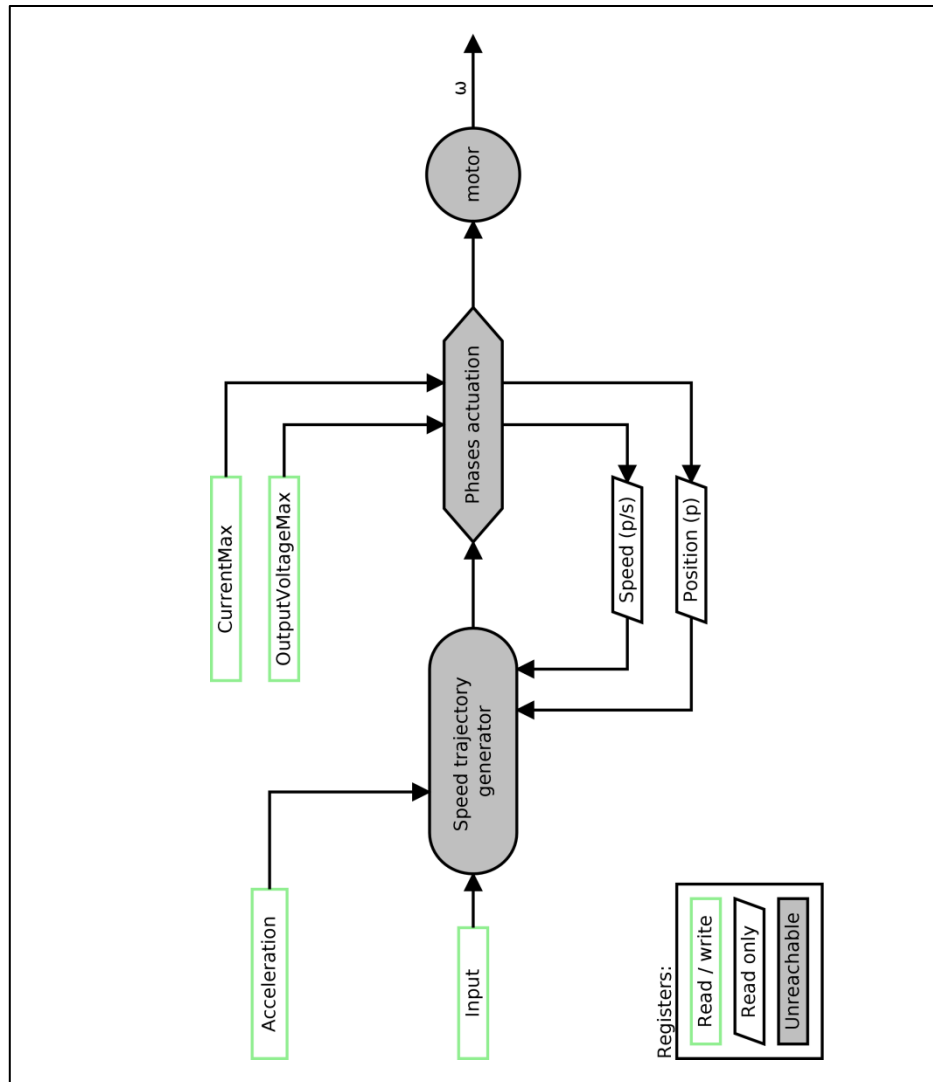
**INPUT** is the speed to reach. If immediate **INPUT** is **DESIRED**, set **ACCELERATION** to max.

If smooth control is needed, **DESIRED** value can accelerate up/down to **INPUT**. **ACCELERATION** defines the slope between the **INPUT** and **DESIRED** registers. **FEEDBACK** is represented by the effective **SPEED** of the motor.

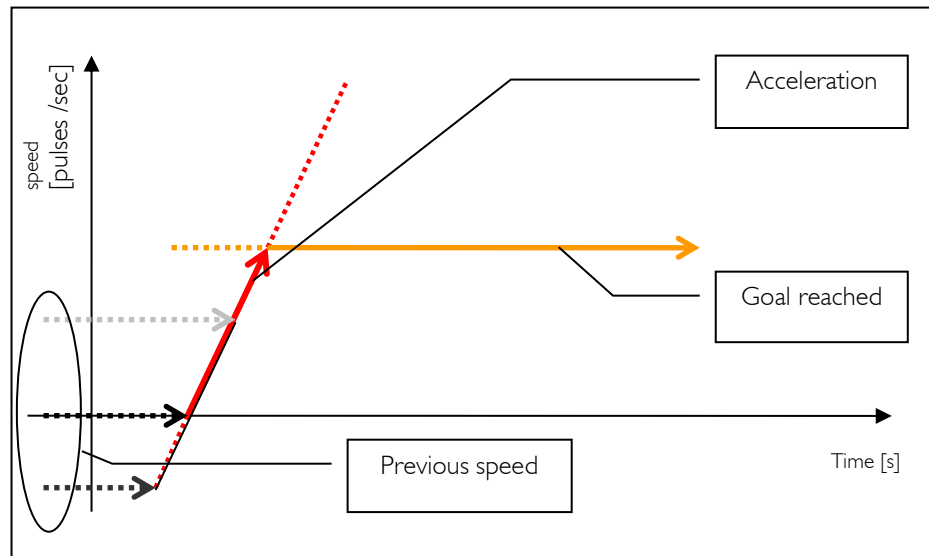
(\*) For brushed and brushless motor controllers



(\*) For stepper motor controllers



“Speed Control mode“ graph example:



Speed profile in “Speed Control mode”

Note: **TOPSPEED** is not used in “Speed Control mode”, only in “Position Control mode”. **INPUTMIN** (0x24) and **INPUTMAX** (0x25) registers provide further possibilities for replacing **TOPSPEED**. After a new **INPUT** (goal) has been set, it is possible to change **INPUT**, or **ACCELERATION** without having to wait for the goal to be reached.

## How to choose the correct PID parameters in Speed Control mode?

(\*) For brushed and brushless motor controllers

The device is able to find approximate good values for the PID and feedforward regulator by itself, see “Auto-tuning” chapter and register **AUTO-TUNING** (0x39) for more information.

After the **AUTO-TUNING** is done, test your application, and modify **ACCELERATION** to suit you.

If you think your application has a lot of friction or inertia, you can adapt feedforward gains. In that case, increment **VFFOFFSET** to counteract friction and **KAFF** and **KDFF** to counteract inertia. (It is better to deactivate PID regulation during the tuning if possible in order to see the effect of feedforward terms).

If you think that the PID regulator needs to be smoother, decrement **LOOPTIME** by 1 step (example: from 2ms to 5ms).

Otherwise, if you think that the PID regulator needs to be more dynamic, increment **KP** by +30% (example: from 1.4 to 1.82). If it is not enough, increment **LOOPTIME** by 1 step (example: from 2ms to 1ms).

When all is done, don't forget to run the function **SAVEUSERPARAMETERS**.

### ***Position Control mode***

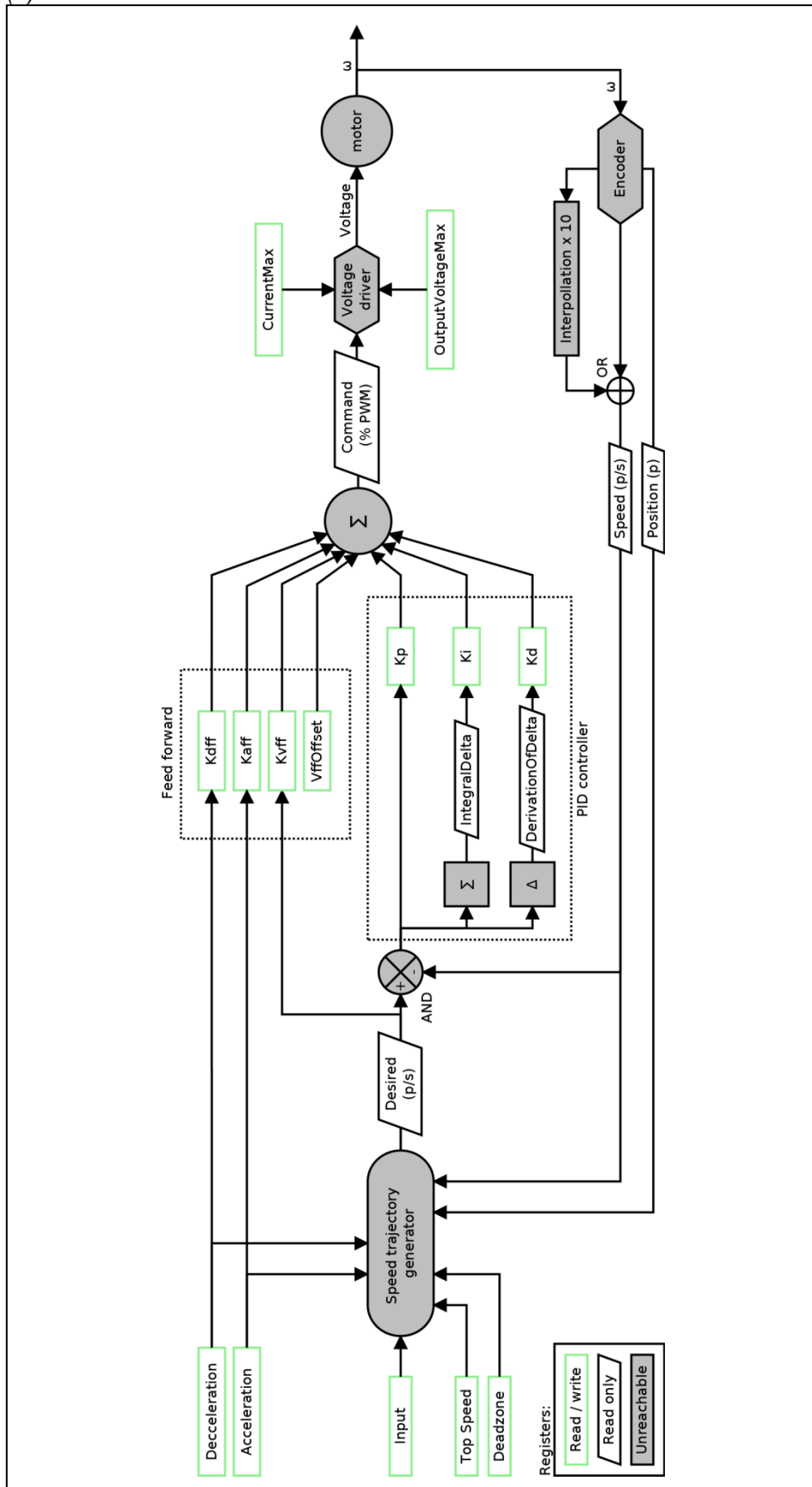
---

*REGULATIONMODE* = 0x05.

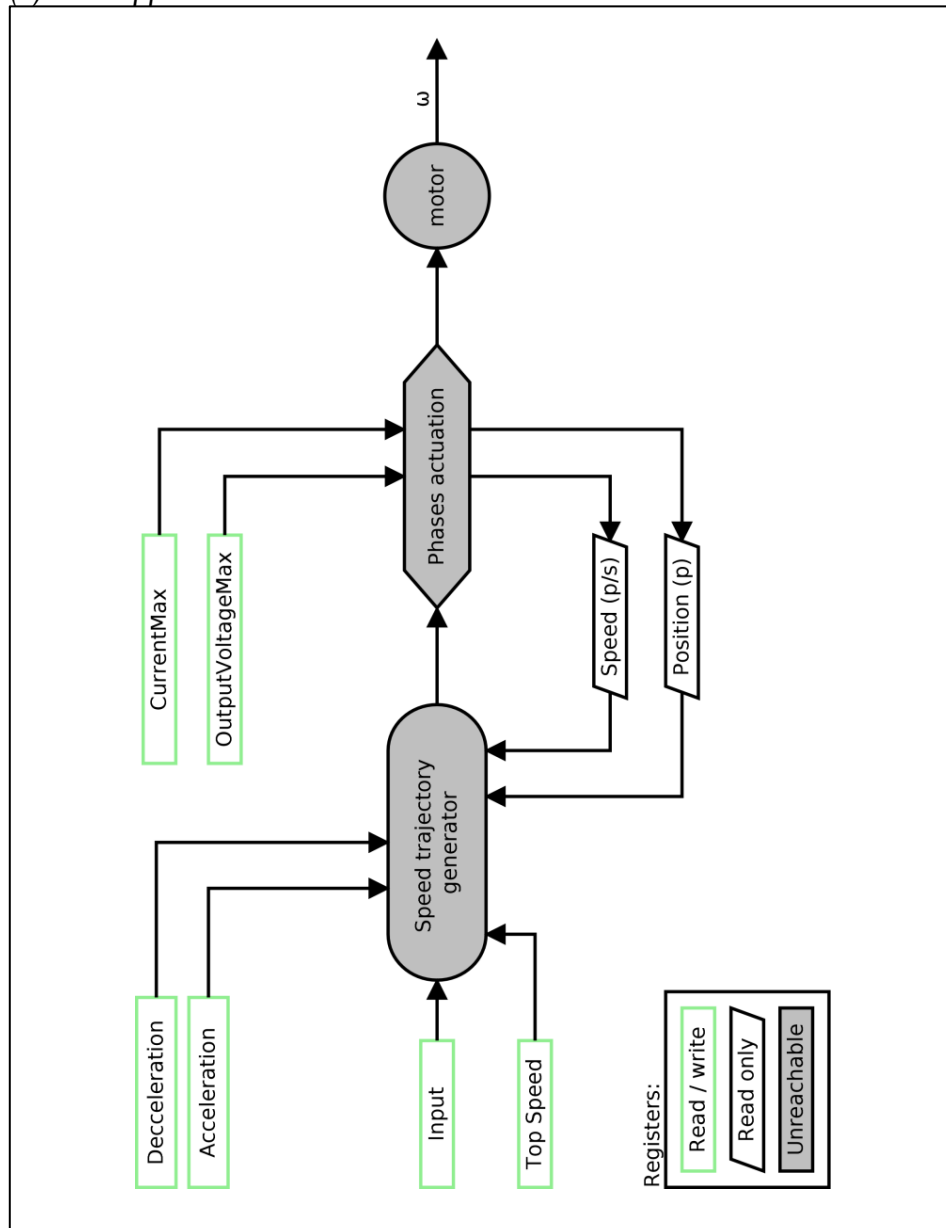
*INPUT* is the position to reach. The trajectory kernel calculates in real-time the *DESIRED* speed for the *PID* regulator and the *FEEDBACK* is represented by the effective *SPEED* and *POSITION* of the motor.



(\*) For brushed or brushless motor controllers



(\*) For stepper motor controllers



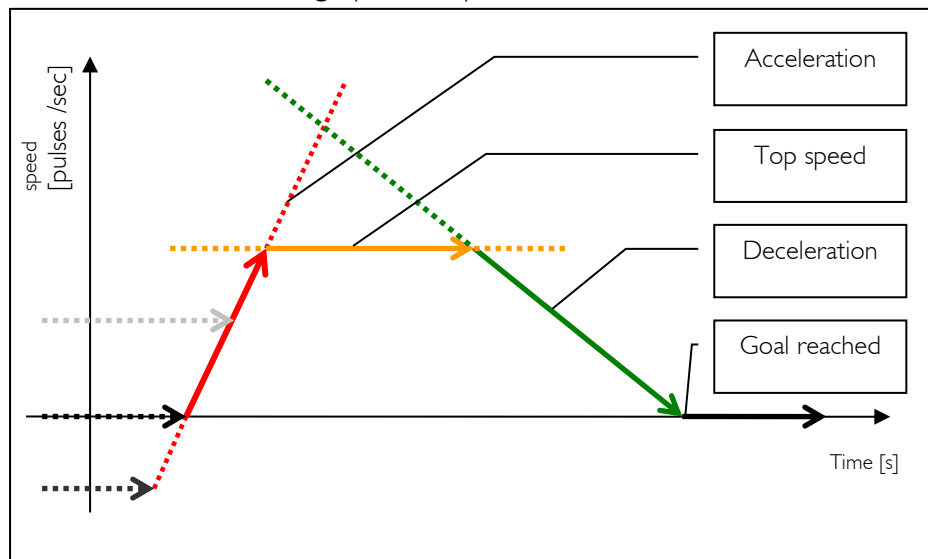
For the position trajectory, a full trapezoid speed profile consists of 3 different movements:

- **ACCELERATION** from actual speed
- **TOPSPEED**
- **DECELERATION** when approaching the goal.

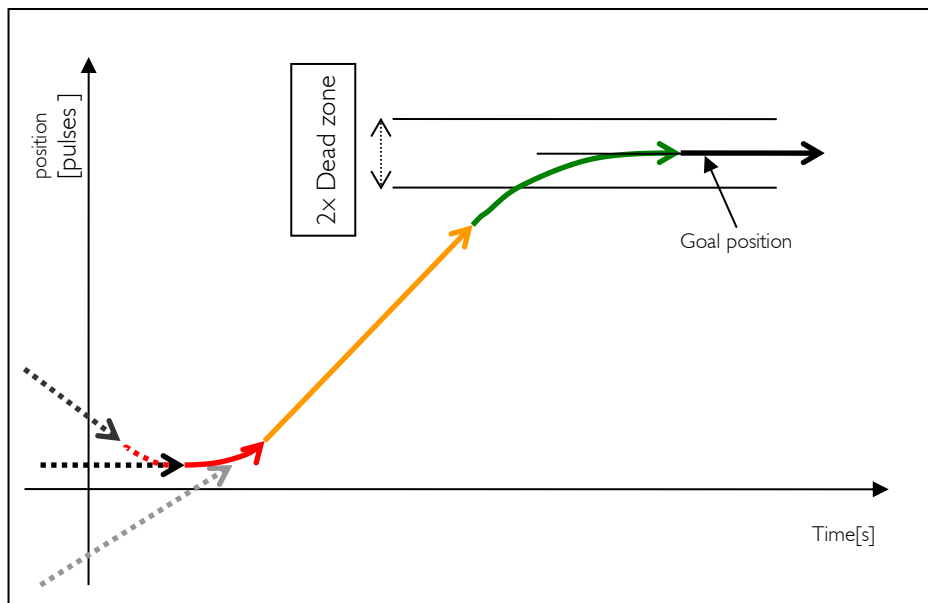
**DEADZONE** is complementary to the deceleration and defines a zone around the goal point ( $INPUT \pm DEADZONE$ ) where the **DESIRED** speed is forced to zero. It is useful when **DECELERATION** is greater than the effective motor deceleration. It will skip oscillations in this case.

**DEADZONE** is not taken into account for stepper motor controllers.

“Position Control mode” graph example:



Speed profile in the “Position Control mode”



Position profile in the “Position Control mode”

## How to choose the correct PID parameters in Position Control mode

(\*) For brushed or brushless motor controllers

The device is capable of detecting approximate correct values for the PID and feedforward regulators – see the chapter on “Auto-tuning”.

After the **AUTO-TUNING** is done, test your application, and modify **ACCELERATION**, **DECELERATION** to your requirements.

If you don't want your motor to run at full speed, reduce it by reducing **TOPSPEED**.

If you think your application has a lot of friction or inertia, you can adapt feedforward gains. In that case, change **VFFOFFSET** to counteract friction and **KAFF** and **KDFF** to counteract inertia. (It is better to deactivate PID regulation during the tuning in order to see the effect of feedforward terms).

If you think that the PID regulator needs to be smoother, decrement **LOPTIME** by 1 step (example: from 2ms to 5ms).

Otherwise, if you think that the PID regulator needs to be more dynamic, increment **KP** by +30% (example: from 1.4 to 1.82). If it is not enough, increment **LOPTIME** by 1 step (example: from 2ms to 1ms).

If the goal position is not accurate enough, try to reduce **DEADZONE**, but if oscillation occurs, increase **DEADZONE** until the oscillation disappears.

When all is done, don't forget to run the function **SAVEUSERPARAMETERS**.

## Standby mode

---

(\*) For *FMod-I2CDCMOT SLP 48/1* and *FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**REGULATIONMODE** = 0x06.

