FUSION Controller Communication Protocol User Manual

Part No. MAN-350016A

Part No: MAN-350016A
Status: Preliminary

Excelitas Technologies Corp.
2545 Railroad Street, Suite 300
Pittsburgh, PA 15222
United States
www.excelitas.com

Phone Europe
+49 (0) 551 6935-0
Phone North America
+1 (800) 429 0257
Phone Asia/Pacific
+65 64 99 7777
Technical support: Inspection@excelitas.com

Contents

List of Figures

List of Tables

Revision History

Revision Date	Rev#	Issued By	Change Details
2026-06-15	A	Timothy Duong	First release.

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms

ADU	Application Data Unit
CE	Conformité Européenne
CCW	Counter-clockwise
COM	Communication
CRC	Cyclic Redundancy Check
CW	Clockwise
DO	Digital Output
FC	Function Code
FCC	Federal Communication Commission
LED	Light Emitting Diode
PDU	Protocol Data Unit
PWM	Pulse Width Modulation
RMA	Return of Merchandise Authorization
RTU	Remote Terminal Unit

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER

1

Introduction

This chapter provides an overview of the Optem® FUSION Controller Communication Protocol using Modbus RTU serial communication.

The following topics are covered:

Introduction

This document outlines how to communicate with the Optem FUSION Controller using Modbus RTU serial communications.

Optem® FUSION Controller Communication Protocol Description

The Optem® FUSION Controller operates as a Modbus slave device, while the host acts as the Modbus master. Communication follows a request-response model.

Serial Communication Settings

The Serial Communication Settings of the Optem® FUSION Controller are Read-Only and cannot be changed.

Table 1 Serial Communication Settings

Field	Value
Baud Rate	115200
Data Bits	8
Stop Bits	1
Parity	None

Modbus RTU Frame Structure

A Modbus RTU message is a binary frame with the following format:

Table 2 Modbus RTU messages

Field	Size (bytes)	Description
Slave Address	1	Address of the target device (1-247).
Function Code	1	Defines the action (e.g., read coils, write register).

Table 2 Modbus RTU messages (continued)

Field	Size (bytes)	Description
Data	N	Parameters or payload (depends on function code).
CRC	2	Cyclic Redundancy Check (low byte first, then high byte).

General Syntax

[Slave Address] [Function Code] [Data...] [CRC Low] [CRC High]

Supported Function codes

This device supports the following Function codes:

- 0x03 - Read Holding Registers
- 0x10 - Write Multiple Registers

NOTE: *The Optem FUSION controller uses 32-bit registers, while Modbus RTU uses 16-bit registers. Therefore, each internal register is mapped to two Modbus registers. Read and write functions must always be performed on even number Modbus registers to preserve 32-bit register alignment.*

Frame examples

Example: Read Acceleration Value (Function Code 0x03)

Request Frame:

0x01 0x03 0x00 0x8A 0x00 0x02 0xE5 0xE1

- 0x01 → Slave address (device 1)
- 0x03 → Function code (read holding registers)
- 0x00 0x8A → Starting register address (0x008A = 138)
- 0x00 0x02 → Number of registers to read (2 registers: 138 and 139)
- 0xE5 0xE1 → CRC checksum

Response Frame:

0x01 0x03 0x04 0x00 0x13 0x88 0x00 0x6D 0xF6

- 0x01 → Slave address
- 0x03 → Function code (read holding registers)
- 0x04 → Byte count (4 bytes = 2 registers x 2 bytes each)

Register Data:

- 0x00 0x13 → Register 138 value = **19 (decimal)**
- 0x88 0x00 → Register 139 value = **34816 (decimal)**

32-bit value = (0x0013 << 16)

= 0x00130000 + 0x8800

= 0x00138800 = 1280000 in decimal

CRC:

- 0x6D 0xF6 → CRC checksum

Modbus RTU Protocol Timing Rules

- **Silent interval**: At least **3.5 character times** between frames.
- **Character spacing**: Max **1.5 character times** between bytes inside a frame.

Changes to Excelitas Products

Excelitas reserves the right to improve, change, or modify products without incurring any obligations to make changes to previous Excelitas equipment.

Technical Support

For technical support, please contact our Technical Support Team at *Inspection@excelitas.com*.

CHAPTER

2

Modbus Communication Using Library

This chapter describes how libraries like libmodbus simplify Modbus communication by handling frame creation and CRC.

The following topics are covered:

Sample Code (C/C++)

Below is sample code in C++ showing how to talk to the Optem FUSION Controller through Modbus RTU protocol, allowing for the reading and writing of parameters without using the Optem FUSION console or Optem FUSION SDK.

