

DFLMBS001 Rev 1.10 Datasheet

Modbus Slave/Server Simulator

© 2024 Dafulai Electronics



Table of Contents

I Features Highlights	4
II Typical Application	4
III How to read this Datasheet?	4
IV Hardware	5
1 Pin Assignment.....	5
2 LED Indication.....	6
V Simulated Server Registers structure	7
VI Software	7
1 Software Interface.....	8
2 How to use?	9
VII State Machine and VBScript Example	30
VIII How to install Modbus Server Simulator Library in Matlab/Simulink?	40
IX MATLAB class	43
1 ModbusRTUASCIiServer class.....	43
2 ModbusTCPiServer class.....	54
X Simulink Blocks	61
1 ModbusServer_setup.....	61
2 readMyHoldings.....	63
3 readMyCoils.....	67
4 writeMyReadOnlyRegs.....	71
5 writeMyDiscreteRegs.....	76
6 resetDevice.....	79
7 enableDevice.....	81
8 COMFault	82
9 wait	83
XI Electrical And Mechanical Characteristics	85

1 Features Highlights

- Support as many as 5 Modbus RTU/ASCII servers, or One Modbus TCP server.
- All Modbus devices can be programmed by Graphic State Machine and simple VB script or by Matlab script or by Simulink.
- Support Bluetooth EDR 4.0 for monitoring Modbus server, USB for both monitoring and configuring Modbus server
- Configurable Modbus baud rates of 4800, 9600, 14400, 19200, 38400, 57600, 115200
- Configurable Modbus data format for none, odd or even parity and 1 or 2 stop bits.
- Configurable Modbus type RTU or ASCII
- Power by external supply range is 5.5V to 40VDC or/and USB Connection.
- Software configures hardware interface RS-232, RS-485, RS-422 without any hardware jumper.

2 Typical Application

- Use this simulator to develop/debug Modbus Master Controller
- Use this simulator to test Modbus Master Controller products in production line

The one benefit using the device simulator is that we can see all variables value of the device in real time by our PC_Utility software. You can know what command your device receives, and current Modbus register value. If you use actual device, you cannot get this data. It is very helpful for product development. The other benefit is that you can simulate device's all fault situations by vb script or Matlab script or Simulink blocks. In actual device, you cannot test all fault situations because you may damage your device. This is important for testing fault handling.

3 How to read this Datasheet?

If you use VBScript only, please only read [section 4](#) and [5](#) and [6](#) and [7](#)

If you use Matlab only, please read [section 4](#) and [5](#) and [6.1/6.2.1](#) and [8](#), and [9](#).

If you use Simulink only, please only read [section 4](#) and [5](#) and [6.1/6.2.1](#) and [8](#), and [10](#)

4 Hardware



Fig 1 Hardware of DFLMBS001

4.1 Pin Assignment

Female DB9 Connector:

DB9 Female

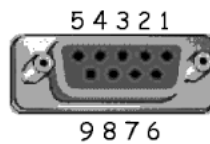


Fig.2 DB9 Pinout

Table 1 Pinout

Pin	Name	Description
1	RS485+/ RS422 TX+	This is RS485 + and RS422 TX +. This is Modbus network connecting point.
2	RS232 TX/RS485-/RS422 TX-	This is RS232 TX, RS485 - and RS422 TX-. This is Modbus network connecting point.
3	RS232 RX/RS422 RX+	This is RS232 RX and RS422 RX+. This is Modbus network connecting point.
4	RS422 RX-	This is 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. You can leave it unconnected, just use USB connection Power supply. Of cause, you can connect 5.5V to 40V, and use Bluetooth without USB
8	Reserved	Leave it unconnected
9	Reserved	Leave it unconnected

4.2 LED Indication

There are 4 LEDs to indicate the DFLMBS001's state. 2 LEDs' color is Green. 2 LEDs' color is Red

1 Mode LED (Red color)

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

If this LED is slow blinking, it means Modbus network is using RS422 interface.

If this LED is fast blinking, it means Bus configuration is in progress.

If this LED is dark, it means Modbus network 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 DFLMBS001 (this simulator) has connected PC.

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 DFLMBS001 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 TX LED (Red color)

When DFLMBS001 transmits any data to Modbus network, this LED will be on.

4 RX LED (Green color)

When DFLMBS001 receives any data from Modbus network, this LED will be on.

5 Simulated Server Registers structure

For Modbus RTU/ASCII, we can simulate as many as 5 servers (we call it 5 devices too). The 1st device to 4th device have the same registers structure, and the same watchdog. The 5th device has different registers structure, different watchdog.

We call one consecutive Modbus register addresses "Segment". For one kind of registers, we can have maximum 10 segments.

For example, we can have maximum of 10 holding registers segments, maximum of 10 input registers segments, maximum of 10 coils registers segments, maximum of 10 discrete registers segments. We use hardware to implement all registers, so it has hardware resource limit. All devices' word registers Quantities cannot be over 720, All devices' bit registers Quantities cannot be over 1024.

We have built-in hardware watchdog. There are 2 different hardware watchdog, one is using Read-only word register. The other is using Read-write word register. Server can only support one kind of watchdog. It cannot support 2 kinds of watchdog at the same time.

For Read-only word register watchdog, it is for master node to identify Modbus communication normal or fault. It has 2 watchdog types, type 0 for increasing 1 every Watchdog periods, type 1 for decreasing 1 every Watchdog periods. The outside master can read this Watchdog register value to know whether slave node is OK.

For Read-write word register watchdog, it is for slave node itself to identify Modbus communication normal or fault. Outside Master must write this watchdog register periodically. It has 3 watchdog types, type 0 for un-changed watchdog value which means value is decided by outside master, type 1 for increasing 1 every Watchdog periods until it equals Preset value, type 2 for decreasing 1 every Watchdog periods from Preset value until it equals 0. So Outside master must write watchdog value periodically, otherwise for any type of watchdog, slave node can easily know abnormal fault state. Customer's VBScript or Matlab script or Simulink block will do judgment of Modbus communication status.

For Modbus TCP, we can simulate only one server. However, we use PC hardware to implement Modbus TCP Server, so we have no limit to register segment Quantities. And we don't support watchdog. Watchdog must be implemented by your Matlab script or simulink block by yourself if you want.

6 Software

PC_Utility is a tool for configuring and monitoring DFLMBS001. (Monitoring is only for RTU/ASCII, cannot use for TCP)

PC_Utility software can be downloaded from our website http://www.dafulaielectronics.com/PC_Utility.zip.

It is free of charge software. This software must be run under Windows Vista/Windows 7 /Windows 8/ Windows 10. After download, you just unzip this software to any folder you like, and then please double click on PC_Utility.exe (or PC_Utility if you hide file suffix) to run this software.

Notes: 1 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 2 PC_Utility software only supports Modbus RTU/ASCII servers, it does not support Modbus TCP Server. For Modbus TCP Server, please use MATLAB or Simulink. You cannot run PC_Utility software and MATLAB/Simulink at the same time.

6.1 Software Interface

Please see screenshot below after you run PC_Utility software:

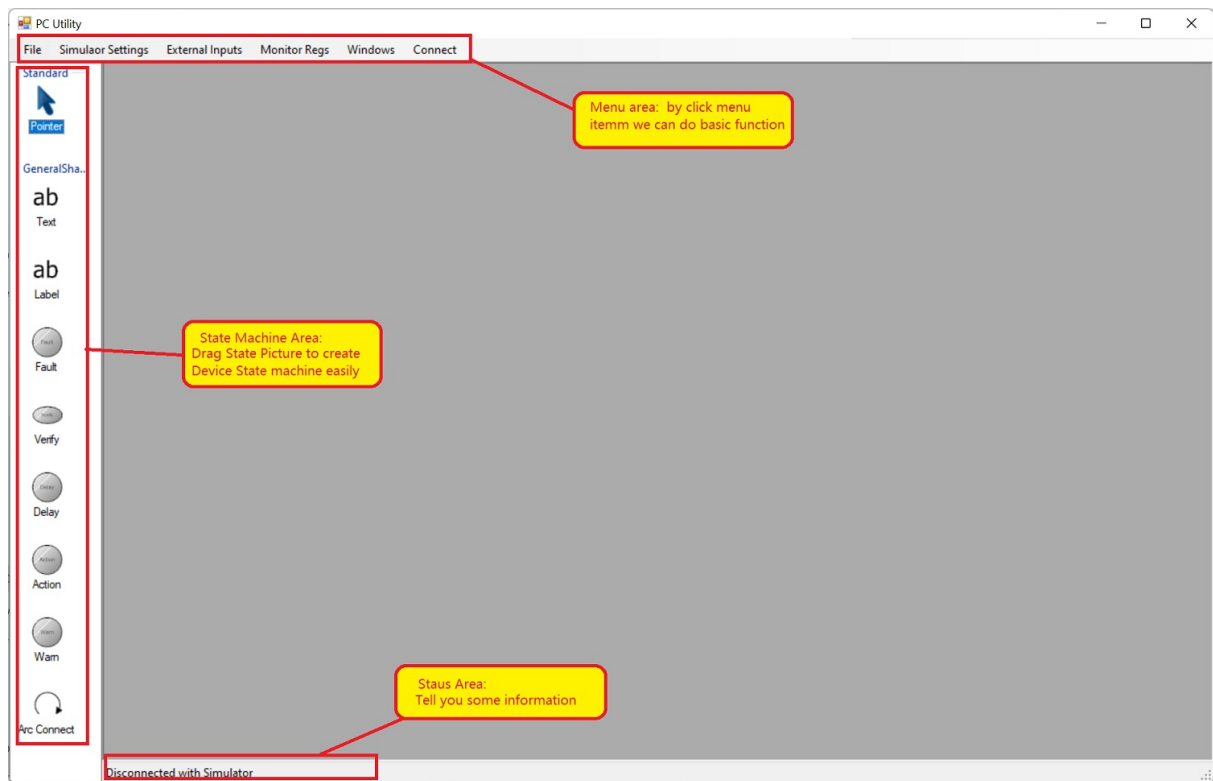


Fig.3 Software Interface

There are 3 areas. Menu area is for basic function, you can use them to configure and open /save state machine, monitor Modbus Registers and give external digital input or set up faults or warnings.

State machine area is for creating each simulated device's finite state machine.

Status Area is for information display such as connection between PC and Simulator.

6.2 How to use?

- 1 The first thing you have to do for new simulator is configuration of Modbus Bus. Please click Menu item: " Simulator Settings / Bus Settings". The following dialog occurs:

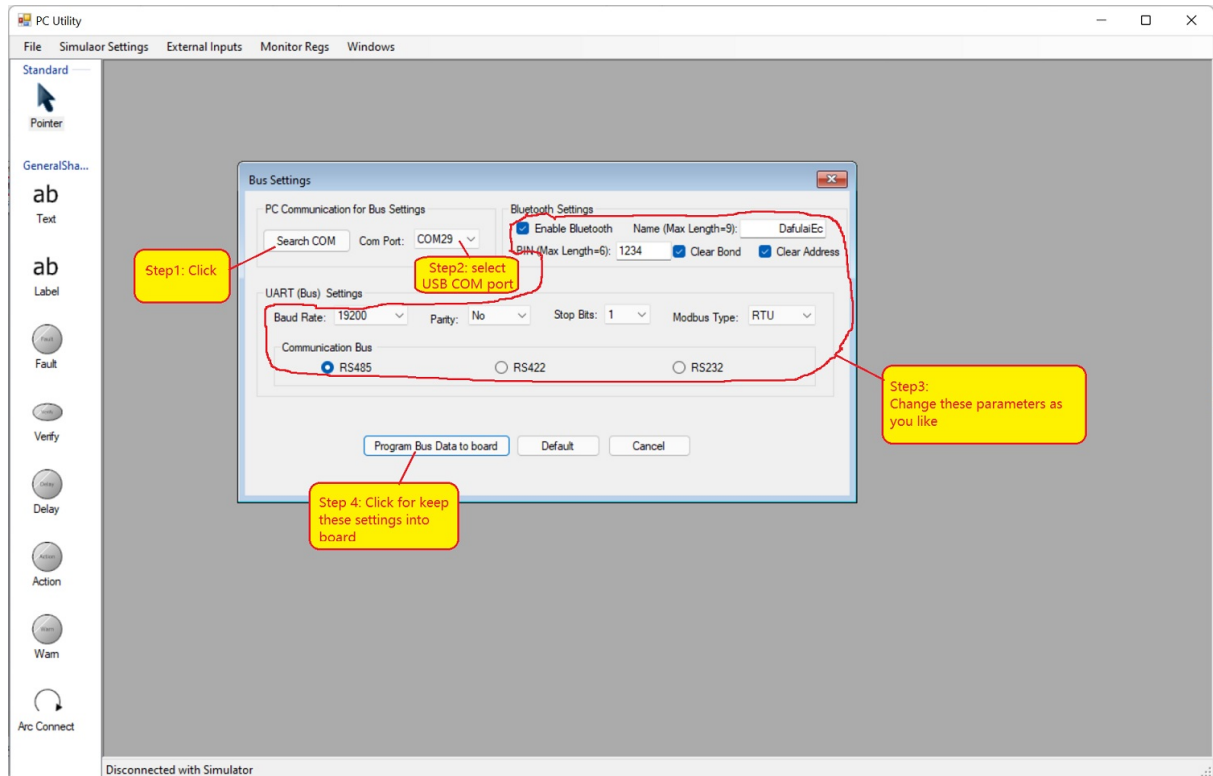


Fig.4 Bus Config

Please follow instructions above screenshot. In general, this bus settings are only done once if your Modbus settings are not changed. Of course, you can do many times as you like. One thing you have to pay more attention is that you only can select USB COM port. You cannot select Bluetooth COM port.

Your team worker may have set up Bus settings, you just use menu item: "File/Open" to open .bcfg suffix file. It will save your time.

Or you want to save your bus configuration for your co-worker use, just use menu item: "File/Save" or "Save as" to save to .bcfg file.

If you only use MATLAB/Simulink, please don't read the remaining content, just read [section 8](#) and [section 9](#) for MATLAB and read [section 8](#) and [section 10](#) for Simulink..

2. You should configure Device-related Settings. Please click Menu item: Simulator Settings / Devices Settings. (In this situation, Com port can be Bluetooth serial port or USB serial port) The

following dialog occurs:

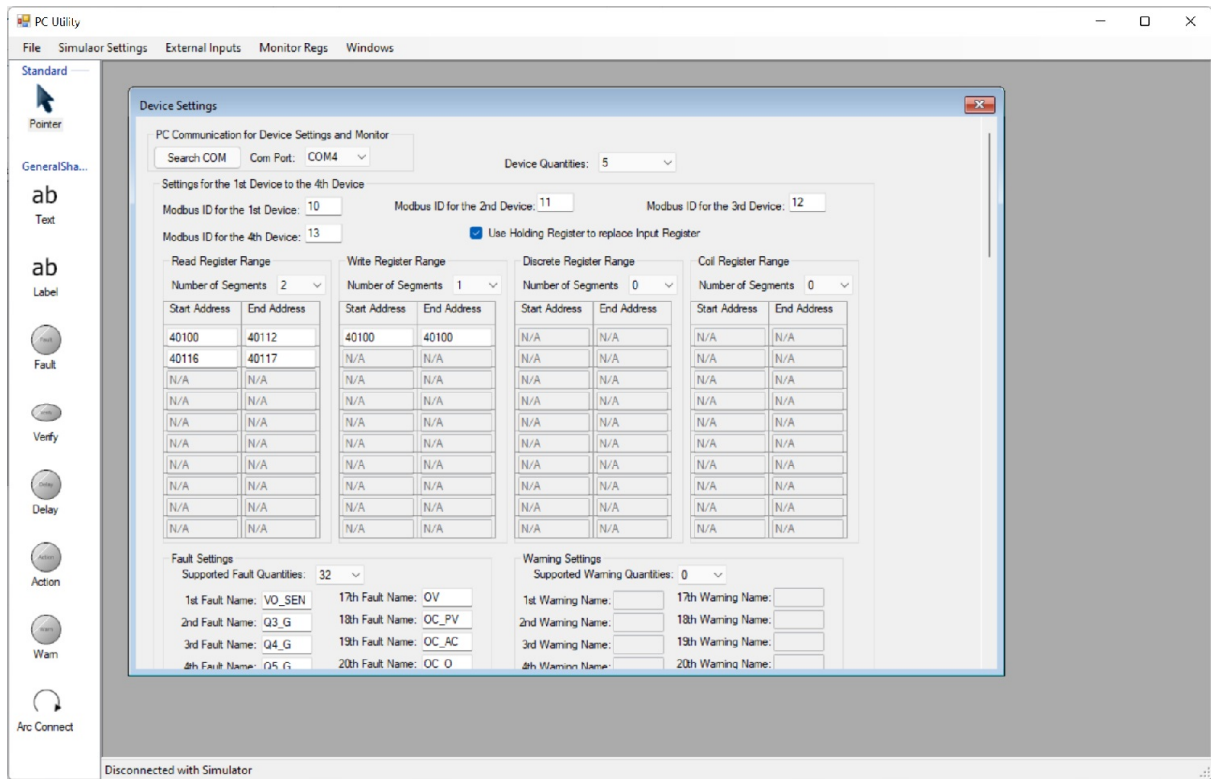


Fig.5 Device Config.

This stage is to set up Device's Modbus registers structure, watchdog and Fault/Warning/Digital Inputs. The Device 1 to 4 have the same Modbus registers structure, watchdog and Fault/Warning/Digital Inputs although their finite state machines can be different.

The 5th device has different Device's Modbus registers structure, watchdog and Fault/Warning/Digital Inputs. Of course, you can set to the same as that of the 1st to the 4th device.

Button "Search COM" is for searching available COM ports, the search result will become the items of com port drop list. Please select correct Simulator COM port. Not like Stage 1, you can select Bluetooth COM port. From "Device Quantities drop list", you can select how many devices you want to simulate. Enter Modbus ID (Or called Modbus Device address) for each device. Check box "Use Holding Register to replace Input Register" is checked when read-only registers use Holding registers.

We divide registers into different segments. One register segment is a register range with continuous address. For any kind of registers, we only provide maximum 10 segments. It is enough for most applications. Every register segment consists of "Start Address" and "End Address". "End Address" must be bigger than or equal to "Start Address". Register segment cannot be overlapped. The latter row segment address must be after former row segment address.

Fault Setting is for simulated fault signals (Boolean value). One device can be as many as 32 faults. Fault name you set here will display in "fault And Discrete Input Dialog". Similarly for warning settings, it is similar to fault.

In above window, we cannot see 5th fault name to 16th fault name, the 21th fault name to the 32th fault name. Please drag vertical scroll bar to display more content, see screenshot below:

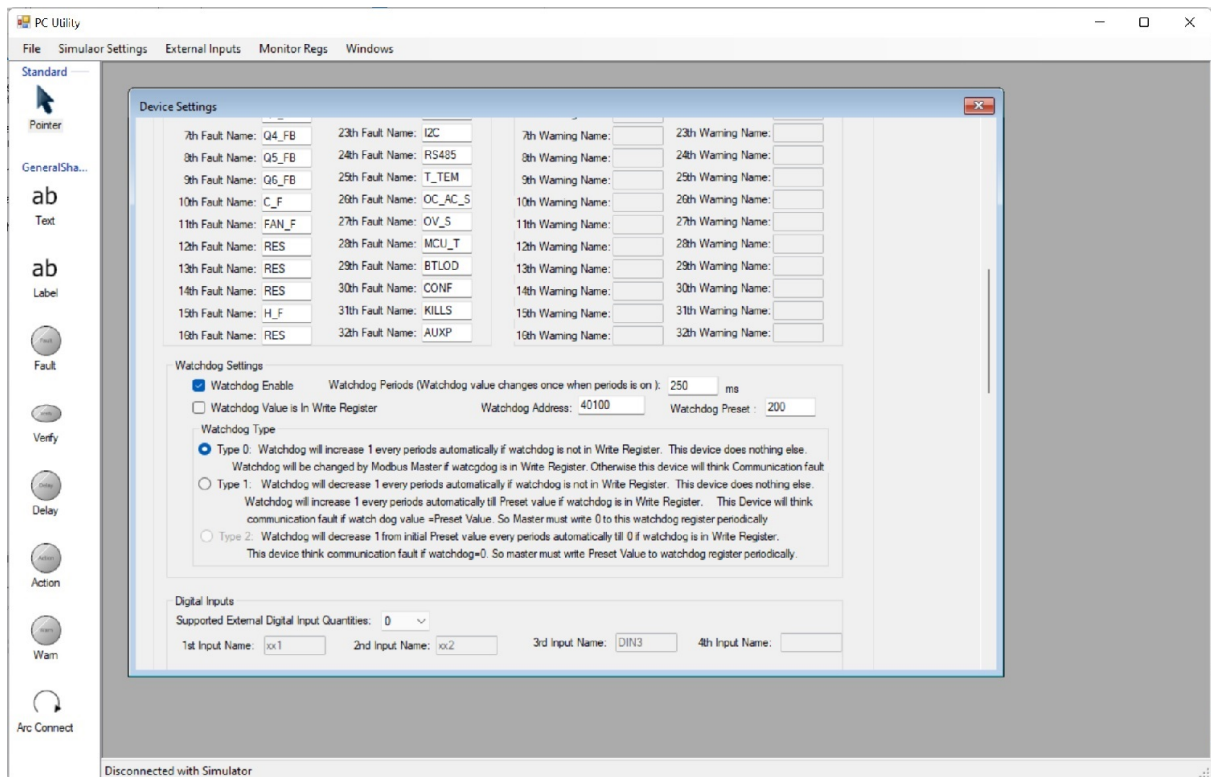


Fig.6 Device Config.

If you send any fault signal by "fault And Discrete Input Dialog", device will enter fault state automatically, you don't need to write script to let device enter fault state. Of course, you can use script to let `FaultInput(i) = true` to enter fault state, or use script to let `FaultInput(i) = false` to clear fault. We will introduce in script paragraph latter.

You can enable/disable device watchdog function. Watchdog register can be in Read register or in Write register. You can choose by related check box.

For read register watchdog situation, there are 2 types, type 0 is watchdog value increased 1 automatically and return to 0 when overflow and increase again (free running). type 1 is decreasing 1 (free running).

This kind of watchdog is for master Modbus to judge communication fault easily. It is not for Modbus slave device.

For write register watchdog situation, there are 3 types, Type 0 is that watchdog value changed

only by master. If no change within Watchdog Periods, Device will have communication fault. Detecting this communication fault should be implemented by your script. Simulator cannot get communication fault automatically. Type 1 is that watchdog value increased 1 every watchdog period automatically until preset Value. If Master write 0 to watchdog periodically, watchdog never reaches preset Value. So your script will read watchdog value, it will set communication fault if watchdog value= preset value.

Type 2 is that watchdog value decreased 1 every watchdog period automatically from initial preset Value until 0. If Master write preset value to watchdog periodically, watchdog never reaches 0. So your script will read watchdog value, it will set communication fault if watchdog value=0.

Digital Inputs is for simulating discrete input (or called digital input), It is Boolean value. Of cause it can be not actual digital inputs, it can be sun light on or dark for PV DC/DC converter. So it can simulate sunrise. you just write script to read DigitalInput(i) , you will set EnableCom(i) =false when sun light is dark. You can set up 0 to 4 digital inputs for each device.

The 5th device setting is the same as the 1st to 4th device. We don't explain more. Drag vertical scroll bar to display 3 buttons: "Program Device Settings into simulator", "Read Device" and "Cancel"

Button "Program Device Settings into simulator" will put settings into simulator flash memory.

Button "Read Device" will read out simulator settings.

Button "Cancel" will cancel settings and close dialog.

Your team worker may has set up device settings, you just use menu item : "File/Open" to open .dcfg suffix file. It will save your time.

Or you want to save your device configuration for your co-worker use, just use menu item : "File/Save" or "Save as" to save to .dcfg file.

Notes: There is resource limit for simulator. The quantities of all write registers and read registers for all devices must be less than or equal to 720. The quantities of all coil registers and discrete registers for all devices must be less than or equal to 1024.

3 In this stage, You will set up finite state machine and script for each device.

What is finite state machine?

"Finite State Machine" (or called "State Machine") is a mathematical model of computation. In any time, system always runs exact only one state of all finite states, the other states are not running.

The jump from one state to another state needs some conditions. When conditions are met, system will execute special action and jump to another state. Please see figure below. The initial State is "State 1". System executes code in State1.

When Condition is met, system executes code in Action and then enters "State 2" and executes code in "State 2"

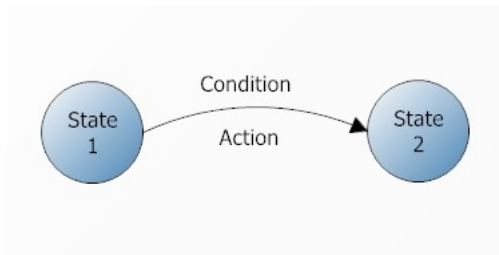


Fig.7 State Machine

We use simplified "Finite State Machine" which has no jump condition and action. All jumps are unconditional and no any action. Please see figure below, its function is the same as figure above.

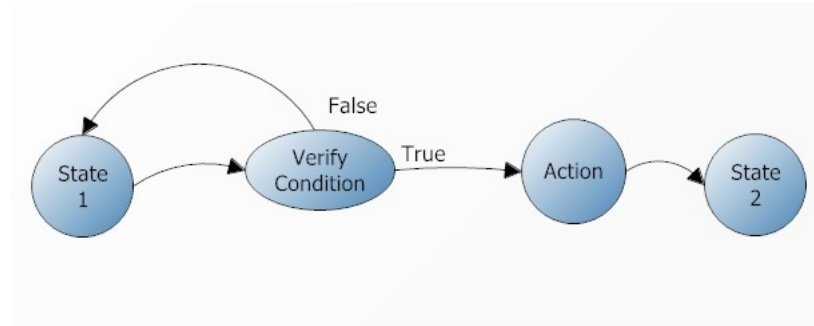


Fig.8 Simplified State Machine

We add a special state "Verify Condition". When Condition is true, it will enter state "Action", other return to "State 1". And code inside "State 1" completes, it will enter "Verify Condition" again unconditionally.

All jumps in the simplified "State machine" are all unconditional.

In general, the result of code running (variable or array value) is non-volatile. So we can use the following state machine to replace it.

