

User's guide

SMAX



RS-485 version



lika

Smart encoders & actuators

This publication was produced by Lika Electronic s.r.l. 2015. All rights reserved. Tutti i diritti riservati. Alle Rechte vorbehalten. Todos los derechos reservados. Tous droits réservés.

This document and information contained herein are the property of Lika Electronic s.r.l. and shall not be reproduced in whole or in part without prior written approval of Lika Electronic s.r.l. Translation, reproduction and total or partial modification (photostat copies, film and microfilm included and any other means) are forbidden without written authorisation of Lika Electronic s.r.l.

The information herein is subject to change without notice and should not be construed as a commitment by Lika Electronic s.r.l. Lika Electronic s.r.l. reserves the right to make all modifications at any moments and without forewarning.

This manual is periodically reviewed and revised. As required we suggest checking if a new or updated edition of this document is available at Lika Electronic s.r.l.'s website. Lika Electronic s.r.l. assumes no responsibility for any errors or omissions in this document. Critical evaluation of this manual by the user is welcomed. Your comments assist us in preparation of future documentation, in order to make it as clear and complete as possible. Please send an e-mail to the following address info@lika.it for submitting your comments, suggestions and criticisms.



General contents

| | |
|--|-----------|
| User's guide..... | 1 |
| General contents..... | 3 |
| Subject Index..... | 5 |
| Typographic and iconographic conventions..... | 6 |
| Preliminary information..... | 7 |
| 1 Safety summary..... | 8 |
| 1.1 Safety..... | 8 |
| 1.2 Electrical safety..... | 8 |
| 1.3 Mechanical safety..... | 9 |
| 2 Identification..... | 10 |
| 3 Mechanical installation..... | 11 |
| 3.1 Overall dimensions..... | 11 |
| 3.2 Magnetic scale..... | 11 |
| 3.3 Mounting the sensor..... | 12 |
| 4 Electrical connections..... | 14 |
| 4.1 Connection scheme..... | 14 |
| 4.1.1 M8 cable specifications..... | 14 |
| 4.1.2 M12 8-pin connector..... | 15 |
| 4.2 Ground connection..... | 15 |
| 4.3 Bus termination resistor..... | 16 |
| 4.4 Diagnostic LED (Figure 3)..... | 17 |
| 5 Quick reference..... | 19 |
| 5.1 Getting started..... | 19 |
| 6 Modbus® interface..... | 20 |
| 6.1 Modbus Master / Slaves protocol principle..... | 20 |
| 6.2 Modbus frame description..... | 21 |
| 6.3 Transmission modes..... | 22 |
| 6.3.1 RTU transmission mode..... | 23 |
| 6.4 Function codes..... | 25 |
| 6.4.1 Implemented function codes..... | 25 |
| 03 Read Holding Registers..... | 25 |
| 04 Read Input Register..... | 27 |
| 06 Write Single Register..... | 29 |
| 16 Write Multiple Registers..... | 31 |
| 7 Programming parameters..... | 35 |
| 7.1 Parameters available..... | 35 |
| 7.1.1 Machine data parameters..... | 35 |
| Resolution [00 hex]..... | 35 |
| Preset value [01 hex]..... | 37 |
| Offset value [02 hex]..... | 39 |
| Operating parameters [03 hex]..... | 39 |
| Scaling function..... | 40 |
| Code sequence..... | 40 |
| Node address [04 hex]..... | 41 |
| Serial com baud rate [05 hex]..... | 41 |
| Control Word [0A hex]..... | 42 |

| | |
|--|-----------|
| Watch dog enable..... | 42 |
| Save parameters..... | 42 |
| Load default parameters..... | 43 |
| Perform counting preset..... | 43 |
| 7.1.2 Input Register parameters..... | 44 |
| Alarms register [00 hex]..... | 44 |
| Machine data not valid..... | 44 |
| Flash memory error..... | 44 |
| Hall sensors error..... | 44 |
| Mounting error..... | 44 |
| Watch dog..... | 45 |
| Current position [01 hex]..... | 45 |
| Current velocity [02 hex]..... | 45 |
| Wrong parameters list [03 hex]..... | 46 |
| SW Version [04 hex]..... | 46 |
| HW Version [05 hex]..... | 47 |
| Status word [06 hex]..... | 47 |
| Scaling..... | 47 |
| Counting direction..... | 47 |
| Alarm..... | 48 |
| 7.2 Exception codes..... | 49 |
| 8 Firmware upgrade..... | 50 |
| 8.1 Information on firmware upgrade..... | 50 |
| 8.2 Preliminary operations and connections..... | 51 |
| 8.3 Launching the firmware upgrade process..... | 51 |
| 9 Programming examples..... | 53 |
| 9.1 Using the 03 Read Holding Registers function code..... | 53 |
| 9.2 Using the 04 Read Input Register function code..... | 54 |
| 9.3 Using the 06 Write Single Register function code..... | 55 |
| 9.4 Using the 16 Write Multiple Registers function code..... | 56 |
| 10 Default parameters list..... | 57 |
| 10.1 List of the Holding Registers with default value..... | 57 |
| 10.2 List of the Input Registers..... | 57 |

Subject Index




| | | |
|--------------------------------|----|---------------------------------------|
| A | | |
| Alarm..... | 48 | |
| Alarms register [00 hex]..... | 44 | |
| C | | |
| Code sequence..... | 40 | |
| Control Word [0A hex]..... | 42 | |
| Counting direction..... | 47 | |
| Current position [01 hex]..... | 45 | |
| Current velocity [02 hex]..... | 45 | |
| F | | |
| Flash memory error..... | 44 | |
| H | | |
| Hall sensors error..... | 44 | |
| HW Version [05 hex]..... | 47 | |
| L | | |
| Load default parameters..... | 43 | |
| M | | |
| Machine data not valid..... | 44 | |
| Mounting error..... | 44 | |
| N | | |
| | | Node address [04 hex].....41 |
| O | | |
| | | Offset value [02 hex].....39 |
| | | Operating parameters [03 hex].....39 |
| P | | |
| | | Perform counting preset.....43 |
| | | Preset value [01 hex].....37 |
| R | | |
| | | Resolution [00 hex].....35 |
| S | | |
| | | Save parameters.....42 |
| | | Scaling.....47 |
| | | Scaling function.....40 |
| | | Serial com baud rate [05 hex].....41 |
| | | Status word [06 hex].....47 |
| | | SW Version [04 hex].....46 |
| W | | |
| | | Watch dog.....45 |
| | | Watch dog enable.....42 |
| | | Wrong parameters list [03 hex].....46 |

Typographic and iconographic conventions

In this guide, to make it easier to understand and read the text the following typographic and iconographic conventions are used:

- parameters and objects both of Lika device and interface are coloured in **GREEN**;
- alarms are coloured in **RED**;
- states are coloured in **FUCSIA**.

When scrolling through the text some icons can be found on the side of the page: they are expressly designed to highlight the parts of the text which are of great interest and significance for the user. Sometimes they are used to warn against dangers or potential sources of danger arising from the use of the device. You are advised to follow strictly the instructions given in this guide in order to guarantee the safety of the user and ensure the performance of the device. In this guide the following symbols are used:

| | |
|---|--|
|  | This icon, followed by the word WARNING , is meant to highlight the parts of the text where information of great significance for the user can be found: user must pay the greatest attention to them! Instructions must be followed strictly in order to guarantee the safety of the user and a correct use of the device. Failure to heed a warning or comply with instructions could lead to personal injury and/or damage to the unit or other equipment. |
|  | This icon, followed by the word NOTE , is meant to highlight the parts of the text where important notes needful for a correct and reliable use of the device can be found. User must pay attention to them! Failure to comply with instructions could cause the equipment to be set wrongly: hence a faulty and improper working of the device could be the consequence. |
|  | This icon is meant to highlight the parts of the text where suggestions useful for making it easier to set the device and optimize performance and reliability can be found. Sometimes this symbol is followed by the word EXAMPLE when instructions for setting parameters are accompanied by examples to clarify the explanation. |

Preliminary information

This guide is designed to provide the most complete information the operator needs to correctly and safely install and operate the **SMAX absolute linear encoder fitted with Modbus interface**.

This sensor is designed to measure linear displacements in industrial machines and automation systems. The measurement system includes a magnetic scale and a magnetic sensor with conversion electronics. The scale is magnetized with a coded sequence of North-South poles generating a pseudo-random absolute pattern. As the sensor is moved along the magnetic scale, it detects the displacement and yields the absolute position information through the Modbus interface.

SMAX is also available with SSI interface (SMAX-BG, SMAX-GG) or voltage (SMAX-AV2) / current (SMAX-AI1) analogue interface. SSI / analogue interface encoders are provided with their own technical documentation.

It is mandatory to pair the sensor with the **MTAX type magnetic scale**.

To make it easier to read the text, this guide can be divided into two main sections.

In the first section general information concerning the safety, the mechanical installation and the electrical connection as well as tips for setting up and running properly and efficiently the unit are provided.

While in the second section, entitled **Modbus Interface**, both general and specific information is given on the Modbus interface. In this section the interface features and the registers implemented in the unit are fully described.



Warning: encoders having order code ending with "/Sxxx" may have mechanical and electrical characteristics different from standard and be supplied with additional documentation for special connections (Technical info).

1 Safety summary



1.1 Safety

- Always adhere to the professional safety and accident prevention regulations applicable to your country during device installation and operation;
- installation and maintenance operations have to be carried out by qualified personnel only, with power supply disconnected and stationary mechanical parts;
- device must be used only for the purpose appropriate to its design: use for purposes other than those for which it has been designed could result in serious personal and/or the environment damage;
- high current, voltage and moving mechanical parts can cause serious or fatal injury;
- warning ! Do not use in explosive or flammable areas;
- failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the equipment;
- Lika Electronic s.r.l. assumes no liability for the customer's failure to comply with these requirements.



1.2 Electrical safety

- Turn OFF power supply before connecting the device;
- connect according to explanation in the "Electrical connections" section;
- in compliance with 2004/108/EC norm on electromagnetic compatibility, following precautions must be taken:
 - before handling and installing the equipment, discharge electrical charge from your body and tools which may come in touch with the device;
 - power supply must be stabilized without noise; install EMC filters on device power supply if needed;
 - always use shielded cables (twisted pair cables whenever possible);
 - avoid cables runs longer than necessary;
 - avoid running the signal cable near high voltage power cables;
 - mount the device as far as possible from any capacitive or inductive noise source; shield the device from noise source if needed;
 - to guarantee a correct working of the device, avoid using strong magnets on or near by the unit;
 - minimize noise by connecting the shield and/or the connector housing and/or the sensor to ground. Make sure that ground is not affected by



noise. The connection point to ground can be situated both on the device side and on user's side. The best solution to minimize the interference must be carried out by the user;

- do not stretch the cable; do not pull or carry by cable; do not use the cable as a handle.