Table 3 libmodbus sample code (C++)

```
int main()
{
    modbus_t* ctx;
    uint16_t reg[10];

    int slave_id;
    int start_reg;
    int num_reg;

    std::cout << "Enter Slave ID: ";
    std::cin >> slave_id;

    std::cout << "Enter Start Register (hex, e.g. 8A): 0x";
    std::cin >> std::hex >> start_reg;

    std::cout << "Enter Number of Registers: ";
    std::cin >> num_reg;

     /*
      * Serial Port Examples
      *
      * Windows:
      *   COM1
      *   COM2
      *   COM3
      *
      * Linux:
      *   /dev/ttyUSB0
      *   /dev/ttyUSB1
      *   /dev/ttyS0
      */
      #ifdef _WIN32
            const char* serialPort = "COM3";
      #else
            const char* serialPort = "/dev/ttyUSB0";
      #endif
```

Table 3 libmodbus sample code (C++) (continued)

```
    ctx = modbus_new_rtu(serialPort, 115200, 'N', 8, 1);

    if (!ctx)
    {
        std::cout << "Context creation failed: " << modbus_strerror(errno) << std::endl;
        return -1;
    }
    modbus_set_slave(ctx, slave_id);
    modbus_set_debug(ctx, TRUE);
    modbus_set_response_timeout(ctx, 2, 0); // may need to add more delay.

    int rc1 = modbus_connect(ctx);

    if (rc1 == -1)
    {
        std::cout << "Connect failed: "<< modbus_strerror(errno) << std::endl;
        return -1;
    }

    std::cout << "Connected successfully!" << std::endl;

    int rc = modbus_read_registers(ctx, start_reg, num_reg, reg);

    if (rc == -1)
    {
    std::cout << "Read failed: " << modbus_strerror(errno) << std::endl;
    }
  else
    {
        int count = (rc < num_reg) ? rc : num_reg;

        for (int i = 0; i < count; i++)
        {
            std::cout << "Register[" << start_reg + i << "] = " << reg[i] << std::endl;
        }
    }
```

Table 3 libmodbus sample code (C++) (continued)

```
 // writing to a register
int start_reg1;

uint32_t value;

std::cout << "Enter the Start Register to write to  (hex, e.g. 8A): 0x";
std::cin >> std::hex >> start_reg1;
std::cout << " enter the new value to write" << std::endl;
std::cin >> std::dec >> value;

uint16_t values[2];

// High word
values[0] = (value >> 16) & 0xFFFF;// upper 16 bits

// Low word
values[1] = value & 0xFFFF;// lower 16 bits (64)

int rc_write = modbus_write_registers(ctx, start_reg1, 2, values);

if (rc_write == -1)
{
std::cout << "Write failed: "<< modbus_strerror(errno)<< std::endl;
}
else
{
std::cout << "Write successful!" << std::endl;
}
// reading back the new set value
uint16_t readback[2];

int rc_read = modbus_read_registers(ctx, start_reg1, 2, readback);

if (rc_read != -1)
{
      uint32_t value =((uint32_t)readback[0] << 16) | readback[1];

      std::cout << "Readback value = " << value << std::endl;
}
```

Table 3 libmodbus sample code (C++) (continued)

```
    modbus_close(ctx);
    modbus_free(ctx);

    return 0;
}
```

The Modbus RTU example supports both Windows and Linux operating systems. On Windows systems, serial communication is typically performed through COM ports such as COM1, COM2, or COM3. On Linux systems, USB-to-serial adapters are commonly assigned device names such as **/dev/ttyUSB0** or **/dev/ttyUSB1**, while native UART interfaces are typically available as **/dev/ttyS0**, **/dev/ttyS1**, and so on. Users should modify the serial port parameter in the **modbus_new_rtu()** function to match the serial interface connected to the target device. On Linux, the assigned serial device can be identified using commands such as **dmesg | grep tty** or **ls /dev/ttyUSB***, which display the available serial interfaces recognized by the operating system.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER

3

Testing Using Modbus Tools

This chapter describes how to use tools like QModMaster or Modbus Poll to test communication.

The following topics are covered:

Overview

QModMaster is an open-source Modbus master application used for testing and debugging Modbus RTU devices over RS485 communication.

It allows users to:

- Send Modbus commands
- Read device registers
- Write values to registers
- Monitor communication responses

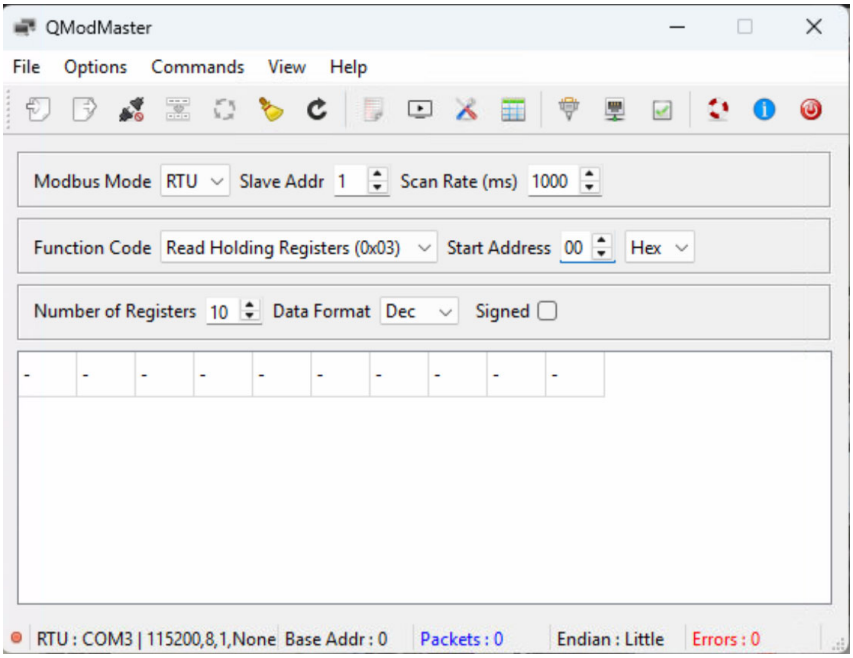

Figure 1 *QModMaster Window Screen*

Establishing a Serial Connection

Perform the following procedure to establish a Serial Connection.