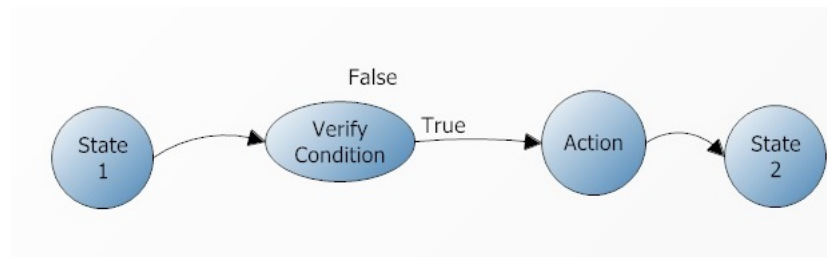


Fig.9 Simplified State Machine

Our State machine is this kind of simplified "Finite State Machine". In the left tool box, we provide 5 kinds of state: "Fault State", "Verify State", "Delay State", "Action State", and "Warn" State.

Let us explain these states:



: Fault State. When any FaultInput(i) (i=0 to Fault Input Qty -1) is true, it will enter this state automatically (no need any script). And state color becomes Red, and run VB Script code inside this state. When all FaultInput(i) (i=0 to Fault Input Qty -1) is false, it will exit automatically, Which state it will exit to is decided by your state connection line.



: Delay State. When enter this state, it will display count-down time in ms and State color becomes Green. When count-down to 0, it will exit.



: Action State. When enter this stat, it will run VB script code inside it once, State color becomes Green. and then exits. State color becomes Grey again.



: Warn State. It is similar to Fault State. When any WarningInput(i) (i=0 to Warning Input Qty -1) is true, it will enter it automatically. And color becomes yellow, and run VB Script code inside this state. When all WarningInput(i) (i=0 to Warning Input Qty -1) is false, it will exit automatically, Which state it will exit to is decided by your state connection line.



: Verify State. When enters this state, it will display count-down time (time out) in ms and State color becomes Green. When condition variable "Verify" in VB Script becomes true, it will exit automatically, Which state it will exit to is decided by your state connection line to "True point". If count-down to 0 and "Verify" is still false, it will exit automatically, Which state it will exit to is decided by your state connection line to "False point". When we set up time out=0, the "Verify" State will execute once to decide next state. Of cause, if "False" does not has any state connected, it will wait for true forever. "Verify State" can do extra action like "Action" State. The left side of "Verify State" (the horizontal line of "True") is True although there is no label True. The Top side of "Verify State" (the vertical line of "False") is False although there is no label False.

Now we can set up state machine for simulated device. For new devices, you should know its

principle, and divide it into many different states. VB Script for every state only needs a few of assignment statements. (If you only use one State, all functions are put into complex scripts, it can work, But your job becomes very complex and not reliable).

Click menu item: "File/New State Machine" will Open 5 empty state machine windows (If your devices are less than 5, don't worry, just create your device state machine, leave unsupported device state machine empty). See Screenshot below:

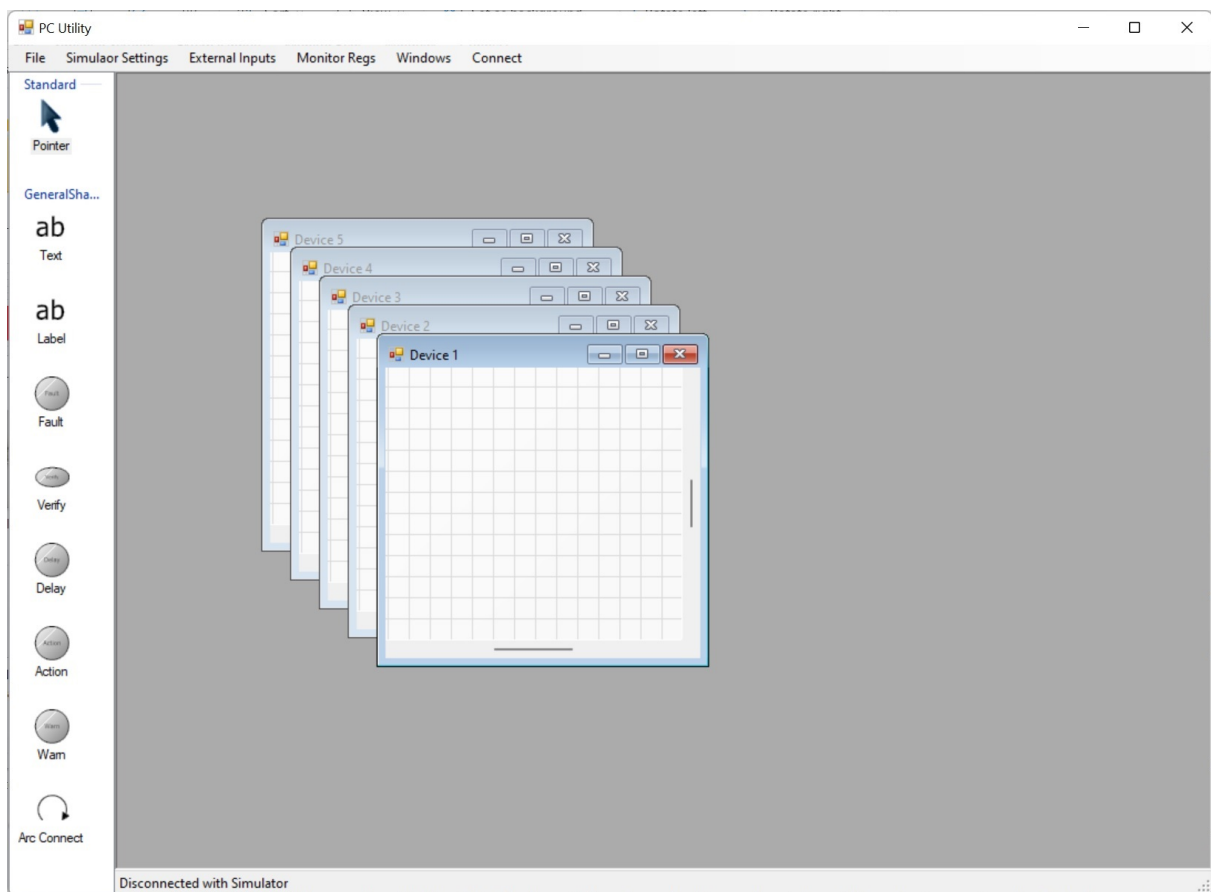


Fig.10 New State Machine

You can click Windows/Horizontal Title or others to organize your State machine window. Firstly we work in Device1 (Windows title is Device 1), Maximize this window, and click state icon in the State Tool box, and click on the client position of Device 1 window. This state will be put on Device 1 state machine. See result for "Action" State as example below:

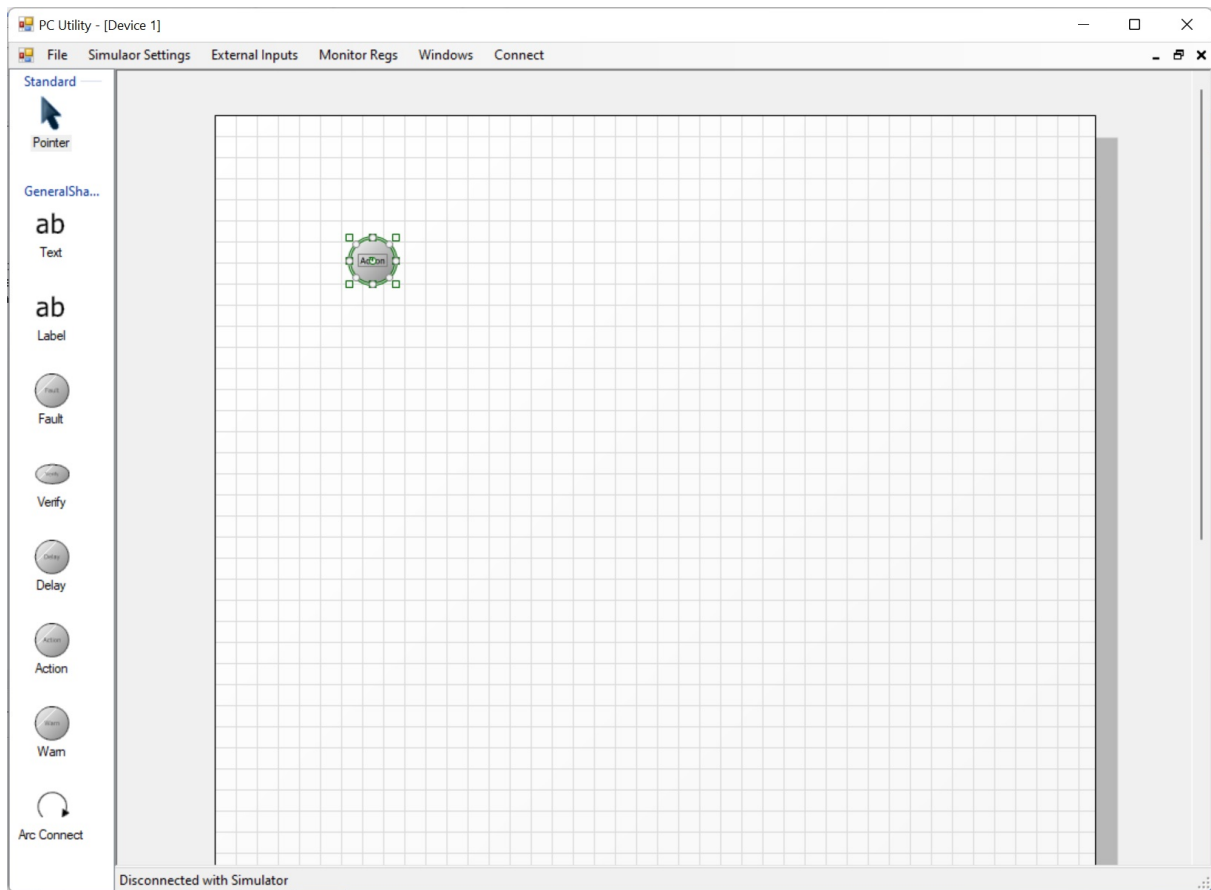


Fig.11 put one state

The 8 green squares can be used to adjust state picture size. You can click on the state name to change default name to any text from "Action". You can drag this state to any position. Single Click on this state will make border of state to become thick, which means it is initial state. If you don't want to change initial state, click menu item "File/ Initial State Lock" to lock initial state. We create 7 states by this way , see screenshot below:

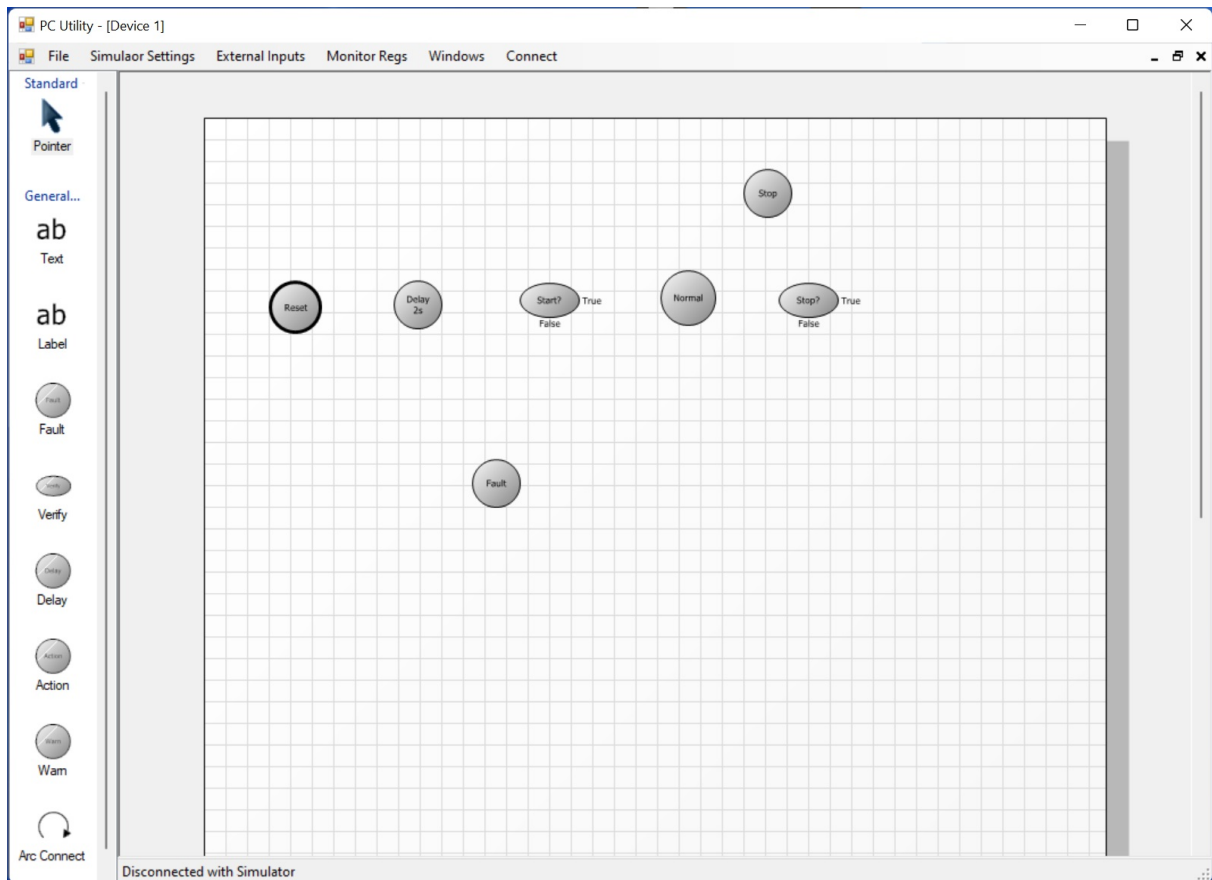


Fig.12 All States

The 7 states are "Reset" (Action), "Delay 2s" (Delay), "Start?" (Verify), "Normal" (Action), "Stop?" (Verify), "Stop" (Action) and "Fault" State.

The basic function is that Power on device and then device reset (all registers will be zero) and delay 2 seconds, and then detect start command. Device will enter Normal run if start command receives. And device will detect stop command. Device will enter Stop state if stop command receives. And then detect start command again, No matter what state device is, device will enter fault state if fault occurs. Device will enter reset state if fault disappears.

This is typical device. Click "Arc connect" icon in state machine tool box. And then put cursor on one of small square of "Reset" state, Cursor will become a small hand, and then click on "small hand".

Move cursor to one of small square of "Delay" state. Cursor becomes a small hand, click on "small hand". The connection between "Reset" and "Delay" will be set up. The connection maybe not good like below:

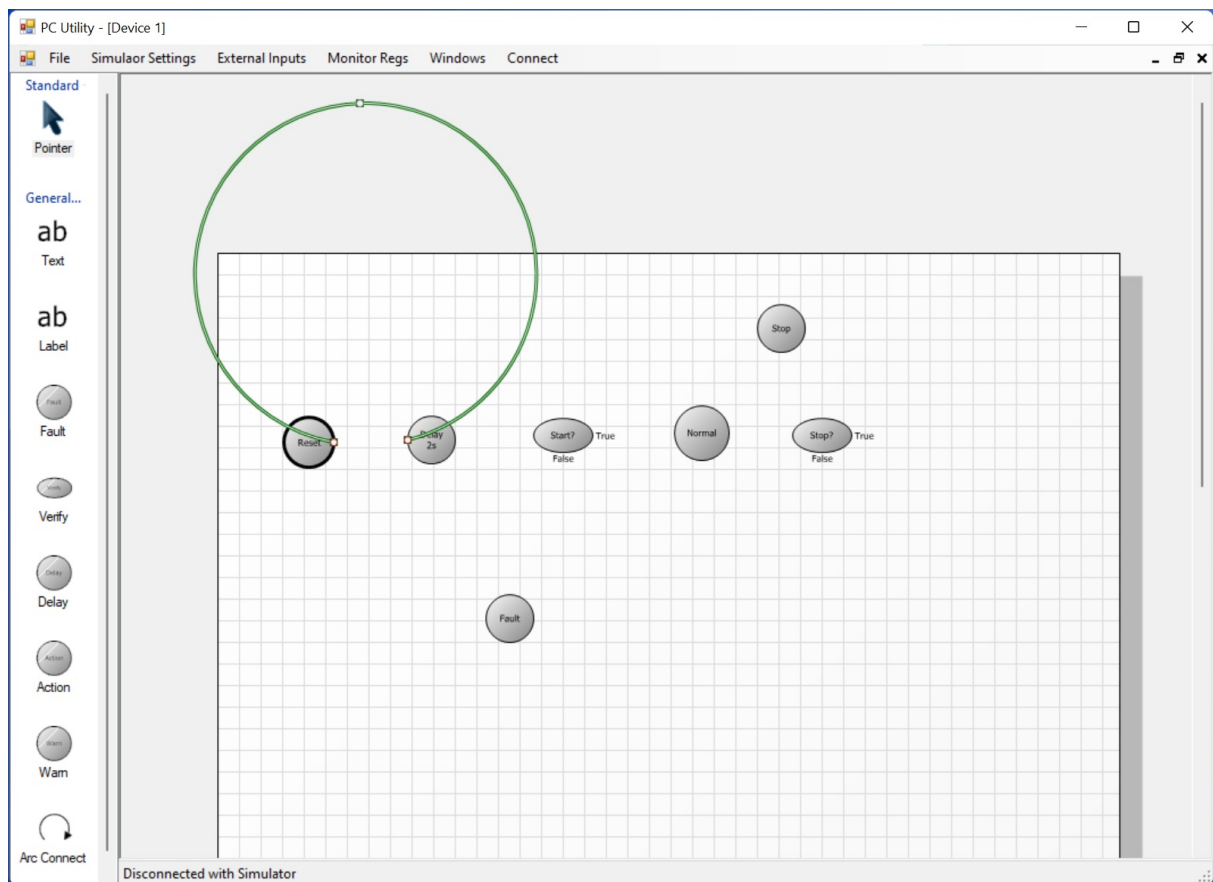


Fig.13 State Connection

Don't worry, just drag middle square of "Arc Connect" line to good position such as screenshot below:

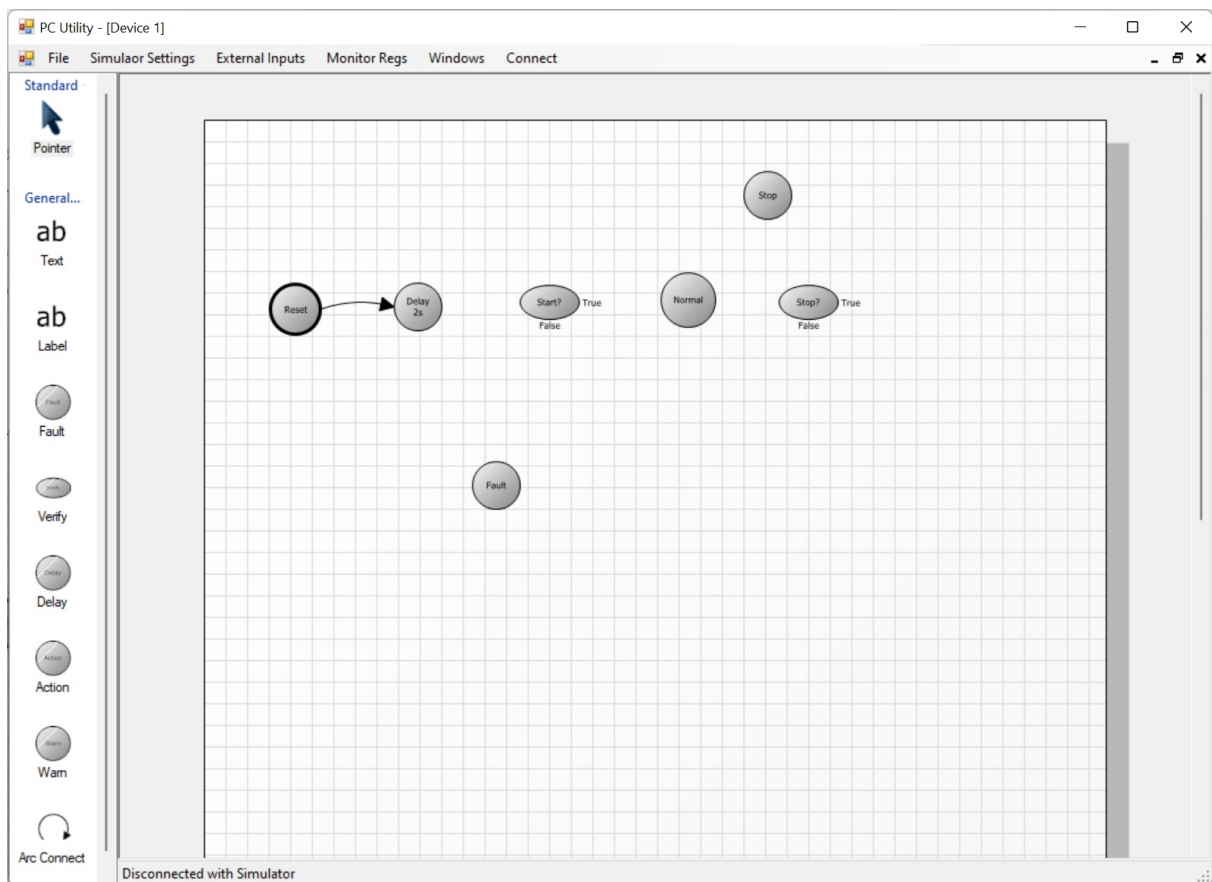


Fig.14 State Connection

We complete all states connections like below:

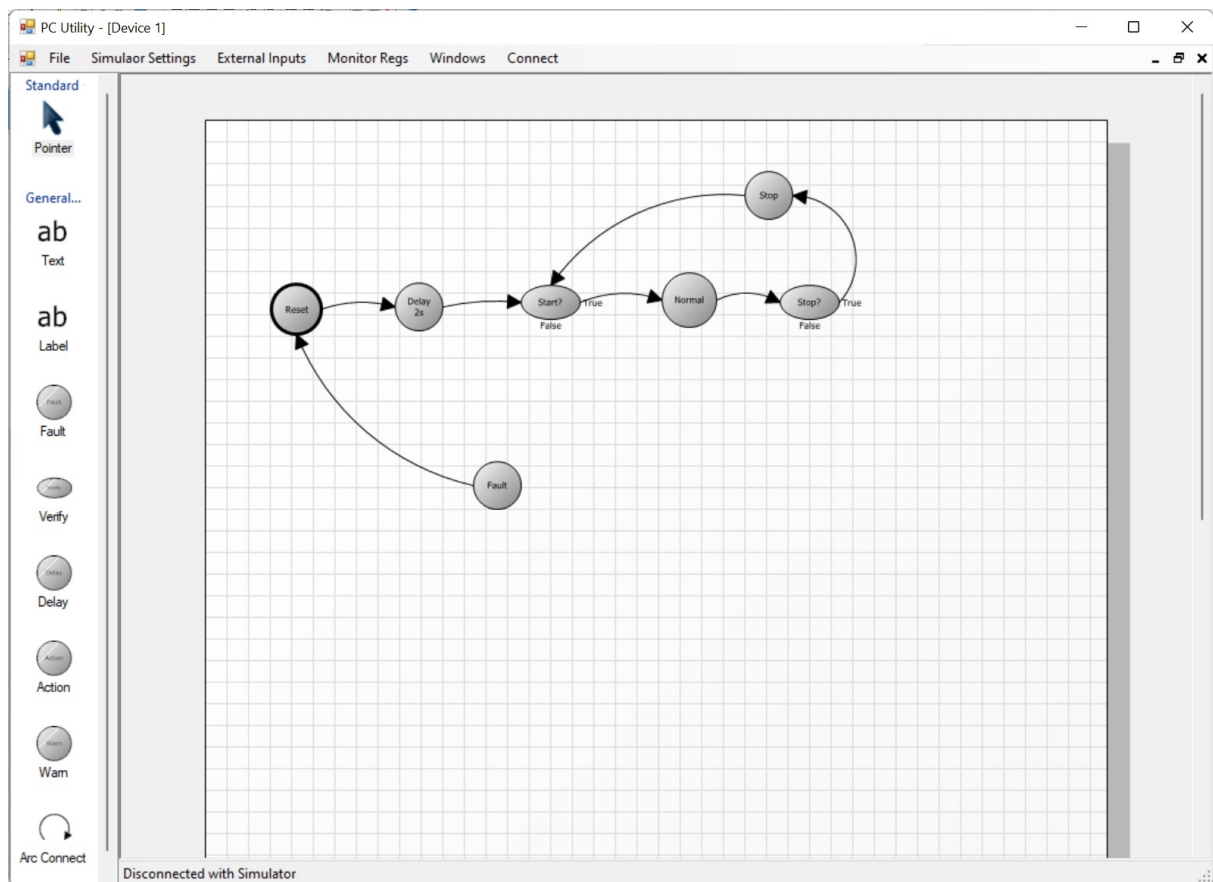


Fig.15 All States Connections

Notes: Sometimes, your connection is not good although visibly it is connected. You can verify whether connection is good by moving state. If state moves and its connection line moves with it automatically, it means connection is OK.

You can add comment by "ab Text" icon, or "ab" Label". What difference between "ab Text" icon, and "ab Label" ? "ab Text" is for entire state machine, it is not attached to any state or "Arc connect". However, "ab Label" is for one state or one "Arc Connect", it is attached to one state or "Arc Connect". When attached state or "Arc Connect" moves, "ab Label" moves with it automatically.

Now lets design scripts.

Script language is VB. Its Syntax is simple. It is almost the same as Microsoft VB.NET. However, it has some difference. In Dim statement, you cannot use UInt16 to replace UShort, cannot use Int16 to replace Short, cannot use Int32 to replace Integer, cannot use UInt32 to replace UInteger. Most of complex predefined class will be not supported. Script does not support "IIF function"

Every Device supports the following global variables or array:

- **ReadReg()** : Modbus Read registers, Type is UShort, index is register address with prefix.

For example, ReadReg(30006) is input register value at address 30006. ReadReg(40006) is holding register (but read only) value at address 40006.