1.3 Mechanical safety

- Install the device following strictly the information in the "Mechanical installation" section;
- mechanical installation has to be carried out with stationary mechanical parts;
- do not disassemble the unit;
- do not tool the unit;
- delicate electronic equipment: handle with care; do not subject the device and the shaft to knocks or shocks;
- protect the unit against acid solutions or chemicals that may damage it;
- respect the environmental characteristics of the product;
- we suggest installing the unit providing protection means against waste, especially swarf as turnings, chips, or filings; should this not be possible, please make sure that adequate cleaning measures (as for instance brushes, scrapers, jets of compressed air, etc.) are in place in order to prevent the sensor and the magnetic scale from jamming.

2 Identification

Device can be identified through the **order code** and the **serial number** printed on the label applied to its body. Information is listed in the delivery document too. Please always quote the order code and the serial number when reaching Lika Electronic for purchasing spare parts or needing assistance. For any information on the technical characteristics of the product [refer to the technical catalogue](#).

3 Mechanical installation

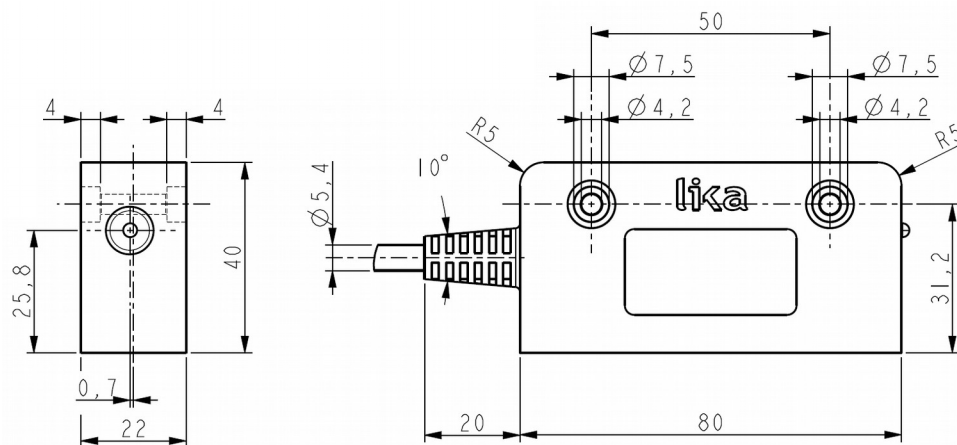


WARNING

Installation and maintenance operations have to be carried out by qualified personnel only, with power supply disconnected. Mechanical parts must be in stop.

For any information on the mechanical data and the electrical characteristics of the encoder please refer to the technical catalogue.

3.1 Overall dimensions



3.2 Magnetic scale

The sensor has to be paired with the **MTAX type magnetic scale** only.

Install the unit providing protection means against waste, especially swarf as turnings, chips or filings; should this not be possible, please make sure that adequate cleaning measures (as for instance brushes, scrapers, jets of compressed air, etc.) are in place in order to prevent the sensor and the magnetic scale from jamming.

Make sure the mechanical installation meets the system's requirements concerning distance, planarity and parallelism between the sensor and the scale indicated in Figure 2 all along the whole measuring length.

MTAX magnetic scale can be provided with a cover strip to protect its magnetic surface (see the order code).

Figure 1 shows how the sensor and the scale must be installed; the arrow indicates the **standard counting direction** (increasing count when the sensor moves in the direction indicated by the arrow; further information can be found in the section "Code sequence" on page 40).



WARNING

The system cannot operate if mounted otherwise than illustrated in Figure 1.

3.3 Mounting the sensor

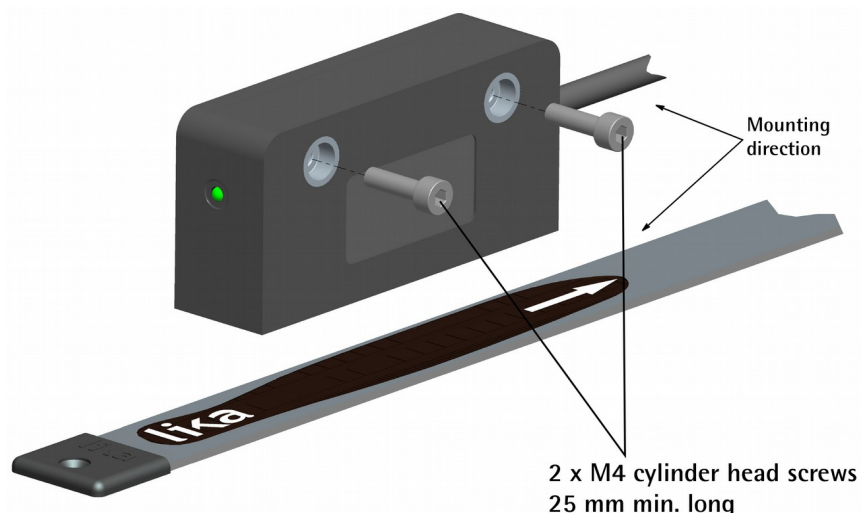


Figure 1

Make sure the mechanical installation complies with the system requirements concerning distance, planarity and parallelism between the sensor and the scale. Avoid contact between the parts. Sensor is fixed by means of **two M4 25 mm min. long cylinder head screws** inserted in the provided holes. Recommended **minimum bend radius** of the cable: **$R \geq 25 \text{ mm}$** . Install the sensor and the magnetic scale as shown in the Figure. The system does not operate if mounted otherwise than illustrated in the Figure. The arrow is intended to indicate the standard counting direction (count up information).

Please note that the MTAX magnetic scale can be provided with a cover strip to protect its magnetic surface (see the order code). Therefore the distance between the sensor and the magnetic scale is different whether the cover strip is applied.

The allowed gap D (see Figure 2) between the sensor and the MTAX magnetic scale has to be as follows:

| without cover strip | with cover strip |
|-----------------------------------|-----------------------------------|
| 0.1 mm ÷ 2.0 mm (0.004" ÷ 0.079") | 0.1 mm ÷ 1.6 mm (0.004" ÷ 0.063") |



WARNING

Make sure the mechanical installation complies with the system requirements concerning distance, planarity and parallelism between the sensor and the scale as shown in Figure 2 all along the whole measuring length.

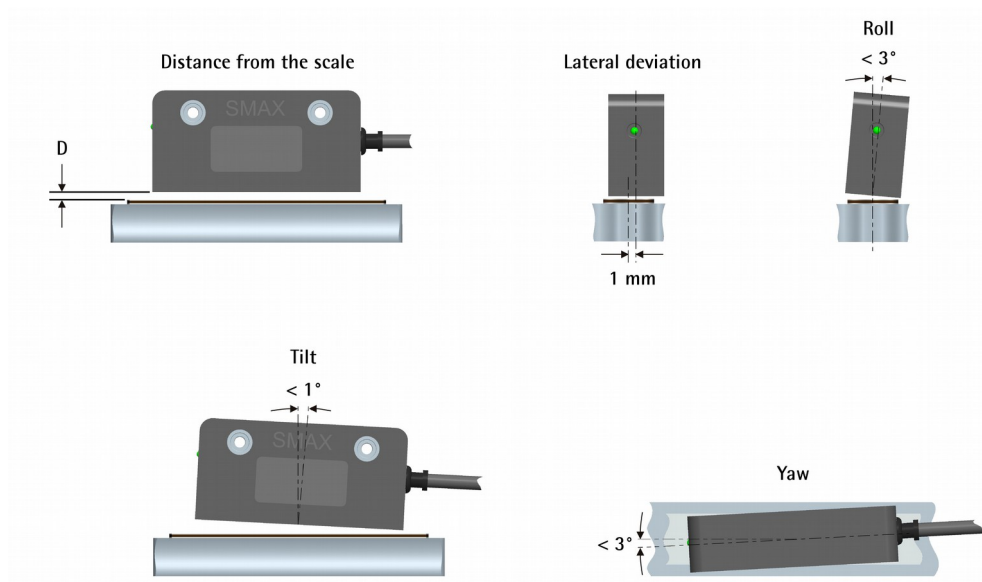


Figure 2



WARNING

After having installed the sensor on the magnetic scale a zero setting operation is compulsorily required. The zero setting operation is further required every time either the sensor or the scale is replaced. For any information on the zero setting operation please refer to the "Perform counting preset" section on page 43.

4 Electrical connections



WARNING

Power supply must be turned off before performing any electrical connection!

For any information on the mechanical data and the electrical characteristics of the encoder please refer to the technical catalogue.

4.1 Connection scheme



WARNING

If wires of unused signals come in contact, irreparable damage could be caused to the device. Please insulate them singularly.

| Function | M8 cable | M12 8-pin |
|--------------------------------|----------|-----------|
| 0Vdc power supply ¹ | Black | 1 |
| +10Vdc +30Vdc power supply | Red | 2 |
| A_RS485 IN | Yellow | 3 |
| B_RS485 IN | Blue | 4 |
| A_RS485 OUT ² | Green | 5 |
| B_RS485 OUT ² | Orange | 6 |
| n.c. | White | 7 |
| n.c. | Grey | 8 |
| Shield | Shield | Case |

¹ 0Vdc of the RS-485 serial line too.

² In order to minimize cable reflections and ensure a defined noise level on the data lines a 120Ω termination resistor must be provided between A_RS485 OUT and B_RS485 OUT if the encoder is the last slave in the line. For any information refer to the section "4.3 Bus termination resistor" on page 16.

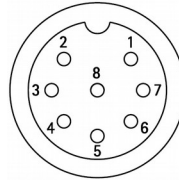
4.1.1 M8 cable specifications

Model : LIKA HI-FLEX M8 cable
 Wires : 6 x 0.14 mm² + 2 x 0.22 mm²
 Shield : tinned copper braid
 External diameter : 5.3 mm ÷ 5.6 mm
 Impedance : 6 x 148 Ω/km, 2 x 90 Ω/km
 Min. bending radius : Ø x 7.5

4.1.2 M12 8-pin connector

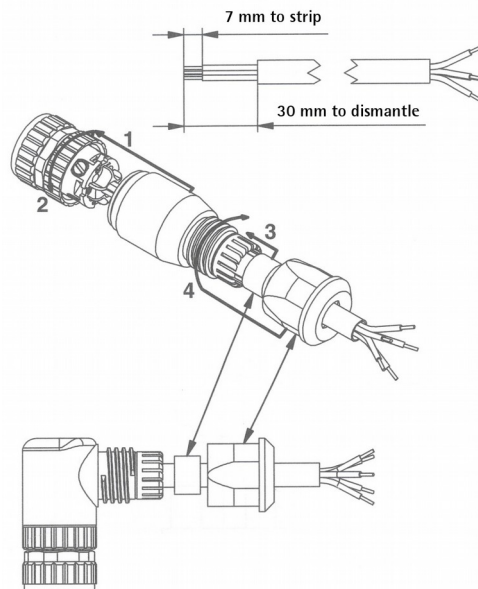
Male, frontal side

A coding



4.2 Ground connection

To minimize noise connect properly the shield and/or the connector housing and/or the frame to ground. Connect properly the cable shield to ground on user's side. Lika's EC- pre-assembled cables are fitted with shield connection to the connector ring nut in order to allow grounding through the body of the device. Lika's E- connectors have a plastic gland, thus grounding is not possible. If metal connectors are used, connect the cable shield properly as recommended by the manufacturer. Anyway make sure that ground is not affected by noise. It is recommended to provide the ground connection as close as possible to the device.