1. Launch QModMaster, then Select RTU Serial Mode.

2. Configure the following parameters:

Table 4 Modbus RTU Setting Configuration Values

Parameter	Value
Serial Port	COM port connected to the controller
Baud Rate	115200
Data Bits	8
Parity	None
Stop Bits	Typically, 1

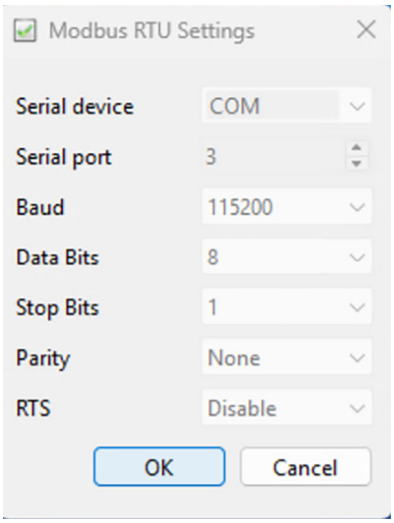

Figure 2 *Modbus RTU Setting configuration example*

3. After configuring these settings, click Connect to establish communication.

NOTE: *Make sure the Base Address is set to 0. This can be done by navigating to Options → Settings, then set the Base Address to 0.*

Serial Read Tab

The Serial Read tab allows reading registers from the device.

Table 5 Serial Read Fields

Field	Description
Slave ID	Device address
Function	Read function (e.g., 03- Read Holding Registers)
Start Address	Starting register address
Number of Registers	Quantity to read

Perform the following procedure to read the registers

- Enter the slave ID. Default value is 1.
- Select function code
- Enter the starting address
- Enter the number of registers

NOTE: *Ensure the entered starting address is an even number. Also ensure the number of requested registers should match the 32-bit register mapping.*

- Click Read
- The returned register values will appear in the results table.

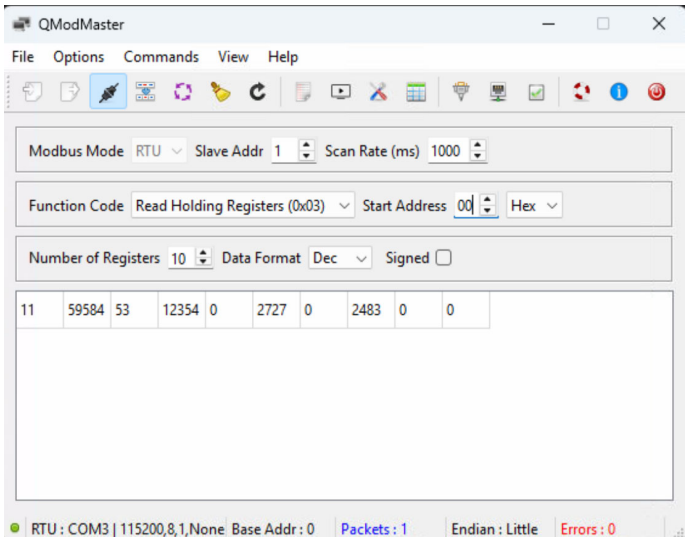

Figure 3 *Example returned register values after performing a serial read*

Serial Write Tab

The Serial Write Tab is used to modify existing register values.

Table 6 Serial Write Fields

Field	Description
Slave ID	Device address
Function	Write function (0x10)
Start Address	Target register
Number of Registers	Data to write

Perform the following procedure to write to the selected registers

- Enter the slave ID. Default value is 1.
- Enter the register address
- Click Write

Example: Write a new PWM Frequency Selection (Function Code 0x10)

Request Frame:

0x01 0x10 0x00 0x1A 0x00 0x02 0x04 0x00 0x00 0x00 0x63 0x32 0xF5

- 0x01 → Slave address (device 1)
- 0x10 → Function code (Write Multiple Registers)
- 0x00 0x1A → Starting register address (0x001A = 26)
- 0x00 0x02 → Number of registers to write (2 registers: 26 and 27)
- 0x04 → Byte count (4 bytes of register data)
- 0x00 0x00 → Register 26 value = 0 (upper 16 bits)
- 0x00 0x63 → Register 27 value = 99 (lower 16 bits)
- 0x32 0xF5 → CRC checksum

Write Response Frame:

0x01 0x10 0x00 0x1A 0x00 0x02 0x60 0x0F

- 0x01 → Slave address
- 0x10 → Function code (Write Multiple Registers)
- 0x00 0x1A → Starting register address acknowledged (26)
- 0x00 0x02 → Number of registers written (2)
- 0x60 0x0F → CRC checksum

Combined 32-bit value:

- 0x0000 0063 = 99

So:

- Register[26] = 0
- Register[27] = 99
- Combined Value = 99
- PWM Frequency = 100000 / (PWM_DIV + 1)= 1000 Hz (1 kHz)

CHAPTER

4

Modbus Registers Device Description

This section describes the Modbus System Control register map for the Optem FUSION controller.