This is the ultra-low power mode, 50nA of current consumption on each supply power at ambient temperature to a maximum of 1µA at higher temperature. This consumption is obviously higher if a constant I<sup>2</sup>C communication is kept between the master and the device. Therefore, when the user wants to reduce power consumption, the controller is put in Standby mode and stops communicating with it until another mode is desired (Position control, speed control, etc.).

The motor power supply is shut-down, as well as the encoder power. Therefore if the motor is moved while in Standby mode, the reference (home) would be lost since the encoder is not powered.

When going out of Standby mode to another motion control mode (e.g. Position control mode), the encoder is powered-up again and, depending on the encoder type, it risks to generate unwanted pulses for a certain period of time. This is the case particularly for magnetic encoders, which perform an algorithm to find its actual position when powered-up. **SKIPPULSETIMER** (0x3A) register allows the user to set a time during which the incoming pulses from the encoder are not counted when powering up the encoder.

## 11. Auto-tuning

This chapter is only relevant for the brushed or brushless motor controllers, since no PID and feed forward regulation is done on stepper motor.

This feature helps application engineer to define the PID (proportional-integral-derivation) gains as well as feedforward gains for speed control or positioning. The motor must be mechanically disconnected from its application, because auto-tuning requires that the motor run at its maximum speed.

The developer first needs to select the maximum current possible (proportional to torque), and if a maximal output voltage is required, then it needs to be set before starting the auto-tuning, when it will be writing to the **AUTO-TUNING** register.

### What does auto-tuning consist of?

- During the test, the top speed will be measured, and 85% of the measured value is put into the **TOPSPEED** register (at that speed, the motor still has torque).
- Computed values of **KP**, **KI**, **KD**, **ACCELERATION**, **DECELERATION** and **DEADZONE** are updated.
- Computed values of **KVFF**, **KAFF**, **KDFF** and **VFFOFFSET** are updated.
- Depending on the speed of the motor and the resolution of the encoder, the refresh **LOOPTIME** value is updated.
- If encoders are swapped, the device automatically inverts the sign of their values (**OPTIONS** register)
- If encoders with poor resolution are used, an interpolation of 10x is set (**OPTIONS** register).
- During auto-tuning, the encoder source is updated for brushless motors only: if no A/B quadrature encoder is connected, then hall sensor states will be used as encoders.
- PWM frequency (**OPTIONS** register) is set to 69kHz (T2 125kHz) for small motors to reduce thermal dissipation of the motor.
  - **CURRENTMAX**<5A for FMod-IPECMOT T1 & T2 and FMod-I2C485ECMOT DB
  - **CURRENTMAX**<1A for FMod-I2CDCMOT DB & SLP
 PWM frequency is set to 35kHz (T2 63kHz) for big motors in order to reduce thermal dissipation.

- **CURRENTMAX**>5A for FMod-IPECMOT T1 & T2 and FMod-I2C485ECMOT DB
- **CURRENTMAX**>1A for FMod-I2CDCMOT DB

### Why do I need to mechanically disconnect the motor for auto-tuning?

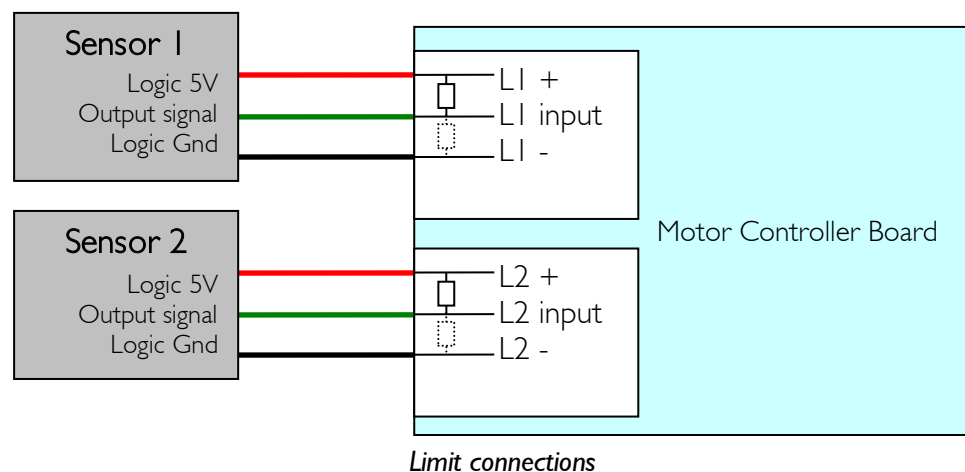
This device doesn't function by inertia computation, hence why the auto-tuning can be (and should be) done with no extra-inertia. The second reason is for the security of the user and the application.

## 12. Limit switches

The term “Limit” used in this manual can also be understood as: “end stroke”, “reference”, “stopper” or “action”. It depends on the external use of each limit signal.

Two independent limit/switch sensors can be connected to the motor controller. Only **Limit 1** can be configured to be the limit for “Homing”.

In FMod-IPECMOT 48/10 T1 & T2 and FMod-I2C485ECMOT DB 48/10, the power supply of 5V is provided from the motor controller board. For the other smaller cards, the power supply of 5V is applied externally.



To configure a limit, it is necessary to update a set of registers:

- **OPTIONS.5** The register selects pull-up or pull-down resistors of 1.5KOhm to both Limits.
- **LIMIT1SETUP** The register configures the activity of limit number 1.
- **LIMIT2SETUP** The register configures the activity of limit number 2.

Refer to the chapter “Register management” for further information.

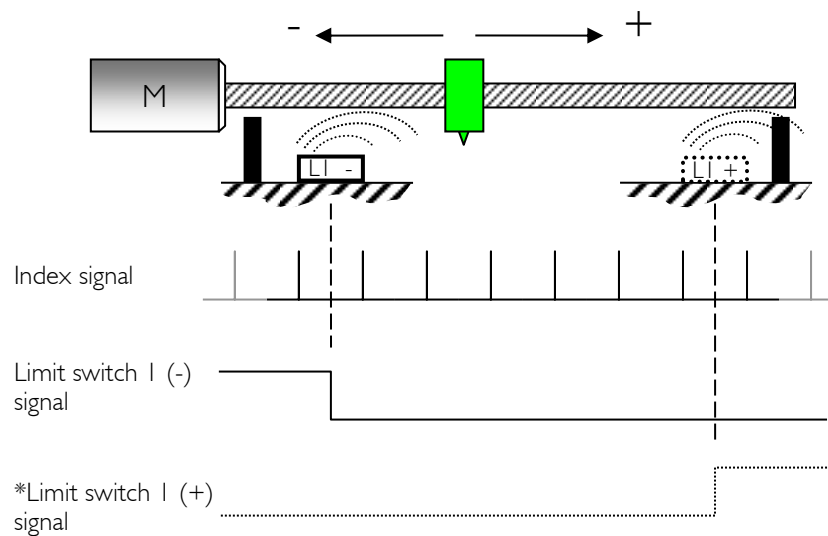
Here are some examples of limits:

- Reset: changes **REGULATIONMODE** to desired mode, sets **POSITION** and **INPUT**.
- Thrust: sets a new **INPUT** to return to normal range.
- Reference: only sets a new **POSITION** once reached.
- Stopper: changes **REGULATIONMODE** to Brake mode, to stop the motor.
- Other: any combination of **REGULATIONMODE**, **POSITION**, **INPUT**.



### 13. Homing (position reference)

The Motion Control card offers multiple methods for finding the home-position (only for positioning the system). This method called “Homing” is intended to be the calibration process for the reference position of the card.



\* *Alternative for Limit I sensor*

A mechanical or electrical signal generated by Limit switch LI can be combined with the index signal to define the home position. (The index signal can be found on the 3<sup>rd</sup> channel on the encoders, as well as the A/B signals).

The index signal has obviously no meaning with stepper motor.

Keep in mind that the **POSITION** value can be automatically saved in EEPROM when the power shuts down, and restored at power-up. See **OPTIONS** (0x2C) register for these details.

### **List of Homing methods**

---

0. Actual position is correct, don't alter.
1. Actual position is home, set new **INPUT**.
2. Move - to the first index.
3. Move + to the first index.
4. Move - to the mechanical limit.
5. Move + to the mechanical limit.
6. Move - to the mechanical limit with index.
7. Move + to the mechanical limit with index.
8. Move - to Limit switch I.
9. Move + to Limit switch I.
10. Move - to Limit switch I with index.
11. Move + to Limit switch I with index.
12. Move of start input.
13. Move of start input to the limit I switch.

All Homing methods set **REGULATIONMODE (0x20)** to Position Control mode. When the Homing conditions are verified, the **HOMINGPOSITION** register is copied to **POSITION**, and the new **INPUT** is set with **HOMINGINPUT**.

For the Homing sequence, the user can program values for **ACCELERATION**, **TOPSPEED** and **CURRENTMAX** different to the standard "Position Control mode" ones. The user can also define in the **HOMINGOPTIONS** register the ratios (% of value of these registers) used during Homing.

When Homing is completed, the standard values are automatically restored from EEPROM.

The following table indicates which homing method can be used with the different controllers. The 4 bits corresponds to **HOMINGOPTIONS**[0-3] (0x48).

4 bits	Homing method	IPECMOT T1/T2 I2C485ECMOT	I2CDCMOT DB/SLP	I2CSTEPMOT SLP
0x0	Already home – don't change	×	×	×
0x1	Already home and set <b>INPUT</b>	×	×	×
0x2	Negative move to the first index	×		
0x3	Positive move to the first index	×		
0x4	Max current detection with negative move	×	×	
0x5	Max current detection with positive move	×	×	
0x6	Max current detection with negative move and index	×	×	
0x7	Max current detection with positive move and index	×		
0x8	Negative move to the limit I switch	×	×	
0x9	Positive move to the limit I switch	×	×	
0xA	Negative move to the limit I switch and index	×		
0xB	Positive move to the limit I switch and index	×		
0xC	Move of start input			×
0xD	Move of start input to the limit I switch			×
0xE	Unused			
0xF	Unused			

Example:

If a standard **TOPSPEED** value in Position Control mode is 100'000 pulses/sec and needs to be reduced by 50% (50'000 pulses/sec) during a Homing method, then configure to 50% the bits relevant to **TOPSPEED** in the **HOMINGOPTIONS** (0x48) register.

For brushed and brushless motor controllers, when a mechanical limit system is used, another parameter, "time", must also be configured. It represents how long the PWM output to the motor needs to remain saturated (=current max reached & torque max reached) before the home position is accepted.

Note:

During Homing no external **INPUT**, **INPUTOFFSET**, **INPUTOFFSETMEASURED** or **REGULATIONMODE** are accepted. Attempts to write these values will therefore be skipped. The only way to stop or modify Homing is to operate the **STOPHOMING** (0x4A) function.

### Homing method 0: Actual position is correct, don't alter

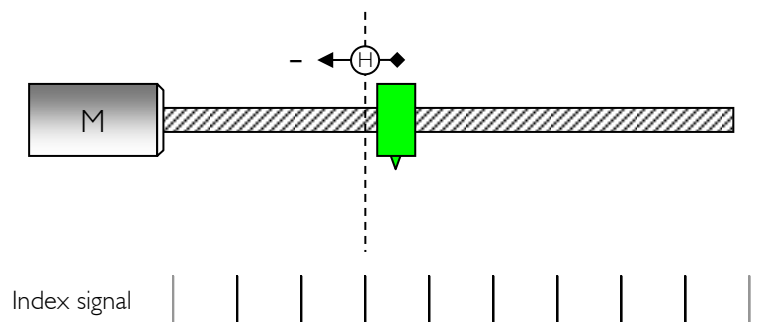
→ The actual *POSITION* is not overwritten, but copied to the *INPUT* register.

### Homing method 1: Actual position is correct, set new INPUT

→ The actual *POSITION* is not overwritten.  
 → The *HOMINGINPUT* register is copied to the *INPUT* register.

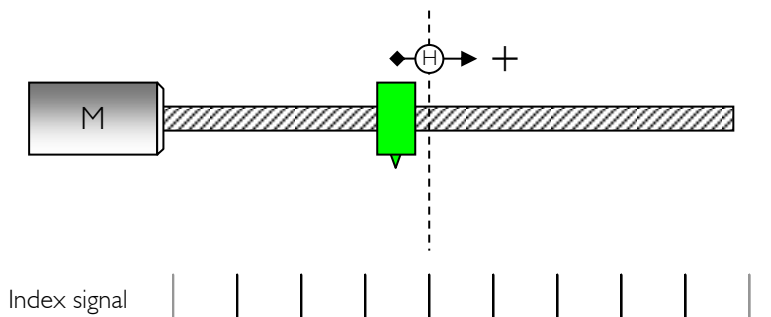
### Homing method 2: Move backward (-) to the first index

→ *HOMINGPOSITION* is copied to *POSITION* when the next index is found.  
 → The *HOMINGINPUT* register is copied to the *INPUT* register.



### Homing method 3: Move forward (+) to the first index

→ *HOMINGPOSITION* is copied to *POSITION* when the next index is found.  
 → The *HOMINGINPUT* register is copied to the *INPUT* register.



Note:

In Homing methods 2 and 3, even if the “home position” is defined (by the index), the motor will continue to move until the position/speed value has reached its goal (*INPUT*).

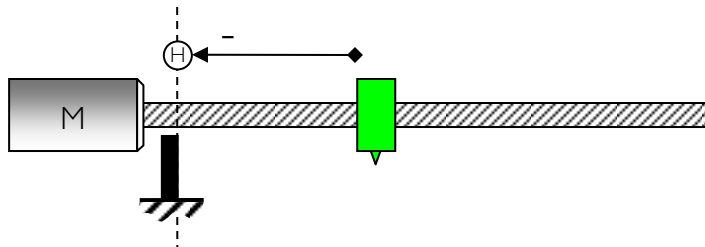
#### Homing method 4: Move backward (-) to the mechanical limit

---

When the module detects a saturated PWM output (torque value control) for more than the time specified in the *HOMINGOPTIONS* register, home is validated:

- ➔ The *HOMINGPOSITION* is copied to the *POSITION* register.
- ➔ The *HOMINGINPUT* is copied to the *INPUT* register.

The (user-definable) value of *CURRENTMAX* and its corresponding ratio (used during the Homing sequence) are very important for this method. Please note that a high current (torque) can destroy mechanical parts of the system (e.g. gears).



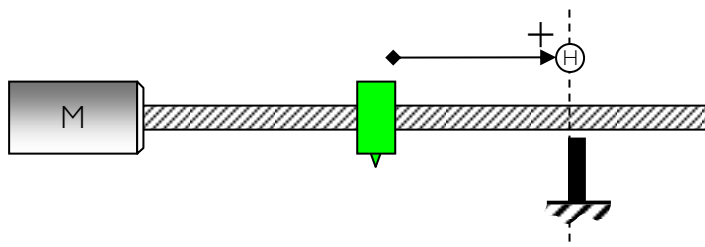
#### Homing method 5: Move forward (+) to the mechanical limit

---

When the module detects a saturated PWM output (torque value control) for more than the specified time with *HOMINGOPTIONS*, home is validated:

- ➔ The *HOMINGPOSITION* is copied to the *POSITION* register.
- ➔ The *HOMINGINPUT* is copied to the *INPUT* register.

The (user-definable) value of *CURRENTMAX* and its corresponding ratio (used during the Homing sequence) are very important for this method. Please note that a high current (torque) can destroy mechanical parts of the system (e.g. gears).

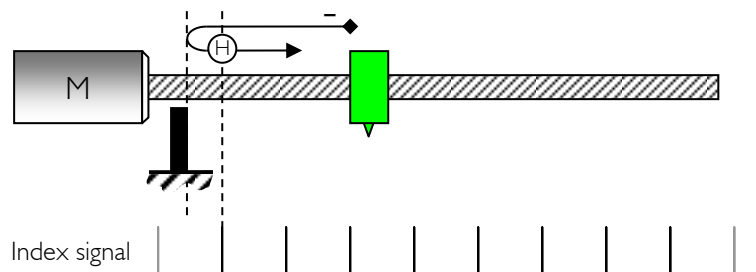


### **Homing method 6: Move backward (-) to a mechanical limit and index**

When the module detects a saturated PWM output for more than the specified time with **HOMINGOPTIONS**, it changes direction (forward) until it finds the index, then home is validated:

- **HOMINGPOSITION** is copied to the **POSITION** register.
- **HOMINGINPUT** is copied to the **INPUT** register.

**CURRENTMAX** and its corresponding Homing ratio are very important for this method. Please note that the high current (torque) can destroy mechanical parts of the system (e.g. gears).

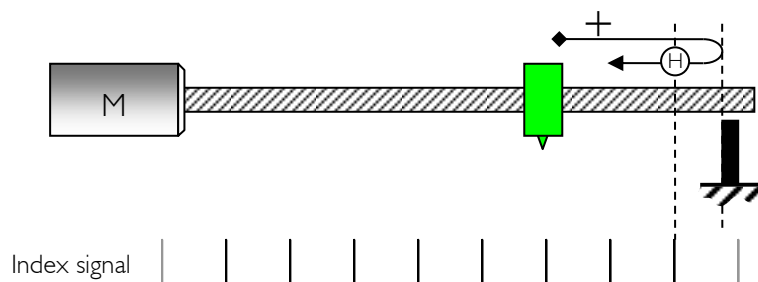


### **Homing method 7: Move forward (+) to a mechanical limit and index**

When the module detects a saturated PWM output for more than the specified time with **HOMINGOPTIONS**, it changes direction (backward) until it finds the index, then home is validated:

- **HOMINGPOSITION** is copied to the **POSITION** register.
- **HOMINGINPUT** is copied to the **INPUT** register.

**CURRENTMAX** and its corresponding Homing ratio are very important for this method. Please note that the high current (torque) can destroy mechanical parts of the system (e.g. gears).

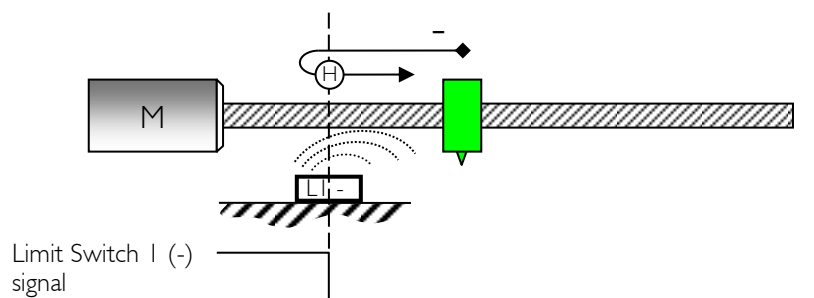


### Homing method 8: Move backward (-) to limit switch I

When the module detects the active state of limit switch I, it changes direction (forward) until the limit activity ends, home is then validated:

- **HOMINGPOSITION** is copied to the **POSITION** register.
- **HOMINGINPUT** is copied to the **INPUT** register.

During Homing with Limit Switch activity, the standard actions defined by **LIMIT1SETUP** are suspended.

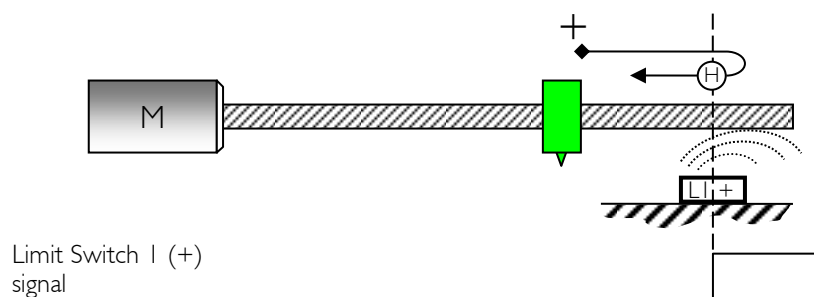


### Homing method 9: Move forward (+) to Limit Switch I

When the module detects the active state of Limit Switch I, it changes direction (backward) until the limit activity ends, home is then validated:

- **HOMINGPOSITION** is copied to the **POSITION** register.
- **HOMINGINPUT** is copied to the **INPUT** register.

