

## **Documentation Author**

Frederic Lambercy for K-Team S.A.  
Rue Galilee 9  
1400 Yverdon-les-Bains  
Switzerland

email: [info@k-team.com](mailto:info@k-team.com)

Url: [www.k-team.com](http://www.k-team.com)

## TRADEMARK ACKNOWLEDGMENTS:

**IBM PC:** International Business Machine Corp.

**Macintosh:** Apple Corp.

**SUN Sparc-Station:** SUN Microsystems Corp.

**LabView:** National Instruments Corp.

**MatLab:** MathWorks Corp.

**Webots:** Cyberbotics

**Khepera:** K-Team and LAMI

## LEGAL NOTICE:

- The content of this manual is subject to change without notice.
- All effort have been made to ensure the accuracy of the content of this manual. However, should any error be detected, please inform K-Team S.A.
- The above notwithstanding K-Team can assume no responsibility for any error in this manual.

# TABLE OF CONTENTS



<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>KoreMotorLE Hardware</b>	<b>4</b>
2.1	Overview . . . . .	4
2.1.1	Dip Switch Settings . . . . .	5
2.1.2	Controllers I2C Addresses . . . . .	5
2.1.3	Motor Connection Pinout . . . . .	6
2.2	KoreMotorLE Connections . . . . .	6
2.2.1	KoreBot Connections . . . . .	6
2.2.2	Motors Supply Connector . . . . .	7
2.3	Hardware Protection . . . . .	7
2.3.1	Electrostatic Discharge Protection . . . . .	7
2.3.2	Motor Controller Fuses . . . . .	8
<b>3</b>	<b>Motor Controller</b>	<b>9</b>
3.1	Controller State or Mode . . . . .	9
3.2	Controller Status Registers . . . . .	10
3.2.1	Target Zone . . . . .	11
3.2.2	SoftStop Limits . . . . .	11
3.3	Controller Option Registers . . . . .	11
3.4	Measurements . . . . .	12
3.4.1	Position . . . . .	12
3.4.2	Speed . . . . .	13
3.4.3	Current . . . . .	13
3.5	Regulation Type . . . . .	13
3.6	Controller Speed Profile . . . . .	14
3.7	SetPoint Sources . . . . .	14
3.8	Motor Blocked Test . . . . .	15
3.8.1	Mechanical Limit Detection . . . . .	16
3.9	Sampling Period . . . . .	17
<b>A</b>	<b>Register Summary</b>	<b>18</b>

# 1 INTRODUCTION

---



The KoreMotorLE controler board can be used only as a KoreBot extension module that is able to control two DC motors. It provides an open loop interface or PID algorithm control with several complementary features such as current limitation, software position limits and internal commands generator.



### 2.1 Overview

The KoreMotorLE hardware provides two motor controllers, each one including a H-Bridge, microcontroller and motor connector. Dip switches control the KoreMotorLE address. Figures 2.1 and ?? describe the board main components.

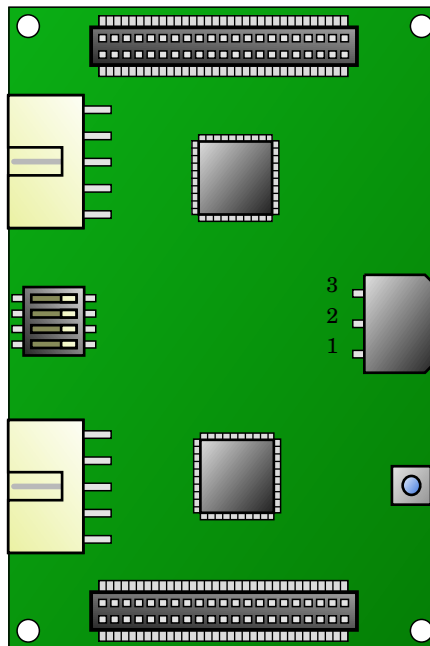


Figure 2.1: KoreMotorLE hardware overview

### 2.1.1 Dip Switch Settings

The Dip Switch is used to select the I2C address of the controller. The only running mode is I2C Standalone: That mode is used to control the board from the korebot I2C bus. The I2C bus is directly connected to the motor controllers, and the I2C addresses for each controller are detailed in section 2.1.2.

