Modbus in - Modbus TCP input mailbox

SD037 | Updated on August 6, 2026

Jules PERRIN
Software Development Engineer
imperix

Table of Contents

The **Modbus_in 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 send data, use the Modbus_out block instead. Alternatively, for periodic transfers without explicit requests, especially at higher rates or with numerous variables, the Ethernet UDP input 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 read.
- **Type**: Selects the Modbus register type to read from
  - *Discrete input (Master mode only)*
  - *Coil*
  - *Input register (Master 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.
- **Initial value**: Sets the default output value.
- **Poll frequency (Hz)** (Master mode only): Defines the rate at which the controller polls the remote Modbus Slave for new data.
- **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:

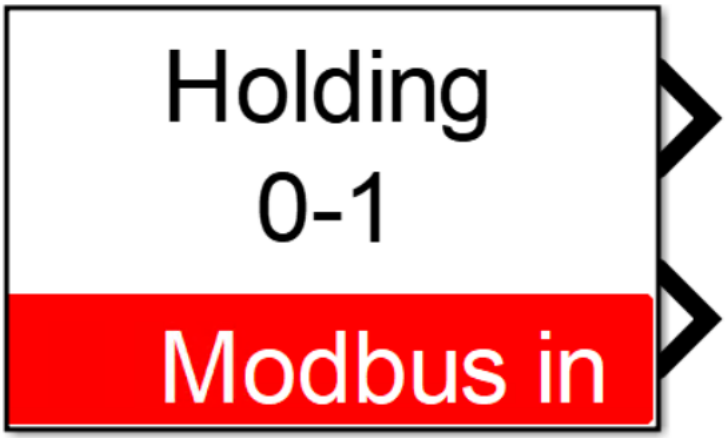
Endianness	Byte swap	Sequence
Big-endian	false	ABCD
Big-endian	true	BADC
Little-endian	true	CDAB
Little-endian	false	DCBA

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

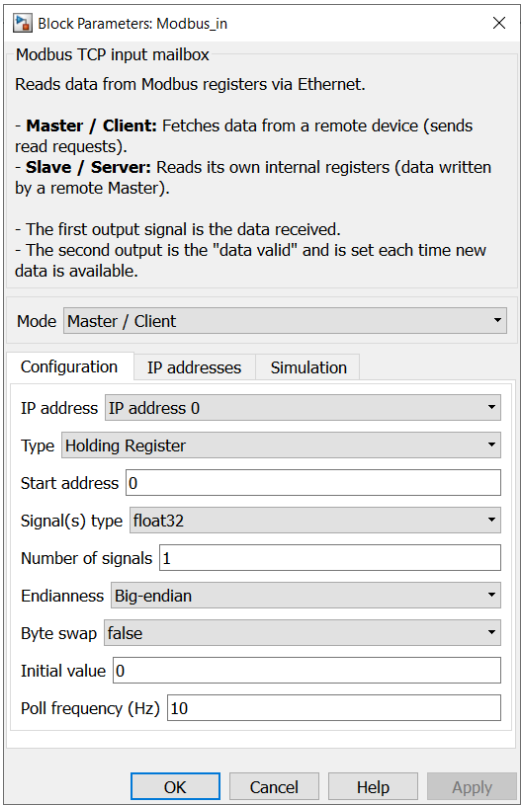
Simulink block

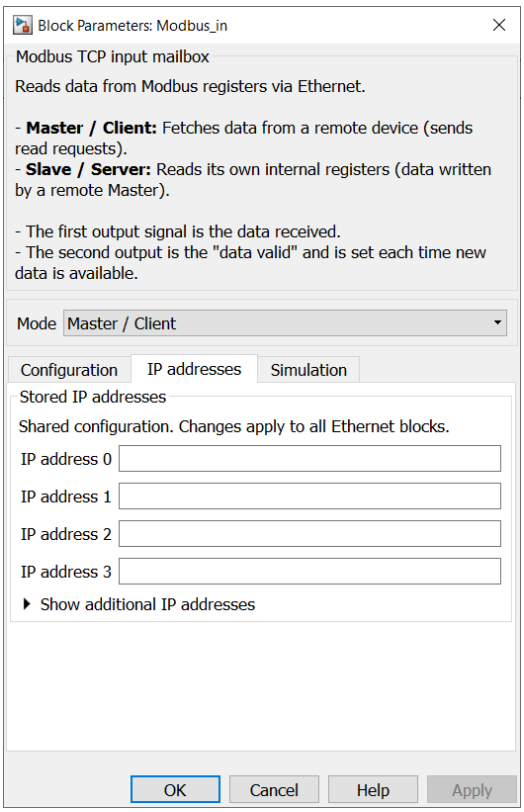
Signal specification

- The **data output** signal returns a vector containing the data read via Modbus TCP. Use the **Number of signals** parameter to set the vector length, and the **Signal(s) type** parameter to set the output data type.
- The second signal is the **data valid** output. It goes high each time new data arrives.