During Homing with Limit Switch activity, the standard actions defined by **LIMIT1SETUP** are suspended.

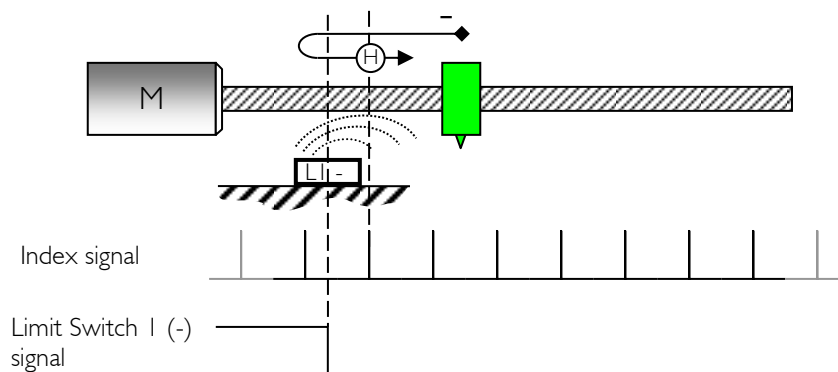


### Homing method 10: Move backward (-) to Limit Switch 1 and index

When the module detects the active state of Limit Switch 1, it changes direction (forward) until the Limit activity ends. With the first index, home is validated:

- ➔ **HOMINGPOSITION** is copied to the **POSITION** register.
- ➔ **HOMINGINPUT** is copied to the **INPUT** register.

During Homing with Limit Switch activity, the standard actions defined by **LIMITSETUP** are suspended.

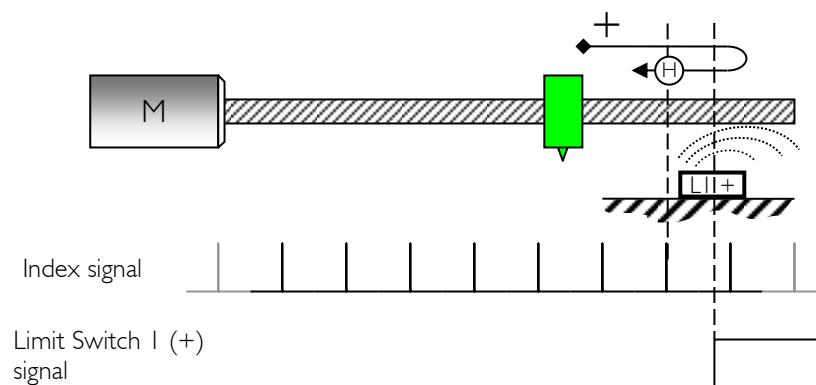


### Homing method 11: Move forward (+) to Limit Switch 1 and index

When the module detects the active state of Limit Switch 1, it changes direction (backward) until the Limit activity ends. With the first index, home is validated:

- ➔ **HOMINGPOSITION** is copied to the **POSITION** register.
- ➔ **HOMINGINPUT** is copied to the **INPUT** register.

During Homing with Limit Switch activity, the standard actions defined by **LIMITSETUP** are suspended.



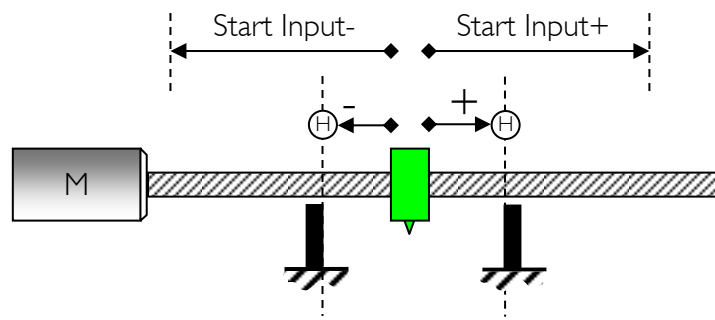


## Homing method 12: Move of Start Input

(\*) For stepper motor controllers only

The module moves of **HOMINGSTARTINPUT** (0x47) to the mechanical limit. When the mechanical limit is reached, the motor will continue until **HOMINGSTARTINPUT** is reached. The sign of **HOMINGSTARTINPUT** indicates whether to move in the positive direction or the negative one. When **HOMINGSTARTINPUT** is reached, the homing is validated.

- ➔ **HOMINGPOSITION** is copied to the **POSITION** register.
- ➔ **HOMINGINPUT** is copied to the **INPUT** register.



As shown in the above figure, if the position for the motor is unknown, the **HOMINGSTARTINPUT** should be at least equal to the mechanical range of the motor to be sure to have reached the mechanical limit.

### Homing method I3: Move of Start Input to the Limit Switch I

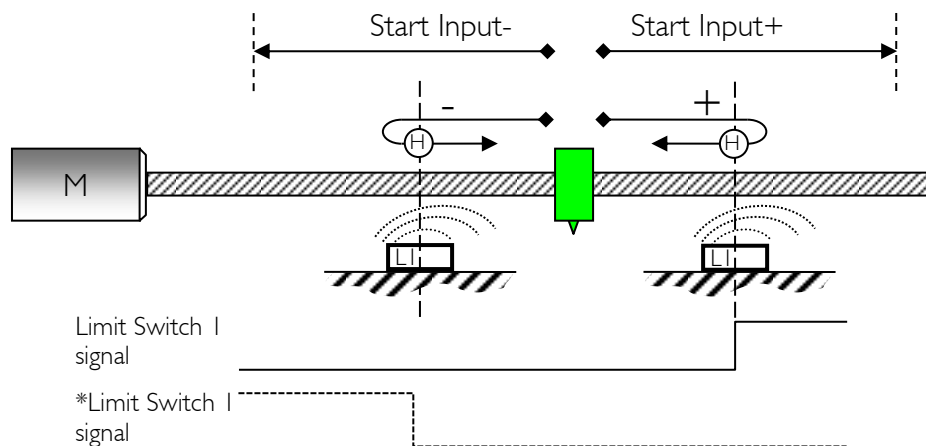
(\*) For stepper motor controllers only

The module moves of **HOMINGSTARTINPUT** (0x47) to the Limit Switch I. When the Limit Switch I is reached, it changes direction (opposite) until the Limit activity ends, home is validated.

The sign of **HOMINGSTARTINPUT** indicates whether to move in the positive direction or the negative one. If **HOMINGSTARTINPUT** is reached before reaching the Limit Switch I, home is not validated.

→ **HOMINGPOSITION** is copied to the **POSITION** register.

→ **HOMINGINPUT** is copied to the **INPUT** register.



\* Alternative for Limit I sensor

As shown in the above figure, if the position for the motor is unknown, the **HOMINGSTARTINPUT** should be at least equal to the mechanical range of the motor to be sure to have reached the limit.

## 14. Loops management

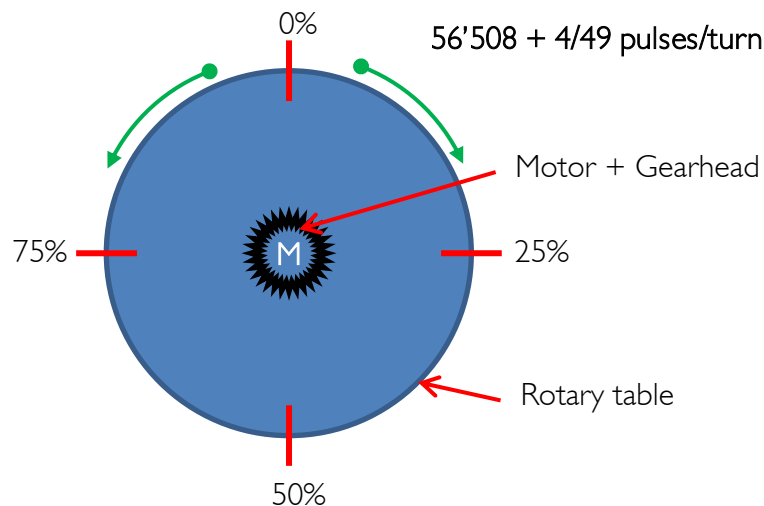
This feature helps the application engineer to have a precise positioning of his system even when using a gearhead on top of the motor. Most often, the couple motor-gearhead does not have an integer number of pulses per turn. Therefore the error on one turn will add up, resulting in imprecision on the final position.

The loops configuration feature takes into account non integer gearhead ratio as long as the user knows the exact fractional reduction of the gearhead. This information can be obtained from the gearhead manufacturer.

The Loops management automatically updates the **INPUT** (0x21) and **POSITION** (0x26) registers as often as needed.

### Overview

Here is an example where Loops mode can be used. The application is a rotary table connected directly to the motor device through a gearhead. The gearhead reduction is given approximately to 14:1, the exact ratio is  $676/49$  (manufacturer data). The motor has  $4'096$  pulses per revolution, which gives  $56'508 + 4/49$  pulses per revolution of the table.



Without the Loops management, after 12.25 turns the table will be shifted one pulse away from its original position. If the system goes on forever, major misalignment will appear.

In Loops mode the correction is made automatically when the incrementing remainder is larger than 1. In the example above, one pulse is added every 12 or 13 turns. A number of features are available in this mode and they will be explained in this chapter.

## Loops configuration

---

Three parameters have to be defined, the number of pulses per turn (*LOOPSCONFIG.[0-31]*), the numerator (*LOOPSCONFIG.[32-63]*) and the denominator (*LOOPSCONFIG.[64-95]*) of the fractional pulse left. In the past example of the rotary table, the pulses per turn are 56'508, the numerator is 4 and the denominator is 49. **The numerator must always be smaller than the denominator.**

## Loops mode

---

The Loops mode is chosen with setting the appropriate value in the LoopsCounter (*ENHANCEDINPUTS.[32-39]*). The LoopsInput (*ENHANCEDINPUTS.[0-31]*) can be defined in pulses or in percent of one full turn. If *ENHANCEDINPUTS.44* = 0, LoopsInput is in pulses and is written on 4 bytes.

If *ENHANCEDINPUTS.44* = 1, LoopsInput is in percent of one full turn and is written on 3 bytes, *ENHANCEDINPUTS.[24-31]* are not used. For example 0x00 FF FF FF = 99.9999%, 0x00 80 00 00 = 50%, 0x00 40 00 00 = 25%, etc.

## Loops

---

$-99$  (0x9D)  $\leq$  LoopsCounter  $\leq$   $+99$  (0x63)

Standard mode, takes the LoopsInput (*ENHANCEDINPUTS.[0-31]*) and LoopsCounter (*ENHANCEDINPUTS.[32-39]*) as inputs. In this mode, the rotary table will make a number of turn equal to what is set in LoopsCounter, and it will stop at the LoopsInput position.

## Shortest Way

---

LoopsCounter = 101 (0x65)

The table will move towards LoopsInput position with taking the shortest path between its actual position and the goal. If the rotary table is at the position 0, and we ask it to move to 75% of one turn, it will turn anti-clockwise towards 75%. Since the LoopsCounter (*ENHANCEDINPUTS.[32-39]*) is used to configure this mode and the following ones, it cannot be used for another purpose.

## 1<sup>st</sup> positive position

---

LoopsCounter = 102 (0x66)

The table will move towards LoopsInput position going in the positive direction.

### 1<sup>st</sup> negative position

---

LoopsCounter = 103 (0x67)

The table will move towards LoopsInput position going in the negative direction.

### Same direction, if speed = 0 then shortest way

---

LoopsCounter = 104 (0x68)

The table will move towards LoopsInput position going in the same direction as it is currently running. If the motor is still, then the shortest way will be taken.

### Same direction, if speed = 0 then 1<sup>st</sup> positive position

---

LoopsCounter = 105 (0x69)

The table will move towards LoopsInput position going in the same direction as it is currently running. If the motor is still, then the LoopsInput will be reached by the positive direction.

### Same direction, if speed = 0 then 1<sup>st</sup> negative position

---

LoopsCounter = 106 (0x6A)

The table will move towards LoopsInput position going in the same direction as it is currently running. If the motor is still, then the LoopsInput will be reached by the negative direction.

### Brake and at speed = 0 set actual position in input

---

LoopsCounter = 107 (0x6B)

The motor will brake using the **ACCELERATION** parameter. Then when it is still, the actual position will be copied in LoopsInput hence the table will not move anymore.

### Take offset input

---

LoopsCounter = 108 (0x6C)

Adds the LoopsInput to the actual **INPUT**, therefore the motor will move of an offset equal to LoopsInput.

### Take offset measured input

---

LoopsCounter = 109 (0x6D)

Adds the **POSITION** to the LoopsInput and performs the Loops management (**INPUT = POSITION + LoopsInput**).

This feature is useful when the user wants to stop the table from turning without accessing the **POSITION** register.

For example, the table is turning at a certain speed and at one point, the table needs to be stopped at 25% of one full turn away from the actual position.

### Infinite positive loops with speed defined by input

---

LoopsCounter = 110 (0x6E)

The table will turn indefinitely in the positive direction at the speed defined in LoopsInput. Although the synchronisation is maintained within the actual turn.

This feature is useful when the user wants to have a constant rotary movement that can be interrupted by an external event at any time. When this event occurs, the application can set the LoopsInput register with one of the mode explained above to stop the rotary movement at the desired position.

### Infinite negative loops with speed defined by input

---

LoopsCounter = 111 (0x6F)

The table will turn indefinitely in the negative direction at the speed defined in LoopsInput.

## Loops Options and Status

---

To validate and send a new Loops command, *ENHANCEDINPUTS.47* has to be set to '1'.

A Loops command is already running if *ENHANCEDINPUTS.46* is equal to '1'. If a wrong LoopsCounter (*ENHANCEDINPUTS.[32-39]*) is sent, *ENHANCEDINPUTS.45* is equal to '1' (LoopsCounter range is [-100;111]).

LoopsInput can be defined in pulses or in percent of one full turn. By setting *ENHANCEDINPUTS.44*, LoopsInput will be in percent, otherwise in pulses.

If *ENHANCEDINPUTS.43* is equal to '1', the peak current management is enabled. Refer to the following section for more information.

## Peak current management

---

Peak current management is useful when the user wants a burst of current at the beginning of the loops movement, hence during the acceleration phase. This feature can be enabled by setting the bit *ENHANCEDINPUTS.[43]*.

The configuration for the peak current is in register *ENHANCEDPARAMS*. *ENHANCEDPARAMS.[0-15]* is the peak current value in 1.1 (unsigned) fixed point notation (ex: 5.75 A = 0x05C0). *ENHANCEDPARAMS.[16-31]* is the peak current duration in milliseconds (ex: 2 seconds = 0x07D0).

Every time a new Loops command is sent and the peak current management is enabled, the maximum current in the motor is set to the peak current specified in the configuration and for the duration configured. After this lapse of time, the maximum current is set back the current specified in the register *CURRENTMAX* (0x2A).

## Using Loops mode example

---

Let's take the same rotary table as the overview of this chapter. The step is to configure the number of pulses per turn (*LOOPSCONFIG.[0-31]*), the numerator (*LOOPSCONFIG.[32-63]*) and the denominator (*LOOPSCONFIG.[64-95]*) of the fractional pulse left. In the past example of the rotary table, the pulses per turn are 56'508, the numerator is 4 and the denominator is 49.

- **LOOPSCONFIG** (0x4E) = 0x00 00 00 31 00 00 00 04 00 00 DC BC  
12 Bytes                      Denominator    Numerator           Pulses/turn

We want to have more current when during the acceleration at the beginning of the movement. Therefore we set the Peak current to be 3.25 A for 500ms (*CURRENTMAX* (0x2A) = 2.5 A). *ENHANCEDPARAMS.[0-15]* is the peak current value in 1.1 fixed point notation and *ENHANCEDPARAMS.[16-31]* is the peak current duration in milliseconds

- **ENHANCEDPARAMS** (0x4F) = 0x00 00 00 00 01 F4 03 40  
8 Bytes                                  Reserved                  Duration    PCurrent

Finally the Loops mode is configured for 54 turn in the positive direction and to stop at 25% of full turn. Therefore LoopsCounter (*ENHANCEDINPUTS.[32-39]*) = 54 = 0x36. The LoopsInput (*ENHANCEDINPUTS.[0-31]*) in percent of one full turn = 25% (3 bytes) = 0x00 40 00 00. *ENHANCEDINPUTS.44* = 1 to have the LoopsInput in percent of one full turn. *ENHANCEDINPUTS.43* = 1 to have the Peak current enabled and finally *ENHANCEDINPUTS.47* = 1 to validate the new loops command and start the movement.

- **ENHANCEDINPUTS** (0x4D) = 0x98 36 00 40 00 00  
6 Bytes                                  Options                  LoopsInput  
LCounter

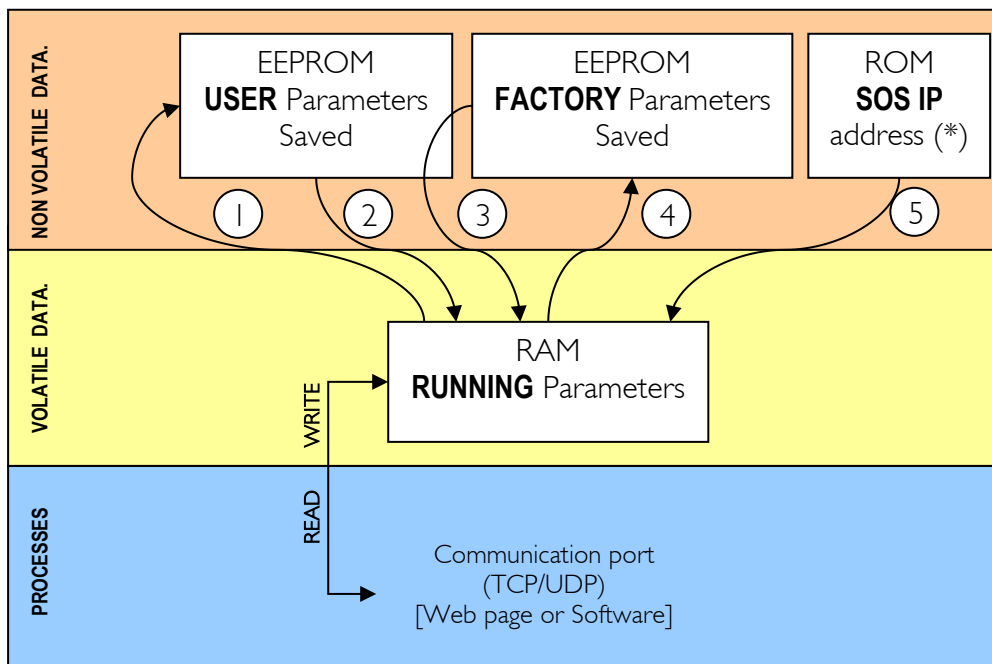
If the LoopsInput were in pulses (56'508\*25% = 14127 pulses = 0x372F) and not in percent, the command would be:

- **ENHANCEDINPUTS** (0x4D) = 0x88 36 00 00 37 2F  
6 Bytes                                  Options                  LoopsInput  
LCounter

## 15. Register management

### Memory organization

The user needs to know that a new register value sent through the communication port is loaded to the running parameters in RAM and used for the current process. All these parameters are lost at power-down. It is necessary to save them to "User Parameters" or "Factory Parameters" with the corresponding function.



