Simulink embedded C Creator Datasheet

Table of Contents

1 Overview

We know that Simulink can create embedded C code. It increases the efficiency of Code production, and it is block-connection programming method which forces module-style code. It also allows for the reuse of code and components, further reducing development time and cost.

However, After you create C code from Simulink, it is difficult to debug C code. General method is that we create another Simulink model for debugging/monitoring embedded target in PC Side. It needs much time for new Simulink model, and you cannot guarantee the consistent between Embedded Target Simulink model and PC debug/monitor Simulink model.

Dafulai Electronics provides embedded C code creator for Simulink. It fixes up disadvantage above.

All blocks from our embedded C code creator library are non-blocking except "simEEPROMWriteFlash" block (Flash erase/writing operation). So it is good for bare metal programming (No any operating system).

Furthermore, it provides "probe" block, "breakpoint" block for debugging/monitoring. And it has "emCustomCBlock" block, which can use mature existing non-blocking C code. Or you can manually write non-blocking C code. Or you can use AI (Copilot or Gemini) to create C code. In this way, Our library provides peripheral driver code extension and application code extension.

If you use "Pure Simulink block" to implement "Finite State Machine" instead of Stateflow, you easily implement "State" Visibility by our block "ColorDisplay". Active State (running State) has "Green" color background.

Embedded Target and PC debug share the same Simulink model.

Our Debug/monitor method is based on "built-in Target modbus Server". However, Debug/Monitor does not occupy any Modbus registers resource. It does not make any side effect to application modbus server.

In order to use our "embedded C code creator library", you need our hardware " Modbus RTU/ASCII Dual Masters adaptor" (Part Number: DFLDMB1) if you like to use debug function or Modbus Server/ Client.

Besides hardware, you need a license for "embedded C code creator library". If you have no license, Target will enter "Demo" mode. In "Demo" mode, Target can works well for Modbus Server/Client during the first 15 minutes. After 15 minutes, Modbus Sever/Client stop. For Microchip MCC/ STM32Cube platforms, when you re-program target, you can get new 15 minutes. For TI F28X SysConfig platform, it is no use to re-program target for getting new 15 minutes, you must power off target for getting new 15 minutes. So you cannot debug/monitor target after 15 minutes if you have no license. This license is a perpetual license which grants indefinite use of software for a one-time payment. Please purchase license on our website.

If you use TI C2000Ware SysConfig, Please directly use Flash version to debug. Don't use RAM version to debug.

2 Features Highlights

- Support Microchip MCC platform including PIC18/PIC24/dsPic33/PIC32 Mips/PIC32 Arm
- Support STM32CubeIDE and STM32CubeMx Platform
- Support TI C2000 CCS Platform with created code by SysConfig
- Support one Modbus RTU Master and/or 1 to 4 Modbus RTU Server
- Support as many as 2 CAN Bus nodes with CAN 2.0A and 2.0B or TI DSP CAN FD with max 8 bytes of data field.
- Support as many as 2 SPI bus masters and/or I2C masters
- Support Digital Inputs with filter and Analog inputs
- Support Customized C code blocks which can create special peripheral driver or use existing mature application C code or use C code AI created
- Support Simulated EEPROM from flash memory.
- Support embedded probe to view input and output values of blocks.
- embedded software breakpoint is supported.
- On-site Tuning parameter by constant block plus embedded probe

3 Debug/Monitor Hardware

Hardware of DFLDMB1

Debug/Monitor hardware is not mandatory if you don't use Debug/Monitor and don't use embedded

Modbus RTU.

Our debugging and monitoring use "Embedded code" inside target. This "Embedded code" is based on "Embedded Modbus RTU Server". However, Debug/Monitor does not occupy any Modbus server register address resource.

3.1 Pin Assignment

Our Debug/Monitor hardware ("Modbus Dual Masters") has 3 UARTs:

- UART1 : It is Modbus network connecting point. Physically, it can be configured as RS232 or RS485 or RS422. It will connect to Embedded Target (Monitored/Debugged Deveice)
- UART2: It is one modbus Master connecting point. Physically, it is USB Device (Logically, it is UART). It will connect PC, which runs Matlab/Simulink
- UART3: It is another modbus Master connecting point. Physically, it can be Bluetooth SPP (Logically, it is UART) or it can be RS485. Both Bluetooth and RS485 share the same UART. So you cannot use them at the same time. It will connect PC, which runs Matlab/Simulink

Usually, You only use one of UART2 and UART3 for connecting PC which runs Matlab/Simulink. If we want isolation between target board and PC debug/monitor, it is good idea to use Bluetooth UART3.

Male DB9 Connector:

Table

Pin	Name	Description
1	1st RS485+/1st RS422 TX+	This is for the first UART: RS485 + and RS422 TX +. This is Modbus network connecting point, which connects target board.
2	1st RS232 TX/1st RS485-/1st RS422 TX-	This is for the first UART: RS232 TX, RS485 - and RS422 TX-. This is Modbus network connecting point, which connects target board.
3	1st RS232 RX/1st RS422 RX+	This is for the first UART: RS232 RX and RS422 RX+. This is Modbus network connecting point, which connects target board.
4	1st RS422 RX-	This is for the first UART: RS422 RX-. This is Modbus network connecting point,which connects target board.

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. If you use USB power supply, just keep this pin unconnected.
8	3rd RS485+	This is for the 3rd UART: RS485+. If you don't use USB, you can use it to connect PC which runs Matlab/Simulink. Otherwise, keep this pin unconnected
9	3rd RS485-	This is for the 3rd UART: RS485-. This is the 2nd master connecting point. If you don't use USB, you can use it to connect PC which runs Matlab/ Simulink. Otherwise, keep this pin unconnected

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

DB9 to Terminal Cable

3.2 LED Indication

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

1 Mode LED (Red color)

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

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

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

2 Bluetooth LED (Green color)

If this LED is half bright, it means Bluetooth is disabled.

If this LED is blinking, it means Bluetooth is searching for connecting, but actually it connects nothing.

If this LED is bright, it means Bluetooth is enabled and DFLDMB1 (this adaptor) has connected one master.

***Notes**: 1 This LED will still blink if you only pair successfully. You must connect Bluetooth COM port in your PC. This LED will be bright after your PC application software connects Bluetooth Serial Port. The baud rate of Bluetooth COM port can be any value when you set up baud rate in PC application (Not our configuration software) .*

If you didn't open Bluetooth COM port, This LED will blink.

2. How to find and pair Bluetooth in your PC? it depends on your PC OS. Please use google to search solution for your operating system.

3 You cannot put DFLDMB1 into Metal Panel BOX for Bluetooth. Bluetooth cannot work when it is in metal container except your PC is in the same metal container.

3 USB TX or UART2 TX LED (Red color)

When USB or UART2 transmits any data, this LED will be on.

4 USB RX or UART2 RX LED (Green color)

When USB or UART2 receives any data, this LED will be on.

5 UART3 or Bluetooth TX LED (Red color)

When UART3 or Bluetooth transmits any data, this LED will be on.

6 UART3 or Bluetooth Rx LED (Green color)

When UART3 or Bluetooth received any data, this LED will be on.

4 Debug/Monitor Configuration

ConfigTool is a tool for configuration of Debug/Monitor Hardware.

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

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

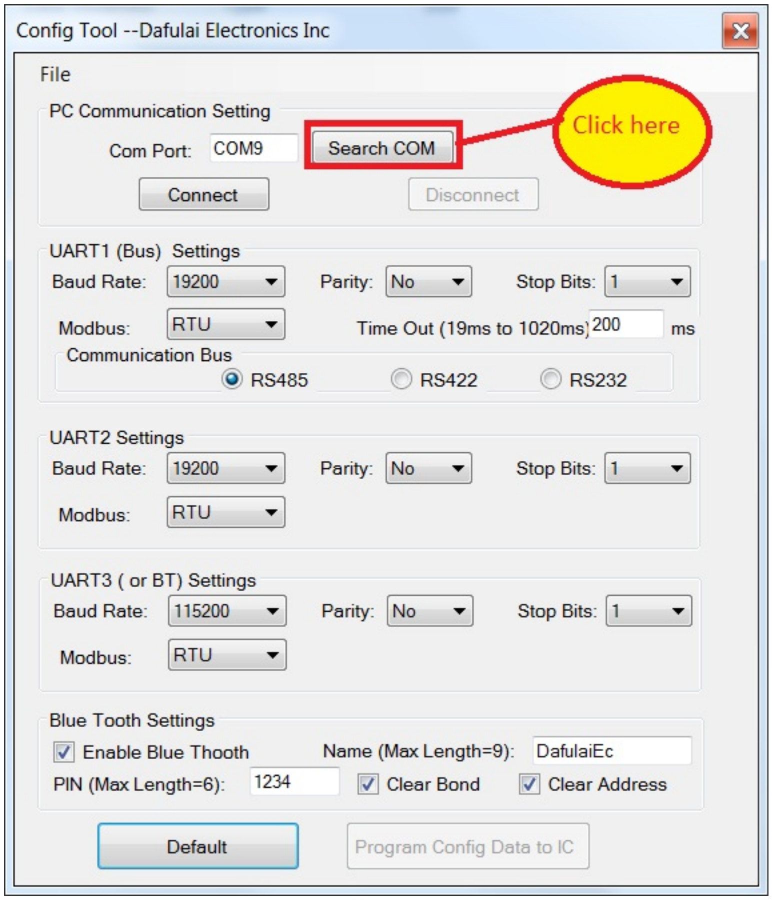
4.1 Debug/Monitor Parameters Setting

Step 1:

Plug in USB cable to DFLDMB1 and PC. unplug DB9 to make UART1 disconnecting (or you have no any Modbus slave nodes power on).
Make sure any Bluetooth Master didn't connect DFLDMB1 (You can Pair, but you cannot open Bluetooth COM port. Bluetooth LED will be blinking, Or Bluetooth LED will be half bright on DFLDMB1)

Step 2:

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



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

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

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

The good way to identity Com port in In multiple COM ports available is that unplug USB cable and execute "Search COM" command again. If one Com port is not in the available COM Ports, this COM Port will be our DFLJ1939Mod1, and Plug USB cable again

Step 3:

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

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

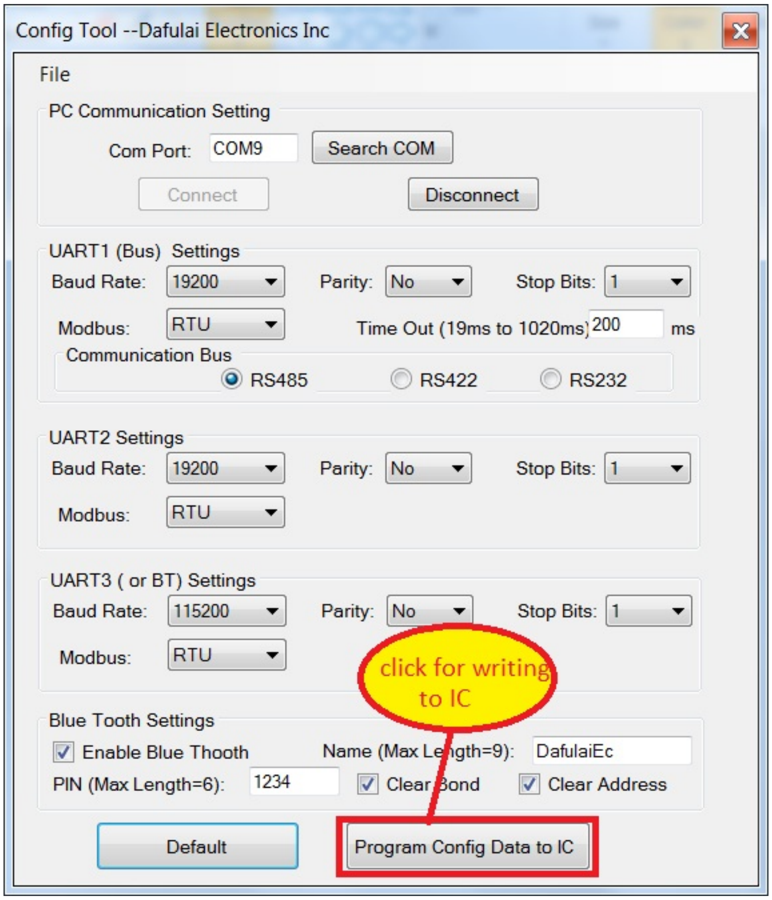
For actual master node, time out value is different from this one, it must be much biggest UART1 time out , and it depends on maximum response length because probably the other master is getting slave response when this master is sending Modbus request.

2. Bluetooth only uses 115200 baud rate (This is not air baud rate, this is our UART3's baud rate). If you choose the other value for UART3 baud rate, Bluetooth will be disabled automatically.

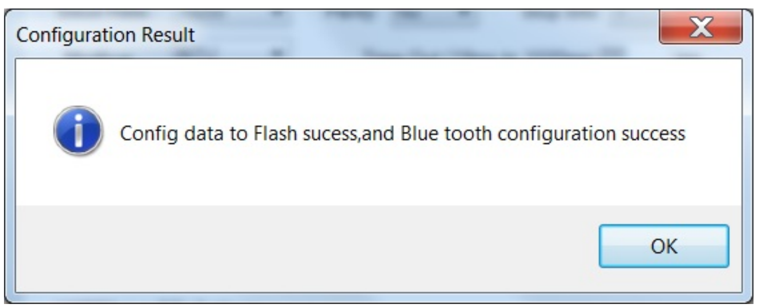
3. Bluetooth name has maximum 9 characters length. First character must be letter. Actually you will get 2 Bluetooth names in your PC. One is with suffix 1, the other is with suffix 2. The one with suffix 1 is Bluetooth EDR, which is what we use. The one with suffix 2 is Bluetooth BLE, which is not what we use. So you should choose one with suffix 1 to connect.

Step 4:

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

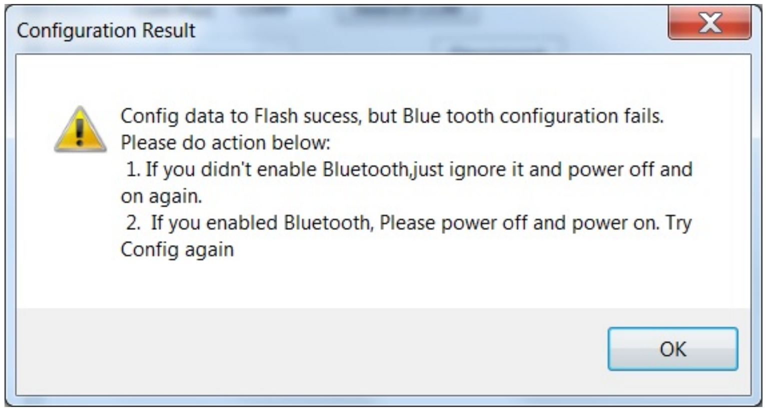


Mode LED will blink fast. You will see progress in PC. After progress arrive at 100%, you will see result below:



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

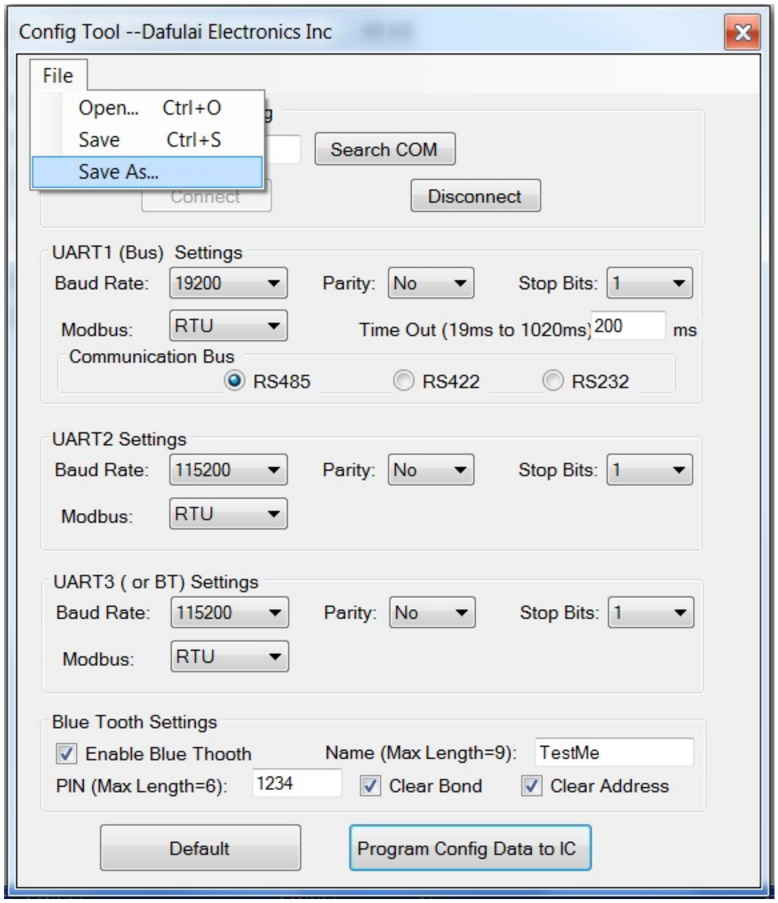
However, if you see result below:



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

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

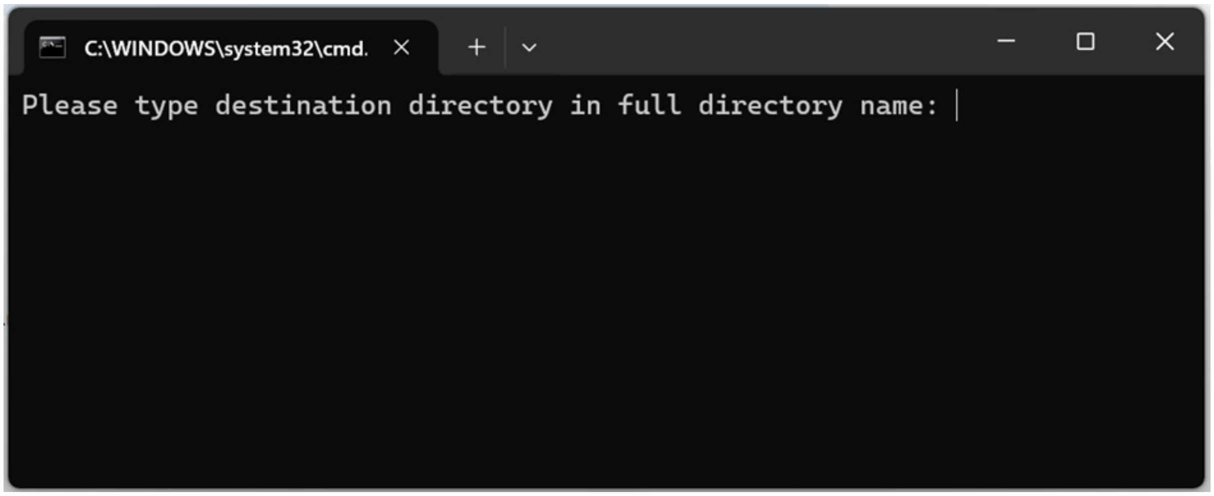
You can save your configuration to a file, your team workers can use your configuration.
Please see screenshot for save :



5 How to install Simulink Embedded C Creator library?

Please follow steps below for installing Simulink Library for windows platform:

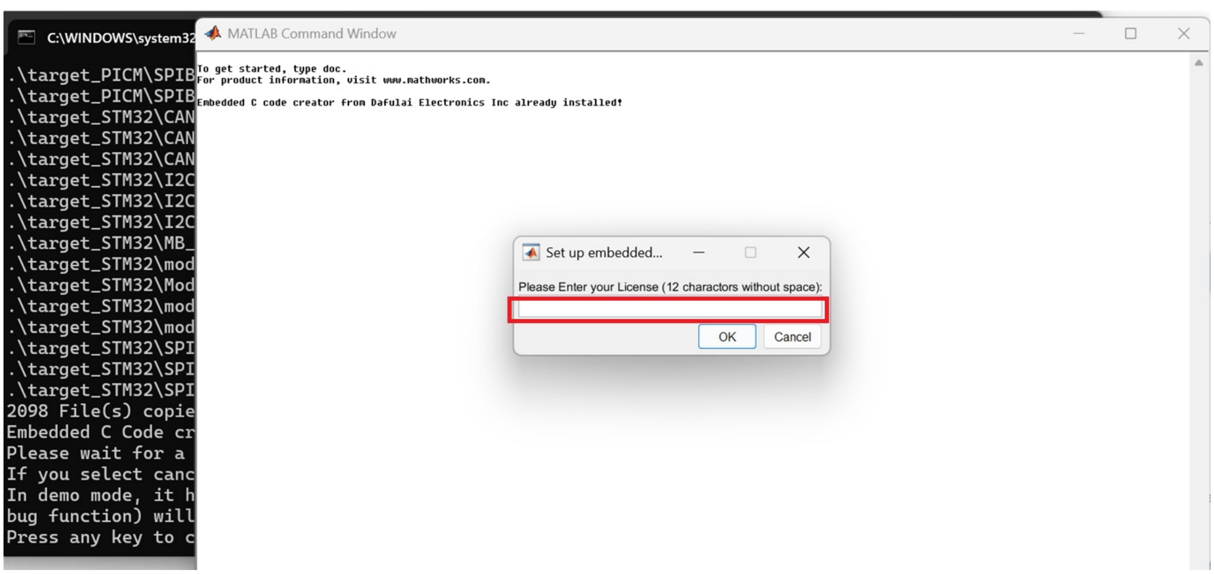
- **Step1** Download Simulink Embedded C Creator library from clicking EmCreator.zip
- **Step2** unzip EmCreator.zip to any directory.
- **Step3** double click on setup.bat. It will display window below:



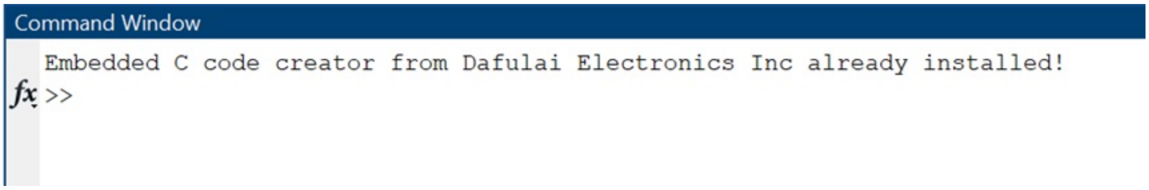
Following the instructions above cmd window, input your directory name you want to install, please use full directory including disk drive name such as "C:\myDir" (without quotation marks).

- **Step4** If you have debug/monitor license, please enter 16 characters license Keys as windows below. Or if you have no license, just click "Cancel" button, Target will work for Modbus only 15 minutes demo time.

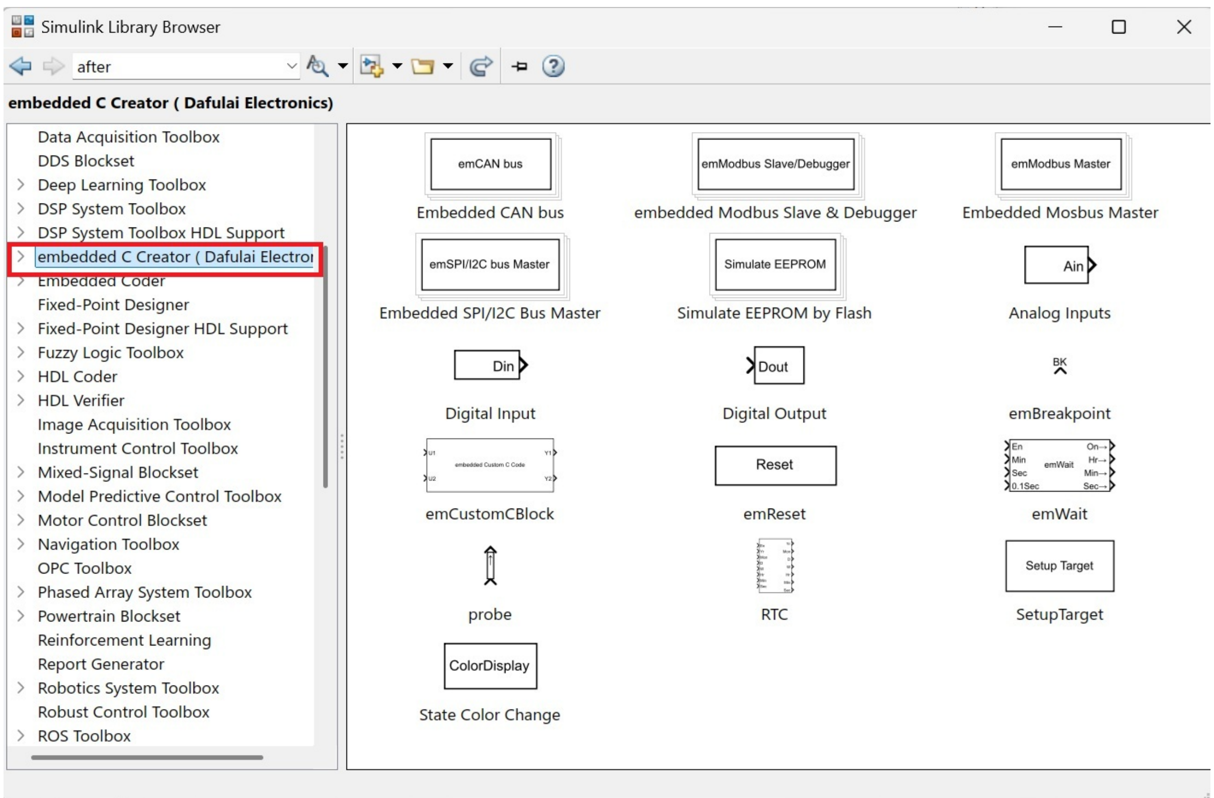
  You still have chance to setup debug/monitor license after installing software. You just right click in empty space of your target Simulink model, it will popup menu with item "Set up Dafulai Embedded C creator License". Click this item, it will setup license keys.



After you finish library installation, please start Matlab. You will see Message below in Matlab command window:



If you saw the above information, Simulink Embedded C Creator library has been installed in your computer successfully.
In your Simulink Library browser, you will see our embedded C creator library as shown as the following screenshot:



There are 5 sub-blocks: "emCAN bus", "emModbus Slave/Debgger", "emModbus Master", "emSPI/I2C Master", "Simulate EEPROM".
You can click these sub-blocks to open relative blocks.

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

6 How to use embedded C creator Library in embedded project

All blocks in our embedded C Creator library are non-blocking except "simEEPROMWriteFlash" block (Flash erase/writing operation). It is good for non-OS embedded system (bare metal programming). Our embedded creator is based on "MCC/Harmony" for Microchip MCU or "STM32Cube" for STM32 MCU or "SysConfig" for TI C2000 DSP.
So the first step is to set up peripheral driver code by these configuration tool software, and compile it. Make sure no any compile error. If any error occurs, please contact MCU company to get technical support. For TI C2000 SysConfig platform, "undefined symbol " link error is allowable if the symbol is ISR. The reason is TI SysConfig does not provide ISR code. However, after build our Simulink model, this link error will disappear.
The second step is that we set up Simulink model by our embedded Creator library. And build it. If any error occurs, it will display Yellow color for error block and error message. Please modify block or adjust peripheral configuration settings.
If no any error, our embedded creator will open embedded system IDE software automatically if IDE is not running, and you will programming code into target by emulator or other ICE/ICD. If IDE is running before build Simulink model, created code will occur in your IDE Editor. Notes: you must save all manual modification to code before build Simulink model if you already modified code manually, otherwise inconsistence will occur between created C code and your manually modified C code. Never manually Modify C code which is created by Simulink even though it is comment if you don't know what created C Code does.
The last step is that we run Simulink in PC to view result of embedded system.

For the first step, you must build it by IDE. We can find bugs in peripheral driver code. Otherwise, you build it in step2, you will mix the Errors from peripheral driver code and Simulink created code. It is not good for troubleshooting.
Here we don't tell you how to use IDE to configure peripheral driver code. You can go to MCU/DSP company website to know how to use IDE and peripherals configuration software.

Notes: *1 Tick for embedded Simulink model uses embedded peripheral timer1 as clock source in Michrochip platform, you must enable timer interrupt and create interrupt every 1ms by this timer. No matter whether embedded target peripheral number is 0-based or 1-based, we always use timer1. For Microchip PIC32 Arm cortex, we use TCC1 as timer firstly. If no TTC1, we will use TC1 Channel 0 as tick. For STM32Cube platform, Tick for embedded Simulink model uses SysTick as Timebase 1ms source. For TI C2000Ware SysConfig platform, Tick for embedded Simulink model uses CPU Timer0 as 1ms Tick source.*

2 *In our embedded Simulink library, all blocks use 1- based peripheral number if block has peripheral number such as Uart or CAN or SPI/I2C. Therefore, when MCU/Dsp uses 0-based peripheral number, Peripheral 1 in Simulink block actually means peripheral 0 in embedded target configuration tool software. When MCU/Dsp uses Alphabet-based peripheral number (Usually TI DSP), Peripheral 1 in Simulink block actually means peripheral A in embedded target configuration tool software.*

3 *The created C code of Simulink is in folder: "Your embedded project directory used in configuration" \ simulink_generated_files. However, your previous created C code of Simulink is in folder: "Your embedded project directory used in configuration" \ simulink_generated_files_backup for Microchip MCU and "Your Computer User directory" \ Simulink Model name (without suffix slx) \ simulink_generated_files_backup for STM32 and TI C2000 DSP. And the one before the previous created C code of Simulink is in folder: "Your embedded project directory used in configuration" \ simulink_generated_files_backup_backup for Microchip MCU and "Your Computer User directory" \ Simulink Model name (without suffix slx) \ simulink_generated_files_backup_backup for STM32 and TI C2000 DSP. So you can find your previous created code*

4 *In order to store your project, the minimum files for your project are simulink slx file and Platform configuration file such as .MC3 for microchip and .ioc for STM32Cube and .syscfg for TI C2000 DSP even though you uses emCustomCBlock. However when you use emCustomCBlock, after complete debugging, you must execute Simulink build command to let emCustomCBlock keep your C source code when you use minimum files to store your project.*

5 *For microchip SAM MCU, we only support MCUs with SerCom peripheral.*

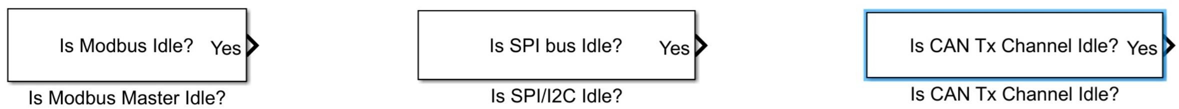
6 *For TI C2000Ware SysConfig platform, please use Flash code version to debug. Don't use RAM code version to debug.*

.

If you want to watch signal value of Simulink model, you cannot directly use "Display" block to watch, you must use "emProbe" block (embedded Probe) except output/input port name contains "→".
"Display" block is from standard Simulink library: Simulink/Sinks/Display.
Input port of emProbe block connects signal you watch, Output port of emProbe block connects "Display" block.
Any Port name with "Right Arrow" (→) symbol contains "built-in" probe. So in PC side, you can watch its value directly by "Display" block, you don't need add "emProbe" block before "Display" block.

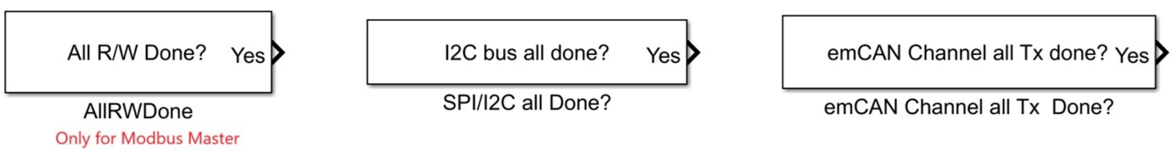
For Simulink blocks from Modbus Master, SPI/I2C master, and CAN bus transmit, we use a special method to access.
Firstly, all operations (except for CAN Bus receiver) must be in "Idle" State . For the first time after

embedded system power on, it is in "Idle" state. You can use the following blocks to know whether it is in "Idle" State.

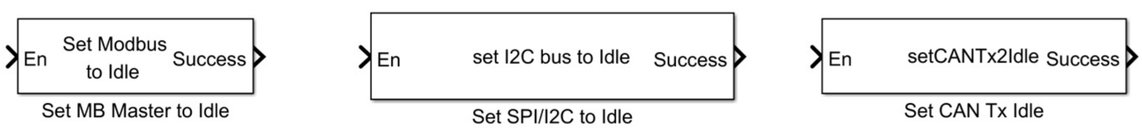


When output "Yes" of these blocks above is true, you can enable related bus access.
All access operations are executing in sequence. Sequence with small number will execute firstly.
For specific access operation blocks, please see each block's help and example.

After all operations done (All sequences done), you must set it to "Idle" state in order to operate normally for next time. You can use the following blocks to know whether all operations are done.



When output "Yes" of these blocks above is true, all sequences are done. You must use the following blocks to set it to "Idle" state:



The input "En" must connect "Yes" output of block " XXXX done?" .

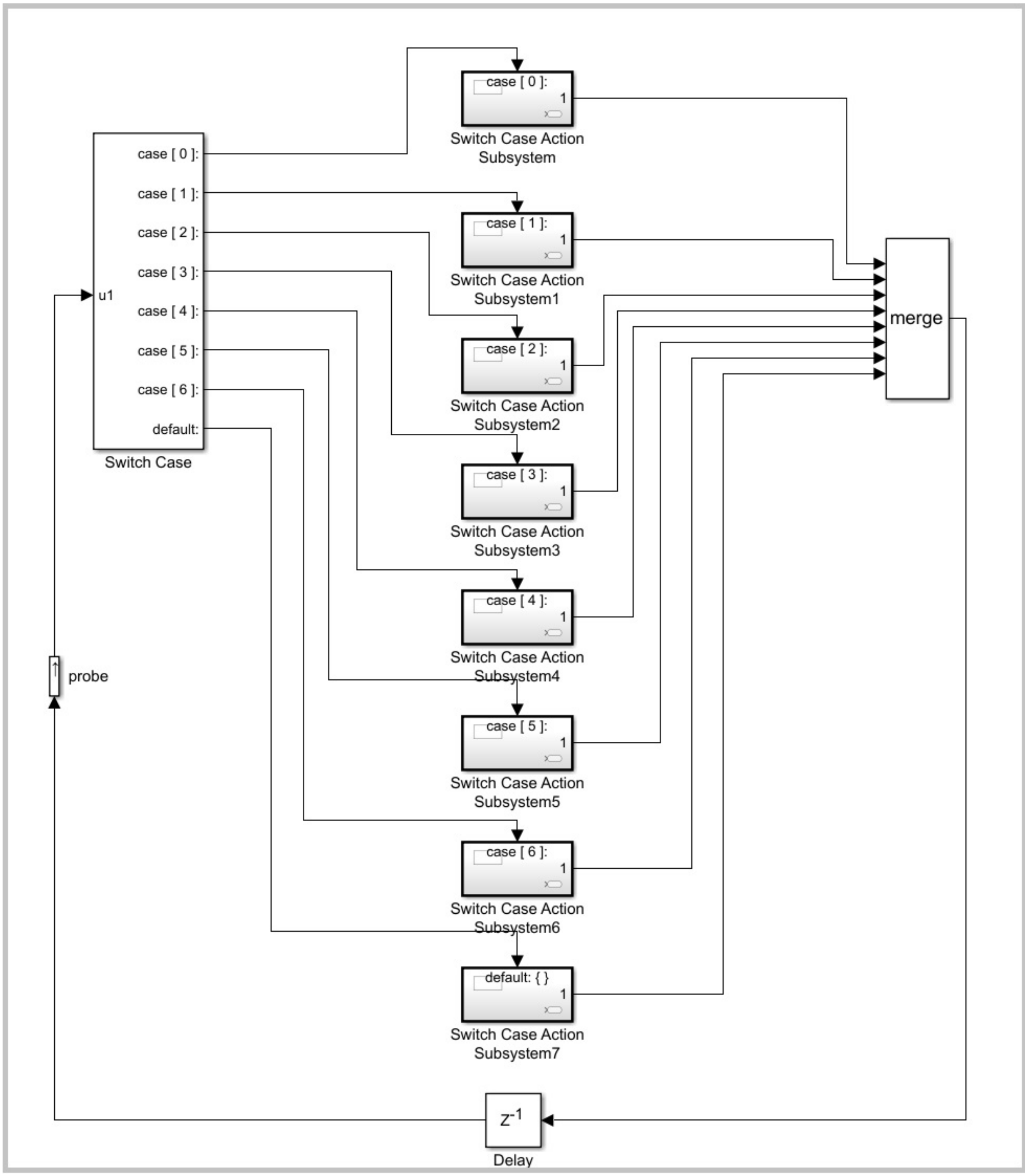
For specific example, please see help of block. (Double click block, pop up dialog, and click help button).

Our code is for " bare metal programming" which means embedded target has no any operating system.
In order to implement Sequence Control of application, we need set up "Finite State machine". In general, "StateFlow" can provide "Finite State machine" function.
However, "StateFlow" cannot provide state executing visibility, and time condition in "StateFlow" is not real-time, it is "Simulated time".

Here we introduce "Pure Simulink block" method to implement "Finite State machine".

We use "Switch case" block for "State variable" which is input of "Switch case" block". This "State variable" generally is uint8 type of scalar in order to save resource of embedded target memory.
The each output of "Switch case" block connects action port of "switch case action subsystem" block which implement specific function for this state.
The output of "switch case action subsystem" block is a uint8 type of scalar which is value of next "State variable".

The entire State machine Simulink blocks structure is shown below:



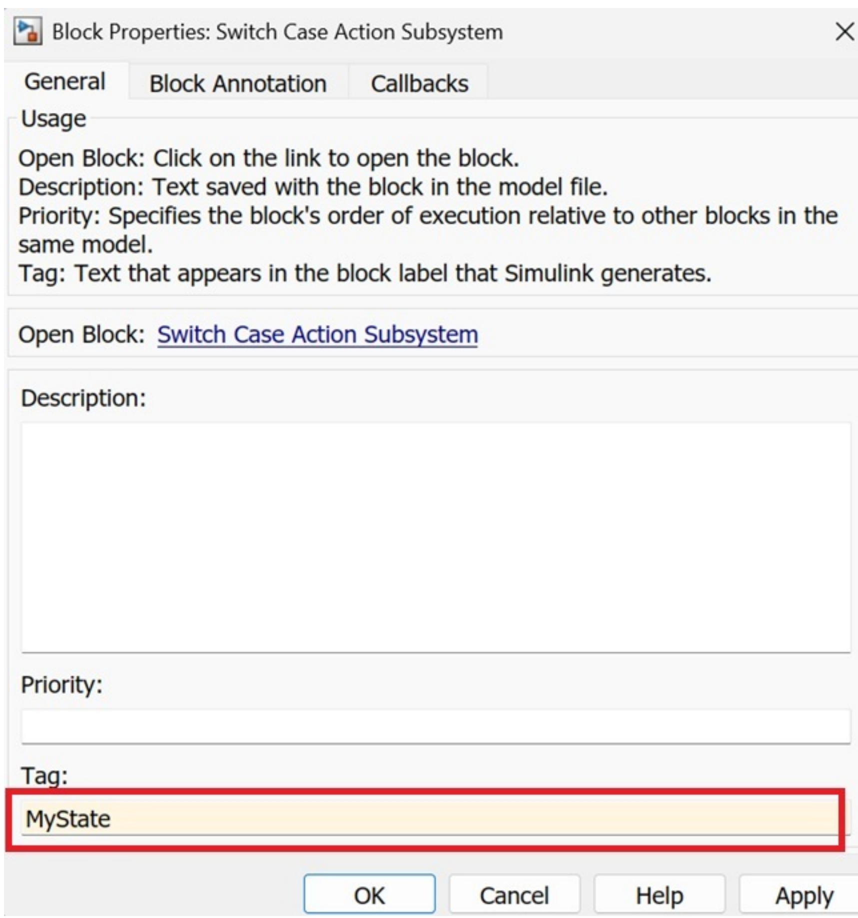
***Notes**: State variable "u1" in "Switch Case" block must be connected by "Probe" output in order to monitor State Machine in PC. Otherwise you cannot get actual value from embedded target.*

Each "Switch Case Action Subsystem" will run from up to down if we set up output value to case value plus 1 when state transition condition is meet. It looks like "PLC Ladder Logic". But it is different from it.