The following topics are covered:

Overview

The Optem FUSION Controller uses 32-bit internal registers, whereas the Modbus RTU protocol operates with 16-bit registers. As a result, each internal register occupies two consecutive Modbus registers. To ensure proper access to the full 32-bit data, read and write operations are restricted to even-numbered Modbus register addresses.

More information regarding Register Mapping and Peripheral Modbus Address Registers can be found in "Motor Controller Registers" on page 35

Device control and status registers

System Control Registers

This section describes the Modbus System Control register map for the Optem® FUSION Controller. All register offsets are relative to the base address. Each register is 32 bits wide unless otherwise specified.

Table 7 Optem FUSION System Control Registers

Modbus Offset	Name	R/W	Type	Default
0x00	FPGA Bitstream WDI P/N	R	Unsigned	780480
0x02	Bitstream Version	R	ASCII	-
0x04 to 0x16	Reserved			

FPGA Bitstream WDI P/N (0x00)- Read Only

This register stores the FPGA bitstream part number (PN) as a 32-bit unsigned integer 780480.

Figure 4 *FPGA bitstream part number register diagram*

Table 8 FPGA bitstream part number register

Bits	Description
31:0	Unsigned integer (70xxxx range)

FPGA Bitstream Version (0x02)- Read Only

Indicates the FPGA bitstream revision using ASCII characters.

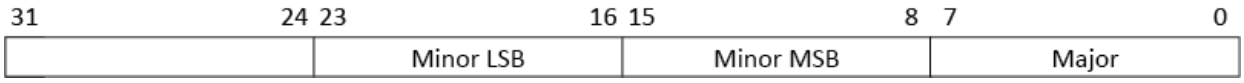

Figure 5 *FPGA bitstream revision register diagram*

Table 9 FPGA bitstream revision register

Bits	Description
31:24	Unused
23:16	Minor Revision LSB (ASCII)
15:8	Minor Revision MSB (ASCII)
7:0	Major Revision (ASCII)

LED channels control and status registers

Table 10 Optem FUSION LED channel control and status

Modbus offset	Name	R/W	Type	Default
0x18	Dual LED Driver Control	RW	Unsigned	256
0x1A	LEDS PWM Frequency Selection	RW	Unsigned	9
0x20	LED 1 PWM Duty Cycle	RW	Unsigned	0
0x22	LED1 Pulse Length	RW	Unsigned	0
0x24	LED1 Pulse Delay	RW	Unsigned	0
0x26	LED1 Current Setting	RW	Unsigned	1365
0x28	LED2 PWM Duty Cycle	RW	Unsigned	0
0x2A	LED2 Pulse Length	RW	Unsigned	0
0x2C	LED2 Pulse Delay	RW	Unsigned	0
0x2E	LED2 Current Setting	RW	Unsigned	1365

Dual LED Driver Control (0x18) - Read/Write

This register configures the LED driver operating mode, trigger source, parallel operation and enable controls. The trigger sources can be selected from the digital inputs available on the device. Since DI4 is reserved, the available inputs for selection are three single ended channels (DI1 to DI3) and one differential channel (DI5)