Action number and description:

- ① **SAVEUSERPARAMETERS** (0x03) function
- ② During standard power-up or calling **RESTOREUSERPARAMETERS** (0x04) function
- ③ **RESTOREFACTORYPARAMETERS** (0x05) function
- ④ + ① **SAVEFACTORYPARAMETERS** (0x06) function [for integrator engineers only]
- ⑤ By pressing "SOS Button(\*)" after power-up
- ③ + ⑤ + ① By pressing "SOS Button(\*)" during power-up

(\*) For FMod-IPECMOT 48/10 T1 & T2 only



## Full description of registers

---

### List of registers

---

Address	Bytes	Name	#Page
<b>General Information</b>			
0x00	4	<i>TYPE</i>	120
0x01	4	<i>VERSION</i>	121
0x02	0 (fct)	<i>RESETCPU</i>	122
0x03	0 (fct)	<i>SAVEUSERPARAMETERS</i>	123
0x04	0 (fct)	<i>RESTOREUSERPARAMETERS</i>	124
0x05	0 (fct)	<i>RESTOREFACTORYPARAMETERS</i>	125
0x06	0 (fct)	<i>SAVEFACTORYPARAMETERS</i>	126
0x07	4	<i>VOLTAGE</i>	127
0x08	4	<i>WARNING</i>	128
0x0B (11)	4	<i>NBPOWERUP</i>	130
0x0C (12)	4	<i>TIMEINSERVICE</i>	131
0x0D (13)	2	<i>STANDBYTIMER</i>	132
<b>Communication</b>			
0x10 (16)	4	<i>COMOPTIONS</i>	133
0x11 (17)	6	<i>ETHERNETMAC</i>	134
0x12 (18)	4	<i>IPADDRESS / I2CADDRESS</i>	135
0x13 (19)	4	<i>SUBNETMASK</i>	136
0x14 (20)	1	<i>TCPTIMEOUT</i>	137
0x15 (21)	16	<i>DEVICENAME</i>	138
0x1A (26)	1	<i>TCPCONNECTIONSOPENED</i>	139
<b>General configuration</b>			
0x20 (32)	1	<i>REGULATIONMODE</i>	140
0x21 (33)	4	<i>INPUT</i>	142
0x22 (34)	4	<i>INPUTOFFSET</i>	144
0x23 (35)	4	<i>INPUTOFFSETMEASURED</i>	145
0x24 (36)	4	<i>INPUTMIN</i>	146
0x25 (37)	4	<i>INPUTMAX</i>	147
0x26 (38)	4	<i>POSITION</i>	148
0x27 (39)	4	<i>POSITIONOFFSET</i>	149
0x28 (40)	4	<i>SPEED</i>	150
0x29 (41)	4	<i>TEMPERATURE</i>	152
0x2A (42)	4	<i>CURRENTMAX</i>	153
0x2B (43)	6	<i>CURRENTSENSE</i>	155
0x2C (44)	4	<i>OPTIONS</i>	156
0x2D (45)	1	<i>LOOPTIME</i>	159
0x2E (46)	4	<i>OUTPUTVOLTAGEMAX</i>	160
0x2F (47)	4	<i>DISSIPATIONVOLTAGE</i>	161

List of registers (continued):

Address	Bytes	Name	#Page
<b>PID group</b>			
0x30 (48)	4	<i>DESIRED</i>	162
0x31 (49)	4	<i>FEEDBACK</i>	163
0x32 (50)	4	<i>COMMAND</i>	164
0x33 (51)	4	<i>KP</i>	165
0x34 (52)	4	<i>KI</i>	166
0x35 (53)	4	<i>KD</i>	167
0x36 (54)	4	<i>ANTIRESETWINDUP</i>	168
0x37 (55)	4	<i>INTEGRALDELTA</i>	169
0x38 (56)	4	<i>DERIVATIONOFDELTA</i>	170
0x39 (57)	1	<i>AUTO-TUNING</i>	171
0x3A (58)	2	<i>SKIPPULSEESTIMER</i>	172
<b>Trajectory group</b>			
0x3B (59)	4	<i>TRACKPOSITION</i>	176
0x3C (59)	4	<i>KPPTRACK</i>	176
0x3D (59)	4	<i>TRACKMAXSPD</i>	176
0x40 (64)	4	<i>ACCELERATION</i>	176
0x41 (65)	4	<i>DECELERATION</i>	177
0x42 (66)	4	<i>TOPSPEED</i>	178
0x43 (67)	4	<i>DEADZONE</i>	179
0x44 (68)	4	<i>ENCODERSRATIO</i>	180
<b>Homing group</b>			
0x47 (71)	4	<i>HOMINGSTARTINPUT</i>	182
0x48 (72)	4	<i>HOMINGOPTIONS</i>	183
0x49 (73)	0 (fct)	<i>HOMING</i>	186
0x4A (74)	0 (fct)	<i>STOPHOMING</i>	187
0x4B (75)	4	<i>HOMINGPOSITION</i>	188
0x4C (76)	4	<i>HOMINGINPUT</i>	189
<b>Loops group</b>			
0x46 (70)	2	<i>LOOPSREST</i>	181
0x4D (77)	6	<i>ENHANCEDINPUTS</i>	190
0x4E (78)	12	<i>LOOPSCONFIG</i>	192
0x4F (79)	8	<i>ENHANCEDPARAMS</i>	193
<b>Limits group</b>			
0x50 (80)	4	<i>LIMIT1SETUP</i>	194
0x51 (81)	1	<i>LIMIT1REGULATIONMODE</i>	195
0x52 (82)	4	<i>LIMIT1POSITION</i>	196
0x53 (83)	4	<i>LIMIT1XINPUT</i>	197
0x55 (85)	2	<i>IOCFG</i>	198
0x56 (86)	4	<i>IOSTATE</i>	199
0x58 (88)	4	<i>LIMIT2SETUP</i>	200

0x59 (89)	1	<i>LIMIT2REGULATIONMODE</i>	201
0x5A (90)	4	<i>LIMIT2POSITION</i>	202
0x5B (91)	4	<i>LIMIT2XINPUT</i>	203

## General configuration 2

0x60 (96)	4	<i>DISSIPTEMPERATURE</i>	204
-----------	---	--------------------------	-----

## Feed forward group

0x61 (97)	4	<i>VFFOFFSET</i>	205
-----------	---	------------------	-----

0x62 (98)	4	<i>KVFF</i>	206
-----------	---	-------------	-----

0x63 (99)	4	<i>KAFF</i>	207
-----------	---	-------------	-----

0x64 (100)	4	<i>KDFF</i>	208
------------	---	-------------	-----

Register name	FM <sub>od</sub> -IPECMOT T2	FM <sub>od</sub> -IPECMOT T1	FM <sub>od</sub> -I2C485ECMOT	FM <sub>od</sub> -I2CDCMOT DB	FM <sub>od</sub> -I2CDCMOT SLP	FM <sub>od</sub> -I2CSTEPMOT SLP
<b>General information</b>						
TYPE	●	●	●	●	●	●
VERSION	●	●	●	●	●	●
RESETCPU	●	●	●	●	●	●
SAVEUSERPARAMETERS	●	●	●	●	●	●
RESTOREUSERPARAMETERS	●	●	●	●	●	●
RESTOREFACTORYPARAMETERS	●	●	●	●	●	●
SAVEFACTORYPARAMETERS	●	●	●	●	●	●
VOLTAGE	●	●	●	●	●	●
WARNING	●	●	●	●	●	●
NBPOWERUP	●	●	●	●	●	●
TIMEINSERVICE	●	●	●	●	●	●
STANDBYTIMER					●	●
<b>Communication</b>						
COMOPTIONS	●	●	●	●	●	●
ETHERNETMAC	●	●				
IPADDRESS	●	●	●	●	●	●
I2CADDRESS						
SUBNETMASK	●	●				
TCPTIMEOUT	●	●				
DEVICENAME	●	●	●	●	●	●
TCPCONNECTIONSOPENED	●	●				
<b>General configuration</b>						
REGULATIONMODE	●	●	●	●	●	●
INPUT	●	●	●	●	●	●
INPUTOFFSET	●	●	●	●	●	●
INPUTOFFSETMEASURED	●	●	●	●	●	●
INPUTMIN	●	●	●	●	●	●
INPUTMAX	●	●	●	●	●	●
POSITION	●	●	●	●	●	●
POSITIONOFFSET	●	●	●	●	●	●
SPEED	●	●	●	●	●	●
TEMPERATURE	●	●	●			
CURRENTMAX	●	●	●	●	●	●
OPTIONS	●	●	●	●	●	●
LOOPTIME	●	●	●	●	●	
OUTPUTVOLTAGEMAX	●	●	●	●	●	
DISSIPATIONVOLTAGE	●		●			

Register name	FM <sub>od</sub> -IPECMOT T2	FM <sub>od</sub> -IPECMOT T1	FM <sub>od</sub> -I2C485ECMOT	FM <sub>od</sub> -I2CDCMOT DB	FM <sub>od</sub> -I2CDCMOT SLP	FM <sub>od</sub> -I2CSTEPMOT SLP
<b>PID group</b>						
DESIRED	●	●	●	●	●	
FEEDBACK	●	●	●	●	●	
COMMAND	●	●	●	●	●	
KP	●	●	●	●	●	
KI	●	●	●	●	●	
KD	●	●	●	●	●	
ANTIRESETWINDUP	●	●	●	●	●	
INTEGRALDELTA	●	●	●	●	●	
DERIVATIONOFDELTA	●	●	●	●	●	
AUTO-TUNING	●	●	●	●	●	
SKIPPULSETIMER					●	
<b>Feed Forward group</b>						
VOFFSET	●	●	●	●	●	
KVFF	●	●	●	●	●	
KAFF	●	●	●	●	●	
KDFF	●	●	●	●	●	
<b>Trajectory group</b>						
TRACKPOSITION	●	●	●	●	●	●
KPTRACK	●	●	●	●	●	●
TRACKMAXSPD	●	●	●	●	●	●
ACCELERATION	●	●	●	●	●	●
DECELERATION	●	●	●	●	●	●
TOPSPEED	●	●	●	●	●	●
DEADZONE	●	●	●	●	●	
ENCODERSRATIO	●					
<b>Homing group</b>						
HOMINGSTARTINPUT						●
HOMINGOPTIONS	●	●	●	●	●	●
HOMING	●	●	●	●	●	●
STOPHOMING	●	●	●	●	●	●
HOMINGPOSITION	●	●	●	●	●	●
HOMINGINPUT	●	●	●	●	●	●
<b>Loops group</b>						
LOOPSREST						●
ENHANCEDINPUTS	●	●	●	●	●	●
LOOPSCONFIG	●	●	●	●	●	●
ENHANCEDPARAMS	●	●	●	●	●	●
<b>Limits group</b>						
LIMIT1SETUP	●	●	●	●	●	●

LIMIT1REGULATIONMODE	●	●	●	●	●	●
LIMIT1POSITION	●	●	●	●	●	●
LIMIT1XINPUT	●	●	●	●	●	●
IOCFG	●	●	●	●	●	
IOSTATE	●	●	●	●	●	
LIMIT2SETUP	●	●	●	●	●	●
LIMIT2REGULATIONMODE	●	●	●	●	●	●
LIMIT2POSITION	●	●	●	●	●	●
LIMIT2XINPUT	●	●	●	●	●	●

## TYPE

---

Register Address	Register Name	Function	Read/Write control
0x00	<i>TYPE</i>	Product ID	Read only

Register Size	Register structure	
4 Bytes	Unsigned Int 16 bits (HH-HL) <i>TYPE</i>	Unsigned Int 16 bits (LH-LL) <i>MODEL</i>

### Description:

Product identifier composed of a *Type* and *Model* number.

Defines the type of peripheral.

Normally different *TYPE* modules are not software compatible.

### Example:

Device with *TYPE* = 0x00060001 means *Type*=6 (6 = EC Motor driver),  
*Model* = 1.

### Limits:

None

### Active:

Each time the processor is running



## VERSION

---

Register Address	Register Name	Function	Read/Write control
0x01	<b>VERSION</b>	Software ID	Read only

Register Size	Register structure	
4 Bytes	2bytes Hardware version (HH-HL)	2 bytes Firmware version (LH-LL)

### Description:

Hardware identifier composed of a **Version** and **Revision** number.

Firmware identifier composed of a **Version** and **Revision** number.

Normally same **Version** with different **Revision** is backward compatible.

### Example:

**VERSION** 0x0108050E

Hardware = 0x0108 = Version 1.8

Firmware = 0x050E = Version 5.14

Firmware 5.14 = **Version** 5, **Revision** 14 (0x0E) is compatible with all earlier revisions of the same version (ver 5.0 to 5.14) but has new functionalities (deactivated by default) or code optimizations.

### Limits:

None

### Active:

Each time the processor is running

## RESET CPU

---

Function Address	Function Name	Function	Read/Write control
0x02	<i>RESETCPU</i>	Restart processor	Write only

Register Size	Register structure	Unit
0 Byte	none	none

**Description:**

Reboots the card. Communication will be lost.

**Active:**

Each time the processor is running.

## SAVE USER PARAMETERS

Function Address	Function Name	Function	Read/Write control
0x03	SAVEUSERPARAMETERS	Saves all in EEPROM	Write only

Register Size	Register structure	Unit
0 Byte	none	none

### Description:

Saves the following parameters to EEPROM user space:

0x0D	STANDBYTIMER (*)	0x48	HOMINGOPTIONS
0x10	COMOPTIONS	0x4B	HOMINGPOSITION
0x12	IPADDRESS/I2CADDRESS	0x4C	HOMINGINPUT
0x13	SUBNETMASK (*)	0x4E	LOOPSCONFIG
0x14	TCPTIMEOUT (*)	0x4F	ENHANCEDPARAMS
0x15	DEVICENAME	0x50	LIMIT1SETUP
0x24	INPUTMIN	0x51	LIMIT1REGULATIONMODE
0x25	INPUTMAX	0x52	LIMIT1POSITION
0x2A	CURRENTMAX	0x53	LIMIT1XINPUT
0x2C	OPTIONS	0x55	LIMIT1XINPUT (*)
0x2D	LOOPTIME (*)	0x58	LIMIT2SETUP
0x2E	OUTPUTVOLTAGEMAX (*)	0x59	LIMIT2REGULATIONMODE
0x33	KP (*)	0x5A	LIMIT2POSITION
0x34	KI (*)	0x5B	LIMIT2XINPUT
0x35	KD (*)	0x61	LIMIT2XINPUT (*)
0x36	ANTIRESETWINDUP (*)	0x62	LIMIT2XINPUT (*)
0x3A	SKIPPULSESTIMER (*)	0x63	LIMIT2XINPUT (*)
0x40	ACCELERATION	0x64	LIMIT2XINPUT (*)
0x41	DECELERATION		
0x42	TOPSPEED		
0x43	DEADZONE (*)		
0x44	ENCODERSRATIO (*)		
0x47	HOMINGSTARTINPUT (*)		

(\*) *Not active on all controllers*

### Active:

Each time the processor is running.

For safety purposes, set **REGULATIONMODE** to Brake mode or driver open mode before running this function.

Do not change any of these parameters while saving!

(\*) *In FMod-I2CDCMOT SLP 48/1 and I2CSTEPMOT SLP 35/1 & 35/0.1 user and factory parameters are the same.*

## RESTORE USER PARAMETERS

---

Function Address	Function Name	Function	Read/Write control
0x04	<b>RESTOREUSERPARAMETERS</b>	Restores saved values	Write only

Register Size	Register Structure	Unit
0 Byte	none	none

### Description:

Restores the user parameters from EEPROM.

See **SAVEUSERPARAMETERS** (0x03) register list.

### Active:

Each time the processor is running.

For safety purposes, set **REGULATIONMODE** to Brake mode or driver open mode before running this function.

*(\*) In FMod-I2CDCMOT SLP 48/1 and I2CSTEPMOT SLP 35/1 & 35/0.1 user and factory parameters are the same.*

## RESTORE FACTORY PARAMETERS

---

Function Address	Function Name	Function	Read/Write control
0x05	<b>RESTOREFACTORYPARAMETERS</b>	Factory default	Write only

Register Size	Register Structure	Unit
0 Byte	none	none

### Description:

Restores factory parameters.

See **SAVEUSERPARAMETERS** (0x03) register list.

### Active:

Each time the processor is running,

**SAVEUSERPARAMETERS** has to be run after setting this function so that the next reboot will retain the same parameters.

For safety purposes, set **REGULATIONMODE** to Brake mode or driver open mode before running this function.

*(\*) In FMod-I2CDCMOT SLP 48/1 and I2CSTEPMOT SLP 35/1 & 35/0.1 user and factory parameters are the same.*

## SAVE FACTORY PARAMETERS

---

Function Address	Function Name	Function	Read/Write control
0x06	<b>SAVEFACTORYPARAMETERS</b>	Saves factory default	Write only

Register Size	Register Structure	Unit
0 Byte	none	none

### Description:

This function is reserved for integrator engineers, and not for the end user. Used when all parameters have been approved for an application. It saves in EEPROM all configurable registers for both factory parameters and user parameters.

This function already includes the **SAVEUSERPARAMETERS** function.

See **SAVEUSERPARAMETERS** (0x03) register list.

### Active:

Each time the processor is running.

For safety purposes, set **REGULATIONMODE** to Brake mode or driver open mode before running this function.

Do not change any of these parameters while saving!

*(\*) In FMod-I2CDCMOT SLP 48/1 and I2CSTEPMOT SLP 35/1 & 35/0.1 user and factory parameters are the same.*

## VOLTAGE

---

Register Address	Register Name	Function	Read/Write Control
0x07	<b>VOLTAGE</b>	Power module voltage	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	Volt

### Description:

Input Voltage

### Limits:

Max        0x7FFFFFFx = 32'767.996

Min        0x000000xx = 0.0

Step       0x000001xx = 0.004

### Example:

When read 0x00234567 = 2'311'527 , Voltage = 35.27 (2'311'527/65'536)

### Information:

Refer to register **REGULATIONMODE** (0x20) to have more information on the voltage limitations influencing the **REGULATIONMODE**.

### Active:

Each time the processor is running.

## WARNING

---

Register address	Register Name	Function	Read/Write Control
0x08	<b>WARNING</b>	Bit to bit state	R/W

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32 bits , each bit independent	none

### Description:

Each information/warning/error is made up of 2 bits: the first one shows the current state, the next one shows whether this state has appeared previously.

Only the bits that show the past states can be cleared by writing 0x00000000 to the **WARNING** register.

### Bits when set

- Warnings.0** Enable pin is not activated. **REGULATIONMODE** is set to **Brake** mode.
- Warnings.1** Minimum once in the past the Enable pin was not activated (if exists).  
 (\*) *Not implemented in FMod-I2CDCMOT DB 48/1.5 & SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*
- Warnings.2** Under-voltage of the power input.
- Warnings.3** Previously active, it can be cleared by user.
- Warnings.4** Over-voltage of the power input.
- Warnings.5** Previously active, it can be cleared by user.
- Warnings.6** Speed/position mode only, when the **INPUT+/- DEADZONE** has not been reached (**DEADZONE** is 0 for FMod-I2CSTEPMOT SLP 35/1 & 35/0.1).
- Warnings.7** Previously active, it can be cleared by user.
- Warnings.8** Absolute value of **COMMAND** register is saturated to 0x0000FFFF (65535)
- Warnings.9** Previously active, it can be cleared by user.  
 (\*) *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*
- Warnings.10** While **HOMING** is running.
- Warnings.11** If home has not been found.
- Warnings.12** Current specified in **CURRENTMAX** is reached, it stops increasing the integrator if the **OPTIONS.9** bit is '1'. This bit is not valid when **REGULATIONMODE** set to 0="brake", or 1="driver open".
- Warnings.13** Previously active, it can be cleared by user.  
 (\*) *Not implemented in FMod-I2CDCMOT DB 48/1.5 & SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*
- Warnings.14** Limit1 pin (if present) reaches its action state.
- Warnings.15** Previously active, it can be cleared by user.
- Warnings.16** Limit2+ pin (if present) reaches its action state.
- Warnings.17** Previously active, it can be cleared by user.



**Warnings.18** Over-temperature state, reduce the output current to 75% of its value when temperature > 115°C, 50% when > 120°C, 25% when > 125°C, 0% when > 130°C.

**Warnings.19** Previously active, it can be cleared by user.  
 (\*) *Not implemented in FMod-I2CDCMOT DB 48/1.5 & SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**Warnings.20** Motor current ripple reaches **CURRENMAX** (0x2A). Hardware power limitation is reached: increase **CURRENMAX**, reduce **ACCELERATION**, else regulation could not give achieve its best result.