### 2.1.2 Controllers I2C Addresses

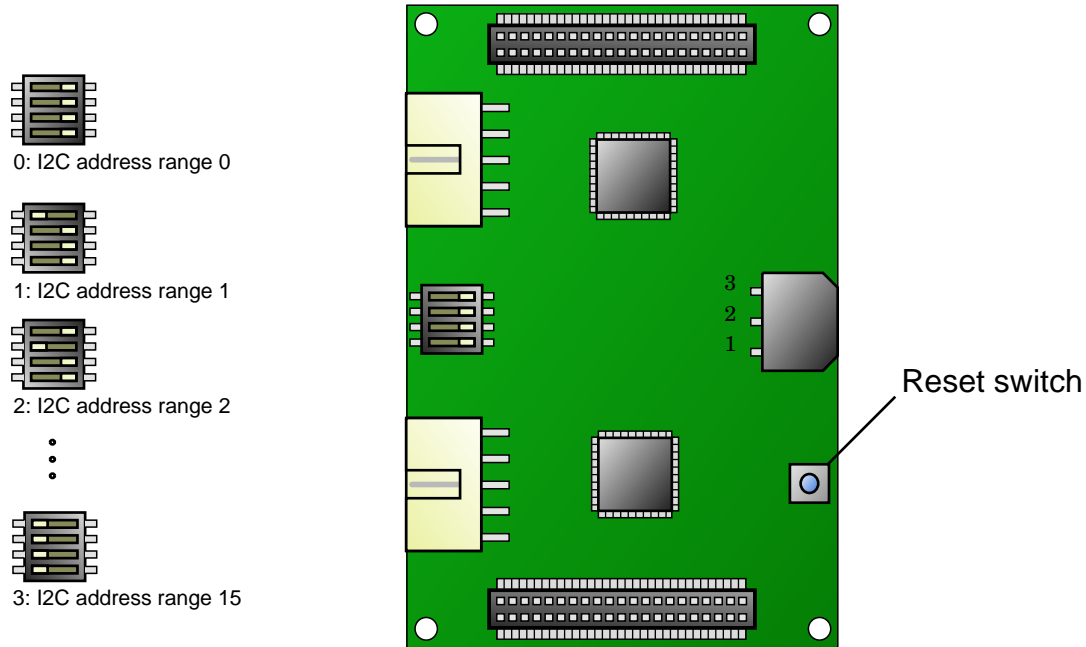


Figure 2.2: KoreMotor I2C address range

The four dip switch, as displayed on figure 2.2 is used to choose the I2C address range for the motor controllers. The address range is useful to stack up to 16 KoreMotor together, using the same I2C bus.

	Motor 0 addr	Motor 1 addr
Range 0	0x1F	0x20
Range 1	0x21	0x22
Range 2	0x23	0x24
Range 3	0x25	0x26
...		
Range 15	0x3D	0x3E

### 2.1.3 Motor Connection Pinout

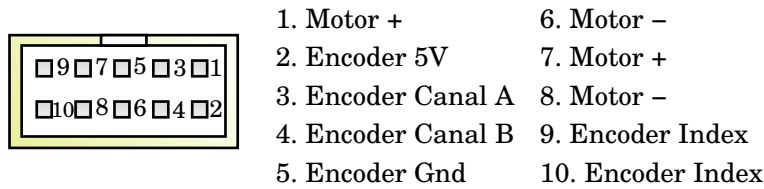


Figure 2.3: KoreMotorLE connector pinout

The KoreMotorLE provides two DC motor connectors. The motor connection may vary from a manufacturer to another, but the signals for quadrature encoders are usually similar. Please refer to your motors manufacturer datasheet for details about the motor connections, contact support@k-team.com if further help is necessary.

