# M4x4_R3 runtime library

Firmware documentation and usage


*Attila Zoltan Tamas*
*athomas*
*tomasati67@gmail.com*
*2025.*

# Content

# 1. Introduction

## 1.1 Objective

The **M4x4_R3** library provides the necessary control layer for the **MOLTINO 4x4** hardware, provided that the control board used is an **Arduino UNO R3 with an ATmega328 processor**.

This library allows the system to operate autonomously under the **MODBUS RTU protocol** via **RS485 communication**. It offers support for:

- 4 digital inputs with advanced functions,
- 4 analog inputs with 16-bit resolution,
- 4 digital outputs through relays,
- 4 analog outputs in the 0–10 V range.

Basic operation can be implemented through a simple Arduino sketch by invoking only two main functions. The library is distributed in precompiled form, but all input/output management functions, along with additional features, are exposed as **public functions**, enabling users to develop **PLC-type programs** that flexibly supervise and control input and output signals in an interdependent manner.

Using the public functions ensures access to all advanced capabilities offered by the library, while maintaining compatibility and extending development possibilities on the platform. Programming is carried out using the **Arduino IDE 2** environment.

## 1.2 Distribution and Support

**The M4x4_R3 library is delivered exclusively in precompiled binary format.** Functional extensions must be implemented through the use of public functions within external sketches. Updates and release notes will be published together with the corresponding binaries.

## 1.3 Development Environment Requirement

For the correct use of the precompiled **M4x4_R3** library file, it is essential to employ the **Arduino IDE version 2.x or higher**. Other versions of the development environment do not guarantee compatibility or proper firmware operation.

# 2. Installation of the M4x4_R3 Library

**The M4x4_R3 library is distributed exclusively in precompiled format and requires the prior installation of two external dependencies.**

## 2.1 Downloading the M4x4_R3 Library

1. Visit the official GitHub page of the M4x4_R3 library:
   **github.com/Moltino/Moltino**
2. Click the **“Code”** button and select **“Download ZIP”**.
3. Save the file to your computer

## 2.2 Installing Dependencies

Before installing the M4x4_R3 library, make sure the following libraries are installed via the Arduino IDE Library Manager:

- **Adafruit_ADS1X15** → required for the ADC converter management
- **WDT** → required for the Arduino sketch

**Installation procedure:**

1. Open the Arduino IDE.
2. Go to **Sketch → Include Library → Manage Libraries…**
3. Search for **Adafruit ADS1X15** and click **Install**.
4. Search for **WDT** and click **Install**.

## 2.3 Installing M4x4_R3

1. Open the Arduino IDE.
2. Go to **Sketch → Include Library → Add .ZIP Library…**
3. Locate and open the extracted **Moltino-main** folder.
4. Select the **M4x4_R3.zip** file downloaded from GitHub and click **Open**.
5. The IDE will add the **M4x4_R3** library to the development environment.

## 2.4 Installation Verification

- Open the menu: **Sketch → Include Library**.
- Verify that **M4x4_R3** appears in the list of installed libraries.

Verify that **M4x4_R3**, **Adafruit_ADS1X15** and **WDT** appear in the list. If all are present, the installation has been successful.

### 2.5 Usage in the Project

To use the library in your main sketch, it is only necessary to include:

#include <M4x4_R3.h>

The dependencie (**Adafruit_ADS1X15** ) are managed internally by **M4x4_R3**, so there is no need to include her manually.

## 3. Main Functions

The firmware is based on the **MODBUS RTU standard**, ensuring compatibility in communication and command interpretation. However, it also incorporates extended functionality: commands can be executed locally, without the need to receive them through the communication line.

Thanks to this feature, it is possible to develop independent Arduino sketches that allow the device to operate autonomously. Programming is mainly focused on assigning values to registers and reading results through the functions exposed in the library's public API, all supported by the **M4x4_R3 firmware**.

When used in MODBUS RTU mode, the system behaves as a fully integrated unit, leveraging all the capabilities offered by the **MOLTINO 4x4 hardware** together with the **M4x4_R3 firmware**.

**Note on Modbus addresses in Moltino:**
All addresses follow the **Modbus standard: base-0 offsets**. For example:

- Holding register number 0 → documented as **40000** in some SCADA systems.
- Holding register number 1 → documented as **40001** in some SCADA systems.

### 3.1 Device Identification and Communication

The **MOLTINO 4x4** uses the **MODBUS RTU standard over RS-485** for data exchange.