- **WriteReg()** : Modbus Write registers, Type is UShort, index is register address with prefix. For example, WriteReg(40010) is holding register value at address 40010.
- **CoilReg()**: Modbus Coil registers, Type is boolean, index is register address with prefix. For example, CoilReg(00010) is Coil register value at address 00010.
- **DiscreteReg()**: Modbus Discrete registers, Type is boolean, index is register address with prefix. For example, DiscreteReg(10010) is discrete register value at address 10010.
- **DigitalInput()**: 4 digital inputs, Type is boolean, index is from 0 to 3, which denotes the 1st to 4th digital input. You can change its value by menu "External Inputs"/"Fault and digital inputs". Please see figure below:

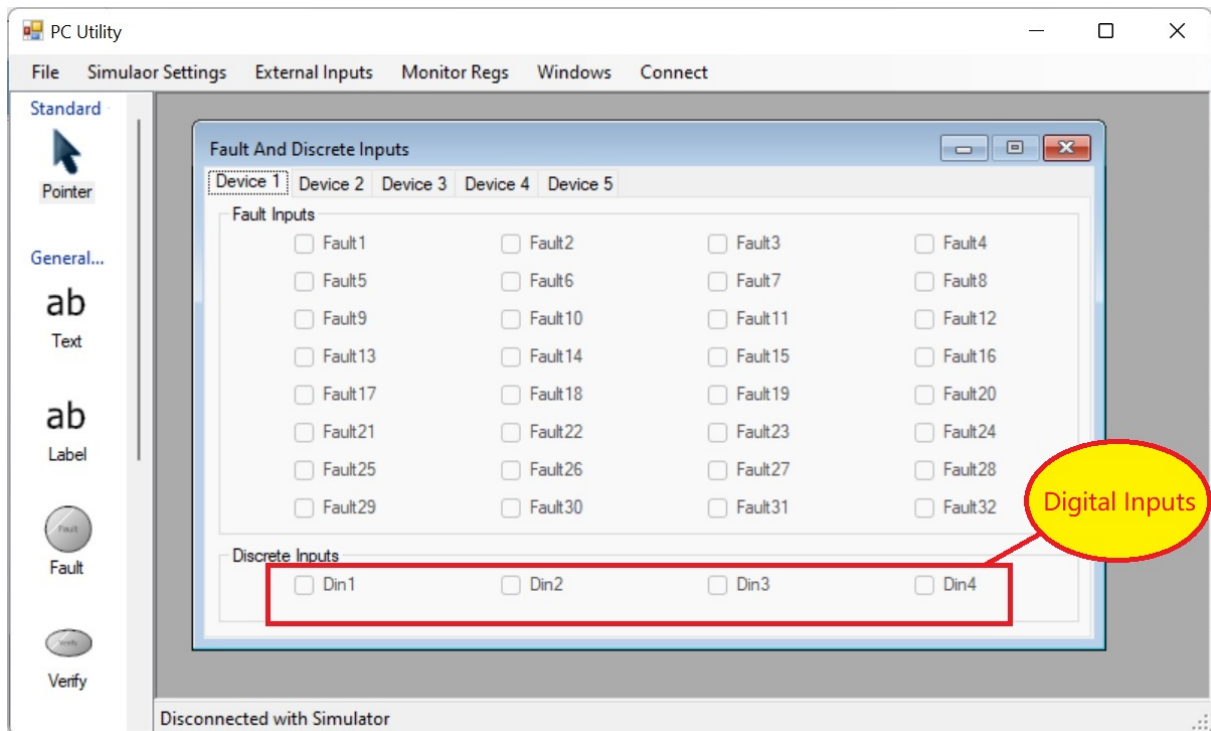


Fig.16 Fault and Digital Inputs

- **FaultInput()**: 32 fault inputs, Type is boolean, index is from 0 to 31, which denotes the 1st to 32th Fault inputs. You can change its value by menu "External Inputs"/"Fault and digital inputs". Of course, you can set/clear by your script.
- **WarningInput()**: 32 Warning inputs, Type is boolean, index is from 0 to 31, which denotes the 1st to 32th Warning inputs. You can change its value by menu "External Inputs"/"Warning inputs". Of course, you can set/clear by your script.
- **EnableCom()**: Enable device Modbus communication, Type is boolean, index is from 0 to 4, which denotes the 1st to 5th device. You can set it to false to disable some device Modbus communication. For example, EnableCom(2) =false will make Device 3 lost Modbus

communication (This is usually for solar device. When at night, No sun light will cause no power, and no Modbus communication)

- **Verify:** "Verify State" result, Type is boolean. This is only for "Verify state" script. For example, "Verify State" to check whether holding register 40001 is 2, the script will be "Verify=(WriteReg(40001)=2) "
- **Reset:** Make device reset. Type is boolean. When it is true, it will make all registers to zero. You don't need to make "Reset" variable to false after you make it to true. System give it to false after executing "reset device".

You don't need to remember these variable names, you just need press "Ctrl+ Space" to display variable name in script Editor.

One special State machine script is "Delay state". The script is only constant or WriteReg variable. For example, Script is " 2400" means delay is 2400ms. Or script is "WriteReg(40108)" means delay value is decided by Holding Register value at address 40108 .

Double click on "Reset" State, a window pop out. see picture below:

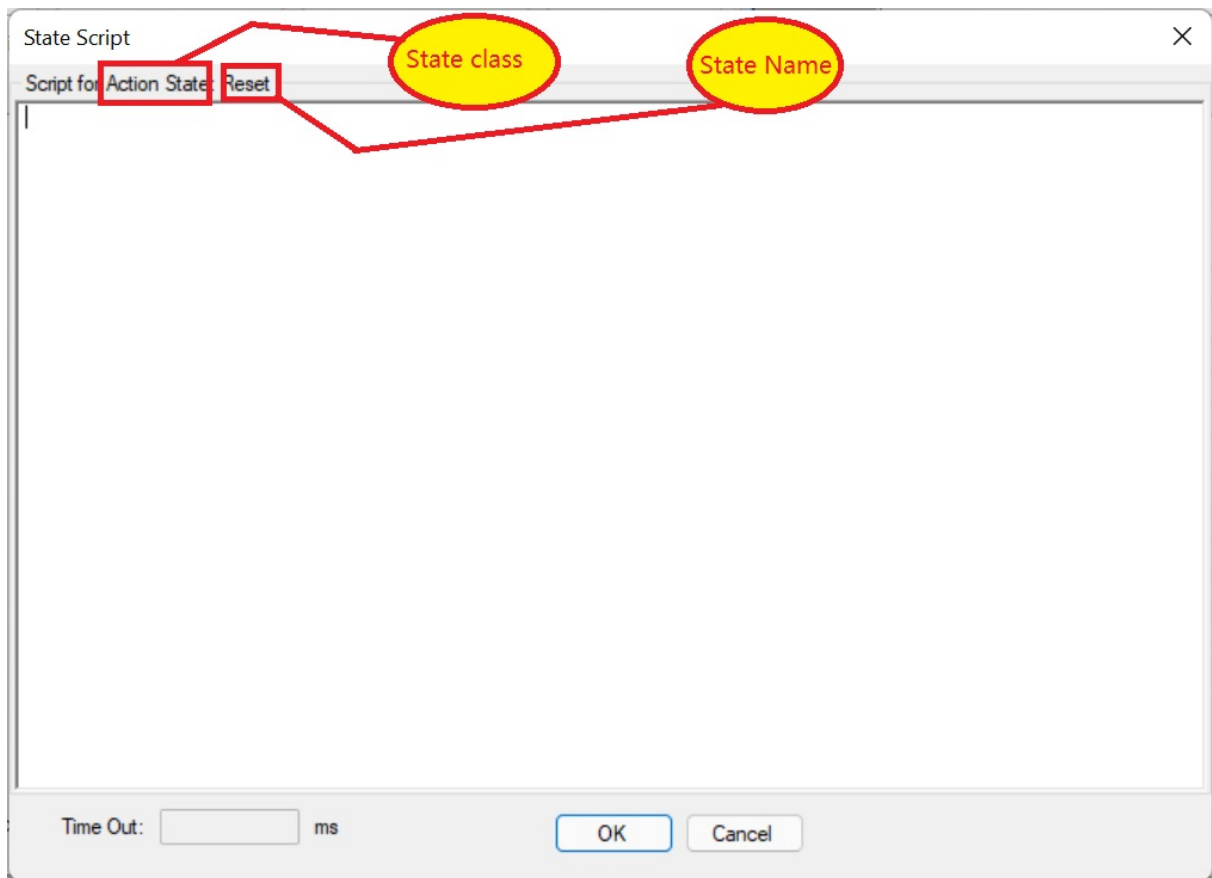


Fig.17 Script window

Our "Reset" State (Action) function is to make all registers to zero. So we click on editor window,

and press "Ctrl+ Space", variable hint occurs, click on "Reset" item, then type "=True"

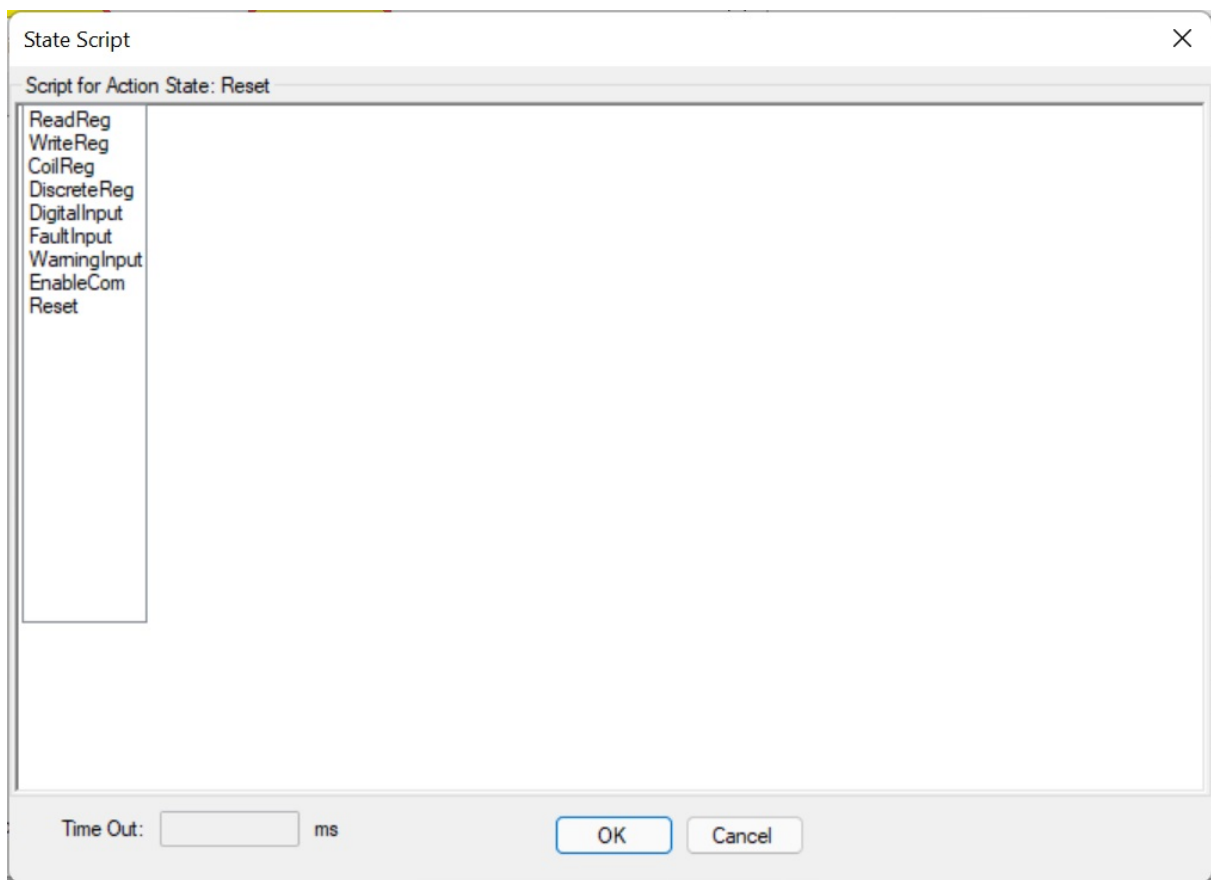


Fig.18 Script window

The script is shown below:

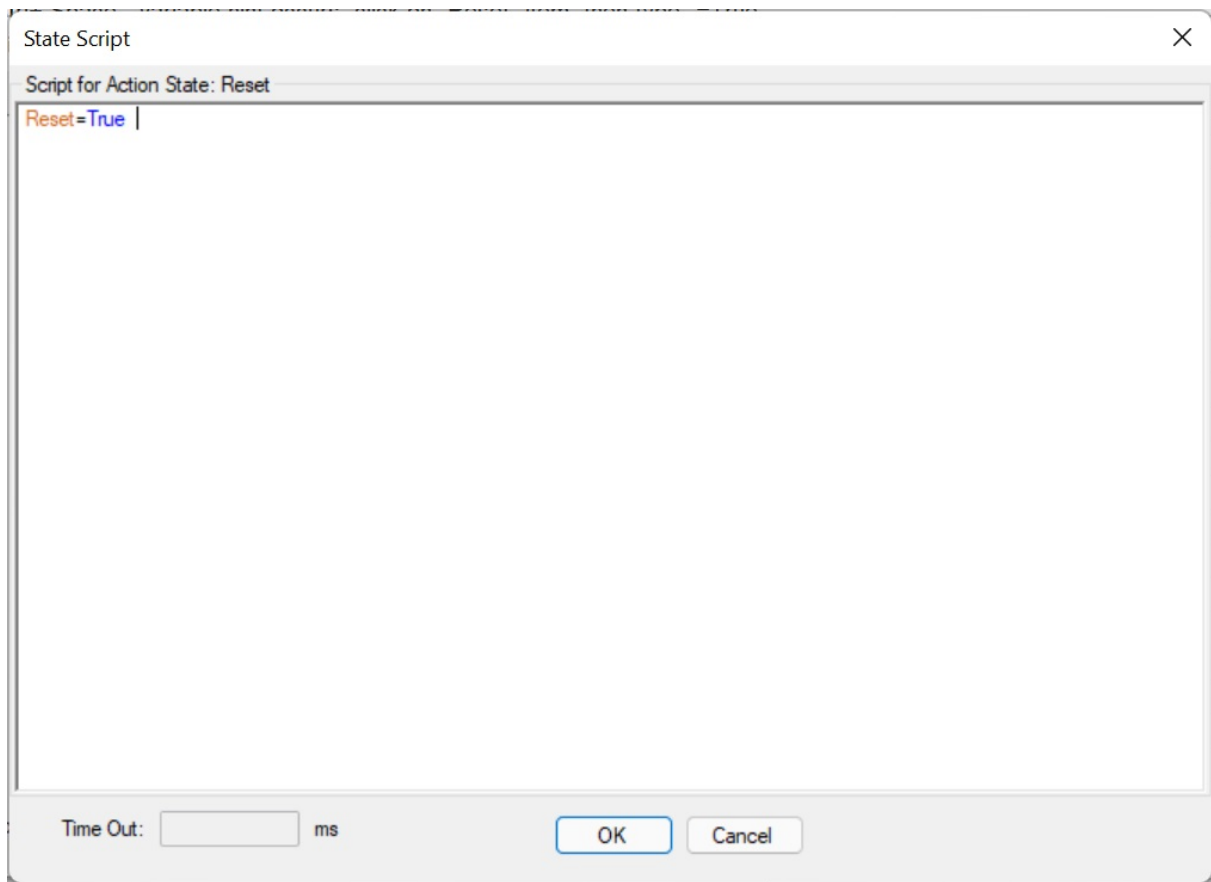


Fig.19 Script window

Click Button "OK" to accept script.

- Notes:*
- 1. There are 2 hints, one is variables, the other is statement keywords. These 2 hints alternates by "Ctrl+Space"*
 - 2. Script has syntax color. If it does not occur syntax color, please right click to click Syntax color menu item.*

Double click on "Delay 2s " (Delay) State. Pop out script window, we complete script such as window below:

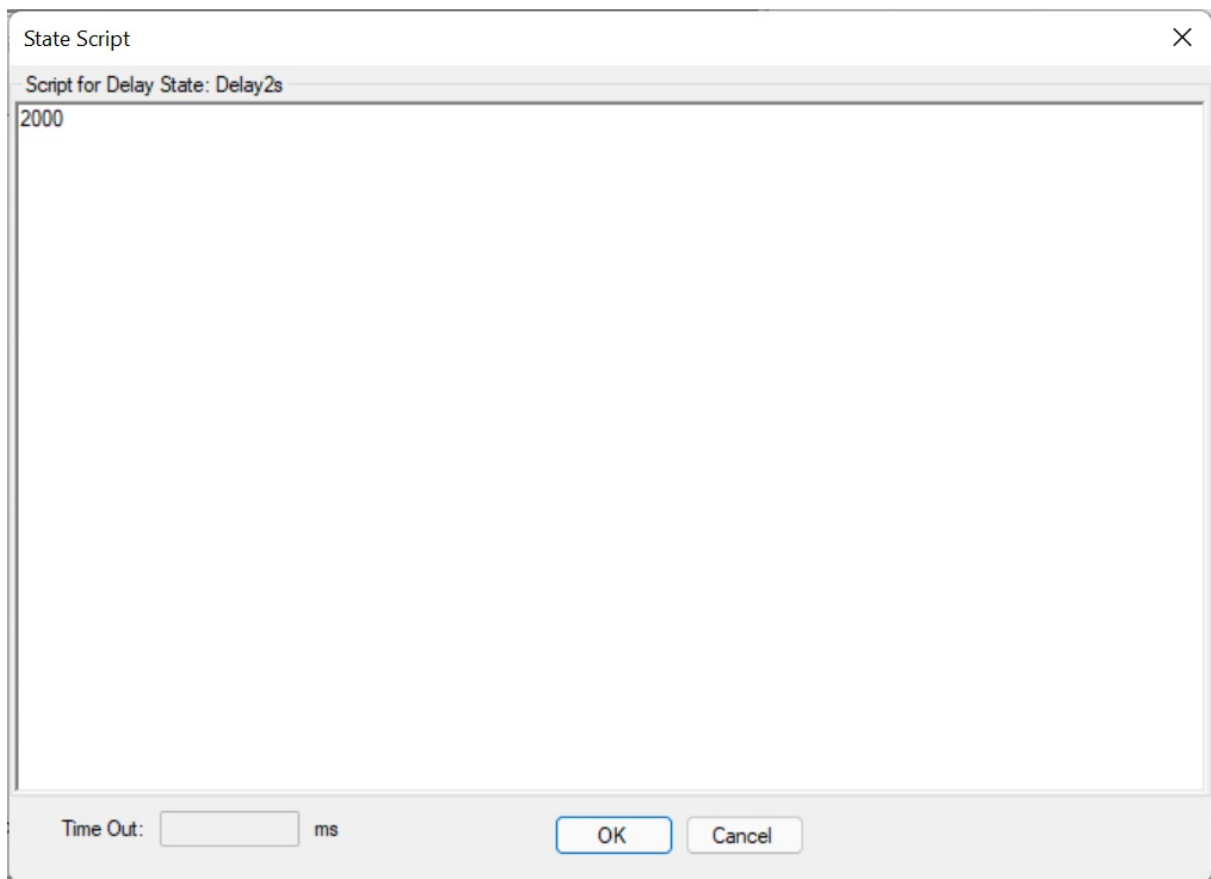


Fig.20 Script for "Delay State"

Click Button "OK" to accept script.

Double click on "Start? " (Verify) State. Pop out script window, Our "Start?" state function is to see if holding register 40100 is 1 (Start command is Holding Register 40100 value with 1). So Script content is shown below:

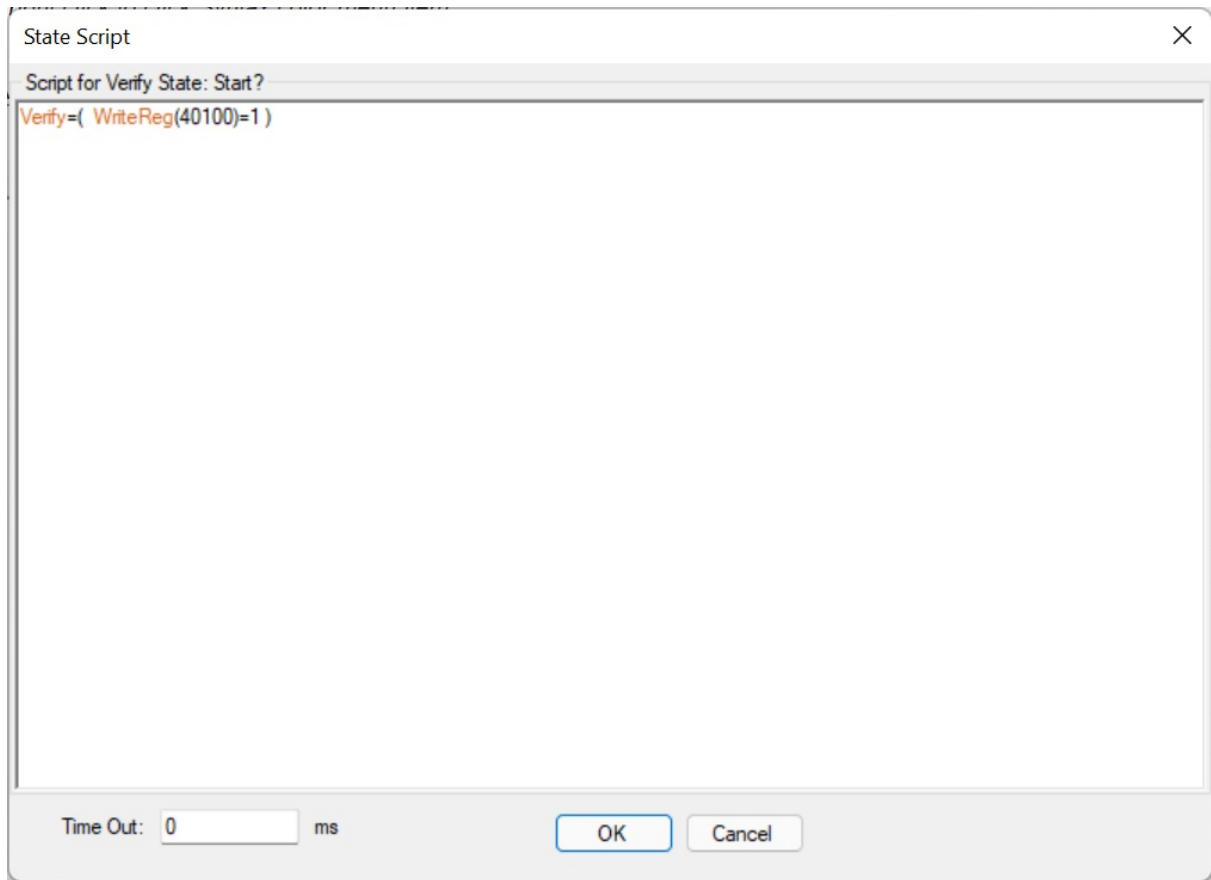


Fig.21 Script for "Verify State"

Time Out = 0 ms means that we wait for true forever. Click Button "OK" to accept script.

Double click on "Normal " (Action) State. Pop out script window, Our "Normal" state function is to let read-only Register 40101=1 (Let outside controller knows it) So Script content is "ReadReg(40101)=1"

Double click on "Stop? " (Verify) State. Pop out script window, Our "Stop?" state function is to see if holding register 40100 is 0 (Stop command is Holding Register 40100 value with 0). So Script content is "Verify=(WriteReg(40100)=0)" "

Double click on "Stop " (Action) State. Pop out script window, Our "Stop" state function is to let read-only Register 40101=0 (Let outside controller knows it) So Script content is "ReadReg(40101)=0"

Double click on "Fault " State. Pop out script window, Our "Fault" state function is to let read-only Register 40102=1 and read-only Register 40101=0 (Let outside controller knows it) So Script content is

```
ReadReg(40102)=1
ReadReg(40101)=0
```

You can Copy/Paste Existing State by "Arrow" select icon and "Ctrl+C"/"Ctrl+V" . And you can use

"Delete" key to remove some states or connections.

For example , we have 4 devices. and 4 devices are the same state machines. So we let Device 1 state machine window focused. Press "Ctrl+A" to select all elements in Device 1 state machine window, and press "Ctrl+ C" to copy all elements to clipboard. We let Device 2 state machine window focused. Press "Ctrl+ V" to paste all elements in device 1 to Device 2 State machine. We let Device 3 state machine window focused. Press "Ctrl+ V" to paste all elements in device 1 to Device 3 State machine. We let Device 4 state machine window focused. Press "Ctrl+ V" to paste all elements in device 1 to Device 4 State machine. This copy/paste includes Script.

