DFLDMB1 Rev 1.10 Datasheet

Modbus RTU/ASCII Dual Masters

Table of Contents

1 Overview

We know that only one Modbus Master is allowable for one Modbus RTU/ ASCII network.

For commissioning or troubleshooting, It is not convenient. We cannot easily examine the Data In Modbus network.

Of cause , we can use Modbus protocol analyzer or logic analyzer to examine traffic. However, the data we examined by this way will be not good to understand.

Usually every vendor of Modbus slave device will provide GUI software (PC Modbus Master) to examine/control device behaviour. In general , we cannot use this GUI software to examine device behaviour when device is put into Modbus network because of single Modbus Master limit (Modbus network already has one embedded or PLC modbus master) . However, with the help of our Modbus Dual Masters device (Part Number: DFLDMB1), we can use this GUI software to examine device behaviour even though there is another Modbus master controller in the network. In addition, our Modbus Dual Masters device can do baud rate transformation, and RTU/ASCII transformation. So two different Modbus network can be connected together even though they have different baud rate or one is RTU, the other is ASCII.

Actually, our "Modbus Dual Masters device" does not occupy any Modbus ID (address), it only provide a path for 2 masters (using UART2, UART3) to access Modbus network (using UART1). For Modbus Master, it is absolutely transparent.

For convenience, we can use Bluetooth UART (SPP Profile) to replace truly UART.

Please see application case 1, there is PC GUI software Modbus master to monitor network which has PLC or embedded Modbus master node.

Firstly, the original diagram without our "Modbus Dual Master" is shown below (Fig.1):

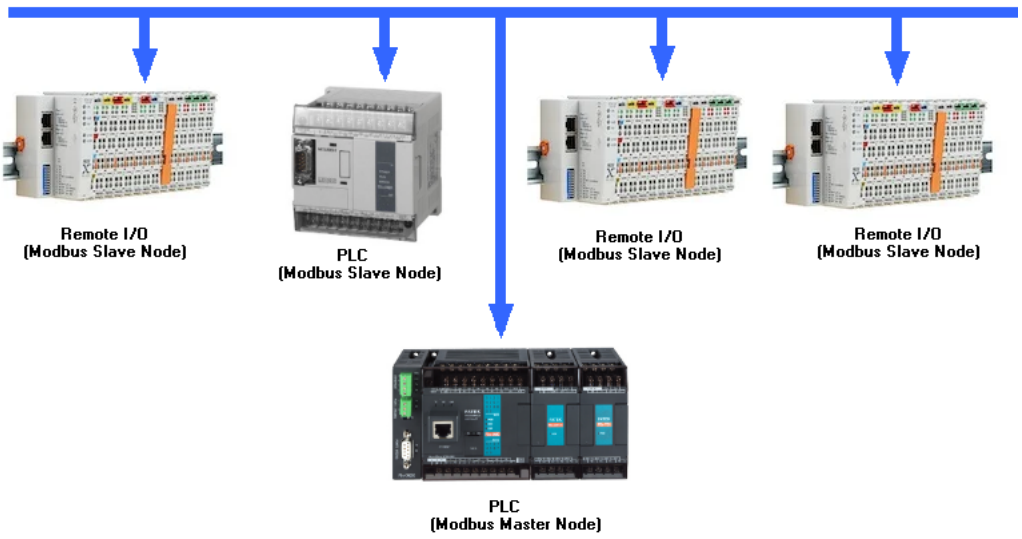


Fig. 1 Original Modbus network Diagram

From the diagram above, we cannot monitor Modbus network by another modbus master. However, we can do it when we add Dafulai Electronics "Modbus Dual Masters" device. Please see diagram Fig.2 below:

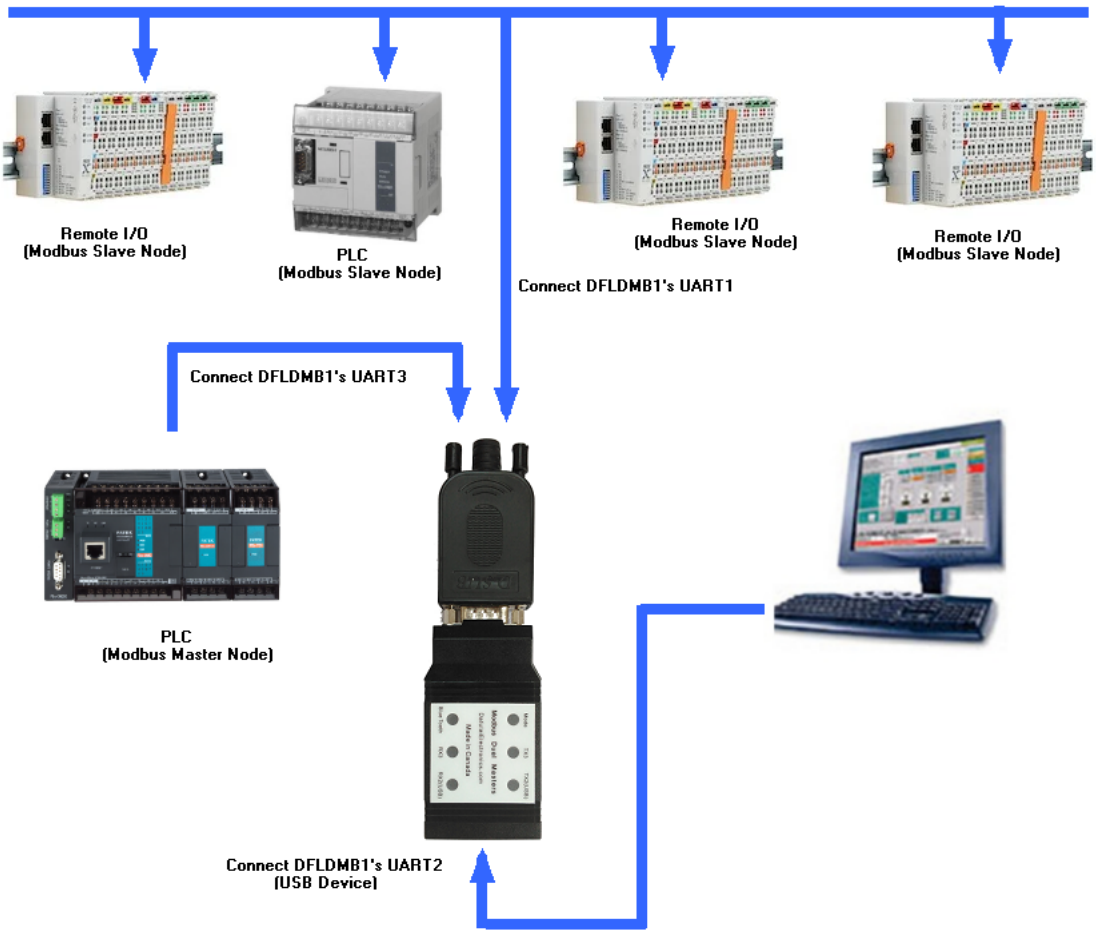


Fig. 2 Diagram with "Modbus Dual Masters" Adaptor.

Notes: If Modbus Master Time Out setting in Fig.1 has no margin (closed to border), you must make Modbus Master Time Out setting bigger in Fig.2 because time one master sends request may be the time the other master get response.

Please see application case 2, we use Bluetooth Modbus Master to connect network, See Fig. 3 for this situation

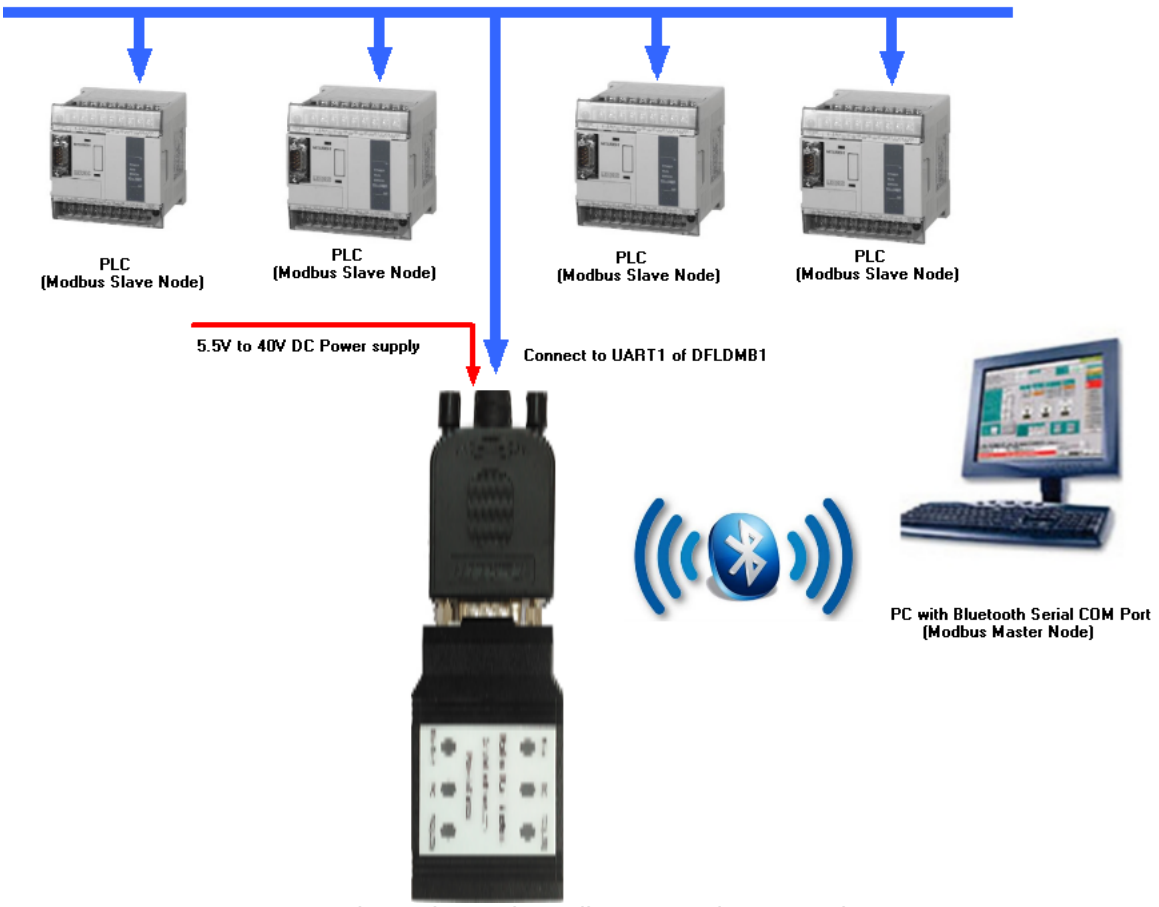


Fig.3 Bluetooth Modbus to monitor network

In application 2, you still can use USB to monitor too. If you use USB, you don't need 5.5V to 40V DC power supply. However, you cannot use RS485 of UART3 if you use Bluetooth.

Please see application case 3, in this situation, modbus network segment 1 has different baud rate or Modbus Type (RTU or ASCII) with modbus network segment 2. You cannot connect these 2 different Modbus networks together without DFLDMB1 adaptor. Please see Fig.4 below

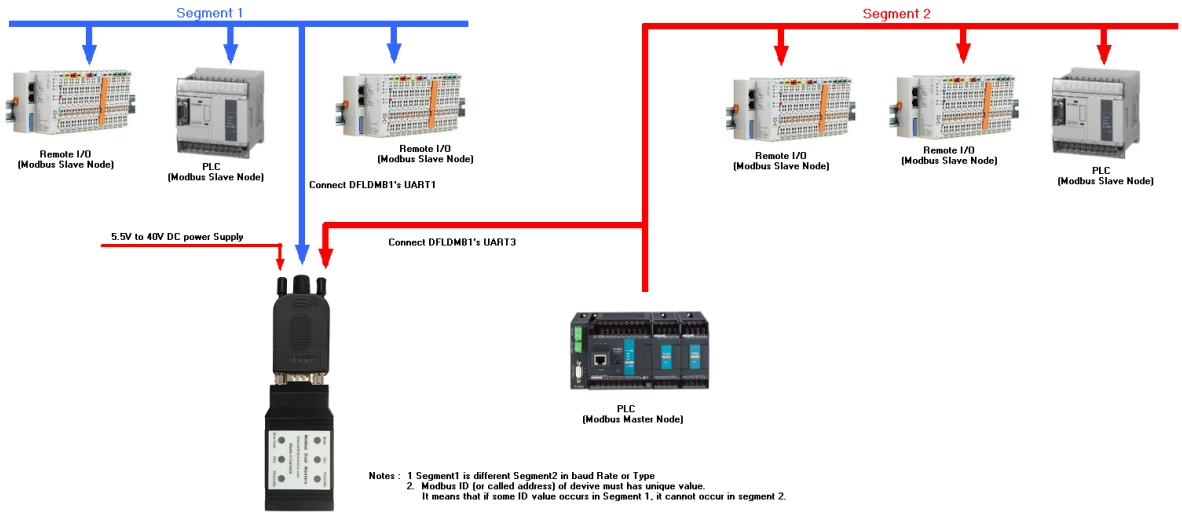


Fig.4 Connect different Modbus network Segment

In application 3, you still can use USB to monitor too. But you can only monitor Modbus Segment1, you cannot monitor Modbus Segment 2 If you use USB, you don't need 5.5V to 40V DC power supply. However, you cannot use Bluetooth.

Notes: You must make Modbus Master Time Out setting bigger for accessing segment1 slave node.

2 Features Highlights

- Support RS-232, RS-485, RS-422 for UART1 (Modbus Slave Network: All nodes are slaves)
- Support USB 2.0 for UART2 (Modbus first master access point: Connect here for the first master node)
- Support RS485 or Bluetooth EDR 4.0 for UART3 (Modbus second master access point: Connect here for the second master node)
- Configurable Modbus baud rates of 4800,9600, 19200, 28800,38400,43000,57600,115200
- Configurable Modbus data format for none, odd or even parity and 1 or 2 stop bits.
- Configurable Modbus type RTU or ASCII
- Provide free Matlab/Simulink Library to access Modbus server even though Modbus TCP server.
- Power supply range is 5.5V to 40VDC, it means both 12VDC and 24VDC are OK
- Software configure interface RS-232, RS-485, RS-422 without any hardware jumper.

3 Typical Application

- Use PC USB Modbus Master to monitor existing Modbus network which already has Modbus master node
- Use Bluetooth Modbus Master to access Modbus slave network.
- Connect 2 different baud rates or type (RTU or ASCII) of networks together.