**Warnings.21** Previously active, it can be cleared by user.

**Warnings.22** Device is currently dissipating.

**Warnings.23** Previously active, it can be cleared by user.  
 (\*) *Not implemented in FMod-IPECMOT 48/10 T1, FMod-I2CDCMOT DB 48/1.5 & SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**Warnings.24** Fault on motor driver.

**Warnings.25** Previously active, it can be cleared by user.  
 (\*) *Not implemented in FMod-I2CDCMOT DB 48/1.5 & SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**Warnings.26** Driver with encoder Index line only, state of Index line (refresh each LoopTime). If duration of index line is shorter than LoopTime, some Index value would be skipped (missing some Index). Use it at low speed only.

**Warnings.27** Previously active, it can be cleared by user.

**Warnings.28-31** Reserved

**Default: bits 31 -> 0**

0x00000000

**Active:**

Each time the processor is running.

## NB POWER UP

---

Register address	Register Name	Function	Read/Write Control
0x0B (11)	<b>NBPOWERUP</b>	Number of power up in device's life	Read only

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32 bits	none

(\*) *Since firmware version 4.0 for FMod-IPECMOT 48/10 T1*

(\*) *Since firmware version 5.0 for FMod-I2CDCMOT 48/1.5 DB*

### Description:

The number of power up is incremented each time the controller's power supply is in the specifications range. For controllers with two externally applied power supply (Logic 5V and motor supply), both of them have to be in the specifications range to increment the number of power up.

### Limits:

Min        0x00 00 00 00 = 0

Max        0xFF FF FF FF = 4'294'967'295

### Active:

Each time the processor is running

## TIME IN SERVICE

---

Register address	Register Name	Function	Read/Write Control
0x0C	<i>TIMEINSERVICE</i>	Time in service in device's life	Read only

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32 bits	Seconds

(\*) Since firmware version 4.0 for FMod-IPECMOT 48/10 TI

(\*) Since firmware version 5.0 for FMod-I2CDCMOT 48/1.5 DB

### Description:

The *TIMEINSERVICE* register is incremented every second when the controller is not in Standby mode.

### Limits:

Min 0x00 00 00 00 = 0 seconds

Max 0xFF FF FF FF = 4'294'967'295 seconds = ~136 years

### Active:

Each time the processor is running, it is saved in EEprom each 18hr (65'536 sec).

## STANDBY TIMER

---

Register address	Register Name	Function	Read/Write Control
0x0D	STANDBYTIMER	Power consumption management	R/W

Register Size	Register Structure	Unit
2 Bytes	Unsigned Int 16 bits	milliseconds

(\*) For *FMod-I2CDCMOT SLP 48/1* and *FMod-I2CSTEPMOT SLP 35/x*

### Description:

It is the time after which the controller is put in Standby mode after reaching its goal position in Position Control mode.

It starts decrementing a timer when **POSITION** is inside [**INPUT-DEADZONE**; **INPUT+DEADZONE**] (**DEADZONE** = 0 for *FMod-I2CSTEPMOT SLP 35/x*).

To use this feature, bit **OPTIONS.17** (register address is 0x2C) has to be set to '1'.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Default:

0x3E8 = 1'000 milliseconds

### Limits:

Min 0x00 00 = 0 milliseconds

Max 0x7F FF = 32'767 milliseconds = 32.767 seconds

### Active:

In position control/trajectory mode and when **OPTIONS.17** bit is set to '1'.

## COM OPTIONS

---

Register Address	Register Name	Function	Read/Write Control
0x10 (16)	<i>COMOPTIONS</i>	Communication options	Read/Write

Register Size	Register Structure	Unit
4 Bytes	32 individual bits	none

**Description:**

This register is reserved for future use.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

## ETHERNET MAC

---

Register Address	Register Name	Function	Read/Write Control
0x11 (17)	<i>ETHERNETMAC</i>	Hardware network ID	Read only

Register Size	Register Structure	Unit
6 Bytes	6 x Unsigned Bytes	none

*(\*) For TCP/IP interface controllers.*

**Description:**

A standard hardware unique identifier (worldwide) for each device on an Ethernet network.

**Note:**

If the user writes to this register, the MAC address will not be modified. This register is available only for information purposes.

## IP ADDRESS / I2C ADDRESS

---

Register Address	Register Name	Function	Read/Write Control
0x12 (18)	<i>IPADDRESS / I2CADDRESS</i>	Network ID	Read/Write

Register Size	Register Structure	Unit
4 Bytes	4 x Unsigned Bytes	none

**(\*) For TCP/IP interface controllers**

### Description:

Network identifier used for TCP/IP and UDP/IP.

The values 255 (0xFF) and 0 (0x00) are reserved for broadcast and network addresses and should not be used in this register.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Notes:

The module will change to a new IP address only when all of its communication ports are closed.

Do not forget to use the **SAVEUSERPARAMETERS** command.

### Default IP address value:

169.254.5.5

### Example:

For the IP=192.168.16.14 (0xC0, 0xA8, 0x10, 0x0E), write 0xC0A8100E to **IPADDRESS**.

**(\*) For I<sup>2</sup>C interface controllers**

### Description:

Network identifier of 7bits used for I2C bus, without R/W bit.

Only the least significant byte is used for the address. Value can be [8-119], [0x08-0x77].

Since the value is Hardware coded (I<sup>2</sup>C address lines 0-7), writing in this register will not affect the effective I<sup>2</sup>C address.

### Limits of I2C ID:

Min = 0x08 (8)

Max = 0x77 (119)

Because b'0000xxx' and b'1111xxx' are reserved for I2C specific actions.

If a wrong ID is coded, the device will automatically use its default value 0x55.

## SUBNET MASK

---

Register Address	Register Name	Function	Read/Write Control
0x13 (19)	<b>SUBNETMASK</b>	IP subnet mask	Read/Write

Register Size	Register Structure	Unit
4 Bytes	4 x Unsigned Bytes	none

(\* *For TCP/IP interface controllers*)

### Description:

Network IP subnet mask used for TCP/IP and UDP/IP.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Notes:

The module will change for a new subnet mask only when all of its communication ports are closed.

Do not forget to use the **SAVEUSERPARAMETERS** command.

If you do not want to use subnets, use the following subnet mask when IP address leftmost byte is:

>0 and <=127 : 255.0.0.0 (Class A addresses)  
 >127 and <=191 : 255.255.0.0 (Class B addresses)  
 >191 and <=223 : 255.255.255.0 (Class C addresses)

### Default value:

255.255.0.0

### Example:

For the IP=10.2.6.45 and subnet mask = 255.255.0.0:

IP address class = A → netID = 10, subNetID = 2 and hostID = 6.45



## TCP TIMEOUT

---

Register Address	Register Name	Function	Read/Write Control
0x14 (20)	<i>TCPTIMEOUT</i>	Timeout for TCP connection	Read/Write

Register Size	Register Structure	Unit
1 Byte	Unsigned Int 8 bits	sec

*(\*) For TCP/IP interface controllers*

### Description:

The TCP timeout is a value (in seconds) after which the user will be disconnected if the board has not been accessed in the meanwhile.

If the value is 0, the TCP timeout is deactivated. In this case however, if the client crashes during connection, the communication will never be closed as far as the module is concerned! Because a maximum of 4 communications are allowed at the same time on the module, one of them will be blocked. If the client crashes four times, all 4 communications will be blocked and the module will have to be reset.

The timeout for each TCP/IP connection is reloaded when there is traffic through the port.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Default value:

30

### Limitations:

Max value: 255

## DEVICE NAME

---

Register Address	Register Name	Function	Read/Write Control
0x15 (21)	<i>DEVICENAME</i>	Device's ASCII name	Read/Write

Register Size	Register Structure	Unit
16 Bytes	16 (only) x Unsigned Bytes (CHAR)	none

### Description:

Name and/or description of the device.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Example:

For the name "Hello Module"; extend to 16 Bytes the name: "Hello Module"+5x space=16 Byte.

So write 0x48656C6C 6F204D6F 64756C65 20202020.

## TCP CONNECTIONS OPENED

---

Register Address	Register Name	Function	Read/Write Control
0x1A (26)	<i>TCPCONNECTIONSOPENED</i>	Number of opened TCP connections	Read only

Register Size	Register Structure	Unit
1 Byte	Unsigned Int 8 bits	none

*(\*) For TCP/IP interface controllers*

**Description:**

Number of users connected to the card using TCP.

Value can be 0 to 4.

## REGULATION MODE

Register Address	Register Name	Function	Read/Write Control
0x20 (32)	<b>REGULATIONMODE</b>	Select regulation mode	R/W

Register Size	Register Structure	Unit
1 Byte	Unsigned Int 8 bits	none

**Description:****(\*) For all brushed and brushless motor controllers**

Selects the regulation mode from:

0x00	Brake	(brakes and stops the motor)
0x01	DriverOpen	(disconnects motor pins from ground and power supply)
0x02	OpenLoop	(input reg is directly converted to PWM outputs)
0x03	WaitMode	(continues actual output (PWM), without regulation)
0x04	SpeedControl	(acceleration to input speed, with PID algorithm)
0x05	PositionControl	(acceleration, top speed and deceleration ramps with PID)

**(\*) For FMod-I2CDCMOT SLP 48/I**

Same as above with the addition of:

0x06	StandByMode	Motor controller goes into low power mode, disables the output driver.
------	-------------	--

**(\*) For FMod-I2CSTEPMOT SLP 35/I & 35/0.1**

Selects the regulation mode from:

0x00	Brake	(brakes and stops the motor)
0x01	DriverOpen	(disconnects motor pins from ground and power supply)
0x02	Not implemented, goes in DriverOpen mode	
0x03	WaitMode	Continues actual phases actuation
0x04	SpeedControl	Acceleration to input speed
0x05	PositionControl	Acceleration, top speed and deceleration ramps
0x06	StandByMode	Motor controller goes into low power mode, disables the output driver.

**Limits:**

Min = 0

Max = 6

If **REGULATIONMODE** > 6, the motion mode is DriverOpen.

**Voltage limitations:**

**(\*) For FMod-IPECMOT 48/I0 T2**

If **VOLTAGE** is below 7.0 V, **REGULATIONMODE** is automatically set to **Brake** mode.

If **VOLTAGE** is higher than 56.0 V, **REGULATIONMODE** is automatically set to **DriverOpen** mode.

**(\*) For FMod-IPECMOT 48/10 TI and FMod-I2C485ECMOT DB 48/10**

If **VOLTAGE** is below 12.0 V, **REGULATIONMODE** is automatically set to **Brake** mode.

If **VOLTAGE** is higher than 56.0 V, **REGULATIONMODE** is automatically set to **DriverOpen** mode.

**(\*) For FMod-I2CDCMOT DB 48/1.5 and FMod-I2CDCMOT SLP 48/1**

If **VOLTAGE** is below 9.0 V, **REGULATIONMODE** is automatically set to **Brake** mode.

If **VOLTAGE** is higher than 54.0 V, **REGULATIONMODE** is automatically set to **DriverOpen** mode.

**(\*) For FMod-I2CSTEPMOT SLP 35/1 & 35/0.1**

If **VOLTAGE** is below 9.0 V, **REGULATIONMODE** is automatically set to **DriverOpen** mode.

If **VOLTAGE** is higher than 38.0 V, **REGULATIONMODE** is automatically set to **DriverOpen** mode.

#### Default:

**(\*) For brushed and brushless motor controllers**

After Power ON, **REGULATIONMODE** is in **Brake** mode.

**(\*) For stepper motor controllers**

After Power ON, **REGULATIONMODE** is in **DriverOpen** mode

**(\*) For FMod-I2CSTEPMOT SLP 35/x and FMod-I2CDCMOT SLP 48/1**

The default value mentioned above is valid except when bit **OPTIONS.18 (StandbyAtWakeUp)** is set to '1'. In that case, the regulation mode is set to **Standby** mode at power-up.

#### Active:

Each time the processor is running

## INPUT

Register Address	Register Name	Function	Read/Write Control
0x21 (33)	<i>INPUT</i>	Master register	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Variable

**Description:**

This register is the input (if needed) for every *REGULATIONMODE*.

**Active:**

When *REGULATIONMODE* = Open Loop, Position Control, Speed Control, or Wait modes.

***REGULATIONMODE* = Open Loop:**

(\*) *For brushed and brushless motor controllers*

*INPUT* unit is % of PWM

Max	0x0000FFFF = 65535	PWM = 100% (65535/65536)
Zero	0x00000000 = 0	PWM = 0% (0/65536)
Min	0xFFFF0001 = -65535	PWM = -100% (-65535/65536)

Higher and lower values are automatically saturated when converted to PWM

***REGULATIONMODE* = Position Control**

(\*) *For brushed and brushless motor controllers*

*INPUT* is the position to be reached, and the unit is "pulse".

(\*) *For stepper motor controllers*

*INPUT* is the position to be reached, and the unit is "μpulse" (1/256 of a full step).

Max	0x7FFFFFFF = 2'147'483'647 pulses or μpulses
Min	0x80000000 = -2'147'483'648 pulses or μpulses

***REGULATIONMODE* = Speed Control**

(\*) *For brushed and brushless motor controllers*

*INPUT* is the speed to be reached, and the unit is in pulses/sec.

(\*) *For stepper motor controllers*

*INPUT* is the speed to be reached, and the unit is "μpulse/sec" (1/256 of a full step/sec).

Max        0x7FFFFFFF = 2'147'483'647 pulses/s or  $\mu$ pulses/s  
Min        0x80000000 = -2'147'483'648 pulses/s or  $\mu$ pulses/s

Default:

0x00000000 = 0

## INPUT OFFSET

---

Register Address	Register Name	Function	Read/Write Control
0x22 (34)	<i>INPUTOFFSET</i>	Master register offset	Write only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Variable

### Description:

This register is added to the *INPUT* register in each regulation mode, and immediately cleared.

This register enables the modification of the *INPUT* register without knowing its value.

See *INPUTMIN* (0x24) and *INPUTMAX* (0x25) registers to define the range of the *INPUT*.

### Active:

When *HOMING* is not running, otherwise it is ignored and cleared.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647

Min        0x80000000 = -2'147'483'648

### Default:

0x00000000 = 0



## INPUT OFFSET MEASURED

---

Register Address	Register Name	Function	Read/Write control
0x23 (35)	<i>INPUTOFFSETMEASURED</i>	New <i>INPUT</i> value with local parameter	Write only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Variable

### Description:

This register is added to the measured value *POSITION* or *SPEED*, depending on the actual regulation mode. When done, *INPUTOFFSETMEASURED* is immediately cleared. The result is copied to the *INPUT* register.

This register allows for a new *INPUT* value to be set while not knowing externally the real-time position or the speed value.

See *INPUTMIN (0x24)* and *INPUTMAX (0x25)* registers to define the range of the *INPUT*.

### Active:

Only when its value is not equal to 0.

When *HOMING* is not running, otherwise it is ignored and cleared.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647

Min        0x80000000 = -2'147'483'648

### Default:

0x00000000 = 0

## INPUT MIN

---

Register Address	Register Name	Function	Read/Write Control
0x24 (36)	<i>INPUTMIN</i>	Lowest <i>INPUT</i> accepted	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Variable

### Description:

This register is a software limitation of the *INPUT (0x21)* reg.

If *INPUT* < *INPUTMIN* then *INPUT* = *INPUTMIN*.

With the complementary *INPUTMAX (0x25)* register, a range of *INPUT* value can be determined.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647

Min        0x80000000 = -2'147'483'648

### Default:

Set to the minimum 0x80000000, therefore never influences the *INPUT* value.

### Active:

Each time the processor is running, but ignored during Homing!

## INPUT MAX

---

Register Address	Register Name	Function	Read/Write Control
0x25 (37)	<i>INPUTMAX</i>	Highest <i>INPUT</i> accepted	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Variable

### Description:

This register is a software limitation of the *INPUT (0x21)* reg.

If *INPUT* > *INPUTMAX* then *INPUT* = *INPUTMAX*.

With the complementary *INPUTMIN (0x24)* register, a range of *INPUT* value can be determined.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647

Min        0x80000000 = -2'147'483'648

### Default:

Set to the maximum 0x7FFFFFFF, therefore never influences the *INPUT* value.

### Active:

Each time the processor is running, but ignored during Homing!

## POSITION

---

Register Address	Register Name	Function	Read/Write Control
0x26 (38)	<b>POSITION</b>	Signed pulses or $\mu$ pulses counter	R/W

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses or $\mu$ Pulses

### Description:

Write to this register to set a new position (calibration). See the **POSITIONOFFSET (0x27)** register for recalibration.

When read, it shows the evolution of the position.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647

Min        0x80000000 = -2'147'483'648

### Default:

After Power ON, **POSITION** is cleared (0x00000000). If **OPTIONS.6** bit is set, the **POSITION** before the last shutdown is reloaded.

### Active:

Each time the processor is running

## POSITION OFFSET

---

Register Address	Register Name	Function	Read/Write Control
0x27 (39)	<i>POSITIONOFFSET</i>	Calibrate <i>POSITION</i> reg.	Write only (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses or $\mu$ Pulses

### Description:

Write to this register to shift the *POSITION* register with an offset.

When the sum of *POSITIONOFFSET* and *POSITION* is completed, *POSITIONOFFSET* is automatically cleared.

It is preferable to write to this register instead of writing to the *POSITION* register when the motor moves, so that the pulses will never be lost.

Example: if for an application, it is necessary to reset the position after each complete rotation.

When *POSITION* > (1 complete rotation),  
write – (1 complete rotation) to *POSITIONOFFSET*,

When *POSITION* < - (1 complete rotation),  
write + (1 complete rotation) to *POSITIONOFFSET*.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647  
Min        0x80000000 = -2'147'483'648

### Active:

Each time the processor is running

## SPEED

---

Register Address	Register Name	Function	Read/Write Control
0x28 (40)	<b>SPEED</b>	Time-based pulse count	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses/sec or $\mu$ Pulses/sec

### Description:

Represents the number of pulses or  $\mu$ pulses that is counted during 1 second.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647  
 Min        0x80000000 = -2'147'483'648

### (\* For brushed and brushless motor controllers

Accuracy depends on the speed refresh rate, which depends on regulation **LOOPTIME (0x2D)**.

For example: with a refresh rate of 1000Hz, 1 pulse added or missing during the time of 1ms represents +/-1000 pulses during 1 second ( $1 \times 1000 = 1000$ ). With a slow refresh rate (**LOOPTIME**), you will have a good speed accuracy, but the regulation loop will perform slower.

### (\* For FMod-IPECMOT 48/10 T2

When using the dual encoder mode, the speed is always calculated using the encoder1 (closest to the motor).

### (\* For stepper motor controllers

Since there is no encoder, the speed is not a measured value; it is the real-time value of what is applied to the motor phases. The **LOOPTIME (0x2D)** register is not used, so it has no effect on the speed.

### Example for brushed or brushless motors:

The motor runs at a maximum speed (PWM saturated) of 123'456 pulses/s:

20Hz     : 123'456 +/-20    0.016% error (max pulses/ speed refresh)  
 200Hz   : 123'456 +/-200   0.16% error  
 2000Hz  : 123'456 +/-2000 1.6% error

**An encoder with more pulses/rotation allows for better accuracy!!**

If **OPTIONS.2 (0x2C)** bit is set, the **SPEED** has a 10x better accuracy, without changing **LOOPTIME**.

**Example for stepper motors:**

A speed of 307'200  $\mu$ pulses/sec represents 1200 full steps/sec (1 full step = 256  $\mu$ pulses). With a 200 full steps per revolution stepper motor, it will rotate at 6 turn/sec or 360 rpm.

**Active:**

Each time the processor is running

## TEMPERATURE

---

Register Address	Register Name	Function	Read/Write Control
0x29 (41)	TEMPERATURE	Power bridge °C	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	°C

(\*) *Not implemented in FMod-I2CDCMOT DB 48/1.5 & SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

### Description:

Gives the temperature of the power bridge only.

This value is used inside the device to reduce the current output at the power bridge to prevent overheating destruction.