To click menu item "Connect" will connect PC with simulator hardware, And for the first time, it will compile all scripts. See picture below:

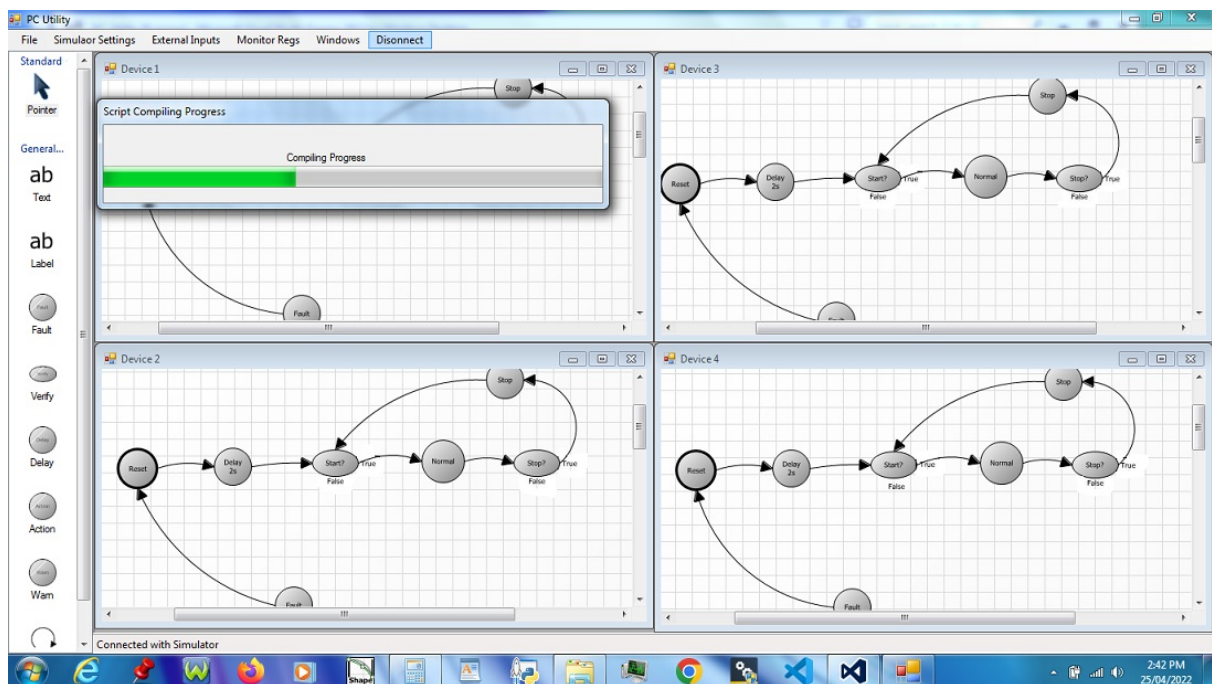


Fig.22 Compiling Scripts

After compiling successfully, it will simulate automatically. Please see picture below:

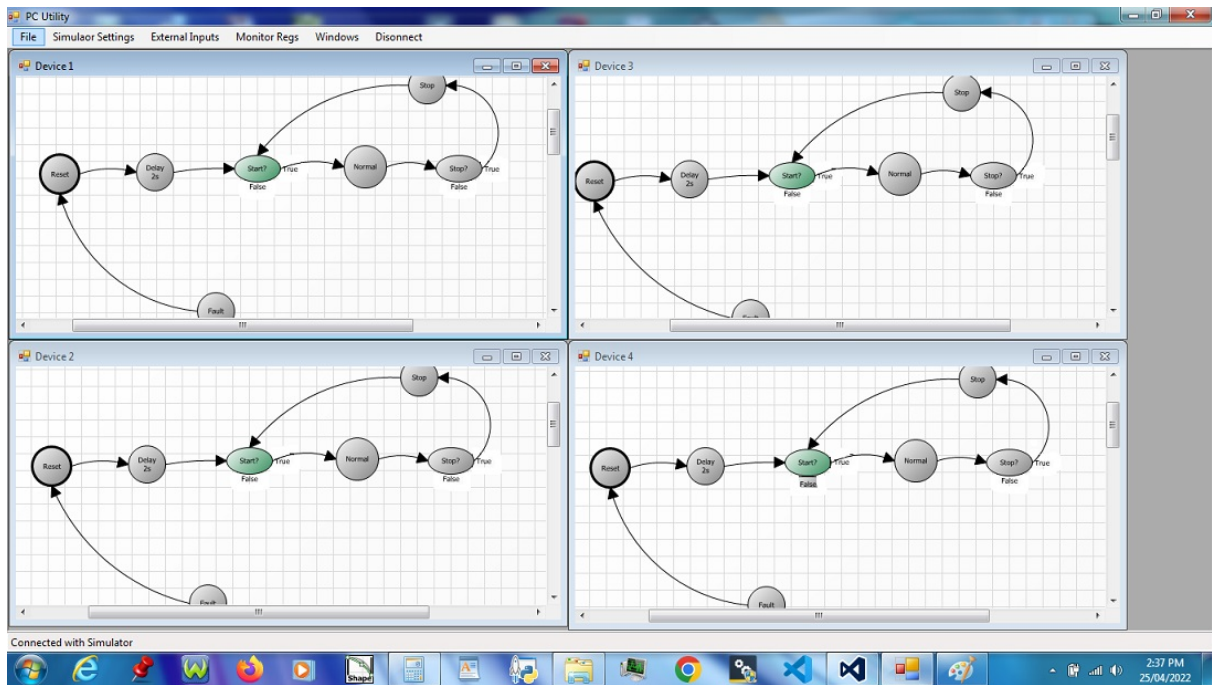


Fig.23 State machine running

The current state is in "Start?" which waits for outside start command. If you click menu item "External Inputs/Fault and Digital Inputs" to open fault inputs , click any fault to be checked in tab Device 1, Device 1 will enter fault State, See picture below:

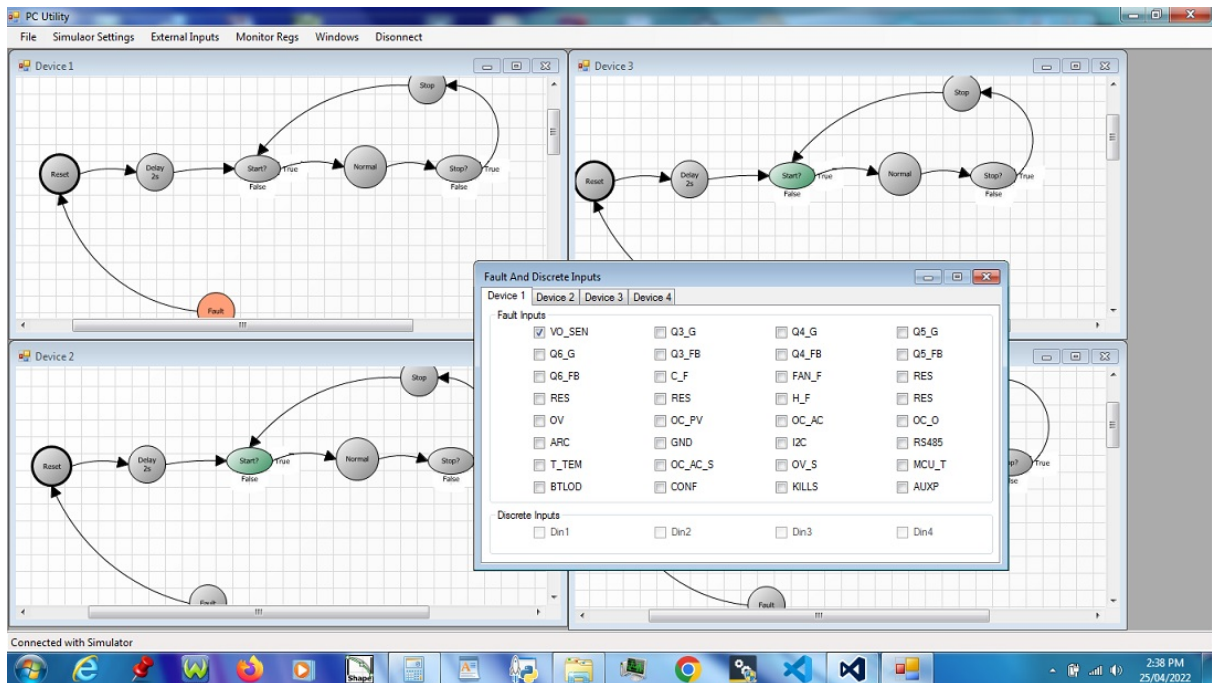


Fig.24 Fault occurs for device 1

In order to see all registers value, please click menu item "Monitor Regs", you will see all registers values as windows below:

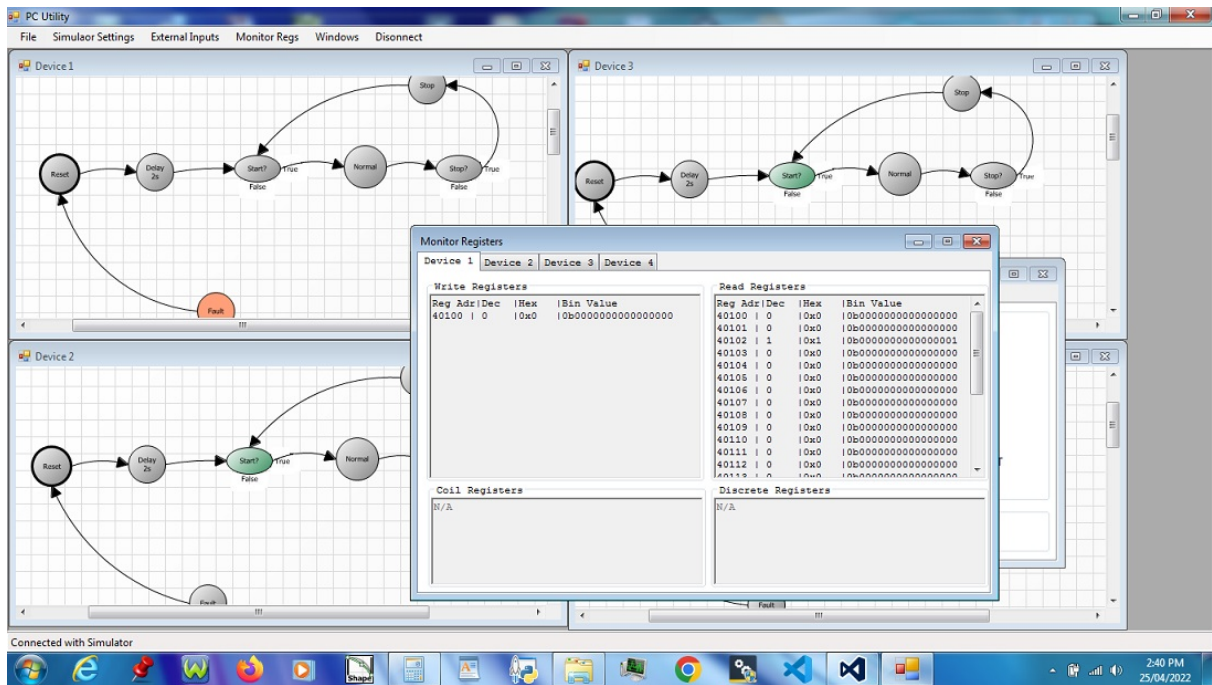


Fig.25 Monitor Registers

You can see Read Register 40102 is 1 for Device 1 (Fault).

At last, you can save your state machine for your team member use. Click item "Disconnect" to stop simulation, and then click menu item "File/Save or /Save As...." to save your state machine in *.nsprj file. Of course, you can open state machine file to save time for drawing state machine and writing script.

For your Simulator, you should have 4 files:

1. *.bcfg file for Bus configuration.
2. *.dcfg file for Device configuration
3. *.nsprj file for state machine
4. *.scr file for script. This file name without suffix is the same as one of *.nsprj. It automatically open or save when you open or save *.nsprj file

So you should copy 4 files above to your co-worker.

Notes: Don't use "Arc connection" connecting One State to itself. If you really want to connect to itself, please use Delay State with delay time=0.

7 State Machine and VBScript Example

Let's take Alecon Solar Converter SPOT Ver6.0 as example.

SPOT is Alecon product. One SPOT has 4 independent channels (Channel A, B, C, D). Each channel has different Modbus ID or device address.

As for purpose of controlling Spot, we only simulate a subset of SPOT registers. That's enough for our purpose.

In our simulator, total 4 devices are simulated for this example. Actually it is one Spot Device.

The 4 devices modbus IDs (Modbus address) are 10, 11, 12, 13.

Spot uses holding registers as read registers,

Please see read register we simulated in the table below:

Table 2 SPOT Read registers

Address	Segment	Name	Type	Description
40100	1	Command	Uint16	Command registers to allow the issuing of commands.
40101		Status	Uint16	Status register to reflect the current status of the DCDC converter.
40102		Diag.	Uint32	Diagnostic flags indicating potential issues or events. Most significant word is first then least significant at address +1.
40104		In Volt.	Uint16	Input DCDC voltage in Volts.
40105		In Curr.	Uint16	Input DCDC current in milliamps.
40106		Out Volt.	Uint16	Output voltage in Volts.
40107		Out Curr.	Uint16	Output current in milliamps.
40108		Sw. Temperature	Uint16	Temperature measured at the transistor in degrees C.
40109		Board Temperature	Uint16	Temperature measured at the MCU in degrees C.
40110		Ground leak Curr.	Uint16	Ground leak current in milliamps.
40111		Sw. Freq.	Uint16	DCDC gate frequency in decahertz.
40112		Aux 12V	Uint16	Auxiliary 12V in millivolts.
40116	2	Temperature Sen.	Uint16	Temperature measured by the humidity sensor in degrees C.
40117		Humidity Sen.	Uint16	Relative humidity as a percentage.

Write register we simulated in the table below:

Table 3 SPOT Write registers

Address	Segment	Name	Type	Description
40100	1	Command	Uint16	Command registers to allow the issuing of commands.

We only simulate 4 commands: value 2 for clear Error code command, value 3 for shutdown command, value 14 for Reset command.

Fault inputs we simulated are in the table below (Fault is the read register 40102 bit number)

Table 4 Fault Inputs

Fault Name	read Register 40102 bit Number	Description
VOUT_SENSOR	Bit 0	output voltage sensor issue
Q3_GateFault	Bit 1	power MOSFET issue
Q4_GateFault	Bit 2	power MOSFET issue
Q5_GateFault	Bit 3	power MOSFET issue
Q6_GateFault	Bit 4	power MOSFET issue
Q3_FeedbackFault	Bit 5	power MOSFET issue
Q4_FeedbackFault	Bit 6	power MOSFET issue
Q5_FeedbackFault	Bit 7	power MOSFET issue
Q6_FeedbackFault	Bit 8	power MOSFET issue
CLOCK_FAULT	Bit 9	MCU clock failure
FAN_FAULT	Bit 10	Fan tachometer not within range
RESERVED	Bit 11	
RESERVED	Bit 12	
RESERVED	Bit 13	
HARD_FAULT	Bit 14	Firmware hard fault detected
RESERVED	Bit 15	
OVP_OUTPUT	Bit 16	over voltage event detected on output measurement
OCP_INPUT_PVDC	Bit 17	over current event detected on DC input measurement
OCP_INPUT_AC	Bit 18	over current event detected on AC measurement (DC/DC converter)
OCP_OUTPUT	Bit 19	over current event detected on output measurement
ARCFault	Bit 20	Arc fault event detected
GROUNDFAULT	Bit 21	ground fault event detected
I2C_ISSUE	Bit 22	internal communication issue on I2C bus detected
RS485_ISSUE	Bit 23	internal communication issue on RS485 bus detected
TRANSISTOR_TEMP	Bit 24	transistor temperature reached limit
OCP_INPUT_AC_SENSOR	Bit 25	over current sensor issue
OVP_SENSOR	Bit 26	over voltage sensor issue
MCU_TEMP	Bit 27	microcontroller temperature reached limit
BOOTLOADER	Bit 28	microcontroller is in bootloader mode
CONFIG_REJECTED	Bit 29	configuration data was rejected
CONFIG_REJECT	Bit 30	physical kill switch is active and power

TED		conversion is disabled
AUX_POWER	Bit 31	auxiliary power supply voltage outside normal ran

Digital Inputs: there is one virtual digital input which simulates shadow (nightfall). If no sun, "Shadow" variable will be true, Solar converter will have no power, and Communication Fault will happen.

This important simulation for 4 channels of SPOT will catch the situation for 4 channels to get power not at the same time.

For Status Register 40101, we only simulate 7 values, please see table below:

Table 5 Status Register

Value	Name	Description
0	NONE	Default value on start, means no communication with the DCDC converter has been established since the modbus server started.
1	INITIALIZING	The DCDC converter is initializing its system components (not producing power).
2	STANDBY	The DCDC converter is ready to produce power but not all run limits are satisfied.
3	STARTUP	The DCDC converter is starting.
4	RUN	The DCDC converter is producing power normally.
5	LIMITING_ON	The DCDC converter is limiting its power production
7	SHUTDOWN	The DCDC converter is in shutdown mode and will not produce power

Please see our Bus Configuration as below:

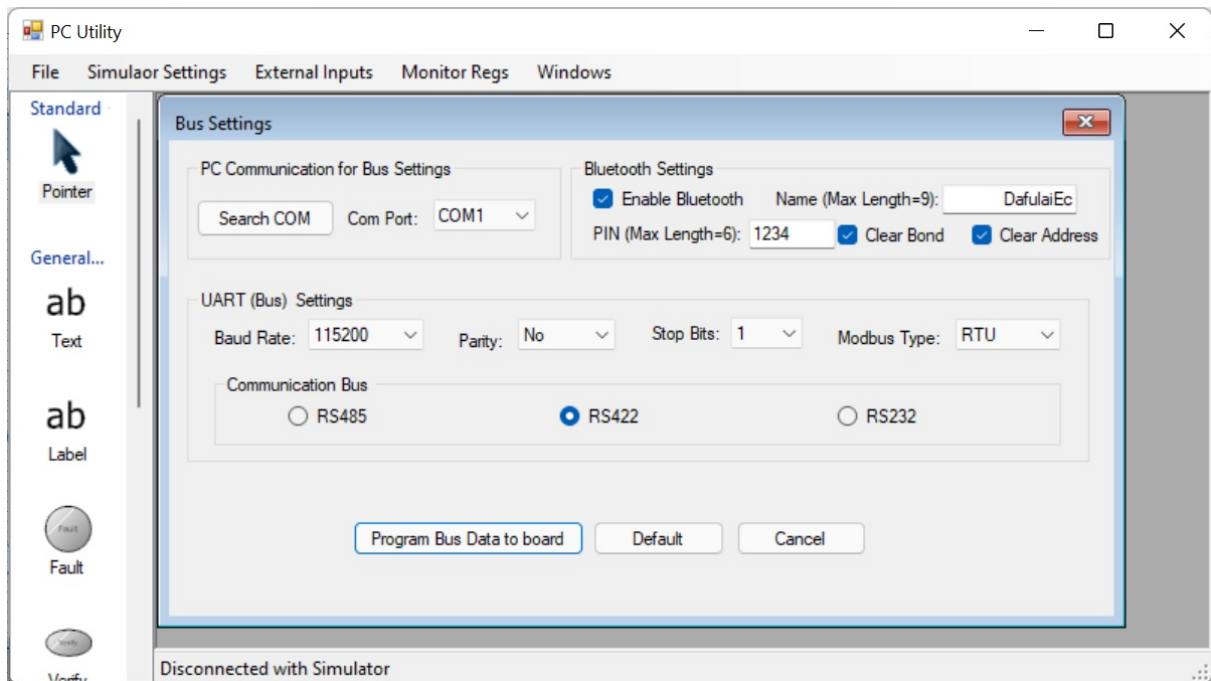


Fig.26 SPOT Bus Config.

It means Baud Rate =115200, No Parity, 1 Stop bit, Modbus RTU, RS422 interface.

For device configuration, Please see screenshot below:

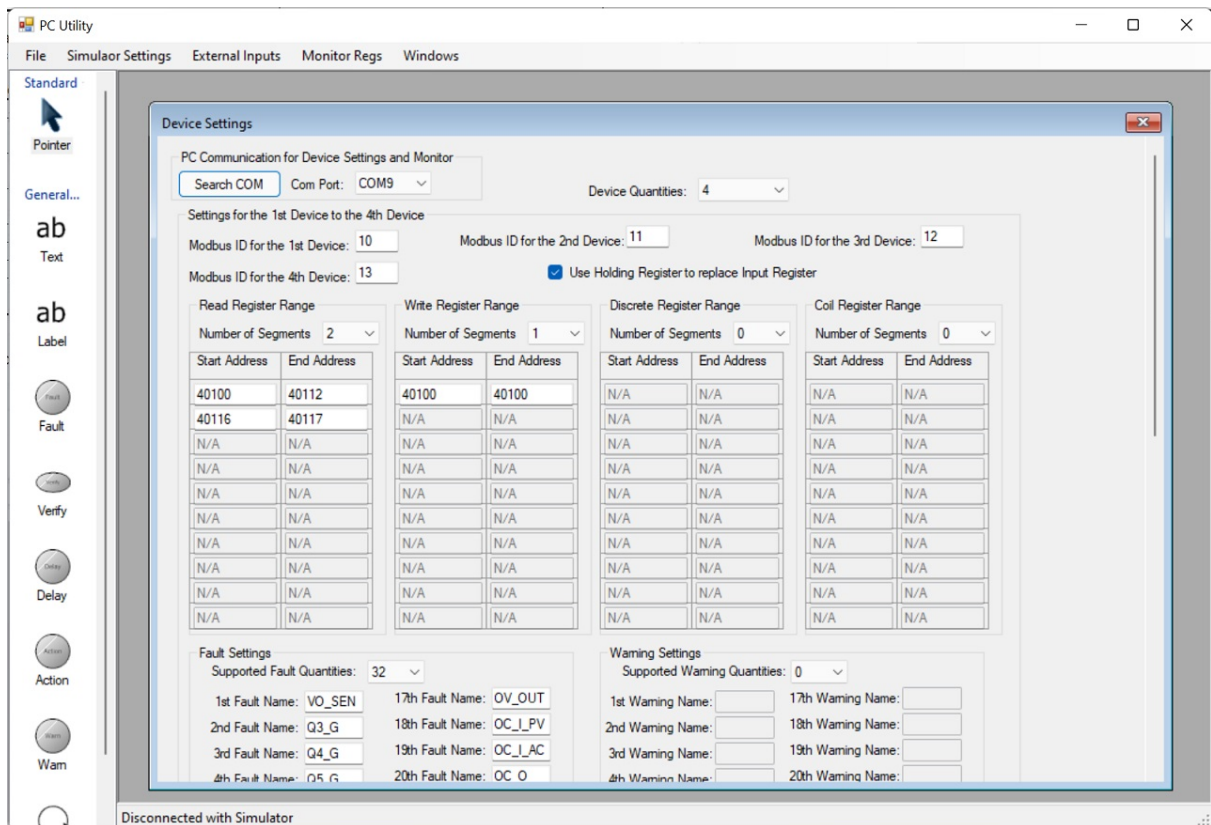


Fig.27 SPOT Device Config

And

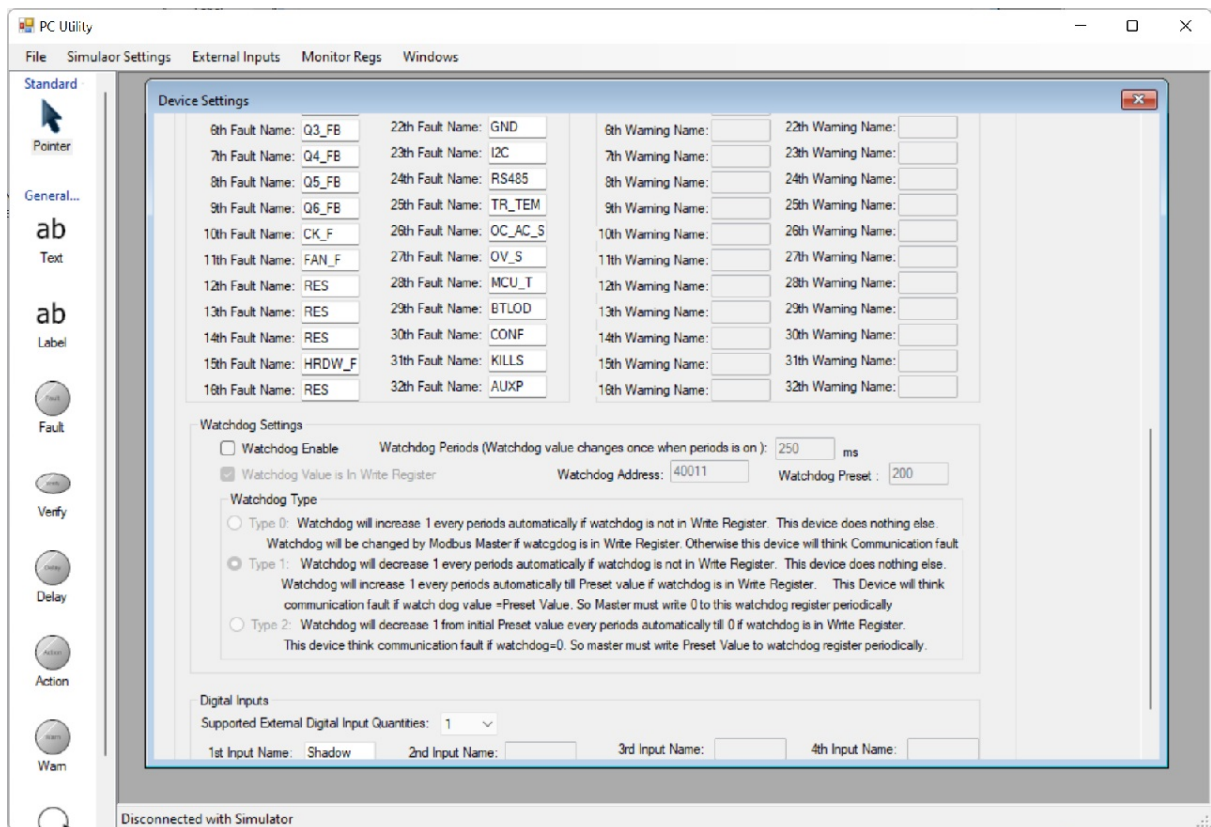


Fig.28 SPOT Device Config

Every Channel (that is device in simulator) has 17 states, Please see screenshot below:

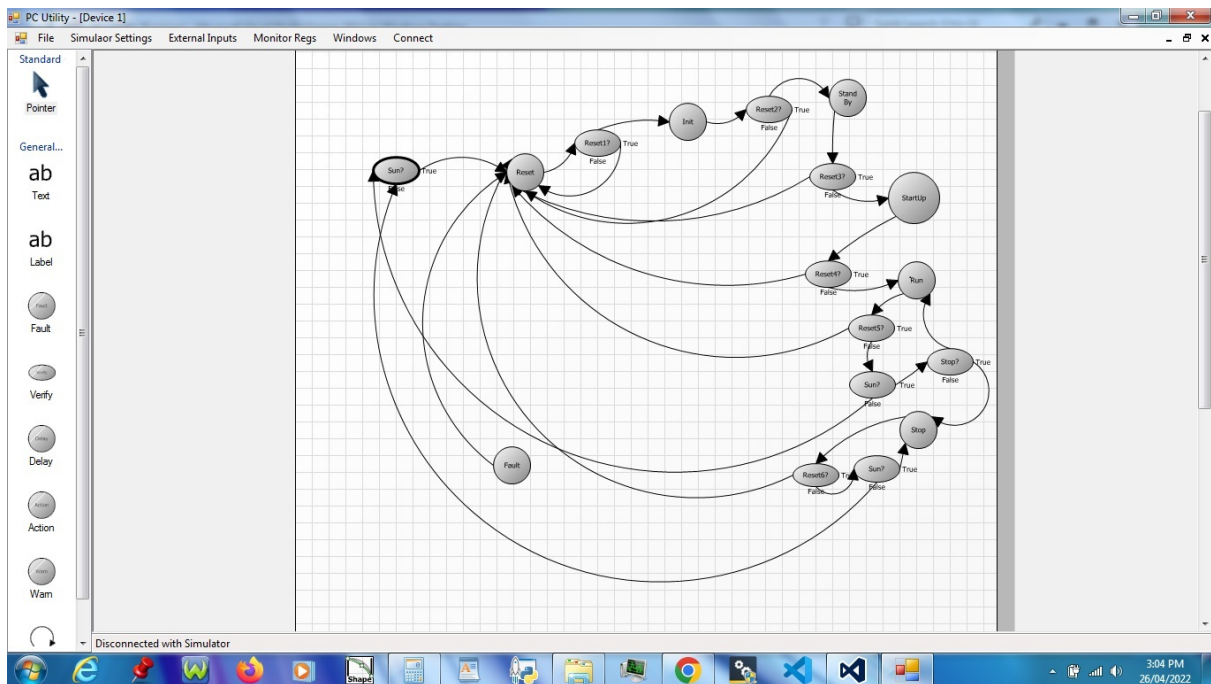


Fig.29 SPOT State Machine

The initial State is "Sun?" state (Verify). It sees whether opposite of shadow variable DigitalInput(0) is true. True means "Sunny", So this channel can go to next State "Reset" (Action). False means "Rainy day or shadow of sun light", this channel will stay in this state.