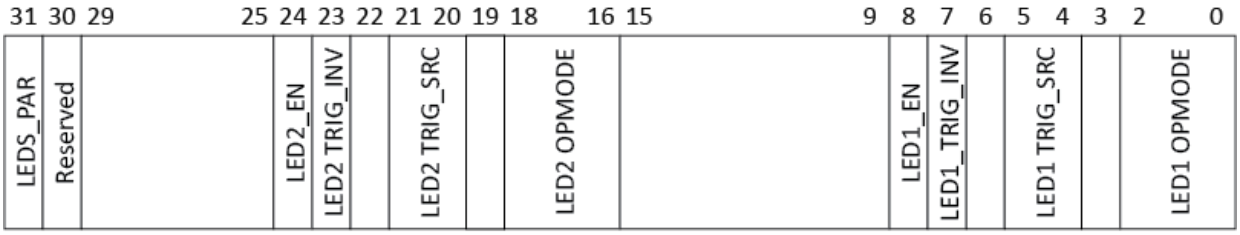

Figure 6 *Dual LED Driver Register Diagram*

Table 11 Dual LED Driver Register

Bits	Description
31	LEDs in Parallel Synchronous mode. When this bit is set, both LED channels are driven synchronously using the LED1 settings. Each LED current must be set the same when Parallel mode is used. All settings for LED2 (bits 24 to 16) are disregarded in parallel mode and settings from LED1 are used instead.
30	Reserved
25:29	Unused
24	LED2_EN, Enables the LED2 driver, Active high. Not required to be set in Parallel mode.
23	LED2_TRIG_INV, Inverts the trigger signal allowing active high or active low trigger: LED2_TRIG_INV = 0 → Active High trigger LED2_TRIG_INV = 1 → Active Low trigger
22	Unused
21:20	LED2 TRIG_SRC - LED trigger source selection. 0→DI1, 1→DI2, 2→DI3, 3→DI5
19	Unused
18:16	LED2 OPMODE - LED Operating mode, 0→CW, 1→PWM, 2→Pulse Triggered, 3→ Pulse Follow
9:15	Unused
8	LED1_EN, Enables the LED1 driver, Active high
7	LED1_TRIG_INV, Inverts the trigger signal allowing active high or active low trigger: LED1_TRIG_INV = 0 → Active High trigger LED1_TRIG_INV = 1 → Active Low trigger
6	Unused
5:4	LED1 TRIG_SRC - LED trigger source selection. 0→DI1, 1→DI2, 2→DI3, 3→DI5
3	Unused
2:0	LED1 OPMODE - LED Operating mode, 0→CW, 1→PWM, 2→Pulse Triggered, 3→ Pulse Follow

LEDS PWM Frequency Selection (0x1A) - Read/Write

The PWM generators use a common source clock of 100 MHz and 0.1% resolution which require 1000 counts for a full PWM cycle. The minimum PWM value that can be entered is 0, which corresponds to **100 KHz**. The maximum PWM value that can be entered is 65,535, which corresponds to a minimum frequency of **1.526 Hz**

The desired f_{PWM} is set by setting the PWM_DIV value between 0 and 16 bit integer unsigned (corresponding to 0 and 65,535), as shown in the equation below.

$$f_{PWM} = \frac{100KHZ}{PWM\_DIV + 1}$$

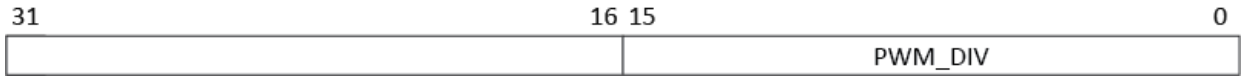

Figure 7 *LEDS PWM Frequency Selection Register Diagram*

Table 12 LEDS PWM Frequency Selection Register

Bits	Description
31:16	Unused
15:0	PWM_DIV value. Determined by the f_{PWM} equation shown above.

LED 1, LED 2 PWM Duty Cycle (0x20, 0x28) - Read/Write

PWM duty cycle in 0.1% increments. Maximum value: 1000 corresponding to 100.0%

Figure 8 *LEDS PWM Duty Cycle Register Format Diagram*

Table 13 LEDS PWM Duty Cycle Register Format

Bits	Description
31:10	Unused
9:0	PWM_DUTY value

LED 1, LED 2 Pulse Length (0x22, 0x2A) - Read/Write

Pulse length for pulse triggered mode in µs. Maximum value: 16.777215 seconds.

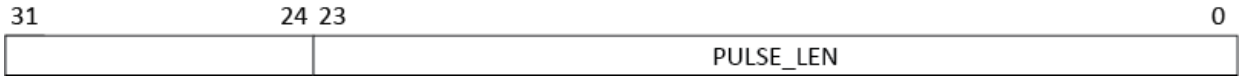

Figure 9 *Pulse Length Register Diagram for LED1 & LED2*

Table 14 Pulse Length Register

Bits	Description
31:24	Unused
23:0	PULSE_LEN value

LED 1, LED 2 Pulse Delay(0x24, 0x2C) - Read/Write

Pulse delay for pulse triggered mode in µs. Maximum value 16.777215 seconds.

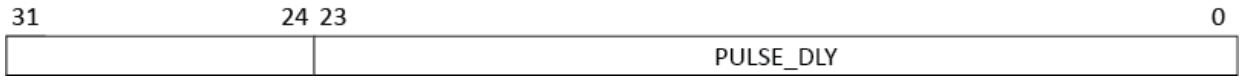

Figure 10 *Pulse Delay Register Diagram for LED1 & LED2*

Table 15 LEDS PWM Pulse Delay Register

Bits	Description
31:24	Unused
23:0	PWM_DLY value

LED 1, LED 2 Current Setting (0x26, 0x2E) - Read/Write

These registers holds the Digital to Analog (DAC) value for setting the LED driver analog dimming (current setting) value.

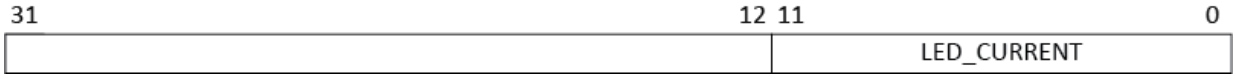

Figure 11 *Current Setting Register for LED1 & LED2*

Table 16 LED1, LED2 Current Setting Register

Bits	Description
31:12	Unused
11:0	LED_CURRENT, sets the current in 12 bit DAC, maximum is 1.5A (4095 = 1.5A)

Motor Controller Registers

Peripheral registers are mapped to Modbus addresses as follows:

Register Mapping:

Each 32-bit internal register is mapped to two consecutive 16-bit Modbus registers:

• Register N : Bits [31:16],Most Significant Word (MSW)

• Register N+1: Bits [15:0],Least Significant Word (LSW)

32-bit value = (MSW << 16) | LSW

Peripheral Modbus Address Ranges:

• 0x00 to 0x52: System registers and LED control Registers

• 0x80 to 0xAA: Motor Controller 1 registers

• 0xC0 to 0xEA: Motor Controller 2 registers

Table 17 contains a list of available motor controller registers.