**Motor+** : First motor connection

**Motor-** : Second motor connection

**Encoder 5V** : Power supply for the encoder

**Encoder Gnd** : Ground reference for the encoder

**Encoder A** : First encoder phase

**Encoder B** : Second encoder phase

**Encoder Index** : Index for the encoder. This signal is not available for all encoders and it is not required.

## 2.2 KoreMotorLE Connections

The KoreMotorLE requires a power supply connection and an external interface connection. The main power supply is the motors supply, it should be set according to the motors requirements, regarding voltage and necessary current, and is common for all motors. Another supply may be necessary for the electronics on the board. This supply is provided from the KoreBot.

### 2.2.1 KoreBot Connections

The KoreBot connection is pretty straightforward, the boards should be simply stacked together. The KoreBot will provide the electronics 5V supply, and only the motors supply should be added.

## 2.2.2 Motors Supply Connector

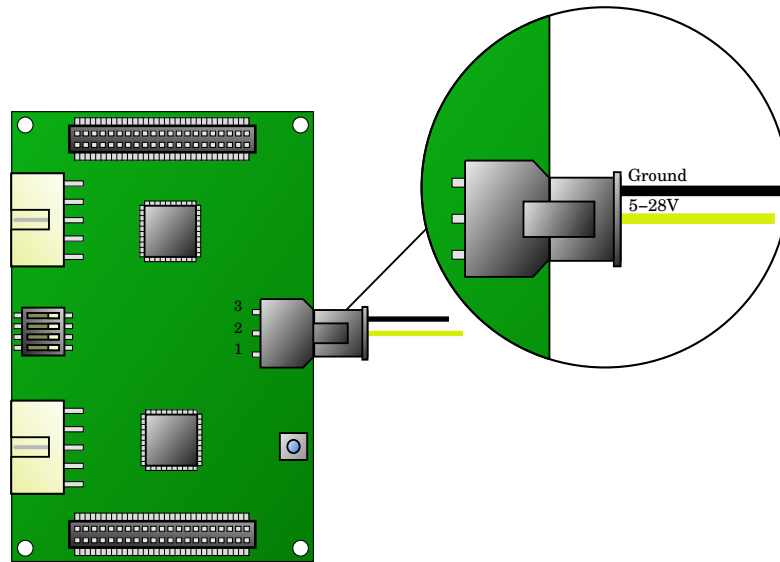


Figure 2.4: Motors power connection

The main power connector will supply all the motors, the voltage should match the motor requirements, and stay within the accepted voltage range for the H-Bridges. The connector pinout is detailed on figure 2.4.

Voltage range	5-28V
Max current per motor	2A

## 2.3 Hardware Protection

### 2.3.1 Electrostatic Discharge Protection

As any electronic device, the KoreMotorLE can be damaged by Electrostatic Discharge. The quadrature encoders interface chip on the board has been identified as very sensitive and special care should be taken to avoid any problem. It is recommended to connect the KoreMotorLE ground signal to the earth or with the robot chassis for embedded use.

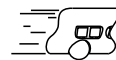
A quadrature encoder interface failure will shut the board ability to read encoder feedback and globally prevent the related motor controller proper operation.

### **2.3.2 Motor Controller Fuses**

The motor controllers are protected from overcurrent with dedicated fuses. Each motor channel is protected with a 2 Amps fuse. Higher peak currents are normally supported by the board and all its components, but continuous operations should respect these limits.

Please contact your K-Team dealer for support if a fuse replacement is required.





### 3.1 Controller State or Mode

Each motor controller status is indicated, or can be modified, using the mode register (0x28). This register should be read to check the controller state, and it can be written to switch from a mode to another. The following modes are available:

**Idle mode:** This is the default startup state (except if hardware option 0 is set). While in idle mode the controller will not execute any regulation and the motor is free wheeling.

**Control mode:** The controller must be switched to this mode for any regulation to start. When exiting Control Mode, the regulation will stop according to the new mode description.

**Stop mode:** When switched to this mode, the controller will hold the motor in blocked state (both motor pins are electrically connected).

**Sleep mode:** The controller is switched to a minimal activity status and the motors are set to free wheeling. This mode can be used to optimize power consumption.