The "Sun?" state script is (Time out =0 ms):

```
Verify =Not(DigitalInput(0) )
EnableCom(0)=False
```

By external Inputs/Fault and Digital Inputs window, you can simulate rainy/sunny days. From above script, you will know Device 1 (Channel A) no communication from statement "EnableCom(0)=False".

The "Reset" state script is below:

```
Reset=True
EnableCom(0)=False
```

So its behaviour of "Reset State" is that clear all read/write register to zero (Statement: *Reset=True*) and disable communication of channel A (Statement: *EnableCom(0)=False*). If arriving at this state from "Sun? state", you don't need the 2nd statement (Statement: *EnableCom(0)=False*) . However, there are lots of way to arrive at this state, so the 2nd statement is necessary.

After complete "Reset" state, it will enter "Reset1?" (Verify).

The State "Reset1?" script is below (Time out =2000 ms):

```
If WriteReg(40100)=14 Then
    Verify=True
Else
    Verify=False
End If
```

It means that it will return to "Reset" (Action) State if it receives "Reset" command (*WriteReg(40100)=14*), otherwise, it will enter next state "Init" (INITIALIZING) (Action) after 2000ms.

State "Init" script is below:

```
EnableCom(0)=True

ReadReg(40101)=1 'Status is Initialization
ReadReg(40104)=602 'Input Voltage is 601V
ReadReg(40105)=3000 'Input Current is 3000mA
ReadReg(40106)=0 'Output Voltage is 0V
ReadReg(40107)=0 'Output Current is 0mA
ReadReg(40108)=21 'Sw Temperature is 21C
ReadReg(40109)=22 'Brd Temperature is 22C
ReadReg(40110)=0 'Ground Leak current is 0 mA
ReadReg(40111)=20000 'DCDC Gate Frequency
ReadReg(40112)=12000 'Auxiliary 12V

ReadReg(40116)=23 'Temperature measured by the humidity sensor: 23C
ReadReg(40117)=58 'Relative humidity as 58%
```

The 1st line of script (*EnableCom(0)=True*) will enable Channel A communication. The 3rd line (*ReadReg(40101)=1*) will set up Status register. The other lines are to set up read registers value.

After complete "Init" state, it will enter "Reset2?" (Verify). "Reset2? " State is the same as "Reset1?" State except different time out although script is different.

The State "Reset2?" script is below (Time out =2087 ms):

```
Verify=( WriteReg(40100)=14 )
```

Although this script is simpler than one of "Reset1?", the function is exactly the same.

The script means that it will return to "Reset" (Action) State if it receives "Reset" command (*WriteReg(40100)=14*), otherwise, it will enter next state "Standby" (INITIALIZING) (Action) after 2087ms.

State "Standby" script is below:

```
ReadReg(40101)=2 'Status is Standby
```

It will set up Status register.

After complete "Standby" state, it will enter "Reset3?" (Verify). "Reset3? " State is the same as "Reset2?" State except different time out. (time out =3033ms).

So, it will return to "Reset" (Action) State if it receives "Reset" command (*WriteReg(40100)=14*), otherwise, it will enter next state "Startup" (STARTUP) (Action) after 3033 ms.

State "Startup" script is below:

```
ReadReg(40101)=3 'Status is Initialization
```

```
ReadReg(40106)=100 'Output Voltage is 100V
```

```
ReadReg(40107)=20 ' Output Current is 20mA
```

The 1st line of script (*ReadReg(40101)=3*) will set up Status register. The 3rd line and 4th line set up output voltage and current.

After complete "Startup" state, it will enter "Reset4?" (Verify). "Reset4? " State is the same as "Reset2?" State except different time out. (time out =5678ms).

So, it will return to "Reset" (Action) State if it receives "Reset" command (*WriteReg(40100)=14*), otherwise, it will enter next state "Run" (RUN and LIMITING_ON) (Action) after 5678 ms.

State "Run" script is below:

```
static counter As Integer=0
```

```
counter =counter +1
```

```
If counter>1000 Then
```

```
    counter =0
```

```
End If
```

```
If counter <500 Then
```

```
    ReadReg(40101)=4 'Status is Run
```

```
    ReadReg(40106)=750 'Output Voltage is 750V
```

```
    ReadReg(40107)=2300 ' Output Current is 2300mA
```

```
    ReadReg(40108)=48 'Sw Temperature is 48C
```

```

ReadReg(40109)=42 'Brd Temperature is 42C
Else
ReadReg(40101)=5 'Status is LIMITING_ON
ReadReg(40106)=750 'Output Voltage is 750V
ReadReg(40107)=2100 ' Output Current is 2100mA
ReadReg(40108)=43 'Sw Temperature is 43C
ReadReg(40109)=32 'Brd Temperature is 32C
End If

```

This script has more statements than previous ones. It alternates between "RUN and LIMITING_ON". The 1st line to 5th line are for one counter which value is 0 to 1000. The variable counter is "static". So it keeps its value when re-enter this script. The 7th line to the 11th line are to set up Status (RUN) and other parameters values as comment (when counter <500). The 13th line to the 17th line are to set up Status (LIMITING_ON) and other parameters values as comment (when counter >500).

After complete "Run" state, it will enter "Reset5?" (Verify). "Reset5? " State is the same as "Reset2?" State except different time out. (time out =0ms).

So, it will return to "Reset" (Action) State if it receives "Reset" command (*WriteReg(40100)=14*), otherwise, it will enter next state "Sun? " (Verify) immediately.

This state "Sun? " is different from previous one although their names are the same. Any State has unique ID although the same name. Of course, we suggest to use different name for debug (you don't know which state script has problem when compiling error if you use the same name).

This state "Sun? " script is below:

```

Verify =Not(DigitalInput(0))

```

The difference is that we have no statement "*EnableCom(0)=False* " because we still have communication if sun light exists.

If sun light exists, it will be true, and enter next state "Stop? " (Verify), otherwise enter initial "Sun?" State.

The State "Stop?" script is below (Time out =0 ms):

```

Verify=( WriteReg(40100)=3 )

```

It means that it will enter "Stop" (Action) State if it receives "Shutdown" command (*WriteReg(40100)=3*), otherwise, it will enter state "Run" (Action) immediately.

State "Stop" script is below:

```

ReadReg(40101)=7 'Status is Shutdown
ReadReg(40104)=602 'Input Voltage is 601V
ReadReg(40105)=3000 ' Input Current is 3000mA
ReadReg(40106)=0 'Output Voltage is 0V
ReadReg(40107)=0 ' Output Current is 0mA
ReadReg(40108)=31 'Sw Temperature is 31C
ReadReg(40109)=28 'Brd Temperature is 28C
ReadReg(40110)=0 'Ground Leak current is 0 mA
ReadReg(40111)=20000 'DCDC Gate Frequency
ReadReg(40112)=12000 'Auxiliary 12V

```

The 1st line is to set up Status to "Shutdown".The other lines are to set up read registers value as comment

After complete "Stop" state, it will enter "Reset6?" (Verify). "Reset6? " State is the same as "Reset2?" State except different time out. (time out =0ms).

So, it will return to "Reset" (Action) State if it receives "Reset" command (*WriteReg(40100)=14*), otherwise, it will enter next state "Sun? " (Verify) immediately.

This state "Sun? " is different from initial one although their names are the same. But it is the same as above one with the same name.

If sun light exists, it will be true, and enter next state "Stop " (Action), otherwise enter initial "Sun?" State.

The state "Fault" is different. It will enter automatically from any state if any *FaultInput(i)* variable is true.

The state "Fault" script is below:

```

ReadReg(40102)=0
For i As Integer =0 to 31
    If FaultInput(i)=True Then
        ReadReg(40102)=ReadReg(40102)+(1<<i)
    End If
Next
If WriteReg(40100)=2 Then
    For i As Integer =0 to 31
        FaultInput(i)=False
    Next
    ReadReg(40102)=0
End If

```

Line 1 to 6 are to set up diagnosis flag for register 40102. Line 7 to 12 are to detect "Clear Error" command (*WriteReg(40100)=2*) . If it receives "Clear Error" command, it will clear *FaultInput(i)* (i=0 to 31).

Now all scripts are done.

We can click menu "Connect" to simulate SPOT. Please see screenshot:

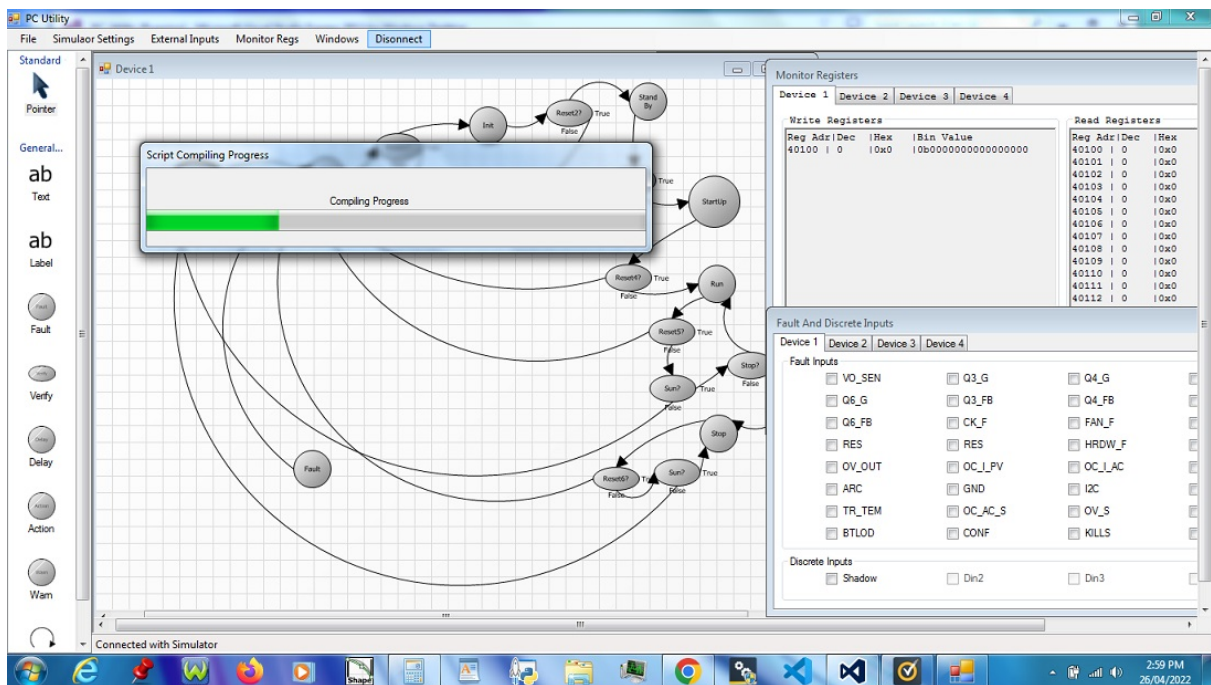


Fig.30 SPOT Compiling

For the first time after you open state machine, it will compile script (Compiling result is in RAM, not in file, so it will compile scripts for the first time.)

After compiling successfully, it will simulate. Please see screenshot below:

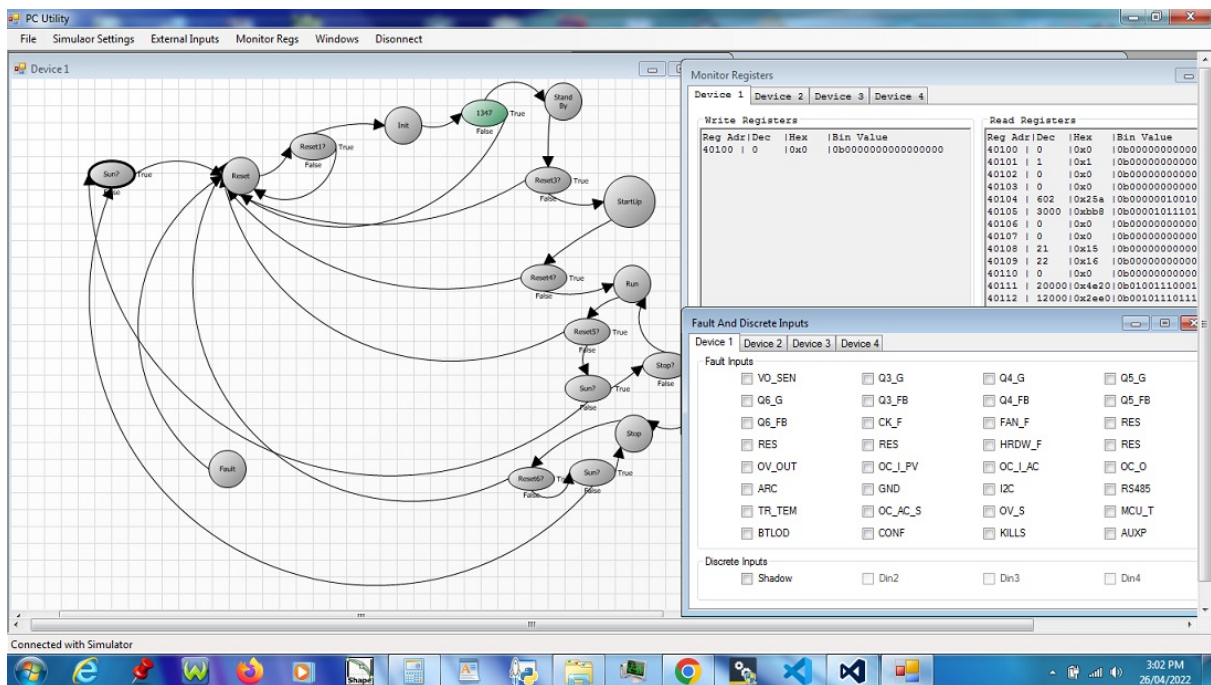


Fig.31 SPOT Running

You can use "Fault And Discrete Inputs" window to simulate different faults, and sunny or rainy day (Different Tab is for different channel). You can use "Monitor Registers" window to view all registers

value in real time.

If we use Device 5 for "GARD", we can simulate one SPOT and one GARD together. And GARD only gets power when any PV of all Channels of SPOT gets sun light. We will set 4 virtual digital inputs for each PV's sun light. And remove SPOT virtual digital input. When any of 4 virtual digital inputs is "True", "GARD" will power on, and "GARD " communication will be normal (EnableCom(4)=True). When "GARD" receives "Channel i ON" command, it will make EnableCom(i)= true if DigitalInput(i) of GARD is true (i=0,1,2,3). We don't configure GARD and script. Customers can set up by them self.

If you are interested in the SPOT simulator, 4 files (Alencon_spot.bcfg, Alencon_spot.dcfg, Alencon_spot.nsprj, Alencon_spot.scr) for simulation are in the "Install folder/Example/Alencon_SPOT"

And the final State machines are a little different from one we explained above because SPOT actually initial statable state is "shut down" instead of "normal run".

In Folder: " Install folder/Example", there are other simulators for EnergPort Battery (SunSpec protocol) and Socomec Inverter (SOCOMECEnergyStorageModbusProtocol_REV_10). We just simulate subset of entire registers for controller purpose.

Please open 4 files in " Install folder/Example/EnergPort_Battery" for EnergPort Battery, open 4 files in " Install folder/Example/SOCOMECE_Inverter" for Socomec Inverter. Actually, Socomec Inverter is Storage System, we just take battery away, and add DC power supply to replace battery, it becomes Inverter.

For other Storage Batteries, we can write scripts for them. Please tell us your device's Modbus Map, and control command, and device behaviour. This service is not free. Please contact us for quotation.

If you don't use MATLAB or SIMULINK, don't need to read remaining content.

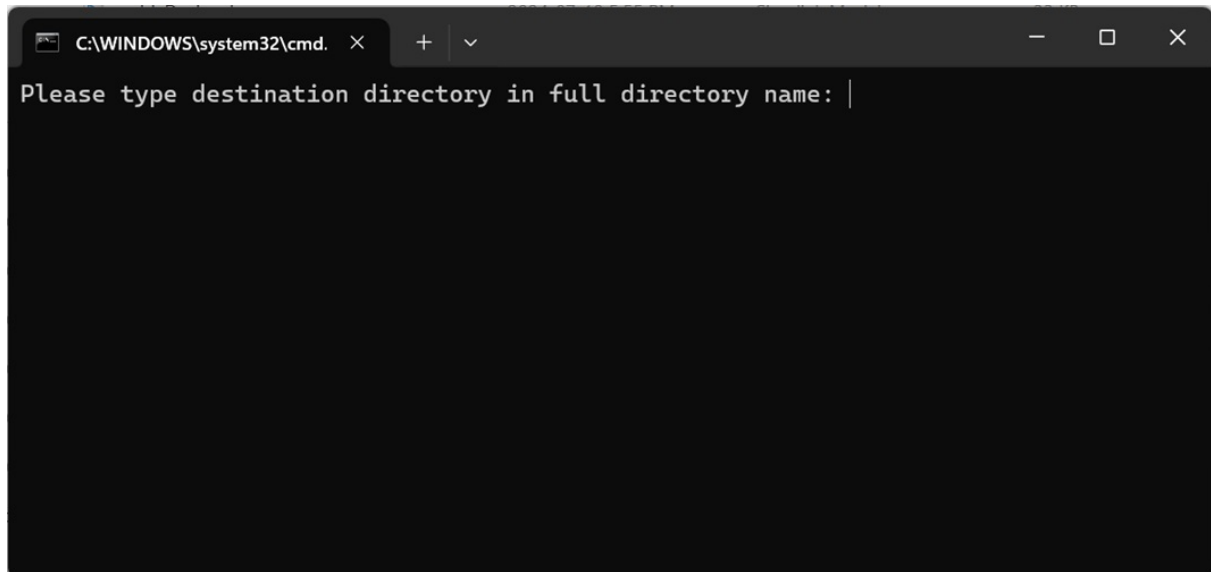
8 How to install Modbus Server Simulator Library in Matlab/Simulink?

Please follow steps below:

In Windows platform

- **Step1** Download Modbus Server Simulator library for Matlab/Simulink from clicking [ModbusServer4Mat.zip](#)

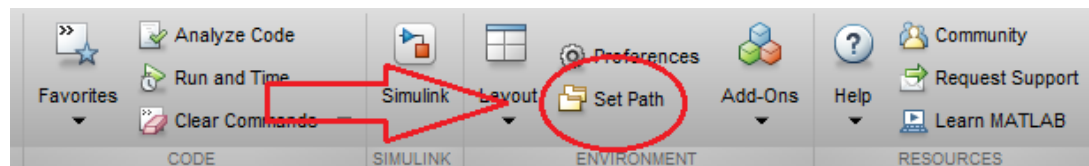
- **Step2** unzip ModbusServer4Mat.zip to any directory.
- **Step3** double click on setup.bat. It will display window below:



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).

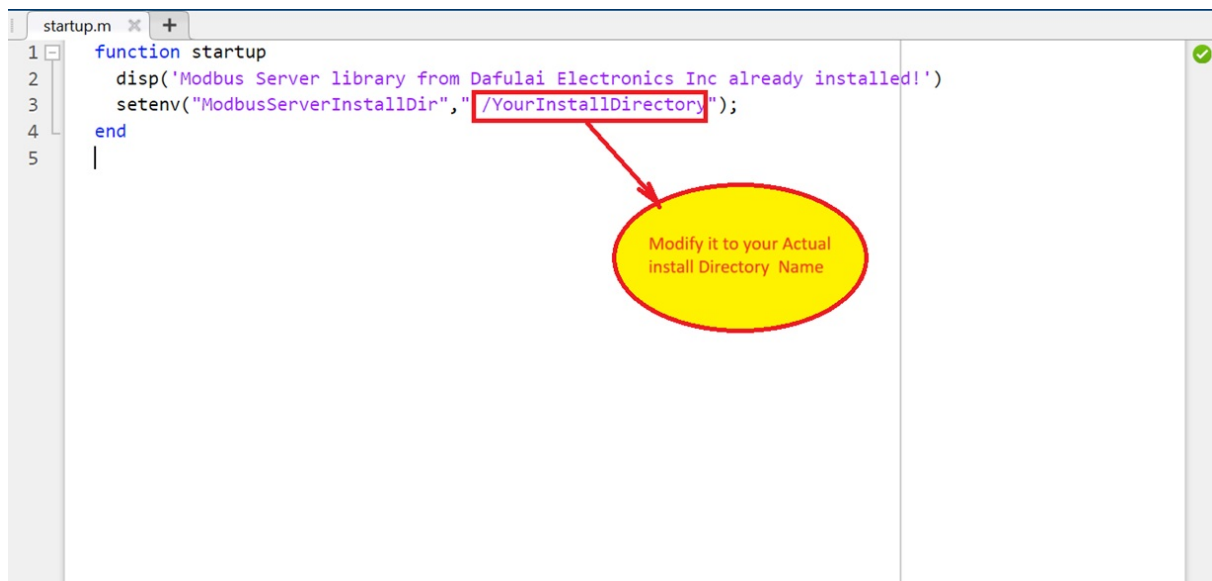
- **Step4** Set Matlab Path contains your destination directory of your Modbus Server Simulator library.

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:



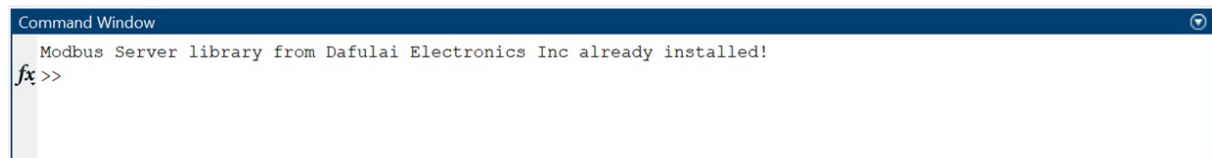
In non-Windows Platform.

- **Step1** Download Modbus Server Simulator library for Matlab/Simulink from clicking [ModbusServer4Mat.zip](#)
- **Step2** unzip ModbusServer4Mat.zip to directory you want to install.
- **Step3** Modify startup.m file. See screenshot below:



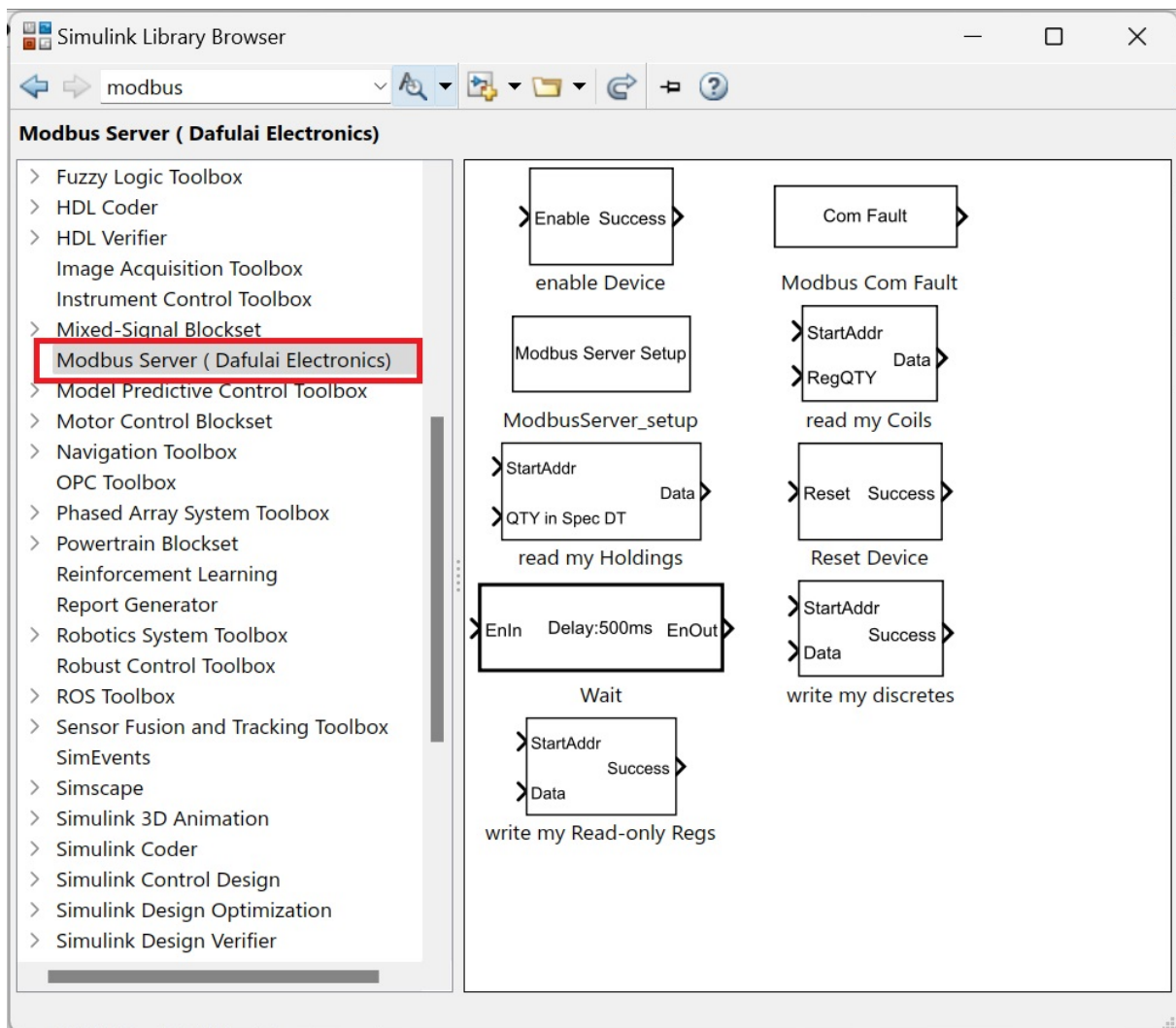
- **Step4** Set Matlab Path contains your directory you unzip. Now it is the same method as Step4 in windows platform.

After you finish Modbus Server Simulator 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 Server Simulator library install in your computer successfully.

In your Simulink Library browser, you will see our Modbus Server Simulator library as shown as the following screenshot:



For the first-time use the simulator, you must run PC_Utility software as Section 6.2 , Please click Menu item: " Simulator Settings / Bus Settings" to set up baud rate and physical interface (RS232/ RS485/RS422) and bluetooth. Of cause, if you want to change theses settings, you can run PC_Utility software again.

