Modbus out - Modbus TCP output mailbox

SD038 | Updated on August 6, 2026

Jules PERRIN
Software Development Engineer
imperix

Table of Contents

The **Modbus_out block** transmits data over Ethernet using the Modbus TCP protocol on TCP port 502. The block can be configured to operate in one of two modes:

- **Master/Client**: the controller actively sends write requests to a remote Slave/Server device.
- **Slave/Server**: the controller updates its own internal registers, making the data available for an external Master/Client device to read.

The block supports the **Register types** listed below.

Register type	Access	Size
Discrete input	R	1 bit (encoded as 1 byte, using the LSB)
Coil (discrete output)	R/W	1 bit (encoded as 1 byte, using the LSB)
Input register	R	16 bit (2 bytes)
Holding register	R/W	16 bit (2 bytes)

A concrete application example is presented in PN203, which runs a Modbus client on a TPI8032 and uses an external master on a computer.

To receive data, use the Modbus_in block instead. Alternatively, for periodic transfers without explicit requests, especially at higher rates or with numerous variables, the Ethernet UDP output mailbox can be used instead.

Parameters

- **Mode**: Selects the mailbox's operating mode (*Master/Client* or *Slave/Server*).
- **IP Address** (Master mode only): Selects the IP address to whom the data will be sent.
- **Type**: Selects the Modbus register type to write to.
  - *Discrete input* (Slave mode only)
  - *Coil*
  - *Input register* (Slave mode only)
  - *Holding register*
- **Start address**: Sets the starting Modbus address for the range of registers to write (0–65535).
- **Signal(s) type**: Defines the data type of the input signal (*int8*, *int16*, *int32*, *uint8*, *uint16*, *uint32*, *float32*, or *float64*).
- **Number of signals**: Specifies the vector size of the data to be sent. The number of registers physically used is determined by the equivalent vector size in 16-bit words.
- **Data transmission mode**: Selects when the controller sends data.
  - *On-demand*: The user manually triggers each transmission via the trigger input (at the rising edge).
  - *Periodically*: The controller sends data at a fixed rate, regardless of whether the data has changed.
- **Tx frequency (Hz)**: Defines the transmission rate in Hz (only relevant in *Periodically* mode).
- **Endianness** (Master mode only): Defines the order of the 16-bit registers used to encode multi-register values.
- **Byte swap** (Master mode only): When enabled, swaps the two bytes inside each 16-bit register.

The four combinations of Byte order and swap produce the following byte sequences, where A is the most significant byte of a 32-bit value and D the least significant:

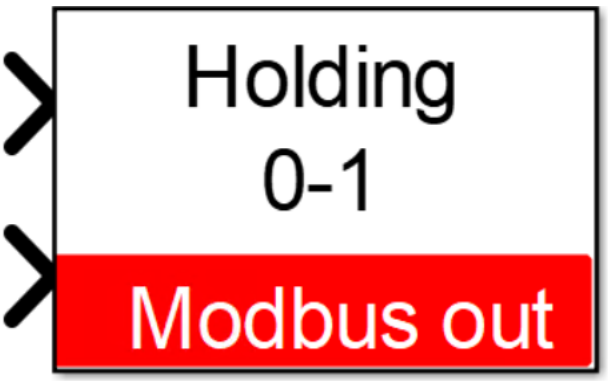
Endianness	Byte swap	Sequence
Big-endian	false	ABCD
Big-endian	true	BADC
Little-endian	true	CDAB
Little-endian	false	DCBA

In Slave/Server mode, the mailbox does not reorder registers; the application writes the raw register contents directly. Since imperix targets use a little-endian CPU, multi-register values written directly by the application typically correspond to the CDBA (Little-endian, byte swap) layout.