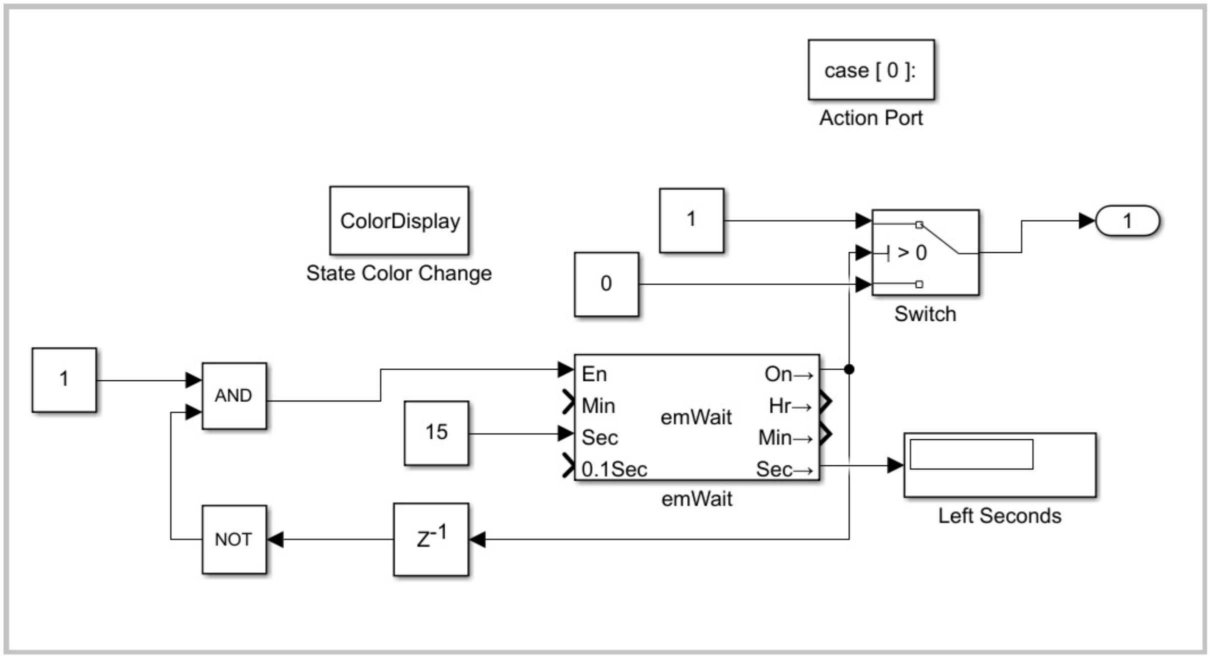
"PLC Ladder Logic" runs all "rows" each scan period. But our "Switch Case Action Subsystem" only run one "Switch Case Action Subsystem" during each sample time.
It is easier to program than "PLC Ladder logic" in this way.

In order to implement "Running State" visibility, we must set up the same Tag Value for all "Switch Case Action Subsystem" blocks and we must add one "ColorDisplay" block inside all "Switch Case Action Subsystem" blocks.
All "ColorDisplay" blocks are the same. You can copy/paste from existing one.

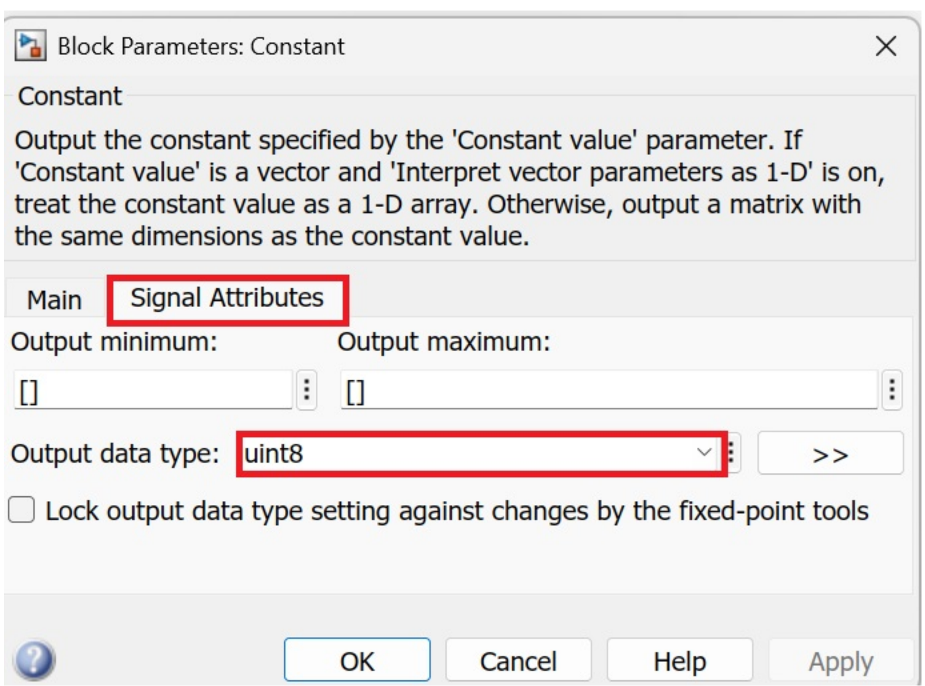
Right click on each "Switch Case Action Subsystem" block, popup one menu, and click on menu item "Properties...", you will see dialog window below:



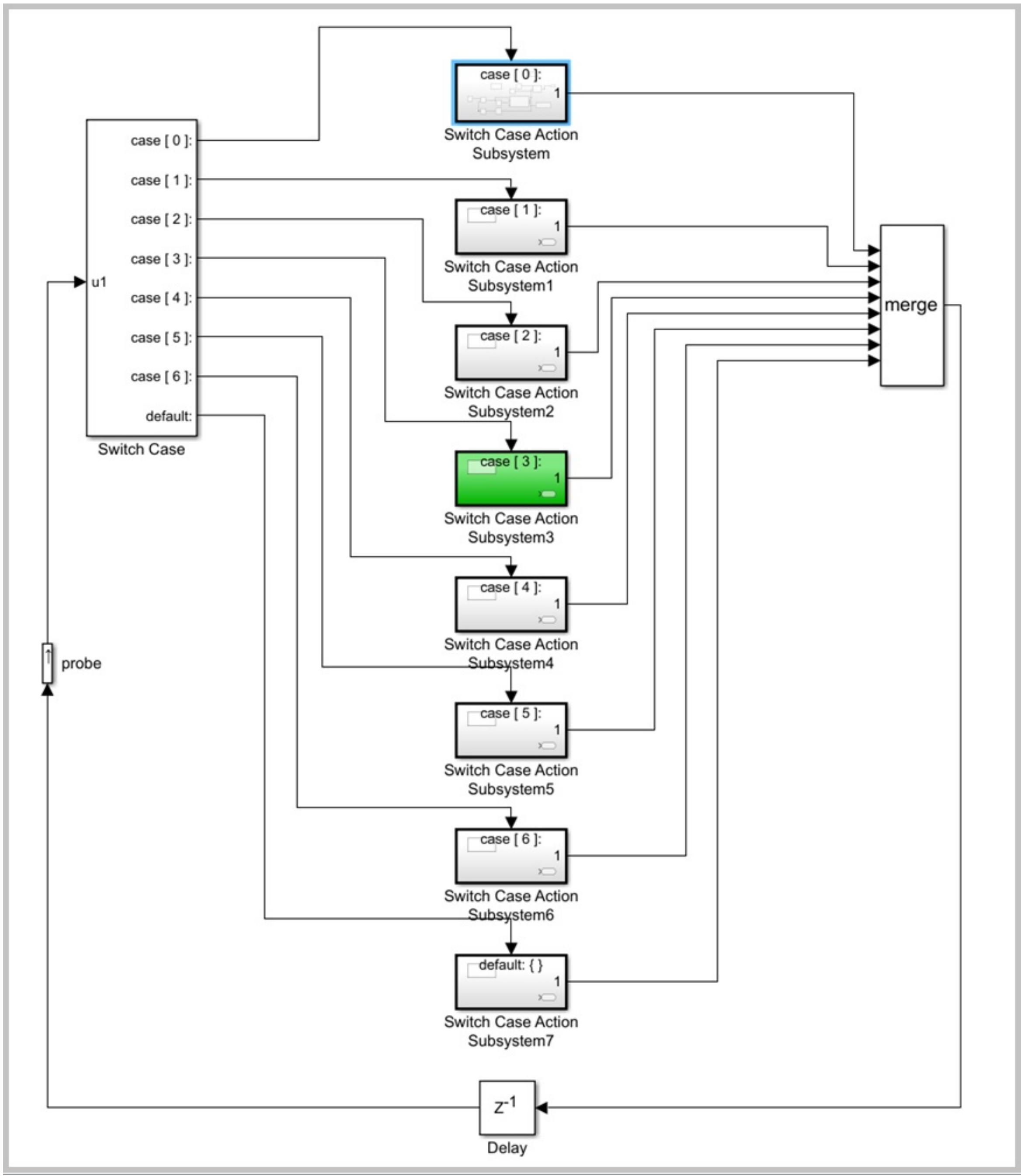
In our example above, I used "MyState" as Tag value. You can use any other valid characters. But all "Switch Case Action Subsystem" blocks for the same state machine should keep the same Tag.
For each state behaviour implementation, please double click on each "Switch Case Action Subsystem". For example, in "case 0", we want to delay 15 seconds to enter next state "case 1" automatically. Please see Simulink blocks below:



In above Figure, input constant value 0 and 1 of Switch block is very important. It decides current state and next state, and its type decides "state variable u1" type. In embedded system, in order to save memory resource, we generally use "uint8" as "state variable u1" type. So constant 1 and 0 of "Switch" block's input should use "uint8" type. Please double click on constant block and follow the picture below to change data type:



When run this State machine, we can see green color for "Activated State". The following Figure shows "Case 3" State is running.



7 How to manually put blocks into ISR

Usually, all blocks run in "dead-loop" in main() function for embedded target.

ADC block can run in interrupt (ADC Done interrupt Service Routine) by setting block parameter. All blocks with sample time inherited from ADC Block will run in ADC ISR or DMA ISR automatically if this

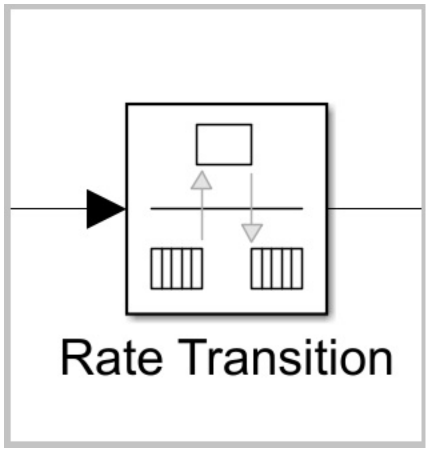
ADC block enabled interrupt or DMA mode. You don't need to modify code manually.
If you want blocks with specified sample time run in interrupt, and the interrupt is not ADC or DMA driven by ADC, you should manually change the embedded target C Code.
How? Firstly you have to know "C code position" for each "sample time" blocks' period executing parts.
Please see figure below for main() function structure:

```
int main()
{
    ...
    ...
    while(1)  {
        // Add your application code
        //simulink add code2----start
        ...
        ...

        if (tick!=tick_old)
        {
            //1ms running
            tick_old=tick;
            ...
        }
        //simulink add code2----end
        //simulink add code3----start
        if (tick-tick4Simulink0>=500U)
        {
            tick4Simulink0=tick;
            Simulink_model_name_step0();
        }
        if (tick-tick4Simulink1>=1500U)
        {
            tick4Simulink1=tick;
            Simulink_model_name_step1();
        }
        if (tick-tick4Simulink2>=3000U)
        {
            tick4Simulink2=tick;
            Simulink_model_name_step2();
        }
        //simulink add code3----end
    }
    ...
    ...
}
```

In above C code, Simulink blocks with the same "Sample Time" will be function call inside red color of rectangle between "//simulink add code3----start" and "//simulink add code3----end". Simulink Function call name is "Simulink model name" plus "underscore" plus "step" plus "0 or 1 or 2 or 3 or 4...."
If your Simulink model only contains one "Sample Time", Function call name is "Simulink model name" plus "underscore" plus "step".

Sample Time is Digital number with suffix "U" inside red color of rectangle. Sample Time unit is "millisecond".

If you want to put one " Simulink Function call " with specified "Sample Time" into ISR, just comment out this "Simulink Function call " by double slash "//", and put this "Simulink Function call " into your ISR or ISR's callback.
One important thing is "Variables" shared between "ISR" code and "Non-ISR" code must use "volatile" qualifier.
And shared signals between different "Sample Time" must use "Rate Transition" block below:



If this shared variable's bit width is greater than CPU data width, you must disable interrupt before accessing this shared variable and enable interrupt after accessing this shared variable in non-ISR code in order to keep data integrity.

8 Simulink Embedded Blocks

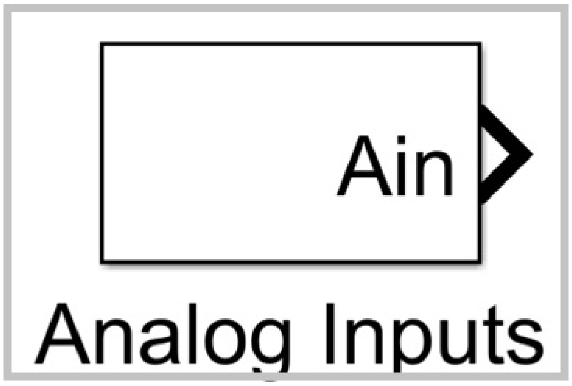
Now we explain all Simulink embedded blocks.

The following blocks are in top directory of library. We list them according to alphabetical order.

Ain

Analog Inputs
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Ain



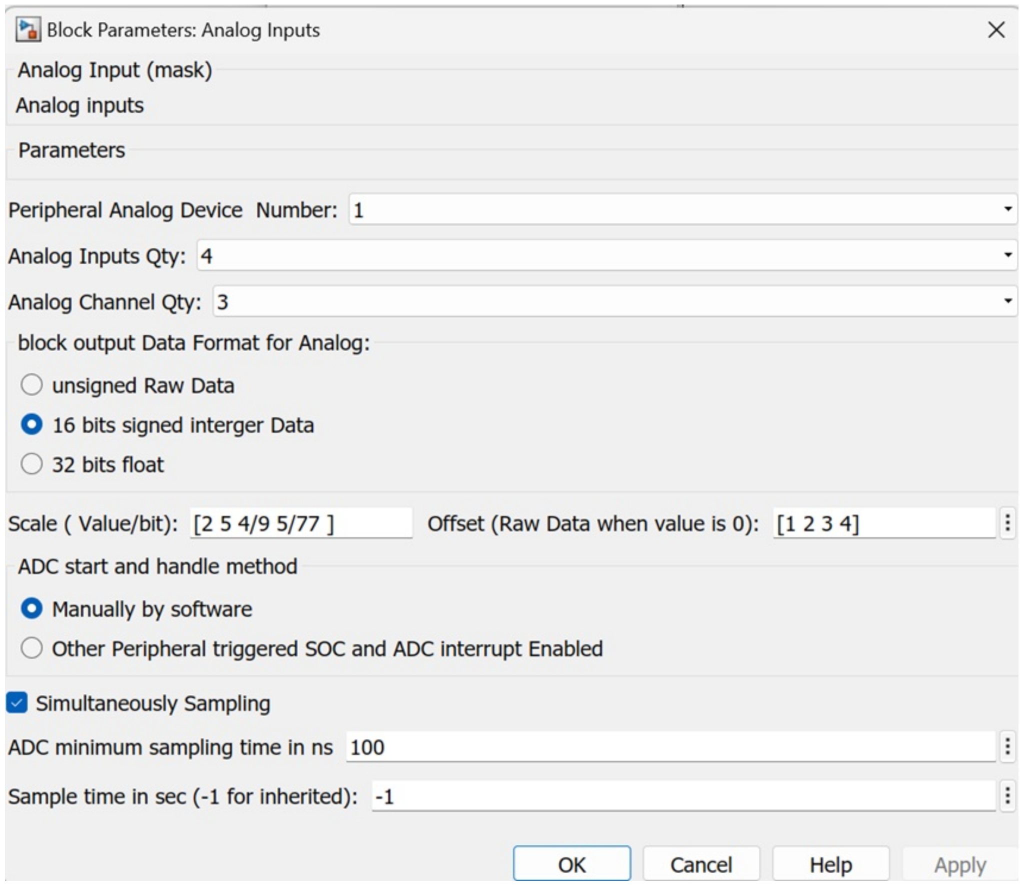
Description

This block will do Analog to Digital Conversion..

> ***Notes**: PIC32 Arm platform does not support this block. You should use "emCustomCBlock" to implement your ADC*

Parameters

Please double click this block to open parameters dialog below (when platform is PIC24/Dspic30/Dspic33):



Let us explain parameters.

- Peripheral Analog Device Number: — drop list (1 or 2). It is ADC Peripheral Number of Micro-controller.

- Analog Inputs Qty — drop list from 1 to 16 which is total analog input quantities.

- Analog Channel Qty — One ADC Peripheral has multiple channels which are able to sample at the same time or individually at different time. This parameter will set up channel QTY for this ADC Peripheral.

- block output Data Format for Analog — radio selection button for analog output data format. It can be one of "unsigned Raw Data"/"16 bits signed integer Data"/"32 bits float". Please pay attention, it is not the "Data format" in IDE GUI configuration. You must choose "unsigned integer" (maybe 10 bits, or 12 bits or 16 bits according to ADC Peripheral) in IDE GUI configuration (It is MCC for Microchip PIC24/Dspic30/Dspic33)

- Scale (Value/bit) — integer or fraction or float type of vector. Length of vector is equal to "Analog Inputs Qty". Every item of vector denotes "physical unit value per bit" for each related analog input. For example, [2 1/3] denotes "Ain1 physical Value = Ain1 ADC raw value x 2, and Ain2 physical Value = Ain2 ADC raw value x 1 / 3 ". If your parameter "block output Data Format for Analog" is "16 bits signed integer Data", we propose you use fraction type of Scale instead of float type of Scale because all calculations will be integer. It will decrease CPU time specially for Micro-controller without hardware FPU.

- Offset (Raw Data when value is 0) — integer type of vector. Length of vector is equal to "Analog Inputs Qty". Every item of vector is ADC raw data value when physical value is zero for each related analog input.

- ADC start and handle method — radio selection button for ADC start (SOC signal) and handle method. It can be one of "Manually by software"/"Other Peripheral triggered SOC and ADC interrupt Enabled"/"Other Peripheral triggered SOC and DMA Enabled" . For "PIC24/ Dspic30/Dspic33" and "PIC18", you cannot select "Other Peripheral triggered SOC and DMA Enabled". When you select "Manually by software" , ADC interrupt is disabled. If you select "Other Peripheral triggered SOC and ADC interrupt Enabled", all blocks, which are inside the same sample period as this ADC block, will be called inside ADC interrupt ISR. And "this sample time for these blocks" cannot decide "actual sample period", "Actual sample period for these blocks" is decided by "ADC interrupt". So please use IDE to configure and create One ADC interrupt when all Analog inputs A/D conversions finish.

- Simultaneously Sampling — logical type of scalar. It only makes sense when multiple channel S&Hs or ADC Cores. It means all channels are sampled at the same time in one SOC signal (Start SOC signal). So you just like taking a picture shot to all channels Analog inputs.

- ADC minimum sampling time in ns — integer type of scalar. It denotes minimum sampling time for any analog input in unit nano-seconds. Minimum number you can input is 10. You must match the parameter value in micro-controller ADC data sheet.

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Notes: *1 Please configure Analog Peripheral. All settings must be done in IDE GUI configuration (It is MCC for Microchip Micro-controller). And it must match this block settings, otherwise unexpected result will occur.*
2 For DsPic33EP, you must make sure creating one ADC interrupt when all Analog inputs complete conversion if you enable ADC Interrupt.
3 When you use ADC Interrupt, all input/output data from other sample time blocks to interrupt Blocks, must use RateTransition block which has -1 as output sample time (inherit). These periodical blocks which connect interrupt block must has 1 to 10 times or 15 times or 20 times or 25 times or 30 times or 40 times or 50 times or 60 times or 70 times or 90 times or 100 times or 200 times or 300 times base sample time.
4 When you use ADC Interrupt, it will run Ain block and blocks which connect Ain block with sample time=-1 (inherit) once when all analogs complete conversion.

Ports

Input

None

Outport

- Ain — Scalar or Vector. Data type is set by parameter "block output Data Format for Analog" (uint16/ int16/single) . Vector length is equal to parameter "Analog Inputs Qty"

ColorDisplay

Change parent subsystem block background color to green when parent Action-subsystem is running.
It is used in FSM (Finite State Machine)
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / ColorDisplay

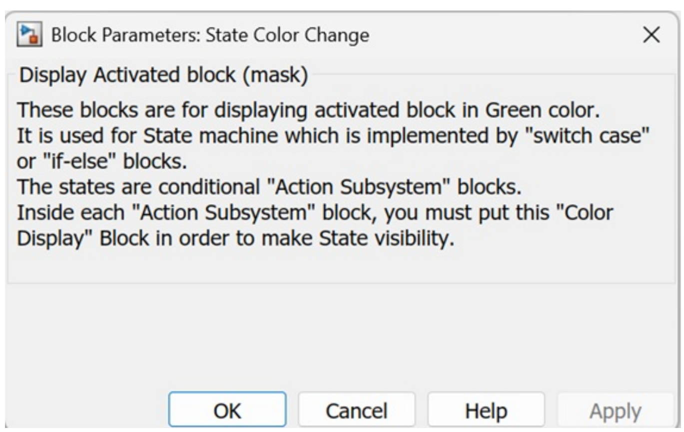
Description

This block is used in "Finite State Machine" visibility when running.
In pure Simulink block system (It means no StateFlow block), How to implement a "Finite State Machine" ? We generally use one "Switch Case" block and multiple "Switch Case Action Subsystem" blocks which denote each "State". Of cause, you can also use one " If " block and multiple " If Action Subsystem" blocks if state is not too many.

You just put this "ColorDisplay" block into each "Action Subsystem" blocks. When you use PC to monitor system, you will see active state becomes "Green Color" when "Action Subsystem" block is running.

Parameters

Please double click this block to open parameters dialog below:



Ports

Input

None

Outport

None.

Examples

Example:

Our embedded platform is dsPIC33EP256GP502 . Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. Our embedded Modbus Server ID=1.

This example function is "Sequence control".

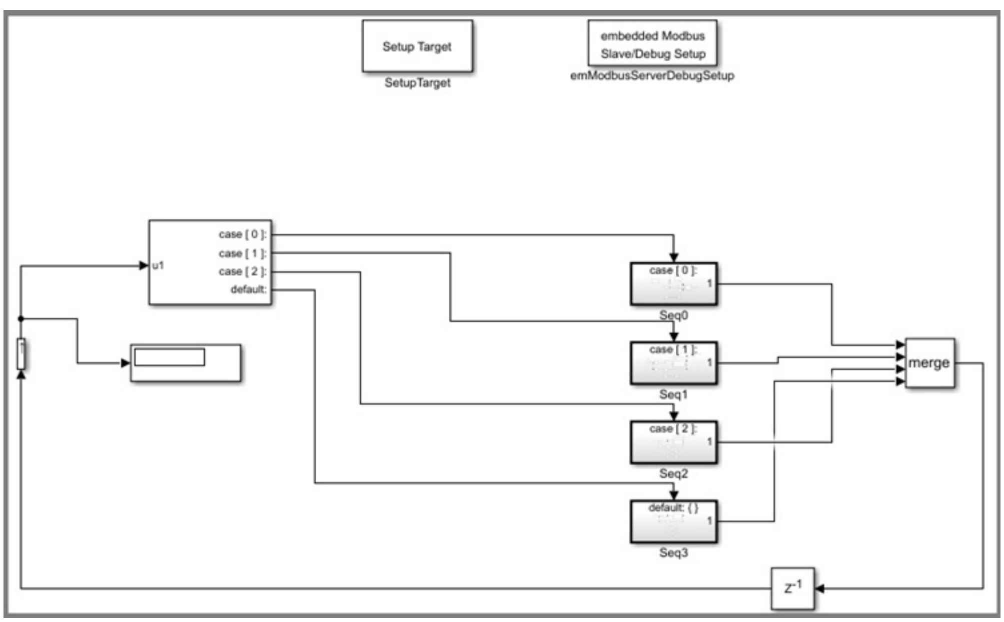
Seq0: When power on, Digital out "RB12", Digital out "RB13", and Digital out "RA4" keep logic 0. Target MCU waits for Digital Input RB7. If RB7 becomes logic 1, MCU will enter next Sequence called "Seq1".

Seq1: Digital out "RB12" keeps logic 1. Timer start to count time elapsed. If 12 seconds elapse, Target MCU will enter next Sequence called "Seq2".

Seq2: Digital out "RB12" keeps logic 0. Digital out "RB13" keeps logic 1. Timer start to count time elapsed. If 12 seconds elapse, Target MCU will enter next Sequence called "Seq3".

Seq3: Digital out "RB13" keeps logic 0. Digital out "RA4" keeps logic 1. Timer start to count time elapsed. If 12 seconds elapse, Target MCU will return to initial Sequence called "Seq0".

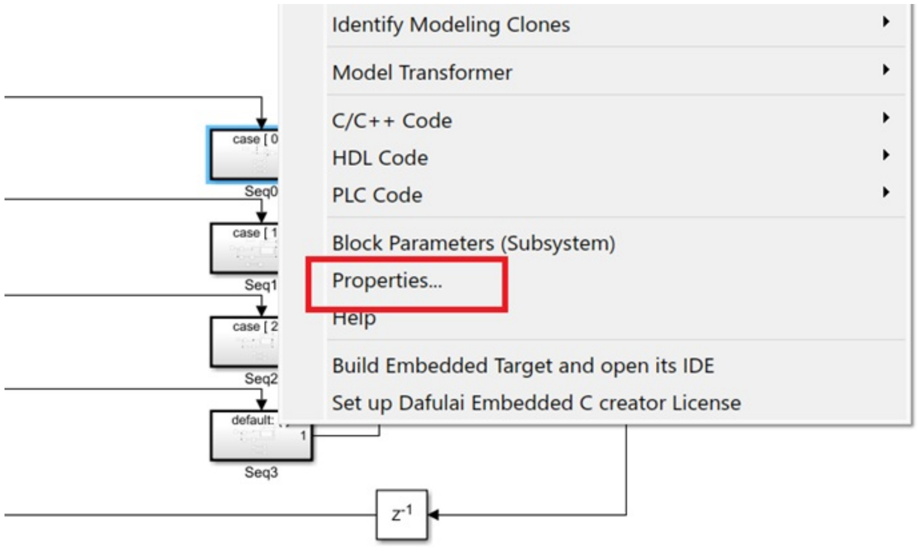
Please see screenshot of model below:



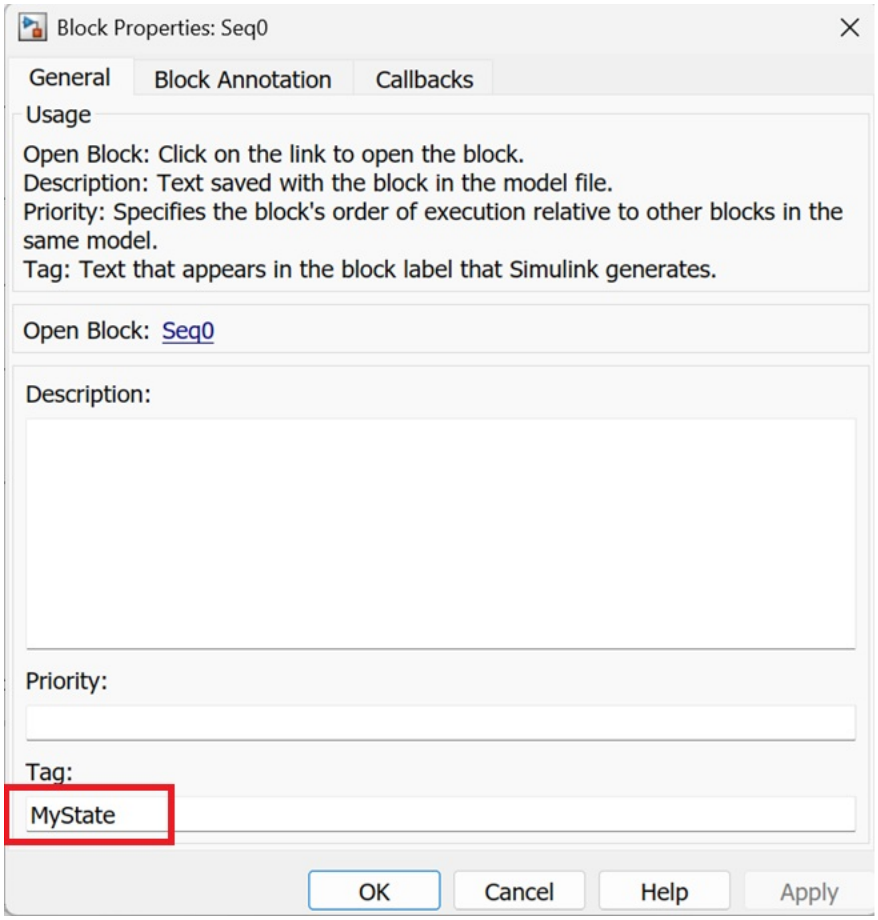
Notes: *Input port u1 of "Switch case" block is "State Variable", You must use output of*

"Probe" to connect it in order to monitor "State Variable" value of "Finite State Machine" in PC side.

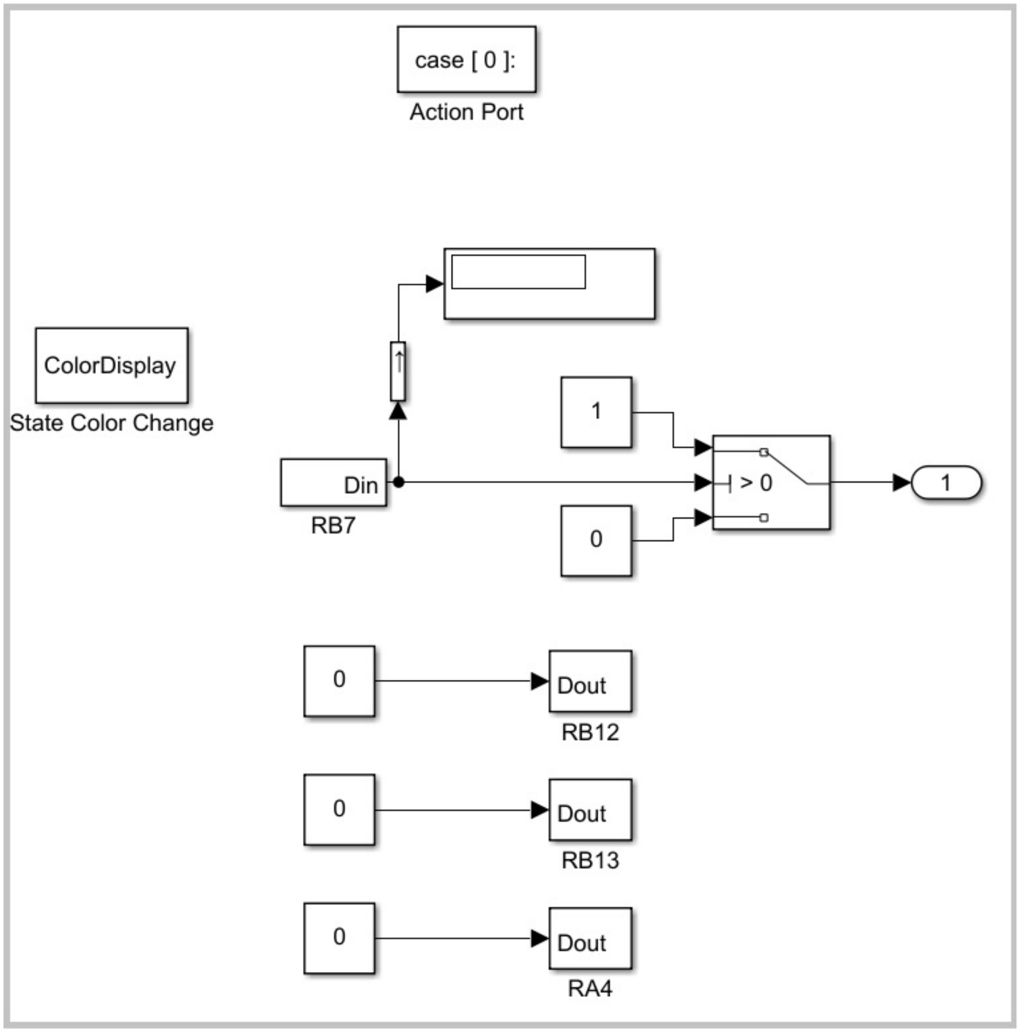
We set up Tag of all "case action subsystem" by right clicking "Properties" on each "Action Subsystem" such as the following picture:



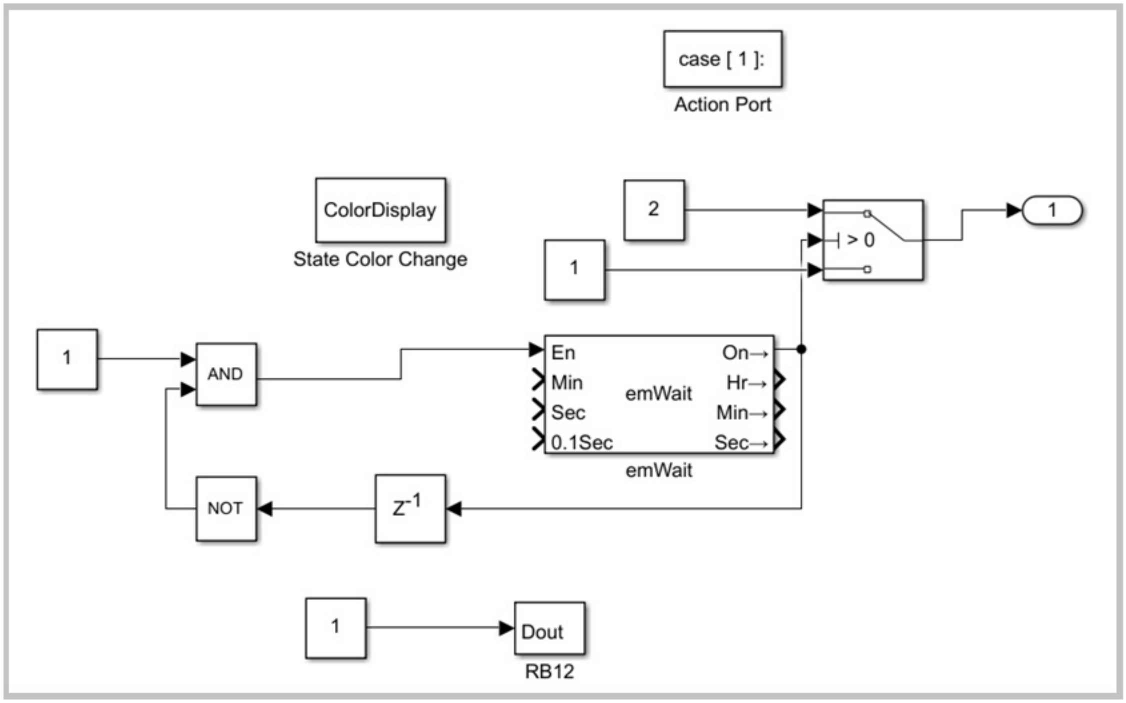
The Tag value must be the same value such as below:



"Case 0 Action Subsystem" , which called "Seq0", implements Function of "Seq0". Double Click "Case 0 Action Subsystem", you will see the implementation below:

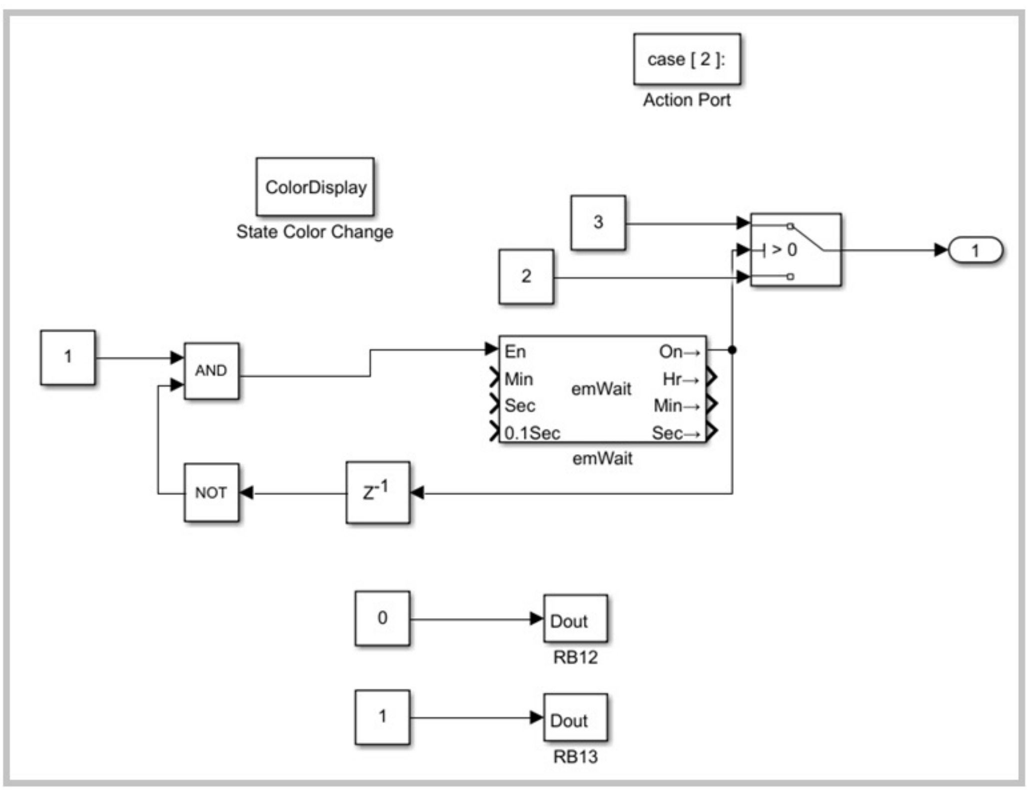


"Case 1 Action Subsystem" , which called "Seq1", implements Function of "Seq1". Double Click "Case 1 Action Subsystem", you will see the implementation below:



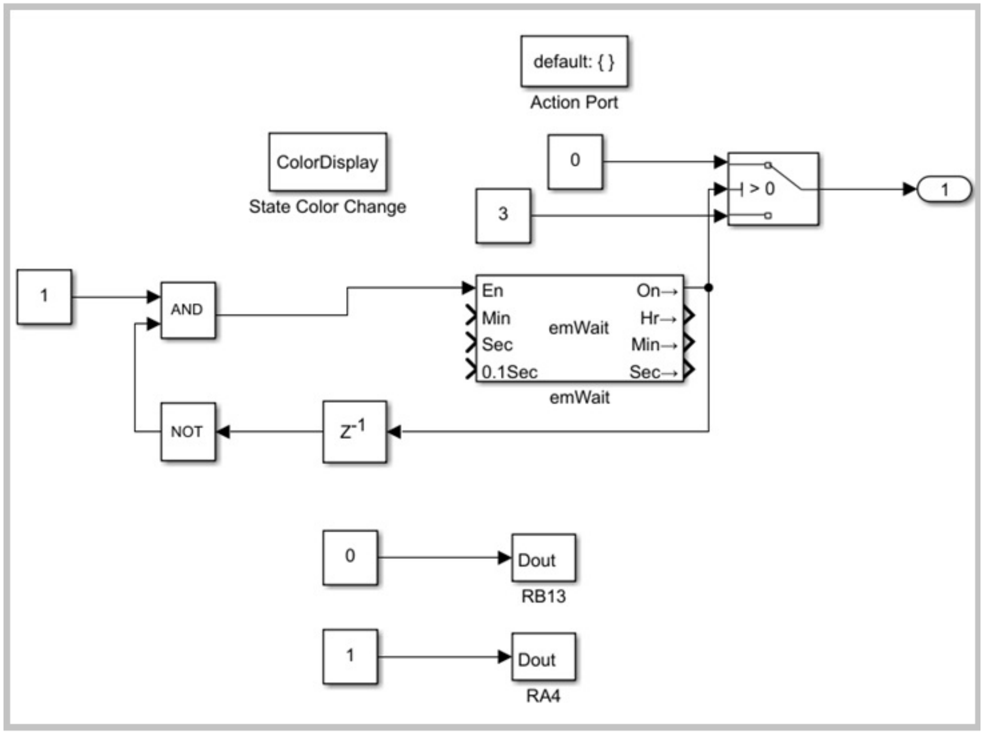
The block "ColorDisplay" is the same as counterpart of Seq0. You just use Copy/Paste.