4 Hardware

Fig 5 Hardware of DFLDMB1

4.1 Pin Assignment

From the application case above, we know that our DFLDMB1 ("Modbus Dual Masters") has 3 UARTs:

- UART1 : It is Modbus network connecting point. Physically, it can be configured as RS232 or RS485 or RS422.
- UART2: It is one modbus Master connecting point. Physically, it is USB Device (Logically, it is UART)
- UART3: It is another modbus Master connecting point. Physically, it can be Bluetooth SPP (Logically, it is UART) or it can be RS485. Both Bluetooth and RS485 share the same UART. So you cannot use the at the same time.

Male DB9 Connector:

Table

Pin	Name	Description
1	1st RS485+/1st RS422 TX+	This is for the first UART: RS485 + and RS422 TX +. This is Modbus network connecting point.
2	1st RS232 TX/1st RS485-/1st RS422 TX-	This is for the first UART: RS232 TX, RS485 - and RS422 TX-. This is Modbus network connecting point.

3	1st RS232 RX/1st RS422 RX+	This is for the first UART: RS232 RX and RS422 RX+. This is Modbus network connecting point.
4	1st RS422 RX-	This is for the first UART: RS422 RX-. This is Modbus network connecting point.
5	Gnd	Signal and Power ground. 5.5 to 40V DC Power supply input - Side.
6	Gnd	Signal and Power ground. 5.5 to 40V DC Power supply input - Side.
7	DC+	5.5 to 40V DC Power supply input + Side.
8	3rd RS485+	This is for the 3rd UART: RS485+. This is the 2nd master connecting point.
9	3rd RS485-	This is for the 3rd UART: RS485-. This is the 2nd master connecting point.

For the 2nd UART, it is USB Port which is the 1st master connecting point..

If you use our standard DB9 to 9 pins terminal cable (free of charge for this cable) , there are a label to tell you every pins definition, please see figure below:

Fig. 6 DB9 to Terminal Cable

4.2 LED Indication

There are 6 LEDs to indicate the DFLDMB1's state. 4 LEDs' color is Green. 4 LEDs' color is Red

1 Mode LED (Red color)

If this LED is bright, it means Modbus network (Uart1) is using RS485 interface.

If this LED is blinking, it means Modbus network (Uart1) is using RS422 interface.

If this LED is dark, it means Modbus network (Uart1) is using RS232 interface.

2 Bluetooth LED (Green color)
If this LED is half bright, it means Bluetooth is disabled.
If this LED is blinking, it means Bluetooth is searching for connecting, but actually it connects nothing.
If this LED is bright, it means Bluetooth is enabled and DFLDMB1 (this adaptor) has connected one master.

Notes: 1 This LED will still blink if you only pair successfully. You must connect Bluetooth COM port in your PC. This LED will be bright after your PC application software connects Bluetooth Serial Port. The baud rate of Bluetooth COM port can be any value when you set up baud rate in PC application (Not our configuration software) .
If you didn't open Bluetooth COM port, This LED will blink.
2. How to find and pair Bluetooth in your PC? it depends on your PC OS. Please use google to search solution for your operating system.
3 You cannot put DFLDMB1 into Metal Panel BOX for Bluetooth. Bluetooth cannot work when it is in metal container except your PC is in the same metal container.

3 USB TX or UART2 TX LED (Red color)
When USB or UART2 transmits any data, this LED will be on.

4 USB RX or UART2 RX LED (Green color)
When USB or UART2 receives any data, this LED will be on.

5 UART3 or Bluetooth TX LED (Red color)
When UART3 or Bluetooth transmits any data, this LED will be on.

6 UART3 or Bluetooth Rx LED (Green color)
When UART3 or Bluetooth received any data, this LED will be on.

5 Configuration Software

ConfigTool is a tool for configuration of DFLDMB1.

Configuration software can be downloaded from our website http://www.dafulaielectronics.com/ConfigTool.zip.
It is free of charge software. This software must be run under Windows Vista/Windows 7 /Windows 8/ Windows 10. After download, you should unzip the files, and don't need to install this software. You just double click on ConfigTool.exe to run it

Notes: For RS485/RS422/RS232, the default is RS485. The default Serial Port is 19200 N 1 (Baud rate :19200, No Parity, 1 stop bit, no flow control). Modbus RTU.

5.1 Configuration Parameters Setting

Step 1:

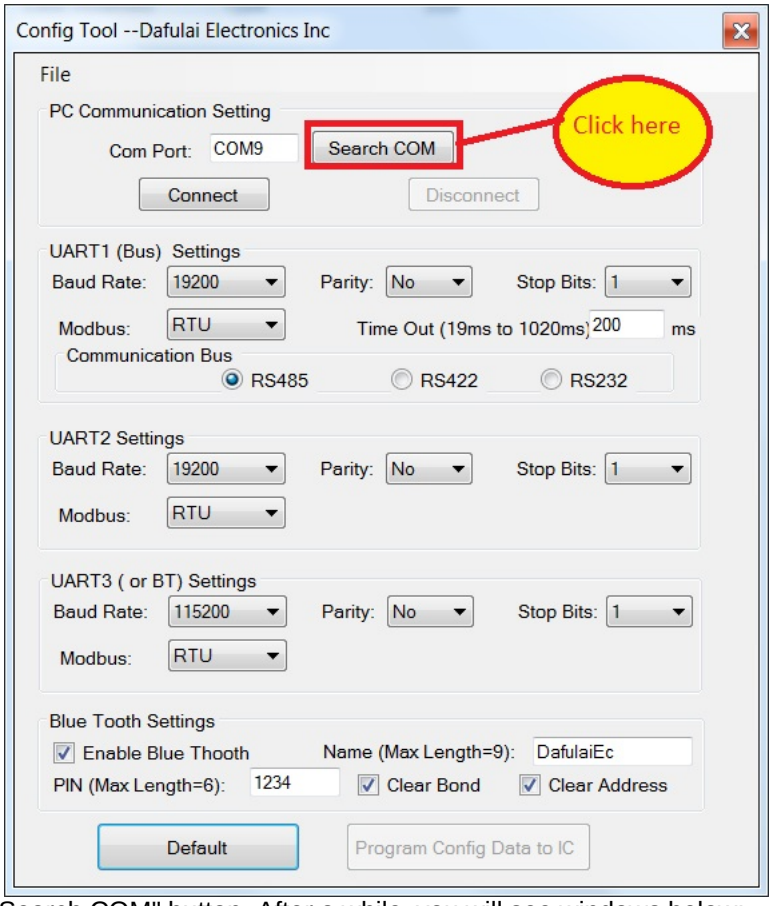
Plug in USB cable to DFLDMB1 and PC. unplug DB9 to make UART1 disconnecting (or you have no any Modbus slave nodes power on).

Make sure any Bluetooth Master didn't connect DFLDMB1 (You can Pair, but you cannot open Bluetooth COM port. Bluetooth LED will be blinking, Or Bluetooth LED will be half bright on

DFLDMB1)

Step 2:

Run Configuration by double click ConfigTool.exe. You will see the windows below:



Please click on "Search COM" button. After a while, you will see windows below:

Click OK, it will automatically choose first available COM Port in the dialog. Of cause, you can modify it to the other port.

Notes: When multiple COM ports are available, if you choose wrong COM Port, the configuration software behaviour will be unexpected.

The good way to identity Com port in In multiple COM ports available is that unplug USB cable and execute "Search COM" command again. If one Com port is not in the available COM

Ports, this COM Port will be our DFLJ1939Mod1, and Plug USB cable again

Step 3:

Please click "Connect" button to make PC connect DFLDMB1. Fill in or select the other parameters.

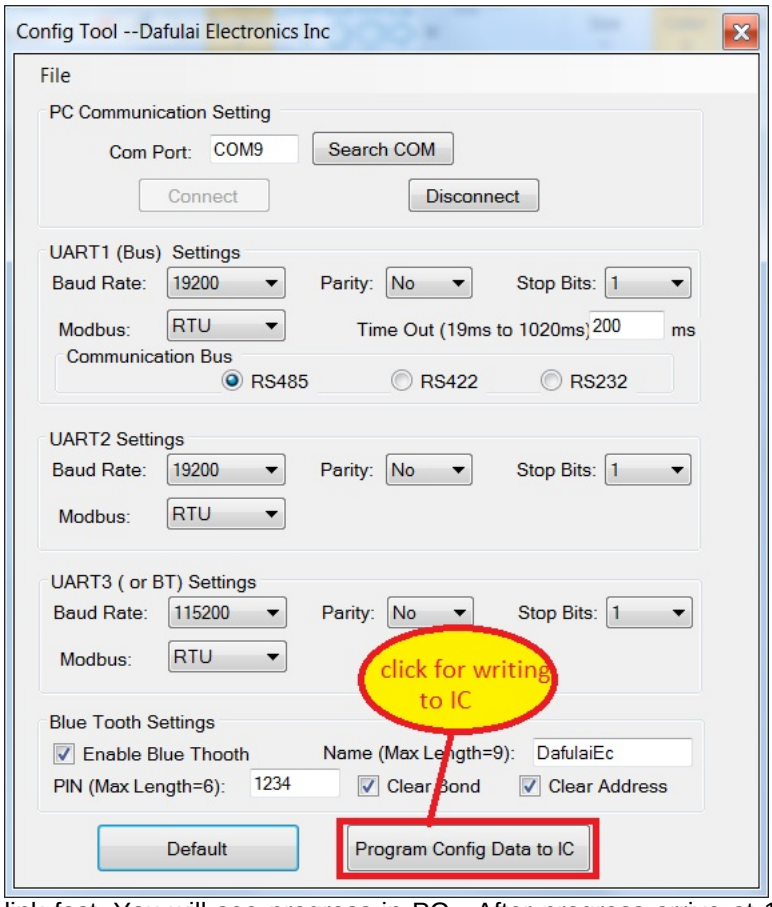
Notes: 1. Time Out is maximum time for UART1 waiting for slave device response. If you set too small, it won't work, please increase time out value. This is not "Time out" for your master node.

For actual master node, time out value is different from this one, it must be much biggest UART1 time out , and it depends on maximum response length because probably the other master is getting slave response when this master is sending Modbus request.

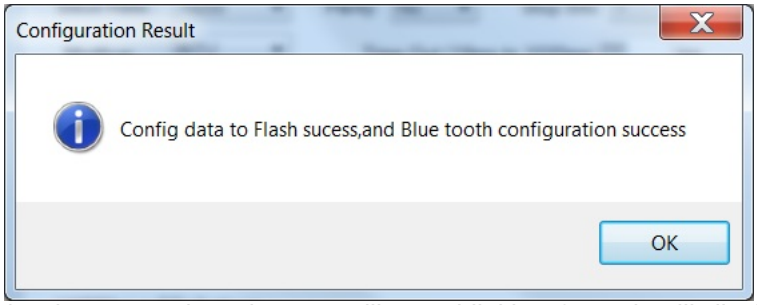
2. *Bluetooth only uses 115200 baud rate (This is not air baud rate, this is our UART3's baud rate). If you choose the other value for UART3 baud rate, Bluetooth will be disabled automatically.*
3. *Bluetooth name has maximum 9 characters length. First character must be letter. Actually you will get 2 Bluetooth names in your PC. One is with suffix 1, the other is with suffix 2. The one with suffix 1 is Bluetooth EDR, which is what we use. The one with suffix 2 is Bluetooth BLE, which is not what we use. So you should choose one with suffix 1 to connect.*

Step 4:

Click button "Program Config Data to IC". please see screenshot below:

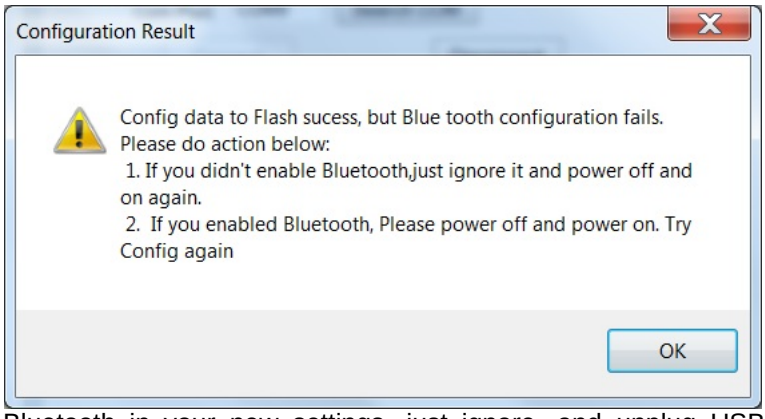


Mode LED will blink fast. You will see progress in PC. After progress arrive at 100%, you will see result below:



It means everything is OK. And Mode LED will stop blinking fast , it will display which interface UART1 uses (RS485, RS422, RS232). You can plug DB9, and DFLDMB1 will work for you.

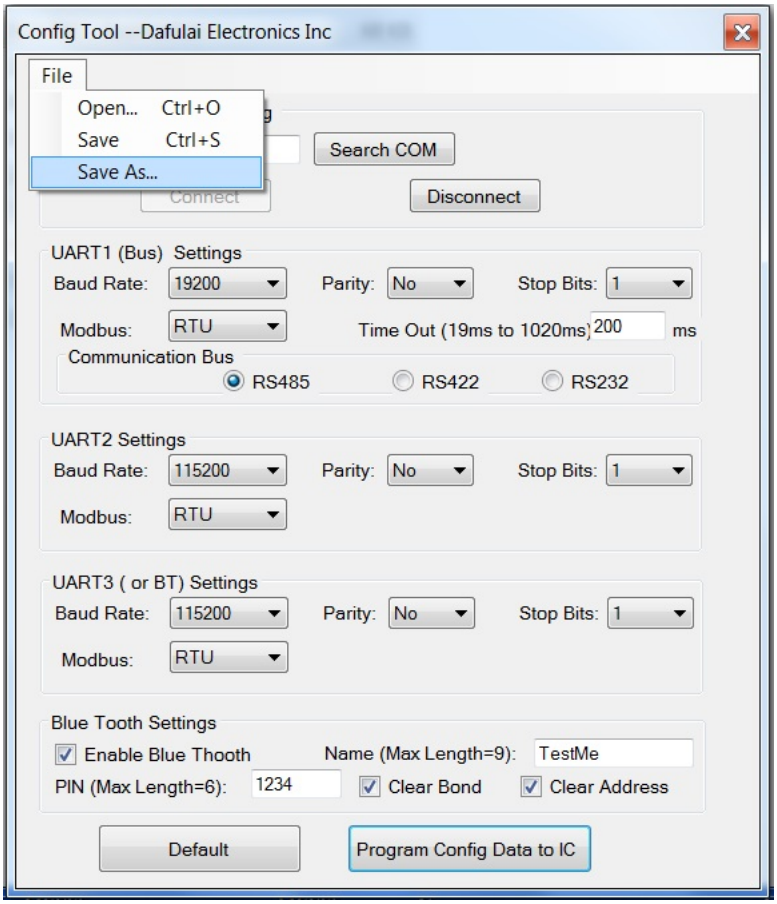
However, if you see result below:



If you disabled Bluetooth in your new settings, just ignore, and unplug USB cable (Power off DFLDMB1). Power on DFLDMB1 again, Mode LED will display which interface UART1 uses (RS485, RS422, RS232). DFLDMB1 will work for you.