Each unit has a unique **slave ID** that allows it to send and receive messages across the MODBUS network:

- The allowed range for identification is **1 to 246**.
- The default value is **1**.
- If a value outside this range is programmed, the firmware will automatically reset it to **1**.

### Universal Programming Address

Identifier **247** is recognized by all **MOLTINO 4x4** devices. It is used exclusively as a resource for configuring a unit when its individual ID is unknown.

**Note:** Although address 247 allows simultaneous programming of all devices, its use is discouraged since it may cause conflicts and undesired behavior. This is because all devices will respond simultaneously to the master, potentially generating communication errors.

### Broadcast Communication

The **MOLTINO 4x4** supports broadcast commands, using identifier **0** as the address. In this mode, all devices execute the received command, but no response is sent back to the master.

The supported broadcast commands are:

- **05dec – WRITE SINGLE COIL**
- **06dec – WRITE SINGLE REGISTER**
- **15dec – WRITE MULTIPLE COILS**
- **65dec – RESET COUNTERS**

### Identification Configuration Register

The device ID number is configured in **HOLDING register 40033**.

## 3.2 Communication Ports

The **MOLTINO 4x4** natively incorporates support for **RS-485 communication**, with the circuitry integrated directly into the main board. The **M4x4_R3** library implements the standard **MODBUS RTU protocol** over this port.

Additionally, the **M4x4_R3** library allows data exchange through the **USB port of the Arduino UNO R3** board installed on the MOLTINO main board.

**Important:** The USB port does **not** supply power to the device. External power must be provided via one of the following:

- **13.4 VDC** through the Arduino board connector, or
- **24 VDC** through the MOLTINO power terminals.

The selection of the communication mode (**RS-485 or USB**) is made using the configuration **jumper** located on the main board.

- The USB port is used exclusively for **data exchange** and for **programming the Arduino UNO R3 board**.

### Serial Communication Parameters

The default configuration is:

- **Format:** 8-N-1 (8 data bits, no parity, 1 stop bit)
- **Baud rate:** 9600 bps

### Supported Baud Rates

The transmission and reception speeds supported by the **M4x4_R3** library are configured through **HOLDING register 40034**, by assigning the corresponding decimal code:

| Baud Rate (bps) | Code in Register 40034 |
|---|---|
| **9600** (default) | 96 |
| **19200** | 19 |
| **38400** | 38 |
| **57600** | 57 |
| **74800** | 74 |
| **115200** | 115 |
| **230400** | 230 |

## 3.3 MODBUS Commands and API Functions

The **MOLTINO 4x4** with the **M4x4_R3 library** accepts the following MODBUS commands:

- **Read Coils** (0x01)
- **Read Discrete Inputs** (0x02)
- **Read Holding Registers** (0x03)
- **Read Input Registers** (0x04)
- **Write Single Coil** (0x05)
- **Write Single Register** (0x06)
- **Write Multiple Coils** (0x0F)
- **Reset Counters** (0x41)

Each command represents a function that can be invoked in an Arduino sketch, giving control to the device as soon as the board is available.

**API Functions**

void begin(); *Initializes and configures the Arduino board for proper operation with the **MOLTINO 4x4** hardware.*

void rtuDataMonitoring(); *Handles standard **MODBUS communication**.*

void readCounters(); *Manages runtime measurement and logs pulses received on the digital inputs.*

void sendTextLcd(); *Optional – used to control an LCD display when one is installed.*

uint16_t readCoils(const uint16_t start, uint16_t quantity); *Returns the instantaneous and latched states of the digital inputs according to the requested register.*

uint16_t readInputReg(const uint16_t address); *Reads the value of the **16-bit analog inputs**, one at a time.*

uint16_t readHoldingReg(const uint16_t address);

*Reads values from:*

- Analog outputs
- 32-bit counters
- Frequency of each digital input
- 4–20 mA current loop error detection
- Configuration parameters of the MOLTINO

void writeSingleCoil(const uint16_t address, uint16_t value); *Activates or deactivates relays individually.*

void writeSingleReg(const uint16_t address, uint16_t value); *Writes to **HOLDING registers** one by one.*
*(Example: configuring the device, setting analog output voltage)*

void writeMultipleCoils(uint16_t outputs_value); *Activates or deactivates relays as a group.*

void resetCounters(const uint16_t register32, uint16_t Count); *Resets counters to **0**.*

- `register32` = address of the first 32-bit register
- `Count` = number of 32-bit registers to be cleared

## 3.4 MOLTINO 4x4 Configurations via the M4x4_R3 Runtime Library

The configuration settings of **MOLTINO** are equally effective whether the device is part of a **MODBUS network** or operating **autonomously**.