"Case 2 Action Subsystem" , which called "Seq2", implements Function of "Seq2". Double Click "Case 2 Action Subsystem", you will see the implementation below:



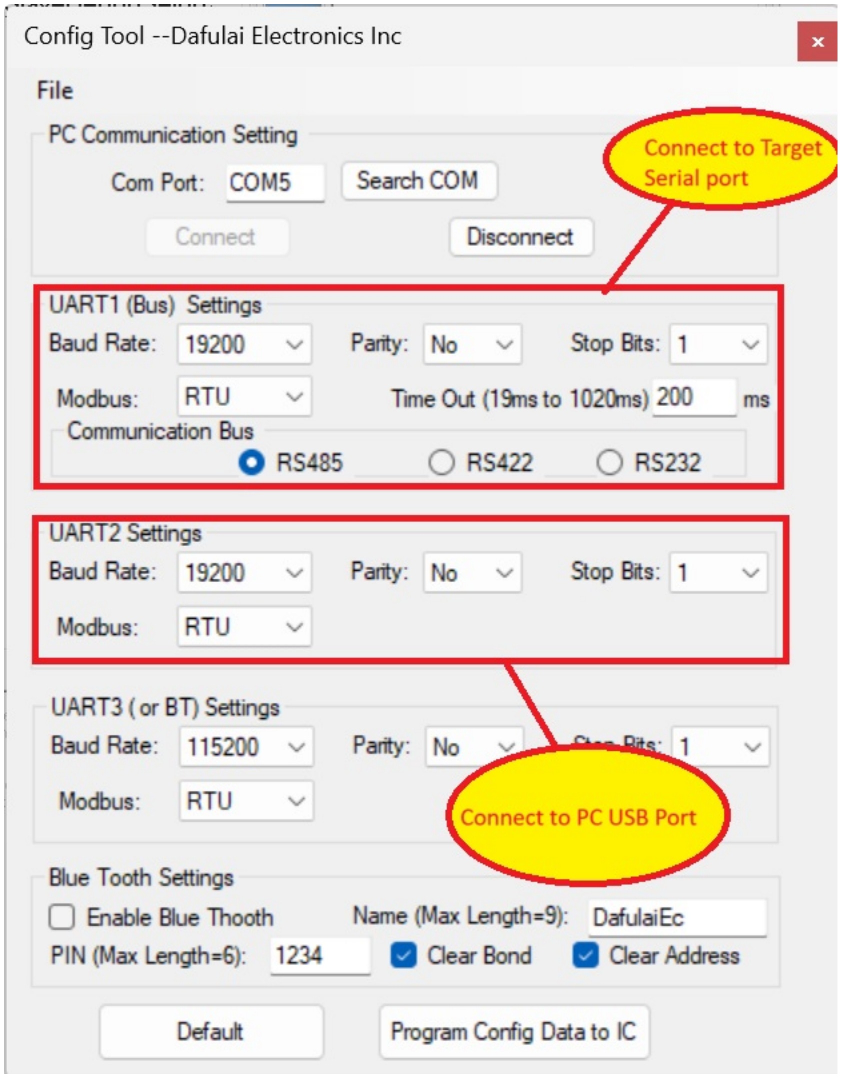
The block "ColorDisplay" is the same as counterpart of Seq0. You just use Copy/Paste.

"default Action Subsystem" , which called "Seq3", implements Function of "Seq3". Double Click "default Action Subsystem", you will see the implementation below:



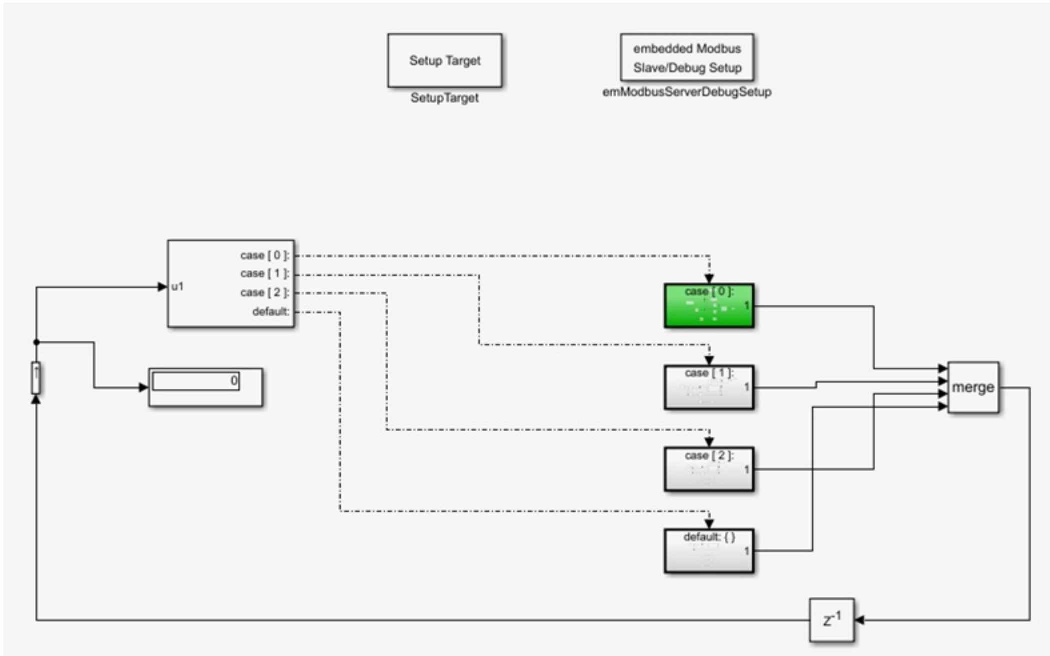
The block "ColorDisplay" is the same as counterpart of Seq0. You just use Copy/Paste.

Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:

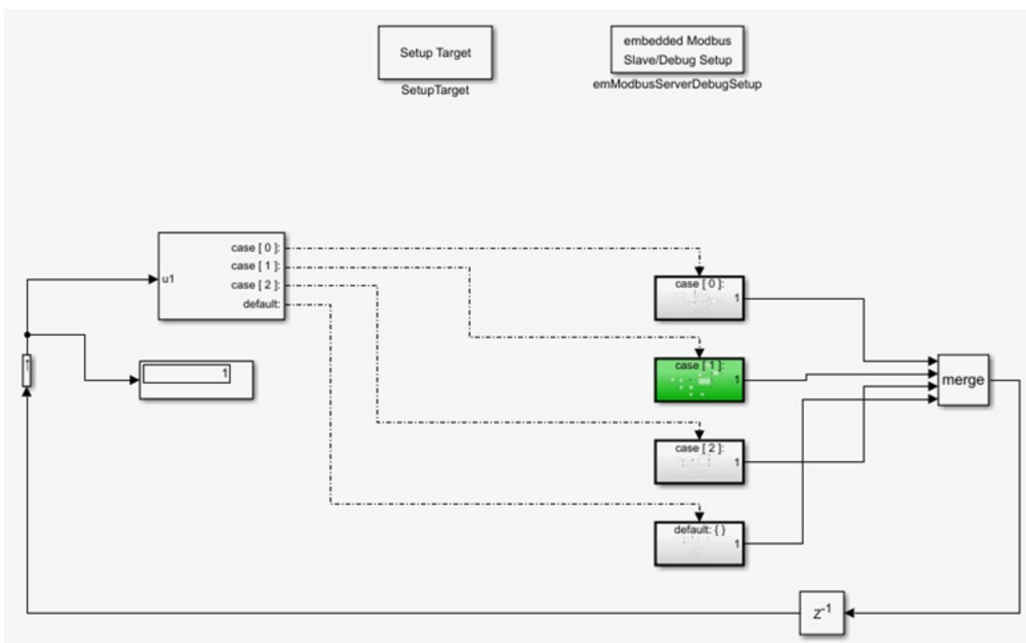


In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

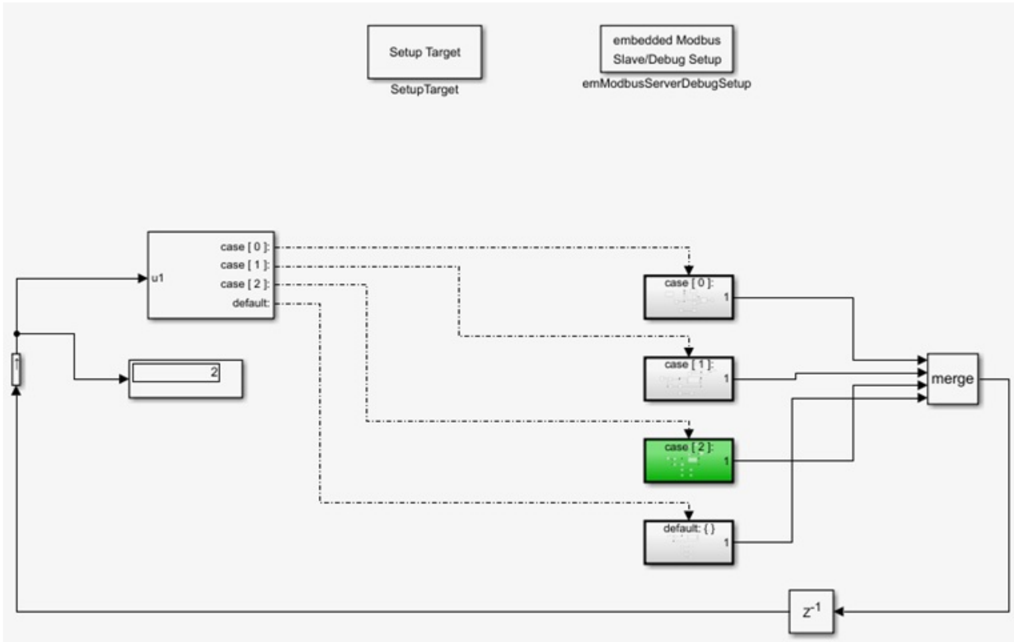
After you build this simulink model by right clicking "Build Embedded Target and open its IDE" on any empty space of model, you program the firmware to target by your emulator. You run simulink model, and you will see picture in PC below:



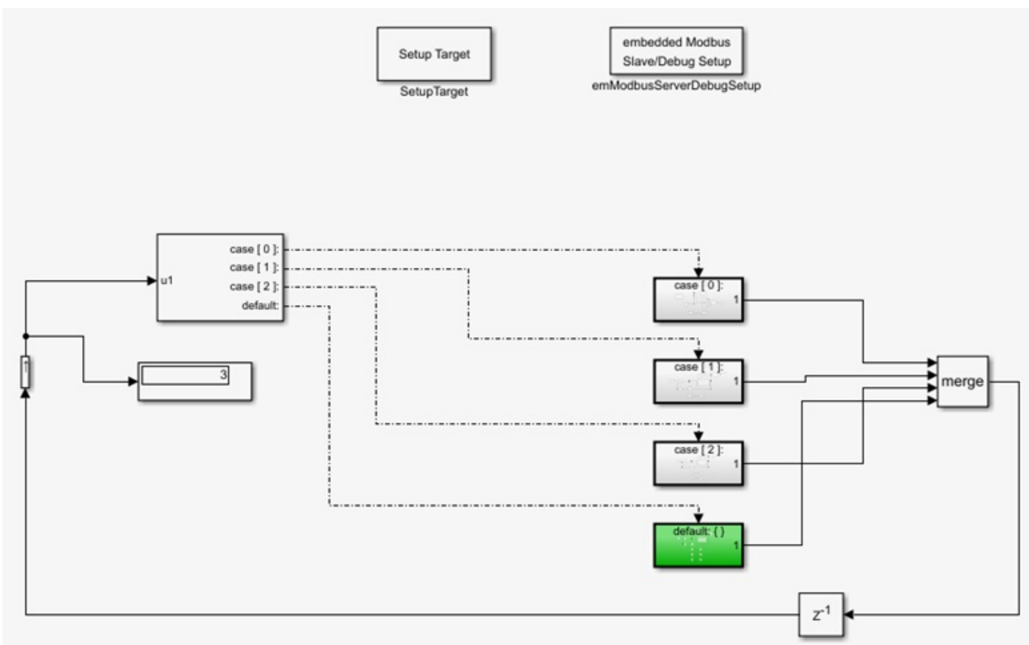
It means that target MCU is in "Seq0" state. Digital Out RB12, RB13, and RA4 all are Logic 0. When momentary button is pressed once which causes digital input RB7 logical 1, Target MCU enters next state "Seq1". Please see picture below:



In state of "Seq1", Digital out RB12 changes to Logic 1. And after 12 seconds, Target MCU enters next state "Seq2". Please see picture below:



In state of "Seq2", Digital out RB12 returns to Logic 0, Digital out RB13 changes to Logic 1. And after 12 seconds, Target MCU enters next state "Seq3". Please see picture below:



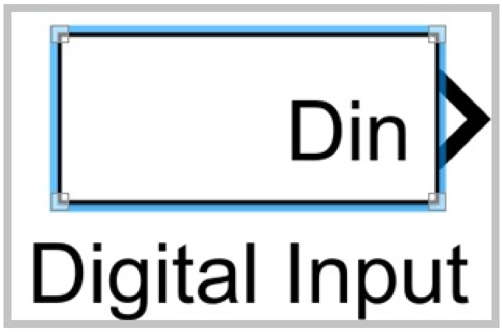
In state of "Seq3", Digital out RB13 returns to Logic 0, Digital out RA4 changes to Logic 1. And after 12 seconds, Target MCU enters initial state "Seq0" to wait for momentary button pressed again.

Please open "Your embedded creator library folder"/examples/example15_StateMachine.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

Din

Digital Input
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Din

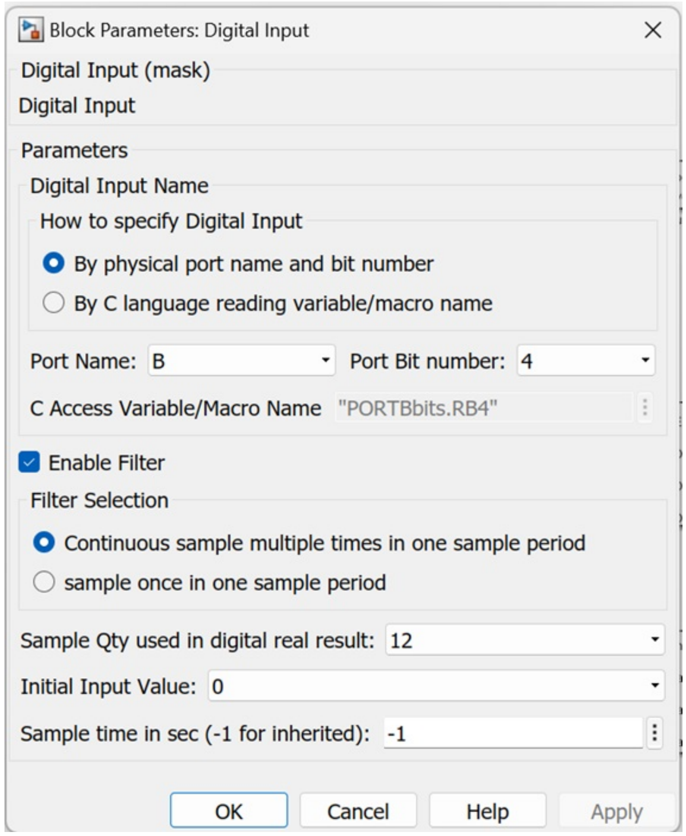


Description

This block will get digital Input.

Parameters

Please double click this block to open parameters dialog below :



Let us explain parameters.

- How to specify Digital Input — radio selection button for Digital input Pin. It can be one of "By physical port name and bit number"/"By C language reading variable/macro name".
- Port Name — drop list for Digital Port name. It is used only when parameter "How to specify Digital Input" is "By physical port name and bit number".
- Port Bit number — drop list for Digital Port Bit Number. It is used only when parameter "How to specify Digital Input" is "By physical port name and bit number".
- C Access Variable/Macro Name — String type of scalar. It can be variable name or macro name in C language for Digital Input. It is used only when parameter "How to specify Digital Input" is "By C language reading variable/macro name".
- Enable Filter — Logical type of scalar. True means "Enable Digital input filter".
- Filter Selection — radio selection button for Digital input Pin filter method. It can be one of "Continuous sample multiple times in one sample period"/"sample once in one sample period". If you select "Continuous sample multiple times in one sample period", firmware will read the same digital input many times continuously. And this block output "Din" only change when it is the same value for every time reading inside this sample period, otherwise keep old value. If you select "sample once in one sample period", firmware will read digital input

once in one sample period. However this block output "Din" only change when it is the same value for consecutive sample periods, otherwise keep old value. So the 1st method's filter causes signal delay to be smaller than the 2nd method's filter

- Sample Qty used in digital real result — Drop list from 2 to 12. Integer type of scalar. It only makes sense when Enable Filter. It means how many times readings consistent for filter .
- Initial Input Value: — Logical type of scalar. It only makes sense when Enable Filter. It is initial value for "Digital Input".
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Notes: *Please configure GPIO. All settings must be done in IDE GUI configuration (It is MCC for Microchip Micro-controller). And it must match this block settings, otherwise unexpected result will occur.*

Ports

Input

None

Outport

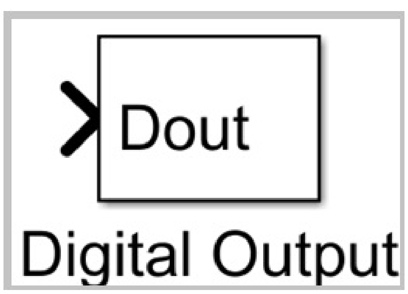
- Din — Logical Scalar. Digital Input Value.

Example: See "ColorDisplay" block example

Dout

Digital output
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Dout

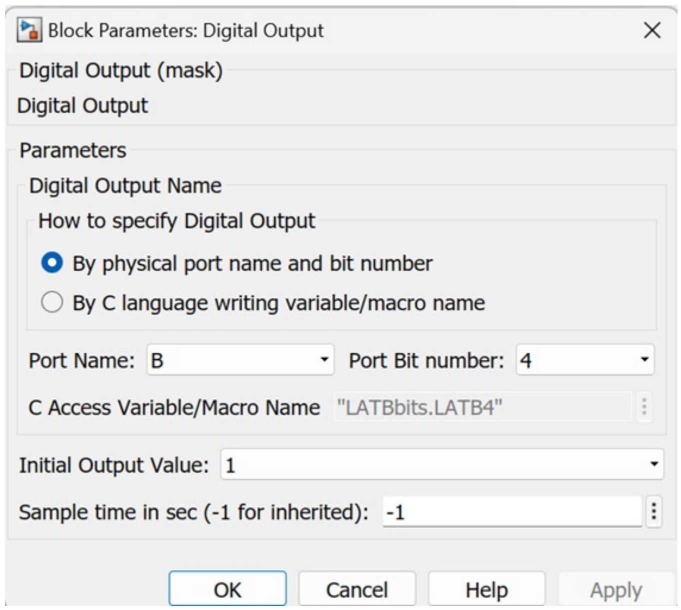


Description

This block sets digital output.

Parameters

Please double click this block to open parameters dialog below :



Let us explain parameters.

- How to specify Digital Output — radio selection button for Digital output Pin. It can be one of "By physical port name and bit number"/"By C language writing variable/macro name".
- Port Name — drop list for Digital Port name. It is used only when parameter "How to specify Digital Output" is "By physical port name and bit number".
- Port Bit number — drop list for Digital Port Bit Number. It is used only when parameter "How to specify Digital Output" is "By physical port name and bit number".
- C Access Variable/Macro Name — String type of scalar. It can be variable name or macro name in C language for Digital output. It is used only when parameter "How to specify Digital Output" is "By C language writing variable/macro name".
- Initial Output Value: — Logical type of scalar. It is initial value for "Digital Output".
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

***Notes**: Please configure GPIO. All settings must be done in IDE GUI configuration (It is MCC for Microchip Micro-controller). And it must match this block settings, otherwise unexpected result will occur.*

Ports

Input

- Dout — Logical Scalar. Digital output Value.

Outport

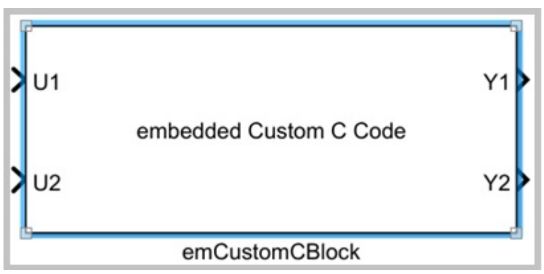
None

Example: See "ColorDisplay" block example

emCustomCBlock

set up user's manual C code Block.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / emCustomCBlock



Description

Simulink and Our Library provide lots of blocks for embedded system. However, we cannot cover all peripherals and applications. This block provides a method to let customer write C code manually. Or customers have existing mature C code, they can still use this C code by this block. Or customers can use AI such as "Copilot"/"Gemini/DeepSeek" to create C code, and put it into this block.
Manual C Code has 3 parts, the 1st part is to define global variables and implementation of interrupt service functions or callback functions or regular functions which are used in this block, the 2nd part is to run once for initialization, the 3rd part is to run periodically in sample time, the 3rd part code must

be non-blocking C code.

For the 1st part, you just open your IDE, and open .c file, which is under folder "Simulink Generated Files", and look for comment "//Customized ISR or Global Code for Block: Your block name ------Start". And add declaration of your global variables and implementation of functions or ISR or callback just under comment "//Add your code" and before comment "//Customized ISR or Global Code for Block: Your block name ------End"

For initialization part, it is the same .c file as the 1st part, and look for comment "//Customized Initialization Code for Block: Your block name ------Start".
And add your code just under comment "//Add your code" and before comment "//Customized Initialization Code for Block: Your block name ------End". This part C code can be blocking or non-blocking. Don't care.

For periodical part, it is the same .c file as the 1st part, and look for comment "//Customized Step (loop) Code for Block: Your block name ------Start".
And add your code just under comment "//Add your code" and before comment "/Customized Step (loop) Code for Block: Your block name ------End". This part C code must be non-blocking. Or blocking but maximum block time is less than 0.5ms.

Before your code, you will see comment "//Input Port1 variable name: U1   Data Type: int16" .... "// Output Port1 variable name: Y1   Data Type: int16" ...
It is convenient to write your code according to these comments for input/output variable name and data type.

By this block, you can implement new function in your C code.
For example, you can implement 3 phase PWMs. You use Configuration Software (such as MCC , STM32CubeMx, or SysConfig) to create PWM initialization code.
Leave "Custom C Block" initialization part empty. Or you don't use any configuration software, just directly set relative registers by your self for initialization part of this block.
At last you write C code manually for periodical part to set duty cycle register value.
Your emCustomCBlock is reusable. You can copy to the other project.

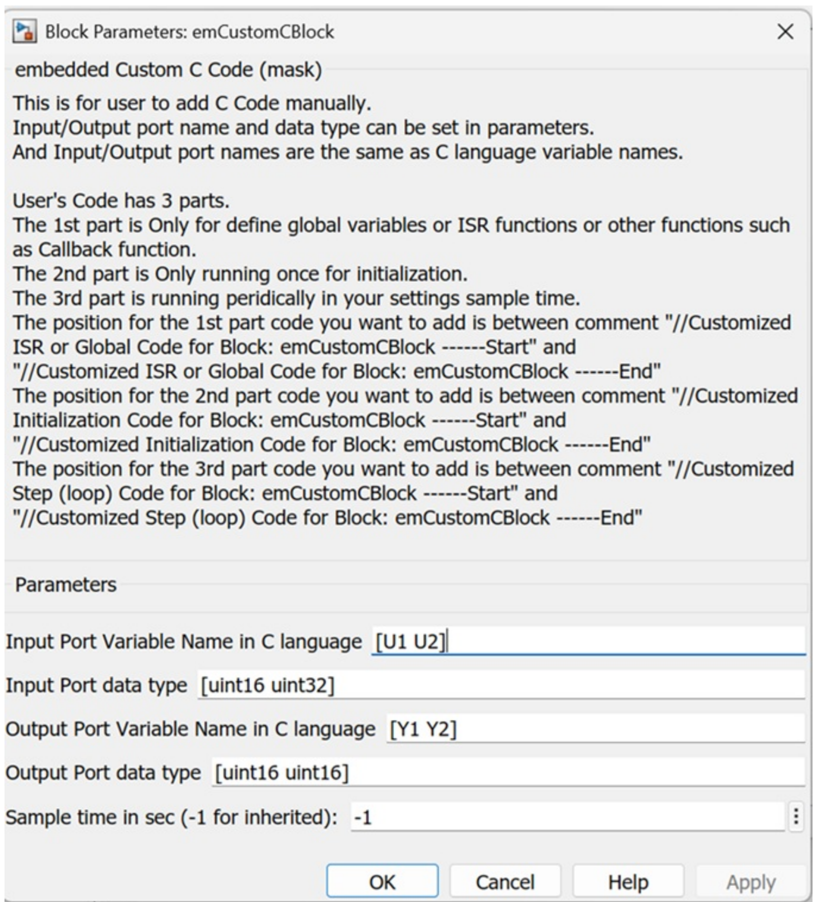
Notes: *1 Please don't edit any comment which is created by Simulink code.*
2 For MplabX IDE project, You can find your previous C code Simulink created including your customized C code in Folder: "Your embedded project directory used in configuration" \ simulink_generated_files_backup. You can find your C code of the one before the previous in Folder: "Your embedded project directory used in configuration" \ simulink_generated_files_backup_backup. For STM32Cube and TI

C2000 CCS Project, You can find your previous C code Simulink created including your customized C code in Folder: "Your Computer User directory" \ Simulink Model name (without suffix slx) \ simulink_generated_files_backup. You can find your C code of the one before the previous in Folder: "Your Computer User directory" \ Simulink Model name (without suffix slx) \ simulink_generated_files_backup_backup.

3 If you modified C code of emCustomCBlock in your IDE, and If you want "this emCustomCBlock" to keep your customized C source code, please execute "build" command from simulink after you debug in IDE. Or your emCustomCBlock cannot keep the latest customized C source code when you copy "this emCustomCBlock" to the other simulink project model.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Input Port Variable Name in C language — It specifies variable name of input ports. It can be scalar or vector. How many vector elements it has is how many input ports it has. All input ports are scalar. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["U1", "U2"], You cannot use " [U1, U2]". You must use [U1, U2] or [U1 U2] or [U1; U2]

- Input Port data type — It specifies data type of all input ports. It can be scalar or vector. element can be "bool","uint8", "int8","uint16","int16","uint32","int32","single". How many vector elements it has is how many input ports it has. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["int8", "uint16"], You cannot use " [int8, uint16]". You must use [int8, uint16] or [int8 uint16] or [int8; uint16].

- Output Port Variable Name in C language — It specifies variable name of output ports. It can be scalar or vector. How many vector elements it has is how many output ports it has. All output ports are scalar. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["Y1", "Y2"], You cannot use " [Y1, Y2]". You must use [Y1, Y2] or [Y1 Y2] or [Y1; Y2]

- Output Port data type — It specifies data type of all output ports. It can be scalar or vector. element can be "bool","uint8", "int8","uint16","int16","uint32","int32","single". How many vector elements it has is how many output ports it has. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["int8", "uint16"], You cannot use " [int8, uint16]". You must use [int8, uint16] or [int8 uint16] or [int8; uint16].

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

For all input ports, it depends on parameter "Input Port Variable Name in C language". The parameter "Input Port Variable Name in C language" vector element QTY is Input data port QTY. Please read parameter "Input Port Variable Name in C language" above. and see description below

- Data1 — scalar. Parameter "Input Port Variable Name in C language" vector's first element decides its input port name. and Parameter "Input Port data type" vector's first element decides its input port data type.

.

- Data2 — scalar. Parameter "Input Port Variable Name in C language" vector's 2nd element decides its input port name. and Parameter "Input Port data type" vector's 2nd element decides its input port data type.

- Data3 — scalar. Parameter "Input Port Variable Name in C language" vector's 3rd element decides its input port name. and Parameter "Input Port data type" vector's 3rd element decides its input port data type.

......

- Data n — scalar. Parameter "Input Port Variable Name in C language" vector's number n element decides its input port name. and Parameter "Input Port data type" vector's number n element decides its input port data type

Outport

For all output ports, it depends on parameter "Output Port Variable Name in C language". The parameter "Output Port Variable Name in C language" vector element QTY is output data port QTY. Please read parameter "Output Port Variable Name in C language" above. and see description below

- Data1 — scalar. Parameter "Output Port Variable Name in C language" vector's first element decides its output port name. and Parameter "Output Port data type" vector's first element decides its output port data type.

.

- Data2 — scalar. Parameter "Output Port Variable Name in C language" vector's 2nd element decides its output port name. and Parameter "Output Port data type" vector's 2nd element decides its output port data type.

- Data3 — scalar. Parameter "Output Port Variable Name in C language" vector's 3rd element decides its output port name. and Parameter "Output Port data type" vector's 3rd element decides its output port data type.

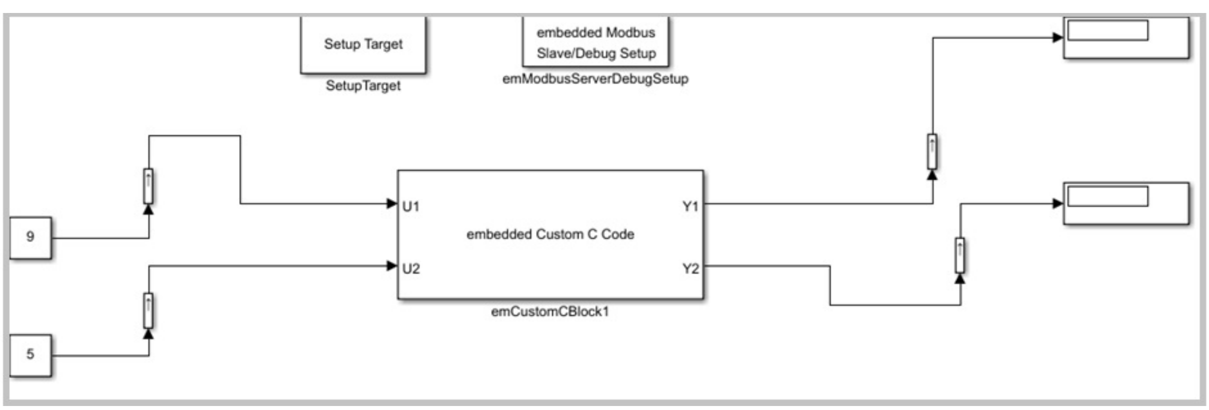
......

- Data n — scalar. Parameter "Output Port Variable Name in C language" vector's number n element decides its output port name. and Parameter "Output Port data type" vector's number n element decides its output port data type

Examples

Our embedded platform is dsPIC33EP256GP502 .

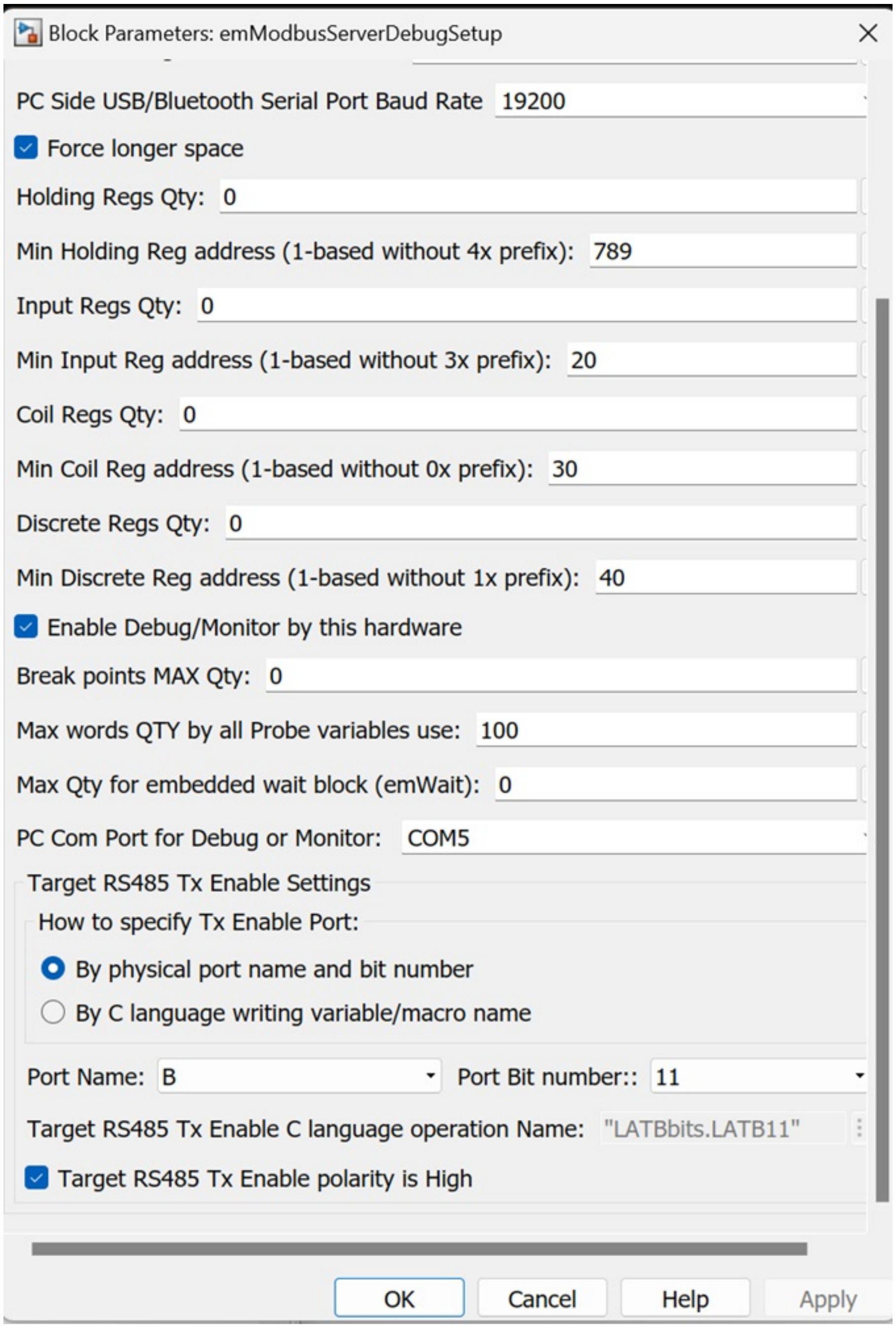
Please see screenshot of model below:



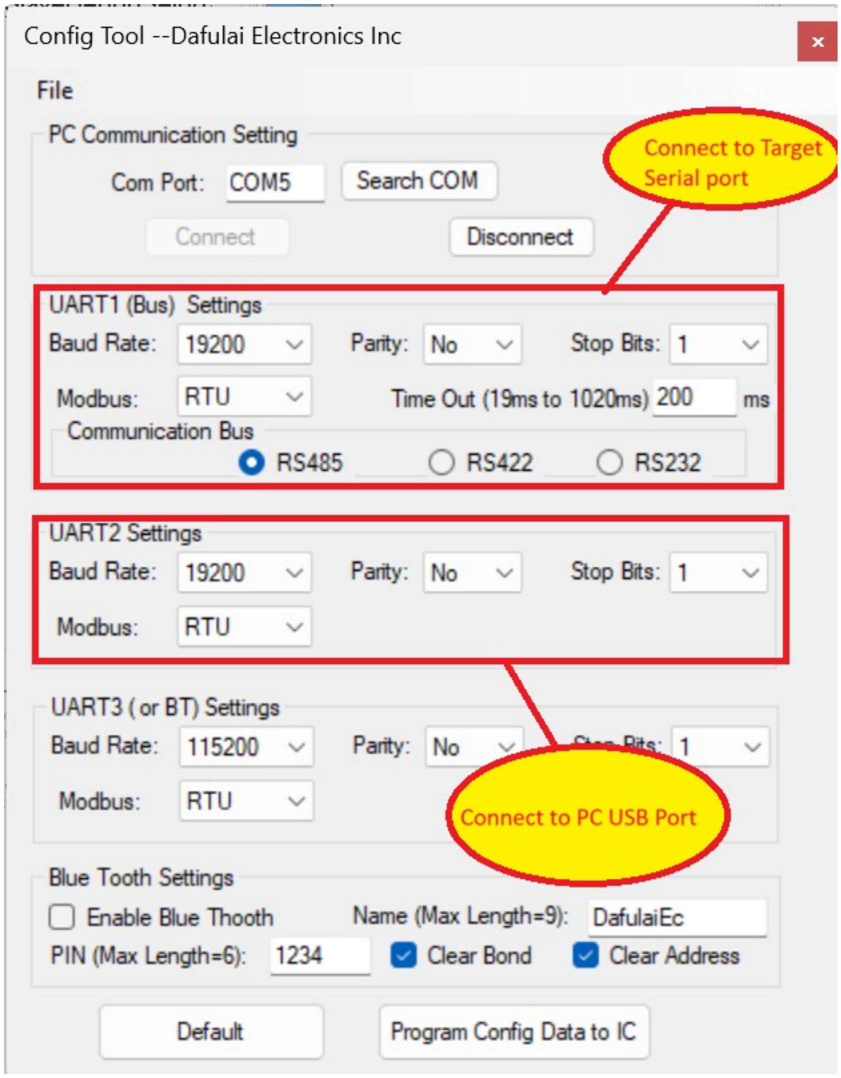
As you see above, in order to watch result, we used "probe" blocks. So we used "Modbus RTU/ASCII

Dual Masters adaptor" as debugger hardware. This hardware connects Target Uart2, and Uart2 controls RS485, TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver.