If you enabled Bluetooth in your new settings, it means Configuration failed, you must click "Disconnect" button and unplug USB cable and go to Step 1 again.

You can save your configuration to a file, your team workers can use your configuration.
Please see screenshot for save :



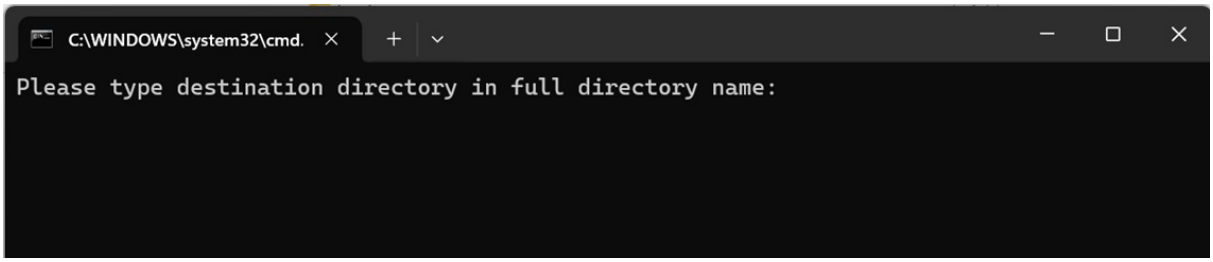
6 How to install Matlab/Simulink Modbus client library?

If you don't use both Matlab and Simulink, just skip this chapter.
This chapter is only for customers who use our free Matlab/Simulink Modbus Client software.

Please follow steps below for installing Matlab/Simulink Library:

In Windows platform

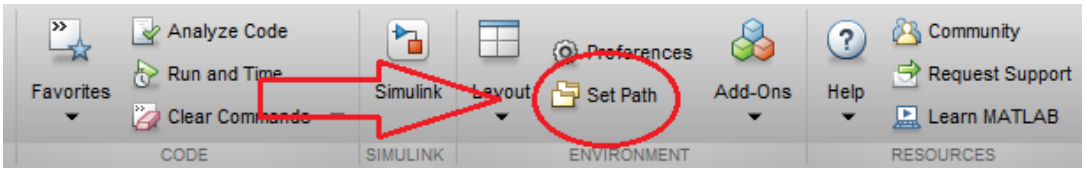
- **Step1** Download Modbus Client library for Matlab/Simulink from clicking ModbusClient4Mat.zip
- **Step2** unzip ModbusClient4Mat.zip to any directory.
- **Step3** double click on setup.bat. It will display window below:



Following the instructions above cmd window, input your directory name you want to install, please use full directory including disk drive name such as "C:\myDir" (without quotation marks).

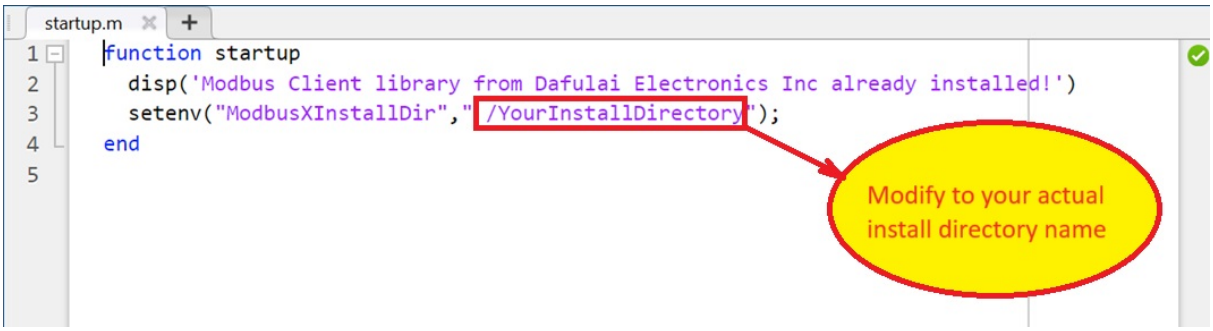
- **Step4** Set Matlab Path contains your destination directory of your Modbus Client library.

On Matlab's toolstrip, you may find the option "Set Path" which allows to select one directory and save it permanently to Matlab's "search path". See screenshot below:



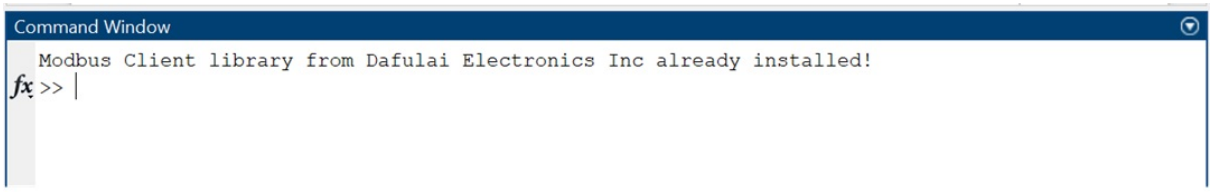
In non-Windows Platform.

- **Step1** Download Modbus Client library for Matlab/Simulink from clicking ModbusClient4Mat.zip
- **Step2** unzip ModbusClient4Mat.zip to directory you want to install.
- **Step3** Modify startup.m file. See screenshot below:

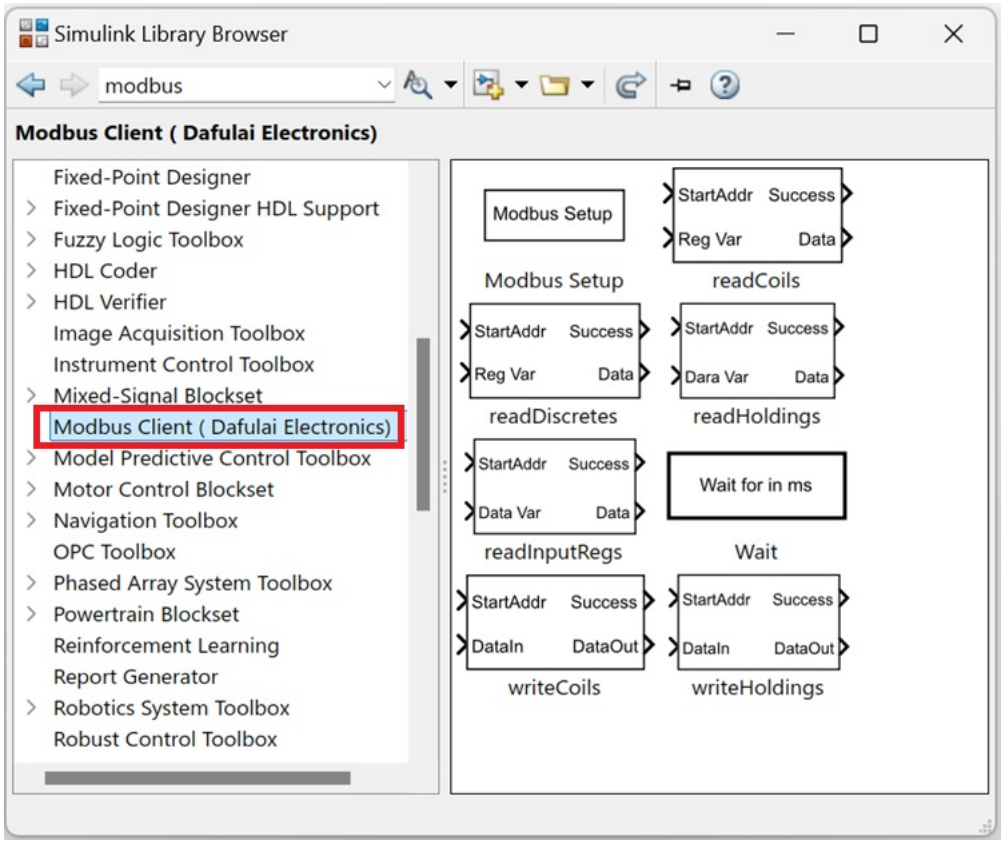


- **Step4** Set Matlab Path contains your directory you unzip. Now it is the same method as Step4 in windows platform.

After you finish Modbus Client library and Matlab Path setting correct, please re-start Matlab. You will see Message below in Matlab command window:



If you saw the above information, your Modbus Client library install in your computer successfully.
In your Simulink Library browser, you will see our Modbus Client library as shown as the following screenshot:



For the first-time use the simulator, you must run ConfigTool.exe. Of cause, if you want to change theses settings, you can run ConfigTool.exe software again.

7 Modbus Client for MATLAB

If you don't use Matlab, just skip this chapter.
This chapter is only for customers who use our free Matlab Modbus Client software.

In Matlab, we use ModbusX object to work on our hardware. Please read the following content for details.

ModbusX

Modbus RTU/ASCii/TCP Client (Master)
Since R2019b

Description

A ModbusX object represents a Modbus Master in the same Modbus network. After creating the object, use dot notation to call methods.
With the help of hardware "Modbus dual Masters Adaptor adaptor" from Dafulai Electronics, you can access Modbus RTU/ASCii Servers or Modbus TCP Servers.
Compared with Matlab built-in modbus object, our ModbusX supports user defined Function Code.
And every method has Error Code return, so you can know modbus exception code.

Almost all methods are compatible with Matlab built-in modbus object. Furthermore, we provide Simulink blocks library for ModbusX
We supports Modbus ASCii, please use ConfigTool.exe PC software to configure physical bus feature before using ModbusX.

Creation

Syntax

```
m = ModbusX(Transport,DeviceAddress)
m = ModbusX(Transport,DeviceAddress,Port)
m = ModbusX(Transport,DeviceAddress,Name,Value)
m = ModbusX(Transport,Port)
m = ModbusX(Transport,Port,Name,Value)
```

Description

example

m = ModbusX(Transport,DeviceAddress) constructs a Modbus object, m, over the transport type Transport using the specified DeviceAddress.When the transport is 'tcpip', DeviceAddress must be specified as the second argument. DeviceAddress is the IP address or host name of the Modbus server..

m = ModbusX(Transport,DeviceAddress,Port) additionally specifies Port. When the transport is 'tcpip', DeviceAddress must be specified. Port is the remote port used by the Modbus server. Port is optional, and it defaults to 502, which is the reserved port for Modbus.

m = ModbusX(Transport,DeviceAddress,Name,Value) specifies additional options with one or more name-value pair arguments using any of the previous syntaxes. For example, you can specify a timeout value. The Timeout property specifies the waiting time to complete read and write operations in seconds, and the default is 1.1.

m = ModbusX(Transport,Port) constructs a Modbus object m over the transport type Transport using the specified Port. When the transport is 'serialrtu', Port must be specified. This argument is the serial port that the Modbus dual Masters Adaptor hardware is connected to PC, such as 'COM3'.

m = ModbusX(Transport,Port,Name,Value) specifies additional options with one or more name-value pair arguments using any of the previous syntaxes. For example, you can specify NumRetries, the number of retries to perform if there is no reply from the server after a timeout.

Input Arguments

Transport -- Physical transport layer for device communication.
character vector | string scalar

Physical transport layer for device communication, specified as a character vector or string. Specify transport type as the first argument when you create the modbus object. You must set the transport type as either 'tcpip' or 'serialrtu' to designate the protocol you want to use.
For example,*m = ModbusX('tcpip','192.168.2.1')*

DeviceAddress — IP address or host name of Modbus server
character vector | string scalar

IP address or host name of Modbus server, specified as a character vector or string.
If transport is TCP/IP, it is required as the second argument during object creation.
Example: *m = ModbusX('tcpip','192.168.2.1')*

Port — Remote port used by Modbus server
502 (default) | double scalar

Remote port used by Modbus server, specified as a double. Optional as a third argument during object creation if transport is TCP/IP. The default of 502 is used if none is specified. Example: *m = ModbusX('tcpip','192.168.2.1',308)*

Port — Serial port Modbus server is connected to PC by this Port.
character vector | string scalar

Serial port "Modbus dual Masters Adaptor hardware USB or Bluetooth" is connected to by this argument, e.g. 'COM1' in windows OS, or "/dev/ttyUSB0" under Linux OS, specified as a character vector or string. If transport is Serial RTU, it is required as the second argument during object creation.
Example: *m = ModbusX('serialrtu','COM3')*

Name-Value Arguments
Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.
Please, use commas to separate each name and value, and enclose Name in quotes.
For example, *m = ModbusX('serialrtu','COM3','Timeout',20)*

You can use Name-Value pairs to set the following arguments:

- **WordOrder**

Value is scalar string or char array. It denote the words order when register data type is "uint32/int32/single/uint64/int64/double". Valid value is "big-endian" or "little-endian". Default="big-endian"

- **Timeout**

  Value is scalar double. Maximum time in seconds to wait for a response from the Modbus server.

  Maximum time in seconds to wait for a response from the Modbus server, specified as the comma-separated pair consisting of 'Timeout' and a positive value of type double. . The default is 1.1. You can change the value either during object creation or after you create the object.

  Example:

  m = ModbusX('serialrtu','COM3','Timeout',20)

- **NumRetries**

  Value is scalar double. Number of retries to perform if there is no reply or exception reply from the server after a timeout, specified as the comma-separated pair consisting of 'NumRetries' and a positive value of type double.You can change the value either during object creation. The default is 3

- **COM**

  Value is scalar string or char array. Modbus dual Masters Adaptor hardware is connected to PC by this Port. (This Name-Value is only used for Modbus TCP). When Modbus Server is Modbus TCP, we used this COM to identify whether "Modbus dual Masters Adaptor hardware" connects PC. Our Modbus object only works when "Modbus dual Masters Adaptor hardware" connects PC. Specified as the comma-separated pair consisting of 'COM' and a value of type string or character vector. You can only change the value during object creation. Default='COM1'.

- **BaudRate**

  Value is scalar double. Default is 19200 bits per second. Valid Value is 4800/9600/19200/28800/38400/43000/57600/115200. Note: This baud rate is not actual Modbus baud rate. It is baud rate for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus baud rate is decided by PC ConfigTool.exe software. Modbus TCP needs this Baud rate to identify Modbus dual Masters Adaptor Hardware.

- **Parity**

  Value is scalar string or char array. Valid choices are 'none' (default), 'even', 'odd'.

- **StopBits**

  Value is scalar double. Valid choices are 1 (default) and 2.