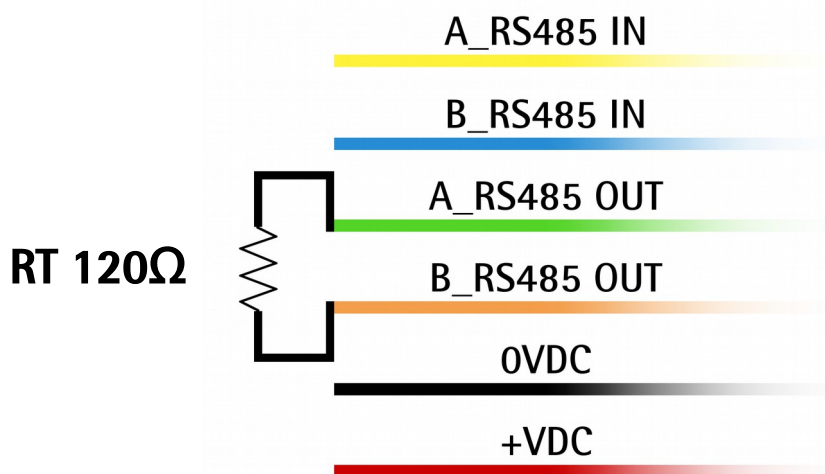
4.3 Bus termination resistor



WARNING

Power supply must be turned off before performing this operation!

As the RS-485 cables have an impedance of typically 120Ω it is necessary to always add a termination resistor (RT) in order to minimize cable reflections and ensure a defined noise level on the data lines. The termination resistor must be installed at both physical ends of the line: at the beginning of the RS-485 communication bus (typically in the PLC) and at the end of the bus, in the last slave of the line. We recommend a 120Ω termination resistor to be used. The termination resistor has to be provided between the A_RS485 OUT and the B_RS485 OUT signal wires in the last slave of the line, as shown in the Figure below.



4.4 Diagnostic LED (Figure 3)

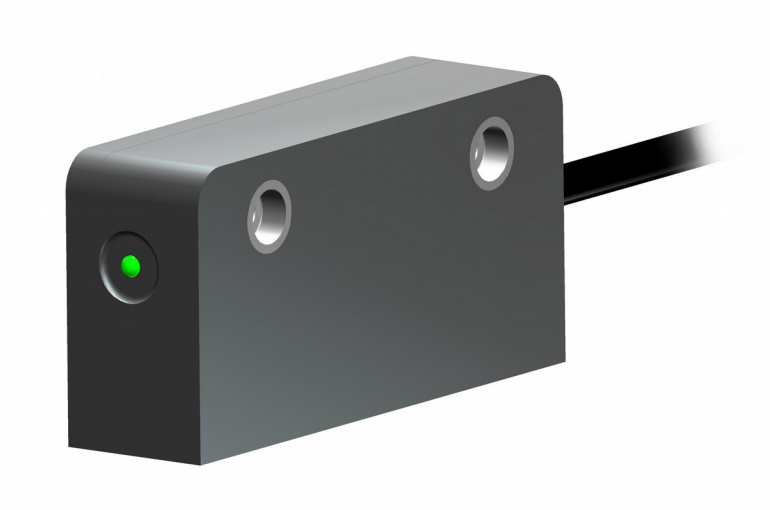


Figure 3: Diagnostic LED

A green LED located in the front of the encoder (see the Figure above) is meant to show visually the operating or fault status of the Modbus interface and the device as well. The LED operation is explained in the following table. In case of error, to know in detail which alarm has been triggered, see the [Alarms register \[00 hex\]](#) variable on page 44.

| GREEN LED | Description |
|---|--|
| ON (Solid GREEN) | The encoder is operating properly, there are no active errors. |
| Blinking at high frequency (100 ms ON / 100 ms OFF) | Machine data parameters error. To see in detail which parameter is wrong please enter the Wrong parameters list [03 hex] register. |
| Blinking slowly (500 ms ON / 500 ms OFF) | Flash memory error, it cannot be restored (bad checksum error, etc.). |
| Blinking very slowly (1 s ON / 1 s OFF) | An error has occurred in the Hall sensors while reading the magnetic scale, the read value does not exist. |
| Single flash (200 ms ON / 1 s OFF) | The encoder is installed too far from the magnetic scale, the installation does not comply with the mounting tolerances between the sensor and the scale (see Figure |

| | |
|--|--|
| | 2) Refer to the section "Mechanical installation" on page 11. |
| Double flash (200 ms ON/OFF twice / 1 s OFF) | Several errors are active at the same time. To know in detail which alarm has been triggered, see the Alarms register [00 hex] variable on page 44. |

While performing the firmware upgrade operation (bootloading), the green LED operates in a specific way, as explained in the following table.

| GREEN LED | Description |
|---|---|
| Blinking at 2 Hz with duty cycle = 50% | The operator has pressed the SET BOOT STATE button in the Firmware Upgrade software tool, the encoder is waiting for the firmware upgrade operation to start. For any information please refer to the "Firmware upgrade" section on page 50. |
| Blinking at 5 Hz with duty cycle = 50% | The operator has pressed the DOWNLOAD FILE button in the Firmware Upgrade software tool, the firmware upgrade operation is in progress. For any information please refer to the "Firmware upgrade" section on page 50. |
| Blinking at 10 Hz with duty cycle = 50% | An error occurred while performing the firmware upgrade operation. You must turn the power off and on again to reset the device and restart the operation. For any information please refer to the "Firmware upgrade" section on page 50. |
| ON (Solid GREEN) | The firmware upgrade operation has been carried out successfully, the encoder is operating properly and no error is active. For any information please refer to the "Firmware upgrade" section on page 50. |

5 Quick reference

5.1 Getting started

The following instructions are provided to allow the operator to set up the device for standard operation in a quick and safe mode.

- Install the device mechanically;
- perform the electrical connections;
- switch +10Vdc ÷ +30 Vdc power supply on;
- if needed, set the data transmission rate (baud rate and parity bit; see the register **Serial com baud rate [05 hex]** on page 41); the default value set by Lika Electronic at factory set-up is "4" = baud rate 19200 bit/s, parity bit Even;
- if needed, set the node address (node ID; see the register **Node address [04 hex]** on page 41); the default value set by Lika Electronic at factory set-up is "1";
- if you want to use the default resolution of the unit (10 = 0.1 mm resolution), please check the **Scaling function** item is disabled (bit 0 in the register **Operating parameters [03 hex]** = 0; see on page 40);
- otherwise if you need a specific resolution, please enable the **Scaling function** item (bit 0 in the register **Operating parameters [03 hex]** = 1; see on page 40) and then set the resolution you need for your application next to the **Resolution [00 hex]** item (register 1; see on page 35);
- now, if you need you can set the Preset next to the register **Preset value [01 hex]** and then execute the **Perform counting preset** command in **Control Word [0A hex]**; see on page 37;
- save new setting values (**Save parameters** register; see on page 42).

6 Modbus® interface

Lika SMAX Modbus series linear encoders are Slave devices and implement the Modbus application protocol (level 7 of OSI model) and the "Modbus over Serial Line" protocol (levels 1 & 2 of OSI model).

For any further information or omitted specifications please refer to "Modbus Application Protocol Specification V1.1b" and "Modbus over Serial Line. Specification and Implementation Guide V1.02" available at www.modbus.org.

6.1 Modbus Master / Slaves protocol principle

The Modbus Serial Line protocol is a Master – Slaves protocol. One only Master (at the same time) is connected to the bus and one or several (247 maximum number) Slave nodes are also connected to the same serial bus. A Modbus communication is always initiated by the Master. The Slave nodes will never transmit data without receiving a request from the Master node. The Slave nodes will never communicate with each other. The Master node initiates only one Modbus transaction at the same time.

The Master node issues a Modbus request to the Slave nodes in two modes:

- **UNICAST mode:** in that mode the Master addresses an individual Slave. After receiving and processing the request, the Slave returns a message (a "reply") to the Master. In that mode, a Modbus transaction consists of two messages: a request from the Master and a reply from the Slave. Each Slave must have a unique address (from 1 to 247) so that it can be addressed independently from other nodes. Lika devices only implement commands in "unicast" mode.
- **BROADCAST mode:** in that mode the Master can send a request to all Slaves at the same time. No response is returned to "broadcast" requests sent by the Master. The "broadcast" requests are necessarily writing commands. The address 0 is reserved to identify a "broadcast" exchange.

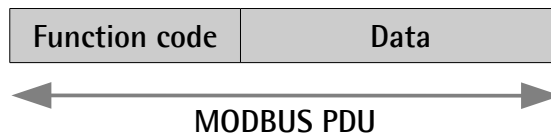


NOTE

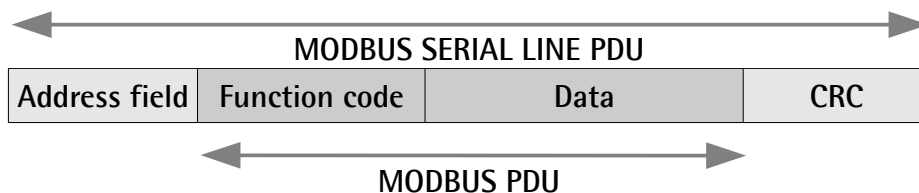
Lika devices do not implement commands in "broadcast" mode.

6.2 Modbus frame description

The Modbus application protocol defines a simple Protocol Data Unit (PDU) independent of the underlying communication layers:



The mapping of Modbus protocol on a specific bus or network introduces some additional fields on the Protocol Data Unit. The client that initiates a Modbus transaction builds the Modbus PDU, and then adds fields in order to build the appropriate communication PDU.