Please see our "emModbusServerDebugSetup" block settings below (only for debugging):

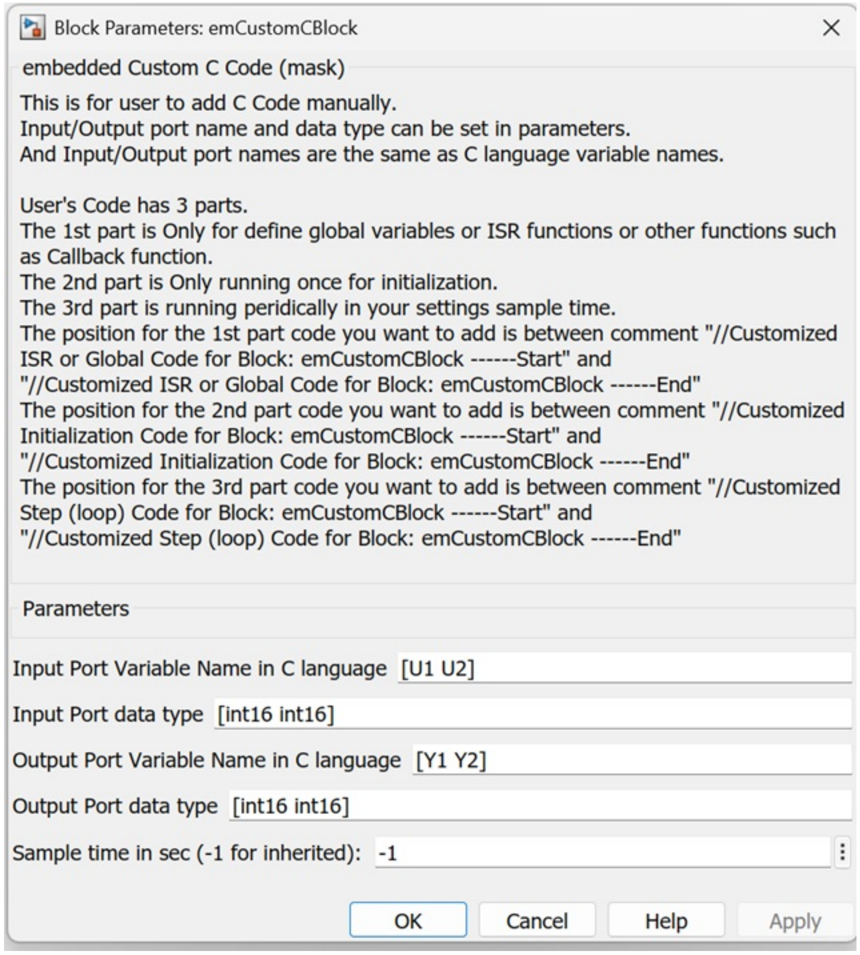


Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

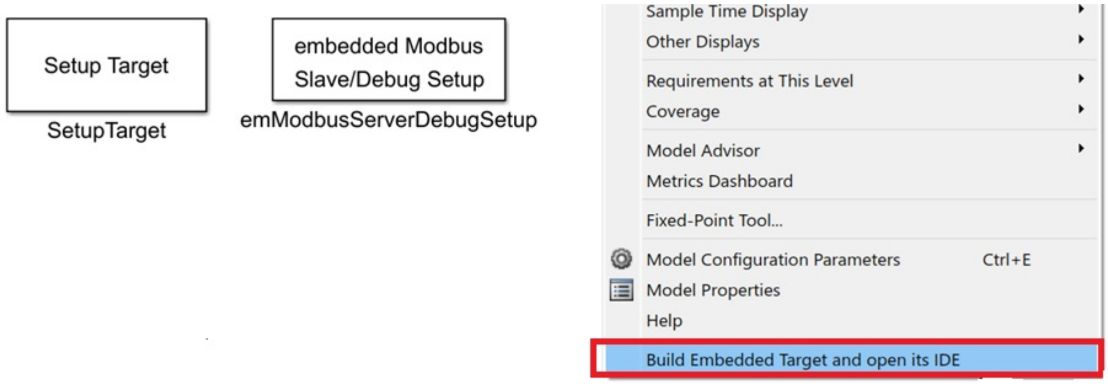
Double click block "emCustomCBlock1", we will see parameters settings for "emCustomCBlock" below:



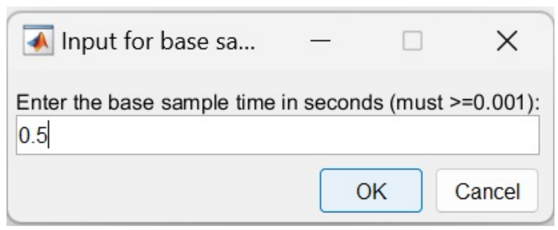
It means that there are 2 input ports which used name U1 and U2 and data type is int16, there are 2 output ports which used name Y1 and Y2 and data type is int16,

Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too)
We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

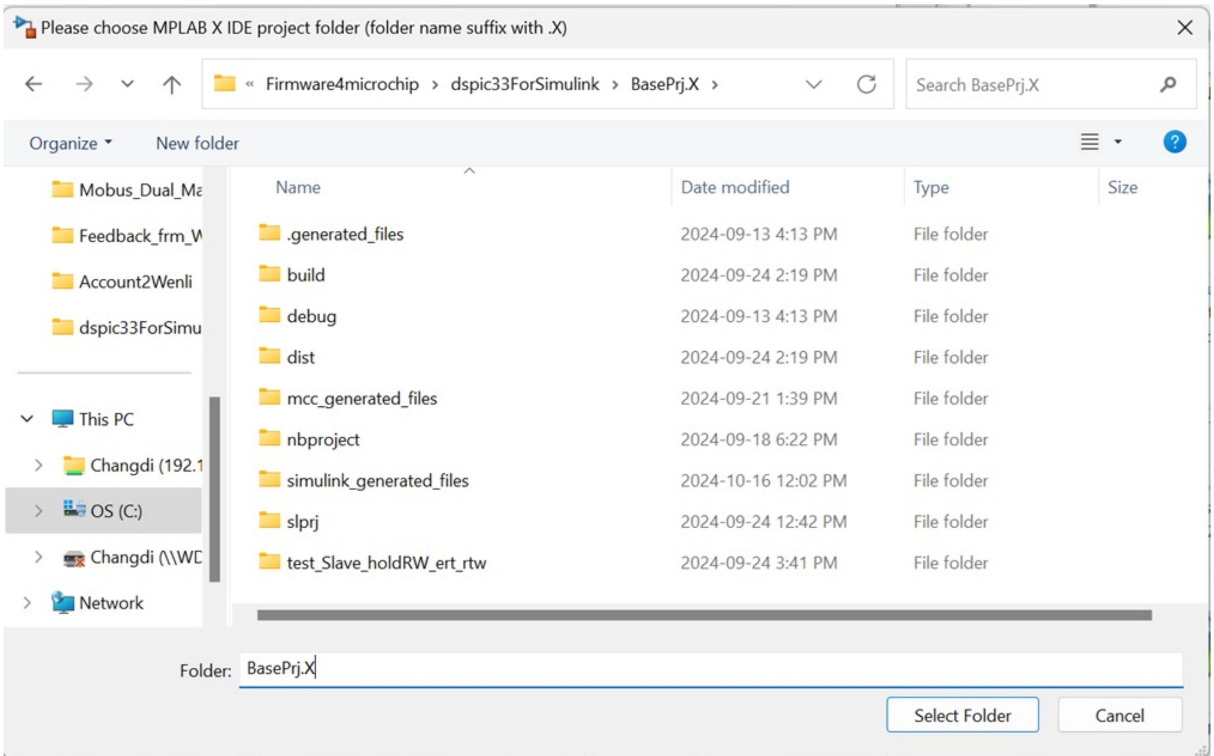
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:



Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



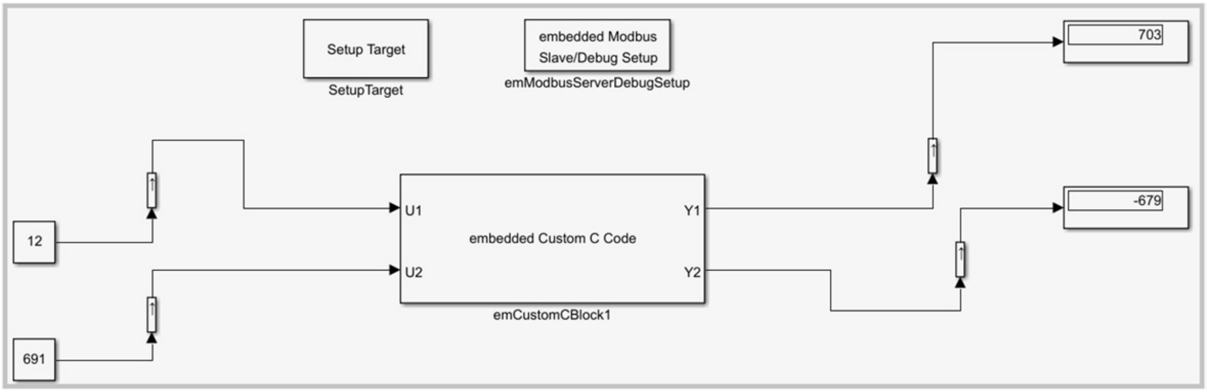
If you give out the correct IDE project directory, Simulink will open IDE software automatically.

We search to "//Customized Step (loop) Code for Block: emCustomCBlock1 ------Start" in IDE editor, we will add code "Y1= U1+U2;" and "Y1= U1-U2;" as below:

```
    /* M-S-Function: '<Root>/emCustomCBlock1' */
#define Y1                              (example14_emCustom_B.emCustomCBlock1_o1)
#define Y2                              (example14_emCustom_B.emCustomCBlock1_o2)

    //Input Port1 variable name: U1     Data Type: int16_t
    //Input Port2 variable name: U2     Data Type: int16_t

    //Output Port1 variable name: Y1    Data Type: int16_t
    //Output Port2 variable name: Y2    Data Type: int16_t

    //Customized Step (loop) Code for Block: emCustomCBlock1 ------Start
    //Add your code
    Y1= U1+U2;
    Y2= U1-U2;

    //Customized Step (loop) Code for Block: emCustomCBlock1 ------End
```

And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

We can use Simulink directly view results. By select stop time= inf, and click "Run" button. You can change U1 and U2 value when run time. you will see result below:

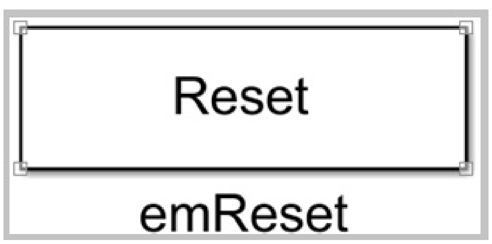


Please open "Your embedded creator library folder"/examples/example14_emCustom.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emReset

Reset Target Micro-Controller
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / emReset



Description

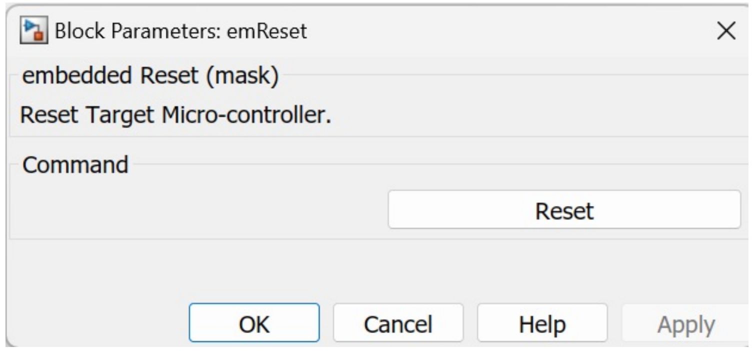
This block resets Target CPU and all peripherals.

> ***Notes***: *You can add only - one embedded Modbus Master or Client. You cannot create 2 or 2+ Modbus Masters or Clients.*

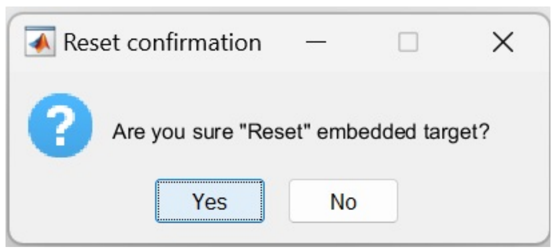
Parameters

None

But when you double click this block, you can open operation dialog below:

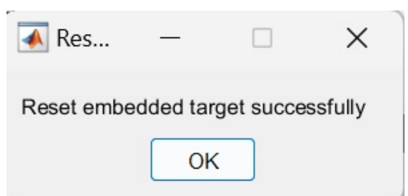


"Reset" is command Button. When you click "Reset" Button during simulink running, A popup window occurs below:

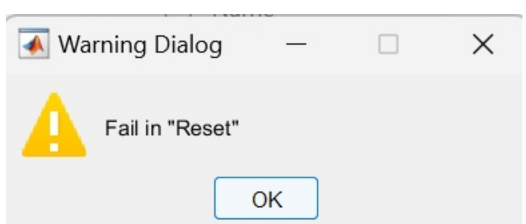


If you click "Yes", it will send "Reset" command to target, otherwise, cancel "Reset" command.

If "Reset" success, it will display window below:



If "Reset" failed, it will display window below:



Ports

Input

None

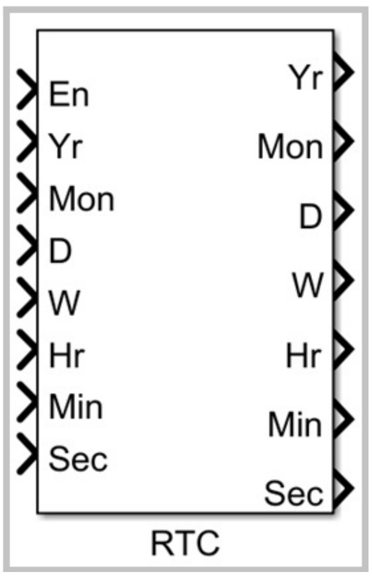
Outport

None

RTC

Embedded Real Time Clock.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / RTC

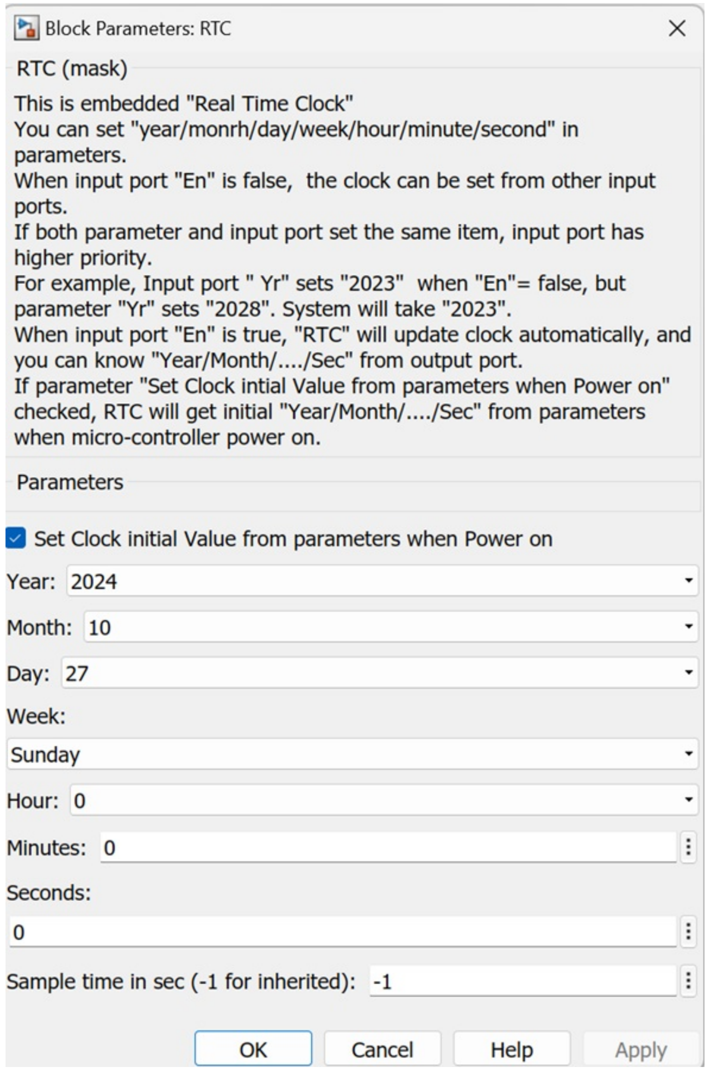


Description

This block will output system clock including year/month/day/week. (TI C2000 does not support this block because most TI F28X DSPs have no RTC peripherals)

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Set Clock initial Value from parameters when Power on — logical scalar. Usually RTC has dedicated small "Coin battery". When micro-controller power off, RTC can stall run by this small "coin battery". So you don't need to initialize RTC's "Year/Month/.../Sec" when micro-controller power on again.   This parameter value "True" means that micro-controller will initialize RTC's "Year/Month/.../Sec" from parameters when micro-controller power on. "False" means that micro-controller will not initialize RTC's "Year/Month/.../Sec" when micro-controller power on.

- Year — drop list from 2024 to 2055 . You just choose from drop list. This is initial year, When micro-controller power on or reset, it will take this year as initial value if parameter "Set Clock initial Value from parameters when Power on" is true.

- Month — drop list from 1 to 12 . You just choose from drop list. This is initial month, When micro-controller power on or reset, it will take this month as initial value if parameter "Set Clock initial Value from parameters when Power on" is true.

- Day — drop list from 1 to 31 or 30 or 29 or 28. You just choose from drop list. This is initial day, When micro-controller power on or reset, it will take this day as initial value if parameter "Set Clock initial Value from parameters when Power on" is true.

- Week — drop list from "Sunday" To "Saturday". You just choose from drop list. This is initial week, When micro-controller power on or reset, it will take this value as initial week if parameter "Set Clock initial Value from parameters when Power on" is true.

- Hour — drop list from 0 to 23. You just choose from drop list. This is initial hour, When micro-controller power on or reset, it will take this value as initial hour if parameter "Set Clock initial Value from parameters when Power on" is true.

- Minutes — integer from 0 to 59. This is initial minutes, When micro-controller power on or reset, it will take this value as initial minute if parameter "Set Clock initial Value from parameters when Power on" is true.

- Seconds — integer from 0 to 59. This is initial seconds, When micro-controller power on or reset, it will take this value as initial second if parameter "Set Clock initial Value from parameters when Power on" is true.

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Notes*:* Please configure RTCC. All settings must be done in IDE GUI configuration (It is MCC for Microchip Micro-controller)

Ports

Input

- En — "logical" data type's scalar. True means "RTC" keeps running and updating output port "Yr/Mon/D/.../Sec" continuously. False means "RTC" stop updating output port "Yr/Mon/D/.../Sec" and set initial values to input port "Yr/Mon/D/.../Sec" if it is connected or set initial values to parameters "Year/Month/.../Seconds" if related input port is not connected.

- Yr — "unsigned 16 bits of integer" data type's scalar. It denotes initial value of "Year". If input port "En" is false, "RTC" will take this initial value.

- Mon — "unsigned 8 bits of integer" data type's scalar from 1 to 12. It denotes initial value of "Month". If input port "En" is false, "RTC" will take this initial value.

- D — "unsigned 8 bits of integer" data type's scalar from 1 to 31. It denotes initial value of "Day". If input port "En" is false, "RTC" will take this initial value.

- W — "unsigned 8 bits of integer" data type's scalar from 0 to 6. It denotes initial value of "Week" (0 is Sunday,1 is Monday) . If input port "En" is false, "RTC" will take this initial value.

- Hr — "unsigned 8 bits of integer" data type's scalar from 0 to 59. It denotes initial value of "Hours". If input port "En" is false, "RTC" will take this initial value.

- Min — "unsigned 8 bits of integer" data type's scalar from 0 to 59. It denotes initial value of "Minutes". If input port "En" is false, "RTC" will take this initial value.

- Sec — "unsigned 8 bits of integer" data type's scalar from 0 to 59. It denotes initial value of "Seconds". If input port "En" is false, "RTC" will take this initial value.

Outport

- Yr — "unsigned 16 bits of integer" data type's scalar. It denotes current value of "Year". If input port "En" is true.

- Mon — "unsigned 8 bits of integer" data type's scalar. It denotes current value of "Month". If input port "En" is true.

- D — "unsigned 8 bits of integer" data type's scalar. It denotes current value of "Day". If input port "En" is true.

- W — "unsigned 8 bits of integer" data type's scalar. It denotes current value of "Week" (0 is Sunday,1 is Monday) . If input port "En" is true.

- Hr — "unsigned 8 bits of integer" data type's scalar. It denotes current value of "Hours". If input port "En" is true.

- Min — "unsigned 8 bits of integer" data type's scalar. It denotes current value of "Minutes". If input port "En" is true.

- Sec — "unsigned 8 bits of integer" data type's scalar. It denotes current value of "Seconds". If input port "En" is true.

setupTarget

set up embedded target platform
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) /setupTarget

Description

This block sets up which Micro-controller family will be used.

***Notes**: 1 For any embedded Micro-controller platform, you must have one*

"SetupTarget" block. And you must install windows IDE software for this platform. For Microchip embedded platform, the IDE software is MPLABX IDE and MCC or Harmony. For STM32 platform, the IDE software is STM32CubeIDE. For TI DSP C28X platform, the IDE software is CCS, SysConfig, C2000ware.

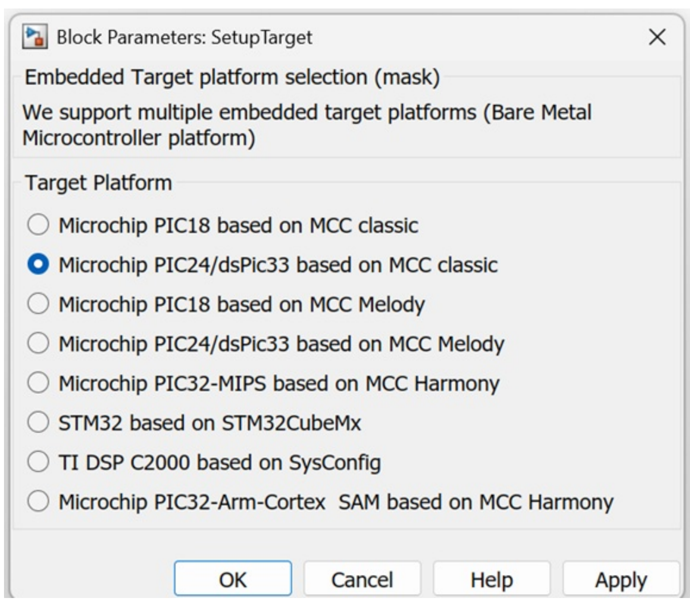
2 Before you use code creator, you must use platform IDE software to configure all hardware resource you will use in Simulink model. You must use timer1 as Simulink 1 ms tick source if it is Microchip Non-Arm Cortex platform. You must use tcc1 as Simulink 1 ms tick source if it is Microchip Arm Cortex platform. You must set Timebase Source to SysTick if it is STM32CubeMx platform. You must use CPU Timer0 as Simulink 1 ms tick source if it is TI DSP C2000Ware SysCfg platform

3 All blocks we provide for embedded target are "non-blocking" in order to keep real-time in bare metal programming environment.

4 For TI C2000Ware SysConfig platform, please directly use Flash version to debug, don't use RAM version to debug

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Microchip PIC18 based on MCC classic — Target MCU is Microchip PIC18 family, and we use MCC classic configuration software.
- Microchip PIC24/dsPic33 based on MCC classic — Target MCU is Microchip PIC24/dsPIC33 family, and we use MCC classic configuration software

- Microchip PIC18 based on MCC Melody — Target MCU is Microchip PIC18 family, and we use MCC Melody configuration software.
- Microchip PIC24/dsPic33 based on MCC Melody — Target MCU is Microchip PIC24/ dsPIC33 family, and we use MCC Melody configuration software
- Microchip PIC32-MIPS based on MCC Harmony — Target MCU is Microchip PIC32 Mips family, and we use MCC Harmony configuration software
- STM32 based on STM32CubeMx — Target MCU is STM32 family, and we use STM32CubeIDE or STM32CubeMx configuration software
- TI DSP C2000 based on SysConfig — Target MCU is TI C2000, and we use SysConfig software to set up low layer's hardware code.
- Microchip PIC32-Arm-Cortex & SAM based on MCC Harmony — Target MCU is Microchip PIC32 ARM Cortex family, and we use MCC Harmony configuration software.

Ports

Input

None

Outport

None.

Examples

Please see "emModbusSlaveReadInputRegs" block example

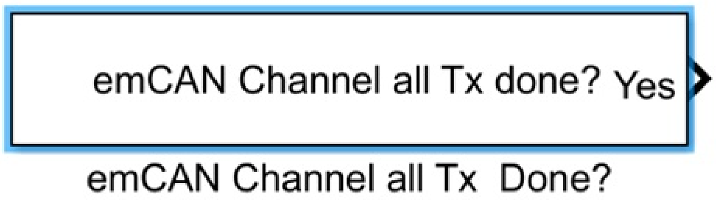
The following blocks are in "Embedded CAN Bus" directory of library. They are all related to CAN Bus operation. We list them according to alphabetical order.
Embedded CAN bus transmits in 3 Channels: A, B, and C. They can be in the different "Sample Time". And each channel transmits according to sequences.
Each channel operation is one by one in sequences. You must set up all transmit operation sequences when this Channel is in idle, and you should set this Channel to Idle when all transmit sequences done in this Channel.

emCANChannelAllTxDone

emCAN bus all transmission sequences Done?
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded CAN Bus /

emCANChannelAllTxDone

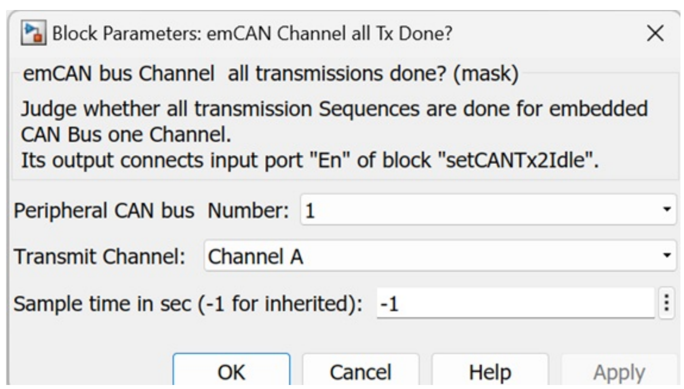


Description

This block is used in judgment whether all sequences finish for CAN bus one transmission channel. In order to start new transmission operations, you must set transmission channel into "Idle" state when all sequences done.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral CAN bus Number — tell system this block is which CAN controller peripheral used. You just choose from drop list items: 1 or 2.
- Transmit Channel: — drop list to select transmission Channel A or Channel B or Channel C.
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

- Yes — "logical" data type's scalar. True means all sequences of this transmission channel done.

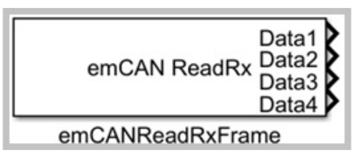
Examples

Please see "emCANTransmit" block example.

emCANReadRxFrame

Read embedded CAN bus receiver Data.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded CAN Bus / emCANReadRxFrame



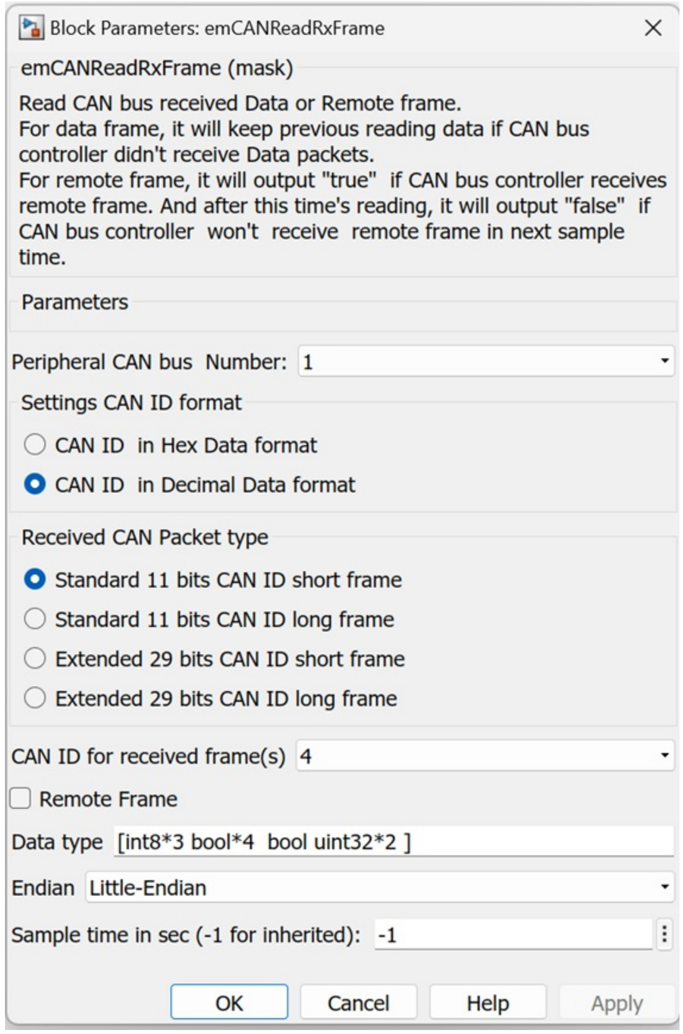
Description

Read CAN bus received Data or Remote frame.
For data frame, it will keep previous reading data if CAN bus controller didn't receive Data packets.
For remote frame, it will output "true" if CAN bus controller receives remote frame. And after this time's reading, it will output "false" if CAN bus controller won't receive remote frame in next sample time.

> ***Notes:*** *You must set up CAN bus receiver interrupt enable in MCU Configuration software such as MPLABX IDE MCC for Microchip technology MCUs.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral CAN bus Number — tell system this block is which CAN controller peripheral used. You just choose from drop list items: 1 or 2.
- Settings input data format — radio button to select CAN ID format is Hex or decimal. You can change this setting in any time to see CAN ID in Hex or Decimal way.
- Received CAN Packet type — radio button to select received CAN ID type. It can be one of "Standard 11 bits CAN ID short frame", "Standard 11 bits CAN ID long frame", "Standard 29 bits CAN ID short frame" and "Standard 29 bits CAN ID long frame"
- CAN ID for received frame(s) — CAN Bus receiver's CAN ID. You just choose from drop list items. These drop list items are from "emCANSetup" block. If your received CAN ID is not in drop list, please modify "emCANSetup" block.
- Remote Frame — tell this block whether received CAN bus is remote frame. If true, output

port will be logical type of scalar "RTR".

- Data type — It specifies data type of data field for this received CAN ID frame. It can be scalar or vector. element can be "bool","uint8", "int8","uint16","int16","uint32","int32","single". How many vector elements it has is how many output ports it has.If data output port is vector, please follow * and vector element Qty. For example, [int8, uint16*2, bool*7] denote that data field has 6 bytes, and block has 3 output port. The 1st output port is "int8" type of scalar. The 2nd output port is "uint16" type of vector with 2 elements. The 3rd output port is "bool" type of vector with 7 elements. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["int8", "uint16*2", "bool*7"], You cannot use " [int8, uint16*2, bool*7]". You must use [int8, uint16*2, bool*7].
  .
- Endian — It is bool type vector of bit order and "int16/uint16/uint32/int32/single" type of byte order. For example, Data type parameter=[bool*3 uint16]. It occupies 3 bytes. D0 denotes the 1st byte of data field. D1 denotes the 2nd byte of data field. D2 denotes the 3rd byte of data field. D0.0 denotes D0's LSb (less significant bit). D0.7 denotes D0's MSb (most significant bit). The output port1 "Data1" is bool type of vector with 3 elements. The output port2 "Data2" is uint16 type of scalar. When we use "Little-Endian", Data1=[D0.0 D0.1 D0.2] and Data2= D2.2 D2.1 D2.0 D1.7 D1.6 D1.5 D1.4 D1.3 D1.2 D1.1 D1.0 D0.7 D0.6 D0.5 D0.4 D0.3. When we use "Big-Endian", Data1=[D0.7 D0.6 D0.5] and Data2= D0.4 D0.3 D0.2 D0.1 D0.0 D1.7 D1.6 D1.5 D1.4 D1.3 D1.2 D1.1 D1.0 D2.7 D2.6 D2.5

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

It depends on parameter "Remote Frame" and "Data type".

When "Remote Frame" is true, there is one output port as below:

- RTR — "logical" data type's scalar. "True" means CAN bus controller received "Remote request frame". And it will become false after next sample time if no more new remote frame received.

When "Remote Frame" is false, parameter "Data type" vector element QTY is Output port QTY. Please read parameter "Data type" above. and see description below:

- Data1 — vector or scalar. Parameter "Data type" vector's first element decides what data type it is. and " *digit" decides vector size

.

- Data2 — vector or scalar. Parameter "Data type" vector's second element decides what data type it is. and " *digit" decides vector size.

- Data3 — vector or scalar. Parameter "Data type" vector's third element decides what data type it is. and " *digit" decides vector size.

......

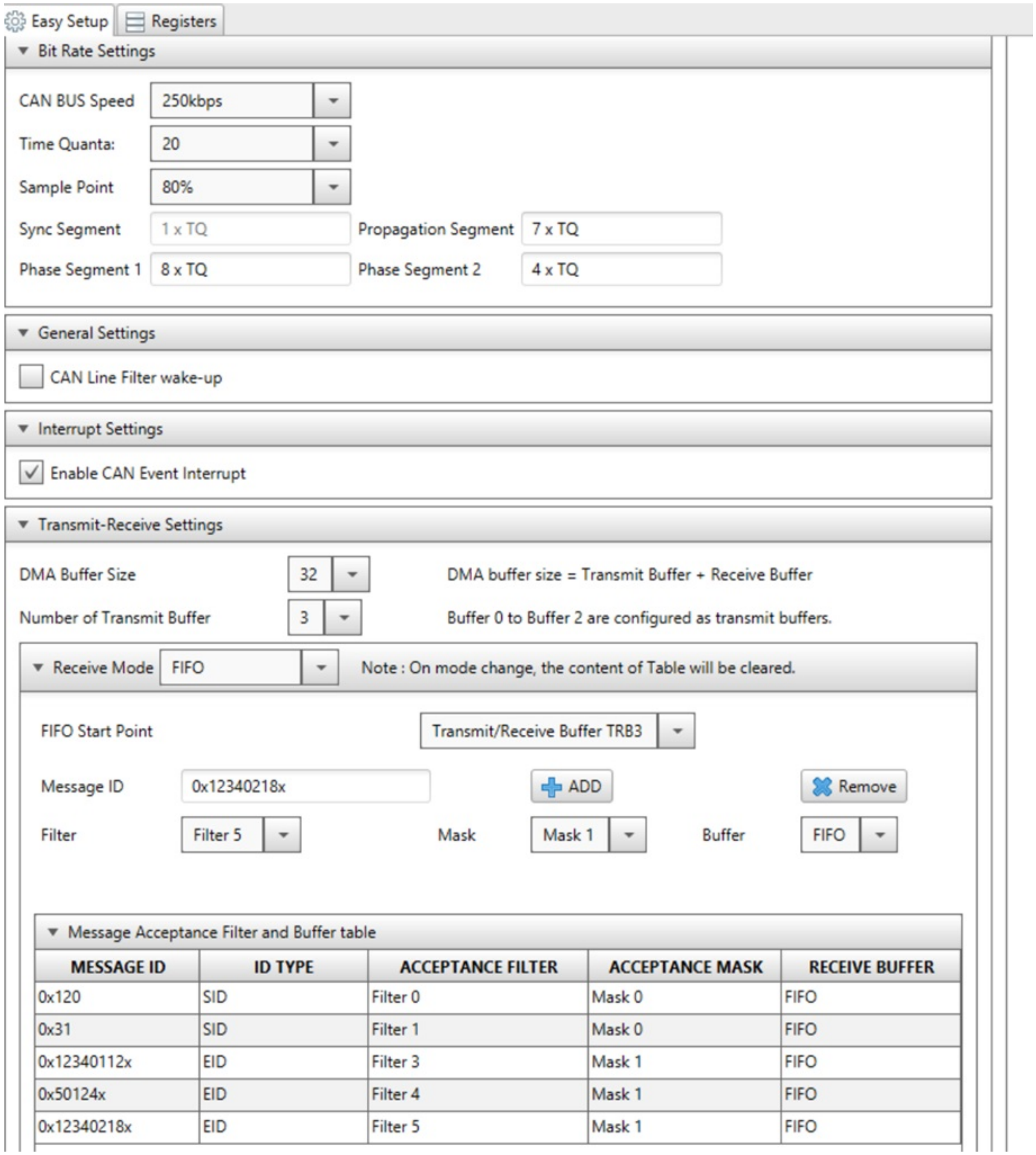
- Data n — vector or scalar. Parameter "Data type" vector's number n element decides what data type it is. and " *digit" decides vector size.

Examples

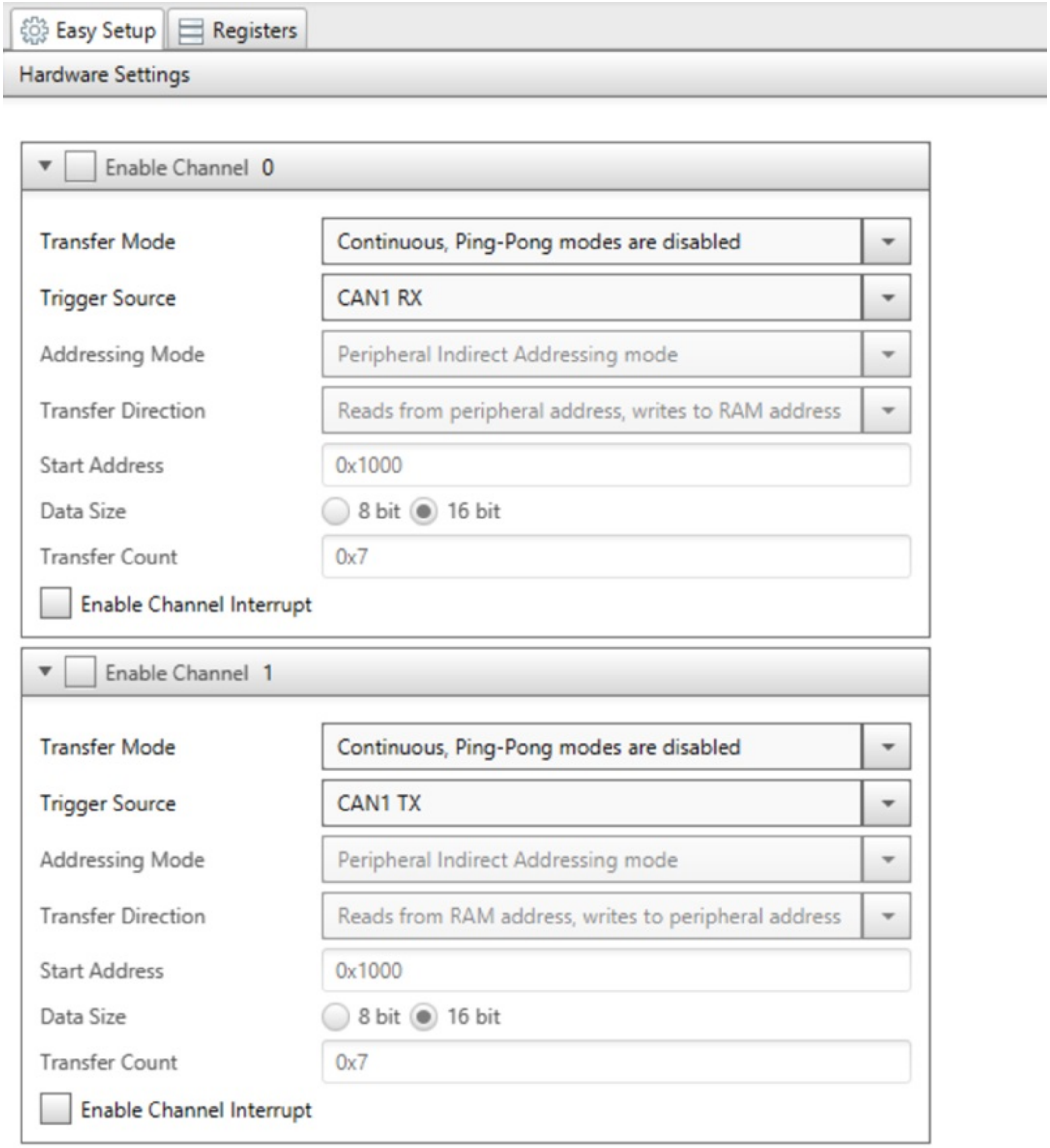
Example1:

Our embedded platform is dsPIC33EP256GP502 . CAN bus peripheral 1 is used with baud rate= 250000 bit/sec. In MPLABX IDE MCC, we set F0= 0x120 std frame used Mask0, F1= 0x31 std frame used Mask0, F1= 0x12340112 ext frame used Mask1, F2= 0x50124 ext frame used Mask1, F1= 0x12340218 ext frame used Mask1.
Number of Transmit Buffer is 3 from Buffer0 to Buffer2.