Mask

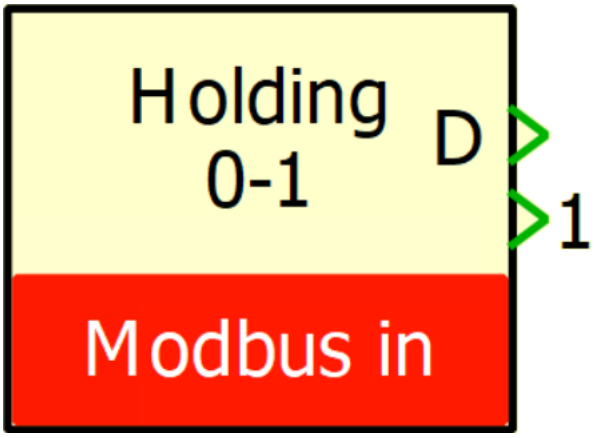




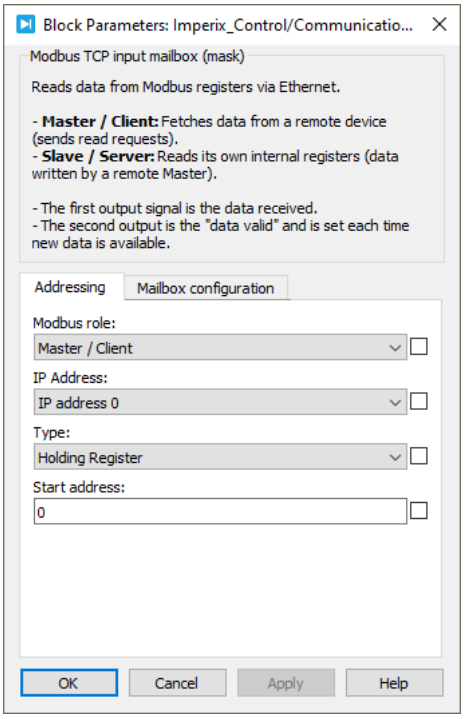
PLECS block

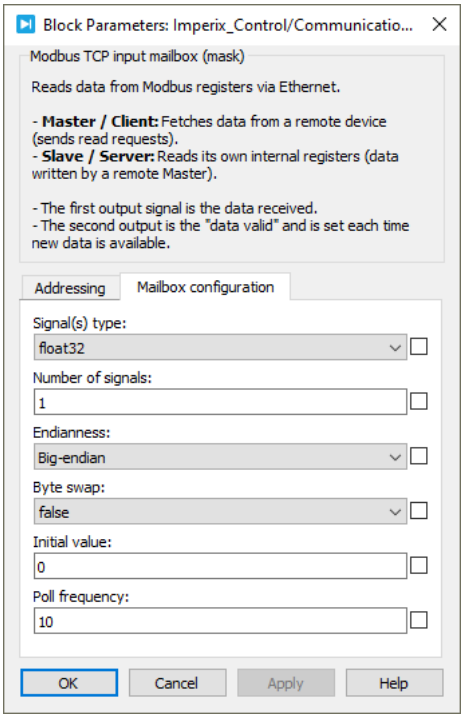
Signal specification

- The **data output** signal (D) returns a vector containing the data read via Modbus TCP. Use the **Number of signals** parameter to set the vector length, and the **Signal(s) type** parameter to set the output data type.
- The second signal is the **data valid** output. It goes high each time new data arrives.



Mask





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

C++ functions

Master

`ModbusMaster_ConfigureHoldingRegisterInputMailbox` **— Configure a Holding Register input mailbox**

```
bool ModbusMaster_ConfigureHoldingRegisterInputMailbox(unsigned int mailboxId, const char* ip, uint16_t port, unsi
```

Configures a Holding Register input mailbox for Modbus Master mode. The controller will periodically poll the specified register(s) on the remote Slave device.

It has to be called in `UserInit()`.

Parameters

- `mailboxId`: a unique ID used to distinguish mailboxes from each other. 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 read (must be a multiple of 2, max 246 bytes).
- `regByteOrder`: register byte order (`tModbusRegByteOrder::STANDARD` or `tModbusRegByteOrder::SWAPPED`).
- `pollingFrequency`: frequency at which to poll the Slave, in Hz.

Return value

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

`ModbusMaster_ConfigureInputRegisterInputMailbox` **— Configure an Input Register input mailbox**

```
bool ModbusMaster_ConfigureInputRegisterInputMailbox(unsigned int mailboxId, const char* ip, uint16_t port, unsign
```

Configures an Input Register input mailbox for Modbus Master mode. The controller will periodically poll the specified input register(s) on the remote Slave device.

It has to be called in `UserInit()`.

Parameters

- `mailboxId`: a unique ID used to distinguish mailboxes from each other. 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 read (must be a multiple of 2, max 246 bytes).
- `regByteOrder`: register byte order (`tModbusRegByteOrder::STANDARD` or `tModbusRegByteOrder::SWAPPED`).
- `pollingFrequency`: frequency at which to poll the Slave, in Hz.

Return value

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

`ModbusMaster_ConfigureCoilInputMailbox` **— Configure a Coil input mailbox**

```
bool ModbusMaster_ConfigureCoilInputMailbox(unsigned int mailboxId, const char* ip, uint16_t port, unsigned int mo
```

Configures a Coil input mailbox for Modbus Master mode. The controller will periodically poll the specified coil(s) on the remote Slave device.