Examples

When the transport is TCP/IP, you must specify the IP address or host name of the Modbus server. And your PC must connect our "Modbus dual Masters Adaptor hardware". So if your hardware USB or

Bluetooth Serial port is not COM1, you must set up "COM" of name-value pair. . The following statement is usually used in general project:

m=ModbusX('tcpip', '192.168.2.1', 'COM', 'COM8', 'BaudRate', 115200, 'Parity', 'none');

When the transport is 'serialrtu', you must specify a Port argument. This is the serial port that the Modbus dual Masters Adaptor hardware is connected to PC. This baudRate is not actual Modbus baud rate. It is baud rate for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus baud rate is decided by PC ConfigTool.exe software. And RTU or ASCii is decided by PC ConfigTool. exe software too.
Create the Modbus RTU/ASCii master object m specifying a Port of 'COM3'..

m = ModbusX('serialrtu','COM3', 'BaudRate',19200,'NumRetries',5);

Properties

- DeviceAddress — IP address or host name of Modbus server
  scalar string or char array. IP address or host name of Modbus server, specified as a character vector or string. If transport is TCP/IP, it is required as the second argument during object creation.

- Port — Remote port used by Modbus server
  scalar double. Remote port used by Modbus server, specified as a double. Optional as a third argument during object creation if transport is TCP/IP. The default of 502 is used if none is specified.

- Port — Serial port hardware is connected to
  scalar string or char array. Serial port "Modbus dual Masters Adaptor hardware USB or Bluetooth" is connected to, e.g. 'COM1', specified as a character vector or string. If transport is Serial RTU, it is required as the second argument during object creation.

- Timeout — Time out for the Modbus server
  scalar double. The default is 1.1. Maximum time in seconds to wait for a response from the Modbus server. You can change the value either during object creation or after you create the object.

- NumRetries — Number of retries to perform if failure
  scalar double. The default is 3. Number of retries to perform if there is no reply or exception reply from the server after a timeout. You can change the value either during object creation or after you create the object.

- BaudRate — Serial Port Baud Rate
  scalar double. Default is 19200 bits per second. Valid Value is 4800/9600/19200/28800/38400/43000/57600/115200. Note: This baud rate is not actual Modbus baud rate. It is baud rate for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus baud rate is decided by PC ConfigTool.exe software. Modbus TCP needs this Baud rate to identify Modbus dual Masters Adaptor Hardware. You can change the property only during object creation.

- Parity — Serial Port Parity
  scalar string or char array. Valid choices are 'none' (default), 'even', 'odd'. Note: This Parity is not actual Modbus RTU/ASCii Parity. It is parity for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus Parity is decided by PC ConfigTool.exe software. Modbus TCP needs this parity to identify Modbus dual Masters Adaptor Hardware. You can change the property only during object creation.

- StopBits — Serial Port Stop bits
  scalar double. Valid choices are 1 (default) and 2. Note: This StopBits is not actual Modbus RTU/ASCii StopBits. It is StopBits for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus StopBits is decided by PC ConfigTool.exe software. Modbus TCP needs this StopBits to identify Modbus dual Masters Adaptor Hardware. You can change the property only during object creation.

- WordOrder — words order for multiple words' data type
  scalar string or char array. It denote the words order when register data type is "uint32/int32/single/uint64/int64/double". Valid value is "big-endian" or "little-endian". Default="big-endian". You can change it in the run-time by assignment

Object Functions (Or Methods)

read	read operation on the connected Modbus server
readUserDefinedInputRegs	read input regs operation by specified FC on the connected Modbus server
readUserDefinedInputs	read discrete regs operation by specified FC on the connected Modbus server
write	write operation to the connected Modbus server.
writeUserDefinedHoldingRegs	write holding regs operation by specified FC to the connected Modbus server.
writeUserDefinedCoils	write coil regs operation by specified FC to the connected Modbus server.
writeRead	write then read operation on the connected Modbus server in a single Modbus transaction
maskWrite	Modify the contents of a holding register using a AND mask OR mask parameters

Events

None

All methods details

read

Perform a read operation on the connected Modbus server.
Since R2019b

Syntax

[moddata,ErrCode] = read(obj,target,address)
[moddata,ErrCode] = read(obj,target,address,count)
[moddata,ErrCode] = read(obj,target,address,count,serverId)
[moddata,ErrCode] = read(obj,target,address,count,'precision')
[moddata,ErrCode] = read(obj,target,address,count,serverId,'precision')

Description

This function will perform a read operation from one of four target addressable areas: Coils, Inputs, Holding Registers, or Input Registers. Each of the four areas corresponds to a unique Modbus function code, which may or may not be supported by the connected Modbus server.

Input Arguments

- obj — ModbusX object. This argument is mandatory.
- target — Specifies the area to read. The valid choices are 'coils', 'inputs', 'inputregs' and 'holdingregs'. This argument is mandatory.
- address — The starting address to read from. This is 1-based address without 4x/3x... prefix. This argument is mandatory.
- count — The number of values to read. Optional, default is 1. Count is a scalar when doing reads of the same data type. Count is a vector of integers for reading multiple contiguous registers containing different data types. The number of values for count must match the number of values for precision.
- serverId — The address of the server to send this command to. Optional, default is 1. Valid values are 1-247.
- precision — Specifies the data format of the register being read from on the Modbus server. Valid values are 'uint16', 'int16', 'uint32', 'int32', 'uint64', 'int64', 'single', and 'double'. Optional, default is 'uint16'

  'single' and 'double' conversions conform to the IEEE 754 floating point standard. For signed integers a two's complement conversion is performed. Note that 'precision' does not refer to the return type (always 'double'), it only specifies how to interpret the register data.

  Precision is a string or char array when doing reads of the same data type. For reading multiple contiguous registers containing different data types, precision must be a cell array of strings or character vectors, or a string array of precisions. The number of precision values must match the number of count values.

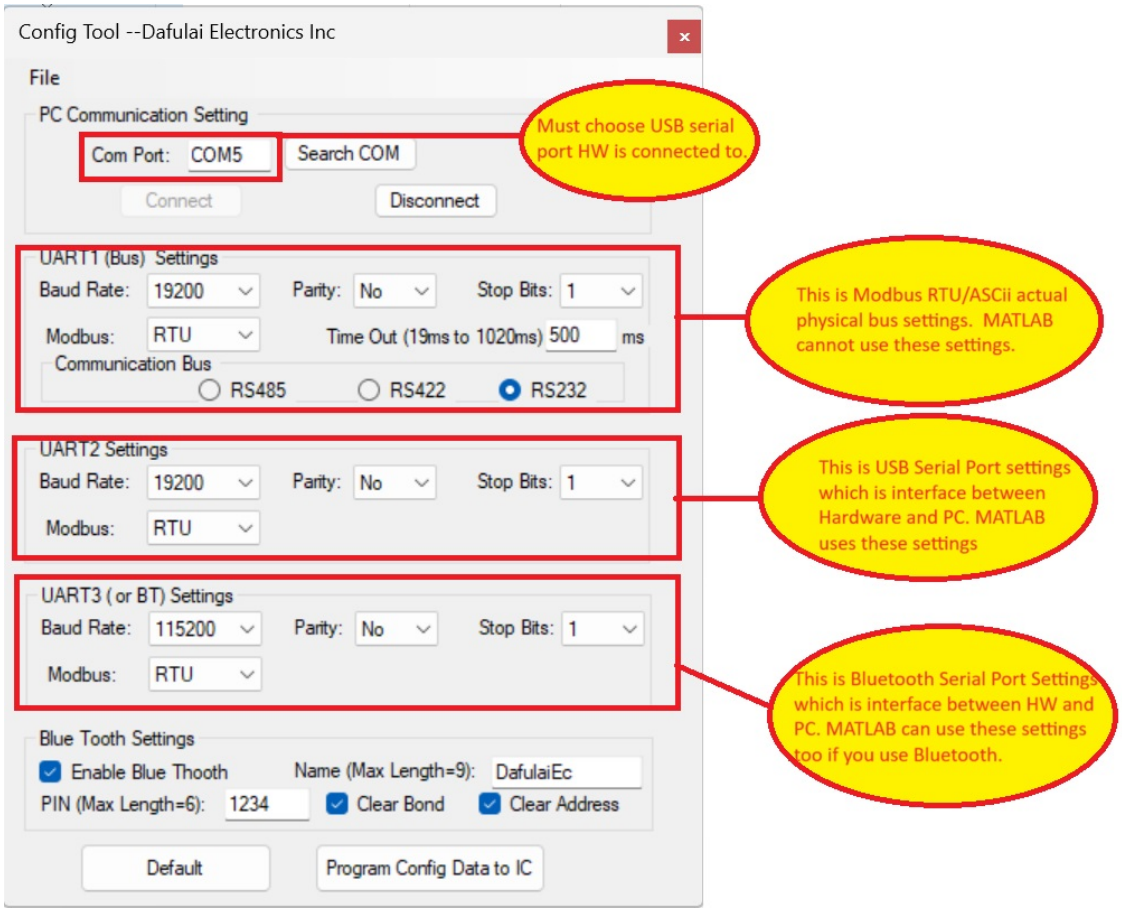
  Only for tagets: inputregs and holdingregs.

Output Arguments

- moddata — Register Values to read.
  Row vector for reading result in double number type. It will be [ ] if failed
- ErrCode — Error Code for reading.
  scalar double. The reason of failure in reading, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format error, 18 denotes CRC error.

Examples

If you are running our "Modbus dual Masters Adaptor hardware" for the first time, please run ConfigTool.exe in Windows OS. Screenshot is shown below:



We must use USB serial port to do configure (You cannot use Bluetooth serial port to do configure because Bluetooth needs to be configured too). In above figure, UART1 is one port actual Modbus RTU/ASCii Server is connected to. So UART1 settings are actual Modbus settings. UART2 is USB Serial port which is connected to our PC. So if you use USB serial port to access Modbus RTU/ASCii Server, please use these settings in MATLAB object construction function (ModbusX object). UART3 is Bluetooth Serial port which is connected to our PC. So if you use Bluetooth serial port to access Modbus RTU/ASCii Server, please use these settings in MATLAB object construction function (ModbusX object).

Firstly, we set up a ModbusX object (Modbus RTU/ASCii Master) with COM5. All other properties are used default values, it means that Baud Rate=19200, Parity='none', StopBits=1, Timeout=1.1, NumRetries=3, and WordOrder='big-endian'

```
m=ModbusX('serialrtu', 'COM5',);
```

```
% Read 3 coil values starting at address 5 from Slave node's Server ID =1

address = 5;
[moddata,Error] = read(m,'coils',address,3,1);
```

```
% Read 2 holding registers whose data format is unsigned 16 bits integer and 4 holding registers
% whose data format is double, in one read, at address 10 and Server ID=2.

address = 10;
precision = {'uint16', 'double'};
```

```
count = [2, 4];
[moddata, Error] = read(m, 'holdingregs', address, count,2, precision);
```

readUserDefinedInputRegs

Perform a input regs read operation by specified FC on the connected Modbus server.
Since R2019b

Syntax

```
[moddata,ErrCode] = readUserDefinedInputRegs(obj,FC,address)
[moddata,ErrCode] = readUserDefinedInputRegs(obj,FC,address,count)
[moddata,ErrCode] = readUserDefinedInputRegs(obj,FC,address,count,serverId)
[moddata,ErrCode] = readUserDefinedInputRegs(obj,FC,address,count,'precision')
[moddata,ErrCode] = readUserDefinedInputRegs(obj,FC,address,count,serverId,'precision')
```

Description

As we know, Standard read input registers function code is 4. However, Some server supports non-standard function code (not 4) for reading input registers. This function is designed for this purpose. This function will perform a input registers read operation by user defined Modbus function code, which may or may not be supported by the connected Modbus server.

Input Arguments

- obj — ModbusX object. This argument is mandatory.
- FC — Specifies the Function code, scalar double, 1 to 255. This argument is mandatory.
- address — The starting address to read from. This is 1-based address without 4x/3x... prefix. This argument is mandatory.
- count — The number of values to read. Optional, default is 1. Count is a scalar when doing reads of the same data type. Count is a vector of integers for reading multiple contiguous registers containing different data types. The number of values for count must match the number of values for precision.
- serverId — The address of the server to send this command to. Optional, default is 1. Valid values are 1-247.
- precision — Specifies the data format of the register being read from on the Modbus server. Valid values are 'uint16', 'int16', 'uint32', 'int32', 'uint64', 'int64', 'single', and 'double'. Optional, default is 'uint16'

  'single' and 'double' conversions conform to the IEEE 754 floating point standard. For signed integers a two's complement conversion is performed. Note that 'precision' does not refer to the return type (always 'double'), it only specifies how to interpret the register data.

  Precision is a string or char array when doing reads of the same data type. For reading multiple contiguous registers containing different data types, precision must be a cell array of strings or character vectors, or a string array of precisions. The number of precision values must match the number of count values.

Output Arguments

- moddata — Register Values to read.
  Row vector for reading result in double number type. It will be [ ] if failed

- ErrCode — Error Code for reading.
  scalar double. The reason of failure in reading, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format error, 18 denotes CRC error.

Examples

Firstly, we set up a ModbusX object (Modbus RTU/ASCii Master) with COM5. All other properties are used default values, it means that Baud Rate=19200, Parity='none', StopBits=1, Timeout=1.1, NumRetries=3, and WordOrder='big-endian'

```
m=ModbusX('serialrtu', 'COM5',);
```

```
% Read 2 input registers whose data format is unsigned 16 bits integer and 4 holding registers
% whose data format is double, in one read, at address 10 and Server ID=3 But FC =14 which is not standard 4.
```

```
address = 10;
precision = {'uint16', 'double'};
count = [2, 4];
[moddata, Error] = readUserDefinedInputRegs(m, 14, address, count,3, precision);
```

readUserDefinedInputs

Perform a discrete regs read operation by specified FC on the connected Modbus server.
Since R2019b

Syntax

```
[moddata,ErrCode] = readUserDefinedInputs(obj,FC,address)
[moddata,ErrCode] = readUserDefinedInputs(obj,FC,address,count)
[moddata,ErrCode] = readUserDefinedInputs(obj,FC,address,count,serverId)
```

Description

As we know, Standard read discrete registers function code is 2. However, Some server supports non-standard function code (not 2) for reading discrete registers. This function is designed for this purpose. This function will perform a discrete registers read operation by user defined Modbus function code, which may or may not be supported by the connected Modbus server.

Input Arguments

- obj — ModbusX object. This argument is mandatory.

- FC — Specifies the Function code, scalar double, 1 to 255. This argument is mandatory.

- address — The starting address to read from. This is 1-based address without prefix. This argument is mandatory.

- count — The number of values to read. Optional, default is 1. Count is a scalar,which denotes numbers of logical inputs.
- serverId — The address of the server to send this command to. Optional, default is 1. Valid values are 1-247.

Output Arguments

- moddata — Register Values to read.
  Row vector for reading result in logical number type. It will be [ ] if failed
- ErrCode — Error Code for reading.
  scalar double. The reason of failure in reading, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format error, 18 denotes CRC error.