Please see MCC configuration for can1 below:



Please be careful, For dspic/Pic24 MCU, CAN bus use DMA to transfer data. We show you the DMA configuration in MPLABX IDE MCC for can1 below:

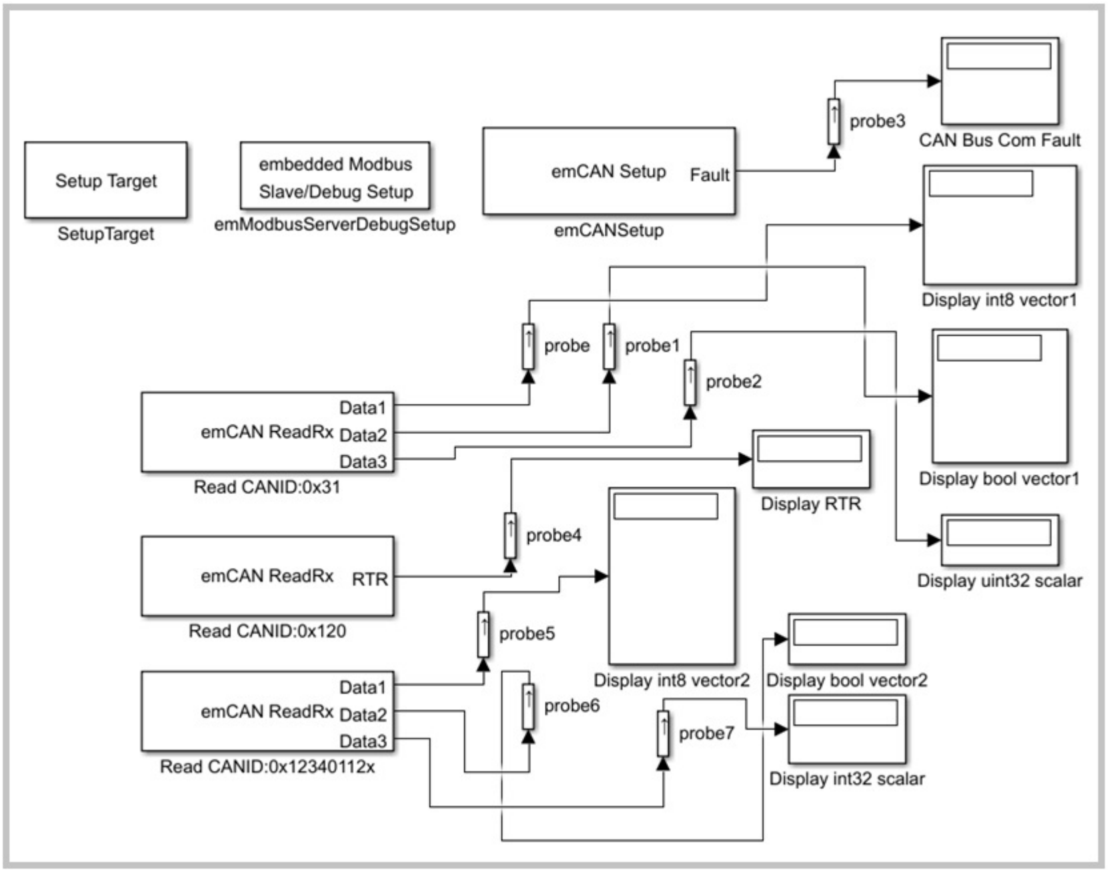


Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. This RS485 is used for Simulink Debug. This example code will read CAN frame every 0.5 seconds. The following operations will be doing during sample time:

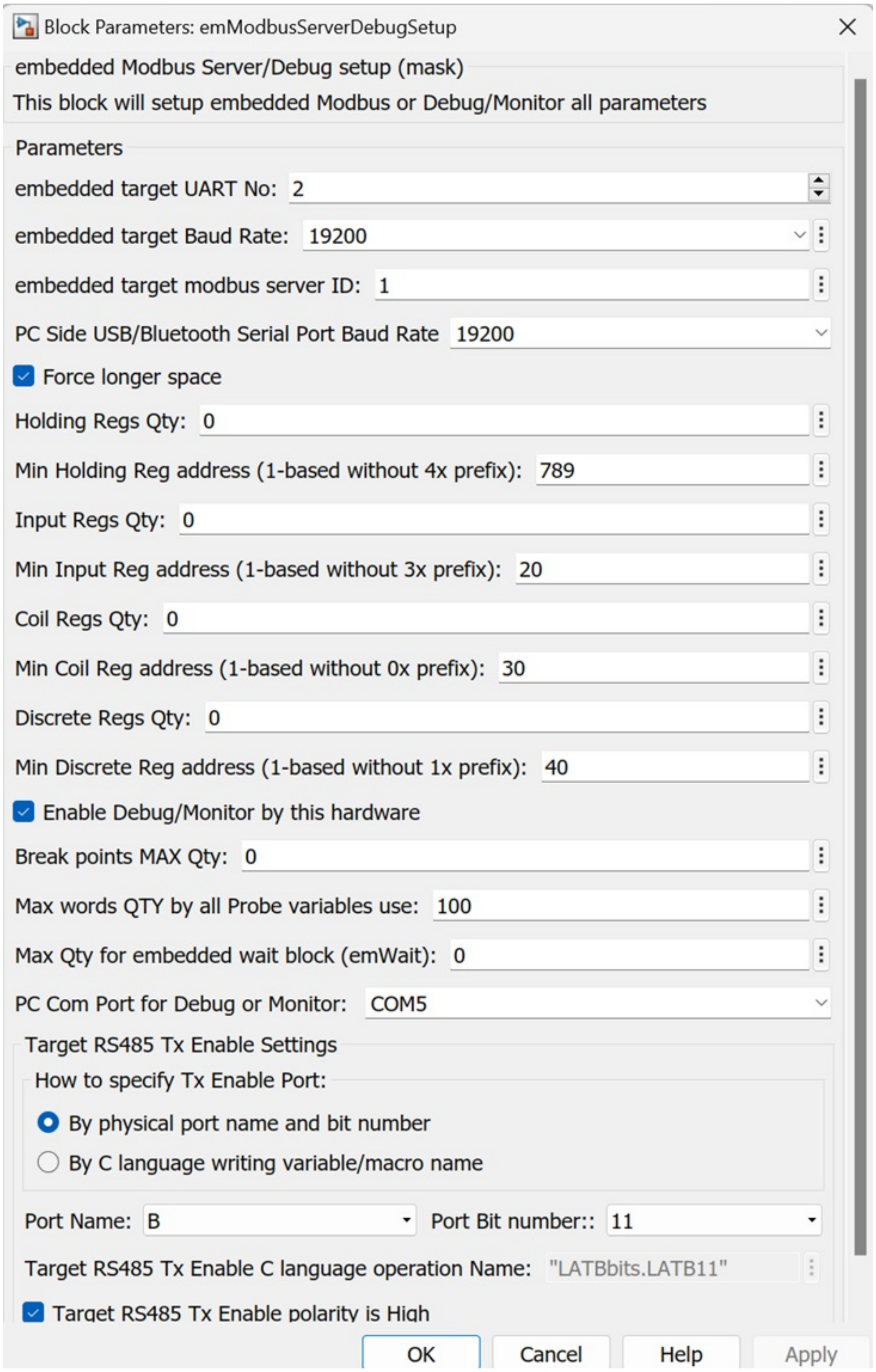
1. Read CAN ID=0x31 standard data frame with DLC=8 (short data frame), little-endian, 3 Data output ports: the 1st output port "Data1" is int8 scalar, the 2nd output port "Data2" is bool vector with 4 elements,the 3rd output port "Data3" is uint32 scalar,
2. Read CAN ID=0x120 standard remote frame.
3. Read CANID=0x12340112 extended Data frame (long data frame), Big-Endian, Total Data length=14 bytes. 3 Data output ports: the 1st output port "Data1" is int8 vector with 5 elements, the 2nd output port "Data2" is bool vector with 2 elements, the 3rd output port "Data3" is int32 vector

with 2 elements,

Please see screenshot of model below:

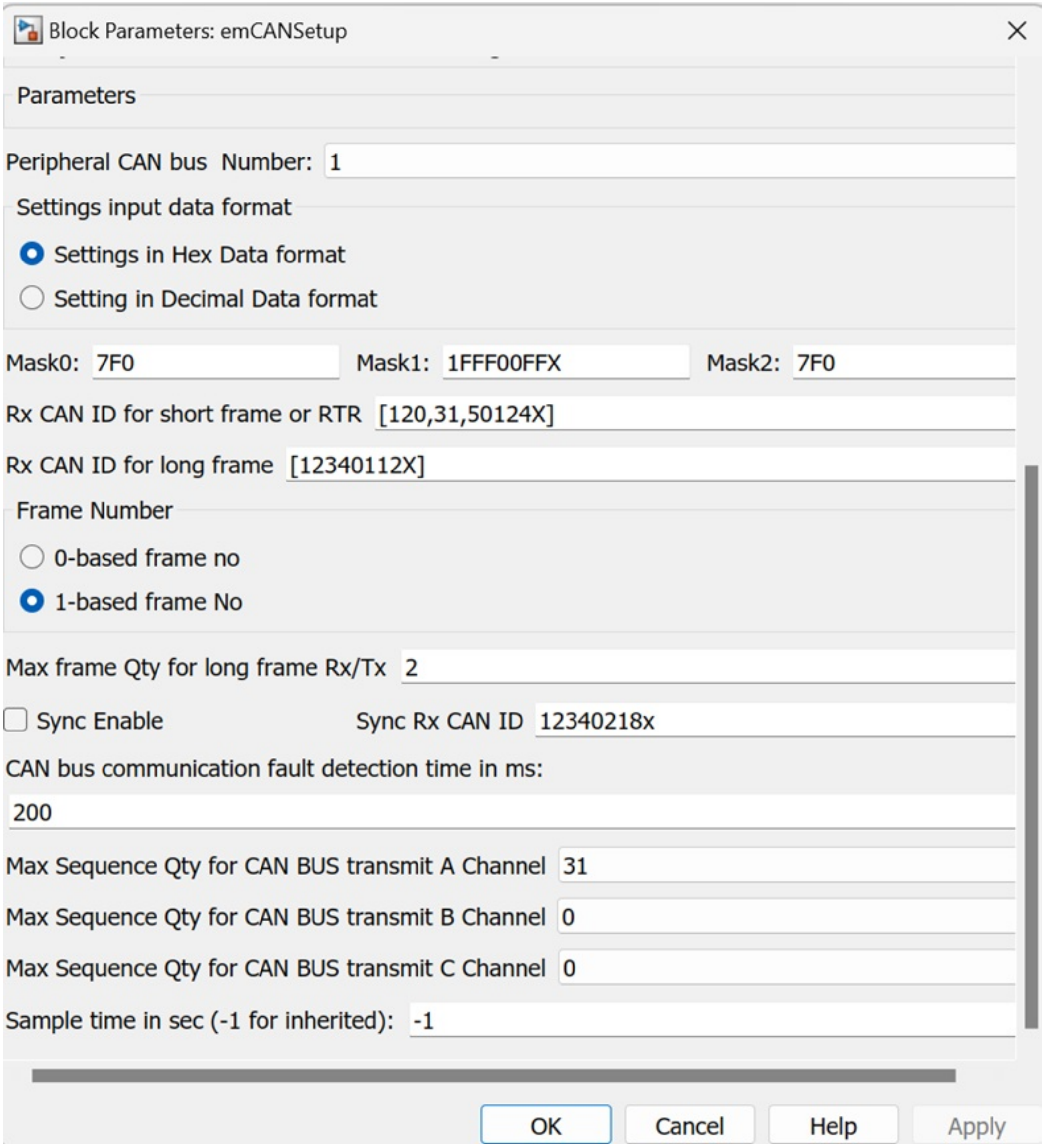


In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Please double click on "emModbusServerDebugSetup" block.



We only use this block for "Probe". So the QTY of modbus server registers is set to 0.

Please double click on "emCANSetup" block. We will see parameters for CAN bus controller below:



Let's explain settings. Mask0= 7F0, it has no suffix "X" or "x". So Mask0 will be standard 11 bits. In previous MCC configuration, Filter0=0x120 and use Mask0. We can know this CAN bus controller can receive standard frame with CAN ID= 0x120 to 0x12F because MASK0 the most right-handed 4 bits are 0 (means we don't care CAN ID's most right-handed 4 bits). In previous MCC configuration, Filter1=0x31 and use Mask0. We can know this CAN bus controller can receive standard frame with CAN ID= 0x30 to 0x3F because MASK0 the most right-handed 4 bits are 0 (means we don't care CAN ID's most right-handed 4 bits).

Mask1=1FFF00FFX, it has suffix "X". So Mask1 will be extended 29 bits. In previous MCC configuration, Filter3=0x12340112 and use Mask1. We can know this CAN bus controller can receive extended frame with CAN ID= 0x12340012 to 0x1234FF12 because MASK1 Bit8 to Bit15 are 0 (means we don't care CAN ID's Bit8 to Bit15).

In previous MCC configuration, Filter4=0x50124 and use Mask1. We can know this CAN bus controller can receive extended frame with CAN ID= 0x50024 to 0x5FF24 because MASK1 Bit8 to Bit15 are 0 (means we don't care CAN ID's Bit8 to Bit15).

In previous MCC configuration, Filter5=0x12340128 and use Mask1. We can know this CAN bus controller can receive extended frame with CAN ID= 0x12340028 to 0x1234FF28 because MASK1 Bit8 to Bit15 are 0 (means we don't care CAN ID's Bit8 to Bit15).

As for Mask2, No filter uses Mask2 in MCC configuration. So we don't care Mask2 value.

Parameter "Rx CAN ID for short frame or RTR" is [120, 31, 50124X]. It means CAN Bus controller for Simulink will receive short standard data/Remote frame with CAN ID=0x120 and 0x31, and short extended data/Remote frame with CAN ID=0x50124. These CAN IDs are covered by MCC configuration,

Parameter "Frame Number" is "1-based frame No". It means that the 1st frame's frame number (the 1st byte of data field) is 1 for long data frame. Of cause, the 2nd frame's frame number (the 1st byte of data field) is 2 for long data frame.

Parameter "Max frame Qty for long frame Rx/Tx" is 2. It means that the longest data byte QTY is 14 (7 bytes each frame, Total 2 frames will have total 14 bytes).

Parameter "Sync Enable " is false. It means that transmitting occurs at any time. It is no use for this example because we do not transmit any CAN bus frame.

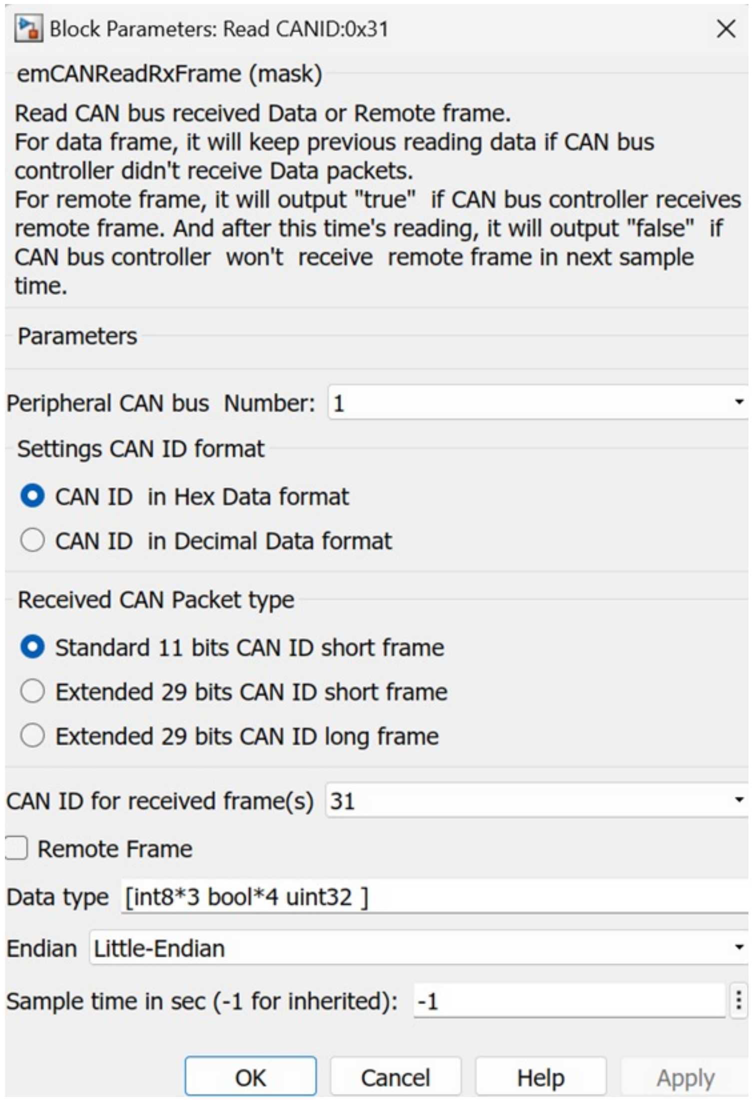
Parameter "Sync Rx CAN ID" is no use for this example because we do not transmit any CAN bus frame.

Parameter "Max Sequence Qty for CAN BUS transmit A Channel" is 31, So we can use Seq0 to Seq30 of Channel A to transmit. But this parameter is no use for this example because we do not transmit any CAN bus frame.

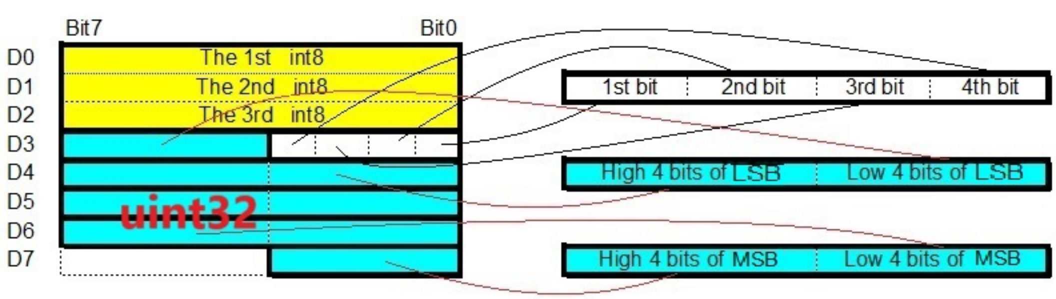
Parameter "Max Sequence Qty for CAN BUS transmit B Channel" is 0, So we cannot use Channel B to transmit. But this parameter is no use for this example because we do not transmit any CAN bus frame

Parameter "Max Sequence Qty for CAN BUS transmit C Channel" is 0, So we cannot use Channel C to transmit. But this parameter is no use for this example because we do not transmit any CAN bus frame. "Sample time in Sec" is -1 which means sample time inherit. (use base sample time 0.5 sec)

Block "Read CANID:0x31" will do operation 1 in previous operations introduce. Please double click "Read CANID:0x31". You will see parameter settings below:

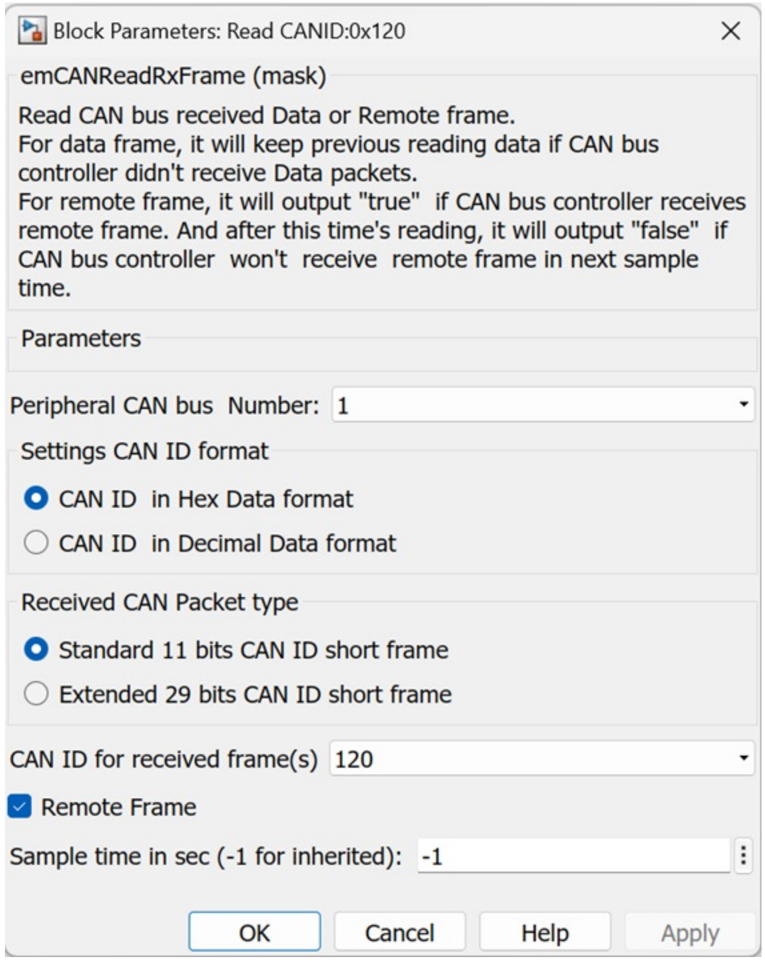


It will receive standard short data frame with CAN ID=0x31, and 8 bytes's data field is int8 vector with 3 elements, bool vector with 4 elements, uint32 scalar. It is little-endian. Please see data field of this CAN bus frame below:



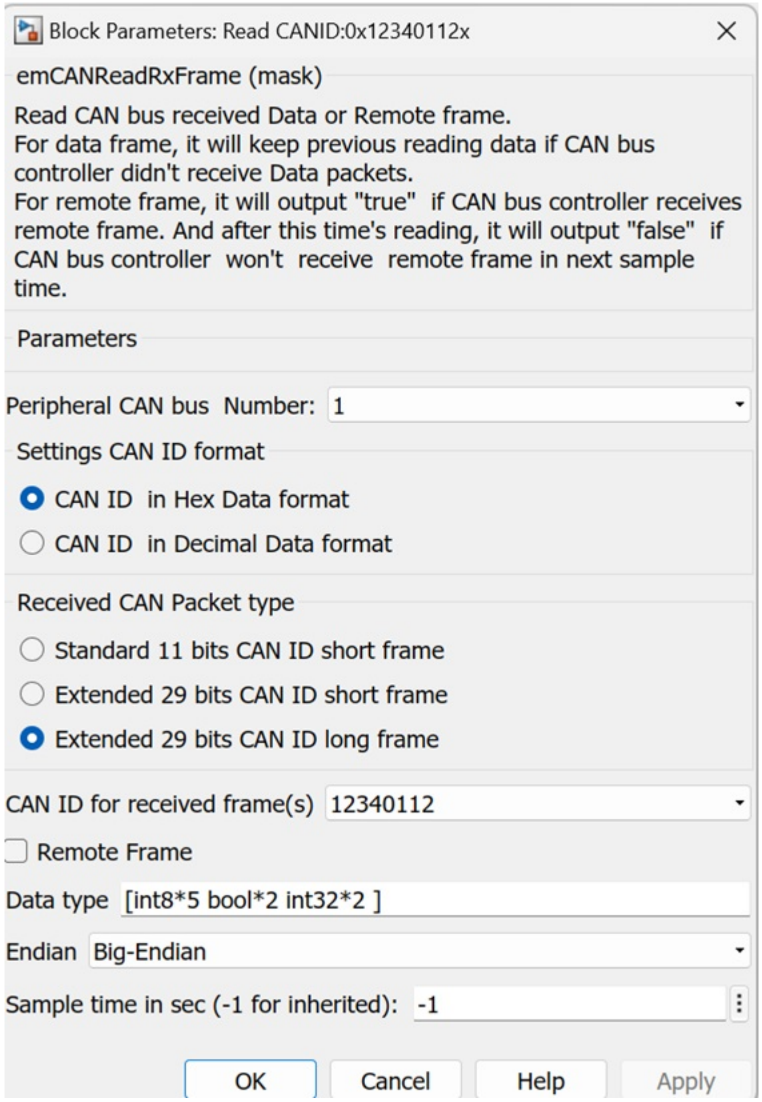
The yellow color is int8 vector data area. The white color is bool vector area (Bit0 is the 1st element of bool vector). The blue color is uint32 scalar area.

Block "Read CANID:0x120" will do operation 2 in previous operations introduce. Please double click "Read CANID:0x120". You will see parameter settings below:



It will receive standard remote frame with CAN ID=0x120.

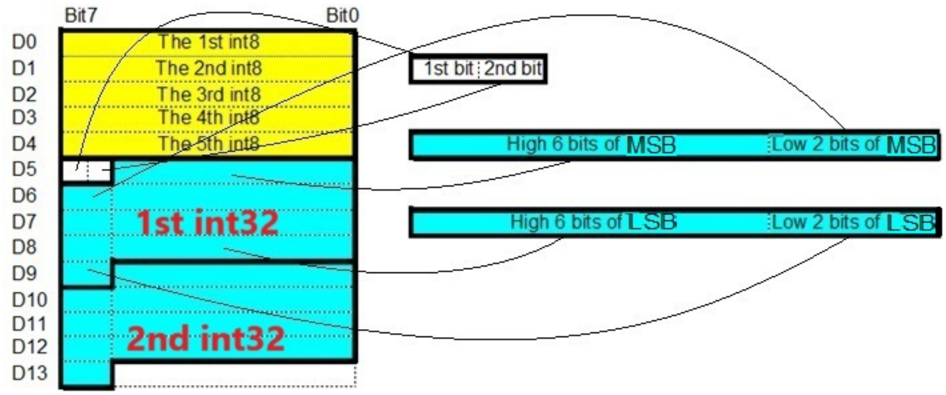
Block "Read CANID:0x12340112x" will do operation 3 in previous operations introduce. Please double click "Read CANID:0x12340112x". You will see parameter settings below:



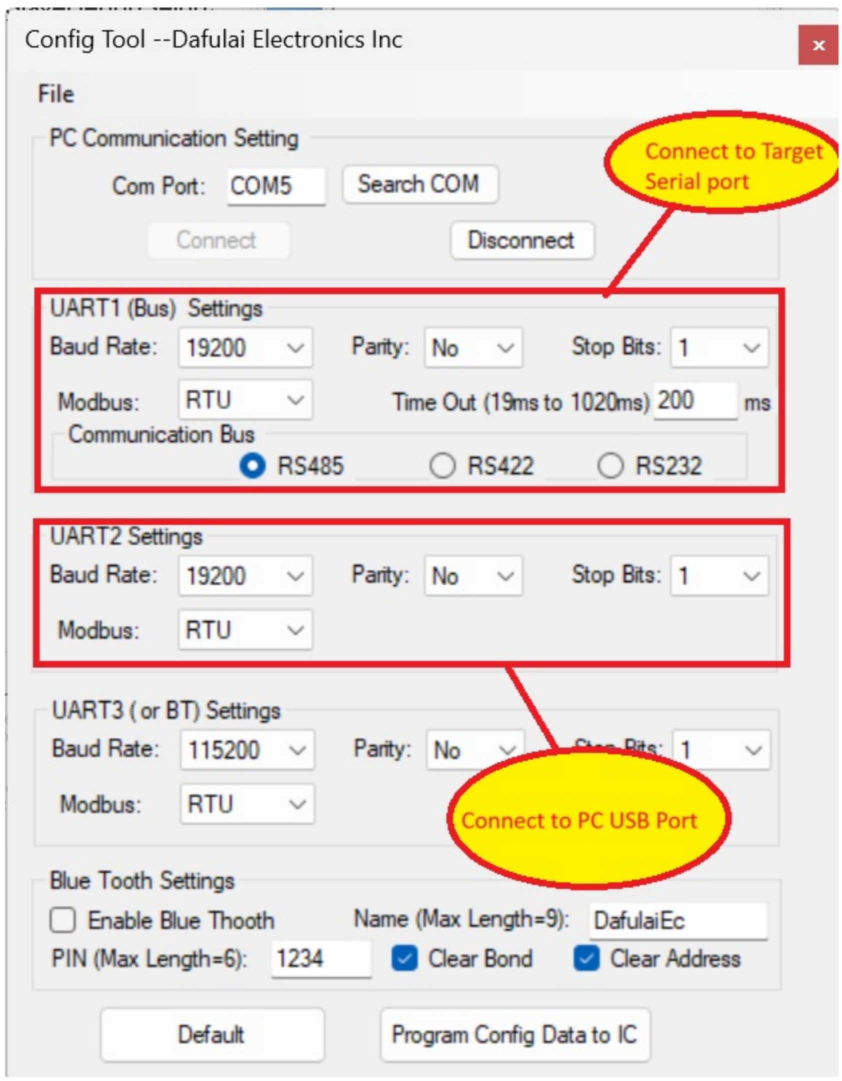
It will receive extended long data frame with CAN ID=0x12340112, and 14 bytes's data field is int8 vector with 5 elements, bool vector with 2 elements, uint32 vector with 2 elements. It is big-endian. Please see CAN bus multiple frames below:

CAN ID	DLC	Data0 (Frame No)	Data1	Data2	Data3	Data4	Data5	Data6	Data7
0x12340112X	8	1	D0	D1	D2	D3	D4	D5	D6
0x12340112X	8	2	D7	D8	D9	D10	D11	D12	D13

And its data structure for big-endian is shown below:

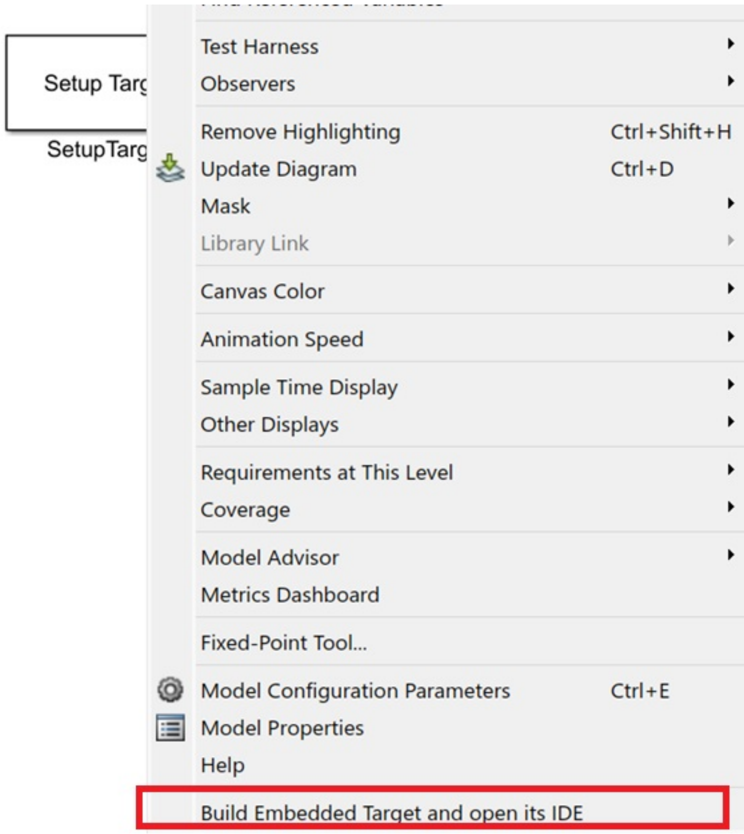


In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:

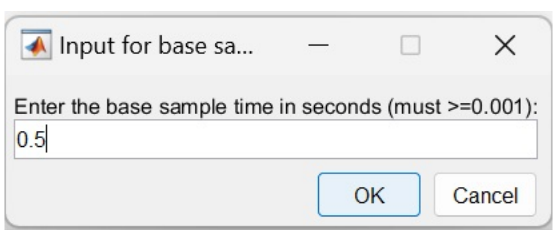


In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

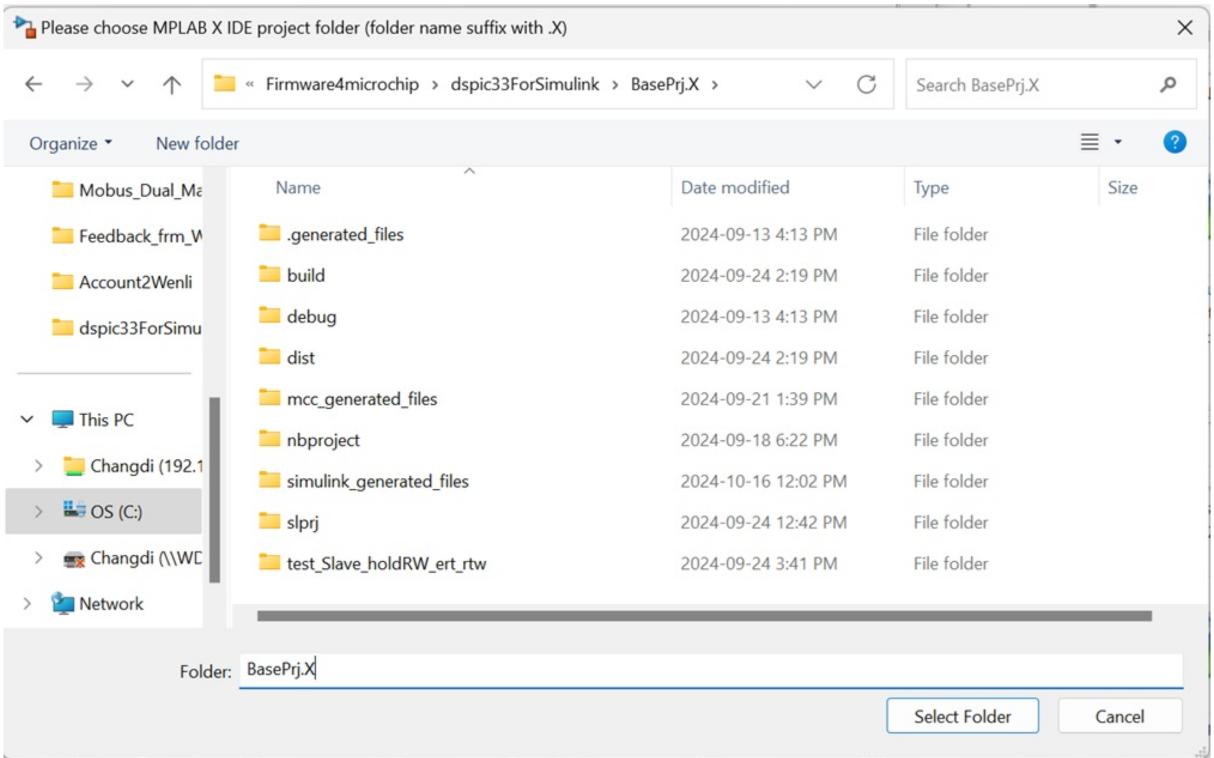
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:



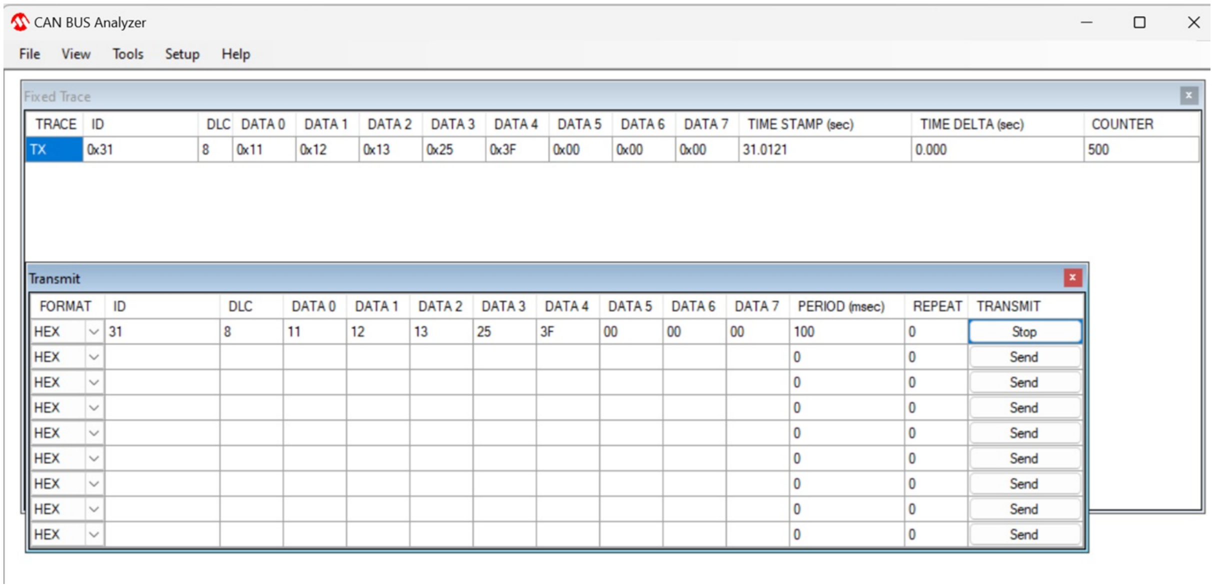
Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

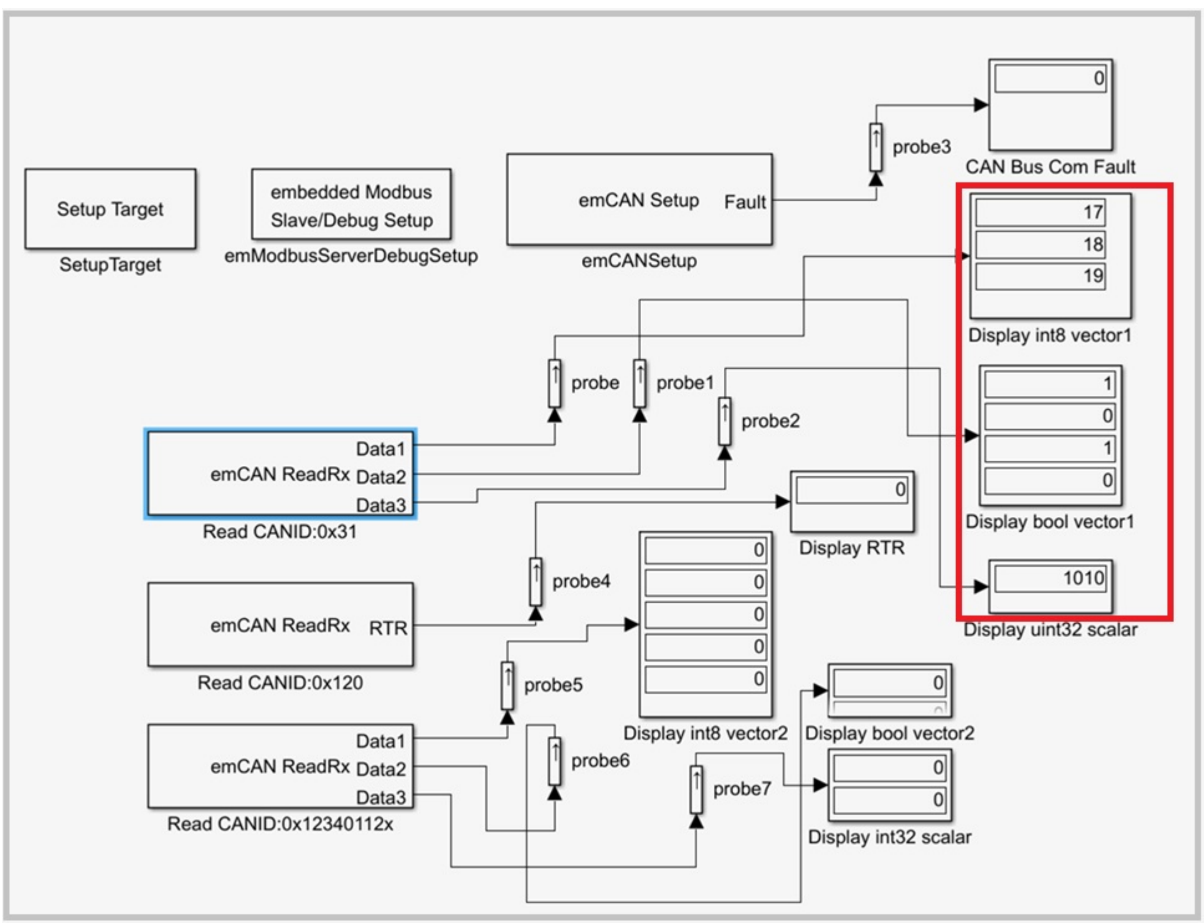
If your firmware program into target successfully, you can use Microchip CAN bus Analyzer or Dafulai Electronic CAN bus Analyzer Hardware/software to transmit CAN BUS frames. Microchip CAN bus Analyzer cannot transmit remote frame, but Dafulai Electronic CAN bus Analyzer can transmit remote frame.