- **ADDRESS FIELD:** on Modbus Serial Line the address field only contains the Slave address. The valid Slave node addresses are in the range of 0 – 247 decimal (see register [Node address \[04 hex\]](#) on page 41). The individual Slave devices are assigned addresses in the range of 1 – 247. A Master addresses a Slave by placing the Slave address in the **ADDRESS FIELD** of the message. When the Slave returns its response, it places its own address in the response **ADDRESS FIELD** to let the Master know which Slave is responding.
- **FUNCTION CODE:** the function code indicates to the Server what kind of action to perform. The function code can be followed by a **DATA** field that contains request and response parameters. For any further information on the implemented function codes refer to the "6.4 Function codes" section on page 25.
- **DATA:** the **DATA** field of messages contains the bytes for additional information and transmission specifications that the server uses to take the action defined by the **FUNCTION CODE**. This can include items such as discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field. The structure of the **DATA** field depends on each **FUNCTION CODE** (refer to the "6.4 Function codes" section on page 25).
- **CRC (Cyclical Redundancy Checking):** error checking field is the result of a "Redundancy Checking" calculation that is performed on the message contents. This is intended to check whether transmission has been performed properly. The CRC field is two bytes, containing 16-bit

binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The device that receives recalculates a CRC during receipt of the message and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results.

The Modbus protocol defines three PDUs. They are:

- **Modbus Request PDU;**
- **Modbus Response PDU;**
- **Modbus Exception Response PDU.**

The **Modbus Request PDU** is defined as {function_code, request_data}, where:
function_code = Modbus function code [1 byte];
request_data = this field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function, etc. [n bytes].

The **Modbus Response PDU** is defined as {function_code, response_data}, where:
function_code = Modbus function code [1 byte];
response_data = this field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function, etc. [n bytes].

The **Modbus Exception Response PDU** is defined as {exception-function_code, exception_code}, where:
exception-function_code = Modbus function code + 80 hex [1 byte];
exception_code = Modbus Exception code, refer to the table "Modbus Exception Codes" in the section 7 of the document "Modbus Application Protocol Specification V1.1b".

6.3 Transmission modes

Two different serial transmission modes are defined in the Modbus serial protocol: the **RTU (Remote Terminal Unit) mode** and the **ASCII mode**. The transmission mode defines the bit contents of message fields transmitted serially on the line. It determines how information is packed into the message fields and decoded. The transmission mode and the serial port parameters must be the same for all devices on a Modbus Serial Line. All devices must implement the RTU mode, while the ASCII mode is an option. Lika devices only implement RTU transmission mode, as described in the following section.

6.3.1 RTU transmission mode

When devices communicate on a Modbus serial line using the RTU (Remote Terminal Unit) mode, each 8-bit byte in a message contains two 4-bit hexadecimal characters. Each message must be transmitted in a continuous stream of characters. Synchronization between the messages exchanged by the transmitting device and the receiving device is achieved by placing an interval of at least 3.5 character times between successive messages, this is called "silent interval". In this way a Modbus message is placed by the transmitting device into a frame that has a known beginning and ending point. This allows devices that receive a new frame to begin at the start of the message and to know when the message is completed. So when the receiving device does not receive a message for an interval of 4 character times, it considers the previous message as completed and the next byte will be the first of a new message, i.e. an address.

When baud rate = 9600 bit/s the "silent interval" is 4 ms.

When baud rate = 19200 bit/s the "silent interval" is 2 ms.

When baud rate = 115200 bit/s the "silent interval" is 3.5 ms.

The format (11 bits) for each byte in RTU mode is as follows:

Coding system: 8-bit binary

Bits per Byte: 1 start bit;

8 data bits, least significant bit (lsb) sent first;

1 bit for parity completion (= Even);

1 stop bit.

Modbus protocol uses a "big-Endian" representation for addresses and data items. This means that when a numerical quantity greater than a single byte is transmitted, the most significant byte (MSB) is sent first.

Each character or byte is sent in this order (left to right):

lsb (Least Significant Bit) ... msb (Most Significant Bit)

| | | | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|---------|------|
| Start | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Parity* | Stop |
|-------|---|---|---|---|---|---|---|---|---------|------|

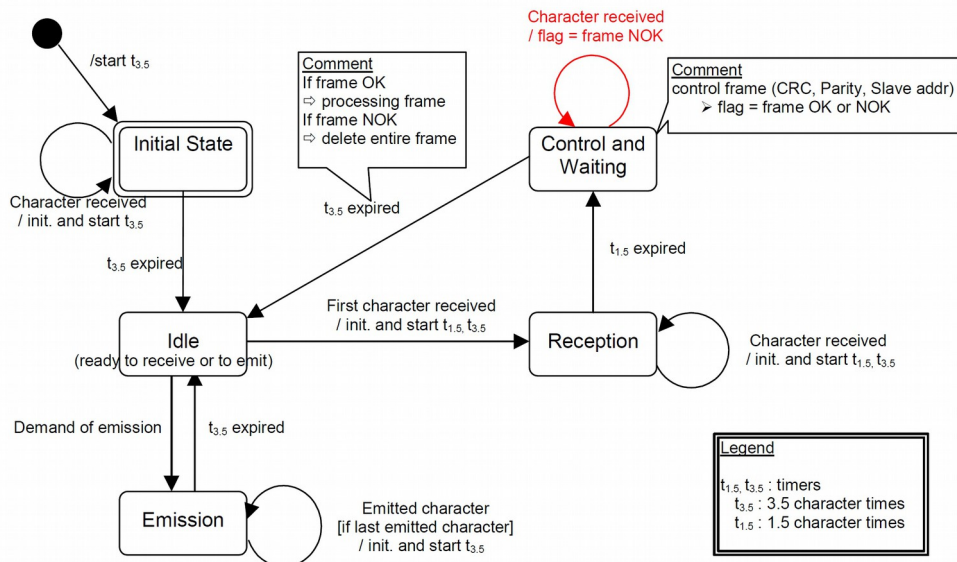
* When "No parity" is activated, the parity bit is replaced by a stop bit.

The default parity mode must be Even parity.

The maximum size of the Modbus RTU frame is 256 bytes, its structure is as follows:

| Slave Address | Function code | Data | CRC |
|---------------|---------------|---------------------|-----------------------------|
| 1 byte | 1 byte | 0 up to 252 byte(s) | 2 bytes CRC Low CRC Hi |

The following drawing provides a description of the RTU transmission mode state diagram. Both "Master" and "Slave" points of view are expressed in the same drawing.



- Transition from **Initial State** to **Idle** state needs an interval of at least 3.5 character times (time-out expiration = $t_{3.5}$).
- **Idle** state is the normal state when neither emission nor reception is active. In RTU mode, the communication link is declared in **Idle** state when there is no transmission activity after a time interval equal to at least 3.5 characters ($t_{3.5}$).
- A request can only be sent in **Idle** state. After sending a request, the Master leaves the **Idle** state and cannot send a second request at the same time.
- When the link is in **Idle** state, each transmitted character detected on the link is identified as the start of the frame. The link goes to **Active** state. Then the end of the frame is identified when no more character is transmitted on the link after the time interval of at least $t_{3.5}$.
- After detection of the end of frame, the CRC calculation and checking is completed. Afterwards the address field is analysed to determine if the frame is addressed to the device. If not, the frame is discarded. In order to reduce the reception processing time the address field can be analysed as soon as it is received without waiting the end of frame. In this case the CRC will be calculated and checked only if the frame is actually addressed to the Slave.

6.4 Function codes

As previously stated, the function code indicates to the Server what kind of action to perform. The function code field of a Modbus data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (the range 128 ... 255 is reserved and used for Exception Responses). When a message is sent from a Client to a Server device the function code field tells the Server what kind of action to perform. Function code "0" is not valid.

There are three categories of Modbus function codes, they are: **public function codes**, **user-defined function codes** and **reserved function codes**.

Public function codes are in the range 1 ... 64, 73 ... 99 and 111 ... 127; they are well defined function codes, validated by the MODBUS-IDA.org community and publicly documented; furthermore they are guaranteed to be unique. Ranges of function codes from 65 to 72 and from 100 to 110 are **user-defined function codes**: user can select and implement a function code that is not supported by the specification, it is clear that there is no guarantee that the use of the selected function code will be unique. **Reserved function codes** are not available for public use.

6.4.1 Implemented function codes

Lika SMAX Modbus series linear encoders only implement public function codes, they are described hereafter.

03 Read Holding Registers

FC = 03 (03 hex) r/o

This function code is used to READ the contents of a contiguous block of holding registers in a remote device; in other words, it allows to read the values set in a group of work parameters placed in order. The Request PDU specifies the starting register address and the number of registers. In the PDU registers are addressed starting at zero. Therefore registers numbered 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits (msb) and the second contains the low order bits (lsb).

For the complete list of holding registers accessible using **03 Read Holding Registers** function code please refer to section "7.1.1 Machine data parameters" on page 35.

Request PDU

| | | |
|-----------------------|---------|----------------------|
| Function code | 1 byte | 03 hex |
| Starting address | 2 bytes | 0000 hex to FFFF hex |
| Quantity of registers | 2 bytes | 1 to 125 (007D hex) |

Response PDU

| | | |
|----------------|---------------------|---------------|
| Function code | 1 byte | 03 hex |
| Byte count | 1 byte | 2 x N* |
| Register value | N* x 2 bytes | |

*N = Quantity of registers

Exception Response PDU

| | | |
|----------------|--------|----------------------------------|
| Error code | 1 byte | 83 hex (=03 hex + 80 hex) |
| Exception code | 1 byte | 01 or 02 or 03 or 04 |



Here is an example of a request to read the parameter **Preset value [01 hex]** (register 2).

| Request | | Response | |
|---------------------|-----------|---------------------|-----------|
| Field name | (Hex) | Field name | (Hex) |
| Function | 03 | Function | 03 |
| Starting address Hi | 00 | Byte count | 02 |
| Starting address Lo | 01 | Register 2 value Hi | 05 |
| No. of registers Hi | 00 | Register 2 value Lo | DC |
| No. of registers Lo | 01 | | |

As you can see in the table, **Preset value [01 hex]** parameter (register 2) contains the value 05 DC hex, i.e. 1500 in decimal notation.

The full frame needed for the request to read the parameter **Preset value [01 hex]** (register 2) to the Slave having the node address 1 is as follows:

Request PDU (in hexadecimal format)

[01][03][00][01][00][01][D5][CA]

where:

[01] = Slave address

[03] = **03 Read Holding Registers** function code

[00][01] = starting address (**Preset value [01 hex]** parameter, register 2)

[00][01] = number of requested registers

[D5][CA] = CRC

The full frame needed to send back the value of the parameter **Preset value [01 hex]** (register 2) from the Slave having the node address 1 is as follows:

Response PDU (in hexadecimal format)

[01][03][02][05][DC][BA][8D]

where:

[01] = Slave address

[03] = **03 Read Holding Registers** function code

[02] = number of bytes (2 bytes for each register)

[05][DC] = value of register 2, 05 DC hex = 1500 dec

[BA][8D] = CRC

04 Read Input Register

FC = 04 (04 hex)

This function code is used to READ from 1 to 125 contiguous input registers in a remote device; in other words, it allows to read some results values and state / alarm messages in a remote device. The Request PDU specifies the starting register address and the number of registers. In the PDU registers are addressed starting at zero. Therefore input registers numbered 1-16 are addressed as 0-15. The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits (msb) and the second contains the low order bits (lsb).