Examples

Firstly, we set up a ModbusX object (Modbus TCP Master) with remote TCP address='192.168.1.32'. And Serial Port our hardware is connected to is 'COM5'. All other properties are used default values, it means that Port=502, Baud Rate=19200, Parity='none', StopBits=1, Timeout=1.1, NumRetries=3, and WordOrder='big-endian'

```
m=ModbusX('tcpip', '192.168.1.32, 'COM', 'COM5');
```

```
% Read 3 coil values starting at address 5 from Slave node's Server ID =1 (default) , but FC=17
```

```
address = 5;
[moddata,Error] = readUserDefinedInputs(m, 17, address,3);
```

write

Perform a write operation to the connected Modbus server.
Since R2019b

Syntax

```
ErrCode = write(obj,target,address,values)
ErrCode = write(obj,target,address,values,serverId)
ErrCode = write(obj,target,address,values,ForceSingleWrite)
ErrCode = write(obj,target,address,values,'precision')
ErrCode = write(obj,target,address,values,serverId,'precision')
ErrCode = write(obj,target,address,values,serverId,ForceSingleWrite)
ErrCode = write(obj,target,address,values,serverId,'precision',ForceSingleWrite)
```

Description

This function will perform a write operation to one of two writable target addressable areas: Coils or Holding Registers. Each of the two areas can accept a write request to a single address, or a

contiguous address range. Each possibility (single coil, multiple coils, single register, multiple registers) corresponds to a unique Modbus function code which may or may not be supported by the connected Modbus server.

Input Arguments

- obj — ModbusX object. This argument is mandatory.
- target — Specifies the area to write. The valid choices are 'coils' and 'holdingregs'. This argument is mandatory.
- address — The starting address to write to. This is 1-based address without 4x/0x... prefix. This argument is mandatory.
- values — Array of values to write. For target 'coils' valid values are logical false and true. For target 'holdingregs' valid values must be in the range of the specified 'precision'. This argument is mandatory.
- serverId — The address of the server to send this command to. Valid values are 0-247, with 0 being the broadcast address. Optional, default is 1.
- precision — Specifies the data format of the register being written to on the Modbus server device. Valid values are 'uint16', 'int16', 'uint32', 'int32', 'uint64','int64', 'single', and 'double'. Optional, default is ‘uint16’.

  The values passed in to be written will be converted to register values based on the specified precision. 'single' and 'double' conversions conform to the IEEE 754 floating point standard. For signed integers a 2's complement conversion is performed.
- ForceSingleWrite — Force single register write. Logical Scalar. true means that we will use FC=6 to replace FC=16 for target 'holdingregs', and FC=5 to replace FC=15 for target 'coils'. Otherwise, false means that we will use FC=16 to replace FC=6 for target 'holdingregs', and FC=15 to replace FC=5 for target 'coils'. Optional, default is false.

Output Arguments

- ErrCode — Error Code.
  scalar double. The reason of failure in writing, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format error, 18 denotes CRC error.

Examples

Firstly, we set up a ModbusX object (Modbus TCP Master) with remote TCP address='192.168.1.32'. And Serial Port our hardware is connected to is 'COM5'. All other properties are used default values, it means that Port=502, Baud Rate=19200, Parity='none', StopBits=1, Timeout=1.1, NumRetries=3, and WordOrder='big-endian'

```
m=ModbusX('tcpip', '192.168.1.32, 'COM', 'COM5');
```

% set the server ID=1, the holding registers at address 22 to the value 2000 in multiple writings (FC=16)

ErrorCode= write(m,'holdingregs',22,2000,1);

% set the server ID=1, the holding registers at address 23 to the value 2000 in single writing (FC=6)

ErrorCode= write(m,'holdingregs',23,2000,1, true);

writeUserDefinedHoldingRegs

Perform a holding registers write operation by specified FC to the connected Modbus server.
Since R2019b

Syntax

ErrCode = writeUserDefinedHoldingRegs(obj,FC,address,values)
ErrCode = writeUserDefinedHoldingRegs(obj,FC,address,values,serverId)
ErrCode = writeUserDefinedHoldingRegs(obj,FC,address,values,'precision')
ErrCode = writeUserDefinedHoldingRegs(obj,FC,address,values,serverId,'precision')

Description

As we know, Standard write holding registers function code is 16 or 6. However, Some server supports non-standard function code (not 16 , not 6) for writing holding registers. This function is designed for this purpose. This function will perform a holding registers write operation by user defined Modbus function code, which may or may not be supported by the connected Modbus server.

Input Arguments

- obj — ModbusX object. This argument is mandatory.
- FC — Specifies the Function code, scalar double, 1 to 255. This argument is mandatory.
- address — The starting address to write to. This is 1-based address without 4x prefix. This argument is mandatory.
- values — Array of values to write. The valid values must be in the range of the specified 'precision'. This argument is mandatory.
- serverId — The address of the server to send this command to. Valid values are 0-247, with 0 being the broadcast address. Optional, default is 1.
- precision — Specifies the data format of the register being written to on the Modbus server device. Valid values are 'uint16', 'int16', 'uint32', 'int32', 'uint64','int64', 'single', and 'double'. Optional, default is ‘uint16’.

  The values passed in to be written will be converted to register values based on the specified precision. 'single' and 'double' conversions conform to the IEEE 754 floating point standard. For signed integers a 2's complement conversion is performed.

Output Arguments

- ErrCode — Error Code.
  scalar double. The reason of failure in writing, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE

DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format error, 18 denotes CRC error.

Examples

Firstly, we set up a ModbusX object (Modbus TCP Master) with remote TCP address='192.168.1.32'. And Serial Port our hardware is connected to is 'COM5'. All other properties are used default values, it means that Port=502, Baud Rate=19200, Parity='none', StopBits=1, Timeout=1.1, NumRetries=3, and WordOrder='big-endian'

```
m=ModbusX('tcpip', '192.168.1.32, 'COM', 'COM5');

% set the server ID=2, the holding registers at address 22 to the value 2000 in FC=14
ErrorCode= writeUserDefinedHoldingRegs(m,14,22,2000,2);
```

writeUserDefinedCoils

Perform a coil registers write operation by specified FC to the connected Modbus server.
Since R2019b

Syntax

ErrCode = writeUserDefinedCoils(obj,FC,address,values)
ErrCode = writeUserDefinedCoils(obj,FC,address,values,serverId)

Description

As we know, Standard write coil registers function code is 15 or 5. However, Some server supports non-standard function code (not 15 , not 5) for writing coil registers. This function is designed for this purpose. This function will perform a coil registers write operation by user defined Modbus function code, which may or may not be supported by the connected Modbus server.

Input Arguments

- obj — ModbusX object. This argument is mandatory.
- FC — Specifies the Function code, scalar double, 1 to 255. This argument is mandatory.
- address — The starting address to write to. This is 1-based address without prefix. This argument is mandatory.
- values — Array of values to write. The valid values must be logical true or false. This argument is mandatory.
- serverId — The address of the server to send this command to. Valid values are 0-247, with 0 being the broadcast address. Optional, default is 1.

Output Arguments

- ErrCode — Error Code.
  scalar double. The reason of failure in writing, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format

error, 18 denotes CRC error.

Examples

Firstly, we set up a ModbusX object (Modbus RTU/ASCii Master) with COM5. All other properties are used default values, it means that Baud Rate=19200, Parity='none', StopBits=1, Timeout=1.1, NumRetries=3, and WordOrder='big-endian'

```
m=ModbusX('serialrtu', 'COM5',);

% set the server ID=8, the coil registers at address 22 to the value true, address 23 to false in FC=13
ErrorCode= writeUserDefinedCoils(m,13,22,[true false ],8);
```

writeRead

Perform a write then read operation on the connected Modbus server in a single Modbus transaction. Since R2019b

Syntax

```
[moddata,ErrCode] = writeRead(obj,writeAddress,writeData,readAddress,readCount)
[moddata,ErrCode] = writeRead(obj,writeAddress,writeData,readAddress,...
                              readCount,serverId)
[moddata,ErrCode] = writeRead(obj,writeAddress,writeData,'writePrecision',...
                              readAddress,readCount,'readPrecision')
[moddata,ErrCode] = writeRead(obj,writeAddress,writeData,'writePrecision',...
                              readAddress,readCount,'readPrecision',serverId)
```

Description

This function is used to perform a combination of one write operation and one read operation on groups of holding registers in a single Modbus transaction. The write operation is always performed before the read. The range of addresses to read must be contiguous, and the range of addresses to write must be contiguous, but each are specified independently and may or may not overlap.

Input Arguments

- obj — ModbusX object. This argument is mandatory.
- writeAddress —The starting address of the registers to write (1-based without prefix). This argument is mandatory.
- writeData — Array of values to write where the first value in the array is written to writeAddress. This argument is mandatory.
- writePrecision — Specifies the data format of the register being written to on the Modbus server. Valid values are 'uint16', 'int16', 'uint32', 'int32', 'uint64', 'int64', 'single', and 'double'. Optional, default is 'uint16'
- readAddress —The starting address of the registers to read (1-based without prefix). This argument is mandatory.

- readCount — The number of registers to read. Optional, default is 1.
- serverId — The address of the server to send this command to. Optional, default is 1. Valid values are 1-247.
- readPrecision — Specifies the data format of the register being read from on the Modbus server. Valid values are 'uint16', 'int16', 'uint32', 'int32', 'uint64', 'int64', 'single', and 'double'. Optional, default is 'uint16'

  'single' and 'double' conversions conform to the IEEE 754 floating point standard. For signed integers a two's complement conversion is performed. Note that 'precision' does not refer to the return type (always 'double'), it only specifies how to interpret the register data.

Output Arguments

- moddata — Register Values to read.
  Row vector for reading result in double number type. It will be [ ] if failed
- ErrCode — Error Code for writing/reading.
  scalar double. The reason of failure in writing/reading, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format error, 18 denotes CRC error.

maskWrite

Modify the contents of a holding register using a AND mask OR mask parameters
Since R2019b

Syntax

ErrCode = maskWrite(obj, address, andMask, orMask)
ErrCode = maskWrite(obj, address, andMask, orMask, serverId)

Description

This function is used to set or clear individual bits in a specific holding register; a read/modify/write operation. This is done by using a combination of an AND mask, an OR mask, and the register's current contents.

The function’s algorithm is:
Result = (register value AND andMask) OR (orMask AND (NOT andMask))

Input Arguments

- obj — ModbusX object. This argument is mandatory.
- address —The starting address of the registers to perform mask write on (1-based without prefix). This argument is mandatory.
- andMask — AND value to use in mask write operation described above. Valid range is 0-65535..

This argument is mandatory.

- orMask — OR value to use in mask write operation described above. Valid range is 0-65535.. This argument is mandatory.
- serverId — The address of the server to send this command to. Optional, default is 1. Valid values are 0-247. with 0 being the broadcast address. Optional, default is 1.

Output Arguments

- ErrCode — Error Code for failure in mask-writing
  scalar double. The reason of failure in mask-writing, 0 denotes success, 1 denotes ILLEGAL FUNCTION, 2 denotes ILLEGAL DATA ADDRESS, 3 denotes ILLEGAL DATA VALUE , 4 denotes SLAVE DEVICE FAILURE, 6 denotes SLAVE DEVICE BUSY, 16 denotes timeout, 17 denotes frame format error, 18 denotes CRC error.

8 Modbus Client for Simulink

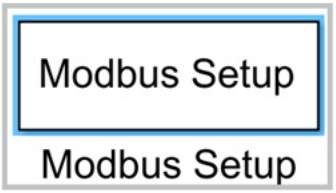
If you don't use Simulink, just skip this chapter.
This chapter is only for customers who use our free Simulink Modbus Client software.

Now we explain all Simulink blocks.

Modbus_Setup

Set up A Modbus Client (Hardware: Modbus dual Masters Adaptor from Dafulai Electronic Inc)
Since R2019b

Library: Modbus Client (Dafulai Electronics) /Modbus_Setup



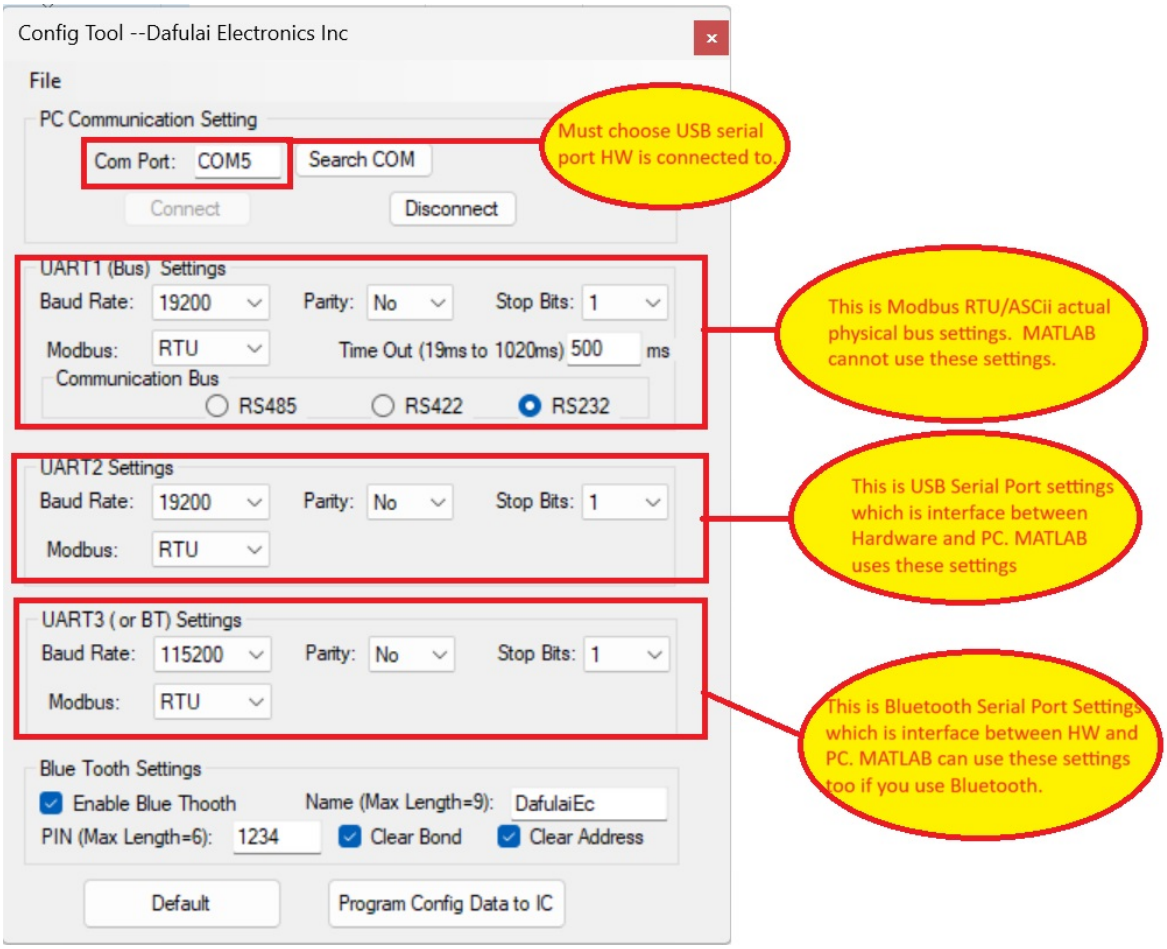
Description

This block sets up all parameters of Modbus Client (Master) which is supported by Hardware "Modbus dual Masters adaptor" from Dafulai Electronic Inc
You must put this block in your simulation model in order to access Modbus Servers by this Modbus Master.