Table 17 Motor Controller Registers

Modbus Offset	Name	R/W	Type	Default
0x00	Motion Status	R	Bitfield	NA
0x02	Motor Position	R	Signed	NA
0x04	Current Velocity	R	Signed	NA
0x06	Stop Position	R	Signed	NA
0x08	Motion Control	RW	Bitfield	0
0x0A	Acceleration	RW	Unsigned	1280000
0x0C	Reserved			
0x0E	Number of Steps	RW	Unsigned	128
0x10	Set Velocity	RW	Signed	199680
0x12	Driver Pulse Width	RW	Unsigned	1
0x14	CW Limit	RW	Signed	0
0x16	CCW Limit	RW	Signed	0
0x28	Motor Driver Control	RW	Bitfield	0
0x2A	Driver Current Control	RW	Unsigned	0

Motion Controller Operation

Position Control

Acceleration, Velocity, N registers are loaded with the desired acceleration, desired speed and the number of steps to move. The Driver Pulse Width must be loaded with the motor driver minimum pulse width or set up time (whichever is greater) in µs from the driver datasheet. The speed sign determines the movement direction: positive – forward motion, negative – reverse motion.

The controller will move the motor N steps following a trapezoidal velocity pattern with constant acceleration determined by the Acceleration setting and constant speed determined by the Set Point Velocity value.

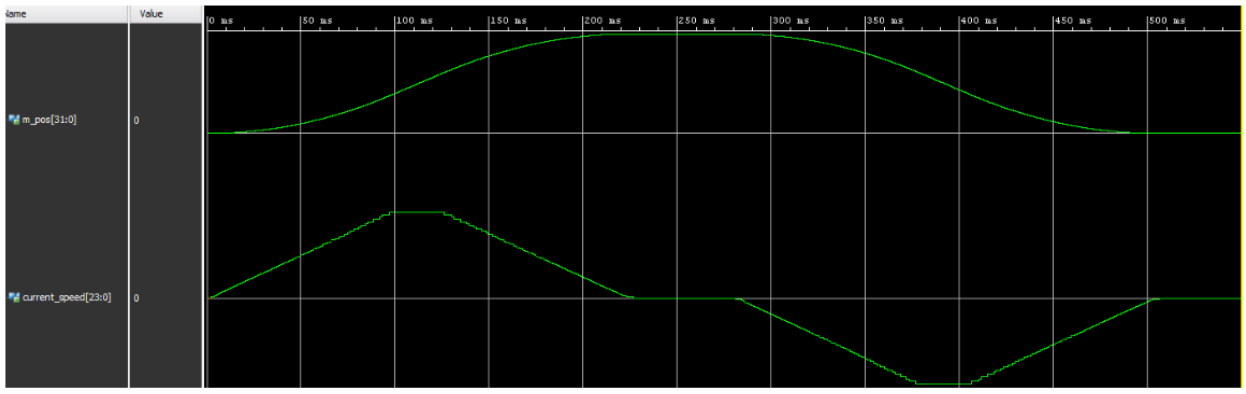

Figure 12 *Example speed profiles for forward and reverse motion in position control mode*

Motion Status

Table 18 provides information regarding the motion status registers

Figure 13 *Motion Status Register Diagram*

Table 18 Motion Status Register (Modbus Offset 0x00)

Bits	Description
31:12	Unused
11	SW_CW limit switch – active when SW CW Limit reached, see CW, CCW Limit Position
10	SW_CCW limit switch – active when SW CCW Limit Reached, see CW, CCW Limit Position
9	CW limit switch – active when tripped
8	CCW limit switch – active when tripped
7	Reserved
6	Motor busy

Table 18 Motion Status Register (Modbus Offset 0x00)

Bits	Description
5	Reserved
4	Reserved
3	Motion inhibited from move for one of the following reasons: switch inhibit request, clears when motor stops, even if the inhibit condition was removed while the motor was still moving.
2:0	Reserved

Motor Position

This register keeps track of the number of step pulses sent to the motor after the move command. If the moving direction is reverse then the value will be decremented. If the position tracking is enabled then the move command doesn't clear the motor position, instead the position clear command can be issued to clear the motor position. See Table 22 for more details.

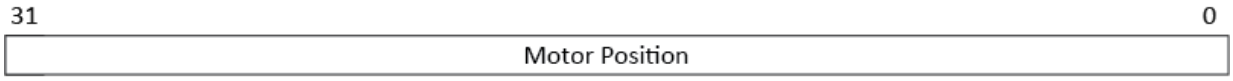

Figure 14 *Motor Position Register Diagram*

Table 19 Motor Position Register (Modbus Offset 0x02)

Bits	Description
31:0	Motor Position

Current Velocity

Motor current velocity. The motor controller calculates discrete velocity steps depending on the programmed acceleration. Therefore, the actual velocity may be slightly different from the programmed

velocity, not only because of the discrete steps but also because of truncation error resulting from integer calculation. This register provides a rough indication of the actual motor velocity.