9 MATLAB class

There are 2 different Matlab classes, one is "ModbusRTUASCiiServer" for Modbus RTU/ASCII server. The other is "ModbusTCPServer" for Modbus TCP server.

9.1 ModbusRTUASCiiServer class

ModbusRTUASCiiServer

Connection to Modbus RTU/ASCII Server simulator
Since R2019b

Description

A ModbusRTUASCiiServer object represents as many as 5 Modbus RTU/ASCII nodes for

communication with the Modbus Master in the same Modbus network. After creating the object, use dot notation to call methods.

Creation

Syntax

```
m = ModbusRTUASCiiServer(portName)
m = ModbusRTUASCiiServer(portName, Name, Value)
```

Description

`m = ModbusRTUASCiiServer(portName)` connects to the Modbus network by USB or Bluetooth serial port specified by `portName`. It will use default Modbus parameters for connecting. example

`m = ModbusRTUASCiiServer(portName, Name, Value)` connects to the Modbus network by USB or Bluetooth serial port specified by `portName` and sets additional private properties using optional name-value pair arguments.

Input Arguments

`portName` -- USB or Bluetooth serial port name which is used by "Modbus slave simulator" hardware.
character vector | string scalar

USB or Bluetooth serial port name, specified as a character vector or string scalar. This is mandatory argument. Use `serialportlist` to get a list of connected ports.

For example, `m=ModbusRTUASCiiServer("COM1")` will use "COM1" in windows. It may be `"/dev/ttyUSB0"` under Linux

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 `portName`, 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=ModbusRTUASCiiServer("COM1","ServerID",[3 4],"SupportedHoldingRegs1to4Segs",{[1 10]},"SupportedInputRegs1to4Segs",{[22 25]})`; It will set Server ID =3 and 4 (Device QTY=2). These 2 server nodes are the same register structure. Its holding register range is address 1 to 10, and Input register range is address 22 to 25.

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

- **ServerID**

Value is scalar/vector double. vector element qty denotes how many devices we support. Maximum Qty=5, and element value is 1 to 247. Default=1

- **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"

- **SupportedHoldingRegs1to4Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous Holding register segments to support for Device 1 to 4.

Elements QTY of cell denotes supported read-write holding register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {[1 10], [15 20]} denotes it supports 2 read-write holding register segments, and the 1st segment address is from 1 to 10, the 2nd segment address is from 15 to 20. Default={[1 350]}

- **SupportedInputRegs1to4Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous read-only input register segments to support for Device 1 to 4.

Elements QTY of cell denotes supported read-only input register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {[31 40], [45 120]} denotes it supports 2 read-only input register segments, and the 1st segment address is from 31 to 40, the 2nd segment address is from 45 to 120. Default={[1 350]}

- **HoldingReplaceInputRegs1to4**

Value is logical true or false. true means we will use holding register to replace input registers for read-only registers when device number is 1 to 4. Default=false

- **SupportDigitalInOut1to4**

Value is logical true or false. true means we support Discrete/Coil registers for Device 1 to 4. Default=true

- **SupportedCoils1to4Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous coil register segments to support for Device 1 to 4.

Elements QTY of cell denotes supported read-write coil register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {[1 10], [15 20]} denotes it supports 2 read-write coil register segments, and the 1st segment address is from 1 to 10, the 2nd segment address is from 15 to 20. Default={[1 500]}

- **SupportedDiscreteRegs1to4Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous read-only discrete register segments to support for Device 1 to 4.

Elements QTY of cell denotes supported read-only discrete register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {[31

40], [45 120]] denotes it supports 2 read-only discrete register segments, and the 1st segment address is from 31 to 40, the 2nd segment address is from 45 to 120. Default={{1 500}}

- **SupportedHoldingRegs5Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous Holding register segments to support for Device 5.

Elements QTY of cell denotes supported read-write holding register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {{1 10], [15 20]] denotes it supports 2 read-write holding register segments, and the 1st segment address is from 1 to 10, the 2nd segment address is from 15 to 20. Default={{1 350}}

- **SupportedInputRegs5Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous read-only input register segments to support for Device 5.

Elements QTY of cell denotes supported read-only input register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {{31 40], [45 120]] denotes it supports 2 read-only input register segments, and the 1st segment address is from 31 to 40, the 2nd segment address is from 45 to 120. Default={{1 350}}

- **HoldingReplaceInputRegs5**

Value is logical true or false. true means we will use holding register to replace input registers for read-only registers when device number is 5. Default=false

- **SupportDigitalInOut5**

Value is logical true or false. true means we support Discrete/Coil registers for Device 5. Default=true

- **SupportedCoils5Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous coil register segments to support for Device 5.

Elements QTY of cell denotes supported read-write coil register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {{1 10], [15 20]] denotes it supports 2 read-write coil register segments, and the 1st segment address is from 1 to 10, the 2nd segment address is from 15 to 20. Default={{1 500}}

- **SupportedDiscreteRegs5Segs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous read-only discrete register segments to support for Device 5.

Elements QTY of cell denotes supported read-only discrete register segment Qty. Maximum segment QTY=10. Row vector denotes "Start address" and "End address". For example, {{31 40], [45 120]] denotes it supports 2 read-only discrete register segments, and the 1st segment address is from 31 to 40, the 2nd segment address is from 45 to 120. Default={{1 500}}

- **WatchdogEnable1to4**

Value is logical true or false. true means we use hardware watchdog for Device 1 to 4.

Default=false

- **WatchdogUseInputReg1to4**

Value is logical true or false. true means we use read-only word register as watchdog for device 1 to 4. Default=false

- **WatchdogType1to4**

Value is scalar number which can be 0, 1, 2 for read-write address or 0, 1 for read-only address. Default=0. It is Device 1 to 4's watchdog type.

Watchdog has 3 different types for read-write holding regs Watchdog.

Type 0: Watchdog Value will keep the value from outside master writing. In normal situation, Master must write different value to Watchdog holding regs periodically.

Type 1: Watchdog initial value is 0. Watchdog is free running up-counter each Watchdog period. When it arrives at Preset value, it will keep it. So if no communication between Master and slave, it will arrive at Preset value.

And Modbus server can know it has problem for communication. In normal situation, Master must write Watchdog holding regs to value which is much smaller than Preset value.

Type 2: Watchdog initial value is Preset Value. Watchdog is free running down-counter each Watchdog period. When it arrives at 0, it will keep 0.

So if no communication between Master and slave, it will arrive at 0 value. And Modbus server can know it has problem for communication. In normal situation, Master must write Watchdog holding regs to value which is much bigger than 0.

Watchdog has 2 different types for read-only input regs Watchdog.

Type 0: Watchdog initial value is 0. Watchdog is free running up-counter each Watchdog period. Overflow will return to zero

Type 1: Watchdog initial value is 0. Watchdog is free running down-counter each Watchdog period. underflow will return to 0xFFFF

- **WatchdogPeriod1to4**

Value is scalar double which can be 1 to 65535. Default=500. It is Device 1 to 4's watchdog periods in ms.

- **WatchdogPreset1to4**

Value is scalar double which can be 1 to 65535. Default=600. It is Device 1 to 4's watchdog Preset Value.

- **WatchdogAddr1to4**

Value is scalar double. Default=2. It is Device 1 to 4's Watchdog 1-based address.

- **WatchdogEnable5**

Value is logical true or false. true means we use hardware watchdog for Device 5.

Default=false

- **WatchdogUseInputReg5**

Value is logical true or false. true means we use read-only word register as watchdog for device 5. Default=false

- **WatchdogType5**

Value is scalar number which can be 0, 1, 2 for read-write address or 0,1 for read-only address. Default=0. It is Device 5's watchdog type.

Watchdog has 3 different types for read-write holding regs Watchdog.

Type 0: Watchdog Value will keep the value from outside master writing. In normal situation, Master must write different value to Watchdog holding regs periodically.

Type 1: Watchdog initial value is 0. Watchdog is free running up-counter each Watchdog period. When it arrives at Preset value, it will keep it. So if no communication between Master and slave, it will arrive at Preset value.

And Modbus server can know it has problem for communication. In normal situation, Master must write Watchdog holding regs to value which is much smaller than Preset value.

Type 2: Watchdog initial value is Preset Value. Watchdog is free running down-counter each Watchdog period. When it arrives at 0, it will keep 0.

So if no communication between Master and slave, it will arrive at 0 value. And Modbus server can know it has problem for communication. In normal situation, Master must write Watchdog holding regs to value which is much bigger than 0.

Watchdog has 2 different types for read-only input regs Watchdog.

Type 0: Watchdog initial value is 0. Watchdog is free running up-counter each Watchdog period. Overflow will return to zero

Type 1: Watchdog initial value is 0. Watchdog is free running down-counter each Watchdog period. underflow will return to 0xFFFF

- **WatchdogPeriod5**

Value is scalar double which can be 1 to 65535. Default=500. It is Device 5's watchdog periods in ms.

- **WatchdogPreset5**

Value is scalar double which can be 1 to 65535. Default=600. It is Device 5's watchdog Preset Value.

- **WatchdogAddr5**

Value is scalar double. Default=2. It is Device 5's Watchdog 1-based address.

Examples

In general, you only need to set up USB or Bluetooth serial port, the default behaviour is disable watchdogs and one Server with Server ID=1. And the holding register address is from 1 to 350, input register address is from 1 to 350, coil register address is from 1 to 500, discrete register address is from 1 to 500. The following statement is usually used in general project:

```
m=ModbusRTUASCiiServer("COM2");
```

In the following example, we support 2 Modbus RTU/ASCII server with server ID=3 and 4, Read-write word register address ranges are 1 to 10 and 50 to 60, Read-only word register address ranges is 22 to 25. We use holding register to replace read-only input register. The coil bit register address ranges is 1 to 15. The discrete bit register address ranges is 2 to 8. We don't support watchdog.

```
m=ModbusRTUASCiiServer("COM5","ServerID",[3 4],"SupportedHoldingRegs1to4Segs",{[1
10],[50 60]}, "SupportedInputRegs1to4Segs", {[22 25]}, ...
"SupportedCoils1to4Segs", {[1,15]}, "SupportedDiscreteRegs1to4Segs",
{[2,8]}, "HoldingReplaceInputRegs1to4", true);
```

You may ask questions about how to decide baud rate and RTU or ASCII in above examples. This is decided by running PC_Utility software to set up Bus settings.

Properties

- **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". This is public Property, so you can change it in the run-time by assignment

Object Functions (Or Methods)

read	read Server registers values
write	write Server read-only registers or discrete bit registers.
resetDevice	Clear all register values Device supports to zero.
enableDevice	Enable Device Modbus communication
geteCOMStatus	get DeviceNo communication state

Events

None

All methods details

read
read Server registers values
Since R2019b

Syntax

```
[moddata,Error] = read(obj,target,address)
[moddata,Error] = read(obj,target,address,count)
[moddata,Error] = read(obj,target,address,count,precision)
[moddata,Error] = read(obj,target,address,count,DeviceNo)
[moddata,Error] = read(obj,target,address,count,precision,DeviceNo)
```

Description

This function will get my Server register values from one of four target addressable areas: Coils, Inputs, Holding Registers, or Input Registers. Usually we only need to read Coils and Holding registers because Inputs (bits) and Input Registers (words) values are set by ours. We should know their values.

Input Arguments

- **obj** — ModbusRTUASCIiServer 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.
- **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.

- **DeviceNo** — 1 to 5. It Specifies which device we read from (Because one hardware can simulate max 5 devices). Optional, default is 1

Output Arguments

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

Logical value. True means fail to read.

Examples

Firstly, we set up a Modbus RTU/ASCII Server object with COM5 and 2 Servers, one server ID is 3, the other server ID is 4. They support Read-write holding registers addressing from 1 to 100. They support Read-only input registers addressing from 22 to 25. They support coil registers addressing from 1 to 15. They support discrete registers addressing from 2 to 8. Word order is big-endian.

```
m=ModbusRTUASCIIServer("COM5","ServerID",[3 4],"SupportedHoldingRegs1to4Segs",{[1
100]}, "SupportedInputRegs1to4Segs", {[22 25]}, ...
"SupportedCoils1to4Segs", {[1,15]}, "SupportedDiscreteRegs1to4Segs", {[2,8]});
```

% Read 3 coil values starting at address 5 from Device1

```
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 Device 2.*

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

write

write Server read-only registers or discrete bit registers.
Since R2019b

Syntax

```
Error=write(obj,target,address,values)
Error=write(obj,target,address,values,DeviceNo)
Error=write(obj,target,address,values,precision)
Error=write(obj,target,address,values,precision,DeviceNo)
```

Description

This function will perform a write operation to one of two simulated target addressable areas: inputs (Discrete) or input Registers (read-only word registers).

Input Arguments

- **obj** — ModbusRTUASCIIServer object. This argument is mandatory.
- **target** — Specifies the area to write. The valid choices are 'inputs' (discrete bit) and 'inputregs' (read-only words). This argument is mandatory.
- **address** — The starting address to write to. This is 1-based address without 4x/3x... prefix. This argument is mandatory.
- **values** — Array of values to write. For target 'inputs' valid values are false and true. For target 'inputregs' valid values must be in the range of the specified 'precision'. This argument is

mandatory.

- DeviceNo — 1 to 5. It Specifies which device we write to (Because one hardware can simulate max 5 devices). 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

- Error — failure for writing.
Logical value. True means fail to write.

Examples

Firstly, we set up a Modbus RTU/ASCII Server object with COM5 and 2 Servers, one server ID is 3, the other server ID is 4. They support Read-write holding registers addressing from 1 to 100. They support Read-only input registers addressing from 22 to 25. They support coil registers addressing from 1 to 15. They support discrete registers addressing from 2 to 8. Word order is big-endian.

```
m=ModbusRTUASCIIServer("COM5","ServerID",[3 4],"SupportedHoldingRegs1to4Segs",{[1 100]}, "SupportedInputRegs1to4Segs", {[22 25]}, ...  
    "SupportedCoils1to4Segs", {[1, 15]}, "SupportedDiscreteRegs1to4Segs", {[2, 8]});
```

% set the input word registers at address 22 device 1 to the value 2000

```
Error = write(m,'inputregs',22,2000,1);
```

resetDevice

Clear all register values Device supports to zero.
Since R2019b

Syntax

Error=resetDevice(obj, DeviceNo)

Description

This function will put all registers simulator DeviceNo supports to zero.

Input Arguments

- obj — ModbusRTUASCIIServer object. This argument is mandatory.
- DeviceNo — 1 to 5. It Specifies which device we want to reset (Because one hardware can simulate max 5 devices). This argument is mandatory.

Output Arguments

- Error — failure for reset.
Logical value. True means fail to reset.

enableDevice

Enable Device Modbus communication
Since R2019b

Syntax

Error=enableDevice(obj, enabled)

Description

This function will enable/disable some simulated devices

Input Arguments

- obj — ModbusRTUASCiiServer object. This argument is mandatory.
- enabled — logical scalar or vector. The n-th element is true means we enable Device n modbus server. The n-th element is false means we disable Device n modbus server. n cannot be over Device qty we support (ServerID vector's element QTY)

Output Arguments

- Error — failure for enable/disable.
Logical value. True means fail to enable/disable.

Note: If we never call this enableDevice method, all devices are enabled.

geteCOMStatus

get DeviceNo communication state.
Since R2019b

Syntax

ComOK=geteCOMStatus(obj, DeviceNo)

Description

This function will get DeviceNo communication state (ok or fault) when DeviceNo supports read-write holding regs Watchdog and Watchdog type is type 1 or type 2.

Input Arguments

- obj — ModbusRTUASciiServer object. This argument is mandatory.
- DeviceNo — 1 to 5. It Specifies which device communication state we want to know. This argument is mandatory.

Output Arguments

- ComOK — Communication OK between DeviceNo and Master node.
Logical value. True means Communication OK between DeviceNo and Master node.

Note: If server does not support watchdog, or watchdog is not type1/type2 and watchdog is not in read-write holding register, then this method will create exception. You can use "Try catch" structure to know this situation.

9.2 ModbusTCPServer class

ModbusTCPServer

Connection to Modbus TCP Server simulator
Since R2019b

Description

A ModbusTCPServer object represents one Modbus TCP Server node for communication with the Modbus TCP Master. After creating the object, use dot notation to call methods. We only support one Modbus TCP Master. However if you use port+1 as TCP Port, we can support another Modbus TCP Master. For example, default port=502, we can support 2 Modbus TCP masters with port =502 and 503.

Creation

Syntax

```
m = ModbusTCPServer()
m = ModbusTCPServer(port)
m = ModbusTCPServer(port, Name, Value)
m = ModbusTCPServer( Name, Value)
```

Description

Example
m=ModbusTCPServer() or m = ModbusTCPServer(port) object connects to the Modbus TCP network by USB or Bluetooth serial port specified by "COM" (default "COM1"). It will use default Modbus parameters for connecting.

m = ModbusTCPServer(port, Name, Value) or m = ModbusTCPServer(Name, Value) connects to the Modbus TCP network by USB or Bluetooth serial port specified by "COM" (default "COM1"). And sets additional private properties using optional name-value pair arguments.

Input Arguments

port -- TCP port number. Default=502

Value is scalar double. We provide 2 ports for TCP Server. One is port, the other is port+1.

Each port can only connect one Modbus TCP Master. So we can connect 2 Modbus TCP Masters

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 port or first string/char array if no port parameter. 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=ModbusRTUASciiServer("COM",

"COM3","ServerID",3,"SupportedHoldingRegsSegs",{[1 10]}, "SupportedInputRegsSegs", {[22 25]}); It will set Server ID =3. Its holding register range is address 1 to 10, and Input register range is address 22 to 25.

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

- **ServerID**

Value is scalar double: 1 to 247. Default=1

- **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"

- **COM**

Value is scalar string or char array. It denote our Serial Port which is used in Modbus Slave Simulator USB or Bluetooth. Only when correct hardware used, we can use Modbus TCP server.

- **IgnoreNodeID**

Value is logical scalar. True means Server will respond to Master if IP address and port number are correct no matter what Server ID is. Default = false

- **SupportedHoldingRegsSegs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous Holding register segments to support.

Row vector denotes "Start address" and "End address". For example, {[1 10], [15 20]} denotes it supports 2 read-write holding register segments, and the 1st segment address is from 1 to 10, the 2nd segment address is from 15 to 20. Compared with Modbus RTU/ASCII Server, we have no limit for segment QTY. Default={[1 65536]}

- **SupportedInputRegsSegs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous read-only input register segments to support.

Row vector denotes "Start address" and "End address". For example, {[31 40], [45 120]} denotes it supports 2 read-only input register segments, and the 1st segment address is from 31 to 40, the 2nd segment address is from 45 to 120. Default={[1 65536]}

- **HoldingReplaceInputRegs**

Value is logical true or false. true means we will use holding register to replace input registers for read-only registers. Default=false

- **SupportDigitalInOut**

Value is logical true or false. true means we support Discrete/Coil registers. Default=true

- **SupportedCoilsSegs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous coil register segments to support for Device 1 to 4.

Row vector denotes "Start address" and "End address". For example, {[1 10], [15 20]} denotes it supports 2 read-write coil register segments, and the 1st segment address is from 1 to 10, the 2nd segment address is from 15 to 20. Default={[1 65536]}

- **SupportedDiscreteRegsSegs**

Value is cell, cell element is 2 elements' row vector. It tells system how many continuous read-only discrete register segments to support.

Row vector denotes "Start address" and "End address". For example, {[31 40], [45 120]} denotes it supports 2 read-only discrete register segments, and the 1st segment address is from 31 to 40, the 2nd segment address is from 45 to 120. Default={[1 65536]}

Examples

In general, you only need to set up USB or Bluetooth serial port, the default behaviour is Server ID=1. And the holding register address is from 1 to 65536, input register address is from 1 to 65536, coil register address is from 1 to 65536, discrete register address is from 1 to 65536, TCP port=502. The following statement is usually used in general project:

```
m=ModbusTCPServer("COM", "COM5");
```

In the following example, we support Modbus TCP Server with server ID=3, port=502, Read-write word register address ranges are 1 to 100, 500 to 560 and 900 to 2048, Read-only word register address ranges is 220 to 250. We use holding register to replace read-only input register. It means that it does not support input word registers, instead, we use read-only holding regs to replace them. The coil bit register address ranges is 1 to 1500. The discrete bit register address ranges is 2 to 1000. We don't support watchdog.

```
m=ModbusTCPServer("COM", "COM5", "ServerID", 3, "SupportedHoldingRegsSegs", {[1 100], [500 560], [900 2048]}, "SupportedInputRegs1to4Segs", {[220 250]}, ...
"SupportedCoils1to4Segs", {[1, 1500]}, "SupportedDiscreteRegs1to4Segs",
{[2, 1000]}, "HoldingReplaceInputRegs1to4", true);
```

Properties

- 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". This is public Property, so you can change it in the run-time by assignment
- ServerID — Modbus TCP Server ID
Value is scalar double: 1 to 247. Default=1. This is public Property, so you can change it in the run-time by assignment

Object Functions (Or Methods)

read	read Server registers values
write	write Server read-only registers or discrete bit registers.
resetDevice	Clear all register values Device supports to zero.
enableDevice	Enable Device Modbus communication

Events

None

All methods details

read
read Server registers values
Since R2019b

Syntax

```
[moddata,Error] = read(obj,target,address)
[moddata,Error] = read(obj,target,address,count)
[moddata,Error] = read(obj,target,address,count,precision)
```

Description

This function will get my Server register values from one of four target addressable areas: Coils, Inputs, Holding Registers, or Input Registers. Usually we only need to read Coils and Holding registers because Inputs (bits) and Input Registers (words) values are set by ours. We should know their values.

Input Arguments

- obj — ModbusTCPServer 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.
- **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 targets: inputregs and holdingregs.

Output Arguments

- **moddata** — Register Values to read.
Row vector for reading result in double number type. It will be [] if failed
- **Error** — failure for reading.
Logical value. True means fail to read.

Examples

Firstly, we set up a Modbus TCP Server object with COM5 and Server ID is 3. They support Read-write holding registers addressing from 1 to 100. They support Read-only input registers addressing from 22 to 25. They support coil registers addressing from 1 to 15. They support discrete registers addressing from 2 to 8. Word order is big-endian.

```
m=ModbusTCPServer(502, "COM", "COM5", "ServerID", 3, "SupportedHoldingRegsSegs", {[1
100]}, "SupportedInputRegsSegs", {[22 25]}, ...
"SupportedCoilsSegs", {[1, 15]}, "SupportedDiscreteRegsSegs", {[2, 8]});
```

% Read 3 coil values starting at address 5

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

% 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.

```
address = 10;
precision = {'uint16', 'double'};
count = [2, 4];
```

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

write

write Server read-only registers or discrete bit registers.
Since R2019b

Syntax

```
Error=write(obj,target,address,values)
Error=write(obj,target,address,values,precision)
```

Description

This function will perform a write operation to one of two simulated target addressable areas: inputs (Discrete) or input Registers (read-only word registers).

Input Arguments

- **obj** — ModbusTCPServer object. This argument is mandatory.
- **target** — Specifies the area to write. The valid choices are 'inputs' (discrete bit) and 'inputregs' (read-only words). This argument is mandatory.
- **address** — The starting address to write to. This is 1-based address without 4x/3x... prefix. This argument is mandatory.
- **values** — Array of values to write. For target 'inputs' valid values are false and true. For target 'inputregs' valid values must be in the range of the specified 'precision'. This argument is mandatory.
- **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

- **Error** — failure for writing.
Logical value. True means fail to write.

Examples

Firstly, we set up a Modbus TCP Server object with COM5 and Server ID is 3. They support Read-write holding registers addressing from 1 to 100. They support Read-only input registers addressing from 22 to 25. They support coil registers addressing from 1 to 15. They support discrete registers addressing from 2 to 8. Word order is big-endian.