$0 < T^{\circ} < \sim 90^{\circ}\text{C}$	Normal temperature
$90^{\circ}\text{C} < T^{\circ} < 115^{\circ}\text{C}$	Critical temperature, but functioning
$115^{\circ}\text{C} < T^{\circ} < 120^{\circ}\text{C}$	Over-temperature state, reduce the output current to 75% of its value
$120^{\circ}\text{C} < T^{\circ} < 125^{\circ}\text{C}$	Reduce the output current to 50% of its value
$125^{\circ}\text{C} < T^{\circ} < 130^{\circ}\text{C}$	Reduce the output current to 25% of its value
$130^{\circ}\text{C} < T^{\circ}$	Reduce the output current to 0% of its value

Automatic re-enabling of the power bridge to its maximum value when  $T^{\circ} < 113^{\circ}\text{C}$ .

### Limits:

Max	0x00960000 = 150°C
Min	0xFFD80000 = -40.0 °C

### Example:

Other    0x00168000 = 1474560 → 22.5°C = (1474560/65536)

### Active:

Each time the processor is running.



## CURRENT MAX

Register Address	Register Name	Function	Read/Write Control
0x2A (42)	<b>CURRENTMAX</b>	Limits output current	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	Ampere

**Description:**

Limits the output current for different reasons:

- Motor torque
- Motor heating ( $R \times I^2$ )

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

**Limits:****(\*) For FMod-IPECMOT 48/10 T2**

Max        0x000A0000 = 10.0 A (limited by **TEMPERATURE**)  
 Min        0x00000000 = 0.0 A  
 Step        0x00000640 = 0.025A

**(\*) For FMod-IPECMOT 48/10 T1 and FMod-I2C485ECMOT DB 48/10**

Max        0x000F0000 = 15.0 A (limited by **TEMPERATURE**)  
 Min        0x00000000 = 0.0 A  
 Step        0x00000640 = 0.025A

**(\*) For FMod-I2CDCMOT DB 48/1.5 & FMod-I2CDCMOT SLP 48/1**

Max        0x00020000 = 2.0 A  
 Min        0x00000000 = 0.0 A  
 Step        0x00000100 = 0.004 A

**(\*) For FMod-I2CSTEPMOT SLP 35/1**

Max        0x00018000 = 1.5 A  
 Min        0x00000000 = 0.0 A  
 Step        0x00000100 = 0.004 A

**(\*) For FMod-I2CSTEPMOT SLP 35/0.1**

Max        0x00002666 = 150 mA  
 Min        0x00000000 = 0.0 mA  
 Step        0x0000000A = 0.15 mA

**Default:**

**(\*) For FMod-IPECMOT 48/10 T1 & T2 and FMod-I2C485ECMOT DB 48/10**  
 0x00050000 = 327680, current limitation 5 A (327680/65536)

*(\*) For FMod-I2CDCMOT DB 48/1.5 & SLP 48/1*

0x00008000 = 32768 , current limitation 0.5 A (32768/65536)

*(\*) For FMod-I2CSTEPMOT SLP 35/1*

0x00008000 = 32768 , current limitation 0.5 A (32768/65536)

*(\*) For FMod-I2CSTEPMOT SLP 35/0.1*

0x0000CCC = 3276 , current limitation 0.05 A (3276/65536)

**Active:**

During every **REGULATIONMODE**. During Wait mode, **CURRENTMAX** is not refreshed.

## CURRENT SENSE

Register Address	Register Name	Function	Read/Write Control
0x2B (43)	<b>CURRENTSENSE</b>	Measurement of the current in the motor windings	Read only

Register Size	Register Structure	Unit
6 Bytes	Signed (2's cplt) Int 8 (H) + 8 bits fixed point (L) Signed (2's cplt) Int 8 (H) + 8 bits fixed point (L) Signed (2's cplt) Int 8 (H) + 8 bits fixed point (L)	Ampere

(\* *Implemented only in FMod-IPECMOT 48/10 T2*)

**Description:**

This register informs the user of the current actually flowing through the motor windings. The current can be negative during regeneration of the braking energy; the motor is acting as a generator.

**CURRENTSENSE.[0-15]** Actual value of the current through the motor windings.

**CURRENTSENSE.[16-31]** Minimum value of the current through the motor windings **since the last read** of this register (**CURRENTSENSE**).

**CURRENTSENSE.[32-47]** Maximum value of the current through the motor windings **since the last read** of this register (**CURRENTSENSE**).

**Limits:**

**CURRENTSENSE.[0-15], CURRENTSENSE.[16-31], CURRENTSENSE.[32-47]**

Max 0x0F00 = 3'840 = 15A

Min 0xF100 = -3'840 = -15A

**Active:**

Each time the processor is running

## OPTIONS

---

Register Address	Register Name	Function	Read/Write Control
0x2C (44)	<b>OPTIONS</b>	Bit to bit settings	Write (Read)

Register Size	Register Structure	Unit
4 Byte	Unsigned Int 32 bits , each bit independent	none

### Description:

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

<b>Bits</b>	<b>when set</b>
<b>Options.0</b>	Swap encoder I pulse-inputs (e.g. if Channel A & B are not correctly wired). This bit is not made to be changed on the fly. (* <i>Not implemented in FMod-I2CSTEPMOT SLP 35/I &amp; 35/0.I</i> )
<b>Options.1</b>	Invert the rotation of the motor (swap pulse-inputs and PWM-outputs). This bit is not made to be changed on the fly.
<b>Options.2</b>	Better accuracy of speed measurement (x10), calculated in (x10) more time (less real-time). (* <i>Not implemented in FMod-I2CSTEPMOT SLP 35/I &amp; 35/0.I</i> )
<b>Options.3</b>	Non-linear PID, decrease integration faster than increase it, may sometimes cancel oscillations. (* <i>Not implemented in FMod-I2CSTEPMOT SLP 35/I &amp; 35/0.I</i> )
<b>Options.4</b>	PWM to 10 bits (35kHz) instead of 9 bits (69kHz). (* <i>Not implemented in FMod-I2CSTEPMOT SLP 35/I &amp; 35/0.I</i> )
<b>Options.5</b>	Pull-Up (0=Pull-Down) resistors on the <b>Limit1Signal</b> and <b>Limit2Signal</b> pins
<b>Options.6</b>	Save <b>POSITION</b> register to EEPROM when shut down occurs ( <b>VOLTAGE</b> <7.0V)
<b>Options.7</b>	Leakage of <b>INTEGRALDELTA</b> when stable (subtract 1/256 of its value each <b>LOOPTIME</b> ) (* <i>Not implemented in FMod-I2CSTEPMOT SLP 35/I &amp; 35/0.I</i> )
<b>Options.8</b>	Use hall sensors as encoders, option for brushless motors only (* <i>Not implemented in FMod-I2CSTEPMOT SLP 35/I &amp; 35/0.I and FMod-I2CDCMOT DB 48/1.5 &amp; SLP 48/I</i> )
<b>Options.9</b>	Stops incrementing the integrator when <b>CURRENTMAX</b> is reached inside the motor. (* <i>Not implemented in FMod-I2CDCMOT DB 48/1.5 &amp; SLP 48/I and FMod-I2CSTEPMOT SLP 35/I &amp; 35/0.I</i> )
<b>Options.10</b>	Activate the use of a second encoder, for instance to counterbalance sliding on the transmission attached to the motor. (* <i>Implemented only in FMod-IPECMOT 48/10 T2</i> )

- Options.11** Swap encoder2 pulse-inputs (e.g. if Channel A & B are not correctly wired).  
(\* **Implemented only in FMod-IPECMOT 48/10 T2**)
- Options.12** Linear deadzone (in position trajectory mode only):  
**0** : inside dead zone, desired speed is 0 (once goal reached or 0 speed measured)  
**1** : inside dead zone, desired speed is linear with 1<sup>st</sup> outside deadzone position down to 0 at exact goal (INPUT). Example: DECELERATION = 1'000'000, DEADZONE= 10, when POSITION = 11, DESIRED state (0x30) = sqrt (2xDECELERATION)= 1414 pls/sec, here are some DESIRED values inside deadzone (Pos 11 = 1414, Pos 10= 1285, Pos 5 = 642, Pos 1 = 128, Pos 0 = 0)
- Options.13** 2 I/Os on address bus  
**2 most significant bits of address bus (RS485 or I2C) can be addresses or IOs.**  
**0: (default) the 2 bits are used for address selection only.**  
**1: 2 msb of address bus (RS485 or I2C) are set to 0, and 2 pins are used as IOs**  
**Theses 2 IOs are output or input, but inputs are only digital 0 or 255 level.**  
**When this bit is changed, SAVEUSERPARAMETERS and reset the device to reload correct address.**  
**Only implemented on device that do not have independent IOs.**  
 (\* **Implemented only in FMod-I2CDCMOT DB 48/1.5 and FMod-I2C485ECMOT 48/10**)
- Options.14** Reserved
- Options.15** Reserved
- Options.16** The dissipation voltage is adapted automatically; it stabilizes 3V above the actual voltage, but not accepting too abrupt changes. Therefore dissipation is active when a rapid increase of power voltage is measured, for instance when braking suddenly from a high speed. If this bit is cleared, dissipation is active when the power voltage is more than the value specified in **DISSIPATIONVOLTAGE (0x2F)** register. More information at the chapter "Dissipation" under "**FMod-IPECMOT 48/10 T1 & T2**" or "**FMod-I2C485ECMOT DB 48/10**".  
 (\* **Implemented only in FMod-IPECMOT 48/10 T2 and FMod-I2C485ECMOT DB 48/10**)
- Options.17** In position control mode, once the goal position is reached, automatically put the card into Standby mode after a time specified in **STANDBYTIMER (0x0D)** register.  
 (\* **Implemented only in FMod-I2CDCMOT SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1**)
- Options.18** Regulation mode is Standby at the start-up of the card.  
 (\* **Implemented only in FMod-I2CDCMOT SLP 48/1 and FMod-I2CSTEPMOT SLP 35/1 & 35/0.1**)
- Options.19** Synchronize the actual position with the real position of the stepper motor. If this bit is active and the actual position is 256  $\mu$ pulses, it means that the motor is physically on a full step (1  $\mu$ pulse = 1/256 of a full step). Therefore if the card goes into Standby mode, the motor should not move since it is on a full step. Refer to chapter "Position and phases synchronisation" under "**FMod-I2CSTEPMOT SLP 35/1 & 35/0.1**" for more informations.  
 (\* **Implemented only in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1**)

**Options.20** The synchronisation with bit **OPTIONS.19** is made on 1 full step, otherwise it is made on 4 full steps.  
 (\*) *Implemented only in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**Options.21** In position control mode, the input lowest byte is masked (=0) to always have the input on a full step.  
 (\*) *Implemented only in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**Options.22-31** Reserved

#### Limits:

None

#### Default:

(\*) *For FMod-IPECMOT 48/10 T2*  
 bits 31 -> 0 : 0x00, 0x00, b'00000000' , b'111111000'

(\*) *For FMod-IPECMOT 48/10 T1*  
 bits 31 -> 0 : 0x00, 0x00, b'00000000' , b'111111000'

(\*) *For FMod-I2C485ECMOT DB 48/10*  
 bits 31 -> 0 : 0x00, 0x00, b'00000000' , b'111111000'

(\*) *For FMod-I2CDCMOT DB 48/1.5*  
 bits 31 -> 0 : 0x00, 0x00, b'00000000' , b'111111100'

(\*) *For FMod-I2CDCMOT SLP 48/1*  
 bits 31 -> 0 : 0x00, 0x00, b'00000000' , b'101111100'

(\*) *For FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*  
 bits 31 -> 0 : 0x00, b'00010000', 0x00, b'00100000'

#### Active:

Each time the processor is running, **OPTIONS.3** and **OPTIONS.7** only when PID in use.

## LOOPTIME

---

Register Address	Register Name	Function	Read/Write Control
0x2D (45)	<i>LOOPTIME</i>	Regulation refresh rate	Write (Read)

Register Size	Register Structure	Unit
1 Byte	Unsigned Int 8 bits	Time

(\* ) *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

### Description:

Refreshes rate of *POSITION*, *SPEED* and regulation loops.

Value	Time	Refresh rate
0x00	50 ms	20 Hz
0x01	20 ms	50 Hz
0x02	10 ms	100 Hz
0x03	5 ms	200 Hz
0x04	2 ms	500 Hz
0x05	1 ms	1000 Hz
0x06	500 $\mu$ s	2000 Hz

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

Max: 0x06

### Example:

In general, set the *LOOPTIME* to 0x06 (2000 Hz of regulation loop). In rare cases, when the encoder of the motor is a poor precision one (less than 200 pulses per revolution) and/or motor turns at a slow speed (~200 pulses/second), the Loop Time can be increased. But first try to use the *OPTIONS.2* (Speed  $\times$ 10 interpolation), to have a smoother regulation on the speed.

### Default:

6 (500  $\mu$ s, 2000 Hz)

### Active:

Each time the processor is running.

## OUTPUT VOLTAGE MAX

---

Register Address	Register Name	Function	Read/Write Control
0x2E (46)	<b>OUTPUTVOLTAGEMAX</b>	Limits output PWM	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	Volt

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

### Description:

This register will modify the scale between the **COMMAND** reg. and the PWM output (duty cycle).

If **VOLTAGE** > **OUTPUTVOLTAGEMAX**, then the PWM duty cycle is reduced by the same factor (**VOLTAGE** / **OUTPUTVOLTAGEMAX**).

!!! Be careful to avoid heating a motor with a lower maximum voltage rather than the **voltage** of the card's power supply !!! The best solution is to select a power supply with the same voltage as the motor's recommended voltage.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max 0x7FFFFFFFxx = 32'767.996

Min 0x000000xx = 0.0, but it is strongly recommended that **OUTPUTVOLTAGEMAX** > **VOLTAGE** / 2

Step 0x000001xx = 0.004

Disabled 0xFFFFFFFF, with this value, no correction is made

### Default:

Disabled 0xFFFFFFFF

### Example:

1) Inactive

VOLTAGE = 23.5 V

**OUTPUTVOLTAGEMAX** = 24.0 V

**COMMAND** range [-65535 ... 65535]

PWM duty cycle [0 – 100%]

2) Active

VOLTAGE = 46.5 V

**OUTPUTVOLTAGEMAX** = 24.0 V

**COMMAND** range [-65535 ... 65535]

PWM duty cycle [0 – 51%]

### Information:

To write **OUTPUTVOLTAGEMAX** = 24.0V, send 0x00180000 = 1572864 (24.0 x 65536)

### Active:

Each time the processor is running.



## DISSIPATION VOLTAGE

---

Register Address	Register Name	Function	Read/Write Control
0x2F (47)	<i>DISSIPATIONVOLTAGE</i>	Sets dissipation voltage	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	Volt

*(\*) Implemented only in FMod-IPECMOT 48/10 T2 and FMod-I2C485ECMOT DB 48/10*

### Description:

This register will set the voltage at which the card will start dissipating. When a motor breaks suddenly, the voltage at the power supply pins can increase significantly and this can lead to damage of other cards connected in parallel to the same power supply, or the controller itself. The dissipation mechanism will avoid this issue, reducing the increase of voltage at the power supply pins. More information at the chapter "Dissipation" under "*FMod-IPECMOT 48/10 T1 & T2*" or "*FMod-I2C485ECMOT DB 48/10*".

### Limits:

Max        0x00330000 = 51.0  
 Min        0x000C0000 = 12.0  
 Step       0x000001xx = 0.004

### Default:

0x00330000 = 51.0

### Information:

To write *DISSIPATIONVOLTAGE* = 24.0V, send 0x00180000 = 1572864 (24.0 x 65536)

### Active:

Each time the processor is running.

## DESIRED

---

Register Address	Register Name	Function	Read/Write Control
0x30 (48)	<b>DESIRED</b>	Positive input of PID	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses/sec

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

### Description:

The calculated speed consign is copied to **DESIRED**. This is an input of the PID regulator; only for PID overview.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647

Min 0x80000000 = -2'147'483'648

### Active:

Updated with every regulation refresh, when PID is selected.

## FEEDBACK

---

Register Address	Register Name	Function	Read/Write Control
0x31 (49)	<b>FEEDBACK</b>	Negative input of PID	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses/sec

*(\*) Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

### Description:

The actual speed of the motor is copied to the **FEEDBACK** register. This is an input of the PID regulator; only for PID overview.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647

Min 0x80000000 = -2'147'483'648

### Active:

Updated with every regulation refresh when PID is selected.

## COMMAND

---

Register Address	Register Name	Function	Read/Write Control
0x32 (50)	COMMAND	Result of PID	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

### Description:

Output of PID and input of PWM driver.

When PID is used:

$$\text{COMMAND} = KP \times (\text{DESIRED-FEEDBACK}) + KI \times \text{INTEGRALDELTA} + KD \times \text{DERIVATIONOFDELTA}.$$

This register is optional, and never needed. Only for PID overview.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647

Min 0x80000000 = -2'147'483'648

### Active:

Updated with every regulation refresh when PID is selected

## KP

---

Register Address	Register Name	Function	Read/Write Control
0x33 (51)	<b>KP</b>	PID Proportional gain	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

### Description:

P-parameter in PID definition.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max 0x7FFFFFFF = 32'767.9999847

Min 0x00000000 = 0.0

Step 0x00000001 = 0.000015

### Example:

To set **KP**= 0.1234 ,

write 0x00001F97 = 8087 (0.1234 × 65536 = 8087.1424)

When read 0x12345678 = 305419896 ,

**KP** = 4660.33776 (305419896/65536)

### Default:

**KP** is between 0.001 and 50.

### Active:

Used with every regulation refresh when PID is selected.

## KI

Register Address	Register Name	Function	Read/Write Control
0x34 (52)	<i>KI</i>	PID Integral gain	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**Description:**

I-parameter in PID definition.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

**Limits:**

Max 0x7FFFFFFF = 32'767.9999847

Min 0x00000000 = 0.0

Step 0x00000001 = 0.000015

**Example:**

To set *KI* = 0.1234 ,

write 0x00001F97 = 8087 (0.1234 × 65536 = 8087.1424)

When read 0x12345678 = 305419896,

*KI* = 4660.33776 (305419896/65536)

**Default:**

*KI* is between 0.01 and 1.0.

**Active:**

Used with every regulation refresh when PID is selected.

## KD

Register Address	Register Name	Function	Read/Write Control
0x35 (53)	KD	PID derivative gain	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

**Description:**

D-parameter in PID definition.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

**Limits:**

Max 0x7FFFFFFF = 32'767.9999847

Min 0x00000000 = 0.0

Step 0x00000001 = 0.000015

**Example:**

To set **KD** = 0.1234 ,

write 0x00001F97 = 8087 (0.1234 × 65536 = 8087.1424)

When read 0x12345678 = 305419896,

**KD** = 4660.33776 (305419896/65536)

**Default:**

**KD** is not used = 0.

**Active:**

Used with every regulation refresh when PID is selected.

## ANTI-RESET WINDUP

---

Register Address	Register Name	Function	Read/Write Control
0x36 (54)	<b>ANTIRESETWINDUP</b>	Integration saturation	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	none

(\*) *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

### Description:

During PID loops, the delta (difference between **DESIRED-FEEDBACK**) is integrated **to INTEGRALDELTA**.

If the range of **INTEGRALDELTA** needs to be limited, **ANTIRESETWINDUP** will be the maximum value that **INTEGRALDELTA** can reach. **ANTIRESETWINDUP** is an absolute value.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647

Min 0x00000000 = 0

### Default:

0x7FFFFFFF. This value should not be modified.

### Active:

Used for every PID calculation, when **KI** does not equal 0.



## INTEGRAL DELTA

---

Register Address	Register Name	Function	Read/Write Control
0x37 (55)	<i>INTEGRALDELTA</i>	PID Integral result	Read normally (Write)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

### Description:

During PID loops the delta (difference between *DESIRED-FEEDBACK*) is integrated in *INTEGRALDELTA*.

If the range *INTEGRALDELTA* needs to be limited, *ANTIRESETWINDUP* will be the maximum that *INTEGRALDELTA* can reach.

*INTEGRALDELTA* is cleared when *KI* = 0x00000000 (0).