The **configuration register region** begins at **Holding Register 40033** and ends at **Holding Register 40047**.

All MOLTINO register addresses are listed in the **register map**, and in addition, they are defined in the **M4x4_R3.h** header file with descriptive names, making it easier for users to reference them in their sketches.

Configuration values are stored in the device's **EEPROM memory**, ensuring persistence across power cycles.

Configuration Registers (40033–40047)

| Address | Function |
|---|---|
| **40033** | Slave ID |
| **40034** | Serial baud rate |
| **40035** | Digital input mask (PNP/NPN) |
| **40036** | Counter memory (enable/disable) |
| **40037** | Counting direction mask (up/down) |
| **40038** | Input analog scaling factor |
| **40039** | Reserved |
| **40040** | 4–20 mA error bit/channel (enable/disable) |
| **40041** | Reserved |
| **40042** | Output analog scaling factor |
| **40043** | Analog input measurement format for LCD display (points, V, mA) |
| **40044** | Relay 1 delay |
| **40045** | Relay 2 delay |
| **40046** | Relay 3 delay |
| **40047** | Relay 4 delay |

## 3.5 Configuration Details

### 3.5.1 Digital Input Mask

The digital input mask allows configuring individually the activation mode of each digital input. The configuration uses the four least significant bits (LSBs) of the least significant byte (LSB).

- **Bit = 1** → The corresponding input is activated on a rising edge (**PNP mode**).
- **Bit = 0** → The corresponding input is activated on a falling edge (**NPN mode**).

This bit-wise configuration ensures flexible adaptation of MOLTINO to different types of sensors and field devices.

truth table

| INPUT | MASK | RECOGNIZED LOGICAL STATE |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |

### 3.5.2 Latched Inputs

The firmware includes a **latch function** for digital inputs, allowing detection of brief events that might otherwise go unnoticed.
When a digital input changes to an active state, this event is stored in memory and can be read through **registers 10008 to 10011**.

Once these registers are read by the user, the latched information is automatically cleared, restoring the register to its inactive state.

**Truth table of latched inputs:**

| Input state | Latch state | Description |
|---|---|---|
| 0 | 0 | Input inactive |
| 1 | 1 | Input active |
| 0 | 1 | Event detected (latched, pending readout) |
| 0 | 0 | Latch reset after read |

### 3.5.3 Counter Memory

Each digital input of the **MOLTINO 4x4 R3** is associated with a **32-bit pulse counter**, capable of processing signals up to **15 kHz**. The firmware includes a backup function that preserves the counter values, preventing data loss in the event of a device power-off.

This function is configured in binary mode:

- **0 → Memory disabled** (counter values are not saved).
- **1 → Memory enabled** (counter values are stored and automatically restored after a reboot).

When memory is enabled, the system recovers the previous counter values after a restart, ensuring the continuity of operations as if no interruption had occurred.

### 3.5.4 Counting Direction (Up/Down)

The mask allows configuring each counter individually. The **four least significant bits (LSBs)** of the least significant byte are used.

- **Bit 0** → Up-counting
- **Bit 1** → Down-counting

**Correspondence:**

- **Bit 0** → Counter 1
- **Bit 1** → Counter 2
- **Bit 2** → Counter 3
- **Bit 3** → Counter 4

This configuration enables flexible selection of counting direction for each digital input counter.

### 3.5.5 Analog Input Output Scaling

The system allows recalibrating the output scale of each analog input, primarily for compatibility purposes. The **A/D converter resolution** is **16 bits**, where the most significant bit indicates the polarity. Consequently, the maximum representable value per channel is **32,767**.

Configuration is performed through a **16-bit register**, which defines the maximum output scale value. This register can take values from **0 to 65,535**:

- **Default value:** 32,767
- **Configured value:** becomes the new maximum of the output scale

The recalibration method adjusts the output to a scale ranging from **0** to the value defined in the register, also allowing values lower than the default.

**Note:** This configuration does **not** modify the physical resolution of the converter; it only redefines the **logical output range**.

### 3.5.6 4–20 mA Error Bit/Channel (On/Off)

The **four least significant bits (LSBs)** of the least significant byte are used. Their function is to **enable or disable the error bits** that indicate whether the current in the corresponding channel is below **4 mA**.