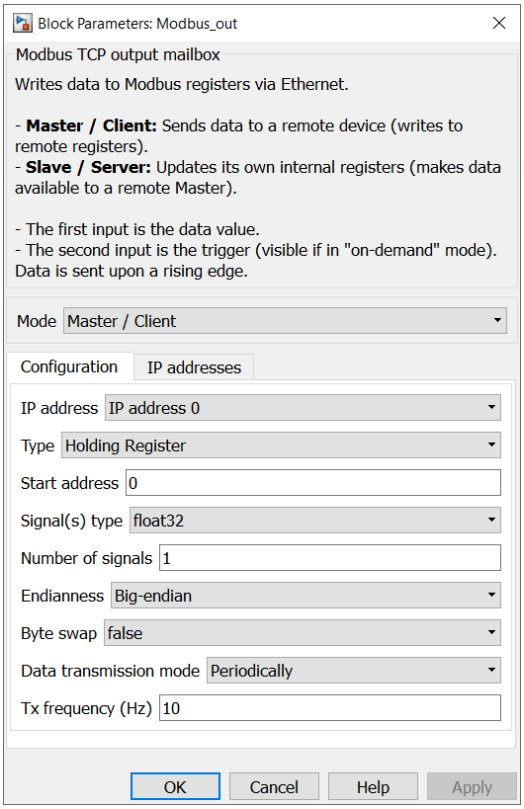
Simulink block

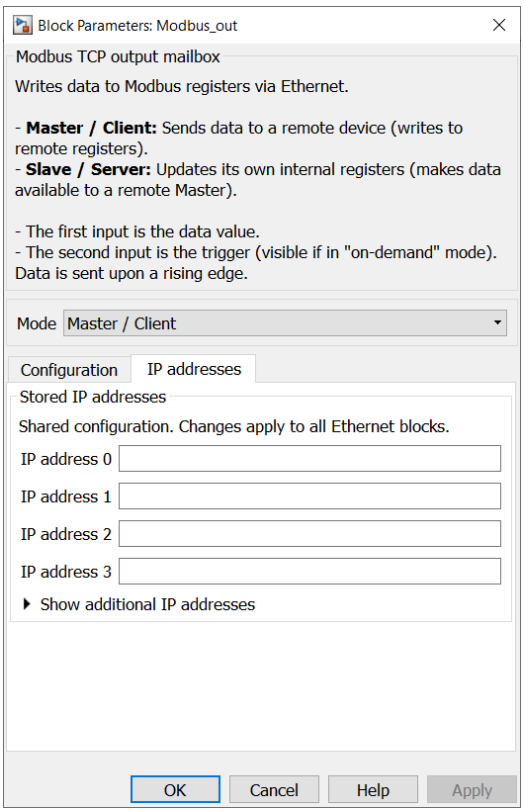
Signal specification

- The **data input** accepts a vector containing the data to write via Modbus TCP. Use the **Number of signals** parameter to set the vector length, and the **Signal(s) type** parameter to set the input data type.
- The second input is the **trigger** (only visible in “on-demand” mode). The block sends data on a rising edge of this signal.



Mask

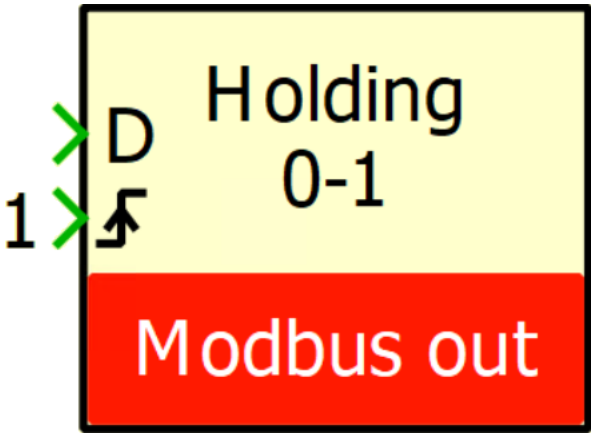




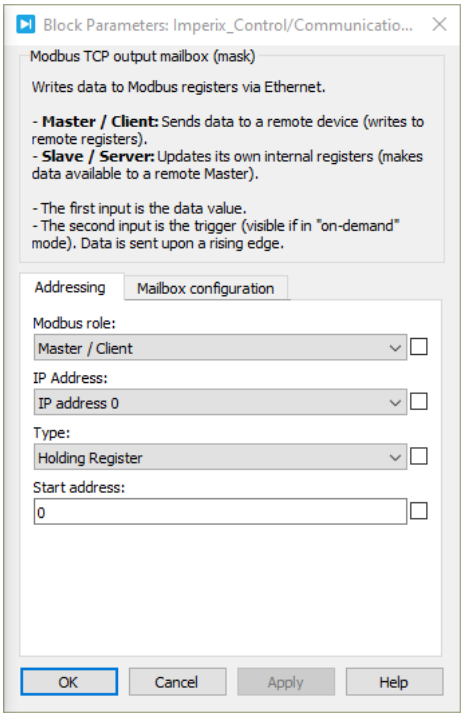
PLECS block

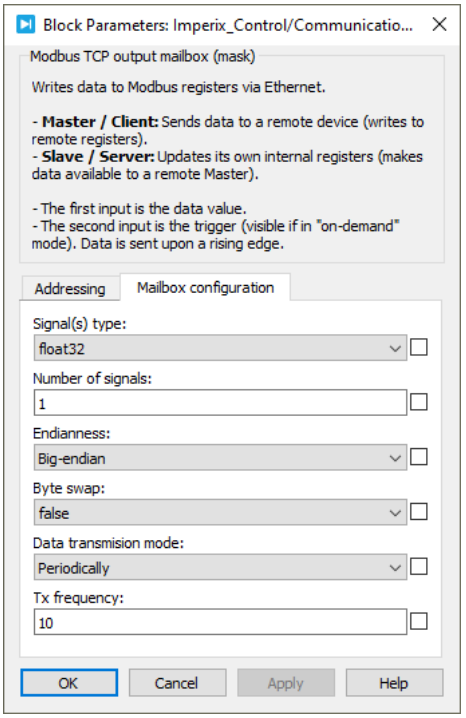
Signal specification

- The **data input** (D) accepts a vector containing the data to write via Modbus TCP. Use the **Number of signals** parameter to set the vector length, and the **Signal(s) type** parameter to set the input data type.
- The second input is the **trigger** (only visible in “on-demand” mode). The block sends data on a rising edge of this signal.