We used Microchip CAN bus Analyzer to test. Run PC Software "CAN BUS Analyzer". Please see screenshot below:



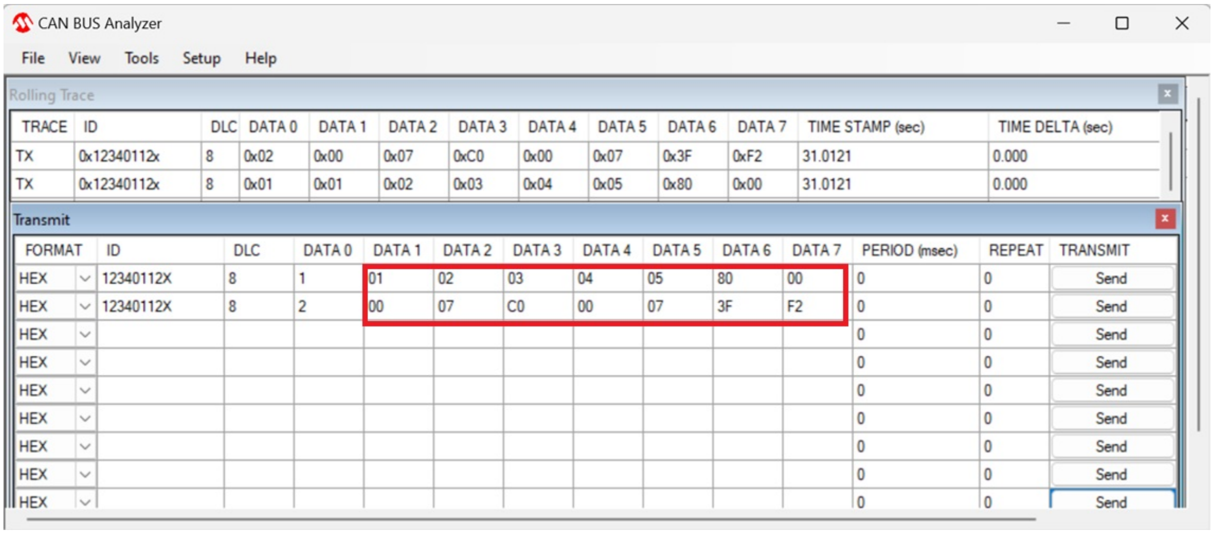
It means that PC transmit CAN bus standard data frame with CAN ID=0x31 and DLC=8, and D0=0x11,D1=0x12, ... , D7=0x00.

We will receive this CAN bus data. We run simulink in PC. The result is shown below:

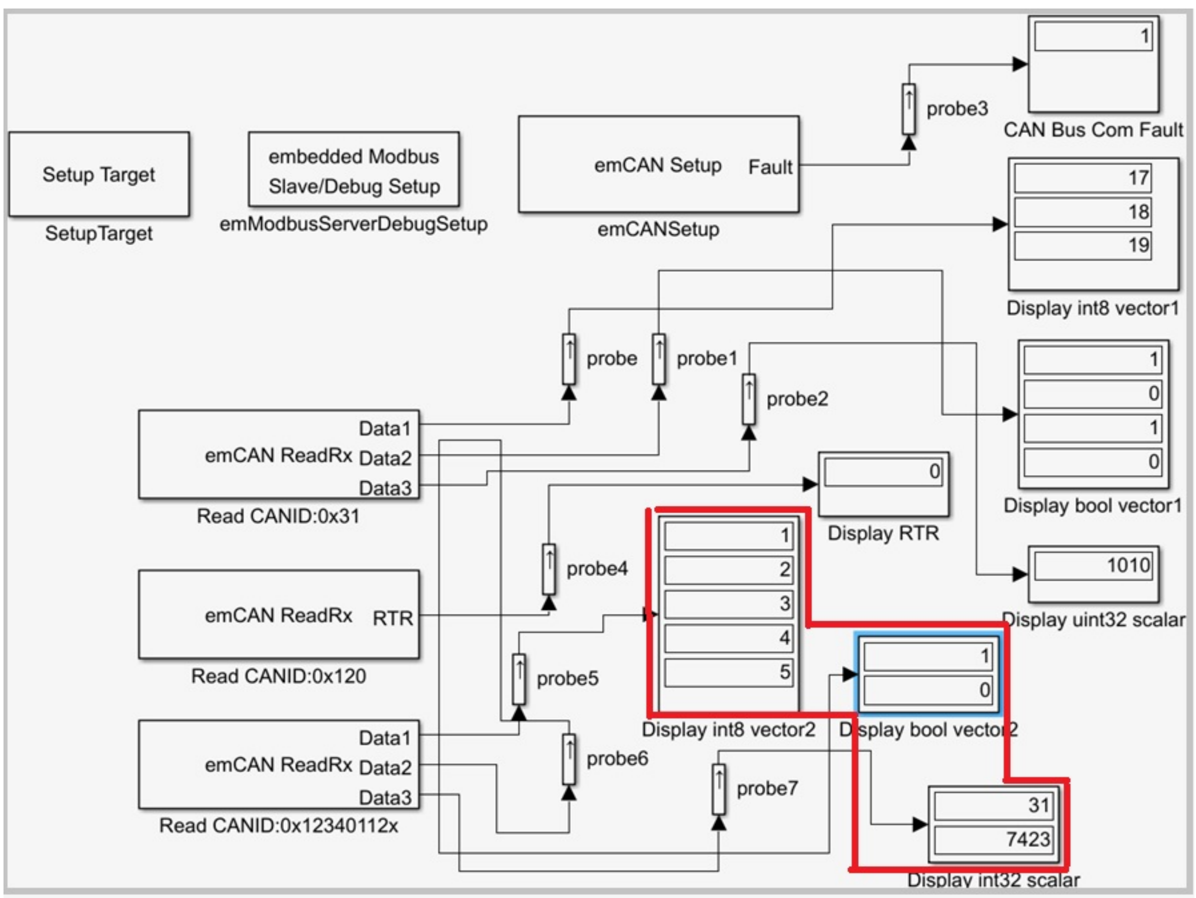


From the result above, it matches what we transmit from PC.

We send out long data frames by CAN Bus analyzer as below:



We saw the result by running simulink, the result is shown below:



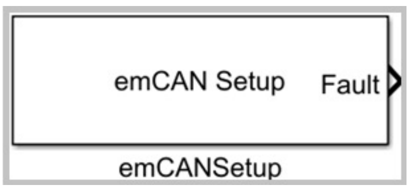
From the result above, it matches what we transmit from PC. But we saw "Fault" output port of "emCANSetup" is true. This is correct because we only send once long data frame , we didn't send any CAN bus frames periodically. So we detect "CAN Bus communication fault".

Please open "Your embedded creator library folder"/examples/example10_emCANRead.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emCANSetup

set up embedded CAN bus.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded CAN Bus / emCANSetup



Description

Set up embedded CAN bus (CAN 2.0 A & B, or CAN FD for TI DSP), and If it has not received any CAN bus packet for some specified time (Time out), output port "Fault" will be true (like communication Watchdog)

Let's introduce Our CAN bus blocks.

CAN Bus Transmission uses Channel conception.

There are 3 Transmission Channels: Channel A, Channel B and Channel C.

Every Channel can have as many as 31 sequences (Seq No is from 0 to 30).

Small Sequence Number has higher priority for transmitting. Every Sequence transmits one CAN bus RTR/Data frame. Or every Sequence can also transmit multiple frames with the same CAN ID but different frame Number in 1st data.

In one channel, transmit CAN bus frame one by one according to Sequence Number from 0 to 30.

All Channels can run at the same time. And they share CAN Bus hardware, Channel A has high priority. Channel B has middle priority, Channel C has low priority

The Seq Number 0 to Seq Number 3 of all channels can also transmit one data frame (short or long) which needs external node respond to. In this situation, only when CAN bus receives response CAN frame(s) or timeout, will this sequence end and start next sequence.

Received CAN bus frames are divided into 2 classes:

Short frame (Data byte length is 0 to 8, or RTR frame)

Long frame (Data byte length is 9 or more, first data byte is used as Frame Number).

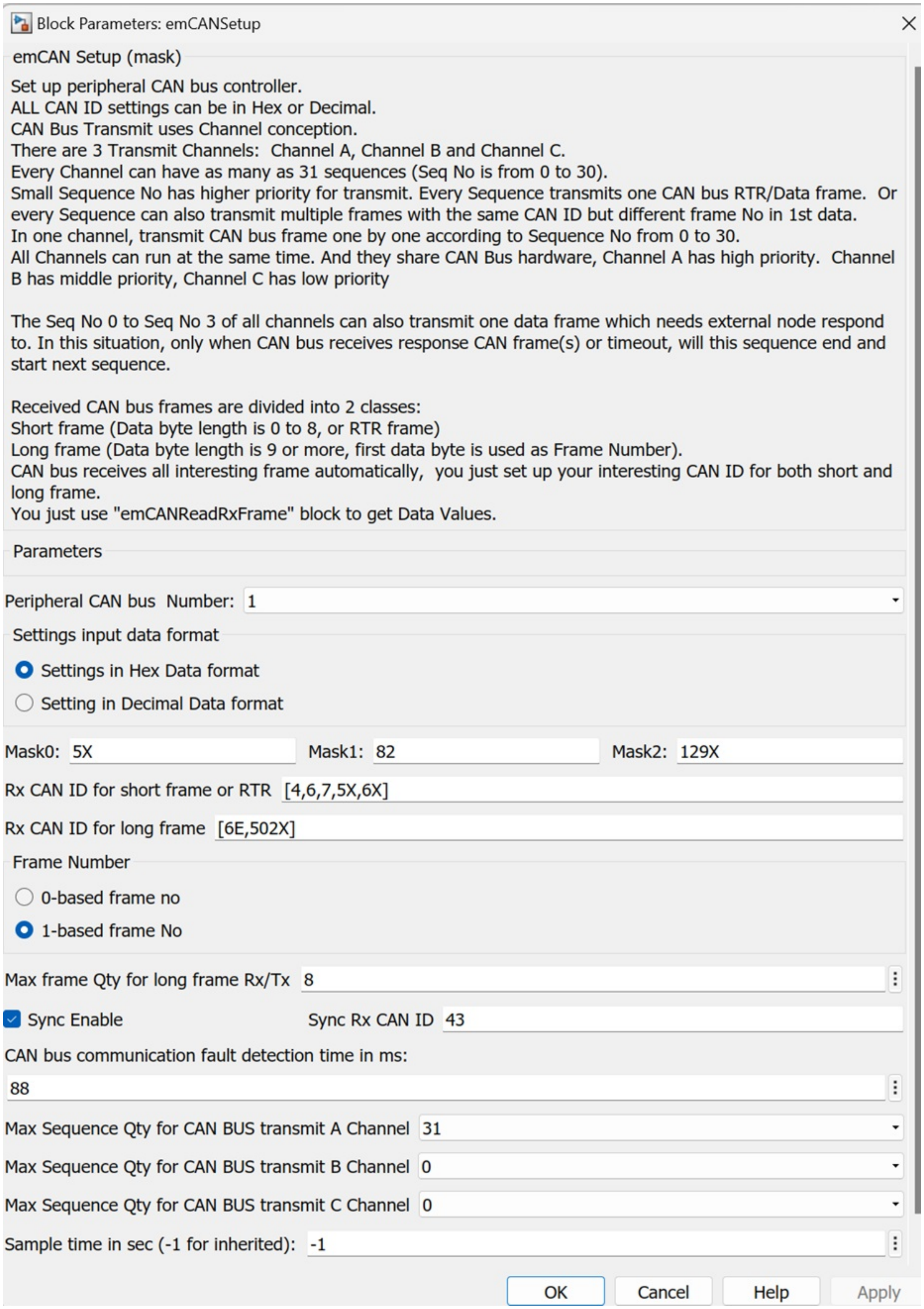
CAN bus receives all interesting frame automatically, you just set up your interesting CAN ID for both short and long frame.

You just use "emCANReadRxFrame" block to get Data Values.

Notes: *1 You must set up CAN bus receiver interrupt enable in MCU Configuration software such as MPLABX IDE MCC for Microchip technology MCUs.*
2 For STM32Cube platform, please use interrupt , don't use DMA for CAN bus.
3 For TI C2000Ware platform, You can enable FD mode, but keep data field maximum 8 bytes length. If we use peripheral 2, but SysConfig only added MCANA, and MCANA does not be used by other emCANSetup block, this block will use MCANA as peripheral. Never use DMA for this block in TI C2000Ware platform.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral CAN bus Number — tell system this block is which CAN controller peripheral used. You just choose from drop list items: 1 or 2.

- Settings input data format — radio button to select CAN ID and Mask Data format is Hex or decimal data. You can change this setting in any time to see data in Hex or Decimal way.

- Mask0 — CAN Bus receiver's the 1st Mask register. If bit of Mask register is 1, CAN Bus receiver will only receive CAN ID which related bit is equal to filter bit value.   Suffix "X" or "x" denotes Mask is Extended 29 bits of data. Mask0/1/2 are only used for dspic33. For other platforms, just ignore them.

- Mask1 — CAN Bus receiver's the 2nd Mask register. If bit of Mask register is 1, CAN Bus receiver will only receive CAN ID which related bit is equal to filter bit value.   Suffix "X" or "x" denotes Mask is Extended 29 bits of data. Mask0/1/2 are only used for dspic33. For other platforms, just ignore them.

- Mask2 — CAN Bus receiver's the 3rd Mask register. If bit of Mask register is 1, CAN Bus receiver will only receive CAN ID which related bit is equal to filter bit value.   Suffix "X" or "x" denotes Mask is Extended 29 bits of data. Mask0/1/2 are only used for dspic33. For other platforms, just ignore them.

- Rx CAN ID for short frame or RTR — It is CAN IDs for receiver's short frame. Short frame means RTR frame or Data frame with data byte QTY<=8. You should use row vector for all CAN IDs. If only one CAN ID you are interested in, you can use scalar. If CAN ID is extended frame, please use suffix "X" or "x". Please don't use any quotation mark. And you must make sure received CAN IDs from the filters and masks in IDE GUI Configuration tool (such as MCC for microchip MCU) cover these parameters.

- Rx CAN ID for long frame — It is CAN IDs for receiver's long frames. Long frame means multiple data frames and the first data byte is frame number. It can be used Data bytes QTY is bugger than 8. You should use row vector for all CAN IDs. If only one CAN ID you are interested in, you can use scalar. If CAN ID is extended frame, please use suffix "X" or "x". Please don't use any quotation mark. And you must make sure received CAN IDs from the filters and masks in IDE GUI Configuration tool (such as MCC for Microchip MCU) cover these parameters.

- Frame Number — It is radio button to select the first frame number for long frame. Two different first frame numbers are 0 and 1.

- Max frame Qty for long frame Rx/Tx — Because frame number for long frame occupies 1 byte, so for 0-based frame number, you can have maximum 256 frames for one long frame. For 1-based frame number, you can have maximum 255 frames for one long frame. Actually, for specific project, we only need a few of frames for one long frame. This parameter specifies the maximum frame Qty for long frame transmitting and receiving in order to save embedded MCU memory resource.

- Sync Enable — When it is true, embedded CAN Bus controller only transmits when it receives a special CAN ID called "Sync Rx frame". In general, if we only have One transmitting Channel, then received Sync CAN ID rate is usually a little smaller than Channel transmitting rate. In this way, Every transmitting will occur when received "Sync Rx frame". If we have multiple transmitting Channels, and received Sync CAN ID rate is a little smaller than the one of the most slowest transmitting rate Channel. Then only the most slowest transmitting rate Channel will synchronize with "Sync Rx frame". The other channels will transmit at any time no matter whether it received "Sync Rx frame" periodically if it received one time "Sync Rx frame".

- Sync Rx CAN ID — CAN ID of "Sync Rx frame"'. Suffix "x" or "X" denotes extended 29 bits CAN ID. If "Sync Enable" is true, embedded CAN Bus controller only transmits when it receives this CAN ID frame.

- CAN bus communication fault detection time in ms: — When embedded CAN bus Controller didn't receive any valid CAN bus frame which is filtered by Filter and mask registers in specified time, this block output port "Fault" becomes "true" to indicate "CAN Bus communication fault". This specified time is called "CAN bus communication fault detection time". It is similar to communication "Watchdog".

- Max Sequence Qty for CAN BUS transmit A Channel — We have introduced this embedded CAN bus function at beginning. Every Channel can have as many as 31 sequences (Seq No is from 0 to 30). However, in order to decrease memory consumption for embedded target, we can specify small sequences QTY. This parameter specifies actual sequences QTY for Channel A.

- Max Sequence Qty for CAN BUS transmit B Channel — We have introduced this embedded CAN bus function at beginning. Every Channel can have as many as 31 sequences (Seq No is from 0 to 30). However, in order to decrease memory consumption for embedded target, we can specify small sequences QTY. This parameter specifies actual sequences QTY for Channel B.

- Max Sequence Qty for CAN BUS transmit C Channel — We have introduced this embedded CAN bus function at beginning. Every Channel can have as many as 31 sequences (Seq No is from 0 to 30). However, in order to decrease memory consumption for embedded target, we can specify small sequences QTY. This parameter specifies actual sequences QTY for Channel C.

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

- Fault — "logical" data type's scalar. "True" means CAN bus controller didn't receive any valid CAN bus frame which is filtered by Filter and mask registers in time specified by parameter "CAN bus communication fault detection time in ms".

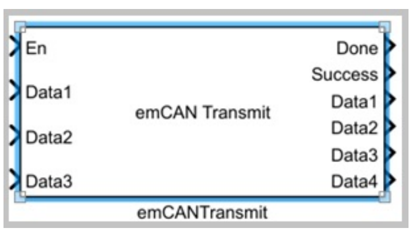
Examples

Please see "emCANReadRxFrame" block example, and "emCANTransmit" block example.

emCANTransmit

Transmit CAN bus frame by one channel and one sequence.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded CAN Bus / emCANTransmit



Description

Transmit CAN bus Data or Remote frame by one sequence from Channel A or Channel B or Channel C.

We can transmit CAN Bus Data/Remote frame in 3 different Channels at the same time.

(Of cause, you'd better configure 3 buffers for transmitting in GUI software like Microchip MCC and STM32CubeIDE).

In general, you can set 3 Channels into different sample time.

Inside every channels, transmitting is in serial by Sequences. Every channel has maximum 31 Sequences: Seq0 to Seq30.

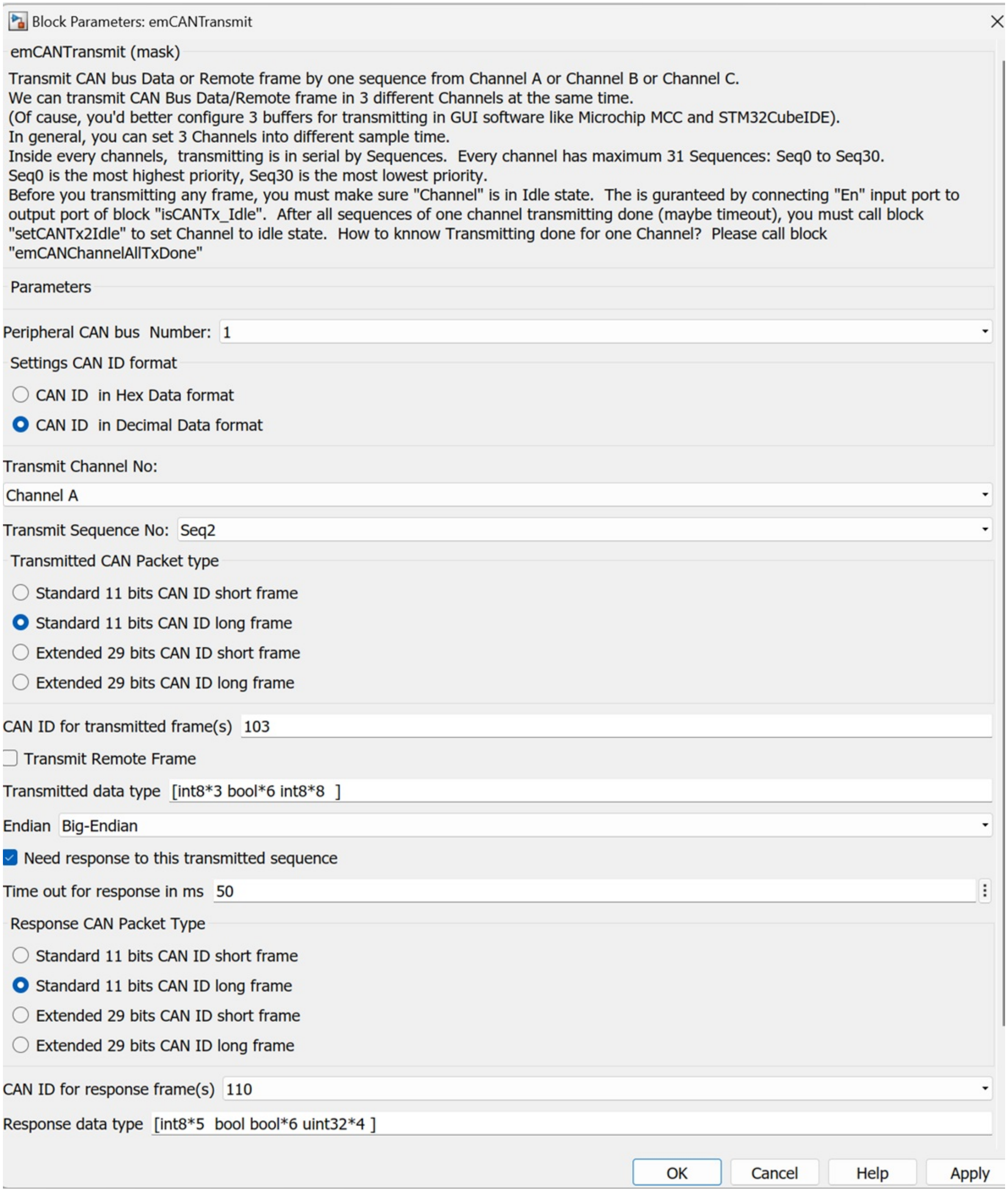
Seq0 is the most highest priority, Seq30 is the most lowest priority.

Before you transmitting any frame, you must make sure "Channel" is in Idle state. The is guaranteed by connecting "En" input port to output port of block "isCANTx_Idle". After all sequences of one channel transmitting done (maybe timeout), you must call block "setCANTx2Idle" to set Channel to idle state. How to know Transmitting done for one Channel? Please call block "emCANChannelAllTxDone"

Notes: *You 'd better set up 3 CAN bus transmitting buffers if you use 3 transmission Channels in MCU Configuration software such as MPLABX IDE MCC for Microchip technology MCUs. Of cause, if you only use one or two Channels, you can setup only one or two CAN bus transmitting buffers.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral CAN bus Number — tell system this block is which CAN controller peripheral used. You just choose from drop list items: 1 or 2.

- Settings input data format — radio button to select CAN ID format is Hex or decimal. You can change this setting in any time to see CAN ID in Hex or Decimal way.

- Transmit Channel No: — drop list to select transmission Channel A or Channel B or Channel C.

- Transmit Sequence No:— drop list to select transmission sequence Number from Seq0 to Maximum Sequence.

- Transmitted CAN Packet type —radio button to select transmission CAN ID type. It can be one of "Standard 11 bits CAN ID short frame", "Standard 11 bits CAN ID long frame", "Standard 29 bits CAN ID short frame" and "Standard 29 bits CAN ID long frame"

- CAN ID for transmitted frame(s) — CAN Bus transmitting CAN ID. This is 11bits or 29 bits of scalar integer.

- Transmit Remote Frame — tell this block whether transmitted CAN bus is remote frame.

- Transmitted data type — It specifies data type of data field for this transmitted CAN bus data frame. It can be scalar or vector. element can be "bool","uint8", "int8","uint16","int16","uint32","int32","single". How many vector elements it has is how many data input ports it has.If data input port is vector, please follow * and vector element Qty. For example, [int8, uint16*2, bool*7] denote that data field has 6 bytes, and block has 3 input data port. The 1st input port "Data1" is "int8" type of scalar. The 2nd input port "Data2" is "uint16" type of vector with 2 elements. The 3rd input port "Data3" is "bool" type of vector with 7 elements. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["int8", "uint16*2", "bool*7"], You cannot use " [int8, uint16*2, bool*7]". You must use [int8, uint16*2, bool*7].

  .

- Endian — It is bool type vector of bit order and "int16/uint16/uint32/int32/single" type of byte order. For example, Transmitted data type parameter=[bool*3 uint16]. It occupies 3 bytes. D0 denotes the 1st byte of data field. D1 denotes the 2nd byte of data field. D2 denotes the 3rd byte of data field. D0.0 denotes D0's LSb (less significant bit). D0.7 denotes D0's MSb (most significant bit). The input port "Data1" is bool type of vector with 3 elements. The input port "Data2" is uint16 type of scalar. When we use "Little-Endian", Data1=[D0.0 D0.1 D0.2] and Data2= D2.2 D2.1 D2.0 D1.7 D1.6 D1.5 D1.4 D1.3 D1.2 D1.1 D1.0 D0.7 D0.6 D0.5 D0.4 D0.3. When we use "Big-Endian", Data1=[D0.7 D0.6 D0.5] and Data2= D0.4 D0.3 D0.2 D0.1 D0.0 D1.7 D1.6 D1.5 D1.4 D1.3 D1.2 D1.1 D1.0 D2.7 D2.6 D2.5

- Need response to this transmitted sequence — Logical scalar. True means that this sequence will wait for special CAN Bus frame received after this sequence's transmission done. When specified CAN Bus frame received or timeout, this sequence ends and it will start next sequence. Seq0 to Seq3 can be set to true for this parameter. The other sequences cannot be set to true for this parameter. Only when this parameter is true, has it

"Data1" to "Datan" output port to denote response data.

- Time out for response in ms — unsigned 16 bits of integer to denote timeout for transmission frame which needs response frame.
- Response CAN Packet Type — radio button to select response CAN ID type. It can be one of "Standard 11 bits CAN ID short frame", "Standard 11 bits CAN ID long frame", "Standard 29 bits CAN ID short frame" and "Standard 29 bits CAN ID long frame"
- CAN ID for response frame(s) — response CAN ID. This is 11bits or 29 bits of scalar integer.
- Response data type — It specifies data type of data field for CAN bus response data frame. It can be scalar or vector. element can be "bool","uint8", "int8","uint16","int16","uint32","int32","single". How many vector elements it has is how many data output ports it has.If data output port is vector, please follow * and vector element Qty. For example, [int8, uint16*2, bool*7] denote that data field has 6 bytes, and block has 3 output data port. The 1st output port "Data1" is "int8" type of scalar. The 2nd output port "Data2" is "uint16" type of vector with 2 elements. The 3rd output port "Data3" is "bool" type of vector with 7 elements. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["int8", "uint16*2", "bool*7"], You cannot use " [int8, uint16*2, bool*7]". You must use [int8, uint16*2, bool*7].
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- En — "logical" data type's scalar. "True" means CAN bus controller will transmit. And it will make this transmission channel into "busy" state. In general, "En" connects the output port of "isCANTx_Idle" block.

For other data input ports, it depends on parameter "Transmit Remote Frame" and "Transmitted data type". When "Transmit Remote Frame" is false, parameter "Transmitted data type" vector element QTY is Input data port QTY. Please read parameter "Transmitted data type" above. and see description below

- Data1 — vector or scalar. Parameter "Transmitted data type" vector's first element decides what data type it is. and " *digit" decides vector size

.

- Data2 — vector or scalar. Parameter "Transmitted data type" vector's second element decides what data type it is. and " *digit" decides vector size.
- Data3 — vector or scalar. Parameter "Transmitted data type" vector's third element decides what data type it is. and " *digit" decides vector size.

......

- Data n — vector or scalar. Parameter "Transmitted data type" vector's number n element decides what data type it is. and " *digit" decides vector size.

Outport

- Done — "logical" data type's scalar. "True" means CAN bus controller has done this sequence transaction. But it may successes or fails.

- Success — "logical" data type's scalar. "True" means CAN bus controller has done this sequence transaction successfully.

For other data output ports,It depends on parameter "Need response to this transmitted sequence". When "Need response to this transmitted sequence" is true, there are data output ports as below:

- Data1 — vector or scalar. Parameter "Response data type" vector's first element decides what data type it is. and " *digit" decides vector size

.

- Data2 — vector or scalar. Parameter "Response data type" vector's second element decides what data type it is. and " *digit" decides vector size.

- Data3 — vector or scalar. Parameter "Response data type" vector's third element decides what data type it is. and " *digit" decides vector size.

......

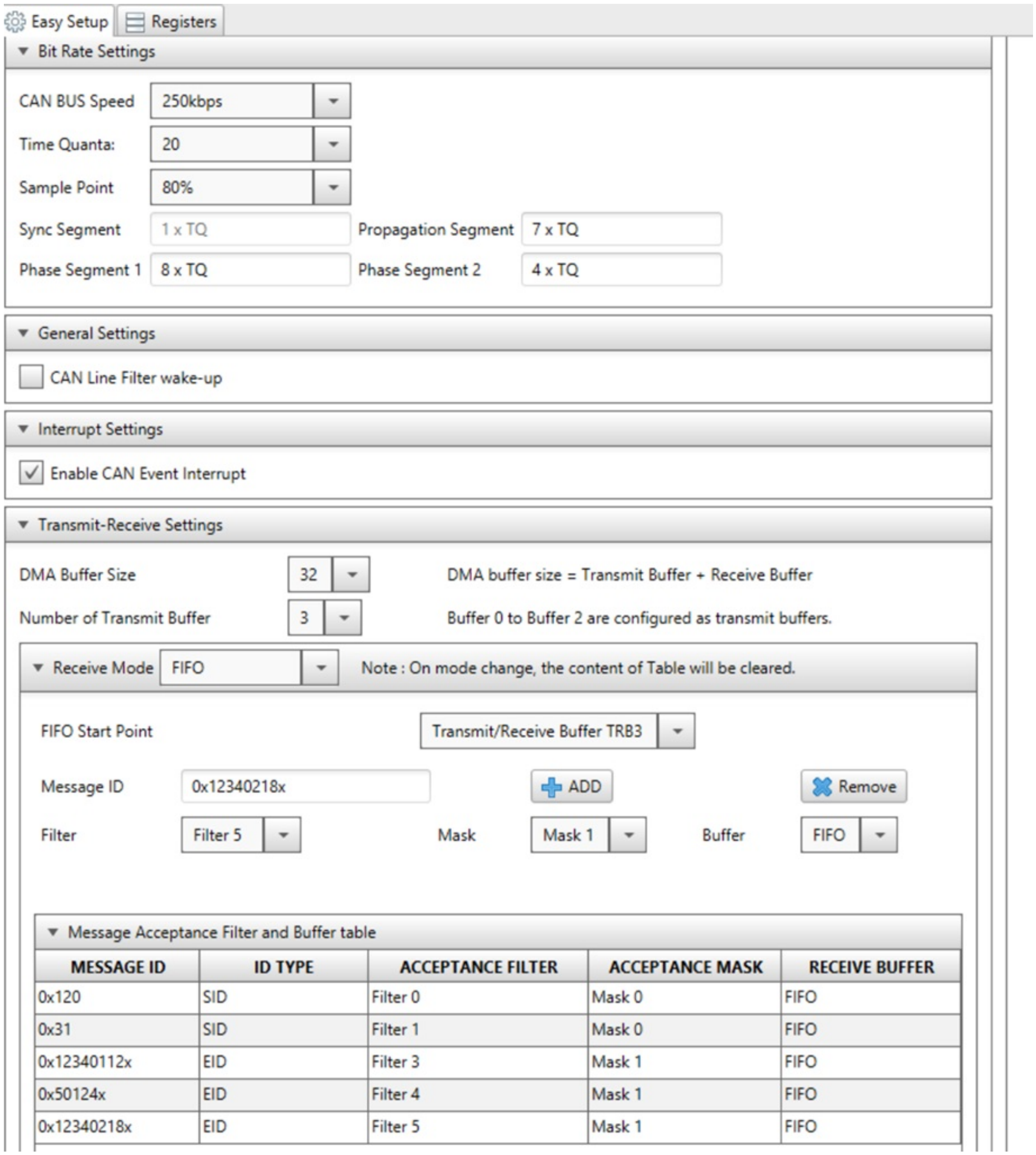
- Data n — vector or scalar. Parameter "Response data type" vector's number n element decides what data type it is. and " *digit" decides vector size.

Examples

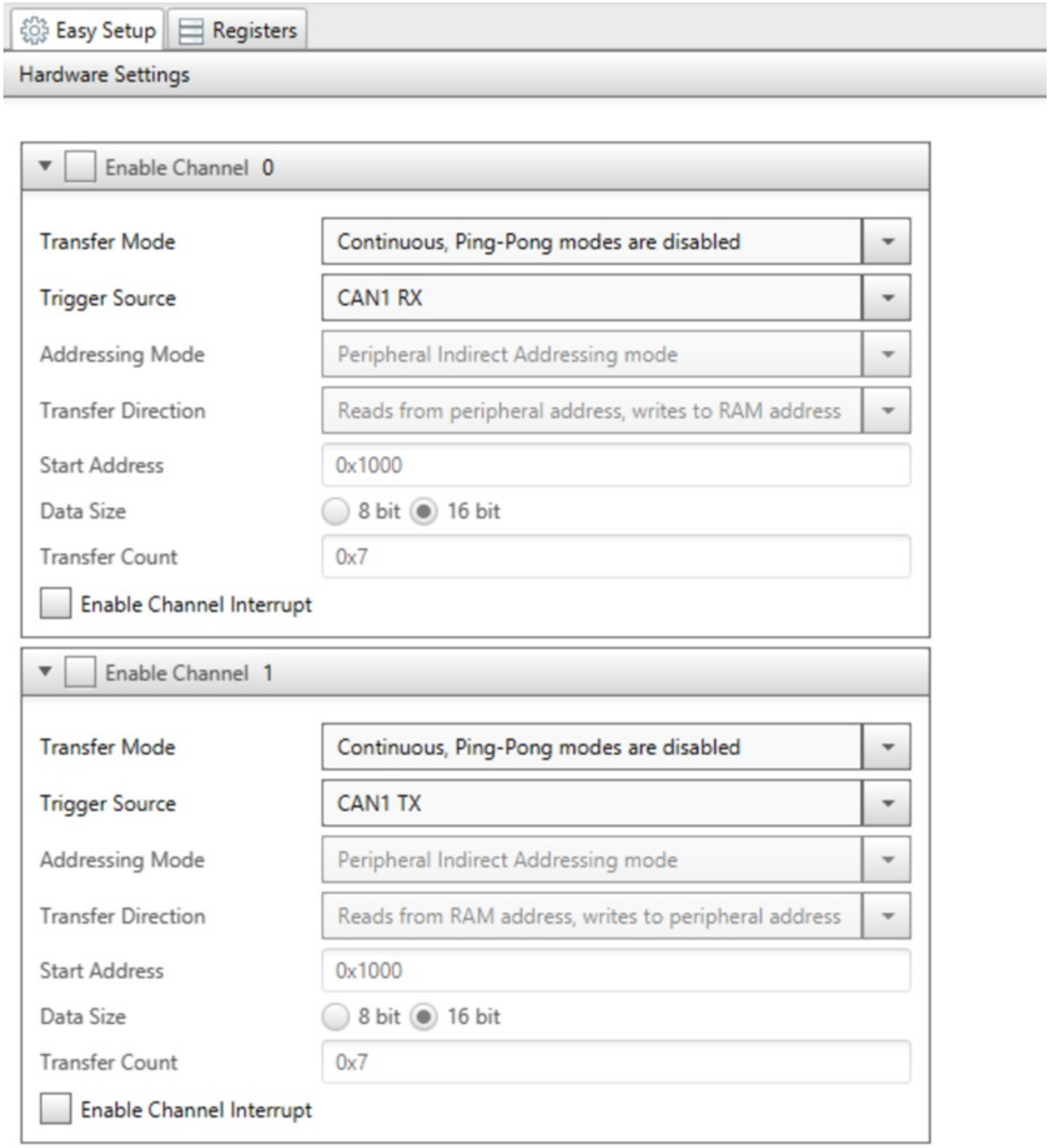
Example1:

Our embedded platform is dsPIC33EP256GP502 . CAN bus peripheral 1 is used with baud rate= 250000 bit/sec. In MPLABX IDE MCC, we set F0= 0x120 std frame used Mask0, F1= 0x31 std frame used Mask0, F1= 0x12340112 ext frame used Mask1, F2= 0x50124 ext frame used Mask1, F1= 0x12340218 ext frame used Mask1.
Number of Transmit Buffer is 3 from Buffer0 to Buffer2.