For the complete list of input registers accessible using **04 Read Input Register** function code please refer to the "7.1.2 Input Register parameters" section on page 44.

Request PDU

| | | |
|-----------------------------|---------|----------------------|
| Function code | 1 byte | 04 hex |
| Starting address | 2 bytes | 0000 hex to FFFF hex |
| Quantity of Input Registers | 2 bytes | 0000 hex to 007D hex |

Response PDU

| | | |
|----------------------|---------------------|---------------|
| Function code | 1 byte | 04 hex |
| Byte count | 1 byte | 2 x N* |
| Input register value | N* x 2 bytes | |

*N = Quantity of registers

Exception Response PDU

| | | |
|----------------|--------|----------------------------------|
| Error code | 1 byte | 84 hex (=04 hex + 80 hex) |
| Exception code | 1 byte | 01 or 02 or 03 or 04 |



Here is an example of a request to read the **Current position [01 hex]** parameter (input register 2).

| Request | | Response | |
|---------------------------|-----------|---------------------|-----------|
| Field name | (Hex) | Field name | (Hex) |
| Function | 04 | Function | 04 |
| Starting address Hi | 00 | Byte count | 02 |
| Starting address Lo | 01 | Register 2 value Hi | 13 |
| Quantity of Input Reg. Hi | 00 | Register 2 value Lo | C5 |
| Quantity of Input Reg. Lo | 01 | | |

As you can see in the table, **Current position [01 hex]** parameter (input register 2) contains the value 13 C5 hex, i.e. 5061 in decimal notation.

The full frame needed for the request to read the **Current position [01 hex]** parameter (input register 2) to the Slave having the node address 1 is as follows:

Request PDU (in hexadecimal format)

[01][04][00][01][00][01][60][0A]

where:

[01] = Slave address

[04] = **04 Read Input Register** function code

[00][01] = starting address (**Current position [01 hex]** parameter, register 2)

[00][01] = number of requested registers

[60][0A] = CRC

The full frame needed to send back the value of the **Current position [01 hex]** parameter (register 2) from the Slave having the node address 1 is as follows:

Response PDU (in hexadecimal format)

[01][04][02][13][C5][74][53]

where:

[01] = Slave address

[04] = **04 Read Input Register** function code

[02] = number of bytes (2 bytes for each register)

[13][C5] = value of register 2 **Current position [01 hex]**, 13 C5 hex = 5061 dec

[74][53] = CRC

06 Write Single Register

FC = 06 (06 hex)

This function code is used to WRITE a single holding register in a remote device. The Request PDU specifies the address of the register to be written. Registers are addressed starting at zero. Therefore register numbered 1 is addressed as 0.

The normal response is an echo of the request, returned after the register contents have been written.

For the complete list of registers accessible using **06 Write Single Register** function code please refer to the "7.1.1 Machine data parameters" section on page 35.

Request PDU

| | | |
|------------------|---------|----------------------|
| Function code | 1 byte | 06 hex |
| Register address | 2 bytes | 0000 hex to FFFF hex |
| Register value | 2 bytes | 0000 hex to FFFF hex |

Response PDU

| | | |
|------------------|---------|----------------------|
| Function code | 1 byte | 06 hex |
| Register address | 2 bytes | 0000 hex to FFFF hex |
| Register value | 2 bytes | 0000 hex to FFFF hex |

Exception Response PDU

| | | |
|----------------|--------|----------------------------------|
| Error code | 1 byte | 86 hex (=06 hex + 80 hex) |
| Exception code | 1 byte | 01 or 02 or 03 or 04 |



Here is an example of a request to write in the **Operating parameters [03 hex]** item (register 4): we need to set the scaling function (**Scaling function = 1**) and the increasing counting when the sensor moves in the direction shown by the arrow in Figure 1 (**Code sequence = 0**).

| Request | | Response | |
|---------------------|-----------|---------------------|-----------|
| Field name | (Hex) | Field name | (Hex) |
| Function | 06 | Function | 06 |
| Register address Hi | 00 | Register address Hi | 00 |
| Register address Lo | 03 | Register address Lo | 03 |
| Register value Hi | 00 | Register value Hi | 00 |
| Register value Lo | 01 | Register value Lo | 01 |

As you can see in the table, the value 00 01 hex, i.e. 0000 0000 0000 0001 in binary notation, is set in the **Operating parameters [03 hex]** item (register 4):

bit 0 **Scaling function** = 1; bit 1 **Code sequence** = 0; the remaining bits are not used, therefore their value is 0.

The full frame needed for the request to write the value 00 01 hex in the **Operating parameters [03 hex]** item (register 4) to the Slave having the node address 1 is as follows:

Request PDU (in hexadecimal format)

[01][06][00][03][00][01][B8][0A]

where:

[01] = Slave address

[06] = **06 Write Single Register** function code

[00][03] = address of the register (**Operating parameters [03 hex]** item, register 4)

[00][01] = value to be set in the register

[B8][0A] = CRC

The full frame needed to send back a response following the request to write in the **Operating parameters [03 hex]** item (register 4) from the Slave having the node address 1 is as follows:

Response PDU (in hexadecimal format)

[01][06][00][03][00][01][B8][0A]

where:

[01] = Slave address

[06] = **06 Write Single Register** function code

[00][03] = address of the register (**Operating parameters [03 hex]** item, register 4)

[00][01] = value set in the register

[B8][0A] = CRC

16 Write Multiple Registers

FC = 16 (10 hex)

This function code is used to WRITE a block of contiguous registers (1 to 123 registers) in a remote device.

The values to be written are specified in the request data field. Data is packed as two bytes per register.

The normal response returns the function code, starting address and quantity of written registers.

For the complete list of registers accessible using **16 Write Multiple Registers** function code please refer to section "7.1.1 Machine data parameters" on page 35.

Request PDU

| | | |
|-----------------------|---------------------|----------------------|
| Function code | 1 byte | 10 hex |
| Starting address | 2 bytes | 0000 hex to FFFF hex |
| Quantity of registers | 2 bytes | 0001 hex to 007B hex |
| Byte count | 1 byte | 2 x N* |
| Registers value | N* x 2 bytes | value |

*N = Quantity of registers

Response PDU

| | | |
|-----------------------|---------|----------------------|
| Function code | 1 byte | 10 hex |
| Starting address | 2 bytes | 0000 hex to FFFF hex |
| Quantity of registers | 2 bytes | 1 to 123 (007B hex) |

Exception Response PDU

| | | |
|----------------|--------|-----------------------------------|
| Error code | 1 byte | 90 hex (= 10 hex + 80 hex) |
| Exception code | 1 byte | 01 or 02 or 03 or 04 |



Here is an example of a request to write the value 00 0A hex (=10) next to the parameter **Node address [04 hex]** (register 5) and the value 00 01 hex (= 1 = baud rate 9600 bit/s, parity bit Even) next to the parameter **Serial com baud rate [05 hex]** (register 6).

| Request | | Response | |
|------------|-----------|------------|-----------|
| Field name | (Hex) | Field name | (Hex) |
| Function | 10 | Function | 10 |

| | | | |
|--------------------------|----|--------------------------|----|
| Starting address Hi | 00 | Starting address Hi | 00 |
| Starting address Lo | 04 | Starting address Lo | 04 |
| Quantity of registers Hi | 00 | Quantity of registers Hi | 00 |
| Quantity of registers Lo | 02 | Quantity of registers Lo | 02 |
| Byte count | 04 | | |
| Register 1 value Hi | 00 | | |
| Register 1 value Lo | 0A | | |
| Register 2 value Hi | 00 | | |
| Register 2 value Lo | 01 | | |

As you can see in the table, the value 00 0A hex, i.e. 10 in decimal notation, is set in the register 5 **Node address [04 hex]** parameter; while the value 00 01 hex, i.e. 1 in decimal notation = baud rate 9600 bit/s, parity bit Even, is set in the register 6 **Serial com baud rate [05 hex]** parameter. Thus the encoder will be programmed to have node address 10 and data transmission rate = 1 = baud rate 9600 bit/s, parity bit Even.

The full frame needed for the request to write the value 10 dec next to the parameter **Node address [04 hex]** (register 5) and the value 1 dec next to the parameter **Serial com baud rate [05 hex]** (register 6) to the Slave having currently the node address 1 is as follows:

Request PDU (in hexadecimal format)

[01][10][00][04][00][02][04][00][0A][00][01][32][29]

where:

[01] = Slave address

[10] = **16 Write Multiple Registers** function code

[00][04] = starting address (**Node address [04 hex]** parameter, register 5)

[00][02] = number of requested registers

[04] = number of bytes (2 bytes for each register)

[00][0A] = value to be set in the register 5, 00 0A hex = 10 dec

[00][01] = value to be set in the register 6, 00 01 hex = 1 dec

[32][29] = CRC

The full frame needed to send back a response following the request to write the value 10 next to the parameter **Node address [04 hex]** (register 5) and the

value 1 next to the parameter **Serial com baud rate [05 hex]** (register 6) from the Slave having the node address 1 previously and the node address 10 currently is as follows:

Response PDU (in hexadecimal notation)

[0A][10][00][04][00][02][6D][BD]

where:

[0A] = new Slave address

[10] = **16 Write Multiple Registers** function code

[00][04] = starting address (**Node address [04 hex]** parameter, register 5)

[00][02] = number of written registers

[6D][BD] = CRC



WARNING

For safety reasons, when the encoder is on, a continuous data exchange between the Master and the Slave has to be planned in order to be sure that the communication is always active; this is intended to prevent danger situations from arising in case of failures in the communication network.

For this purpose the Watch dog function is implemented and can be activated as optional. Watch dog function is a safety timer that uses a time-out to detect loop or deadlock conditions. For instance, should the serial communication be cut off while a command is still active and running Watch dog safety system immediately takes action and commands an alarm to be triggered. To enable the Watch dog function, set to "1" the **Watch dog enable** bit 0 in the **Control Word [0A hex]** variable. If "0" is set the Watch dog is disabled; if "1" is set the Watch dog is enabled. When the Watch dog function is enabled, if the device does not receive a message from the Server within 1 second, the system forces an alarm condition (the **Watch dog** alarm message is invoked to appear as soon as the Modbus network communication is restored).

7 Programming parameters

7.1 Parameters available

Hereafter the parameters available for the Modbus encoders are listed and described as follows:

Parameter name [Register address]

[Register number, data types, attribute]

- The register address is expressed in hexadecimal notation.
- The register number is expressed in decimal notation.
- Attribute:
 - ro = read only access
 - rw = read and write access

7.1.1 Machine data parameters

Machine data parameters are accessible for both writing and reading; to read the value set in a parameter use the **03 Read Holding Registers** function code (reading of multiple registers); to write a value in a parameter use the **06 Write Single Register** function code (writing of a single register) or the **16 Write Multiple Registers** (writing of multiple registers); for any further information on the implemented function codes refer to the "6.4.1 Implemented function codes" section on page 25.