**Reset mode:** Mode to reset the controller. The controller will then switch to the startup mode, according to hardware option 0.

**Save configuration mode:** Save all registers to EEPROM for backup. A special protection pattern must be written in register 0x28 to access this mode. Values 0x55, 0xAA and 0x05 must be written sequentially to enter mode 5. The first two values are not stored to the register and cannot be read. The controller will switch back to idle mode when the backup is completed.

**Search limit mode:** This mode will start a mechanical limit detection routine. The routine will use speed regulation to perform the test, using the current target point setting and motor blocked test (see section 3.8). Resulting 32bit positions are stored in SoftStopMin (0x4A-0x4D) and SoftStopMax (0x4E-0x51). The controller will switch back to idle mode when the test is completed.

## 3.2 Controller Status Registers

Two registers detail the motor controller status. Bit will be set by the controller in the Error register (0x2D) to signal problems. Generic information about the controller are given in the Status register (0x2E).

Some errors, from error 0 to 4, are blocking errors, that means the controller will stop until the error are not cleared. The error will be cleared only if the error condition has been resolved and after the Error register is cleared from the application, that means writing 0x0 to the error register. If a blocking error is triggered because of a physical condition, such as a blocked motor, the condition itself should be resolved before clearing the error register.

### Error register

bit0 :	Sample time too small	The defined sample time is too small for a complete calculation cycle.
bit1 :	Watchdog timer overflow	Refer to the PIC16F876 datasheet
bit2 :	Brown-out	Refer to the PIC16F876 datasheet
bit3 :	Software stoped motor	The motor position is off-limits and the <code>sw_stop_error</code> option is enabled.
bit4 :	Motor blocked	The motor blocked condition is met (see section 3.8) and the <code>sw_blocked_stop</code> option is enabled.
bit5 :	Position out of range	Not implemented
bit6 :	Speed out of range	Not implemented
bit7 :	Torque out of range	Not implemented

### Status register

bit0 :	Movement detected	Motor speed is not null
bit1 :	Direction	Movement direction 0=negative 1=positive
bit2 :	On setpoint	The regulated value match the Set-Point value
bit3 :	Near setpoint	The regulated value is inside the target zone
bit4 :	Command saturated	The PWM ratio has reached 100%
bit5 :	Antireset windup active	Set if the command is saturated and if <code>sw_windup</code> is enabled
bit6 :	Current control active	
bit7 :	Softstop active	The position is off-limits and the corresponding option is enabled (section 3.2.2)

### 3.2.1 Target Zone

The near setpoint flag is set whenever the regulated value is such as:

$$|SetPoint - RegulatedValue| \leq NearTargetMargin$$

That means the NearTargetMargin register defines a zone around the target point. which can be useful for specific applications.

### 3.2.2 SoftStop Limits

The Controller can be configured to setup virtual position limits for movements. This feature is very useful to protect mechanical devices powered by the KoreMotor. As soon as the position is under SoftStopMin (0x4A-0x4D) or over SoftStopMax (0x4E-0x51), the controller will shutdown the motor driver to prevent any further movement regardless of the target setpoint and controller mode.

The SoftStopMax limit is only active if the `sw_stopmax` option is enabled and the SoftStopMin limit is only active if the `sw_stopmin` option is enabled. Any combination of these two options is valid, and they should be set according to each specific application requirements.

## 3.3 Controller Option Registers

The controller behaviour can be configured using the two option registers (0x2A and 0x2B) where each bit will enable or disable a feature. The registers can be read at any time to retrieve the current option settings and options are activated or disabled as soon as a register is written.

### Software option register

bit0 :	sw_separated	Use alternate algorithm PID derivation. The derivate part is calculated using the process variable rather than the error.
bit1 :	sw_windup	Activate the anti reset windup routine.
bit2 :	sw_stopmin	Stop the motor if the min position is reached (section 3.2.2).
bit3 :	sw_stopmax	Stop the motor if the max position is reached (section 3.2.2).
bit4 :	sw_stop_error	Generate an error when position is out of limits, in this case the error must be re-set before any further commands can be executed.
bit5 :	sw_blocked_stop	Stop the motor if the blocked condition is met (section 3.8).
bit6 :	sw_current_ctrl	Activate software current limitation (Not implemented)
bit7 :	sw_dir_inv	Invert the motor direction.