Please see MCC configuration for can1 below:



Please be careful, For dspic/Pic24 MCU, CAN bus use DMA to transfer data. We show you the DMA configuration in MPLABX IDE MCC for can1 below:

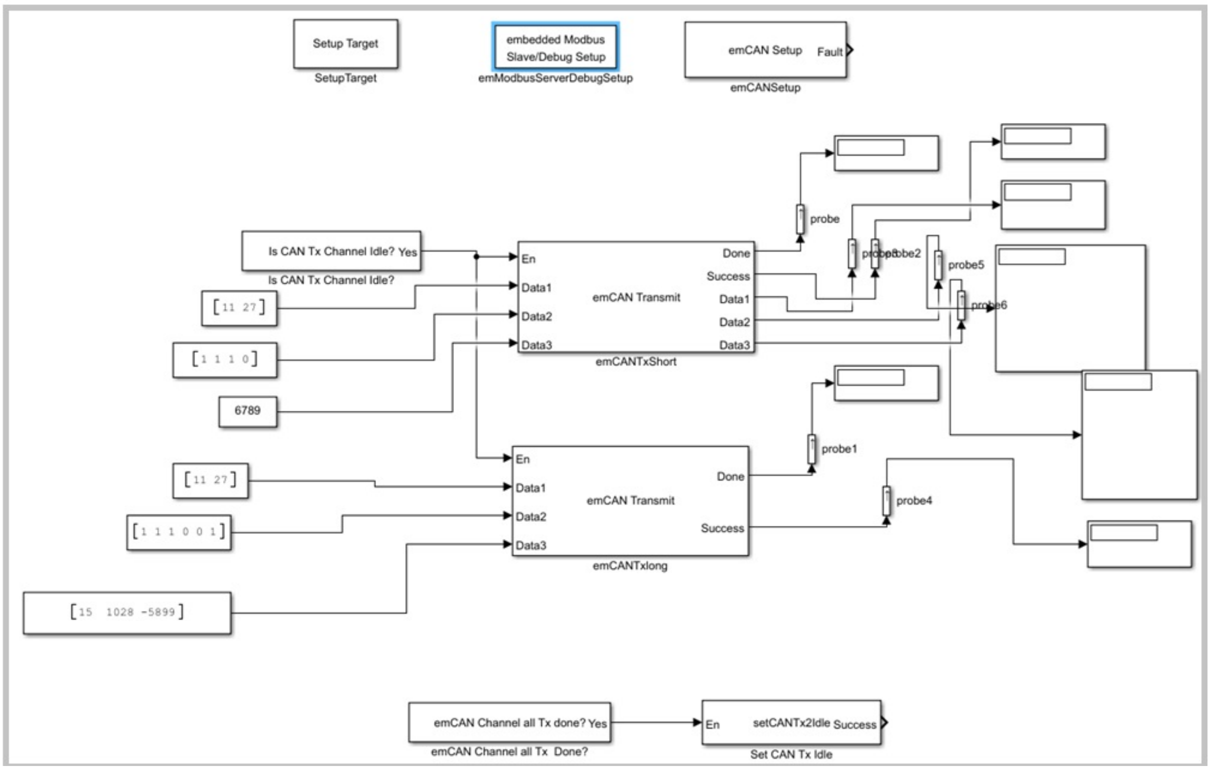


Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. This RS485 is used for Simulink Debug. This example code will read CAN frame every 0.5 seconds. The following operations will be doing during sample time:

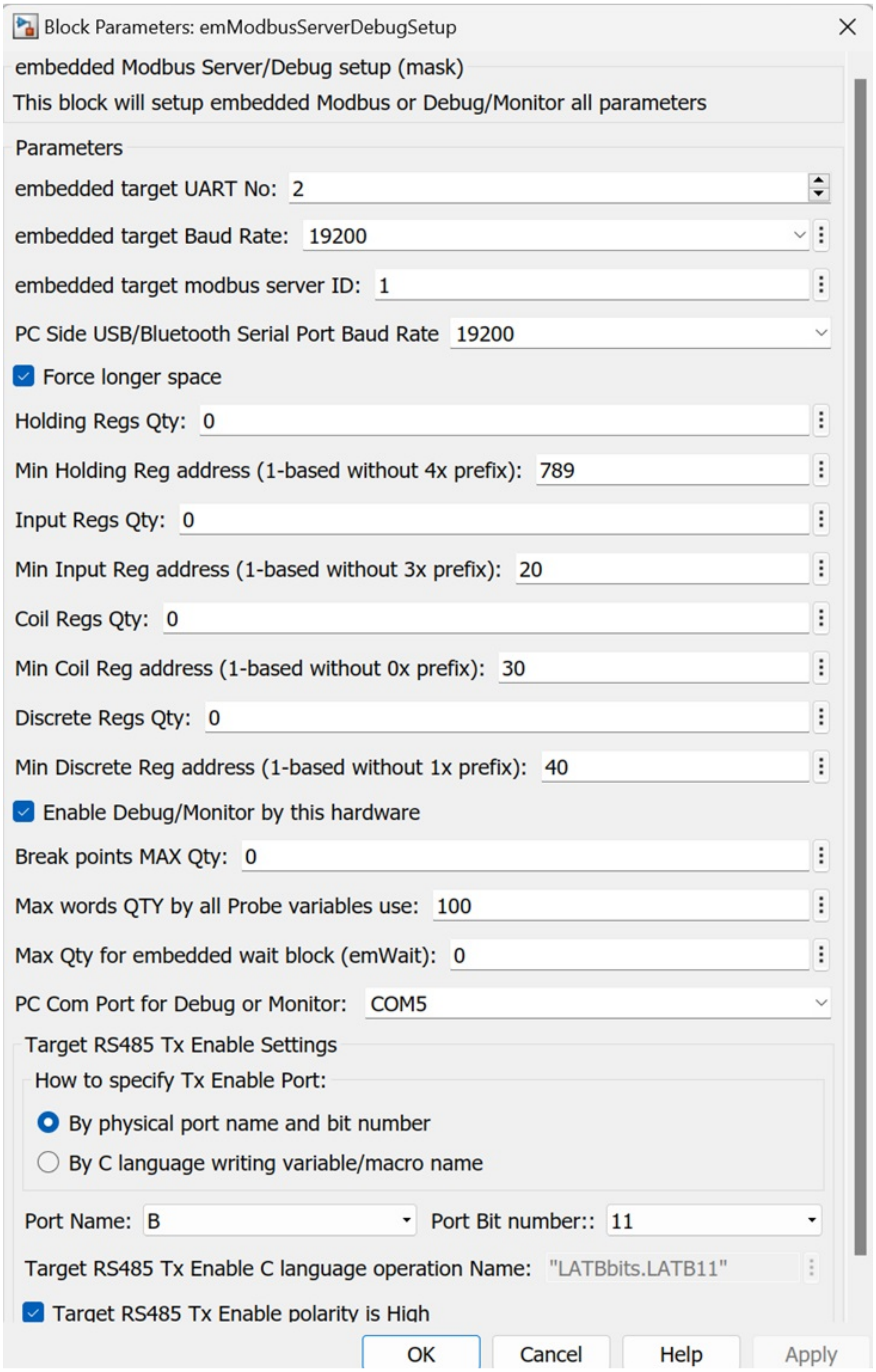
1. Read CAN ID=0x31 standard data frame with DLC=8 (short data frame), little-endian, 3 Data output ports: the 1st output port "Data1" is int8 scalar, the 2nd output port "Data2" is bool vector with 4 elements,the 3rd output port "Data3" is uint32 scalar,
2. Read CAN ID=0x120 standard remote frame.
3. Read CANID=0x12340112 extended Data frame (long data frame), Big-Endian, Total Data length=14 bytes. 3 Data output ports: the 1st output port "Data1" is int8 vector with 5 elements, the 2nd output port "Data2" is bool vector with 2 elements, the 3rd output port "Data3" is int32 vector

with 2 elements,

Please see screenshot of model below:

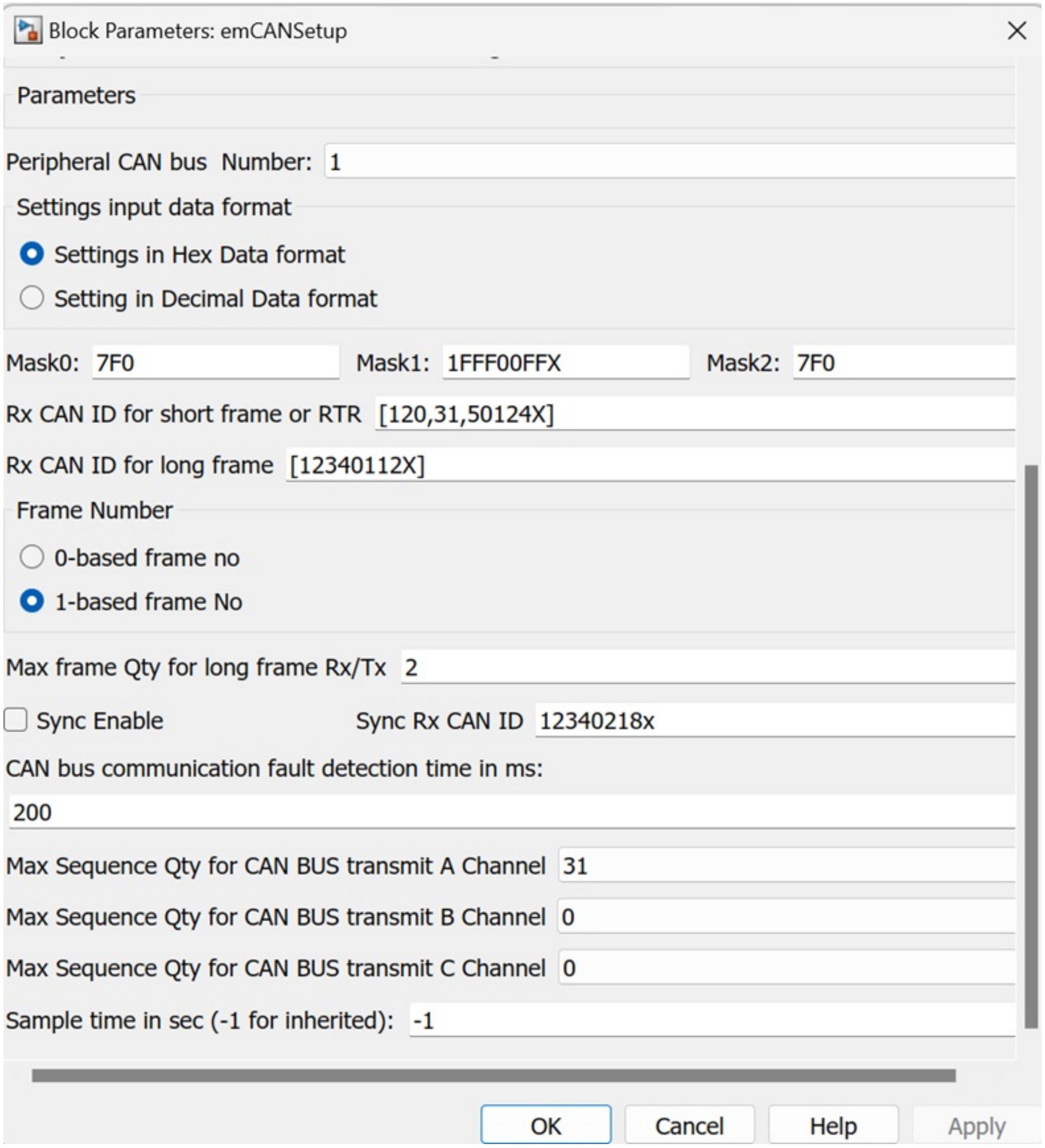


In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Please double click on "emModbusServerDebugSetup" block.



We only use this block for "Probe". So the QTY of modbus server registers is set to 0.

Please double click on "emCANSetup" block. We will see parameters for CAN bus controller below:



Let's explain settings. Mask0= 7F0, it has no suffix "X" or "x". So Mask0 will be standard 11 bits. In previous MCC configuration, Filter0=0x120 and use Mask0. We can know this CAN bus controller can receive standard frame with CAN ID= 0x120 to 0x12F because MASK0 the most right-handed 4 bits are 0 (means we don't care CAN ID's most right-handed 4 bits). In previous MCC configuration, Filter1=0x31 and use Mask0. We can know this CAN bus controller can receive standard frame with CAN ID= 0x30 to 0x3F because MASK0 the most right-handed 4 bits are 0 (means we don't care CAN ID's most right-handed 4 bits).

Mask1=1FFF00FFX, it has suffix "X". So Mask1 will be extended 29 bits. In previous MCC configuration, Filter3=0x12340112 and use Mask1. We can know this CAN bus controller can receive extended frame with CAN ID= 0x12340012 to 0x1234FF12 because MASK1 Bit8 to Bit15 are 0 (means we don't care CAN ID's Bit8 to Bit15).

In previous MCC configuration, Filter4=0x50124 and use Mask1. We can know this CAN bus controller can receive extended frame with CAN ID= 0x50024 to 0x5FF24 because MASK1 Bit8 to Bit15 are 0 (means we don't care CAN ID's Bit8 to Bit15).

In previous MCC configuration, Filter5=0x12340128 and use Mask1. We can know this CAN bus controller can receive extended frame with CAN ID= 0x12340028 to 0x1234FF28 because MASK1 Bit8 to Bit15 are 0 (means we don't care CAN ID's Bit8 to Bit15).

As for Mask2, No filter uses Mask2 in MCC configuration. So we don't care Mask2 value.

Parameter "Rx CAN ID for short frame or RTR" is [120, 31, 50124X]. It means CAN Bus controller for Simulink will receive short standard data/Remote frame with CAN ID=0x120 and 0x31, and short extended data/Remote frame with CAN ID=0x50124. These CAN IDs are covered by MCC configuration,

Parameter "Frame Number" is "1-based frame No". It means that the 1st frame's frame number (the 1st byte of data field) is 1 for long data frame. Of cause, the 2nd frame's frame number (the 1st byte of data field) is 2 for long data frame.

Parameter "Max frame Qty for long frame Rx/Tx" is 2. It means that the longest data byte QTY is 14 (7 bytes each frame, Total 2 frames will have total 14 bytes).

Parameter "Sync Enable " is false. It means that transmitting occurs at any time. .

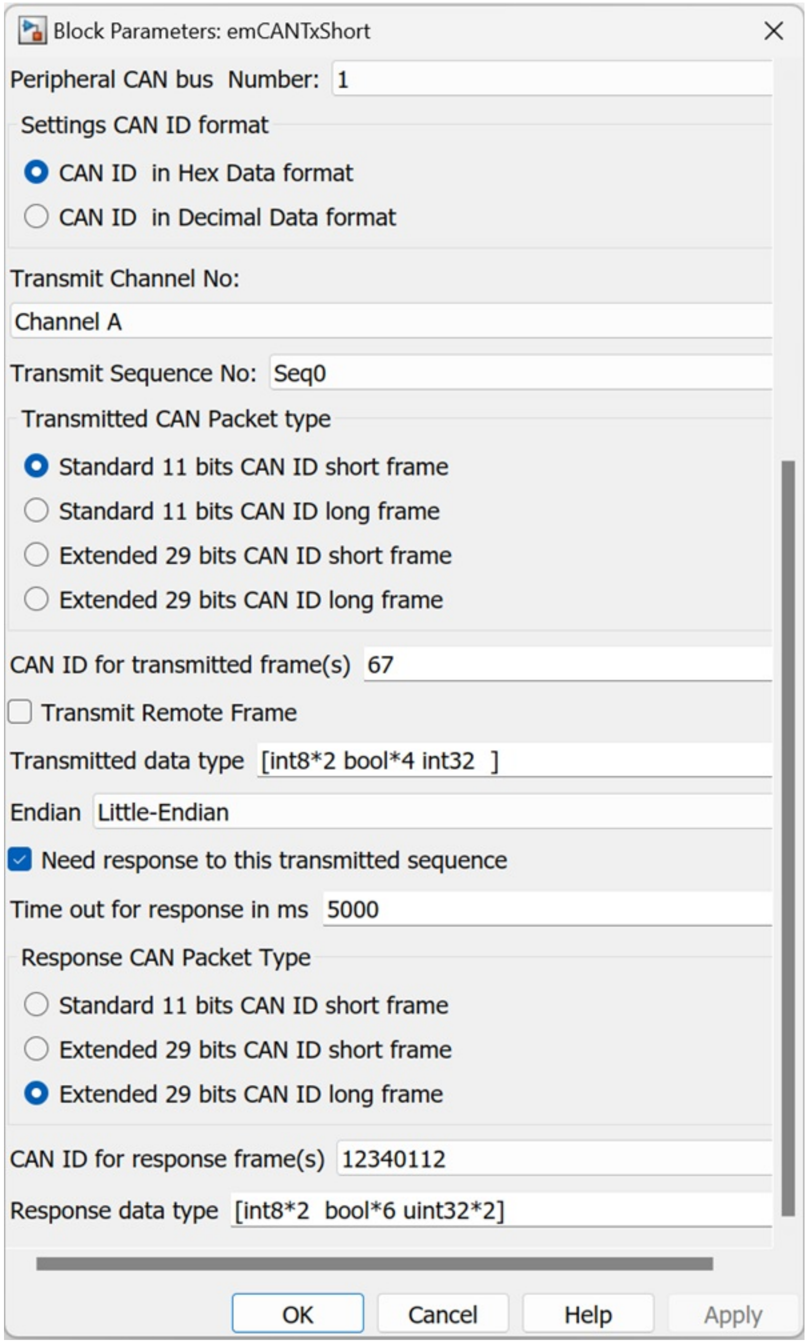
Parameter "Sync Rx CAN ID" is no use for this example because we disable "Sync".

Parameter "Max Sequence Qty for CAN BUS transmit A Channel" is 31, So we can use Seq0 to Seq30 of Channel A to transmit. But this parameter is no use for this example because we do not transmit any CAN bus frame.

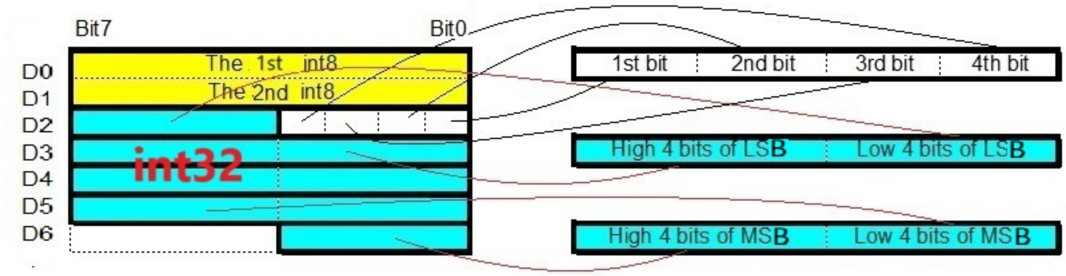
Parameter "Max Sequence Qty for CAN BUS transmit B Channel" is 0, So we cannot use Channel B to transmit. But this parameter is no use for this example because we do not transmit any CAN bus frame

Parameter "Max Sequence Qty for CAN BUS transmit C Channel" is 0, So we cannot use Channel C to transmit. But this parameter is no use for this example because we do not transmit any CAN bus frame. "Sample time in Sec" is -1 which means sample time inherit. (use base sample time 0.5 sec):

Double click block "emCANTxShort", we will see the parameter settings below:



It means that this block will transmit standard short data frame with CAN ID=0x67. Because parameter "Transmitted data type"=[int8*2 bool*4 int32]. So DLC=2+1+4=7, and 3 data input ports. The 1st Data input is int8 vectors with 2 elements. The 2nd Data input is bool vectors with 4 elements. The 3rd Data input is int32 scalar. It is Little-endian. Please see data field of this CAN bus frame below:

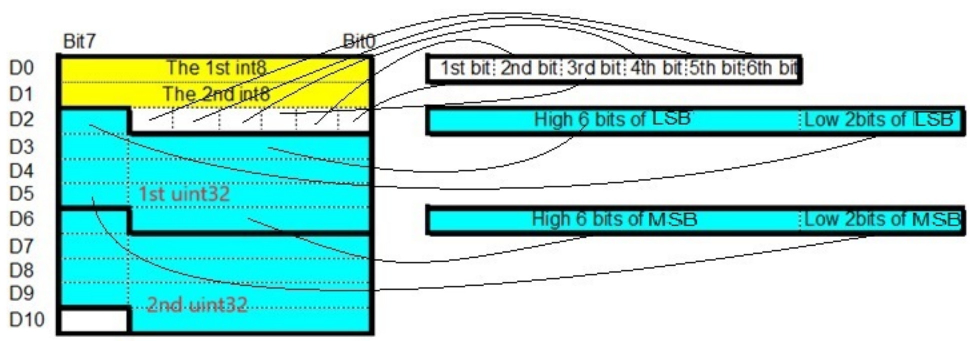


The yellow color is int8 vector data area. The white color is bool vector area (Bit0 is the 1st element of bool vector). The blue color is uint32 scalar area.

The parameter "Need response to this transmitted sequence" is true. So after transmitting, controller will wait for received special CAN bus frames which is extended CAN ID =12340112 and this response CAN data frame is long data frame. Response Data type=[int8*2 bool*6 uint32*2]. It means it has 3 Data output ports, the 1st data output port "Data1" is int8 vector with 2 elements, the 2nd data output port "Data2" is bool vector with 6 elements, the 3rd data output port "Data3" is uint32 vector with 2 elements. Please see response CAN bus data long frame structure below:

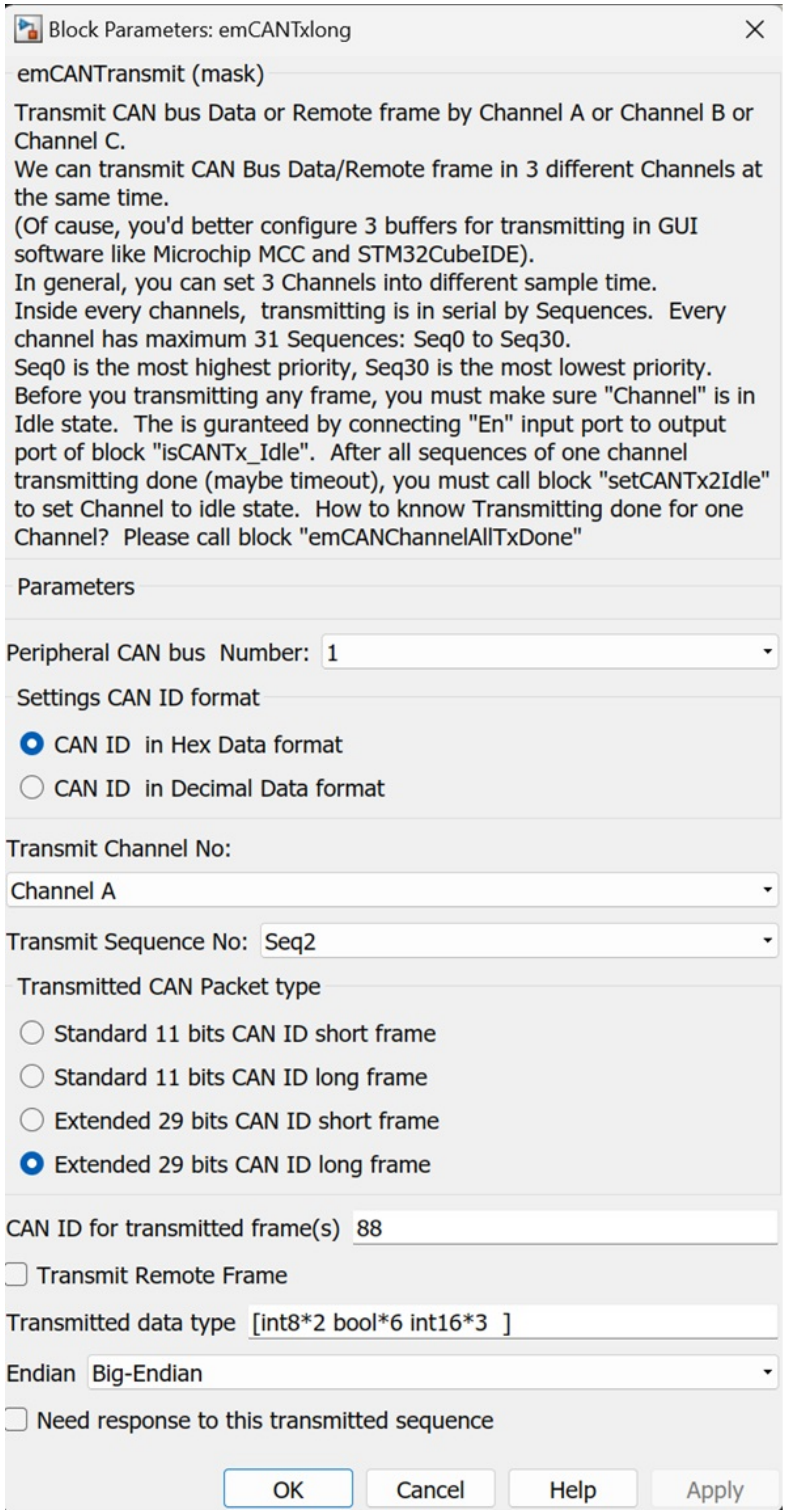
CAN ID	DLC	Data0 (Frame No)	Data1	Data2	Data3	Data4	Data5	Data6	Data7
0x12340112X	8	1	D0	D1	D2	D3	D4	D5	D6
0x12340112X	5	2	D7	D8	D9	D10			

Please see data field of this response long frame below:



The yellow color is int8 vector data area. The white color is bool vector area (Bit0 is the 1st element of bool vector). The blue color is uint32 vector area.

Double click block "emCANTxlong", we will see the parameter settings below:

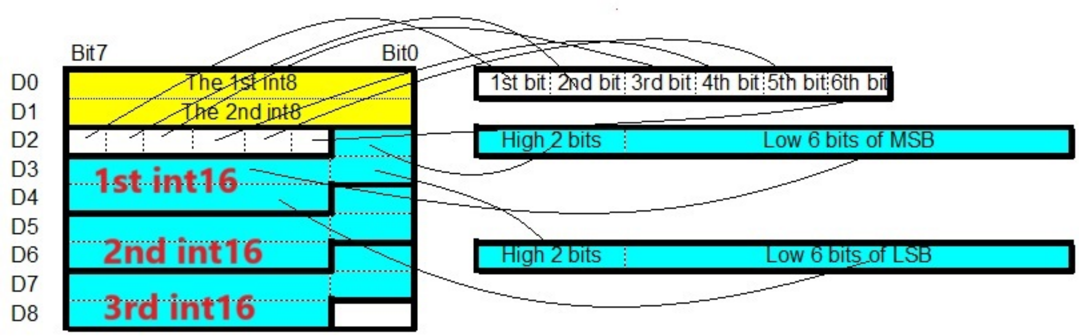


It means that this block will transmit extended long data frame with CAN ID=0x88. Because parameter "Transmitted data type"=[int8*2 bool*6 int16*3]. So DLC=2+1+6=9, and 3 data input ports. The 1st Data input is int8 vectors with 2 elements. The 2nd Data input is bool vectors with 6 elements. The 3rd

Data input is int16 vectors with 3 elements. It is Big-Endian. Please see response CAN bus data long frame structure below:

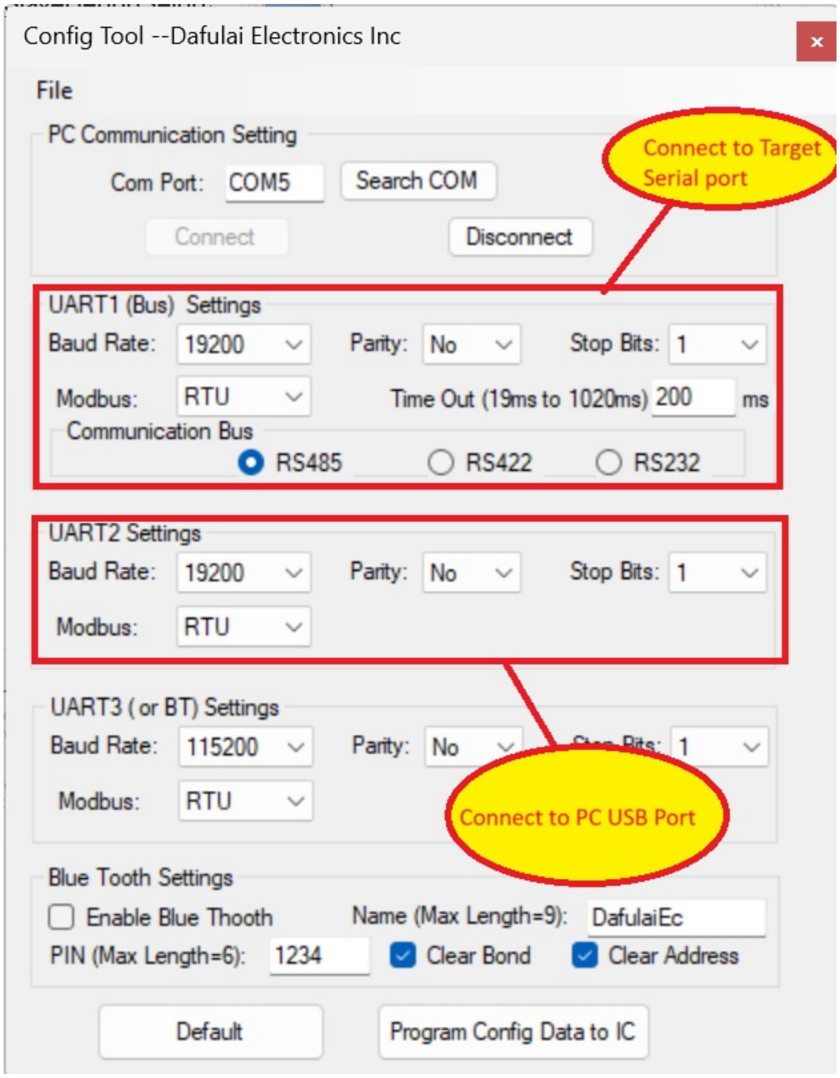
CAN ID	DLC	Data0 (Frame No)	Data1	Data2	Data3	Data4	Data5	Data6	Data7
0x88X	8	1	D0	D1	D2	D3	D4	D5	D6
0x88X	3	2	D7	D8					

Please see data field of this long frame below:



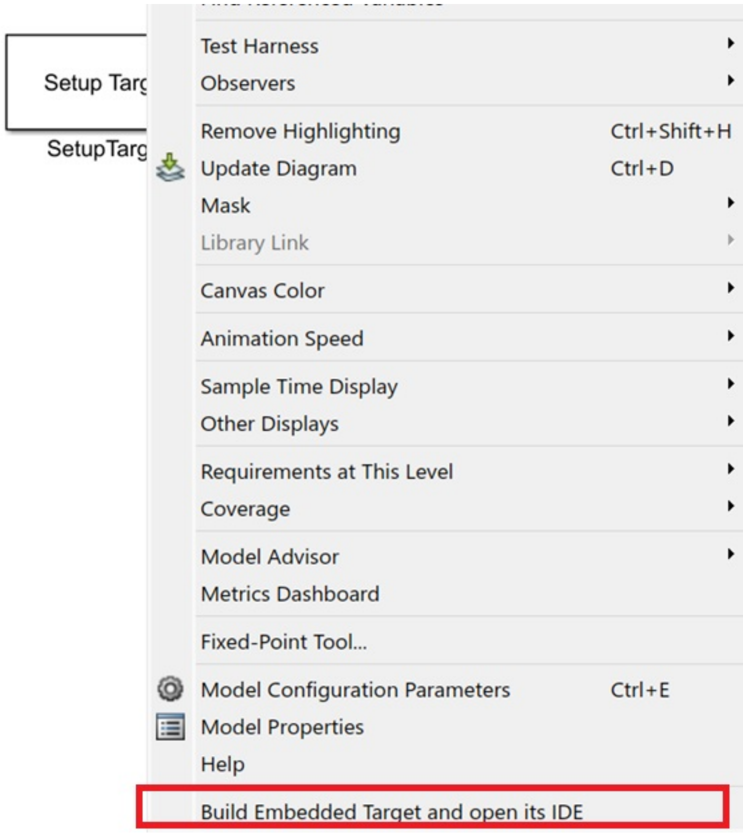
The yellow color is int8 vector data area. The white color is bool vector area (Bit7 is the 1st element of bool vector). The blue color is int16 vector area.

In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:

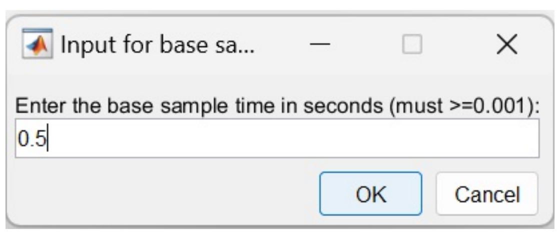


In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

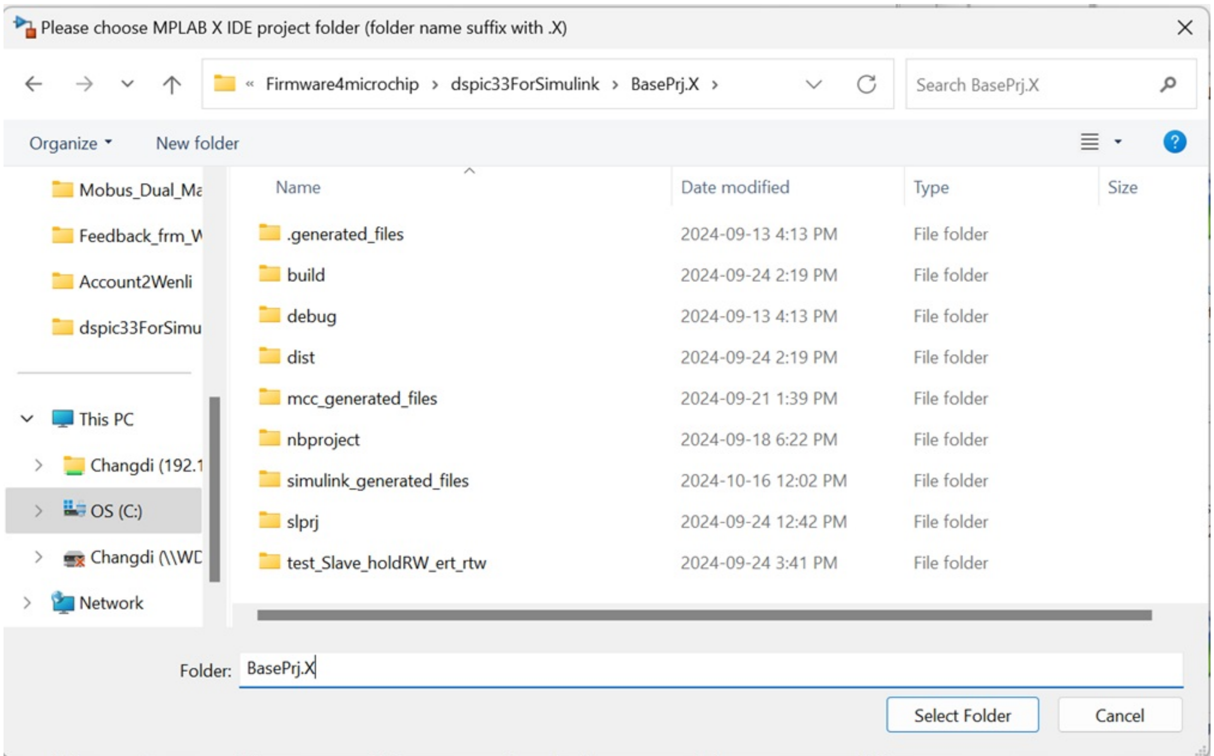
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:



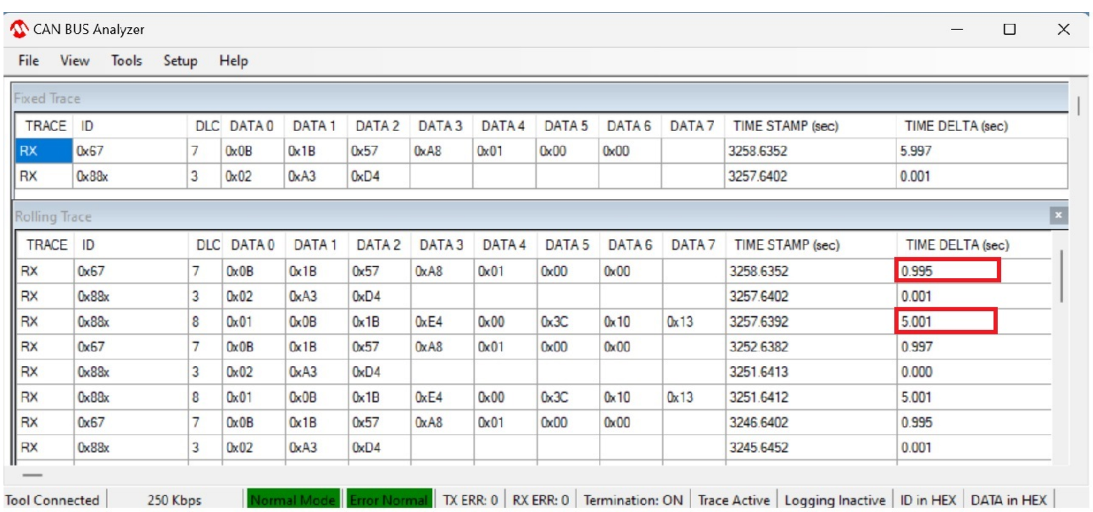
Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

If your firmware program into target successfully, you can use Microchip CAN bus Analyzer or Dafulai Electronic CAN bus Analyzer Hardware/software to transmit CAN BUS frames. Microchip CAN bus Analyzer cannot transmit remote frame, but Dafulai Electronic CAN bus Analyzer can transmit remote frame.