Mask





The Ethernet IP addresses can be configured in the controller's target window (*Coder* → *Coder option* → *Target*).

C++ functions

Master

`ModbusMaster_ConfigureHoldingRegisterOutputMailbox` **— Configure a Holding Register output mailbox**

```
bool ModbusMaster_ConfigureHoldingRegisterOutputMailbox(unsigned int mailboxId, const char* ip, uint16_t port, uns
```

Configures a Holding Register output mailbox for Modbus Master mode. Once configured, the controller sends the mailbox data to the specified register(s) on the remote Slave device.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: a unique ID that distinguishes this mailbox from all others. This ID must be unique throughout the code for all ETH, CAN, and Modbus input/output mailboxes.
- `ip`: IP address of the remote Modbus Slave (e.g., `"192.168.1.100"`).
- `port`: TCP port of the Modbus Slave (typically 502).
- `modbusAddress`: starting Modbus register address on the Slave (0–65535).
- `size`: size in bytes of the data to write (must be a multiple of 2, max 246 bytes).
- `regByteOrder`: register byte order — either `tModbusRegByteOrder::STANDARD` or `tModbusRegByteOrder::SWAPPED`.
- `pollingFrequency`: maximum transmission frequency in Hz.

Return value

- `bool`: returns `false` if the configuration fails (e.g., too many mailboxes or invalid parameters).

`ModbusMaster_ConfigureCoilOutputMailbox` **— Configure a Coil output mailbox**

```
bool ModbusMaster_ConfigureCoilOutputMailbox(unsigned int mailboxId, const char* ip, uint16_t port, unsigned int m
```

Configures a Coil output mailbox for Modbus Master mode. Once configured, the controller sends the mailbox coil values to the remote Slave device.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: a unique ID that distinguishes this mailbox from all others. This ID must be unique throughout the code for all ETH, CAN, and Modbus input/output mailboxes.
- `ip`: IP address of the remote Modbus Slave (e.g., `"192.168.1.100"`).
- `port`: TCP port of the Modbus Slave (typically 502).
- `modbusAddress`: starting Modbus coil address on the Slave (0–65535).
- `size`: number of coils to write.
- `pollingFrequency`: maximum transmission frequency in Hz.

Return value

- `bool`: returns `false` if the configuration fails.

`ModbusMaster_WriteHoldingRegister` **— Write Holding Register data**

```
bool ModbusMaster_WriteHoldingRegister(unsigned int mailboxId, const void* data, unsigned int size);Code language: C+
```

Writes data to a Holding Register output mailbox for transmission to the remote Slave.

Call this function during the control interrupt.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the data to write.
- `size`: size in bytes (must match the configured mailbox size).

Return value

- `bool`: returns `true` if the data was successfully queued for transmission.

`ModbusMaster_WriteCoil` **— Write Coil data**

```
bool ModbusMaster_WriteCoil(unsigned int mailboxId, const uint8_t* data, unsigned int size);Code language: C++ (cpp)
```

Writes data to a Coil output mailbox for transmission to the remote Slave. Each byte in the input array represents one coil value.

Call this function during the control interrupt.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the coil data array.
- `size`: number of coils (must match the configured mailbox size).

Return value

- `bool`: returns `true` if the data was successfully queued for transmission.

`ModbusMaster_ConfigureRegisterMailboxEndianness` — **Configure register mailbox endianness**

```
bool ModbusMaster_ConfigureRegisterMailboxEndianness(unsigned int mailboxId, tEndianness endianness, unsigned int
```

Configures how a Modbus Master holding-register or input-register mailbox interprets multi-register values. By default, register mailboxes are interpreted as independent 16-bit registers (`bytesPerValue = 2`, `endianness = BIG_ENDIAN`). The `endianness` parameter only affects values spanning more than one 16-bit register, so it has no effect when `bytesPerValue` is 1 or 2.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: unique mailbox identifier.
- `endianness`: register order inside each logical value (`LITTLE_ENDIAN` = low word first, `BIG_ENDIAN` = high word first).
- `bytesPerValue`: size in bytes of one interpreted value (must be 1, or an even number ≥ 2, and divide the mailbox size).