Resolution [00 hex]

[Register 1, Unsigned16, rw]

This parameter is used to set a custom resolution (measuring step).

This register can be programmed only if the bit 0 **Scaling function** in the **Operating parameters [03 hex]** item is set to "=1"; otherwise the system will use the standard resolution "=10" = 0.1 mm (the encoder will provide 6000 information for the whole travel, 13 bits).

The resolution can be defined as the smallest change in the underlying quantity that produces a response in the measurement, the response being the information that is provided to output.

By default the resolution is 0.1 mm, thus the encoder provides 6000 information (13 bits) for the whole travel, the measuring length of the MTAX scale being 600 mm.

You are allowed to set whatever value between 0.1 mm (10 hundredths of a millimetre) and 1.25 mm (125 hundredths of a millimetre). The entered value has to be expressed in hundredths of a millimetre.

If you set a value greater than the maximum resolution allowed, after sending the Request PDU the **Machine data not valid** error message will be sent back while the relevant bit in the **Wrong parameters list [03 hex]** item will be set to 1.

Default = 10 (min. = 10, max. = 125)



NOTE

If you have set the preset, when you change the value next to the **Resolution [00 hex]** parameter, then you must check the value in the **Preset value [01 hex]** parameter and perform the homing operation (bit 11 **Perform counting preset** in **Control Word [0A hex]** = 1).



EXAMPLE

The main and default features of the SMAX linear encoder are as follows:

- **Default resolution** = 0.1 mm
- **MTAX max. measuring length** = 600 mm
- **Max. number of information** = 6000 (13 bits)

As stated, the number of information provided to output is calculated as follows:

$$\text{number of information} = \frac{\text{measuring length}}{\text{resolution}}$$

Thus, in a default configuration the number of information is:

$$\text{number of information} = \frac{\text{measuring length}}{\text{resolution}} = \frac{600}{0.1} = 6000$$

Let's assume that you need 2000 information to be provided for the max. measuring length. It follows that you need to calculate and then set a custom resolution.

The resolution value results from the following calculation:

$$\text{resolution} = \frac{\text{measuring length}}{\text{number of information}}$$

Thus, in the example the resolution will be:

$$\text{resolution} = \frac{\text{measuring length}}{\text{number of information}} = \frac{600}{2000} = 0.3$$

As the value next to the **Resolution [00 hex]** parameter has to be expressed in hundredths of a millimetre, then you have to enter the value **30**.

The complete programming sequence will be:

- Enable the **Scaling function: Operating parameters [03 hex]**, bit 0 = 1
- Set the resolution: **Resolution [00 hex]** = 30 (0000 001E hex)
- Save the set parameters (**Save parameters** register; see on page 42)



NOTE

Please note that when the count is decreasing (count down information, see **Code sequence** in the **Operating parameters [03 hex]** register) and you cross the zero, the value immediately after 0 will be 2^{N-1} , where N is the overall information expressed in bits. In the above example the overall information is 2000, i.e. 11 bits (11 bits are necessary to represent 2000 information). $2^{11} = 2048$, thus the value after 0 will be 2047.

| | | | | | | | | | | |
|-----|------|------|------|------|---|---|---|---|---|-----|
| ← | | | | | | | | | | |
| ... | 2044 | 2045 | 2046 | 2047 | 0 | 1 | 2 | 3 | 4 | ... |

Preset value [01 hex]

[Register 2, Unsigned16, rw]

This register is intended to set the Preset value. Preset function is meant to assign a desired value to a physical position of the encoder. The chosen physical position will get the value set next to this item and all the previous and following positions will get a value according to it. For instance, this can be

useful for getting the zero point of the encoder and the zero point of the application to match. The preset value will be set for the position of the encoder in the moment when the **Perform counting preset** command (bit 11) in the **Control Word [0A hex]** register is sent.

Default = 0 (min. = 0, max. = 8191)



Example

Let's take a look at the following example to better understand the preset function and the meaning and use of the related registers and commands: **Preset value [01 hex]**, **Offset value [02 hex]** and **Perform counting preset**.

The encoder position which is transmitted results from the following calculation:

Transmitted value = **read position** (it does not matter whether the position is physical or scaled) + **Preset value [01 hex]** - **Offset value [02 hex]**.

If you never set the **Preset value [01 hex]** and the homing command has been executed never before (**Perform counting preset** command), the transmitted value and the read position are necessarily the same as **Preset value [01 hex]** = 0 and **Offset value [02 hex]** = 0.

When you set the **Preset value [01 hex]** and then execute the **Perform counting preset** command in the **Control Word [0A hex]**, system saves the current encoder position in the register **Offset value [02 hex]**. It follows that the transmitted value and the **Preset value [01 hex]** are the same as **read position** - **Offset value [02 hex]** = 0; in other words, the value set next to the **Preset value [01 hex]** item is paired with the current position of the encoder as you wish.

For example, let's assume that the value "50" is set next to the **Preset value [01 hex]** item and you execute the **Perform counting preset** command when the encoder position is "1000". In other words, you want to receive the value "50" when the encoder reaches the position "1000".

We will obtain the following information sequence:

Transmitted value = **read position** (= "1000") + **Preset value [01 hex]** (= "50") - **Offset value [02 hex]** (= "1000") = 50.

The following transmitted value will be:

Transmitted value = **read position** (= "1001") + **Preset value [01 hex]** (= "50") - **Offset value [02 hex]** (= "1000") = 51.

And so on.



NOTE

- If the **Scaling function** is disabled (bit 0 in the register **Operating parameters [03 hex]** = 0), **Preset value [01 hex]** must be lower than or

equal to the maximum number of information for the default resolution - 1 ($2^{13} - 1 = 8191$).

- If the **Scaling function** is enabled (bit 0 in the register **Operating parameters [03 hex]** = 1), **Preset value [01 hex]** must be lower than or equal to the maximum number of information for the custom resolution - 1 (for instance -see the example on page 36: 2000 information, number of bits = 11, the max. value you are allowed to enter for the Preset is: $2^{11} - 1 = 2047$).



WARNING

After having entered a new value in the register **Resolution [00 hex]** it is compulsory to check the **Preset value [01 hex]** and then perform a homing operation (bit 11 **Perform counting preset** in **Control Word [0A hex]** = 1).

Offset value [02 hex]

[Register 3, Unsigned16, ro]

As soon as you send the **Perform counting preset** command (see bit 11 in **Control Word [0A hex]**), the current position of the encoder is saved in this register. The offset value is then used in the preset function in order to calculate the encoder position value to be transmitted. To zero set the value in this register you must upload the factory default values (see bit 10, **Load default parameters** command, in **Control Word [0A hex]** on page 43).

For any further information on the preset function and the meaning and use of the related registers and commands **Preset value [01 hex]**, **Offset value [02 hex]** and **Perform counting preset** refer to page 37.

Default = 0 (min. = 0, max. = 8191)

Operating parameters [03 hex]

[Register 4, Unsigned16, rw]

Byte structure of the **Operating parameters [03 hex]** register:

| byte | MSB | | | LSB | | |
|------|-----|-----|-----|-----|-----|-----|
| bit | 15 | ... | 8 | 7 | ... | 0 |
| | msb | | lsb | msb | | lsb |

Byte 0

Scaling function

bit 0

This is meant to enable / disable the scaling parameter **Resolution [00 hex]**. When the scaling function is disabled (bit 0 = 0), the encoder uses its own default resolution; otherwise, when the scaling function is enabled (bit 0 = 1), the encoder uses the resolution set next to the register **Resolution [00 hex]**. Complete information at the register **Resolution [00 hex]** on page 35.

The default features of the SMAX linear encoder are:

- **Default resolution** = 0.1 mm
- **MTAX max. measuring length** = 600 mm
- **Max. number of information** = 6000 (13 bits)

To know whether the **Scaling function** is currently enabled, you can read the **Scaling** bit 0 of the **Status word [06 hex]**, see on page 47.

Code sequence

bit 1

This is intended to set if the count is increasing (count up information) either when the sensor moves in the direction indicated by the arrow in Figure 1 or when the sensor moves in reverse of the standard direction, i.e. in the opposite direction to the one shown by the arrow in Figure 1. Setting 0 (bit 1 = 0) causes the encoder counting to increment when the sensor moves as indicated by the arrow in Figure 1; setting 1 (bit 1 = 1) causes the encoder counting to increment when the sensor moves in reverse of the standard direction, i.e. in the opposite direction to the one shown by the arrow in Figure 1.

To know whether the **Code sequence** is currently enabled, you can read the **Counting direction** bit 1 of the **Status word [06 hex]**, see on page 47.



NOTE

Please note that when the count is decreasing (count down information) and you cross the zero, the value immediately after 0 will be 2^{N-1} , where N is the overall information expressed in bits. Let's suppose the overall information is 2000, i.e. 11 bits (11 bits are necessary to represent 2000 information). $2^{11} = 2048$, thus the value after 0 will be 2047.



bit 2 ... 7 Not used.

Byte 1 Not used.

Node address [04 hex]

[Register 5, Unsigned16, rw]

This register allows to set the node address of the device.

The default address is 1.

The address 0 is reserved to identify a "broadcast" exchange (Master sends a request to all Slaves connected to the Modbus network). See the "6.1 Modbus Master / Slaves protocol principle" section on page 20.

The Modbus Master node has no specific address, only the Slave nodes must have an address. Each Slave must have a unique address.

Addresses from 248 to 255 are reserved.

If you set the address 0, device will be set to 1 automatically.

If you set an address higher than 247, device will be set to 247 automatically.

Default = 1 (min. = 1, max. = 247)

Serial com baud rate [05 hex]

[Register 6, Unsigned16, rw]

This is meant to set the data transmission rate (baud rate and parity bit) of the serial port. The default value is 04 hex = Baud rate 19200 bit/s, Parity bit Even.

Default = 4 (min. = 0, max. = 8)

| Value | Baud rate | Parity bit |
|--------|------------|------------|
| 00 hex | 9600 bit/s | No parity |
| 01 hex | 9600 bit/s | Even |

| | | |
|-------------------------|--------------------|-------------|
| 02 hex | 9600 bit/s | Odd |
| 03 hex | 19200 bit/s | No parity |
| 04 hex (default) | 19200 bit/s | Even |
| 05 hex | 19200 bit/s | Odd |
| 06 hex | 115200 bit/s | No parity |
| 07 hex | 115200 bit/s | Even |
| 08 hex | 115200 bit/s | Odd |

Control Word [0A hex]

[Register 11, Unsigned16, rw]

This variable contains the commands to be sent in real time to the Slave in order to manage it.

Byte structure of the **Control Word [0A hex]** register:

| byte | MSB | | | LSB | | |
|------|-----|-----|-----|-----|-----|-----|
| bit | 15 | ... | 8 | 7 | ... | 0 |
| | msb | | lsb | msb | | lsb |

Byte 0 Not used.