### Hardware option register

bit0 :	hw_startup	Startup mode (0 = idle mode, 1 = control mode)
bit1 :	hw_analog_set	Use analog input for setpoint (Not Implemented)
bit2 :	hw_led	Not Implemented
bit3 :	hw_resolution	Resolution for the encoder (0 = 100%, 1 = 25%)
bit4 :	hw_torque_inv	Invert the internal current measurement
bit5 :	hw_opt1	Not Implemented
bit6 :	hw_opt2	Not Implemented
bit7 :	hw_opt3	Not Implemented

## 3.4 Measurements

Each controller can return the motor incremental position, the motor current speed, and the current through the motor. Each measurement can be retrieved reading the corresponding registers, the most significant bits for each value should always be read first to ensure data consistency.

### 3.4.1 Position

The returned motor position is an accumulated counter of encoder pulses. The physical position can be calculated knowing a reference position and the

pulse per turn value for the encoder. The position registers can be written to set the current position to a given value at any time.

### 3.4.2 Speed

The speed value is a division of a constant value by the time between encoder pulsations. In default mode (pulsation x2 and postcaler 1:4), a measure is made every two pulsations. The constant value is define by the maximum time multiplied by 256 (0xFFFF \* 256 = 16'776'960). This operation allows a better pid calculation for the lower speed.

$$MotorSpeed = \frac{16'776'960}{Timer5value}$$

To convert into a real time, use the following calculation:

$$Time = \frac{Timer5value}{\frac{f_{osc}/4}{Tmr5Prescaler}}$$

Where  $f_{osc} = 20\text{MHz}$  and  $Tmr5Prescaler = 8$  (default).

### 3.4.3 Current

The measured current (Torque registers) is proportional with the maximum supported current of the H-bridge. The max current supported by the H-bridge is 6 Amps, but the default fuse is an 2 Amps. The measured current value vary from 0 to 1024. There is no negative value, it's an absolute measure. See below the reference values, please contact K-Team for further information.

Absolute current	Measured value
0Amps	0
500mA	80
1.5A	245
3A	490
6A	980

## 3.5 Regulation Type

The controler supports six different type of regulation. Each regulation will use a specific set of PID coefficient, depending on the parameter which is actually regulated. The PID coefficients are usual proportional, derivate and integral gains for the controller, several other settings may be required for each specific mode and application.

The 32 bit SetPoint value (0x2F-0x32) is always used as the controller input. According to the regulation type, this register will set the target position, speed, or current. The actual controller internal input is only updated when the least significant byte is written (0x2F) to ensure consistency.

**Open Loop control:** The open loop mode does not use the PID controller. The SetPoint value will directly set the output PWM ratio. The PWM timer for the controller is a 10 bit timer, that means the PWM ratio can vary from +1024 to -1024 with the sign indicating the direction. A null value generates no signal, while a 1024 value generate a 100% PWM ratio (continuous signal).

**Position control:** The motor position is regulated to the SetPoint value. The position coefficients are required for the PID controller.

**Position control with speed profile:** A speed profile is used to reach the target position defined in the SetPoint registers. The speed regulation is actually used to follow the speed profile that is why speed coefficients are required for the PID controller. The position PID is not used in this mode. The speed profile is defined by the MaxSpeed register (0x6D) and Acceleration register (0x52), see section 3.6 for further details. The final position is held using the blocked motor mode in the target zone, which is defined using NearTargetMargin (0x60).

**Speed control:** The motor speed is regulated to the Setpoint value. As speed is a 16 bit measurement, no speed beyond  $\pm 2^{15}$  can be regulated. The speed coefficients are required for the PID controller.

**Speed control with acceleration profile:** The motor speed is regulated to the SetPoint value, but an acceleration ramp is used to reach the final speed. According to the Acceleration register (0x52) and current speed, the speed is gradually increased or decreased until on SetPoint.