```
m=ModbusTCPServer("COM", "COM5", "ServerID", 3, "SupportedHoldingRegsSegs", {[1 100]},
"SupportedInputRegsSegs", {[22 25]}, ...
"SupportedCoilsSegs", {[1, 15]}, "SupportedDiscreteRegsSegs", {[2, 8]});
```

% set the input word registers at address 22 to the value 2000

```
Error = write(m,'inputregs',22,2000);
```

resetDevice

Clear all register values Device supports to zero.
Since R2019b

Syntax

Error=resetDevice(obj)

Description

This function will put all registers simulator supports to zero.

Input Arguments

- obj — ModbusTCPServer object. This argument is mandatory.

Output Arguments

- Error — failure for reset.
Logical value. True means fail to reset.

enableDevice

Enable Device Modbus communication
Since R2019b

Syntax

Error=enableDevice(obj, enabled)

Description

This function will enable/disable simulated server

Input Arguments

- obj — ModbusTCPServer object. This argument is mandatory.
- enabled — logical scalar. true means we enable Modbus server communication. false means we disable Modbus server communication.

Output Arguments

- Error — failure for enable/disable.
Logical value. True means fail to enable/disable.

Note: If we never call this enableDevice method, all devices are enabled.

10 Simulink Blocks

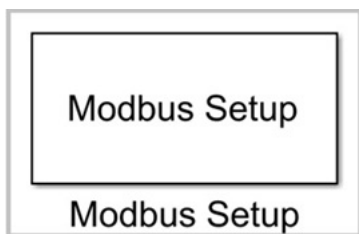
We will introduce every block diagram node for Modbus Server in the following sections.

10.1 ModbusServer_setup

ModbusServer_setup

Set up Modbus Server (Modbus Slave Simulator from [Dafulai Electronic Inc](#))
Since R2019b

Library: Modbus Server (Dafulai Electronics) /ModbusServer_setup



Description

This block sets up all parameters of Modbus Slave (Server) Simulator from [Dafulai Electronic Inc](#). You must put this block in your simulation model in order to control Modbus Server.

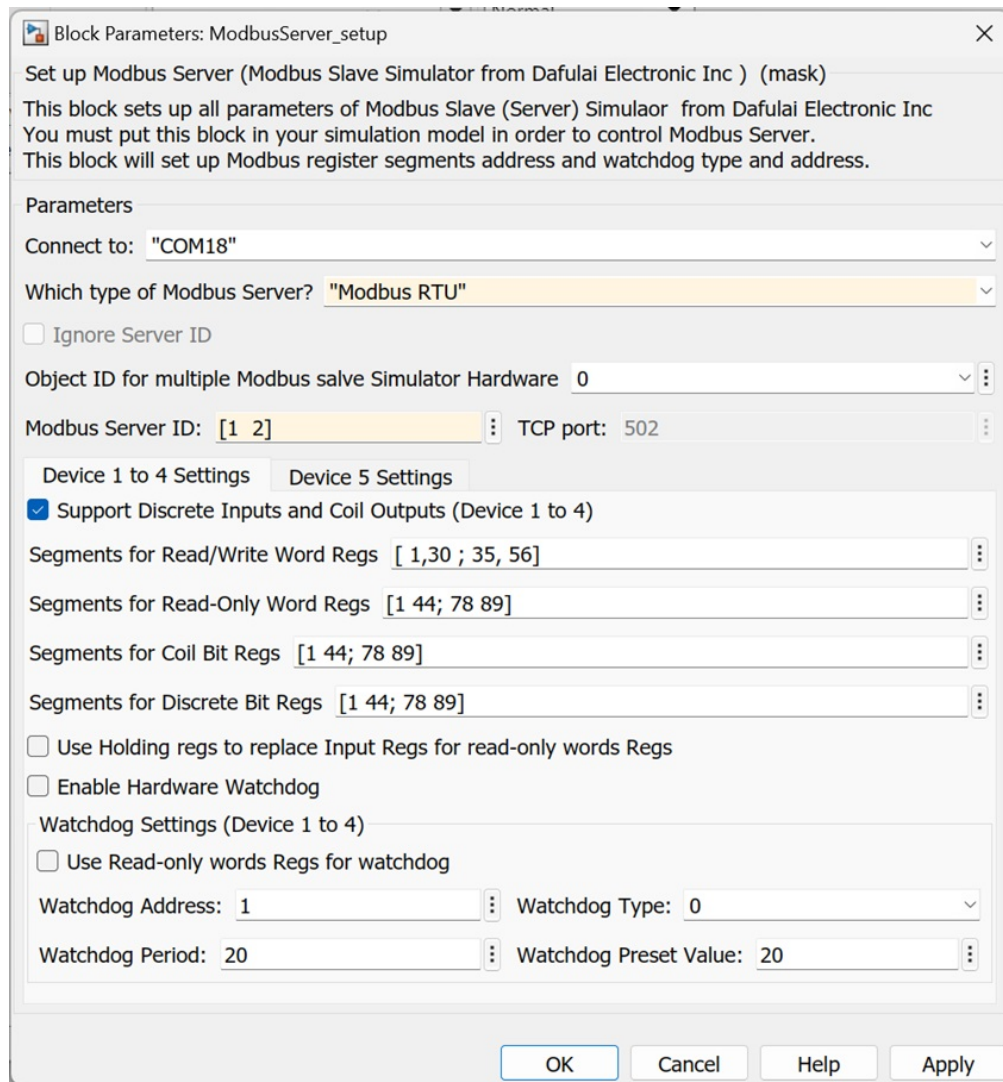
For Modbus RTU/ASCII Server, it can support as many as 5 servers. So you can see tab "Device 1 to 4 Settings" and tab "Device 5 Settings" (Device1 to4 has the same register structures). When you use Modbus TCP server, it can only support one server, So only tab "Device 1 to 4 Settings" is enabled, tab "Device 5 Settings" is grey color to disable.

Modbus TCP Server does not support watchdog. So all watchdog settings are grey color to disable.

This block will set up Modbus register segments address and watchdog type/address.

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 slave simulator hardware. You must connect USB or Bluetooth Serial port for our Modbus slave simulator hardware even though you only use Modbus TCP server.
- **Object ID for multiple Modbus slave Simulator Hardware** — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.
- **Segments for XXX regs** — It is double type Matrix (n x 2). Every row of matrix denotes register's one segment start/end address. Row numbers are segment quantities. For Modbus RTU/ASCII server, Max row numbers are 10. For Modbus TCP Server, there is no limit for row numbers.
- **Use Holding regs to replace Input Regs for read-only words Regs** — It means Server does not support FC=04 to read input registers. Instead of it, Server uses FC=03 to read holding regs (Read-only).

Ports

Input

None

Output

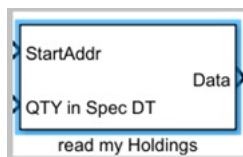
None

10.2 readMyHoldings

readMyHoldings

read Modbus Server Read-write Holding register value
Since R2019b

Library: Modbus Server (Dafulai Electronics) /readMyHoldings



Description

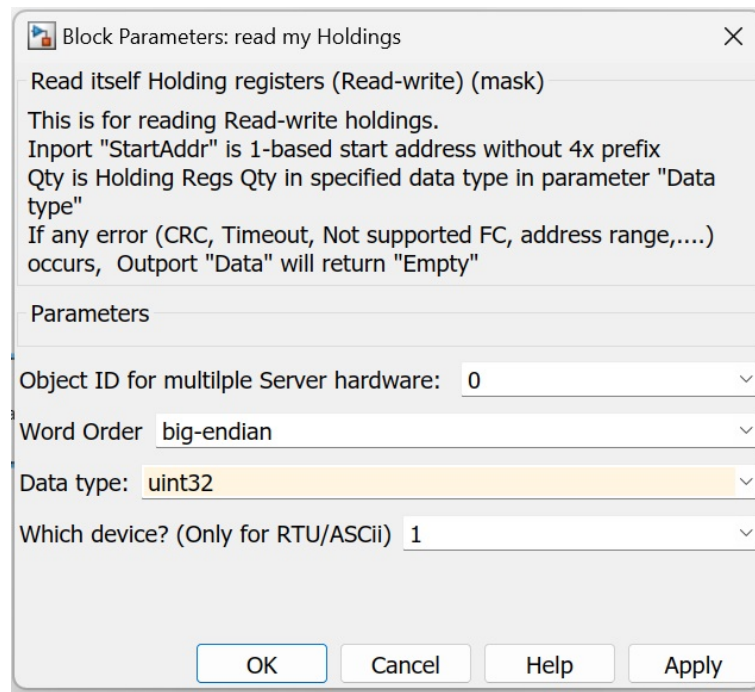
This block reads Read-Write Holding register value. Start Address is from Input port "StartAddr" (1-based address without 4x prefix), Read Quantities are from Input port "QTY in Spec DT". However this Quantities is in unit of Destination Data type. For example, if Destination Data type is "uint32", and Input port "QTY in Spec DT" is 5. Actually the words Quantities will be $5 \times 2 = 10$. (From "StartAddr" to "StartAddr"+9)

Those Read-Write Holding register values are controlled by "Outside Modbus Master", Simulator cannot control these values except Read-Write watchdog and watchdog type=1 or 2

If any error (CRC, Timeout, Not supported FC, address range,...) occurs, Output "Data" will return "Empty"

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple Modbus slave Simulator Hardware — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.
- Word Order — It denote the words order when register data type is "uint32/int32/single/uint64/int64/double".
- 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".
- Which device? (Only for RTU/ASCII) — This is useful for Modbus RTU/ASCII useful. it denotes which device we read from. For Modbus TCP Server, just ignore it.

Ports

Input

- StartAddr — "double" data type's scalar. It is holding Regs start address (1 based without 4X prefix) you want to read.
- QTY in Spec DT — "double" data type's scalar. It is Holding Regs QTY you want to read in specified data type in parameter "Data type".

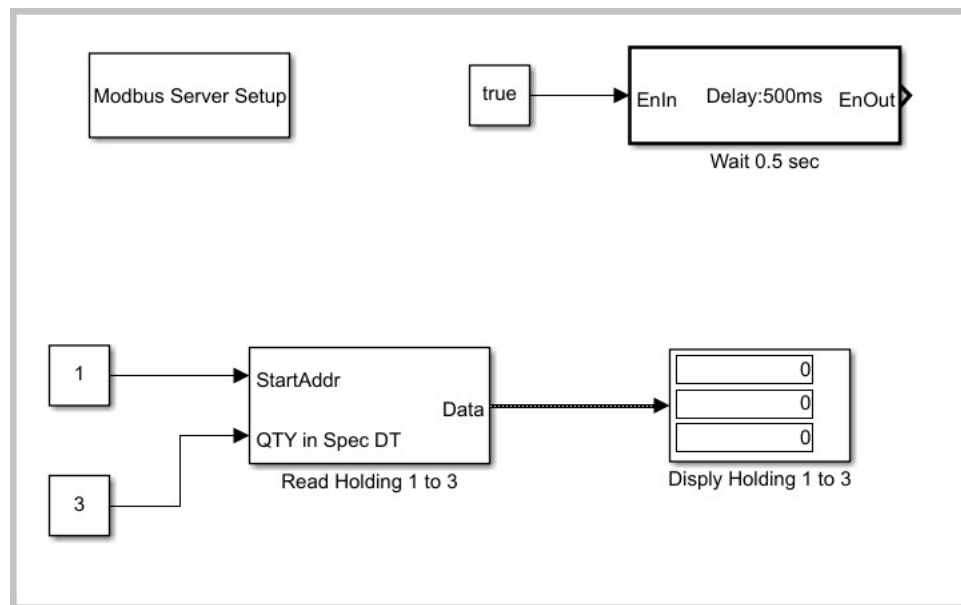
Output

- 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 become empty

Examples

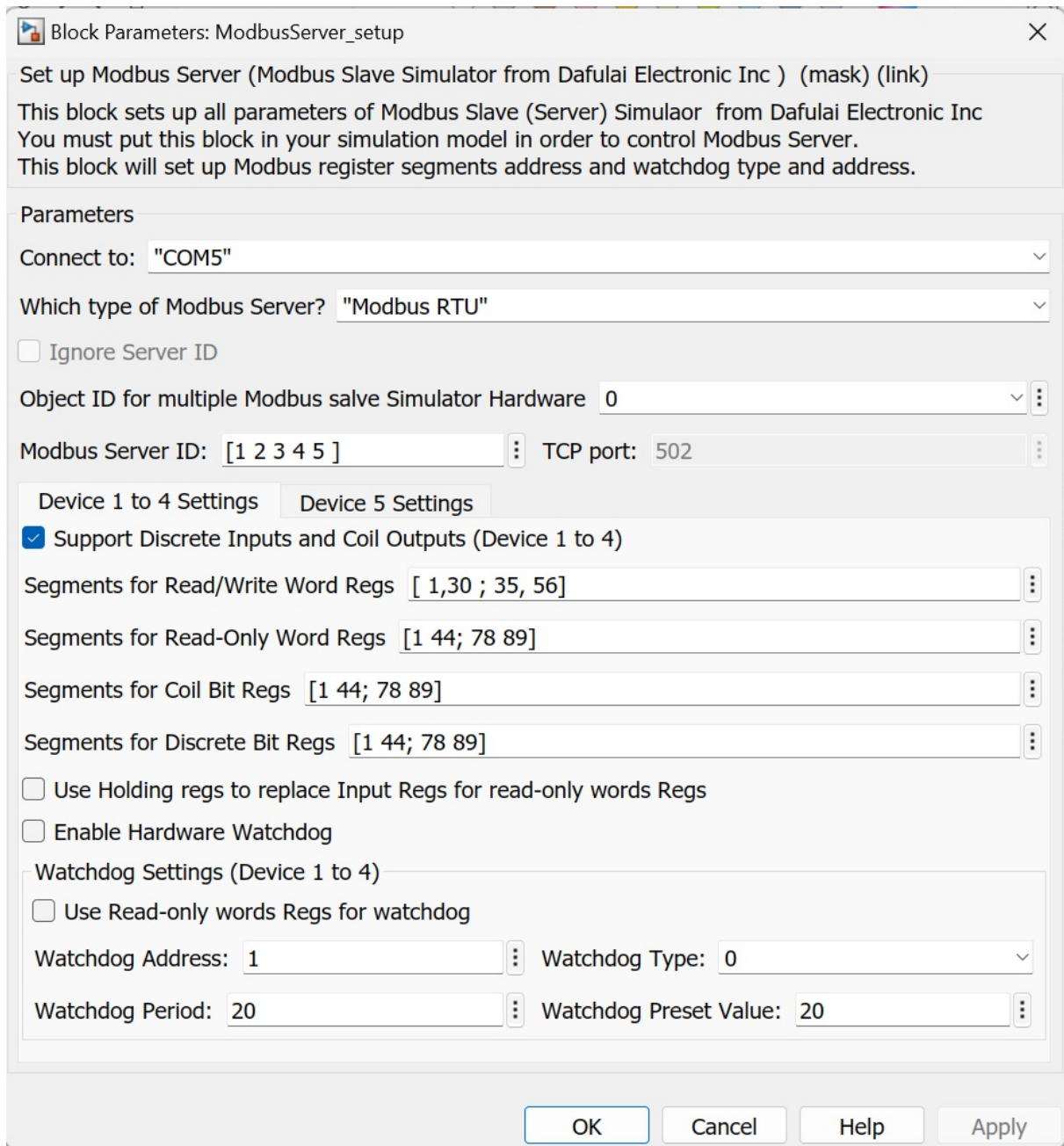
Example:

Every 500ms (Wait 0.5 sec block), We are reading read-write holding registers address from 1 to 3 of Modbus RTU Server Device1.



Please open "your Modbus Server library folder"/examples/example1_readHolding.slx (You must change USB serial Port number in Modbus Server Setup block according to your physical USB port number).

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

Block Parameters: ModbusServer_setup

Set up Modbus Server (Modbus Slave Simulator from Dafulai Electronic Inc) (mask) (link)

This block sets up all parameters of Modbus Slave (Server) Simulaor from Dafulai Electronic Inc
You must put this block in your simulation model in order to control Modbus Server.
This block will set up Modbus register segments address and watchdog type and address.

Parameters

Connect to: "COM5"

Which type of Modbus Server? "Modbus RTU"

☐ Ignore Server ID

Object ID for multiple Modbus salve Simulator Hardware 0

Modbus Server ID: [1 2 3 4 5] TCP port: 502

Device 1 to 4 Settings Device 5 Settings

☒ Support Discrete Inputs and Coil Outputs (Device 1 to 4)

Segments for Read/Write Word Regs [1,30 ; 35, 56]

Segments for Read-Only Word Regs [1 44; 78 89]

Segments for Coil Bit Regs [1 44; 78 89]

Segments for Discrete Bit Regs [1 44; 78 89]

☐ Use Holding regs to replace Input Regs for read-only words Regs

☐ Enable Hardware Watchdog

Watchdog Settings (Device 1 to 4)

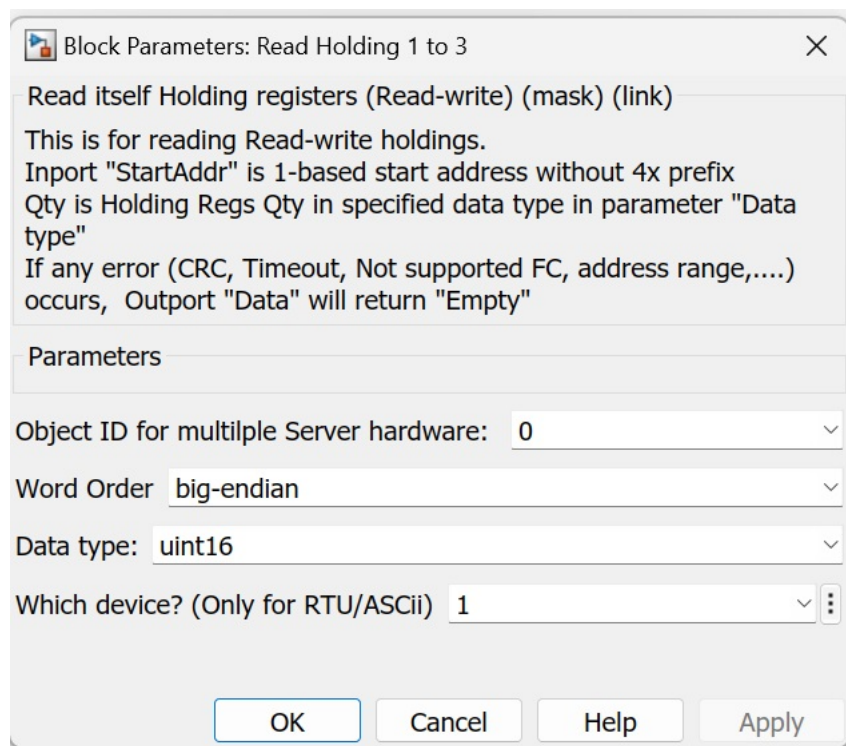
☐ Use Read-only words Regs for watchdog

Watchdog Address: 1 Watchdog Type: 0

Watchdog Period: 20 Watchdog Preset Value: 20

OK Cancel Help Apply

For "Read Holding 1 to 3 " block, the parameters are set up below:



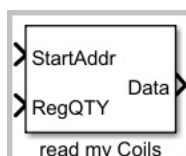
You can run general Modbus master software such as Modbus Poll or ModScan32 to change Holding registers values addressing 1 to 3, you will see its result in Simulink Display block.

10.3 readMyCoils

readMyCoils

read Modbus Server Coils register value
Since R2019b

Library: Modbus Server (Dafulai Electronics) /readMyCoils



Description

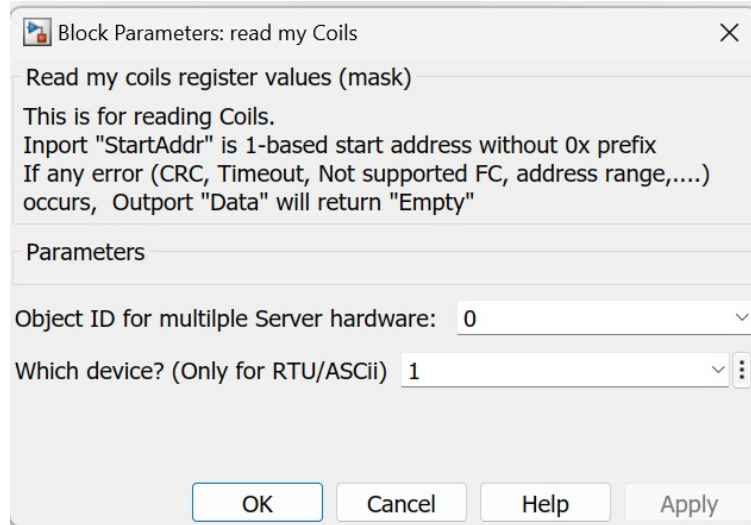
This block reads Coils register values. Start Address is from Input port "StartAddr" (1-based address without 0X prefix), Read Quantities are from Input port "RegQTY".

Those Coils register values are controlled by "Outside Modbus Master", Simulator cannot control these values.

If any error (CRC, Timeout, Not supported FC, address range,...) occurs, Outport "Data" will return "Empty"

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple Modbus slave Simulator Hardware — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.
- Which device? (Only for RTU/ASCII) — This is useful for Modbus RTU/ASCII useful. it denotes which device we read from. For Modbus TCP Server, just ignore it.

Ports

Input

- StartAddr — "double" data type's scalar. It is Coil Regs start address (1 based without 0X prefix) you want to read.
- RegQTY — "double" data type's scalar. It is coil Regs QTY you want to read

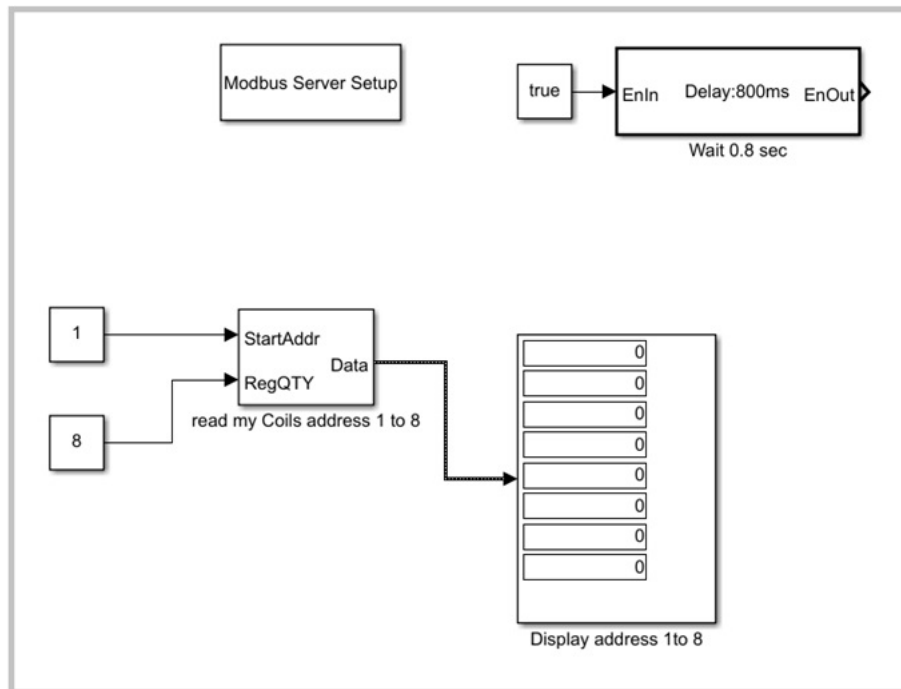
Output

- 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 become empty

Examples

Example:

Every 800ms (Wait 0.8 sec block), We are reading coil registers address from 1 to 8 of Modbus TCP Server.



Please open "your Modbus Server library folder"/examples/example2_readCoils.slx (You must change USB serial Port number in Modbus Server Setup block according to your physical USB port number).

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

Block Parameters: ModbusServer_setup

Set up Modbus Server (Modbus Slave Simulator from Dafulai Electronic Inc) (mask) (link)

This block sets up all parameters of Modbus Slave (Server) Simulaor from Dafulai Electronic Inc
You must put this block in your simulation model in order to control Modbus Server.
This block will set up Modbus register segments address and watchdog type and address.

Parameters

Connect to: "COM5"

Which type of Modbus Server? "Modbus TCP"

☐ Ignore Server ID

Object ID for multiple Modbus salve Simulator Hardware 0

Modbus Server ID: 1 TCP port: 502

Device 1 to 4 Settings Device 5 Settings

☒ Support Discrete Inputs and Coil Outputs (Device 1 to 4)

Segments for Read/Write Word Regs [1,30 ; 35, 56]

Segments for Read-Only Word Regs [1 44; 78 89]

Segments for Coil Bit Regs [1 44; 78 89]

Segments for Discrete Bit Regs [1 44; 78 89]

☐ Use Holding regs to replace Input Regs for read-only words Regs

☐ Enable Hardware Watchdog

Watchdog Settings (Device 1 to 4)

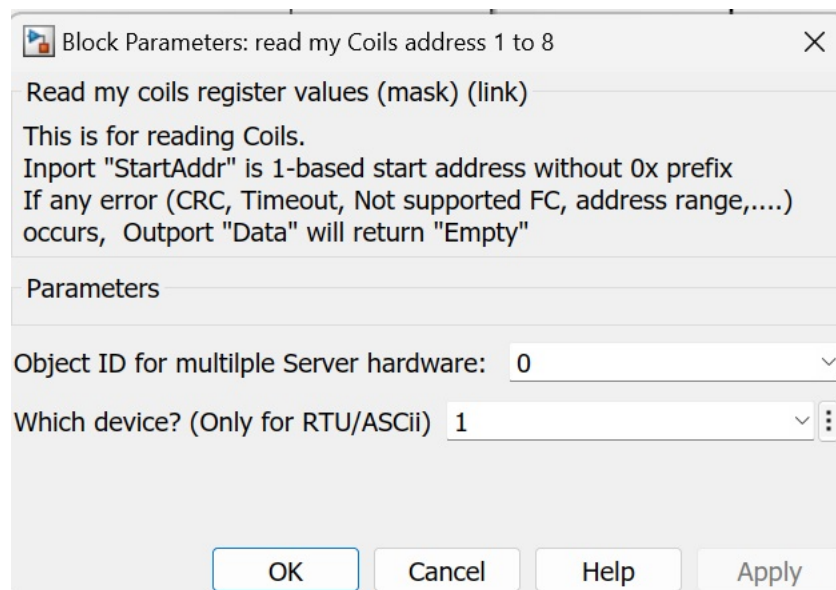
☐ Use Read-only words Regs for watchdog

Watchdog Address: 1 Watchdog Type: 0

Watchdog Period: 20 Watchdog Preset Value: 20

OK Cancel Help Apply

For "Read my Coils address 1 to 8 " block, the parameters are set up below:



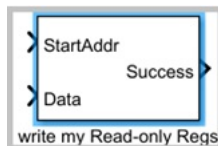
You can run general Modbus master software such as Modbus Poll or ModScan32 to change coil registers values addressing 1 to 8, you will see its result in Simulink Display block.

10.4 writeMyReadOnlyRegs

writeMyReadOnlyRegs

write Modbus Server Read-only register value
Since R2019b

Library: Modbus Server (Dafulai Electronics) /writeMyReadOnlyRegs



Description

This block writes Read-only register values. Start Address is from Input port "StartAddr" (1-based address without 3x or 4x prefix), write Quantities are decided by Input port "Data" vector's element numbers and parameter "Data type". For example, if parameter Data type is "uint32", and Input port "Data" contains 5 elements vector. Actually the words Quantities will be $5 \times 2 = 10$. (From "StartAddr" to "StartAddr"+9)

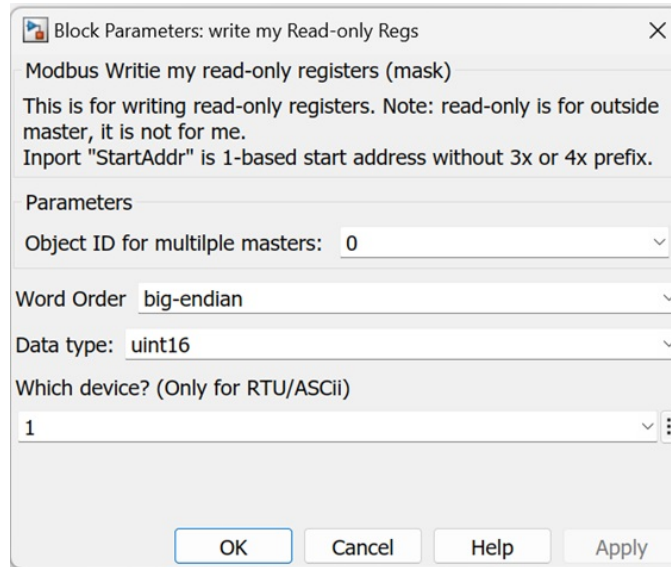
Those Read-only registers are only readable for "Outside Modbus Master". Simulator (Our PC) will control these values (writable).