Byte 1

Watch dog enable

bit 8 Setting the **Watch dog enable** bit to "1" causes the Watch dog function to be enabled; setting the **Watch dog enable** bit to "0" causes the Watch dog function to be disabled. When the Watch dog function is enabled, if the device does not receive any message from the Server within 1 second, the system forces an alarm condition (the **Watch dog** alarm is invoked to appear as soon as the Modbus network communication is restored). Watch dog function is a safety timer that uses a time-out to detect loop or deadlock conditions. For instance, should the serial communication be cut off while a command is still active and running Watch dog safety system immediately takes action and commands an alarm to be triggered.

Save parameters

bit 9 Data is saved on non-volatile memory at each rising edge of this bit; in other words, data save is performed each time

this bit is switched from logic level low ("0") to logic level high ("1").

Load default parameters

bit 10

Default parameters (they are set at the factory by Lika Electronic engineers to allow the operator to run the device for standard operation in a safe mode) are restored at each rising edge of this bit; in other words, the default parameters uploading operation is performed each time this bit is switched from logic level low ("0") to logic level high ("1"). The complete list of machine data and relevant default parameters preset by Lika Electronic engineers is available on page 57.



WARNING

The execution of this command causes all parameters which have been previously set to be overwritten!

Perform counting preset

bit 11

It allows to perform a homing operation of the encoder. As soon as the command is sent, the position value which will be transmitted for the current position of the encoder is the one set next to the register **Preset value [01 hex]** and all the previous and following positions will get a value according to it. Operation is performed at each rising edge of this bit, i.e. each time this bit is switched from logic level low ("0") to logic level high ("1"). When this command is sent, the current encoder position is temporarily saved in the register **Offset value [02 hex]**. For any further information on the preset function and the meaning and use of the related registers and commands **Preset value [01 hex]**, **Offset value [02 hex]** and **Perform counting preset** refer to page 37.



WARNING

To save the current encoder position in the register **Offset value [02 hex]** permanently, please execute the **Save parameters** command. Should the power be turned off without saving data, the **Offset value [02 hex]** will be lost!

bit 12 ... 15

Not used.



NOTE

Save the set values using **Save parameters** function.
Should the power be turned off all data not saved will be lost!

7.1.2 Input Register parameters

Input Register parameters are accessible for reading only; to read the value set in an input register parameter use the **04 Read Input Register** function code (reading of multiple input registers); for any further information on the implemented function codes refer to the "6.4.1 Implemented function codes" section on page 25.

Alarms register [00 hex]

[Register 1, Unsigned16, ro]

This variable is meant to show the alarms currently active in the device. When an alarm is active, also the LED shows visually the fault condition (see the section "4.4 Diagnostic LED (Figure 3)" on page 17).

Structure of the alarms byte:

| byte | MSB | | | LSB | | |
|------|-----|-----|-----|-----|-----|-----|
| bit | 15 | ... | 8 | 7 | ... | 0 |
| | msb | | lsb | msb | | lsb |

The available alarm error codes are listed hereafter:

Byte 0

Machine data not valid

bit 0 One or more parameters are not valid, set proper values to restore normal work condition. To see in detail which parameter is wrong please enter the **Wrong parameters list [03 hex]** register.

Flash memory error

bit 1 Flash memory internal error, it cannot be restored (bad checksum error, etc.).

Hall sensors error

bit 2 An error has occurred in the Hall sensors while reading the magnetic scale, the read value does not exist.

Mounting error

bit 3 The encoder is installed too far from the magnetic scale, the installation does not comply with the mounting tolerances between the sensor and the scale (see Figure 2). Refer to the "Mechanical installation" section on page 11.

bit 4 ... 7 Not used.

Byte 1

bit 8 ... 10 Not used.

Watch dog

bit 11 When the Watch dog function is enabled (**Watch dog enable** in **Control Word [0A hex]** is set to "1"), if the device does not receive any message from the Server within 1 second, the system forces an alarm condition (the **Watch dog** alarm bit is activated). The alarm is invoked to appear as soon as the Modbus network communication is restored. Watch dog function is a safety timer that uses a time-out to detect loop or deadlock conditions. For instance, should the serial communication be cut off while a command is still active and running, Watch dog safety system immediately takes action and commands an alarm to be triggered.

bits 12 ... 15 Not used.



NOTE

Please note that should the alarm be caused by wrong parameter values (see **Machine data not valid** and **Wrong parameters list [03 hex]** register), normal work status can be restored only after having set proper values. The **Watch dog** alarm is deleted automatically as soon as the communication is restored. The **Flash memory error** alarm cannot be reset.

Current position [01 hex]

[Register 2, Integer16, ro]

This register is meant to show the current position of the device at the moment when the request is sent. The output value is scaled according to the set scaling parameters, see **Scaling function** on page 40. Value is expressed in pulses.

Current velocity [02 hex]

[Register 3, Integer16, ro]

This register is not used and reserved for future use.

Wrong parameters list [03 hex]

[Register 4, Unsigned16, ro]

The operator has entered invalid data and the **Machine data not valid** alarm has been triggered. This variable is meant to show in detail (bit value = HIGH) which parameter is wrong, according to the following table.

Please note that the normal work status can be restored only after having set proper values.

| Bit | Parameter |
|----------|-------------------------------|
| 0 | Not used |
| 1 | Resolution [00 hex] |
| 2 | Preset value [01 hex] |
| 3 | Offset value [02 hex] |
| 4 | Operating parameters [03 hex] |
| 5 | Node address [04 hex] |
| 6 | Serial com baud rate [05 hex] |
| 7 ... 15 | Not used |

SW Version [04 hex]

[Register 5, Unsigned16, ro]

This is meant to show the software version of the encoder.

The major number shows the firmware edition, while the minor number shows the firmware revision.

The meaning of the 16 bits in the register is as follows:

| | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|--------------|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| Ms bit | | | | | | | | Ls bit | | | | | | | |
| Major number | | | | | | | | Minor number | | | | | | | |

Value 01 00 hex in hexadecimal notation corresponds to the binary representation 00000001 00000000 and has to be interpreted as: firmware edition 01, firmware revision 00.

HW Version [05 hex]

[Register 6, Unsigned16, ro]

This is meant to show the hardware (PCB) version of the encoder.

The major number shows the hardware edition, while the minor number shows the hardware revision.

The meaning of the 16 bits in the register is as follows:

| | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|--------------|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| Ms bit | | | | | | | | Ls bit | | | | | | | |
| Major number | | | | | | | | Minor number | | | | | | | |

Value 01 01 hex in hexadecimal notation corresponds to the binary representation 00000001 00000001 and has to be interpreted as: hardware edition 01, hardware revision 01.

Status word [06 hex]

[Register 7, Unsigned16, ro]

This register contains information about the current state of the device. The eight bits of Byte 0 (LSB) shows the currently set values of the **Operating parameters [03 hex]** register Byte 0 (LSB); while bit 8 of MSB is used to signal active alarms.

Byte structure of the **Status word [06 hex]** register:

| byte | MSB | | | LSB | | |
|------|-----|-----|-----|-----|-----|-----|
| bit | 15 | ... | 8 | 7 | ... | 0 |
| | msb | | lsb | msb | | lsb |

Byte 0

Scaling

bit 0

It shows the value which is currently set next to the parameter **Scaling function**. In other words, it is intended to show whether the scaling function is enabled or disabled. If the value is "0" the scaling function is disabled; if the value is "1" instead the scaling function is enabled. For any further information on setting and using the scaling function refer to the **Scaling function** parameter on page 40.

Counting direction

bit 1

It shows the value which is currently set next to the parameter **Code sequence**. If the bit is "0" the

output encoder position value has been set to increment when the sensor moves in the direction shown by the arrow in Figure 1; if the bit is "1" instead the output encoder position value has been set to increment when the sensor moves in reverse of the standard direction, i.e. in the opposite direction to the one shown by the arrow in Figure 1. For any further information on setting and using the counting direction function refer to the [Code sequence](#) parameter on page 40.

bit 2 ... 7

Not used.

Byte 1

Alarm

bit 8

If value is "1" one or more alarms are active; to know in detail which alarm has been triggered, see the [Alarms register \[00 hex\]](#) variable on page 44.

bit 9 ... 15

Not used.

7.2 Exception codes

When a Client device sends a request to a Server device it expects a normal response. One of four possible events can occur from the Master's query:

- If the Server device receives the request without a communication error and can handle the query normally, it returns a normal response.
- If the Server does not receive the request due to a communication error, no response is returned. The client program will eventually process a timeout condition for the request.
- If the Server receives the request, but detects a communication error (parity, CRC, ...), no response is returned. The client program will eventually process a timeout condition for the request.
- If the Server receives the request without a communication error, but cannot handle it (for example, if the request is to read a non-existent output or register), the Server will return an exception response informing the Client about the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

FUNCTION CODE FIELD: in a normal response, the Server echoes the function code of the original request in the function code field of the response. All function codes have a most significant bit (msb) of 0 (their values are all below 80 hexadecimal). In an exception response, the Server sets the msb of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response. With the function code's msb set, the client's application program can recognize the exception response and can examine the data field for the exception code.

DATA FIELD: in a normal response, the Server may return data or statistics in the data field (any information that was requested in the request). In an exception code, the Server returns an exception code in the data field. This defines the Server condition that caused the exception.

For any information on the available exception codes and their meaning refer to the "MODBUS Exception Responses" section on page 48 of the "MODBUS Application Protocol Specification V1.1b" document.

8 Firmware upgrade



WARNING

Firmware upgrading operation has to be accomplished by skilled and competent personnel. If a wrong or incompatible firmware program is installed then the unit may not be updated correctly, in some cases preventing the unit from working. It is mandatory to perform the upgrade according to the instructions provided in this section.

Before installation always ascertain that the firmware program is compatible with the hardware and software of the device. Furthermore never turn off power during flash upgrade.

8.1 Information on firmware upgrade

This operation allows to upgrade the unit firmware by downloading upgrading data to the flash memory.

Firmware is a software program which controls the functions and operation of a device; the firmware program, sometimes referred to as "user program", is stored in the flash memory integrated inside the unit. These encoders are designed so that the firmware can be easily updated by the user himself. This allows Lika Electronic to make new improved firmware programs available during the lifetime of the product.

Typical reasons for the release of new firmware programs are the necessity to make corrections, improve and even add new functionalities to the device.

The firmware upgrading program consists of a single file having .BIN extension to be downloaded to the unit using a dedicated software tool. Both files are released by Lika Electronic Technical Assistance & After Sale Service.

If the latest firmware version is already installed in the unit, you do not need to proceed with any new firmware installation. The current firmware version can be verified in the **SW Version [04 hex]** register after having connected to the unit (see on page 46).



NOTE

If you are not confident that you can perform the update successfully please contact Lika Electronic Technical Assistance & After Sale Service.