Return value

- `bool`: returns `false` if the mailbox does not exist, is not a master holding-register or input-register mailbox, or if the size is invalid. Otherwise, returns `true`.

Slave

`ModbusSlave_ConfigureHoldingRegisterOutputMailbox` **— Configure a Holding Register output mailbox**

```
bool ModbusSlave_ConfigureHoldingRegisterOutputMailbox(unsigned int mailboxId, unsigned int modbusAddress, unsigne
```

Configures a Holding Register output mailbox for Modbus Slave mode. After configuration, your code writes data to this mailbox, and an external Modbus Master can read it.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: a unique ID that distinguishes this mailbox from all others. This ID must be unique throughout the code for all ETH, CAN, and Modbus input/output mailboxes.
- `modbusAddress`: starting Modbus register address (0–65535).
- `size`: size in bytes (must be a multiple of 2, max 246 bytes).

Return value

- `bool`: returns `false` if the configuration fails.

`ModbusSlave_ConfigureInputRegisterOutputMailbox` **— Configure an Input Register output mailbox**

```
bool ModbusSlave_ConfigureInputRegisterOutputMailbox(unsigned int mailboxId, unsigned int modbusAddress, unsigned
```

Configures an Input Register output mailbox for Modbus Slave mode. After configuration, your code writes data to this mailbox, and an external Modbus Master can read it as input registers.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: a unique ID that distinguishes this mailbox from all others. This ID must be unique throughout the code for all ETH, CAN, and Modbus input/output mailboxes.
- `modbusAddress`: starting Modbus register address (0–65535).
- `size`: size in bytes (must be a multiple of 2, max 246 bytes).

Return value

- `bool`: returns `false` if the configuration fails.

`ModbusSlave_ConfigureCoilOutputMailbox` — Configure a Coil output mailbox

```
bool ModbusSlave_ConfigureCoilOutputMailbox(unsigned int mailboxId, unsigned int modbusAddress, unsigned int size)
```

Configures a Coil output mailbox for Modbus Slave mode. After configuration, your code writes coil values to this mailbox, and an external Modbus Master can read them.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: unique mailbox identifier.
- `modbusAddress`: starting Modbus coil address (0–65535).
- `size`: number of coils (each coil uses 1 byte in the data array).

Return value

- `bool`: returns `false` if the configuration fails.

`ModbusSlave_ConfigureDiscreteInputOutputMailbox` — Configure a Discrete Input output mailbox

```
bool ModbusSlave_ConfigureDiscreteInputOutputMailbox(unsigned int mailboxId, unsigned int modbusAddress, unsigned
```

Configures a Discrete Input output mailbox for Modbus Slave mode. After configuration, your code writes discrete input values to this mailbox, and an external Modbus Master can read them.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: unique mailbox identifier.
- `modbusAddress`: starting Modbus discrete input address (0–65535).
- `size`: number of discrete inputs (each uses 1 byte in the data array).

Return value

- `bool`: returns `false` if the configuration fails.

`ModbusSlave_WriteHoldingRegister` — Write Holding Register data

```
bool ModbusSlave_WriteHoldingRegister(unsigned int mailboxId, const void* data, unsigned int size);Code language: C++
```

Writes data to a Holding Register output mailbox, making it available for an external Master to read.

Call this function during the control interrupt.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the data to write.
- `size`: size in bytes (must match the configured mailbox size).

Return value

`bool`: returns `true` if the data was successfully written.

`ModbusSlave_WriteInputRegister` — Write Input Register data

```
bool ModbusSlave_WriteInputRegister(unsigned int mailboxId, const void* data, unsigned int size);Code language: C++ (
```

Writes data to an Input Register output mailbox, making it available for an external Master to read.

Call this function during the control interrupt.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the data to write.
- `size`: size in bytes (must match the configured mailbox size).

Return value

`bool`: returns `true` if the data was successfully written.

`ModbusSlave_WriteCoil` — Write Coil data

```
bool ModbusSlave_WriteCoil(unsigned int mailboxId, const uint8_t* data, unsigned int size);Code language: C++ (cpp)
```

Writes data to a Coil output mailbox, making it available for an external Master to read. Each byte in the input array represents one coil value.

Call this function during the control interrupt.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the coil data array.
- `size`: number of coils (must match the configured mailbox size).

Return value

`bool`: returns `true` if the data was successfully written.

`ModbusSlave_WriteDiscreteInput` — Write Discrete Input data

```
bool ModbusSlave_WriteDiscreteInput(unsigned int mailboxId, const uint8_t* data, unsigned int size);Code language: C+
```

Writes data to a Discrete Input output mailbox, making it available for an external Master to read. Each byte in the input array represents one discrete input value.

Call this function during the control interrupt.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the discrete input data array.
- `size`: number of discrete inputs (must match the configured mailbox size).

Return value

`bool`: returns `true` if the data was successfully written.