**States:**

- **1** → Indicator bit enabled
- **0** → Indicator bit disabled

The corresponding channels start from the **least significant bit**:

- **Bit 0** → Channel 1 indicator bit
- **Bit 1** → Channel 2 indicator bit
- **Bit 2** → Channel 3 indicator bit
- **Bit 3** → Channel 4 indicator bit

This configuration allows monitoring of low-current errors for each 4–20 mA input channel individually.

### 3.5.7 Analog Output Scaling

The method is the same as for **analog inputs**, except that in this case it is used to **rescale the control of the analog outputs**.

The **physical resolution** is **8 bits**, but for compatibility purposes, the system can handle **control values up to 16 bits**.

### 3.5.8 Analog Input Measurement Format Displayed on the LCD

The LCD display is optional, but when installed, it allows monitoring **instantaneous values** and current configuration settings. This configuration determines the **unit of measurement** in which the LCD shows the values of the analog input measurements.

There are **three measurement formats**:

- **Voltage (Volt)** – up to 3 decimal places
- **Current (mA)** – up to 3 decimal places
- **Points** – raw values directly from the A/D converter

Configuration:

| Channel | *4* | | *3* | | *2* | | *1* | | |
|---|---|---|---|---|---|---|---|---|---|
| Bit / byte | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** | |
| raw point | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0: all channels projekted in points** |
| Value (dec) | 0 | | 0 | | 0 | | 0 | | |
| raw point | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | **85:all channels projekted in points** |
| Value (dec) | 64 | | 16 | | 4 | | 1 | | |
| Voltage | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | **250:all channels projekted in Volts** |
| Value (dec) | 128 | | 32 | | 8 | | 2 | | |
| mA | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **255:all channels projekted in mA** |
| Value (dec) | 192 | | 48 | | 12 | | 3 | | |

This configuration allows **each channel** to be displayed in a different measurement unit.

To calculate the configuration value, sum the decimal values corresponding to each channel as shown in the table. For example:

| Channel | Unit | Decimal Value |
|---|---|---|
| 1 | Volt | 2 |
| 2 | mA | 12 |
| 3 | Points | 0 |
| 4 | Volt | 128 |

Calculation:

2 + 12 + 0 + 128 = 142 decimal = 1000 1110 binary

### 3.5.9 Relay delay

The **MOLTINO 4x4** has **four digital outputs** implemented via relays, which can be configured in two operating modes:

- **Bistable**
- **Monostable**

Relay configuration allows:

- Defining the **operating mode**
- Setting a **delay time** ranging from **1 second** up to **18 hours, 12 minutes, and 15 seconds**

**Behavior:**

- If the delay time is set to **0**, the corresponding relay operates in **bistable mode**.
- If the configured time is **greater than 0**, the relay is activated and **automatically deactivated** once the delay time elapses.

Each relay has a **16-bit register**, where **each bit represents 1 second**, allowing precise definition of the timing duration.

# 4. Usage in Arduino Sketch

The **MOLTINO 4x4** allows any user to develop **custom firmware** tailored to specific needs. The **M4x4_R3 runtime library** is used by including and invoking it from an Arduino sketch.

The **examples** folder includes several practical sketches demonstrating the correct use of the library.

For basic operation, it is sufficient to copy and run the example sketch **Moltino4x4_R3**.

Within the sketch, the user should call the **API functions** provided by the **M4x4_R3 library** and process the obtained data using standard Arduino functions.

## 4.1 M4x4_R3.h File

The **M4x4_R3.h** file, located in the **M4x4_R3/src** folder, contains all the necessary elements to program an Arduino sketch using the runtime library.

This file includes:

- **Definitions of internal addresses** using symbolic names to facilitate identification and readability in the code.
- **Prototypes of all API functions**.
- **Definitions of texts displayed** on the LCD screen.

## 4.2 LCD Display Customization

The device includes a **2×16 LCD display**. Information is organized as follows:

- **Top line:** indicates the input/output corresponding to the displayed value.
- **Bottom line:** shows the numbered address and the value with its unit of measurement.

The values are provided by the **precompiled library**. However, the **top line is customizable**, allowing users to modify the text directly in the **M4x4_R3.h** file.

**Limitations and Recommendations:**

- The display supports **only 16 characters per line**; any additional characters will not be visible.
- It is recommended **not to modify the default configuration**, except when customizing the displayed names.