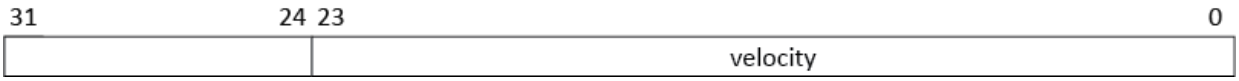

Figure 15 *Current Velocity Register Diagram*

Table 20 Current Velocity Register (Modbus Offset 0x04)

Bits	Description
31:24	Unused
23:0	Signed velocity value

Stop Position

This register stores the motor position the instant a stop condition was triggered (limit switch, soft limit, stop request).

It is useful when the limits switches trigger a stop for quickly determining the homing distance.

The final stop position is equal to the Stop Position plus the deceleration steps. The motor can be returned to the limit switch position by performing a move command with the number of steps equal with the difference.

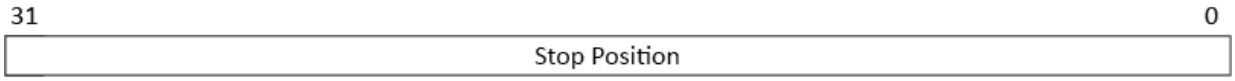

Figure 16 *Stop Position Register Diagram*

Table 21 Stop Position Register (Modbus Offset 0x06)

Bits	Description
31:0	Signed motor position when stop request occurred.

Motion Control Register

Controls motion execution and safety features.

Key Functions:

- Start motion
- Enable/disable limits
- Position reset

Figure 17 *Motion Control Register Diagram*

Table 22 Motion Control Register (Modbus Offset 0x08)

Bits	Description
31	CW invert input switch signal
30	CCW invert input switch signal
29	Enable SW Position Limits – when high, uses the CW and CCW registers to restrict motion in addition to the wired limit switches. Motion is stopped when either hard or soft limits are reached. If HWLIM_DISAB is high, then only the soft limits will apply
28	HWLIM_DISAB - Set high to disable the hard wired limit switches
27	SWAP_HWLIMSW - swap CCW and CW signals
10:26	Unused
9	CWLIM_DISAB – set high to disable hard CW limit switches - HWLIM_DISAB overrides this bit if set
8	CCWLIM_DISAB – set high to disable hard CCW limit switches - HWLIM_DISAB overrides this bit if set
7	Unused
6	Start MOVE - write one (1), automatically cleared
5	Disable Position Tracking – when high, prevents Motor Position from being accumulated. Instead, the position register will be automatically cleared at the beginning of the motion.

Table 22 Motion Control Register (Modbus Offset 0x08)

Bits	Description
4	Reserved
3:2	Unused
1	Motor Position Clear, write one (1), automatically cleared. Resets the accumulated motor position. This command is accepted **only** when the motor is not moving (**BUSY** bit is low)
0	Reserved

Acceleration Register

Figure 18 *Acceleration Register Diagram*

Motor acceleration, measured in microsteps per seconds squared (µsteps/s^2).

Table 23 Acceleration Register (Modbus Offset 0x0A)

Bits	Description
31:24	Unused
23:0	Desired acceleration value in µsteps/s^2

N - Number of steps to move

The motor can be told to move a precise number of steps by sending a corresponding number of pulses.

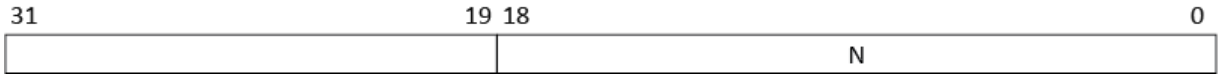

Figure 19 *Step Number Register Diagram*

Table 24 Step Number Register (Modbus Offset 0x0E)

Bits	Description
31:19	Unused
18:0	Number of pulses to generate

Set Point Velocity

Desired velocity in position control mode. A zero speed value prevents the motor from starting to move, even if all other conditions are met.

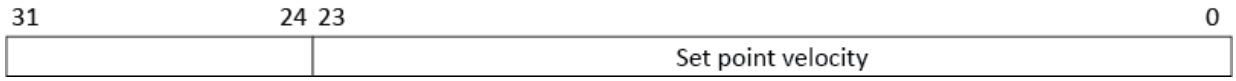

Figure 20 *Set Point Velocity Register Diagram*

Table 25 Set Point Velocity Register (Modbus Offset 0x10)

Bits	Description
31:24	Unused
23:0	Signed velocity value. The sign determines the direction of movement: positive corresponds to forward movement, and negative to reverse motion.

Driver Pulse Width

Minimum motor driver pulse width or minimum setup time (whichever is greater) as shown in the driver datasheet.