## 3.6 Controller Speed Profile

When using position control with speed profile mode, the controller aim for a *target position* using a *speed control* PID. That means the position itself is not regulated, even though the given SetPoint is a position. On the other hand, the motor speed is regulated according to the built-in speed profile. Figure 3.1 describe the speed profile, as it can be configured using the MaxSpeed and Acceleration registers.

## 3.7 SetPoint Sources

The target point (see section 3.5) for the controller can be set from various sources. The most common use is to set the target point by writing into the SetPoint registers (0x2F-0x32) but other sources may be useful for specific applications.

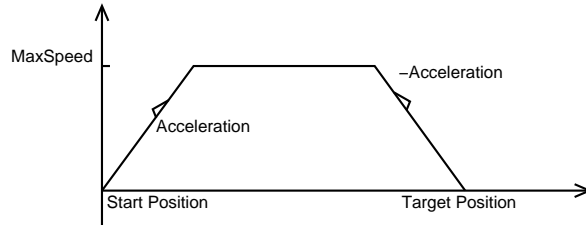


Figure 3.1: Speed Profile

The source can be modified using the SetPoint Source register (0x29). The default mode 0 requires writing to the SetPoint register, other modes will use internal generators as described bellow.

**Square Generator:** As displayed on figure 3.2, the mode 2 will generate a square wave signal for the SetPoint according to the IntGenPeriod, IntGenAmplitude and IntGenOffset registers. IntGenOffset is a signed value.

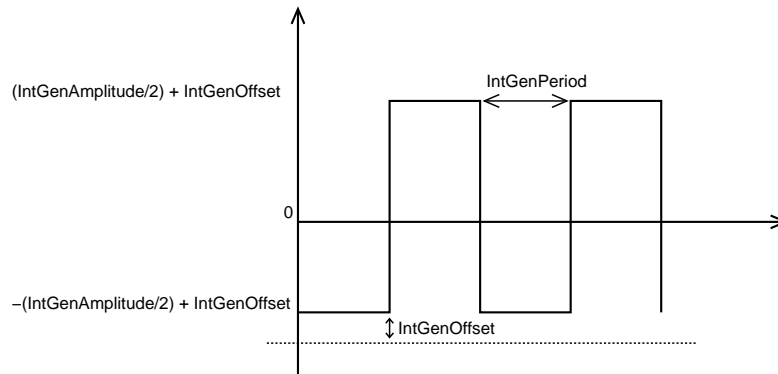


Figure 3.2: Square waveform

**Triangle Generator:** As displayed on figure 3.3, the mode 3 will generate a triangle signal for the SetPoint according to the IntGenPeriod, IntGenAmplitude and IntGenOffset registers. IntGenOffset is a signed value.

### 3.8 Motor Blocked Test

The controller uses a built-in routine, based on movement and current measurement, to detect if the motor abnormally blocked. A blocked condition is

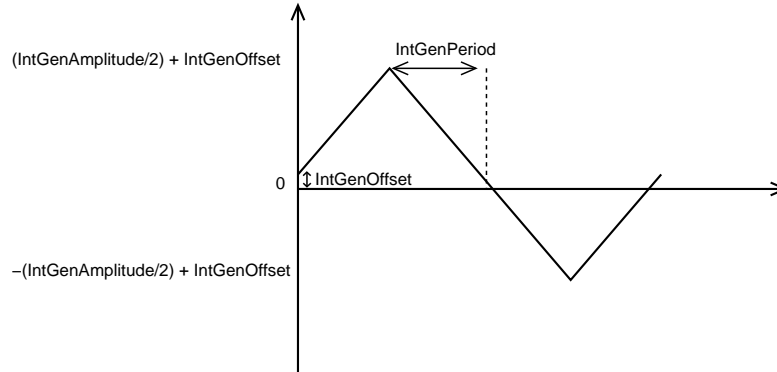


Figure 3.3: Triangle waveform

met only if the motor is not moving and if the measured current is over the defined SWCurrentLimit (0x56-0x57). The motor blocked flag and eventual error are only triggered if the both conditions are met during at least a BlockedTime (0x46) period.