Modbus TCP Client uses PC TCP/IP stacks. However, Our software must detect Hardware "Modbus dual Masters adaptor" is connected to PC (by USB or by Bluetooth).
So it still needs Serial Port number / baud rate / parity / Stop bits.

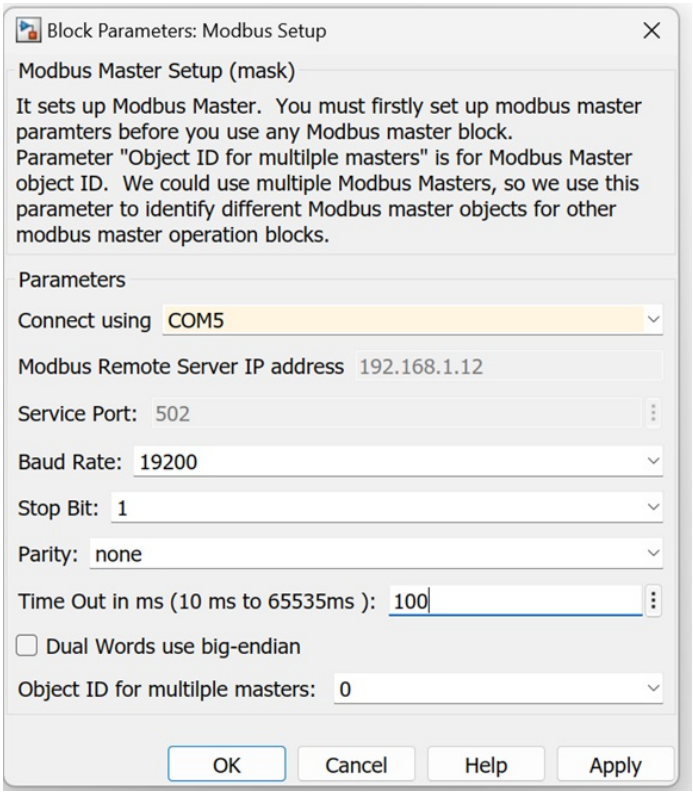
This block will set up Modbus Master parameters. The baud rate / parity / Stop bits in this block is for Hardware "Modbus dual Masters adaptor" which is connected to PC by USB or Bluetooth serial Port. It is NOT for Modbus RTU/ASCii client, Modbus RTU/ASCii client baud rate / parity / Stop bits and physical interface (RS232/RS485/RS422) are decided by ConfigTool.exe Software. If you use Hardware "Modbus dual Masters adaptor" for the first time, you must run ConfigTool.exe Software under Windows OS to configure hardware.

Please see Screenshot below for running ConfigTool.exe



Parameters

There are many parameters for this block. Please double click this block to open parameters dialog below:



Many parameters are self-explanation from the label. We only explain some special parameters.

- Connect to — This is USB or Bluetooth Serial port for our Modbus dual masters adaptor hardware. You must connect USB or Bluetooth Serial port for our Modbus dual masters adaptor hardware even though you only use Modbus TCP server.
- Modbus Remote Server IP address — This is only useful for Modbus TCP server. It is remote Modbus server IP address or hostname.
- Service Port — This is only useful for Modbus TCP server. It is remote Modbus server TCP port number. Default is 502.
- Baud Rate — Baud rate for Modbus dual masters adaptor hardware connecting PC. Valid Baud rates are 4800/9600/19200/28800/38400/43000/57600/115200. Please use drop list to select one. It is NOT Modbus RTU/ASCii client Baud Rate. It is Baud Rate for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus Baud Rate is decided by PC ConfigTool.exe software. Modbus TCP needs this baud rate to identify Modbus dual Masters Adaptor Hardware.
- Stop Bit —Valid choices are 1 (default) and 2. Note: This StopBits is not actual Modbus RTU/ASCii StopBits. It is StopBits for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus StopBits is decided by PC ConfigTool.exe software. Modbus TCP needs this StopBits to identify Modbus dual Masters Adaptor Hardware.
- Parity — Valid choices are 'none' (default), 'even', 'odd'. Note: This Parity is not actual Modbus RTU/ASCii Parity. It is parity for Modbus dual Masters Adaptor Hardware interface with PC. Actual Modbus Parity is decided by PC ConfigTool.exe software. Modbus TCP needs this parity to identify Modbus dual Masters Adaptor Hardware.

- Object ID for multiple Modbus master — In one PC, we may use multiple "Modbus dual masters adaptor" hardware, this is for identifying each one.
- Hardware USB or BT Serial Port — It is only displayed in Modbus TCP. It is Serial port for Modbus dual Masters Adaptor Hardware interface with PC. Modbus TCP needs this Serial Port to identify Modbus dual Masters Adaptor Hardware.

Ports

Input

None

Outpot

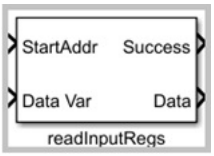
None

---

readInputRegs

read Modbus Server input registers values (Hardware: Modbus dual Masters Adaptor from Dafulai Electronic Inc)
Since R2019b

Library: Modbus Client (Dafulai Electronics) /readInputRegs



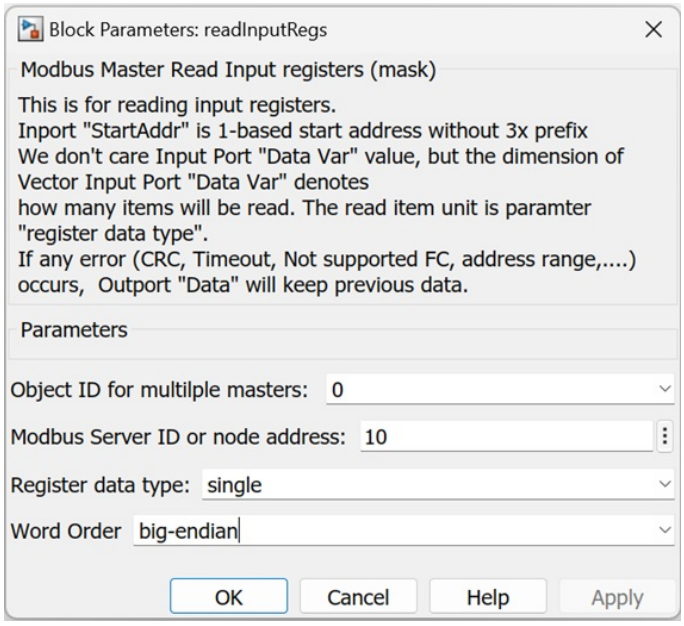
Description

This block reads input registers values. Start Address is from Input port "StartAddr" (1-based address without 3x prefix), Read Quantities are from the dimension of Input port "Data Var" . However this Quantities is in unit of Destination Data type. For example, if Destination Data type (parameter: Register data type) is "uint32", and Input port "Data Var" is 5 elements's vector. Actually the words Quantities will be 5 x 2 =10. (From "StartAddr" to "StartAddr"+9). Why do we use the dimension of Input port "Data Var" instead of "Input Data QTY" scalar? The reason is for "Embedded Code generator", in this way, embedded code will know variable 's input register address easily.

If any error (CRC, Timeout, Not supported FC, address range,....) occurs, Outport "Data" will keep previous value, and Outport "Success" will be false.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple masters — In one PC, we may use multiple "Modbus dual masters adaptor" hardware, this is for identifying each one.
- Modbus Server ID or node address — The address of the server to send this "read input registers" command to.
- Register data type — Specifies the data format of the register being read from on the Modbus server. It is not Data type of output port "Data". The data type of output port "Data" is always "double".
- Word Order — It denote the words order when register data type is "uint32/int32/single/ uint64/int64/double".

Ports

Input

- StartAddr — "double" data type's scalar. It is input Regs start address (1 based without 3X prefix) you want to read.
- Data Var — "double" data type's vector. The dimension of vector is input Regs QTY you want to read in specified data type in parameter "Register data type".

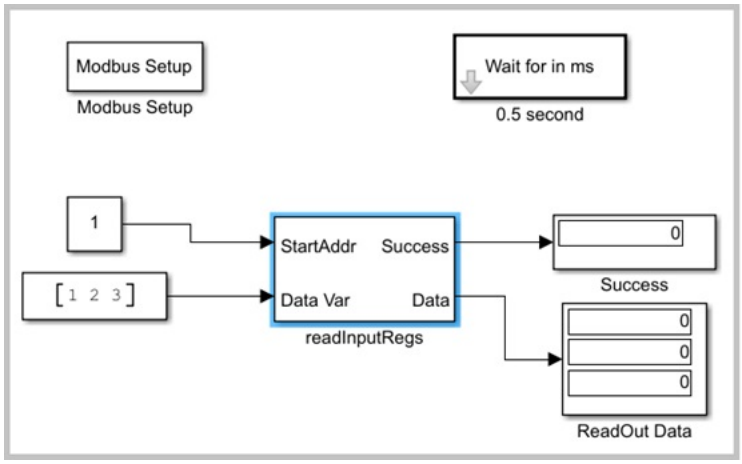
Outport

- Success — "logical" data type's scalar. true means read out successfully. false means that failure in reading out data.
- Data — "double" data type's vector. It is all Data you read out. If we didn't read out any data, the output port Data will keep previous values. Initial value is zero vector.

Examples

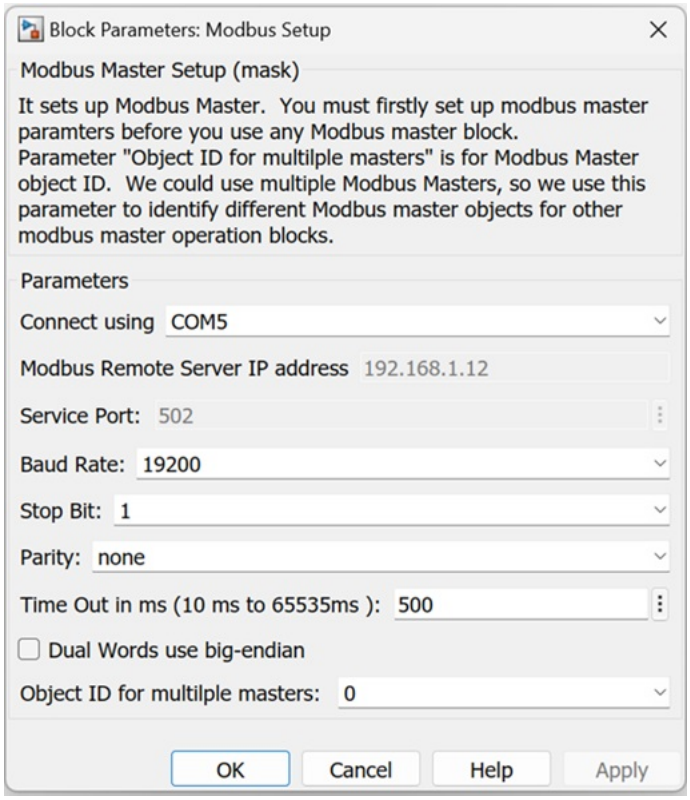
Example:

Every 500ms (Wait 0.5 sec block), We are reading input registers address from 1 to 3 of Modbus RTU/ASCii Server with Server ID=10.

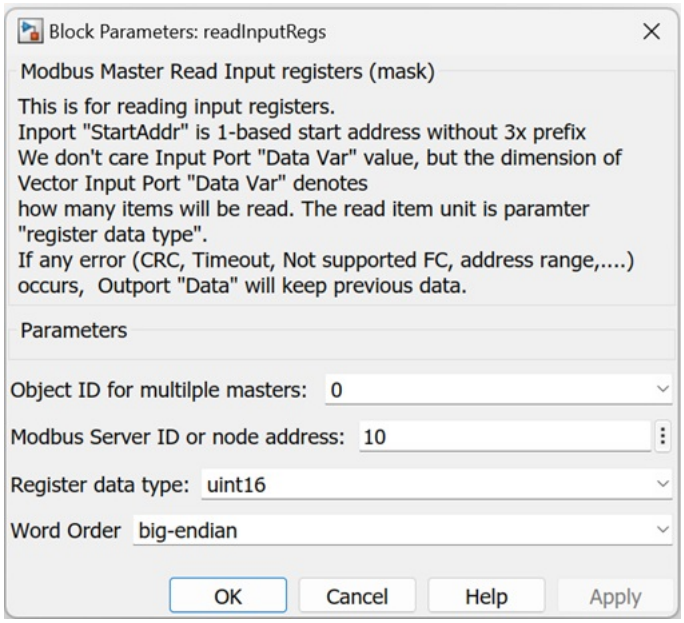


Please open "your Modbus Client library folder"/examples/example1_readinputregs.slx (You must change USB serial Port number in Modbus Setup block according to your physical USB port number).