It has to be called in `UserInit()`.

Parameters

- `mailboxId`: unique mailbox identifier.
- `ip`: IP address of the remote Modbus Slave.
- `port`: TCP port of the Modbus Slave (typically 502).
- `modbusAddress`: starting Modbus coil address on the Slave (0–65535).
- `size`: number of coils to read.
- `pollingFrequency`: frequency at which to poll the Slave, in Hz.

Return value

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

`ModbusMaster_ConfigureDiscreteInputInputMailbox` — Configure a Discrete Input input mailbox

```
bool ModbusMaster_ConfigureDiscreteInputInputMailbox(unsigned int mailboxId, const char* ip, uint16_t port, unsign
```

Configures a Discrete Input input mailbox for Modbus Master mode. The controller will periodically poll the specified discrete input(s) on the remote Slave device.

It has to be called in `UserInit()`.

Parameters

- `mailboxId`: unique mailbox identifier.
- `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 discrete input address on the Slave (0–65535).
- `size`: number of discrete inputs to read.
- `pollingFrequency`: frequency at which to poll the Slave, in Hz.

Return value

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

`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`.

`ModbusMaster_ConfigureInputMailboxInitialValue` — Configure a Master input mailbox initial value

```
bool ModbusMaster_ConfigureInputMailboxInitialValue(unsigned int mailboxId, void* data, unsigned int size);Code lan
```

Sets the initial value that the read functions return before any data arrives from the remote Slave. If the provided value is smaller than the mailbox size, it repeats to fill the entire mailbox.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the initial value data.
- `size`: size of the initial value in bytes (must divide the mailbox size evenly).

Return value

`bool`: returns `false` if the mailbox doesn't exist or the size is invalid.

`ModbusMaster_ReadHoldingRegister` — Read Holding Register data

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

Reads data from a Holding Register input mailbox that the controller has polled from the remote Slave. The output remains unchanged until new data arrives.

Call this function during the control interrupt.

Parameters

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

Return value

`bool`: returns `true` if new data has arrived since the last read, `false` otherwise.

`ModbusMaster_ReadInputRegister` **— Read Input Register data**

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

Reads data from an Input Register input mailbox that the controller has polled from the remote Slave. The output remains unchanged until new data arrives.

Call this function during the control interrupt.

Parameters

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

Return value

`bool`: returns `true` if new data has arrived since the last read, `false` otherwise.

`ModbusMaster_ReadCoil` **— Read Coil data**

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

Reads data from a Coil input mailbox that the controller has polled from the remote Slave. Each byte in the output array represents one coil value.

Call this function during the control interrupt.

Parameters

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

Return value

`bool`: returns `true` if new data has arrived since the last read, `false` otherwise.

`ModbusMaster_ReadDiscreteInput` **— Read Discrete Input data**

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

Reads data from a Discrete Input input mailbox that the controller has polled from the remote Slave. Each byte in the output array represents one discrete input value.

Call this function during the control interrupt.

Parameters

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

Return value

`bool`: returns `true` if new data has arrived since the last read, `false` otherwise.

Slave

`ModbusSlave_ConfigureHoldingRegisterInputMailbox` **— Configure a Holding Register input mailbox**

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

Configures a Holding Register input mailbox for Modbus Slave mode. After configuration, an external Modbus Master can write data to this mailbox, and your code can then 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_ConfigureCoilInputMailbox` — Configure a Coil input mailbox

```
bool ModbusSlave_ConfigureCoilInputMailbox(unsigned int mailboxId, unsigned int modbusAddress, unsigned int size);
```

Configures a Coil input mailbox for Modbus Slave mode. After configuration, an external Modbus Master can write coil values to this mailbox.

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_ConfigureInputMailboxInitialValue` — Configure a Slave input mailbox initial value

```
bool ModbusSlave_ConfigureInputMailboxInitialValue(unsigned int mailboxId, void* data, unsigned int size);Code langu
```

Sets the initial value for a Modbus Slave input mailbox. If the provided value is smaller than the mailbox size, it repeats to fill the entire mailbox.

Call this function in `UserInit()`.

Parameters

- `mailboxId`: unique mailbox identifier.
- `data`: pointer to the initial value data.
- `size`: size of the initial value in bytes (1–8 bytes, must divide the mailbox size evenly).

Return value

- `bool`: returns `false` if the mailbox doesn't exist or the size is invalid.

`ModbusSlave_ReadHoldingRegister` — Read Holding Register data

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

Reads data from a Holding Register input mailbox containing values that an external Modbus Master has written. The output remains unchanged until the Master writes new values.

Call this function during the control interrupt.

Parameters

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

Return value

- `bool`: returns `true` if new data has arrived since the last read, `false` otherwise.

`ModbusSlave_ReadCoil` — Read Coil data

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

Reads data from a Coil input mailbox containing values that an external Modbus Master has written.

Call this function during the control interrupt.

Parameters

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

Return value

`bool`: returns `true` if new data has arrived since the last read, `false` otherwise.