*INTEGRALDELTA x KI* is added to PID result (*COMMAND*)

This register is optional, and never needed. Only for PID overview.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647

Zero 0x00000000 = 0

Min 0x80000000 = -2'147'483'648

### Active:

Updated with every regulation refresh when PID is selected.

## DERIVATION OF DELTA

---

Register Address	Register Name	Function	Read/Write Control
0x38 (56)	<i>DERIVATIONOFDELTA</i>	PID derivative result	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

### Description:

During PID loops, the delta (difference between *DESIRED-FEEDBACK*) is derivated (delta-deltaold) to *DERIVATIONOFDELTA reg.*

*DERIVATIONOFDELTA x KD* is added to PID result (*COMMAND*).

This register is optional, and never needed. Only for PID overview.

### Limits:

Max    0x7FFFFFFF = 2'147'483'647

Zero   0x00000000 = 0

Min    0x80000000 = -2'147'483'648

### Active:

Updated with every regulation refresh when PID is selected.

## AUTO-TUNING

---

Register Address	Register Name	Function	Read/Write Control
0x39 (57)	<i>AUTO-TUNING</i>	Set PID automatically	R/W

Register Size	Register Structure	Unit
1 Byte	4 bits of function parameters (bits 4-7) + 4 bits function id (bits 0-3)	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/I & 35/0.I*)

### Description:

Before starting this feature, the motor should be disconnected from its mechanical output.

This feature helps the user to automatically find approximate correct values for PID and trajectory parameters.

When Auto-tuning is accurately completed, the following registers will be updated: *KP, KI, KD, ANTIRESETWINDUP, TOPSPEED, ACCELERATION, DECELERATION, DEADZONE, LOOPTIME and OPTIONS.*

### Case of AUTO-TUNING values :

Value	Function id	Function parameters
0x00	0= Tuning finished ok	- (0) PID only
0x10	0= Tuning finished ok	- (1) PID + predictive
0x01	1= Starts auto-tuning	- (0) PID only
0x11	1= Starts auto-tuning	- (1) PID + predictive
0x02	2= Abort auto-tuning	- (0)
0x03	3= Running	- (0)
0x04	4= Error	0 = tuning aborted
0x14		1 = no speed detected (no hall sensor detected)
0x24		2 = no speed detected (hall sensor detected)
0x34		3 = oscillations detected (hall sensor detected)
other values are reserved		

### Using it:

Write 0x01 (start auto-tuning) and wait (poll the register) until 0x00 (finished correctly) or 0x...4 (error) occurs.

Don't forget to use the *SAVEUSERPARAMETERS* (reg 0x03) function to save all the settings in EEPROM, when the settings are completed.

### Active:

When Homing function (find home position) is not running.

## SKIP PULSES TIMER

---

Register Address	Register Name	Function	Read/Write Control
0x40 (58)	<b>SKIPPULSESTIMER</b>	False pulses counting when going out of stand by mode correction	Write (Read)

Register Size	Register Structure	Unit
2 Byte	Unsigned int 16 bits	milliseconds

(\* *Implemented only in FMod-I2CDCMOT SLP 48/1*)

### Description:

In Standby mode the encoder of the motor is powered down. Then when choosing another motion control mode (e.g. Position control mode), the encoder is powered-up and, depending on the encoder type, risks to generate unwanted pulses for a certain period of time. This is the case particularly for magnetic encoders, which perform an algorithm to find its actual position when powered-up. **SKIPPULSESTIMER** register allows the user to set a time during which the incoming pulses from the encoder are not counted when powering up the encoder.

To determine the value in milliseconds, the user can test some values (e.g. 1ms, 10ms, 100ms, etc.) until the device does not count any "false" pulses when going out of Standby mode.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max        0x7FFF = 32'767 milliseconds  
 Min        0x0000 (not used)

### Default:

0x0000

### Active:

When going out of Standby mode to another motion control mode.

## TRACKPOSITION

---

Register Address	Register Name	Function	Read/Write Control
0x3B (59)	<b>TRACKPOSITION</b>	Waypoint movement	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses or $\mu$ Pulses

### Description:

With TRACKPOSITION, KPPTRACK, TRACKMAXSPD it is possible to define specific movement, with multiple (position+speed) waypoints.

The bus master (PC) write continuously (typ each 10ms) a new waypoint. This adds to functionality to synchronize multi-axes systems, when refreshing waypoints each timeslot on all drivers.

REGULATIONMODE must be set in speed mode, INPUT represents speed of the waypoint, TRACKPOSITION = position of the same waypoint.

$$\text{INPUTCopy}(\text{realtime}) = \text{INPUT} + K_{pp} * (\text{TRACKPOSITION} - \text{POSITION})$$

The position error compensation  $K_{pp} * (\text{waypoint position error})$  is maximize with  $[-\text{TRACKMAXSPD}; +\text{TRACKMAXSPD}]$ .

### Limits:

Max 0x3FFFFFFF = 1'073'741'823 (~1 Billion)  
 Min 0xC0000001 = -1'073'741'823 (~ -1 Billion)

### Example:

With KPPTRACK = 10, actual POSITION 1'000, SPEED 10'000, a new waypoint of TRACKPOSITION= 1010, INPUT = 1'200 p/s. Position compensation =  $10 * (1010 - 1000) = 100$  p/s are add to INPUT. So new speed goal is 1'300p/s , from 1'000 p/s increasing with ACCELERATION.

### Default:

Copy POSITION to TRACKPOSITION before setting KPPTRACK.

### Active:

When **REGULATIONMODE** is in Speed Control mode.

$K_{ppTrack} \neq 0$  , tracking position is enable.

(  $K_{ppTrack} = 0$  , normal speed trajectory (no error compensation))

## KPPTRACK

---

Register Address	Register Name	Function	Read/Write Control
0x3C (60)	<b>KPPTRACK</b>	Waypoint movement	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	Pulses or $\mu$ Pulses

### Description:

With TRACKPOSITION, KPPTRACK, TRACKMAXSPD it is possible to define specific movement, with multiple (position+speed) waypoints.

See TRACKPOSITION, for more info.

### Limits:

Max 0x7FFFFFFF = 32'767.9999847  
 Min 0x00000000 = 0.0  
 Step 0x00000001 = 0.000015

### Default:

Not saved in EEPROM, at reboot = 0.

### Active:

When **REGULATIONMODE** is in Speed Control mode.

KppTrack != 0 , tracking position is enable.

KppTrack = 0 to stop trackingposition , back to normal speed trajectory (no error compensation), eventually set INPUT p/s to 0 to stop movement or set REGULATIONMODE to position.

## TRACKMAXSPD

---

Register Address	Register Name	Function	Read/Write Control
0x3D (61)	<b>TRACKMAXSPD</b>	Waypoint movement	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses/s or $\mu$ Pulses/s

### Description:

With TRACKPOSITION, KPPTRACK, TRACKMAXSPD it is possible to define specific movement, with multiple (position+speed) waypoints.

See TRACKPOSITION, for more info.

### Limits:

Max 0x3FFFFFFF = 1'073'741'823 (~1 Billion)

Min 0x00000001 = 1 (p/s)

### Default:

Not saved in EEPROM, at reboot = 0.

### Active:

When **REGULATIONMODE** is in Speed Control mode.

KppTrack != 0 when tracking position is enable.

## ACCELERATION

---

Register Address	Register Name	Function	Read/Write Control
0x40 (64)	<b>ACCELERATION</b>	Speed acceleration	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulses/sec <sup>2</sup> or μPulses/sec <sup>2</sup>

### Description:

When **REGULATIONMODE** is in Position or Speed Control mode, **ACCELERATION** represents the evolution of the **DESIRED** speed over the first segment of the trajectory.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647  
Min 0x00000000 = 0

### Example:

If a motor (that has a null speed) needs to be at 10'000 Pulse/sec in 0.1 sec, write 0x000186A0 (100'000) to **ACCELERATION**. The calculation used to determine the previous value is the desired speed value divided by the time allowed/needed to perform that operation (10'000/0.1).

### Information:

**ACCELERATION** is always related to encoder1, even in dual encoder mode with **FMod-IPECMOT 48/10 T2**.

### Default:

If you are unsure, set **ACCELERATION** = maximum speed of the motor. The motor will take approximately 1 second to accelerate.

### Active:

Used when **REGULATIONMODE** is in Position or Speed Control mode.



## DECELERATION

---

Register Address	Register Name	Function	Read/Write Control
0x41 (65)	<i>DECELERATION</i>	Speed deceleration	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulse/sec <sup>2</sup> or μPulses/sec <sup>2</sup>

### Description:

When *REGULATIONMODE* is in Position Control mode only, *DECELERATION* represents the evolution of the *DESIRED* speed over the previous segment of the trajectory.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647  
Min 0x00000000 = 0

### Example:

If a motor is at *TOPSPEED* of 100'000 Pulse/sec and needs to brake to the desired *INPUT* position in 0.2 second, write 0x0007A120 (500'000) to *DECELERATION*. The calculation to obtain the previous value is the speed before braking divided by the time required to perform that operation. (100'000/0.2).

### Information:

*DECELERATION* is always related to encoder1, even in dual encoder mode with *FMod-IPECMOT 48/10 T2*.

### Default:

If you are unsure, set *DECELERATION* = *TOPSPEED*. The motor will take approximately 1 second to decelerate.

### Active:

Used when *REGULATIONMODE* is in Position Control mode.

## TOP SPEED

---

Register Address	Register Name	Function	Read/Write Control
0x42 (66)	<b>TOPSPEED</b>	Maximum speed	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulse/sec or μPulses/sec

### Description:

When **REGULATIONMODE** is in Position Control mode only, **TOPSPEED** represents the evolution of the constant speed for the middle segment of the trajectory.

If the output of the motor does not accept high speeds (e.g. gear head, or inertial energy,...) use a lower **TOPSPEED** than the maximum one specified in the motor datasheet. Otherwise write the maximum value according to your application.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max     0x7FFFFFFF = 2'147'483'647  
 Min     0x00000000 = 0

### Example:

If a motor is physically limited to 100'000 (μ)Pulses/sec, therefore its output torque will be close to 0 Nm at this speed.

This is not very useful for an application.

If the **TOPSPEED** is 50'000 (μ)Pulses/sec, the motor load can change and still the 3 predefined segments of the trajectory will be followed, never exceeding **TOPSPEED**.

### Information:

**TOPSPEED** is always related to encoder1, even in dual encoder mode with **FMod-IPECMOT 48/10 T2**.

### Default:

0x7FFFFFFF, no speed limitations.

### Active:

Used when **REGULATIONMODE** is in Position Control mode only.

## DEAD ZONE

---

Register Address	Register Name	Function	Read/Write Control
0x43 (67)	<b>DEADZONE</b>	Cancels speed zone	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulse

(\*) *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

### Description:

When **REGULATIONMODE** is in Position Control mode:

After the target position **INPUT** has been reached ( $=\text{POSITION}$ ), **DEADZONE** represents a range of positions where the trajectory regulator forces the **DESIRED** speed to 0. This activity lasts from  $\text{INPUT}-\text{DEADZONE}$  to  $\text{INPUT}+\text{DEADZONE}$ .

With smooth trajectories, no mechanical elasticity and good PID settings **DEADZONE** can set down to 1. With a null **DEADZONE**, if **DECELERATION** is too big, overshoots will appear with oscillations. Decrease **DECELERATION** or increase **DEADZONE** value.

Faster regulation refresh rates (**LOOPTIME**) improve the regulation and decrease overshoots. Speed x10 interpolation (**OPTIONS.2**) increases overshoots.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647

Min 0x00000000 = 0

### Example:

To know the actual overshoot of a system, set **DEADZONE** to 1000 pulses.

Set a new position **INPUT** (e.g. 100'000).

Check the **POSITION** when movement is completed. E.g.  $\text{POSITION}=100'017$  → overshoot of 17pulses). Write 0x00000011 (17) to **DEADZONE**.

### Default:

0x00000001 (1) when no overshoot of goal position is configured (**KP**, **KI**, **KD**, **DECELERATION**).

### Active:

Used when **REGULATIONMODE** is in Position or Speed Control mode only.

When **REGULATIONMODE** is in Position or Speed Control mode:

If  $(\text{INPUT}-\text{DEADZONE}) \leq \text{measured value (position or speed)} \geq (\text{INPUT}+\text{DEADZONE})$  then **WARNING.6** bit (input not reached) is cleared, otherwise it remains set.

## ENCODERS RATIO

---

Register Address	Register Name	Function	Read/Write Control
0x44 (68)	<i>ENCODERSRATIO</i>	Dual encoder management	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32, 2.2 fixed point notation	None

(\* *Implemented only in FMod-IPECMOT 48/10 T2*)

### Description:

It is the value of the 1<sup>st</sup> encoder divided by the 2<sup>nd</sup> encoder pulses. It means that, if an increase of 1000 pulses of the 1<sup>st</sup> encoder increases the 2<sup>nd</sup> encoder of 500 pulses, the *ENCODERSRATIO* value is equal to 2. The register is a 4 bytes unsigned int with a 2.2 fixed point notation, therefore a ratio of 2 is written 0x00020000.

Other ratios:

10.25 = 0x000A4000  
 0.92 = 0x0000EB85  
 1002.3 = 0x03EA4CCC

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

Max = 0xFFFFFFFF  $\cong$  65535.999  
 Min = 0x00000001

### Default:

0x00010000 = 1

### Active:

In position control mode and when *OPTIONS.10* is set to '1'.

## LOOPS REST

---

Register Address	Register Name	Function	Read/Write Control
0x46 (70)	<b>LOOPSREST</b>	Loops management (re)initialization	Write (Read)

Register Size	Register Structure	Unit
2 Bytes	Signed (2's complement) Int 16	$\mu$ Pulse

*(\*) Implemented only in stepper motor controllers*

### Description:

This is the rest of the movement that is automatically calculated by the Loops management mode. The user can set this value to '0' when a (re)initialization of the Loops mode wants to be made. This feature is useful only when the stepper motor controller is configured in "Position and phases synchronisation" mode (**OPTIONS.19** set to '1').

E.g. Some loops commands were sent to the controller and the user wants to start over the loops management system. If the **LOOPSREST** is not set to '0', the rest of the movement will remain the same as in the last Loops management, which would lead to a misalignment.

If **OPTIONS.19** = 0

**LOOPSREST** is unchanged

If **OPTIONS.19** = 1 and **OPTIONS.20** = 1 (sync. on 1 full-step)

**LOOPSREST** range is [-253;253] ( $\pm 1$  full-step)

If **OPTIONS.19** = 1 and **OPTIONS.20** = 0 (sync. on 4 full-steps)

**LOOPSREST** range is [-1023;1023] ( $\pm 4$  full-steps)

### Limits:

Max        0x7FFF = 32'767

Min        0x8000 = -32'768

### Default:

0x0000 = 0

### Active:

When using the Loops mode management.

## HOMING START INPUT

---

Register Address	Register Name	Function	Read/Write Control
0x47 (71)	<i>HOMINGSTARTINPUT</i>	Homing with stepper motor	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	μPulse

*(\*) Implemented only in stepper motor controllers*

### Description:

This is the movement in [μPulses] that the motor will move while performing its Homing. Since it is a signed int variable, the direction of the movement is given with the sign of the register. Refer to homing methods 12-13 under the chapter "Homing (position reference)" to have more information on the usage of this register.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max        0x7FFFFFFF = 2'147'483'647  
 Min        0x80000000 = -2'147'483'648

### Default:

0x0000002710 = 10'000

### Active:

Only during Homing.

## HOMING OPTIONS

---

Register Address	Register Name	Function	Read/Write Control
0x48 (72)	<i>HOMINGOPTIONS</i>	Bit to bit settings	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32 bits , each bit independent	none

### Description:

The homing needs to attain a specific condition (chosen among Homing methods) in order to set the **HOMINGPOSITION** home reference. It is only meaningful in Position Control mode. This register defines the Homing method and different parameters used during Homing. See Table I below to know to which controllers each homing method applies.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Bits

HomingOptions.0-3

### When set

Defines the Homing method (See the “Homing method table” below).

HomingOptions.4-6

Unused

HomingOptions.7

Auto-homing at power-up (internally calling the **HOMING** function).

HomingOptions.8-11

Defines the ratio that will be multiplied with the **CURRENTMAX** reg during **HOMING** (see the “Ratio table” below).

HomingOptions.12-15

Defines the ratio that will be multiplied with the **TOPSPEED** reg during Homing (see the “Ratio table” below).

HomingOptions.16-19

Defines the ratio that will be multiplied with the **ACCELERATION** reg during Homing (see the “Ratio table” below).

HomingOptions.20-23

Defines the output-saturated-time, in percent of 1 second (see the “Ratio table” below), in the Homing methods configured for current detection and limitation (Unused for stepper motor controllers).

HomingOptions.24-31

Unused.

Homing method table

4 bits	Homing method	FMod-IPECMOT T1/T2 FMod-I2C485ECMOT DB	FMod-I2CDCMOT DB/SLP	FMod-I2CSTEPMOT SLP
0x0	Already home – don't change	X	X	X
0x1	Already home and set <b>INPUT</b>	X	X	X
0x2	Negative move to the first index	X		
0x3	Positive move to the first index	X		
0x4	Max current detection with negative move	X	X	
0x5	Max current detection with positive move	X	X	
0x6	Max current detection with negative move and index	X	X	
0x7	Max current detection with positive move and index	X		
0x8	Negative move to the limit I switch	X	X	
0x9	Positive move to the limit I switch	X	X	
0xA	Negative move to the limit I switch and index	X		
0xB	Positive move to the limit I switch and index	X		
0xC	Move of start input			X
0xD	Move of start input to the limit I switch			X
0xE	Unused			
0xF	Unused			

Ratio table

4 bits	Corresponding ratio
0x0	2.3%
0x1	3.1%
0x2	4.7%
0x3	6.3%
0x4	9.4%
0x5	12.5%
0x6	18.8%
0x7	25%
0x8	37.5%
0x9	50%
0xA	75%
0xB	100%
0xC	150%
0xD	200%
0xE	300%
0xF	400%



**Default value of the 4 bytes:** [bits 31 -> 0]

0x00'BB'BB'00 equivalent to b0000'0000'1011'1011'1011'1011'0000'0000

Which represents :

- Homing method : Already home – don't change
- No homing at power-up
- Current max ratio : 100%
- Top Speed ratio : 100%
- Acceleration ratio : 100%
- Output saturated time : 1 sec

*(\*) For stepper motor controllers, output saturated time value is not taken into account.*

**Active:**

If **HomingOptions.7** bit is set upon power up, or every time the **HOMING** (0x49) function is called.

## HOMING

---

Function Address	Function Name	Function	Read/Write Control
0x49 (73)	<i>HOMING</i>	Starts to find home	Write only

Register Size	Register Structure	Unit
0 Bytes	none	none

### Description:

When this function is called, it starts to find a home defined by the method described in the *HOMINGOPTIONS* register.

When this function is called:

*REGULATIONMODE* is internally changed to “position control mode”.

*WARNING.10* and *WARNING.11* bits (homing, no home found) are set to 1.

*INPUT* is locked and does not accept external commands during Homing.

This function can be started automatically after power-up if the *HOMINGOPTIONS.7* bit is set.

### Active:

Each time the processor is running.

## STOP HOMING

---

Function Address	Function Name	Function	Read/Write Control
0x4A (74)	<i>STOPHOMING</i>	Stops a homing sequence	Write only

Register Size	Register Structure	Unit
0 Bytes	none	none

### Description:

A call of this function stops the Homing sequence (if running). At the end of this function, the **CURRENTMAX**, **ACCELERATION** and **TOPSPEED** will be internally reloaded with their previously saved values. **INPUT** is cleared, **REGULATIONMODE** is set to Brake mode or Driver Open mode for stepper motor controllers. **WARNING.11** bit (Homing) is cleared.

There are three possibilities to stop the Homing:

- 1) When the home is found (**REGULATIONMODE** is set to position control).
- 2) When **STOPHOMING** is called (**REGULATIONMODE** is set to Brake mode or Driver Open mode for stepper motor controllers).
- 3) Power-down the whole card.