That means the blocked flag will never be set if the motor is moving, even very slowly, and it will never be set if the current limit is not defined properly, according to the controlled system.

The blocked time period can be calculated from the sampling period such as:

$$BlockedPeriod = 2 * SamplingPeriod * BlockedTime$$

### 3.8.1 Mechanical Limit Detection

Controller mode 6 can be configured to use the built-in mechanical limit detection routine. The mechanical limit detection is based on the motor blocked test, that should be configured properly before switching to this mode. The routine will use the current SetPoint value as the target speed to perform the test, that is why the SetPoint register should be adjusted as well.

Once all relevant registers are set, switching to mode 6 will start the detection routine. The routine will search for mechanical limits in both directions. If no limit is reached in one direction or another, the test will fail and the routine will run continuously, driving the motor at the given SetPoint speed, until the controller mode is manually changed.

Resulting 32bits positions for mechanical limits are stored in SoftStopMin (0x4A-0x4D) and SoftStopMax (0x4E-0x51). The controller will switch back to idle mode when the test is completed.



### 3.9 Sampling Period

The controller sampling period is the time between two output calculations. That means the PWM ratio to control the motor is only updated once every sampling period.

A shorter sampling period may result in a more accurate control but requires a better encoder resolution. If the encoder resolution is too bad, the controller might not count any pulse during a sampling period thus assuming no movement, even if the motor is rotating. Moreover, if the sampling period is too small, error 0 may occur, indicating that the microcontroller is too slow to complete all the PID calculations in a single period.

On the other hand, if the sampling period is too long, the controller might not be quick enough to regulate the system, as its reaction time might become too long.

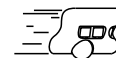
Setting a correct sampling period is critical to ensure proper motor control.

The Sampling period should be set using the `SampleTimeH` register (0x45). The internal timer provide a  $1.6\mu s$  resolution for the sampling time so that the sampling period can be calculated as:

$$SamplingPeriod = (SampleTimeH * 256) * 1.6\mu s$$

The `SampleTimeL` register (0x44) can be used for finer tuning of the Sampling period, down to a  $1.6\mu s$  resolution, but period under  $1300\mu s$  are very unlikely to be acceptable.

## A REGISTER SUMMARY



Name	Address	Description
Mode	0x28	Selection of mode (0..7) 0 = Idle Mode 1 = Normal Control Mode 2 = Stop Motor 3 = Sleep Mode 4 = Reset Mode 5 = Save configuration parameters (page 0) in E2PROM 6 = Search Limit Mode 7 = Unused
SetPointSource	0x29	Source of SetPoint (0..7) 0 = External I2C 1 = External Analogic (not implemented) 2 = Internal Square Wave Generator 3 = Internal Triangle Generator 4 = Internal Sinus Generator (not implemented) 5 = Not used 6 = Not used 7 = Not used
HW_Options	0x2A	Hardware options (Flags) Bit0: Define Startup Mode, 0=Idle Mode / 1=Normal Control Mode Bit1: Analog SetPoint Input, 0=disabled / 1=enabled Bit2: LED, 0=disabled / 1=enabled Bit3: Encoder resolution, 0=100% / 1=25% Bit4: Torque Inversion Bit5: Driver Option 1 Bit6: Driver Option 2 Bit7: Driver Option 3
SW_Options	0x2B	Software options (Flags) Bit0: Seperate D, 0=disabled / 1=enabled Bit1: Antireset Windup, 0=disabled / 1=enabled Bit2: SoftStop MIN, 0=disabled / 1=enabled Bit3: SoftStop MAX, 0=disabled / 1=enabled Bit4: Error on SoftStop, 0=disabled / 1=enabled Bit5: Stop Motor when blocked, 0=disabled / 1=enabled Bit6: Current Control by Software, 0=disabled / 1=enabled Bit7: Direction inversion, 0=disabled / 1=enabled