8.2 Preliminary operations and connections

Before proceeding with the firmware upgrade please ascertain that the following requirements are fully satisfied:

- the encoder is connected to a PC through a RS-485 serial COM port;
- the **Modbus_Firmware_Update_English.exe** executable file is installed in your PC;
- you have the .BIN file for firmware upgrade.



NOTE

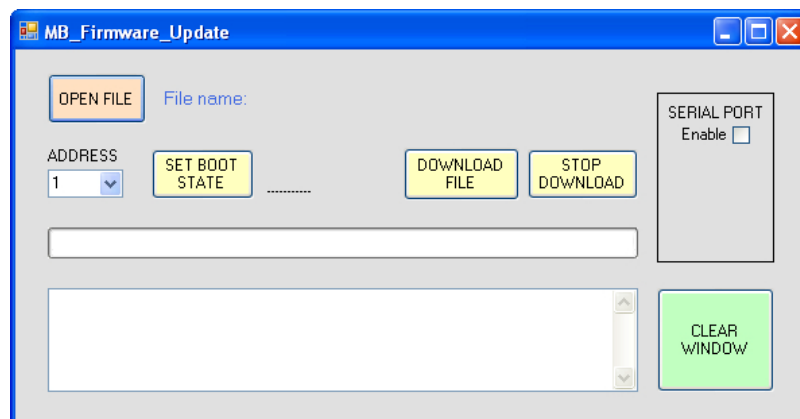
Before starting the program, connect the device to the personal computer through a serial port. The serial interface of the Modbus encoder unit is a RS-485 type connector, while the serial standard in the personal computers (when available) is a RS-232 type connector. Therefore you must install a RS-485 / RS-232 converter, easily available in the market. Should the personal computer not be equipped with a serial port (RS-232 or RS-485), you must install a USB / RS-485 converter, easily available in the market too. For any information on the connection scheme and the cable pinout refer to the instruction sheet provided with the converter.

On the ENCODER side the cable must be connected as described in the "Electrical connections" section on page 14. Please always make sure that the RD of the MODBUS ENCODER is cross-wired to the TD of the PC while the RD of the PC is cross-wired to the TD of the MODBUS ENCODER.

8.3 Launching the firmware upgrade process

To upgrade the firmware program please proceed as follows:

1. launch the MODBUS_FIRMWARE_UPDATE_ENGLISH.EXE executable file; the following page will appear;



2. in the page that appears on the screen enable the serial ports by selecting the SERIAL PORT ENABLE check box first and then choose the port the encoder is connected to in the drop-down box;
3. press the OPEN FILE button; once you press the button the Open dialogue box appears on the screen: open the folder where the firmware upgrading .BIN file released by Lika Electronic is located, select the file and confirm;
4. select the address of the device you want to upgrade in the ADDRESS drop-down combo box;
5. press the SET BOOT STATE button; if the encoder is connected properly and the system is able to enter the boot state successfully, a green square appears on the right of the button and the LED fitted in the encoder starts blinking at 2 Hz with duty cycle = 50%; the encoder is now ready for the firmware upgrade operation to start;



WARNING

Before installation always ascertain that the firmware program is compatible with the hardware and software of the device.
Never turn off power during flash upgrade.

6. press the DOWNLOAD FILE button to start the firmware upgrading process; the LED fitted in the encoder starts blinking at 5 Hz with duty cycle = 50%;
7. as soon as the operation is carried out successfully, a message appears on the screen;
8. turn the encoder power off and then on to complete the operation.



NOTE

While downloading the firmware upgrading program, unexpected conditions may arise which could lead to a failure of the installation process. When such a matter occurs, download process cannot be carried out successfully and thus the operation is aborted; the LED fitted in the encoder starts blinking at 10 Hz with duty cycle = 50%; you must turn the power off and on again to reset the device and restart the operation.



NOTE

When available, messages, warnings and alarms appear in the window in the bottom of the page. To clear the messages press the CLEAR WINDOW button.

9 Programming examples

Hereafter are some examples of both reading and writing parameters. Unless otherwise stated, all values are expressed in hexadecimal notation.

9.1 Using the **03 Read Holding Registers** function code



Example 1

Request to read the parameter **Preset value [01 hex]** (register 2) to the Slave having the node address 1.

Request PDU (in hexadecimal notation)

[01][03][00][01][00][01][D5][CA]

where:

[01] = Slave address

[03] = **03 Read Holding Registers** function code

[00][01] = starting address (**Preset value [01 hex]** parameter, register 2)

[00][01] = number of requested registers

[D5][CA] = CRC

Response PDU (in hexadecimal notation)

[01][03][02][05][DC][BA][8D]

where:

[01] = Slave address

[03] = **03 Read Holding Registers** function code

[02] = number of bytes (2 bytes for each register)

[05][DC] = value of register 2, 05 DC hex = 1500 dec

[BA][8D] = CRC

Preset value [01 hex] parameter (register 2) contains the value 05 DC hex, i.e. 1500 in decimal notation; in other words the value set in the **Preset value [01 hex]** parameter is 1500 dec.

9.2 Using the 04 Read Input Register function code



Example 1

Request to read the **Current position [01 hex]** parameter (register 2) to the Slave having the node address 1.

Request PDU (in hexadecimal notation)

[01][04][00][01][00][01][60][0A]

where:

[01] = Slave address

[04] = **04 Read Input Register** function code

[00][01] = starting address (**Current position [01 hex]** parameter, register 2)

[00][01] = number of requested registers

[60][0A] = CRC

Response PDU (in hexadecimal notation)

[01][04][02][13][C5][74][53]

where:

[01] = Slave address

[04] = **04 Read Input Register** function code

[02] = number of bytes (2 bytes for each register)

[13][C5] = value of register 2 **Current position [01 hex]**, 13 C5 hex = 5061 dec

[74][53] = CRC

Current position [01 hex] parameter (register 2) contains the value 13 C5 hex, i.e. 5061 in decimal notation.

9.3 Using the 06 Write Single Register function code



Example 1

Request to write in the **Operating parameters [03 hex]** register (register 4) to the Slave having the node address 1: we need to set the scaling function (**Scaling function** = 1) and the increasing counting when the sensor moves in the direction shown by the arrow in Figure 1 (**Code sequence** = 0). The value to set is 00 01 hex (= 0000 0000 0000 0001 in binary notation: bit 0 **Scaling function** = 1; bit 1 **Code sequence** = 0; the remaining bits are not used, therefore their value is 0).

Request PDU (in hexadecimal notation)

[01][06][00][03][00][01][B8][0A]

where:

[01] = Slave address

[06] = **06 Write Single Register** function code

[00][03] = address of the register (**Operating parameters [03 hex]** item, register 4)

[00][01] = value to be set in the register

[B8][0A] = CRC

Response PDU (in hexadecimal notation)

[01][06][00][03][00][01][B8][0A]

where:

[01] = Slave address

[06] = **06 Write Single Register** function code

[00][03] = address of the register (**Operating parameters [03 hex]** item, register 4)

[00][01] = value set in the register

[B8][0A] = CRC

The value 00 01 hex is set, i.e. 0000 0000 0000 0001 in binary notation: bit 0 **Scaling function** = 1; bit 1 **Code sequence** = 0; the remaining bits are not used, therefore their value is 0.

9.4 Using the 16 Write Multiple Registers function code



Example 1

Request to write the value 00 0A hex (= 10 dec) next to the parameter **Node address [04 hex]** (register 5) and the value 00 01 hex (= 1 dec = baud rate 9600 bit/s, parity bit Even) next to the parameter **Serial com baud rate [05 hex]** (register 6) of the Slave having currently the node address 1.

Request PDU (in hexadecimal notation)

[01][10][00][04][00][02][04][00][0A][00][01][32][29]

where:

[01] = Slave address

[10] = **16 Write Multiple Registers** function code

[00][04] = starting address (**Node address [04 hex]** parameter, register 5)

[00][02] = number of requested registers

[04] = number of bytes (2 bytes for each register)

[00][0A] = value to be set in the register 5, 00 0A hex = 10 dec

[00][01] = value to be set in the register 6, 00 01 hex = 1 dec

[32][29] = CRC

Response PDU (in hexadecimal notation)

[0A][10][00][04][00][02][6D][BD]

where:

[0A] = new Slave address

[10] = **16 Write Multiple Registers** function code

[00][04] = starting address (**Node address [04 hex]** parameter, register 5)

[00][02] = number of written registers

[6D][BD] = CRC

The values 00 0A hex, i.e. 10 in decimal notation, is set in the register 5 **Node address [04 hex]** parameter; while the value 00 01 hex, i.e. 1 in decimal notation = baud rate 9600 bit/s, parity bit Even, is set in the register 6 **Serial com baud rate [05 hex]** parameter. Thus the encoder is programmed to have node address 10 and data transmission rate = 1 = baud rate 9600 bit/s, parity bit Even.

10 Default parameters list

10.1 List of the Holding Registers with default value

| Registers list and address | Default value | | |
|--|---------------|--|--|
| Resolution [00 hex] hundredths of a mm | 10 | | |
| Preset value [01 hex] | 0 | | |
| Offset value [02 hex] | 0 | | |
| Scaling function in Operating parameters [03 hex] | 0 | | |
| Code sequence in Operating parameters [03 hex] | 0 | | |
| Node address [04 hex] | 1 | | |
| Serial com baud rate [05 hex] | 4 | | |
| Watch dog enable in Control Word [0A hex] | 0 | | |
| Save parameters in Control Word [0A hex] | - | | |
| Load default parameters in Control Word [0A hex] | - | | |
| Perform counting preset in Control Word [0A hex] | - | | |

10.2 List of the Input Registers

| Registers list and address | Description of the bits |
|--------------------------------|--|
| Alarms register [00 hex] | 0 Machine data not valid 1 Flash memory error 2 Hall sensors error 3 Mounting error 11 Watch dog |
| Current position [01 hex] | - |
| Current velocity [02 hex] | - |
| Wrong parameters list [03 hex] | 1 Resolution [00 hex] 2 Preset value [01 hex] 3 Offset value [02 hex] 4 Operating parameters [03 hex] 5 Node address [04 hex] 6 Serial com baud rate [05 hex] |
| SW Version [04 hex] | - |
| HW Version [05 hex] | - |
| Status word [06 hex] | 0 Scaling 1 Counting direction 8 Alarm |

This page intentionally left blank

This page intentionally left blank



| HW-SW release | Document release | Description |
|---------------|------------------|---|
| 1.0-1.0 | 1.0 | First issue |
| 1.0-1.0 | 1.1 | Hexadecimal value characters updating, general review, mounting tolerances correction |



LIKA Electronic
Via S. Lorenzo, 25
36010 Carrè (VI) • Italy
Tel. +39 0445 806600
Fax +39 0445 806699

Italy: eMail info@lika.it - www.lika.it
World: eMail info@lika.biz - www.lika.biz