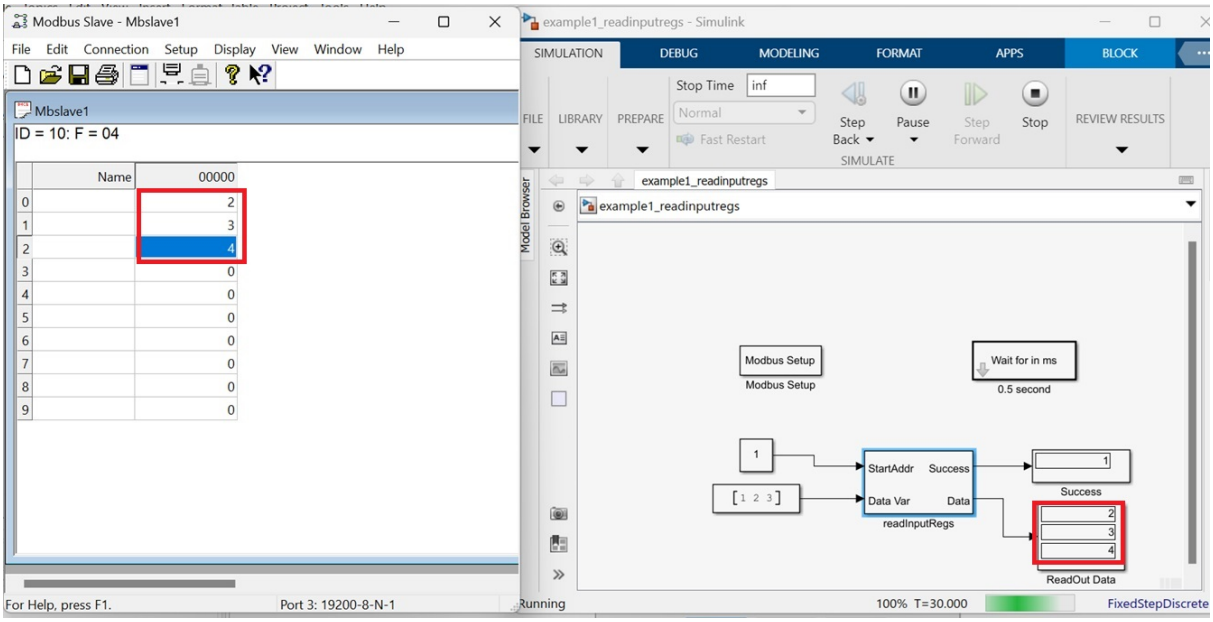
For "Modbus Setup" block, the parameters are set up below:



For "readInputRegs" block, the parameters are set up below:



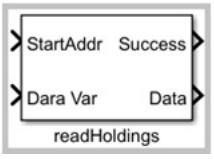
Our example can access both Modbus RTU and Modbus ASCii. RTU or ASCii is decided by ConfigTool.exe software.
You can run general Modbus Slave Simulator software such as "Modbus Salve" to change Input registers values addressing 30001 to 30003, you will see its result in our Simulink example.



readHoldingRegs

read Modbus Server holding registers values (Hardware: Modbus dual Masters Adaptor from Dafulai Electronic Inc)
Since R2019b

Library: Modbus Client (Dafulai Electronics) /readHoldingRegs



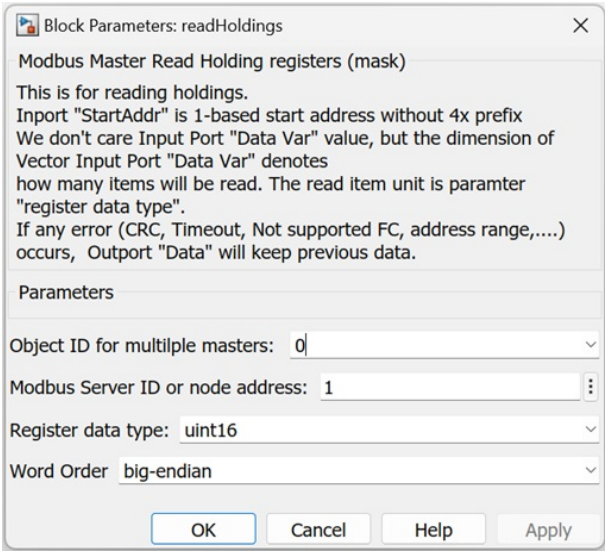
Description

This block reads holding registers values. Start Address is from Input port "StartAddr" (1-based address without 4x prefix), Read Quantities are from the dimension of Input port "Data Var" . However this Quantities is in unit of Destination Data type. For example, if Destination Data type (parameter: Register data type) is "uint32", and Input port "Data Var" is 5 elements's vector. Actually the words Quantities will be 5 x 2 =10. (From "StartAddr" to "StartAddr"+9). Why do we use the dimension of Input port "Data Var" instead of "holding Data QTY" scalar? The reason is for "Embedded Code generator", in this way, embedded code will know variable 's holding register address easily.

If any error (CRC, Timeout, Not supported FC, address range,....) occurs, Outport "Data" will keep previous value, and Outport "Success" will be false.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple masters — In one PC, we may use multiple "Modbus dual masters adaptor" hardware, this is for identifying each one.
- Modbus Server ID or node address — The address of the server to send this "read holding registers" command to.
- Register data type — Specifies the data format of the register being read from on the Modbus server. It is not Data type of output port "Data". The data type of output port "Data" is always "double".
- Word Order — It denote the words order when register data type is "uint32/int32/single/ uint64/int64/double".

Ports

Input

- StartAddr — "double" data type's scalar. It is holding Regs start address (1 based without 3X prefix) you want to read.
- Data Var — "double" data type's vector. The dimension of vector is holding Regs QTY you want to read in specified data type in parameter "Register data type".

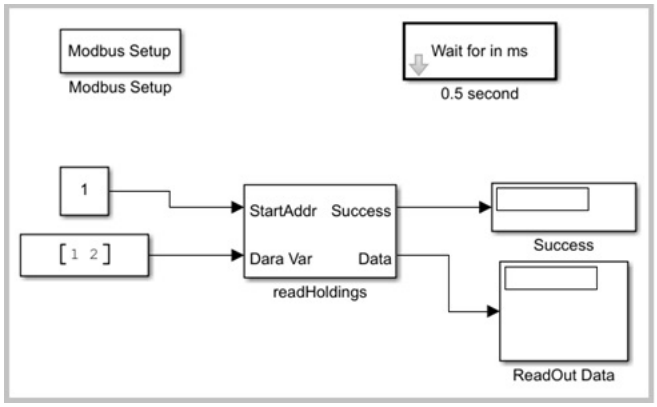
Outport

- Success — "logical" data type's scalar. true means read out successfully. false means that failure in reading out data.
- Data — "double" data type's vector. It is all Data you read out. If we didn't read out any data, the output port Data will keep previous values. Initial value is zero vector.

Examples

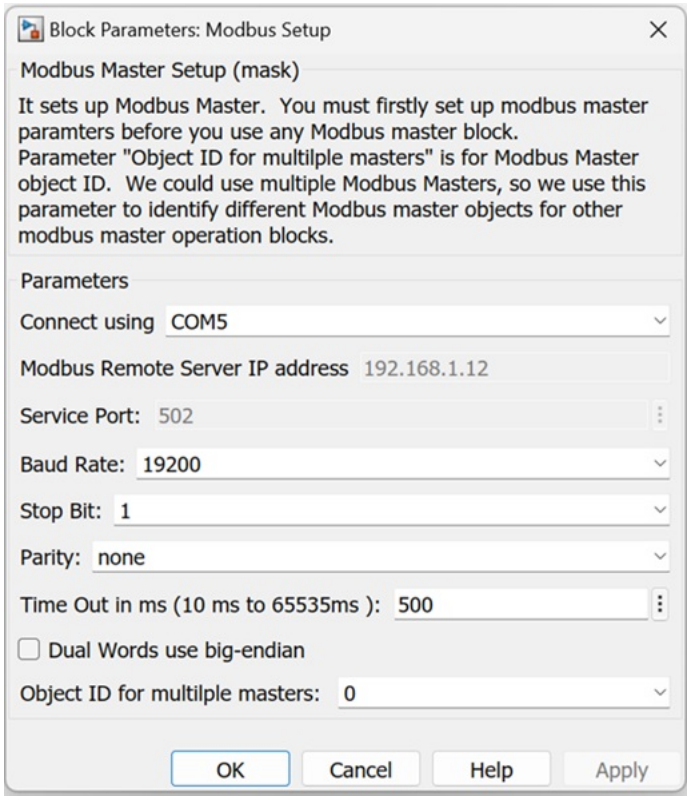
Example:

Every 500ms (Wait 0.5 sec block), We are reading holding registers address from 1 to 3 of Modbus RTU/ASCii Server with Server ID=2.

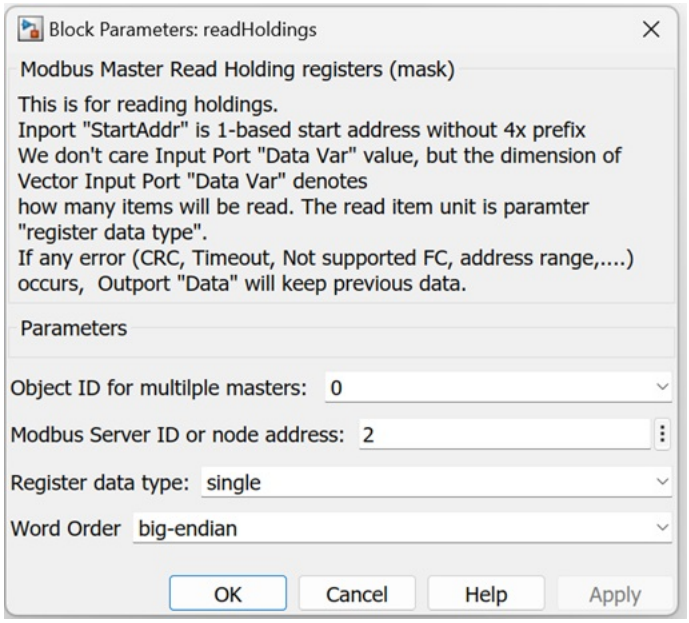


Please open "your Modbus Client library folder"/examples/example2_readHoldingregs.slx (You must change USB serial Port number in Modbus Setup block according to your physical USB port number).

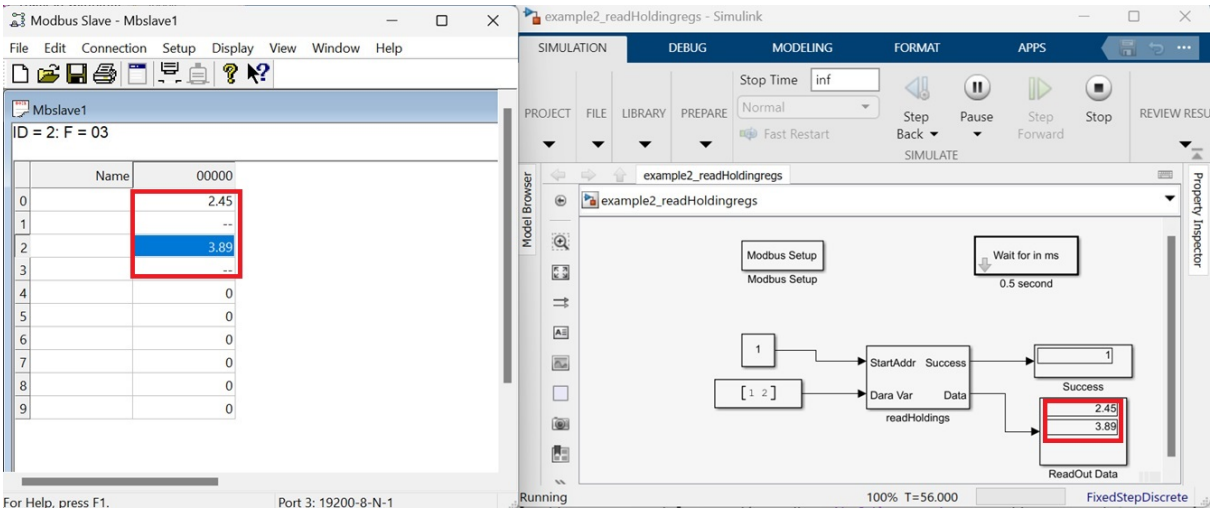
For "Modbus Setup" block, the parameters are set up below:



For "readHoldingRegs" block, the parameters are set up below:



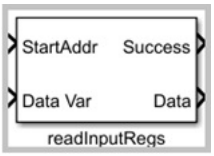
Our example can access both Modbus RTU and Modbus ASCii. RTU or ASCii is decided by ConfigTool.exe software.
You can run general Modbus Slave Simulator software such as "Modbus Salve" to change holding registers values addressing 40001 to 40004, you will see its result in our Simulink example.



readDiscretes

read Modbus Server discrete input registers values (Hardware: Modbus dual Masters Adaptor from Dafulai Electronic Inc)
Since R2019b

Library: Modbus Client (Dafulai Electronics) /readDiscretes



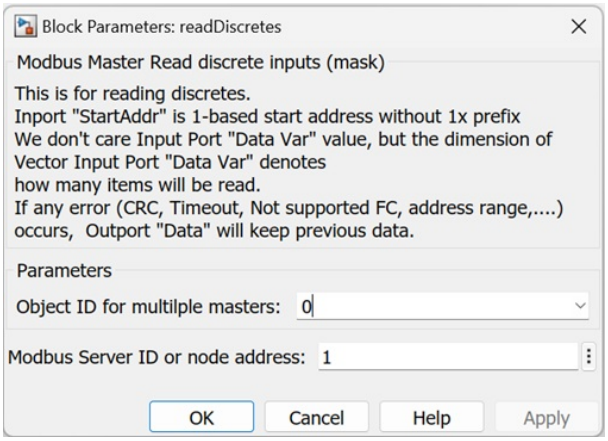
Description

This block reads discrete input registers values. Start Address is from Input port "StartAddr" (1-based address without 1x prefix), Read Quantities are equal to the dimension of Input port "Data Var" . Why do we use the dimension of Input port "Data Var" instead of "discrete input Data QTY" scalar? The reason is for "Embedded Code generator", in this way, embedded code will know variable 's discrete input register address easily.

If any error (CRC, Timeout, Not supported FC, address range,....) occurs, Outport "Data" will keep previous value, and Outport "Success" will be false.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple masters — In one PC, we may use multiple "Modbus dual masters adaptor" hardware, this is for identifying each one.
- Modbus Server ID or node address — The address of the server to send this "read discrete input registers" command to.

Ports

Input

- StartAddr — "double" data type's scalar. It is discrete input Regs start address (1 based without 1X prefix) you want to read.
- Data Var — "logical" data type's vector. The dimension of vector is discrete input Regs QTY you want to read.

Outport

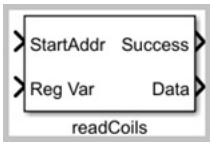
- Success — "logical" data type's scalar. true means read out successfully. false means that failure in reading out data.
- Data — "logical" data type's vector. It is all Data you read out. If we didn't read out any data, the output port Data will keep previous values. Initial value is false vector.

readCoils

read Modbus Server coil registers values (Hardware: Modbus dual Masters Adaptor from Dafulai Electronic Inc)

Since R2019b

Library: Modbus Client (Dafulai Electronics) /readCoils



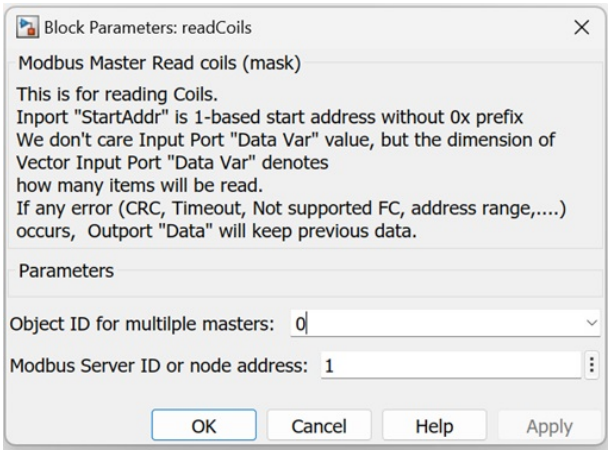
Description

This block reads coil registers values. Start Address is from Input port "StartAddr" (1-based address without 0x prefix), Read Quantities are equal to the dimension of Input port "Data Var" . Why do we use the dimension of Input port "Data Var" instead of "coil Data QTY" scalar? The reason is for "Embedded Code generator", in this way, embedded code will know variable 's coil register address easily.