If succeed to write, output port "Success" will become true, otherwise false.

This is very important block. All simulated data are from this block for words registers.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple Modbus slave Simulator Hardware — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.
- Word Order — It denote the words order when register data type is "uint32/int32/single/uint64/int64/double".
- Data type — Specifies the data format of the register being write to on the Modbus server. It is not Data type of input port "Data". The data type of input port "Data" is always "double".
- Which device? (Only for RTU/ASCII) — This is useful for Modbus RTU/ASCII useful. it denotes which device we write to. For Modbus TCP Server, just ignore it.

Ports

Input

- StartAddr — "double" data type's scalar. It is read-only Regs start address (1 based without 3X or 4X prefix) you want to write.
- Data — "double" data type's vector. It is all Data you write to.

Outport

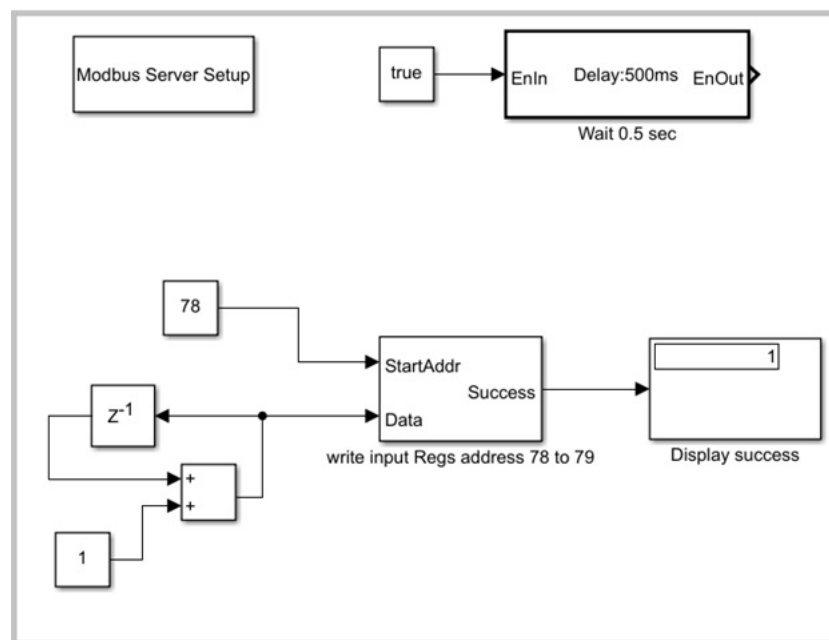
- Success — "logical" data type's scalar. If succeed to write, it will become true, otherwise false.

Examples

Example:

Every 500ms (Wait 0.5 sec block), We set Input registers addressing at 30078 and 30079 (it is address 78 and 79 if no prefix) increase 1 for Modbus RTU Server Device1.

Note: addressing at 30078 and 30079 combines one "uint32" data with "big-endian"



Please open "your Modbus Server library folder"/examples/example3_writeInputRegs.slx (You must change USB serial Port number in Modbus Server Setup block according to your physical USB port number).

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

Block Parameters: ModbusServer_setup [X]

Set up Modbus Server (Modbus Slave Simulator from Dafulai Electronic Inc) (mask) (link)

This block sets up all parameters of Modbus Slave (Server) Simulaor from Dafulai Electronic Inc
 You must put this block in your simulation model in order to control Modbus Server.
 This block will set up Modbus register segments address and watchdog type and address.

Parameters

Connect to: "COM5" [v]

Which type of Modbus Server? "Modbus RTU" [v]

☐ Ignore Server ID

Object ID for multiple Modbus salve Simulator Hardware 0 [v]

Modbus Server ID: [1 2 3 4 5] TCP port: 502 [v]

Device 1 to 4 Settings Device 5 Settings

☒ Support Discrete Inputs and Coil Outputs (Device 1 to 4)

Segments for Read/Write Word Regs [1,30 ; 35, 56] [v]

Segments for Read-Only Word Regs [1 44; 78 89] [v]

Segments for Coil Bit Regs [1 44; 78 89] [v]

Segments for Discrete Bit Regs [1 44; 78 89] [v]

☐ Use Holding regs to replace Input Regs for read-only words Regs

☐ Enable Hardware Watchdog

Watchdog Settings (Device 1 to 4)

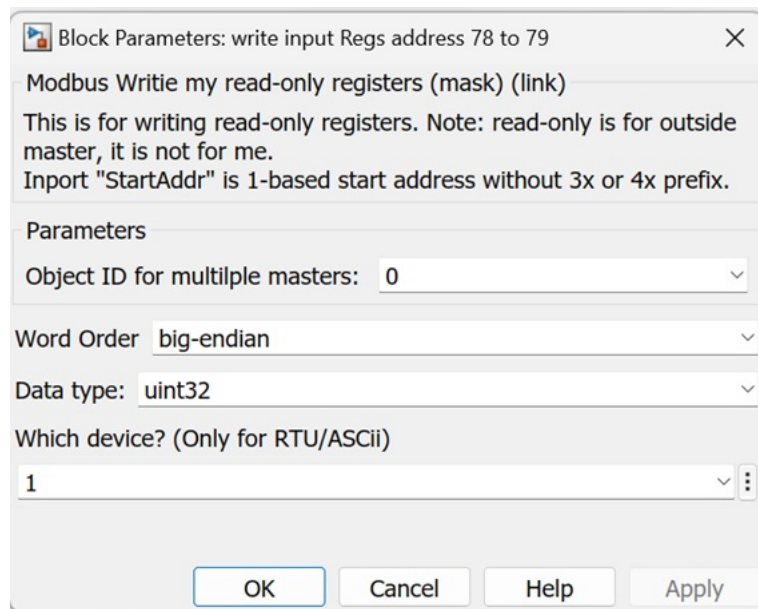
☐ Use Read-only words Regs for watchdog

Watchdog Address: 1 [v] Watchdog Type: 0 [v]

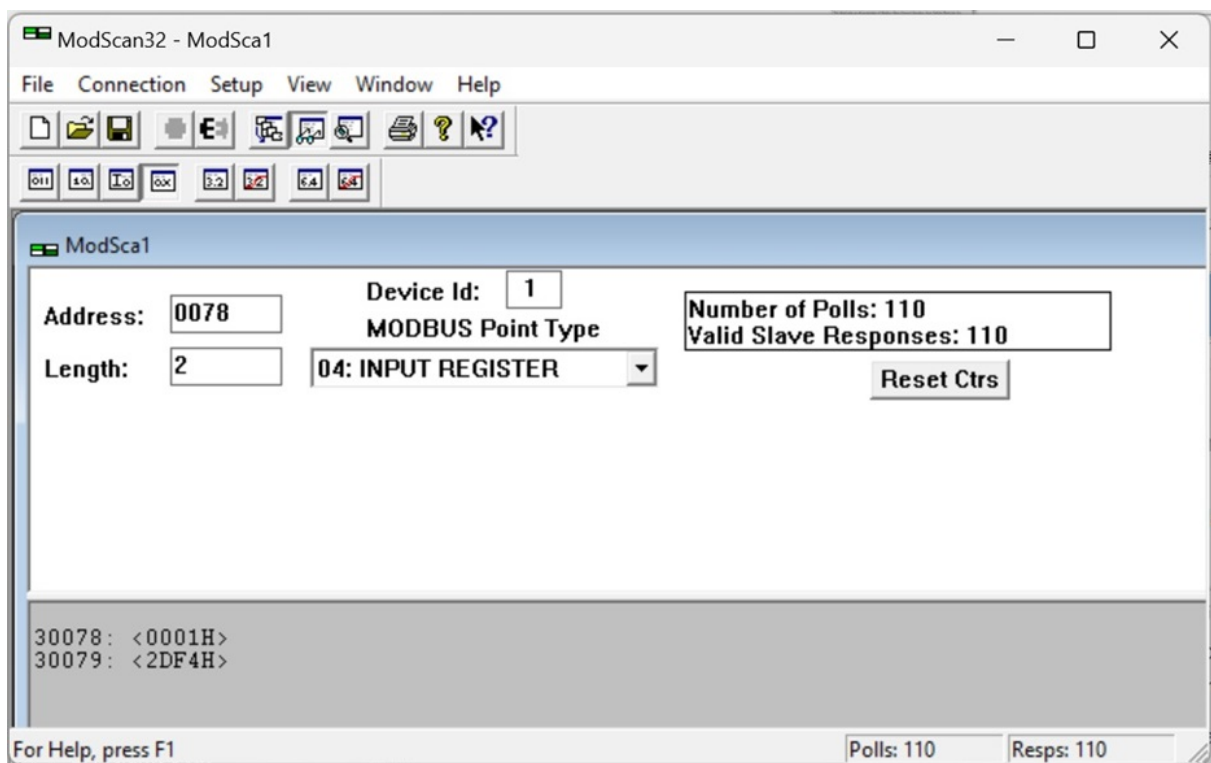
Watchdog Period: 20 [v] Watchdog Preset Value: 20 [v]

OK Cancel Help Apply

For "write input Regs address 78 to 79" block, the parameters are set up below:



You can run general Modbus master software such as Modbus Poll or ModScan32 to view input registers values addressing 30078 to 30079. We put ModScan32 result below:



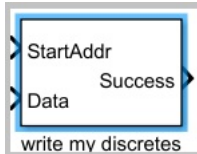
Notes: Because input port "Data" is "double" type, when increasing 1 causes over 0xFFFFFFFF, although input port "Data" can be over 0xFFFFFFFF, the register result will be limit to 0xFFFFFFFF due to "uint32" data type of register. If you want to return to zero when uint32 data is over 0xFFFFFFFF, you must modify Simulink model.

10.5 writeMyDiscreteRegs

writeMyDiscreteRegs

write Modbus Server discrete register value
Since R2019b

Library: Modbus Server (Dafulai Electronics) /writeMyDiscreteRegs



Description

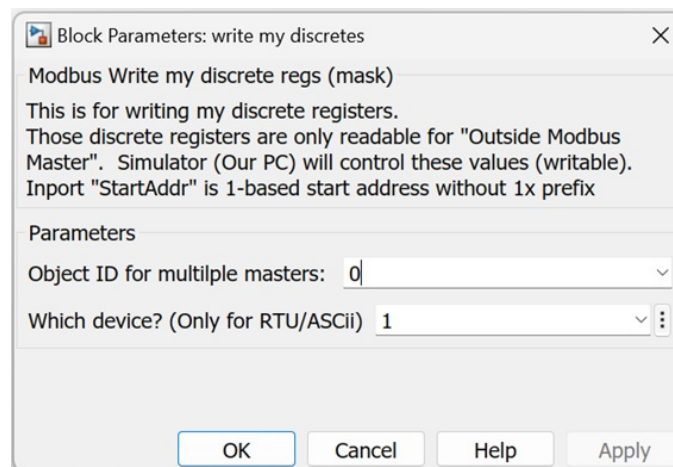
This block writes Discrete register values. Start Address is from Input port "StartAddr" (1-based address without 1X prefix), write Quantities are decided by Input port "Data" vector's element numbers.

Those discrete registers are only readable for "Outside Modbus Master". Simulator (Our PC) will control these values (writable).
If succeed to write, output port "Success" will become true, otherwise false.

This is very important block. All simulated data are from this block for bit registers.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple Modbus slave Simulator Hardware — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.
- Which device? (Only for RTU/ASCII) — This is useful for Modbus RTU/ASCII useful. it denotes which device we write to. For Modbus TCP Server, just ignore it.

Ports

Input

- StartAddr — "double" data type's scalar. It is Discrete Regs start address (1 based without 1X prefix) you want to write.
- Data — "logical" data type's vector. It is all Data you write to.

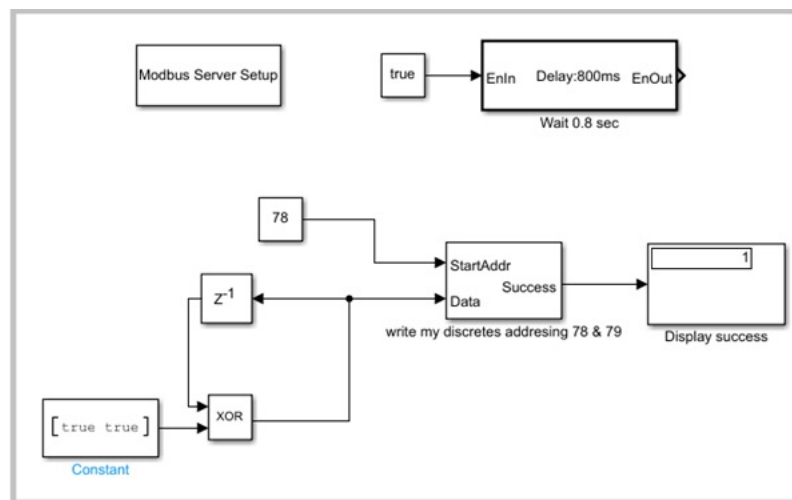
Output

- Success — "logical" data type's scalar. If succeed to write, it will become true, otherwise false.

Examples

Example:

Every 800ms (Wait 0.8 sec block), We toggle discrete registers address from 78 to 79 of Modbus TCP Server.



Please open "your Modbus Server library folder"/examples/example4_writeDiscrete.slx (You must change USB serial Port number in Modbus Server Setup block according to your physical USB port number).

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

Block Parameters: ModbusServer_setup

Set up Modbus Server (Modbus Slave Simulator from Dafulai Electronic Inc) (mask) (link)

This block sets up all parameters of Modbus Slave (Server) Simulaor from Dafulai Electronic Inc
You must put this block in your simulation model in order to control Modbus Server.
This block will set up Modbus register segments address and watchdog type and address.

Parameters

Connect to: "COM5"

Which type of Modbus Server? "Modbus TCP"

☐ Ignore Server ID

Object ID for multiple Modbus salve Simulator Hardware 0

Modbus Server ID: 1 TCP port: 502

Device 1 to 4 Settings Device 5 Settings

☒ Support Discrete Inputs and Coil Outputs (Device 1 to 4)

Segments for Read/Write Word Regs [1,30 ; 35, 56]

Segments for Read-Only Word Regs [1 44; 78 89]

Segments for Coil Bit Regs [1 44; 78 89]

Segments for Discrete Bit Regs [1 44; 78 89]

☐ Use Holding regs to replace Input Regs for read-only words Regs

☐ Enable Hardware Watchdog

Watchdog Settings (Device 1 to 4)

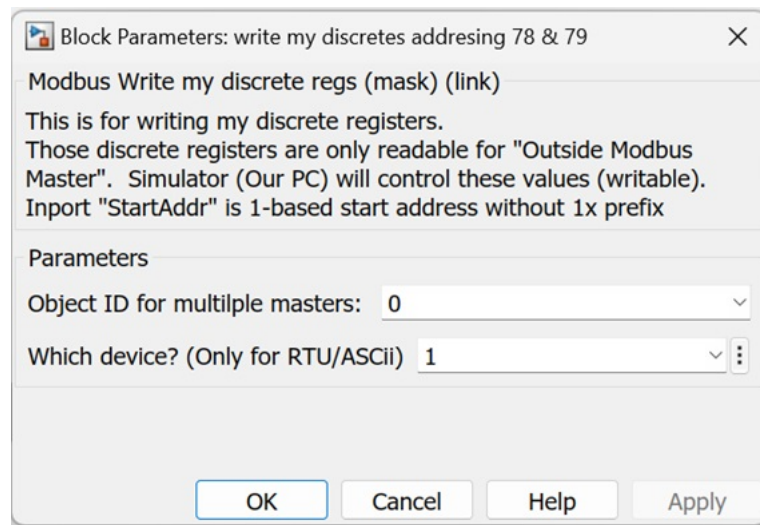
☐ Use Read-only words Regs for watchdog

Watchdog Address: 1 Watchdog Type: 0

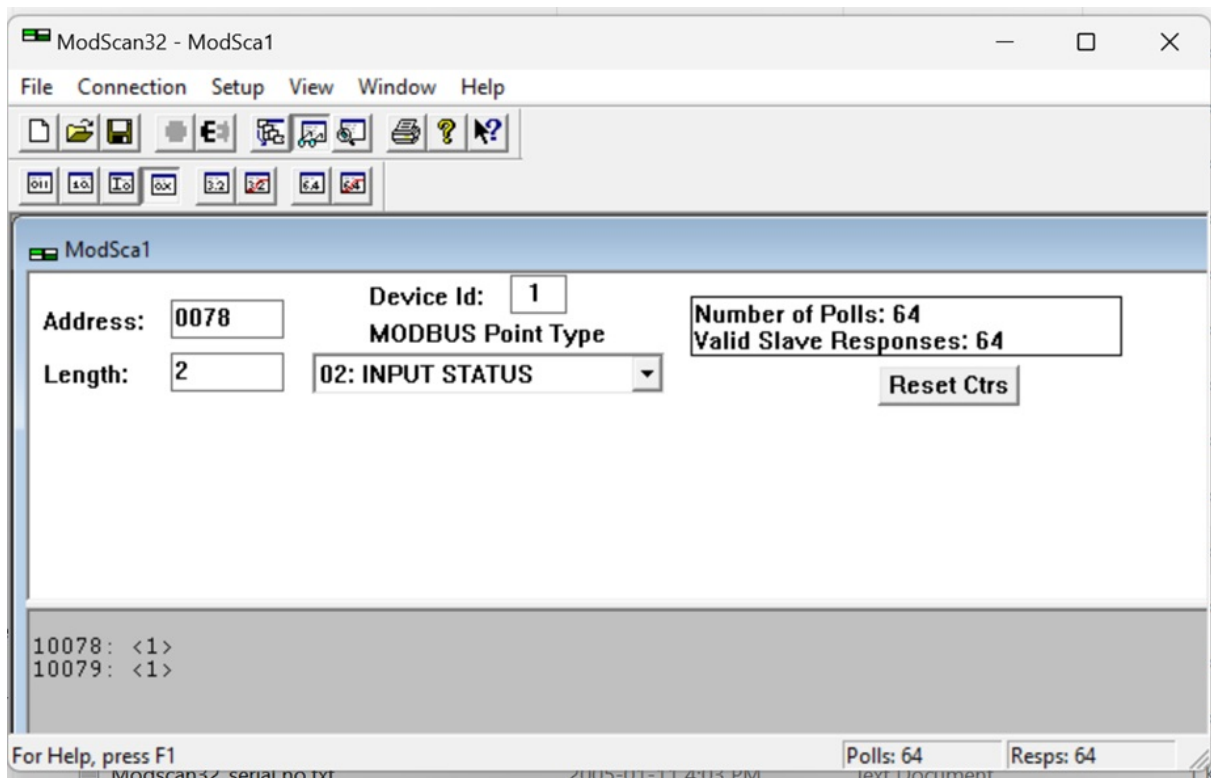
Watchdog Period: 20 Watchdog Preset Value: 20

OK Cancel Help Apply

For "write my discretes addressing 78 & 79 " block, the parameters are set up below:



You can run general Modbus master software such as Modbus Poll or ModScan32 to view discrete registers values addressing 10078 to 10079. We put ModScan32 result below (it will toggle every 0.8 sec):



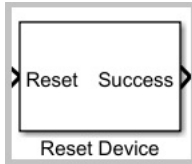
10.6 resetDevice

resetDevice

reset Modbus Server all register values

Since R2019b

Library: Modbus Server (Dafulai Electronics) /resetDevice

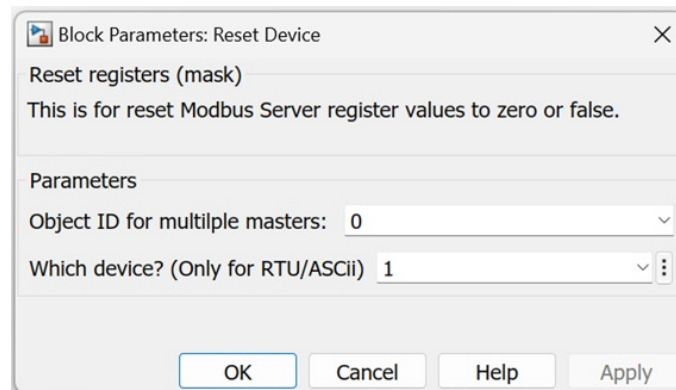


Description

This block resets all register values to zero or false.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- **Object ID for multiple Modbus slave Simulator Hardware** — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.
- **Which device? (Only for RTU/ASCII)** — This is useful only for Modbus RTU/ASCII Server. it denotes which device we want to reset. For Modbus TCP Server, just ignore it.

Ports

Input

- **Reset** — "logical" data type's scalar. Rising Edge from false to true causes "Reset registers to zero or false Action".

Output

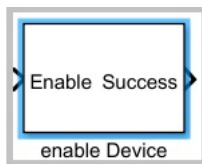
- Success — "logical" data type's scalar. If succeed to reset, it will become true, otherwise false.

10.7 enableDevice

enableDevice

enable Modbus Server communication
Since R2019b

Library: Modbus Server (Dafulai Electronics) /enableDevice

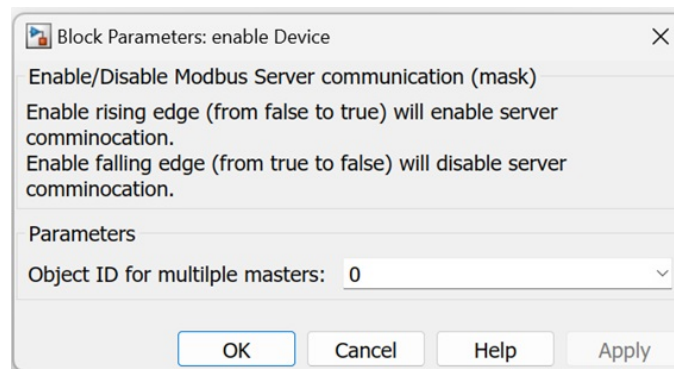


Description

This block enables/Disables Modbus servers Communication.
If we never use this block, the Modbus Servers communication will be enabled.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple Modbus slave Simulator Hardware — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.

Ports

Input

- Enable — "logical" data type's scalar for Modbus TCP server or logical" data type's vector for Modbus RTU/ASCII servers. In Modbus TCP Server situation, rising Edge from false to true causes "Enable server communication". Falling Edge from true to false causes "Disable server communication". In Modbus RTU/ASCII Server situation, Enable vector element numbers must be equal to device QTY, and any element changes will force to cause "Enable/Disable relative server communication". If true, then cause "Enable relative server communication", or false causes "Disable relative server communication"

Output

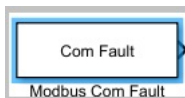
- Success — "logical" data type's scalar. If succeed to enable/disable, it will become true, otherwise false.

10.8 COMFault

COMFault

get Modbus Server communication fault signal
Since R2019b

Library: Modbus Server (Dafulai Electronics) /COMFault

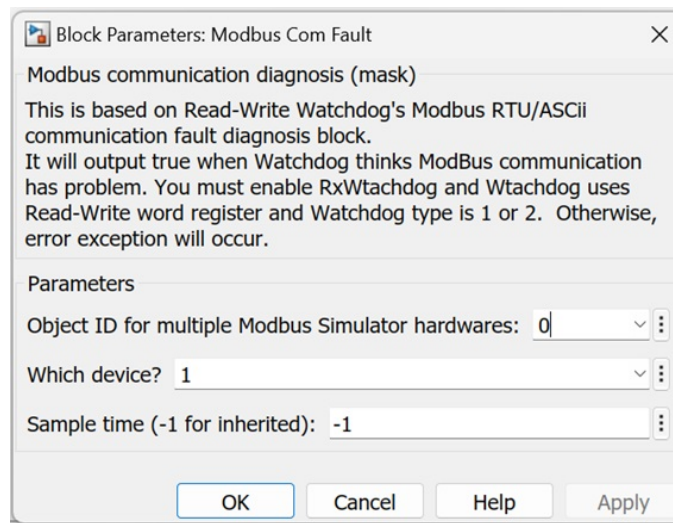


Description

This block is only useful for Modbus RTU/ASCII server and Watchdog is read-write register and watchdog type is type 1 or 2.
For other situations, you cannot use this block, otherwise you will get error exception

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Object ID for multiple Modbus slave Simulator Hardware — In one PC, we may use multiple Modbus slave simulators, this is for identifying each one.
- Which device? — This is useful for Modbus RTU/ASCII useful. it denotes which device we want to get COM fault.
- Sample time — This is sample time. double data scalar type. -1 is inherited from model sample time.

Ports

Input

None

Output

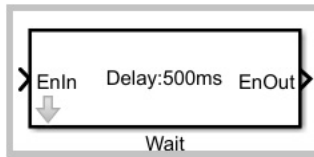
- "logical" data type's scalar. true means Communication Fault. false means communication is normal.

10.9 wait

wait

wait some time to pass.
Since R2019b

Library: CAN Bus Controller (Dafulai Electronics) /wait



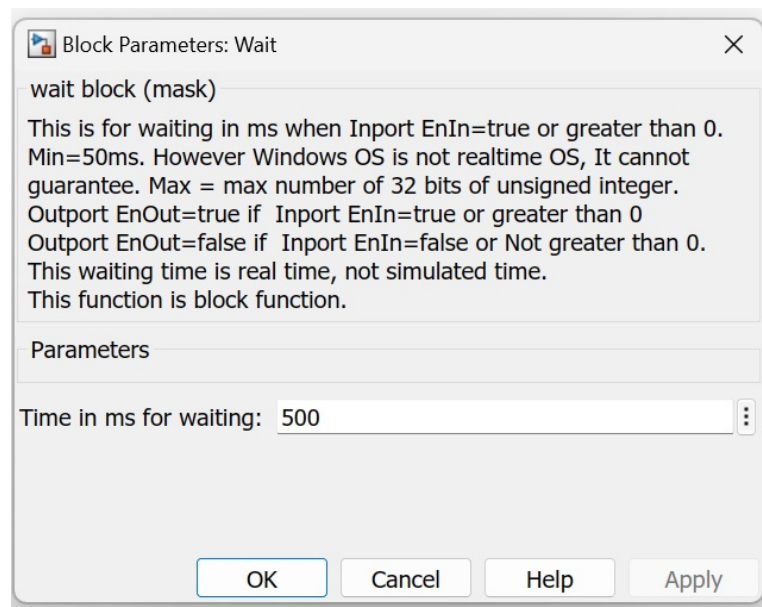
Description

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

EnIn — "number" scalar or "logical" scalar. true or >0 will delay specified time. false will be no any delay.

Output

EnOut — "logical" scalar. It is EnIn value. If EnIn is "number" type, EnOut= true when EnIn>0.

11 Electrical And Mechanical 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, -5°C to +65°C with Bluetooth
Dimensions79.35mm x 42.67mm x 23..47mm (L x W x H)
DC Power Supply5.5 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

The product may not be duplicated without authorization. Dafulai Company holds all copyright. Unauthorized duplication will be subject to penalty.