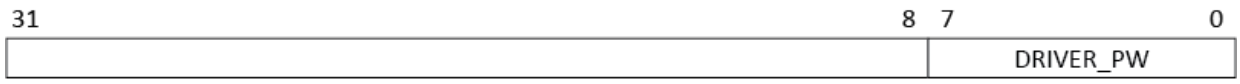

Figure 21 *Driver Pulse Width Register Diagram*

Table 26 Driver Pulse Width Register (Modbus Offset 0x12)

Bits	Description
31:8	Unused
7:0	DRIVER_PW, Unsigned pulse width value in µs.

CW,CCW Limit Position

This pair of two registers sets the traveling limits under software control. The software application must set bit SW_POSLIM_EN in the Motion Control Register for this feature to be enabled.

When one of the limits is reached, the motor will start decelerating to a stop. The final stop position depends on the speed at the moment the limit was hit as well as the programmed acceleration.

A move command in the opposite direction can be initiated but moving in the same direction that triggered the limit will not be permitted.

Position Tracking in Motion Control Register must not be disabled (DIS_POS_TRACK = 0) for this feature to work properly.

Figure 22 *CW_LIMIT/CCW_LIMIT Position Register Diagram*

Table 27 CW Limit Position Register (Modbus Offset 0x14)

Bits	Description
31:0	Signed limit position, normally positive for CW direction.

Table 28 CCW Limit Position Register (Modbus Offset 0x16)

Bits	Description
31:0	Signed limit position, normally negative for CCW direction.

Motor Driver Control

Controls the state of the motor driver. Set to 1 (Active High) to enable the motor driver; set to 0 to disable the motor driver.

Figure 23 *Motor Driver Control*

Table 29 Motor Driver Enable Register (Modbus Offset 0x28)

Bits	Description
31:1	Reserved
0	DRV_EN, Motor Driver Enable, Active High, enables the Motor driver chip

Motor Driver Reference Current

This register sets the PWM duty cycle of the motor driver current reference input.

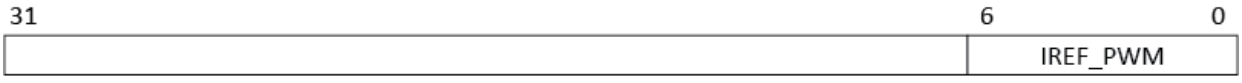

Figure 24 *Motor Driver Reference Current Register Diagram*

Table 30 Motor Driver Reference Current Register (Modbus Offset 0x2A)

Bits	Description
31:7	Unused
6:0	IREF_PWM in %, from 0 to 100%, 100% corresponds to 1.35A

Motor Homing (Modbus register address 0x9A)

The homing procedure uses several parameters stored in EEPROM as follows:

- **Home offset:** desired final offset from the detected limit position.
- **Travel range:** full range of motion.
- **Steps per millimeter:** motor steps per mm.
- **Velocity:** target velocity in mm/s

The homing procedure implemented in controller follows a four-step limit switch-based routine:

1) Find the hard limit: the motor fast jogs 1.2 times travel range which guarantees it hits the limit switch. The controller polls motion status until the drive is not longer busy and either CW or CCW limit is asserted.
2) Back off: the motor 1 mm back in the opposite direction and wait until motion completes.
3) Re-approach slowly for a clean edge: it slowly approaches the limit again with 0.05 times target speed. This step is completed when CW/CCW limit is asserted again.
4) Move to the configured home offset and reset position: the controller moves to the home offset with target velocity and the position is reset.

Used to trigger motor homing, writing value 0 triggers homing of motor 1 and writing value 1 for motor 2. Before the homing start or upon successful completion of homing, reading this register returns value 0. The Motor Homing Register has a special register address, as shown in Table 31

- **Homing Status**: Reading the motor homing register during the homing process returns the current homing status. A value of 0 indicates that homing has not started or has completed successfully. Any non-zero value represents an intermediate homing state while the controller is executing the homing procedure, including the start, wait, and check phases of the four homing steps.

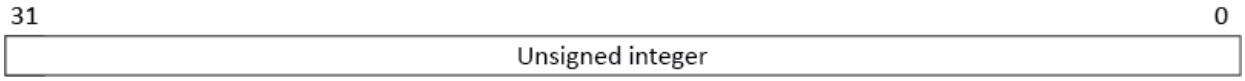

Figure 25 *Motor Homing Register*

Table 31 Motor Homing Register

Bits	Description
31:0	Unsigned integer

Motor Microstepping (Modbus register address 0xF6)

Allows setting the number of microsteps (µsteps) the motor takes for each full step.

The available number of microsteps are: 1, 2, 4, 8, 16, 32, 64, 128 and 256. Any other values will not be accepted.

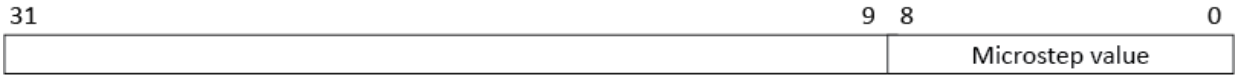

Figure 26 *Motor Microstepping Register Diagram*

Table 32 Motor Microstepping Register

Bits	Description
31:9	Unused
8:0	Microstep value.

Index

THIS PAGE INTENTIONALLY LEFT BLANK