If any error (CRC, Timeout, Not supported FC, address range,....) occurs, Outport "Data" will keep previous value, and Outport "Success" will be false.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple masters — In one PC, we may use multiple "Modbus dual masters adaptor" hardware, this is for identifying each one.
- Modbus Server ID or node address — The address of the server to send this "read coil registers" command to.

Ports

Input

- StartAddr — "double" data type's scalar. It is coil Regs start address (1 based without 0X prefix) you want to read.
- Data Var — "logical" data type's vector. The dimension of vector is coil Regs QTY you want to read.

Outport

- Success — "logical" data type's scalar. true means read out successfully. false means that failure in reading out data.
- Data — "logical" data type's vector. It is all Data you read out. If we didn't read out any data, the output port Data will keep previous values. Initial value is false vector.

writeHoldingRegs

write Modbus Server holding registers (Hardware: Modbus dual Masters Adaptor from Dafulai Electronic Inc)
Since R2019b

Library: Modbus Client (Dafulai Electronics) /writeHoldingRegs

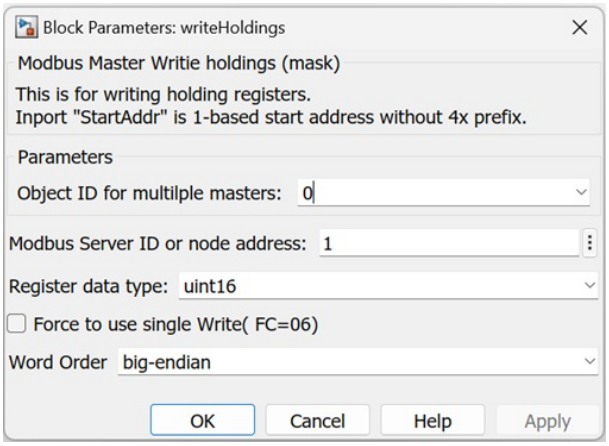
Description

This block writes holding registers. Start Address is from Input port "StartAddr" (1-based address without 4x prefix), Write Quantities are from the dimension of Input port "DataIn" . However this Quantities is in unit of Destination Data type. For example, if Destination Data type (parameter: Register data type) is "uint32", and Input port "DataIn" is 5 elements's vector. Actually the words Quantities will be 5 x 2 =10. (From "StartAddr" to "StartAddr"+9).

If any error (CRC, Timeout, Not supported FC, address range,....) occurs, Outport "DataOut" will keep previous value, and Outport "Success" will be false.
Otherwise, write successfully, Outport "DataOut" will be input port "DataIn", and Outport "Success" will be true.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple masters — In one PC, we may use multiple "Modbus dual masters adaptor" hardware, this is for identifying each one.
- Modbus Server ID or node address — The address of the server to send this "write holding registers" command to.
- Register data type — Specifies the data format of the register being written to on the Modbus server. It is not Data type of input port "DataIn". The data type of input port "DataIn" is always "double", the data type of output port "DataOut" is always "double" too.
- Force to use single Write (FC=06) — true means that we we will use multiple single writings (FC=06) to replace one multiple writing (FC=16).
- Word Order — It denote the words order when register data type is "uint32/int32/single/ uint64/int64/double".

Ports

Input

- StartAddr — "double" data type's scalar. It is holding Regs start address (1 based without 4X prefix) you want to write.
- DataIn — "double" data type's vector. It is data values you want to write, but it will be changed to data type in parameter "Register data type" from double when writing to registers.

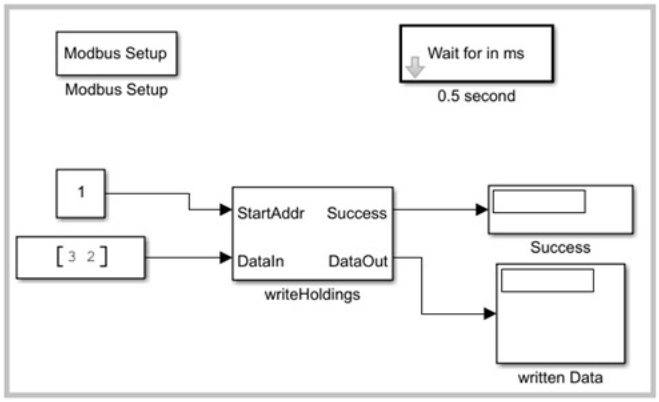
Outport

- Success — "logical" data type's scalar. true means write successfully. false means that failure in writing.
- DataOut — "double" data type's vector. It is input port "DataIn" values when write successfully. The output port "DataOut" will keep previous values if fail in writing. Initial value is zero vector.

Examples

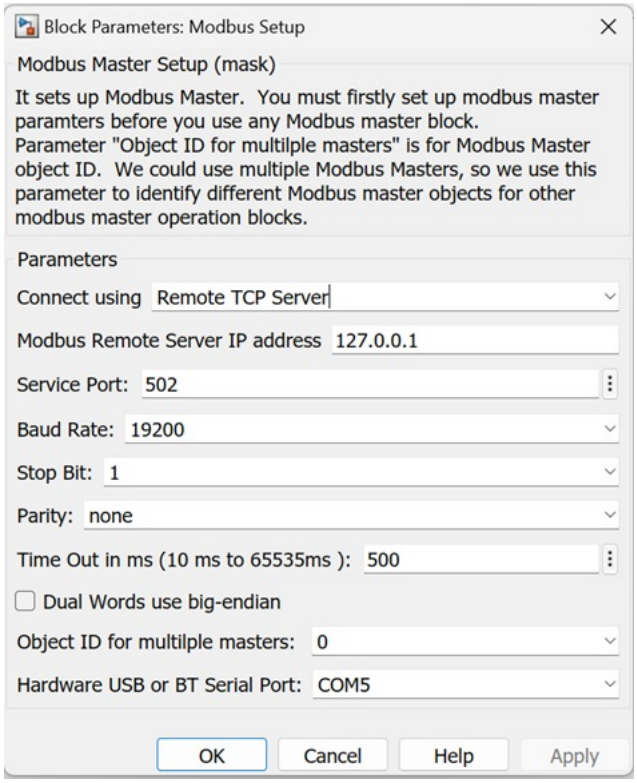
Example:

Every 500ms (Wait 0.5 sec block), We are writing holding registers address from 1 to 2 of Modbus TCP Server (ip address: 127.0.0.1, port=502) with Server ID=1.

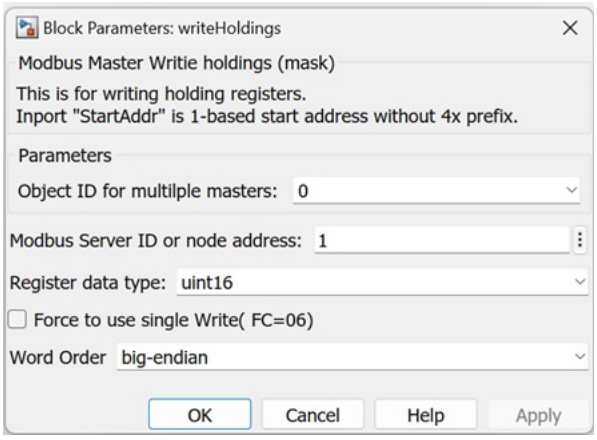


Please open "your Modbus Client library folder"/examples/example3_writeHoldingregs.slx (You must change "Hardware USB or BT Serial Port" in Modbus Setup block according to your physical USB port number).

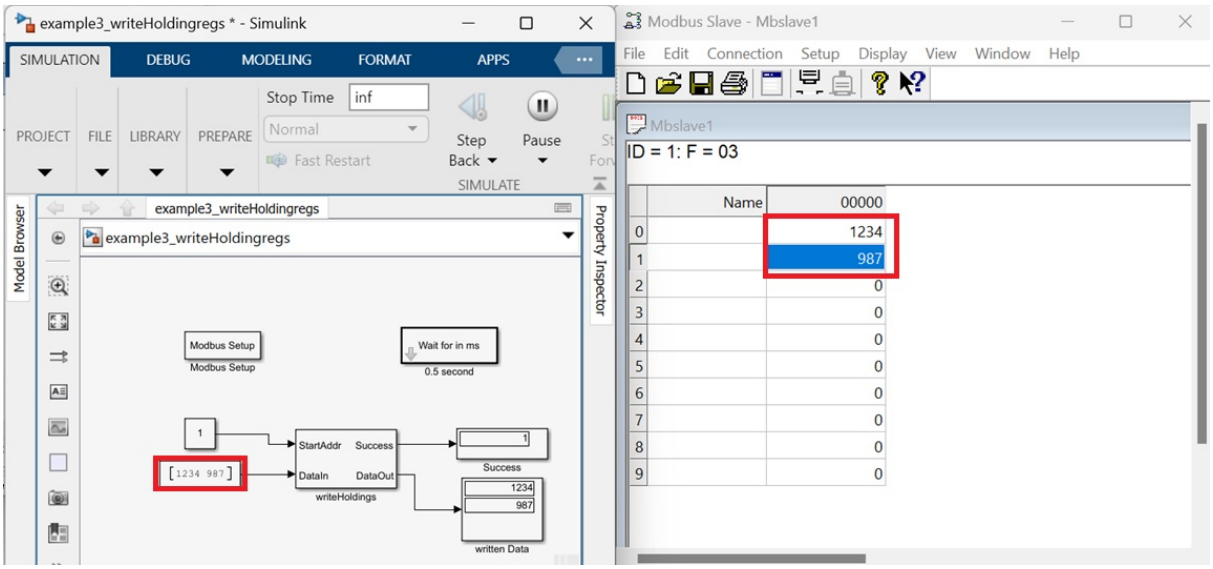
For "Modbus Setup" block, the parameters are set up below:



For "writeHoldingRegs" block, the parameters are set up below:



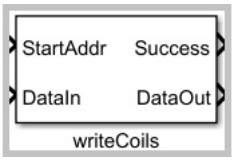
You can run general Modbus Slave Simulator software such as "Modbus Salve" to see holding registers values addressing 40001 to 40002 when we modify values we write to holdings.



writeCoils

write Modbus Server coil registers (Hardware: Modbus dual Masters Adaptor from Dafulai Electronic Inc)
Since R2019b

Library: Modbus Client (Dafulai Electronics) /writeCoils



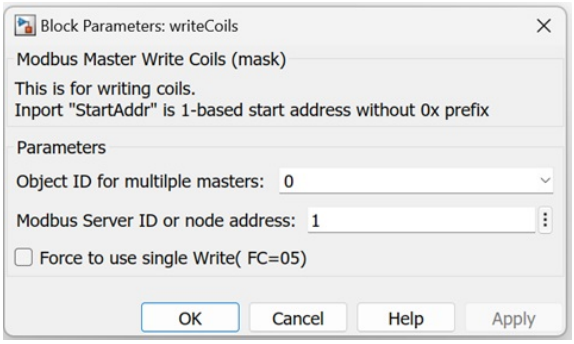
Description

This block writes coil registers. Start Address is from Input port "StartAddr" (1-based address without 0x prefix), Read Quantities are equal to the dimension of Input port "DataIn" .

If any error (CRC, Timeout, Not supported FC, address range,....) occurs, Output port "DataOut" will keep previous value, and Output port "Success" will be false.
Otherwise, write successfully, Out port "DataOut" will be input port "DataIn", and Out port "Success" will be true.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple masters — In one PC, we may use multiple "Modbus dual masters adaptor" hardware, this is for identifying each one.
- Modbus Server ID or node address — The address of the server to send this "write coil registers" command to.
- Force to use single Write (FC=05) — true means that we we will use multiple single writings (FC=05) to replace one multiple writing (FC=15).

Ports

Input

- StartAddr — "double" data type's scalar. It is Coil Regs start address (1 based without 0X prefix) you want to read.
- DataIn — "logical" data type's vector. It is data values you want to write.

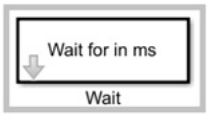
Outport

- Success — "logical" data type's scalar. true means write successfully. false means that failure in writing.
- DataOut — "logical" data type's vector. It is input port "DataIn" values when write successfully. The output port "DataOut" will keep previous values if fail in writing. Initial value is false vector.

wait

wait some time to pass.
Since R2019b

Library: Modbus Client (Dafulai Electronics) /wait



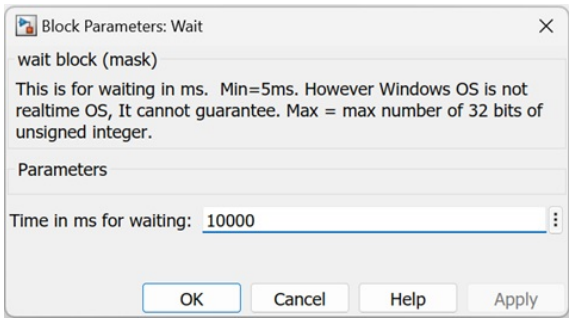
Description

This block will wait some milliseconds. This is block-function, it is different from simulated-sampling time, it is truly time. After the truly time passed, this block completes and it can run other remaining blocks in the entire model.

> ***Notes***: *Windows/Linux/MacOS is not real time OS. So this block cannot guarantee real time.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Time in ms for waiting — Delay time in milliseconds.

Ports

Input

None

Outport

None

9 Electrical And Mechenical Characteristics

Storage temperature .....-40°C to +85°C without Bluetooth, -5°C to +65°C with Bluetooth
Operating temperature .....-40°C to +85°C without Bluetooth,-15°C to +65°C with Bluetooth
Dimensions .....79.35mm x 42.67mm x 23..47mm (L x W x H)
DC Power Supply .....5.5 to 40VDC
Power Supply Current.....20mA at 12VDC Power supply
Maximum Bluetooth SPP Distance..... 30 Meters

IMPORTANT NOTICE

The information in this manual is subject to change without notice.

Dafulai's products are not authorized for use as critical components in life support devices or systems. Life support devices or systems are those which are intended to support or sustain life and whose failure to perform can be reasonably expected to result in a significant injury or death to the user. Critical components are those whose failure to perform can be reasonably expected to cause failure of a life support device or system or affect its safety or effectiveness.

COPYRIGHT