ControlTyp	0x2C	Type of PID Control (0..7) 0 = Open Loop 1 = Position control with no speed Profile 2 = Position control with Trapezoal Speed Profile 3 = Speed control with no speed profile 4 = Speed control with Trapezoal acceleration Profile 5 = Torque control 6 = Zero Friction control (not implemented) 7 = Not implemented
ErrorFlags	0x2D	Flags indicating an error (read only!) Bit0: Sample time to small Bit1: Watchdog timer overflow Bit2: Brown-out Bit3: SoftStop happened (only if SoftStop enabled) Bit4: Motor blocked (only if Motor stop while blocked enabled) Bit5: Position out of range (Overflow), 0=No / 1=Yes Bit6: Speed out of range (Overflow), 0=No / 1=Yes Bit7: Torque out of range (Overflow), 0=No / 1=Yes
StatusFlags	0x2E	Flags indicating the status of the controller (read only!) Bit0: Movement detected, 0=No / 1=Yes Bit1: Direction of movement, 0=negative / 1=positive Bit2: On SetPoint, 0=No / 1=Yes Bit3: Near SetPoint (+/-5units), 0=No / 1=Yes Bit4: Saturation of Driver Command, 0=No / 1=Yes Bit5: Antireset Windup active / Integrator Owerflow, 0=No / 1=Yes Bit6: Current Control active, 0=No / 1=Yes Bit7: SoftStop active, 0=No / 1=Yes
SetPointLL	0x2F	32bit target point
SetPointLH	0x30	Target point is only updated when the LL value is writen
SetPointHL	0x31	
SetPointHH	0x32	
PositionLL	0x34	
PositionLH	0x35	
PositionHL	0x36	Always first read the PositionHH register!!!
PositionHH	0x37	These 4 variables contain a copy of the 32bit position
SpeedL	0x38	Always first read the SpeedHH register!!!
SpeedH	0x39	These 2 variables contain a copy of the 16bit Speed
TorqueL	0x3A	Always first read the TorqueHH register!!!
TorqueH	0x3B	These 2 variables contain a copy of the 16bit Torque
TorqueBiasL	0x3C	These 2 variables contain the 16bit Bias of the Torque Measurement
TorqueBiasH	0x3D	

KpSpeedL	0x3E	Kp for speed PID
KpSpeedH	0x3F	
KdSpeedL	0x40	Kd for speed PID
KdSpeedH	0x41	
KiSpeedL	0x42	Ki for speed PID
KiSpeedH	0x43	
SampleTimeL	0x44	Sampling time [h=SamplingTime*1.6us] (20MHz)
SampleTimeH	0x45	
BlockedTime	0x46	Time to wait before the motor is considered blocked [T=BlockedTime*256*h]
IntGenPeriod	0x47	Period of internal function generator [T=IntGenPeriod*256*h] (0..255)
IntGenAmplitude	0x48	Amplitude of internal function generator (0..255)
IntGenOffset	0x49	Offset of internal function generator (-127..127)
SoftStopMin	0x4A	32bit Position of SoftStop minimum
SoftStopMax	0x4E	32bit Position of SoftStop maximum
Acceleration	0x52	Acceleration for trapezoidal speed profile (0..255)
StaticFriction	0x54	Friction of the system (0..255)
SWCurrentLimitL	0x56	Software current limit (16 bit)
SWCurrentLimitH	0x57	used for blocked detection and software current limit
MinSampleTimeL	0x58	Time used to pass one cycle
MinSampleTimeH	0x59	
MinSpeedL	0x5E	Minimal consigne Speed for regulation control with trapezoidal or constant speed
MinSpeedH	0x5F	
NearTargetMargin	0x60	Margin for near target flag setting and speed profile
KpPos	0x61	Kp for Position PID
KdPos	0x63	Kd for Position PID
KiPos	0x65	Ki for Position PID
KpTorque	0x67	Kp for Current PID
KdTorque	0x69	Kd for Current PID
KiTorque	0x6B	Ki for Current PID
MaxSpeed	0x6D	Maximum speed for trapezoidal speed profile