### Active:

When Homing sequence is running, otherwise it has no effect.

## HOMING POSITION

---

Register Address	Register Name	Function	Read/Write Control
0x4B (75)	<i>HOMINGPOSITION</i>	Sets the Home reference value	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulse or $\mu$ Pulse

### Description:

Write values to this register to set a new reference position once Homing is finished. The reference position is obviously valid only in position control mode.

If *HOMINGOPTIONS (0x48)* is rightly configured and the *HOMING* function is running, *HOMINGPOSITION* is copied to the *POSITION* register when the Homing conditions are reached. **WARNING. I0** bit (no home) is cleared since a (new) home have been found.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647  
 Min 0x80000000 = -2'147'483'648

### Default:

0x00000000 = 0

### Active:

After *HOMING* function is called, when *HOMINGOPTIONS* conditions are reached.

## HOMING INPUT

---

Register Address	Register Name	Function	Read/Write Control
0x4C (76)	<i>HOMINGINPUT</i>	Sets a new goal when home is found	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulse or $\mu$ Pulse

### Description:

Write a value to this register to set a new *INPUT* when Homing is finished. Only for position control mode.

If *HOMINGOPTIONS* (0x48) is rightly configured and the *HOMING* function is running. When the Homing conditions are reached, *HOMINGINPUT* is copied into the *INPUT* register. **WARNING.11** bit (Homing) is cleared when the Homing is completed.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

Max 0x7FFFFFFF = 2'147'483'647  
 Min 0x80000000 = -2'147'483'648

### Default:

0x00000000 = 0

### Active:

After the *HOMING* function is called, when *HOMINGOPTIONS* conditions are reached.

## ENHANCED INPUTS

---

Register Address	Register Name	Function	Read/Write Control
0x4D (77)	<i>ENHANCEDINPUTS</i>	Inputs for Loops mode	Write (Read)

Register Size	Register Structure	Unit
6 Bytes	Unsigned Int 32 bits + Signed (2's complement) Int 8 bits + 8 bits independant	Variable

### Description:

Write a value to this register to set a new Loops command. For more information refer to the chapter "Loops management".

*ENHANCEDINPUTS*.[0-31] 4 bytes Loops input if Loops input is in pulses (*ENHANCEDINPUTS*.44 = 0).

*ENHANCEDINPUTS*.[0-23] 3 bytes Loops input if Loops input is in percent (*ENHANCEDINPUTS*.44 = 1). In that case *ENHANCEDINPUTS*.[24-31] are unused. E.g. Write 0x00800000 to have 50% of one full turn.

*ENHANCEDINPUTS*.[32-39] 1 byte Loops counter signed int.

*ENHANCEDINPUTS*.43 Peak current enable when set to '1'.

*ENHANCEDINPUTS*.44 Specifies if Loops input is in pulses or in percent of one full turn; when set to '1', Loops input is in percent.

*ENHANCEDINPUTS*.45 Error, Loops counter (*ENHANCEDINPUTS*.[32-39]) outside its range.

*ENHANCEDINPUTS*.47 Execute the new Loops command and force position control mode in *REGULATIONMODE* (0x20).

*ENHANCEDINPUTS*.[40,41,42,46] are reserved.

**Limits:**Loops Input in pulses (*ENHANCEDINPUTS*.[0-31])

Max 0xFFFFFFFF = 4'294'967'295 pulses

Min 0x00000000 = 0

Loops Input in percent (*ENHANCEDINPUTS*.[0-31])Max 0x xFFFFFF = 16'777'215  $\cong$  99.999%

Min 0x xx000000 = 0

Loops counter (*ENHANCEDINPUTS*.[32-39])

Max 0x6F = 111

Min 0x9D = -99

**Default:**

0x000000000000

**Active:**

Each time the processor is running.

## LOOPS CONFIG

---

Register Address	Register Name	Function	Read/Write Control
0x4E (78)	<i>LOOPSCONFIG</i>	Configuration for Loops mode	Write (Read)

Register Size	Register Structure	Unit
12 Bytes	Unsigned Int 32 bits + Unsigned Int 32 bits + Unsigned Int 32 bits	Pulse or $\mu$ Pulse

### Description:

Three parameters have to be defined for the configuration of the Loops mode: the number of pulses per turn, the numerator and the denominator of the fractional pulse left. **The numerator must always be smaller than the denominator.**

For more information refer to the chapter "Loops management".

*LOOPSCONFIG.[0-31]* Integer value of pulses per turn.

*LOOPSCONFIG.[32-63]* Numerator of the fractional part.

*LOOPSCONFIG.[64-95]* Denominator of the fractional part.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Example:

The application is a rotary table connected directly to the motor device through a gearhead. The gearhead reduction is given approximately to 14:1, the exact ratio is 676/49 (manufacturer data). The motor has 4'096 pulses per revolution, which gives 56'508 + 4/49 pulses per revolution of the table.

Therefore the values in *LOOPSCONFIG* are the following:

*LOOPSCONFIG* = 0x00 00 00 31 00 00 00 04 00 00 DC BC  
 12 Bytes                      *Denominator*      *Numerator*              *Pulses/turn*  
    = 49                              = 4                              = 56'508

### Limits:

*LOOPSCONFIG.[0-31]*, *LOOPSCONFIG.[32-63]* and *LOOPSCONFIG.[64-95]*

Max            0xFFFFFFFF = 4'294'967'295

Min            0x00000000 = 0

### Default:

0x000000000000000000000000

### Active:

When executing a Loops mode command.



## ENHANCED PARAMETERS

---

Register Address	Register Name	Function	Read/Write Control
0x4F (79)	<i>ENHANCEDPARAMS</i>	Parameters for Loops mode	Write (Read)

Register Size	Register Structure	Unit
8 Bytes	Reserved 32 bits + Unsigned Int 16 bits + Unsigned Int 16 bits	Variable

### Description:

Every time a new Loops command is sent and the peak current management is enabled (*ENHANCEDINPUTS.43* = 1), the maximum current in the motor is set to the peak current specified in *ENHANCEDPARAMS.[0-15]* and for the duration in *ENHANCEDPARAMS.[16-31]*. After this lapse of time, the maximum current is set back the current specified in the register *CURRENTMAX* (0x2A).

For more information refer to the chapter “Loops management”.

*ENHANCEDPARAMS.[0-15]* Peak current in unsigned int 1.1 fixed point notation (E.g. 2.75 A = 0x02C0)

*ENHANCEDPARAMS.[16-31]* Peak current duration in milliseconds. For 2 seconds = 2'000 ms, write 0x07D0.

*ENHANCEDPARAMS.[32-63]* Reserved

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Limits:

*ENHANCEDPARAMS.[0-15]*

Same limits defined in *CURRENTMAX* (0x2A) register, looking only at the 2 middle bytes.

*ENHANCEDPARAMS.[16-31]*

Max 0xFFFF = 65535

Min 0x0000 = 0

### Default:

0x0000000000000000

### Active:

When executing a Loops mode command and when *ENHANCEDINPUTS.43* = 1 (peak current enabled).

## LIMIT I (Home) SETUP

Register Address	Register Name	Function	Read/Write Control
0x50 (80)	<i>LIMITISETUP</i>	Configures limit n° I	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32 bits , each bit independent	none

### Bits description:

This register configures the event depending on the “Limit Is” input pin. (Refer to the chapter “12. Limit switches” for more information)

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### When set to 1:

Bit number	Bit name	Bit description
<i>LIMITISETUP .0</i>	<i>bLimitIEnable</i>	Activates Limit numberI detection
<i>LIMITISETUP .1</i>	<i>bLimitIActivHigh</i>	When clear, the active setting is low
<i>LIMITISETUP .2</i>	<i>bLimitIRegulationMode</i>	Copy <i>LIMITIREGULATIONMODE</i> to <i>REGULATIONMODE</i>
<i>LIMITISETUP .3</i>	<i>bLimitIPosition</i>	Copy <i>LIMITIPOSITION</i> to <i>POSITION</i> (once)
<i>LIMITISETUP .4</i>	<i>bLimitIIndex</i>	Not used (0), future functionality
<i>LIMITISETUP .5</i>	<i>bLimitIxInputL</i>	Defines where <i>LIMITIXINPUT</i> must be copied to
<i>LIMITISETUP .6</i>	<i>bLimitIxInputH</i>	With previous bit
<i>LIMITISETUP .5-31</i>	none	Not used (0)

### *bLimitIxInputH*, *bLimitIxInputL* action:

0 0	<i>LIMITIXINPUT</i> unused
0 1	<i>LIMITIXINPUT</i> copied to <i>INPUT</i> (each regulation loop)
1 0	<i>LIMITIXINPUT</i> copied to <i>INPUTOFFSET</i> (once)
1 1	<i>LIMITIXINPUT</i> copied to <i>INPUTOFFSETMEASURED</i> (each regulation loop)

### Example:

With a PNP proximity detector (mechanical, optical, magnetic ...), a pull-down resistor (on the limit input pin) can be selected with the **OPTIONS.5** bit. If the detector is active high, set **LIMITISETUP.1** bit.

If the detector is located at the reference position, set **LIMITISETUP.3** bit and write the reference value to the **LIMITIPOSITION** register. Activate **LIMITISETUP.0** bit to enable the activity of the Limit I.

To configure a limit, first deactivate it and then select pull-up or pull-down resistors, write only necessary registers between **LIMITIXINPUT**, **LIMITIPOSITION**, **LIMITIREGULATIONMODE**. Once this has been done, write **LIMITISETUP** with enable bit set.

### Default:

0x00000000 (deactivated)

### Active:

When Homing is not running.

## LIMIT I SET REGULATION MODE

---

Register Address	Register Name	Function	Read/Write Control
0x51 (81)	<i>LIMIT I REGULATIONMODE</i>	Limit I regulation mode	Write (Read)

Register Size	Register Structure	Unit
1 Byte	Unsigned Int 8 bits	none

### Description:

When "Limit I" has been reached, and if *LIMIT I SETUP.0* is set (enabled) and *LIMIT I SETUP.2* is set as well (new regulation mode), then *LIMIT I REGULATIONMODE* will be copied to *REGULATIONMODE*.

See *REGULATIONMODE* (0x20) for further details.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Active:

When Limit I is reached.

## LIMIT I SET POSITION

---

Register Address	Register Name	Function	Read/Write Control
0x52 (82)	<i>LIMIT I POSITION</i>	Sets a new position	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulse

### Description:

When Limit I is reached, if *LIMIT I SETUP.0* is set (enable) and *LIMIT I SETUP.3* is set (new position), then *LIMIT I POSITION* will be copied to *POSITION*.

See *POSITION (0x26)* for further details.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Active:

When Limit I is reached.

## LIMIT I SET X- INPUT

---

Register Address	Register Name	Function	Read/Write Control
0x53 (83)	<i>LIMITIXINPUT</i>	Sets a new input	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Variable

### Description:

When Limit I is reached, if *LIMITISETUP.0* is set (enable) and *LIMITISETUP.5-6* is not 00, then *LIMITIXINPUT* will be copied to *INPUT* or *INPUTOFFSET* or *INPUTOFFSETMEASURED*.

See *LIMITISETUP* (0x50) for destination description.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

### Active:

When Limit I is reached.

## I/O Config

---

Register Address	Register Name	Function	Read/Write Control
0x55 (85)	<i>IOCFG</i>	A/D measurements	Write (Read)

Register Size	Register Structure	Unit
2 Bytes	4 x 4bits (nible)	None

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*)

### Description:

IOCFG.0-3 (4bits) defines the configuration (16 states) of I/O 1  
 IOCFG.4-7 (4bits) defines the configuration (16 states) of I/O 2  
 IOCFG.8-15 are reserved.

### All 16 states:

- 0x0 pin as input only (0-5V)
- 0x1 pin as output at 5V (\* max 2mA)
- 0x2\* pin as output at 0V
- 0x3\* hardware error "live", disappears when error disappears
  - undervoltage
  - overvoltage
  - overtemperature
  - impossible hall signal 000 or 111
  - mosfet driver error
- 0x4\* hardware error "hold"
 

Same as 0x3 but a detected error is hold until next reboot or IOCfg change to "live" 0x3 and wait for no error, when the error disappears back to mode 0x4 is possible.
- 0x5\* Phases short circuit relay or motor brake
 

When driver is **off or not enough powered** an external relay can short circuit the motor power lines (with or without brake shunt resistors),
- 0x6\* Phases short circuit by internal MOSFET transistors, indicates that the driver actually short-circuit all motor power lines to 0V (e.g. after 2 seconds of REGULATIONMODE = 0 = STOP. If this signal is used to add an additional brake, take care to be able to release brake in about 1ms (max 10ms). If this 1ms is impossible, prefer to use mode 0x5.
- 0x7\* Indicates that DEADZONE value is reached (in positioning or speed REGULATIONMODE)
- 0x8-0xF reserved

### Active:

IOCfg checked each looptime. IOs output updates each looptime.

## I/O STATE

---

Register Address	Register Name	Function	Read/Write Control
0x56 (86)	<i>IOSTATE</i>	A/D measurements	Read only

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32 bits	None

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*  
*Only I/O 1 exists in FMod-IPECMOT 48/10 TI*

### Description:

I/O 1 and 2 can be used in input to measure a voltage. The voltage range is [0-5V]. Even when the IOs are in output the A/D continue to work.

*IOSTATE.[0-7]* is the A/D measurement of I/O 1 and *IOSTATE.[8-15]* is the A/D measurement of I/O2.

The values are given in a ratio of the Logic supply voltage, 0xFF = 255 is the maximum value ( $\cong 5V$ ), 0x80 = 128  $\cong 2.51V$ , 0x00 = 0 is the minimum one (0V). The equation is Voltage IO1 [V] = *IOSTATE.[0-7]*/255 \* 5 and Voltage IO2 [V] = *IOSTATE.[8-15]*/255 \* 5.

*IOSTATE.[16-31]* are reserved.

### Limits:

Max        0xFF = 255  $\cong 5V$   
 Min        0x00 = 0 = 0V

### Active:

Each time the processor is running. Each IO A/D Values is updated in ~5ms.

## LIMIT 2 SETUP

---

Register Address	Register Name	Function	Read/Write Control
0x58 (88)	<i>LIMIT2SETUP</i>	Configures limit n° 2	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Unsigned Int 32 bits , each bit independent	none

**Bits description:**

*LIMIT2SETUP* is the same register as *LIMIT1SETUP* but for Limit n° 2.  
See *LIMIT1SETUP (0x50)* for more information.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.



## LIMIT 2 SET REGULATION MODE

---

Register Address	Register Name	Function	Read/Write Control
0x59 (89)	<i>LIMIT2REGULATIONMODE</i>	Limit 2 regulation mode	Write (Read)

Register Size	Register Structure	Unit
1 Byte	Unsigned Int 8 bits	none

**Description:**

*LIMIT2REGULATIONMODE* is the same register as *LIMIT1REGULATIONMODE* but for Limit n° 2.

See *LIMIT1REGULATIONMODE (0x51)* for more information.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

## LIMIT 2 SET POSITION

---

Register Address	Register Name	Function	Read/Write Control
0x5A (90)	<i>LIMIT2POSITION</i>	Sets a new position	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Pulse

**Description:**

*LIMIT2POSITION* is the same register as *LIMIT1POSITION* but for Limit n° 2. See *LIMIT1POSITION (0x52)* for more information.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

## LIMIT 2 SET X- INPUT

---

Register Address	Register Name	Function	Read/Write Control
0x5B (91)	<i>LIMIT2XINPUT</i>	Sets a new input	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	Variable

### Description:

*LIMIT2XINPUT* is the same register as *LIMIT1XINPUT* but for Limit n° 2.  
See *LIMIT1XINPUT (0x53)* for more information.

This register is saved when calling the function *SAVEUSERPARAMETERS* or *SAVEFACTORYPARAMETERS*.

## DISSIPATION TEMPERATURE

Register Address	Register Name	Function	Read/Write Control
0x60 (96)	<i>DISSIPTEMPERATURE</i>	Estimated temperature of the dissipation transistors	Read only

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	C°

(\* *Implemented only in FMod-IPECMOT 48/10 T2*)

**Description:**

This register informs the user of the temperature of the dissipation transistors.

When *DISSIPTEMPERATURE* < 170°, the dissipation of the breaking energy can be enabled by the controller.

When *DISSIPTEMPERATURE* > 170°, the dissipation of the breaking energy is disabled to prevent damaging the controller.

For more information, refer to the chapter "Dissipation" (p.18).

**Limits:**

Max        0x00AB0000 = 171 °C

Min        0xFFD80000 = -40.0 °C

**Active:**

Each time the processor is running

## VFF OFFSET

Register Address	Register Name	Function	Read/Write Control
0x61 (97)	VFFOFFSET	Feed forward offset	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's complement) Int 32	none

(\* *Not implemented in FMod-I2CSTEP MOT SLP 35/1 & 35/0.1*

**Description:**

Offset value added to the command in feed forward regulation.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

**Limits:**

Max 0x0000FFFF = 65'535  
 Min 0xFFFF0000 = -65'536

**Default:**

VFFOFFSET = -1000.

**Active:**

Used with every regulation refresh when feed forward is selected.

## KVFF

Register Address	Register Name	Function	Read/Write Control
0x62 (98)	KVFF	Velocity feed forward gain	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 16 (HH-HL) + 16 bits fixed point (LH-LL)	none

(\*) *Not implemented in FMod-I2CSTEPMOT SLP 35/I & 35/0.I*

**Description:**

Velocity contribution of the feed forward regulation.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

**Limits:**

Max 0x7FFFFFFF = 32'767.9999847

Min 0x00000000 = 0.0

Step 0x00000001 = 0.000015

**Example:**

To set **KVFF** = 0.1234,

write 0x00001F97 = 8087 (0.1234 × 65536 = 8087.1424)

When read 0x12345678 = 305419896,

**KVFF** = 4660.33776 (305419896/65536)

**Active:**

Used with every regulation refresh when feed forward is selected.

## KAFF

Register Address	Register Name	Function	Read/Write Control
0x63 (99)	KAFF	Acceleration feed forward gain	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 8 (HH) + 24 bits fixed point (HL-LH-LL)	none

(\* ) *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

**Description:**

Acceleration contribution of the feed forward regulation.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

**Limits:**

Max 0x7FFFFFFF = 127.99999994

Min 0x00000000 = 0.0

Step 0x00000001 = 0.00000006

**Example:**

To set **KAFF** = 0.1234,

write 0x001F9724 = 2070308 (0.1234 × 16777216 = 2070308.454)

When read 0x12345678 = 305419896,

**KAFF** = 18.20444 (305419896/16777216)

**Active:**

Used with every regulation refresh when feed forward is selected.

## KDFF

---

Register Address	Register Name	Function	Read/Write Control
0x64 (100)	<i>KDFF</i>	Deceleration	Write (Read)

Register Size	Register Structure	Unit
4 Bytes	Signed (2's cplt) Int 8 (HH-HL) + 24 bits fixed point (LH-LL)	none

(\* *Not implemented in FMod-I2CSTEPMOT SLP 35/1 & 35/0.1*

### Description:

Deceleration contribution of the feed forwards regulation.

This register is saved when calling the function **SAVEUSERPARAMETERS** or **SAVEFACTORYPARAMETERS**.

### Limits:

Max 0x7FFFFFFF = 127.99999994

Min 0x00000000 = 0.0

Step 0x00000001 = 0.00000006

### Example:

To set *KDFF* = 0.1234,

write 0x001F9724 = 2070308 (0.1234 × 16777216 = 2070308.454)

When read 0x12345678 = 305419896,

*KDFF* = 18.20444 (305419896/16777216)

### Active:

Used with every regulation refresh when feed forward is selected.



**Contact address:**

FiveCo - Innovative Engineering  
En Budron H11  
CH-1052 Le Mont-sur-Lausanne  
Switzerland  
Tel: +41 21 632 60 10  
Fax: +41 21 632 60 11

[www.fiveco.com](http://www.fiveco.com)  
[info@fiveco.com](mailto:info@fiveco.com)