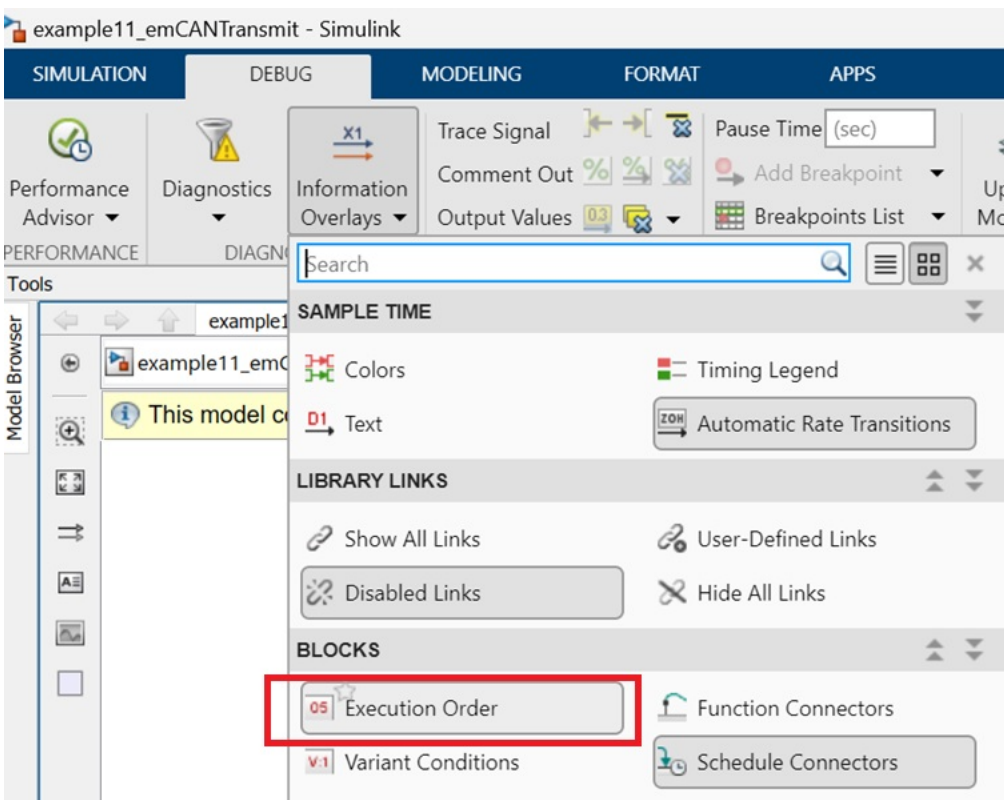
We used Microchip CAN bus Analyzer to test. Run PC Software "CAN BUS Analyzer". Please see screenshot below:



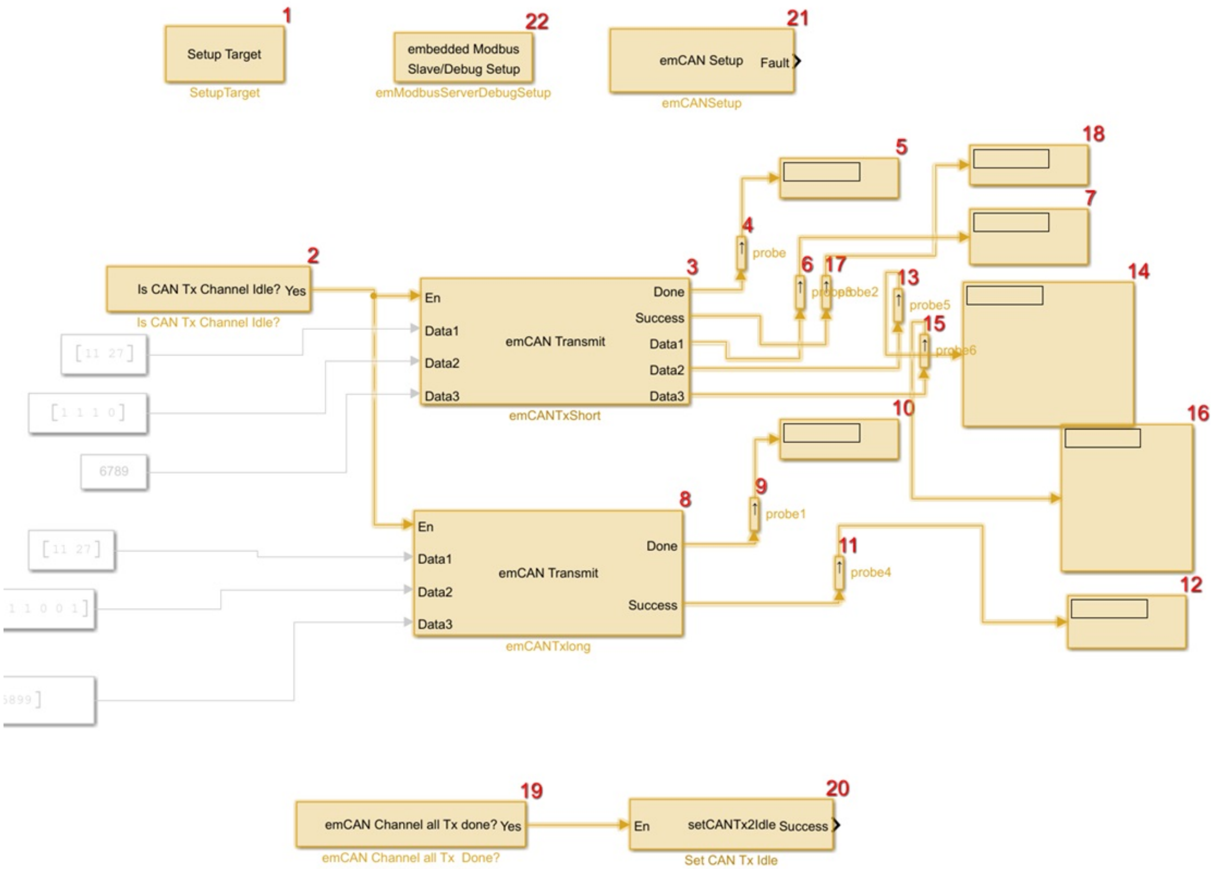
We see the Target transmit CAN ID =0x67 standard frame and CAN ID=0x88 extended long data frames.

From screenshot above, we see our Target transmitted CAN bus frames. We saw CAN ID=0x88x (extended frame) is transmitted once after 5.001 Sec (Red color: around 5 sec) from transmission of CAN ID=0x67. This is correct because CAN ID=0x67 need response and its timeout is 5 sec. We didn't send out response. So timeout 5 Sec. However, we saw the time from CAN ID=0x88x second frame to CAN ID=0x67 is 0.995 sec (Red color: around 1 sec). Our sample time is 0.5 Sec. Why it is 0.5x2= 1 Sec? The reason is block "emCAN Channel all Tx Done?" and "Set CAN Tx Idle" must run before block "Is CAN Tx Channel Idle?".

How to make "emCAN Channel all Tx Done?" and "Set CAN Tx Idle" run before block "Is CAN Tx Channel Idle?" ? Please click "Debug/Information Overlays/Blocks/Execution Order" like screenshot below:

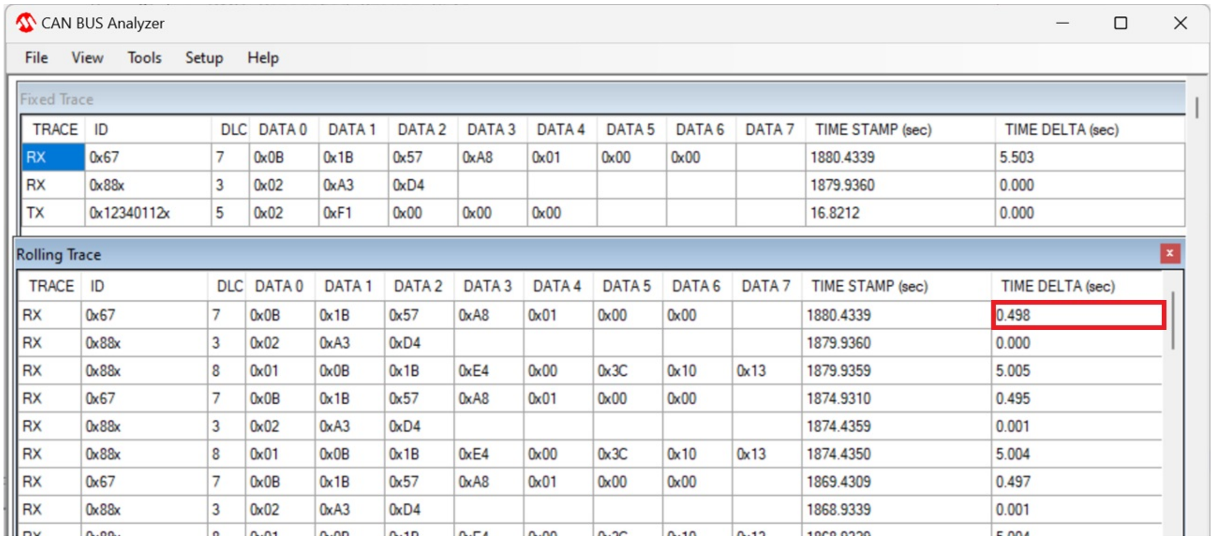


You will see model as below:



Right click on block "Is CAN Tx Channel Idle?" ,"Set CAN Tx Idle", and "emCAN Channel all Tx Done?" to display context menu. Click "Properties..." to make Priority of block "emCAN Channel all Tx Done?" and "Set CAN Tx Idle" smaller than Priority of block "Is CAN Tx Channel Idle?" (Small number has higher priority).

You must re-build model and re-program target. After programming again, we open CAN bus Analyzer again. Please see screenshot below:



CAN BUS Analyzer

File View Tools Setup Help

Fixed Trace

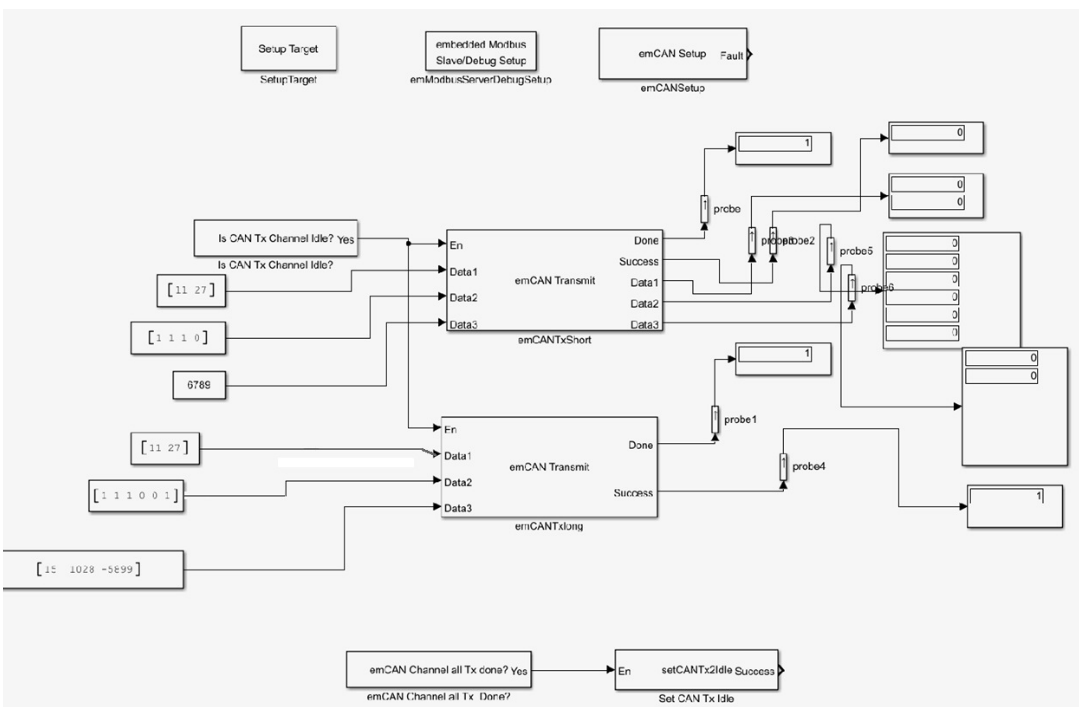
TRACE	ID	DLC	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	TIME STAMP (sec)	TIME DELTA (sec)
RX	0x67	7	0x0B	0x1B	0x57	0xA8	0x01	0x00	0x00		1880.4339	5.503
RX	0x88x	3	0x02	0xA3	0xD4						1879.9360	0.000
TX	0x12340112x	5	0x02	0xF1	0x00	0x00	0x00				16.8212	0.000

Rolling Trace

TRACE	ID	DLC	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	TIME STAMP (sec)	TIME DELTA (sec)
RX	0x67	7	0x0B	0x1B	0x57	0xA8	0x01	0x00	0x00		1880.4339	0.498
RX	0x88x	3	0x02	0xA3	0xD4						1879.9360	0.000
RX	0x88x	8	0x01	0x0B	0x1B	0xE4	0x00	0x3C	0x10	0x13	1879.9359	5.005
RX	0x67	7	0x0B	0x1B	0x57	0xA8	0x01	0x00	0x00		1874.9310	0.495
RX	0x88x	3	0x02	0xA3	0xD4						1874.4359	0.001
RX	0x88x	8	0x01	0x0B	0x1B	0xE4	0x00	0x3C	0x10	0x13	1874.4350	5.004
RX	0x67	7	0x0B	0x1B	0x57	0xA8	0x01	0x00	0x00		1869.4309	0.497
RX	0x88x	3	0x02	0xA3	0xD4						1868.9339	0.001

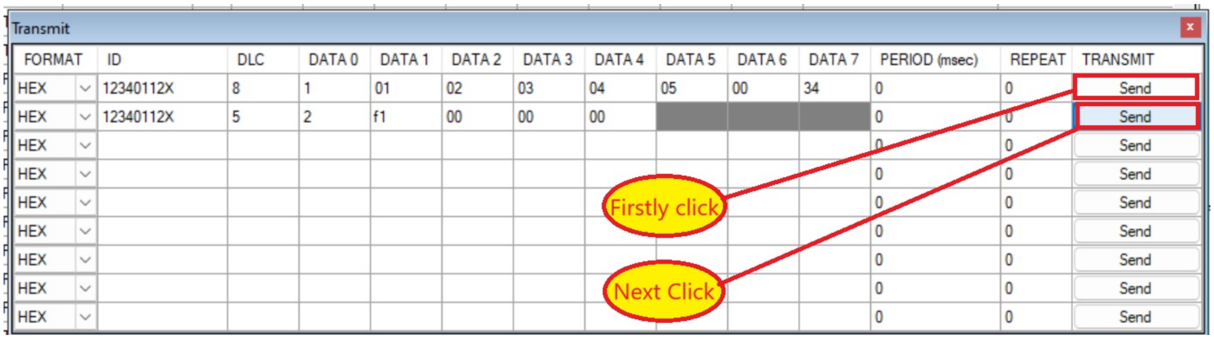
You will see red color 0.498 second (around 0.5 Sec). That's what we expect.

We run Simulink in PC, the result is shown below:

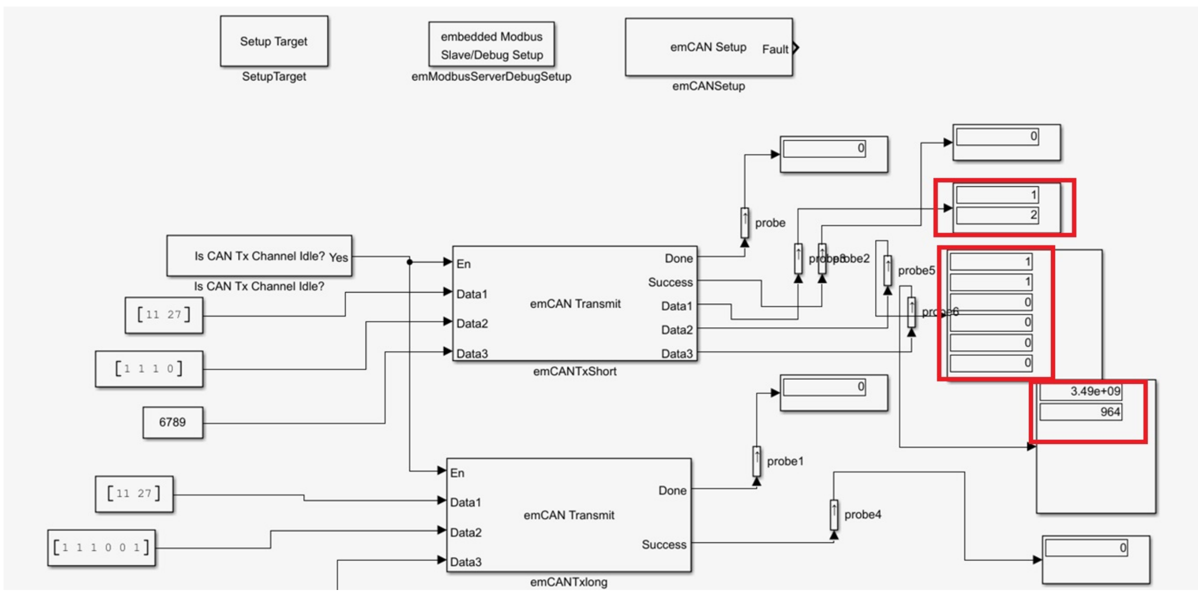


We saw the block all data output ports of "emCANTxShort" are 0. This is because we didn't send out response frames from CAN bus Analyzer.

We transmit 2 CAN bus data frames by CAN Bus analyzer as below:



Now we can see Simulink running result below:



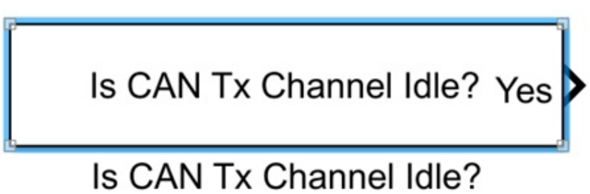
Now we see result in red color. It is data we respond to.

Please open "Your embedded creator library folder"/examples/example11_emCANTransmit.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

isCANTx_Idle

Is CAN bus transmission Channel in Idle?
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded CAN Bus / isCANTx_Idle



Description

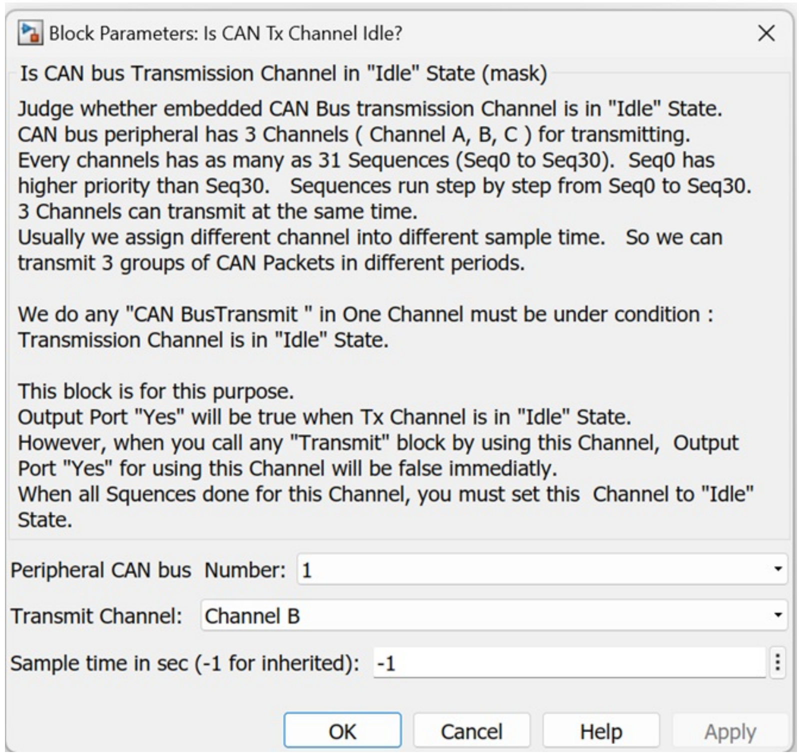
Judge whether embedded CAN Bus transmission Channel is in "Idle" State.
One CAN bus peripheral has 3 Channels (Channel A, B, C) for transmitting.
Every channels has as many as 31 Sequences (Seq0 to Seq30). Seq0 has higher priority than Seq30. Sequences run step by step from Seq0 to Seq30.
3 Channels can transmit at the same time.
Usually we assign different channel into different sample time. So we can transmit 3 groups of CAN Packets in different periods.

We do any "CAN BusTransmit " in One Channel must be under condition : Transmission Channel is in "Idle" State.
When embedded CAN bus transmission channel is in "Idle" state, You can set up all sequences of transmission operations, and then this transmission channel will exit "Idle" state and operate in sequences automatically. We provide block to check if all sequences finish. In order to start new transmission operations, you must set transmission channel into "Idle" state when all sequences done.

> ***Notes:*** *You 'd better set up 3 CAN bus transmitting buffers if you use 3 transmission Channels in MCU Configuration software such as MPLABX IDE MCC for Microchip technology MCUs. Of cause, if you only use one or two Channels, you can setup only one or two CAN bus transmitting buffers.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral CAN bus Number — tell system this block is which CAN controller peripheral used. You just choose from drop list items: 1 or 2.
- Transmit Channel: — drop list to select transmission Channel A or Channel B or Channel C.
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

- Yes — "logical" data type's scalar. True means "CAN bus transmission Channel is in Idle state, you can start any transmission operation by this channel". False means "CAN bus transmission Channel is in busy state, you can not start any transmission operation by this channel"

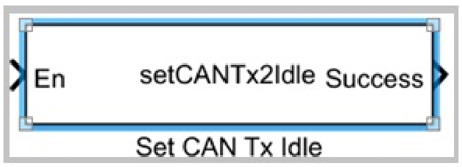
Examples

Please see "emCANTransmit" block example.

setCANTx2Idle

set CAN bus transmission Channel into Idle
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded CAN Bus / setCANTx2Idle

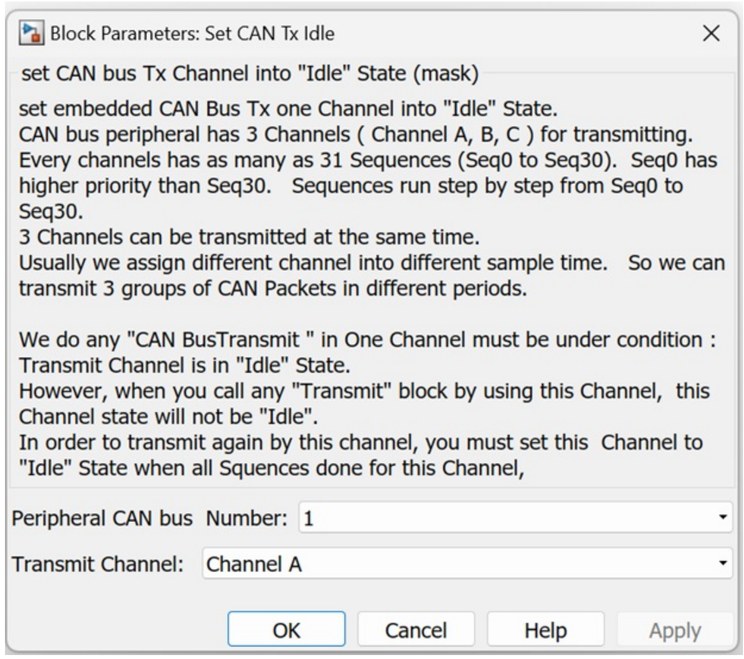


Description

This block sets up CAN bus transmission channel to "Idle" State. You must call it under condition: CAN bus all sequences operations done for this transmission channel. Otherwise, you will get unexpected result. Only when CAN bus transmission channel is in "Idle" state, can you start new operations for this transmission channel

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral CAN bus Number — tell system this block is which CAN controller peripheral used. You just choose from drop list items: 1 or 2.
- Transmit Channel: — drop list to select transmission Channel A or Channel B or Channel C.

Ports

Input

- En — "logical" data type's scalar. True means " it sets one CAN Bus transmission channel to idle state". False means nothing happens.

Outport

- Success — "logical" data type's scalar. It is equal to "En".

Examples

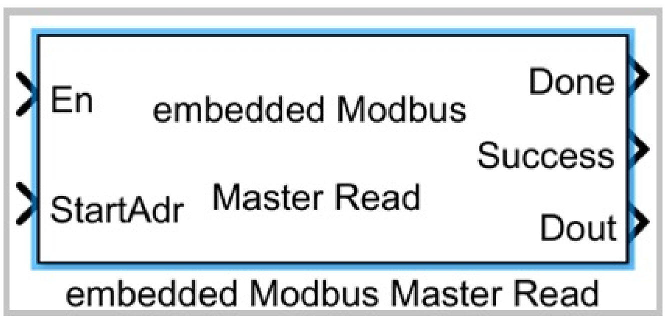
Please see "emCANTransmit" block example.

The following blocks are in "Embedded Modbus Master" directory of library. They are all related to Modbus Master operation. We list them according to alphabetical order.
Embedded Modbus Master operation is one by one in Sequence. You should set up all sequences operation when Modbus master is in idle. And you should set up Modbus master to Idle when it completes all sequences operation.

emModbusMasterRead

embedded Modbus Master reading registers
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded Modbus Master / emModbusMasterRead

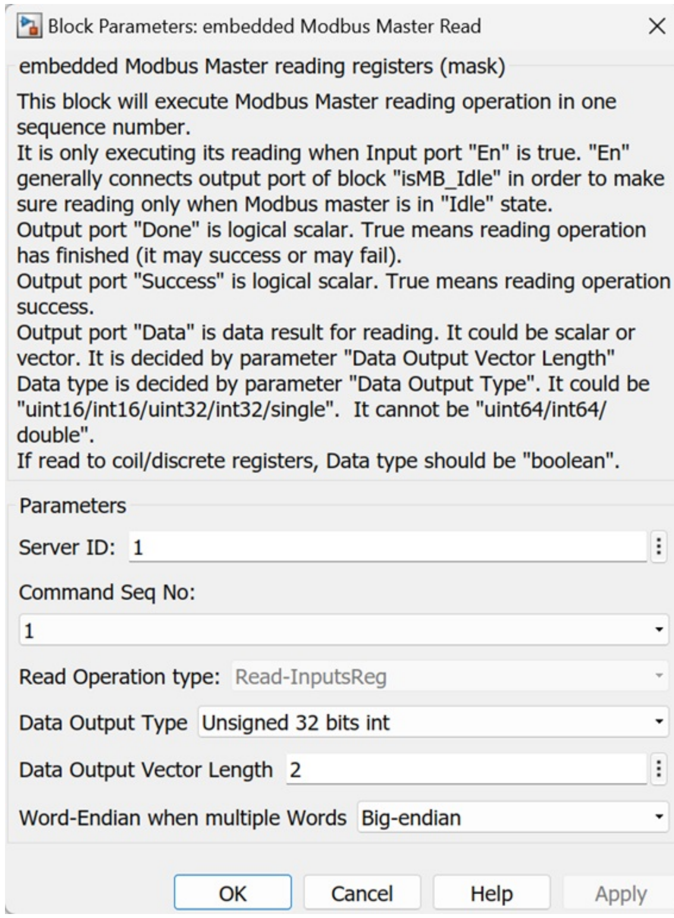


Description

This block sends reading register command to a modbus server. You can only send command when modbus master "idle". So input port "En" must connect block "isMB_Idle" output port.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Server ID — Scalar. It is Modbus server ID or called Server address. Modbus master will read to device with this Server ID. Value range is 1 to 247.

- Command Seq No — drop list from 1 to 16 which is sequence Number in tab "Read Command Seqs" of block "emModbusMasterSetup". The big sequence number has low writing priority. All reading commands have lower priority than writing commands.

- Read Operation type — It is selected automatically from block "emModbusMasterSetup" according to "Command Seq No". If you want to change it, please modify in block "emModbusMasterSetup". It can be one of "Read-InputsReg"/"Read-Holdings"/"Read-Discretes"/"Read-Coils".

- Data Output Type — drop list for output port data type. It can be one of "Logical boolean"/"Unsigned 16 bits int"/"Signed 16 bits int"/"Unsigned 32 bits int"/"Signed 32 bits int "/"32 bits float"

- Data Output Vector Length — Output port "Data" vector length.

- Word-Endian when multiple Words — drop list from "Big-endian" and "Little-endian". When read data type is dual words, it tell us word's endian.

Ports

Input

- En — "logical" data type's scalar. Only when it is true will this block executes reading function. So "En" generally connects output port of block "isMB_Idle" in order to make sure reading only when Modbus master is in "Idle" state.

- StartAddr — "uint16" data type's scalar. It is Modbus Server registers' start address (1-based without prefix "4X","3X", "1X", "0X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.

Outport

- Done — "logical" data type's scalar. It denotes whether communication between PC and Target has finished. "Done" does not mean success. It may success or fail.

- Success — "logical" data type's scalar. It denotes whether reading success.

- Data — Scalar or Vector. Data type is set in parameter "Data Output Type" . Vector length is set in parameter "Data Output Vector Length"

Examples

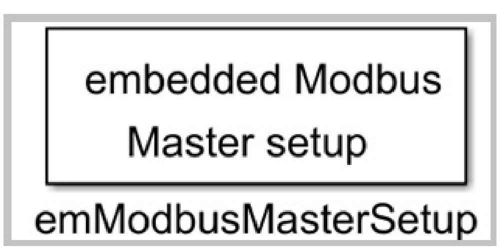
Example1/2/3:

Please see example in block "emModbusMasterWrite".

emModbusMasterSetup

set up embedded Modbus Master
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded Modbus Master / emModbusMasterSetup



Description

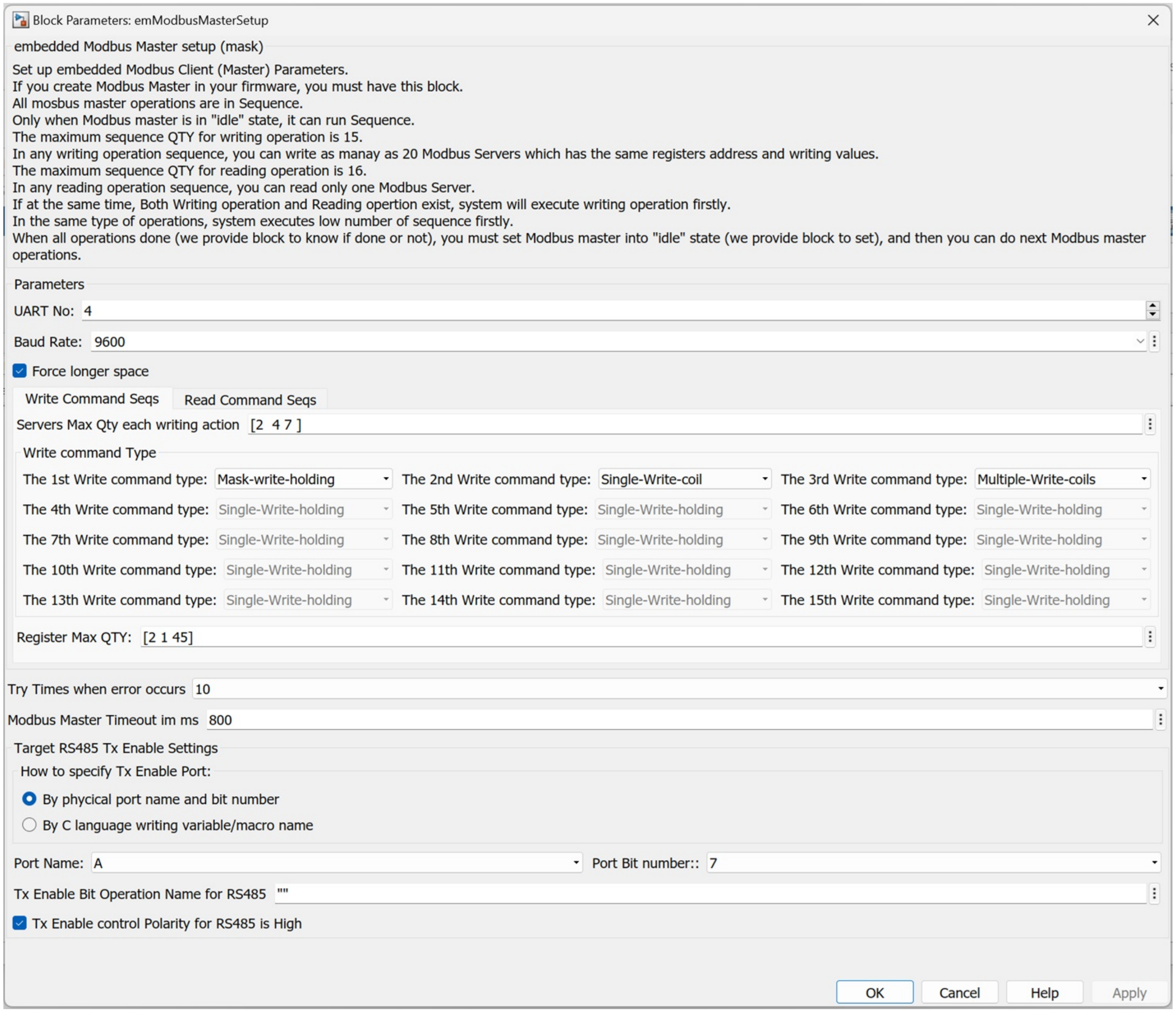
This block sets up Modbus master parameters. Every embedded target which uses Modbus master must have only-one of this block. When master is in "Idle" state, You can set up all sequences of modbus master operations (Reading blocks or Writing blocks), and then Master will exit "Idle" state and operate in sequences. We provide block to check if all sequences finish. In order to start new group of modbus operations, you must set Modbus master into "Idle" state when all sequences done.

In general, you can assign one device's operation (this device is modbus server) in one sequence. So in modbus master "idle", you can assign all devices operations (in different sequences). For device itself sequence operations, for example, after 2 secs when all devices operation done, we send other different operations to all devices. You just use "finite state machine" to do this controls.

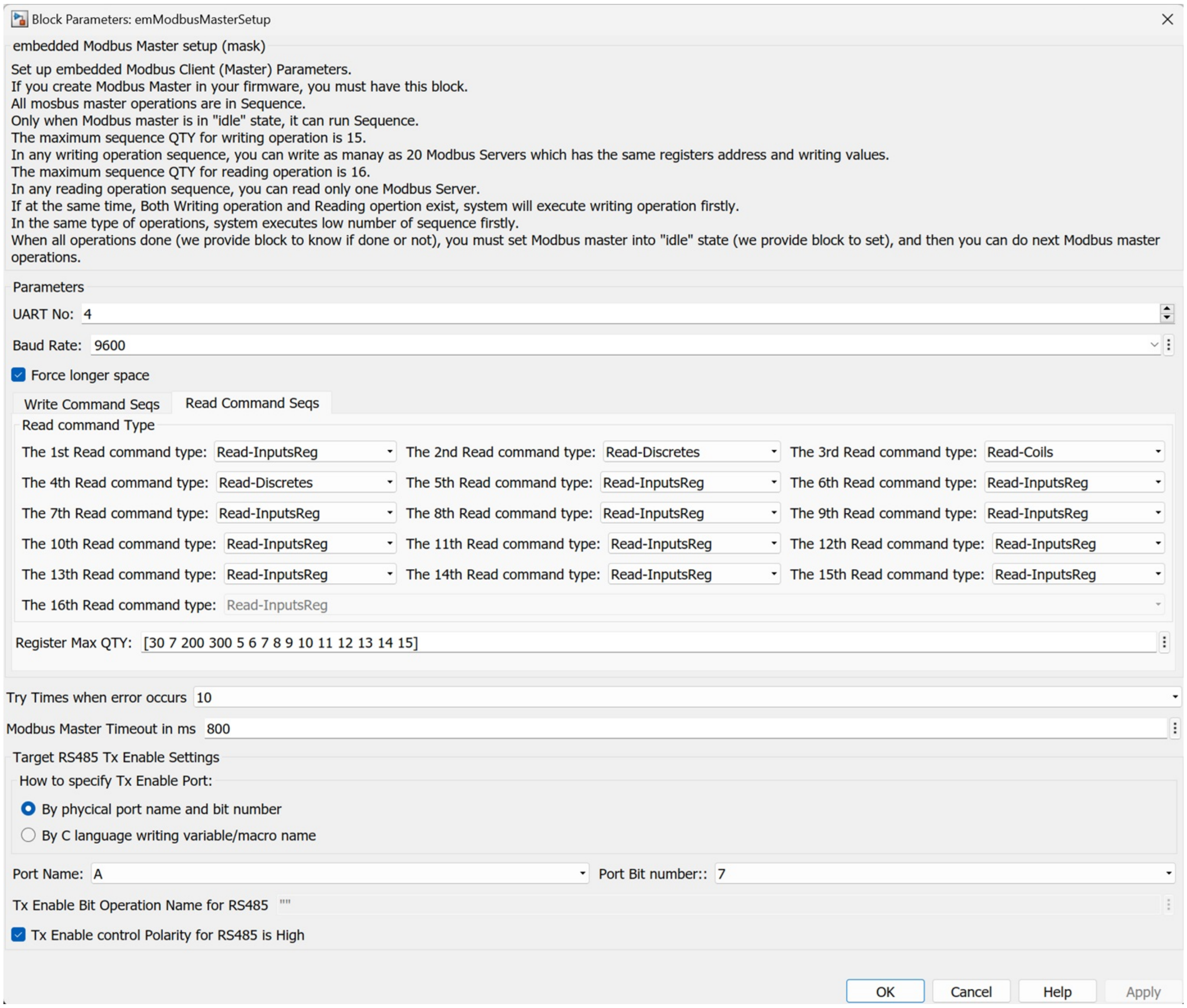
> ***Notes****: You can add only - one embedded Modbus Master or Client. You cannot create 2 or 2+ Modbus Masters or Clients.*

Parameters

Please double click this block to open parameters dialog below:



When you click on tab "Read Command Seqs", it will display "Read command Types" instead of "Write command Types" :



Let us explain parameters.

- UART No: — In one embedded Micro-controller, there are many serial ports. This drop list parameter will tell target which serial port of Micro-controller will be used for this embedded modbus master.

- Baud Rate — embedded serial port baud rate in unit bit/sec. Actually, actual baud rate is decided by embedded target IDE configuration software. Here just use it to decide time for Modbus RTU packet separation (3.5 characters)

- Force longer space — true will make Modbus Master request packet to be separated with server response packet much longer time (in standard Modbus RTU Protocol, it is 3.5 characters time). It will increase anti-noise ability. false will use standard 3.5 characters space time.

- Servers Max Qty each writing action — Scalar or vector. The length of Vector denotes how many sequences for writing operation. Max length is 15. Vector's element value denotes how many servers for this writing operations. For example, in figure above, vector=[2 4 7], it tells us total 3 sequences for writing operation. The 1st writing sequence has 2 Modbus servers. The 2nd writing sequence has 4 Modbus servers. The 3rd writing sequence has 7 Modbus servers.

- The 1st Write command type — This is the 1st writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 2nd Write command type — This is the 2nd writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 3rd Write command type — This is the 3rd writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding"
- The 4th Write command type — This is the 4th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 5th Write command type — This is the 5th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 6th Write command type — This is the 6th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 7th Write command type — This is the 7th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 8th Write command type — This is the 8th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 9th Write command type — This is the 9th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 10th Write command type — This is the 10th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 11th Write command type — This is the 11th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 12th Write command type — This is the 12th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- The 13th Write command type — This is the 13th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".

- The 14th Write command type — This is the 14th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".

- The 15th Write command type — This is the 15th writing operation type, it has 5 options from drop list: "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".

- Register Max QTY (inside tab "Write command Seqs") — Scalar or vector. The length of Vector denotes how many sequences for writing operation. Max length is 15. So it must be equal to the length of parameter vector "Servers Max Qty each writing action". Vector's element value denotes Maximum Registers QTY for this writing operations, For example, in figure above, vector=[2 1 45], it tells us total 3 sequences for writing operation. The 1st writing sequence has Maximum 2 registers involved. The 2nd writing sequence has Maximum 1 register involved. The 3rd writing sequence has Maximum 45 registers involved.

- The 1st Read command type — This is the 1st reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 2nd Read command type — This is the 2nd reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 3rd Read command type — This is the 3rd reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 4th Read command type — This is the 4th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 5th Read command type — This is the 5th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 6th Read command type — This is the 6th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 7th Read command type — This is the 7th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 8th Read command type — This is the 8th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 9th Read command type — This is the 9th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 10th Read command type — This is the 10th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 11th Read command type — This is the 11th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 12th Read command type — This is the 12th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".

- The 13th Read command type — This is the 13th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".
- The 14th Read command type — This is the 14th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".
- The 15th Read command type — This is the 15th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".
- The 16th Read command type — This is the 16th reading operation type, it has 4 options from drop list: "Read-InputsReg"/ "Read-Holdings" / "Read-Discretes" /"Read-Coils".
- Register Max QTY (inside tab "Read command Seqs") — Scalar or vector. The length of Vector denotes how many sequences for reading operation. Max length is 16. Vector's element value denotes Maximum Registers QTY for this reading operations, For example, in figure above, vector=[30 7 200 300 5 6 7 8 9 10 11 12 13 14 15], it tells us total 15 sequences for reading operation. The 1st reading sequence has Maximum 30 registers involved. The 2nd reading sequence has Maximum 7 register involved. The 3rd reading sequence has Maximum 200 registers involved, ... , The 15th reading sequence has Maximum 15 registers involved.
- Try Times when error occurs — Scalar. When Modbus Master operation has communication error, Modbus Master can try how many times before it reports communication error.
- Modbus Master Timeout in ms — Scalar. When Modbus Master sends request to Modbus server, this is maximum time Modbus master can wait for Modbus server to response.
- How to specify Tx Enable Port — In RS485/422 interface, generally, you need one GPIO output to control Tx Enable. You can choose one of 2 ways to specify this GPIO outport. "By physical port name and bit number" will specify which GPIO Port Name (drop list) and which Port bit number (drop list) used for "Tx Enable". "By C language writing variable/macro name" will directly specify C language's GPIO output variable or macro for "Tx Enable". If your hardware don't use "Tx Enable", you just give this parameter empty string "".
- Port Name — It is for "Tx Enable". When you use "By physical port name and bit number", it specify GPIO Port Name (drop list).
- Port Bit number — It is for "Tx Enable". When you use "By physical port name and bit number", it specify which Port bit number (drop list).
- Target RS485 Tx Enable C language operation Name — It is for "Tx Enable". When you use "By C language writing variable/macro name", it will directly specify C language's GPIO output variable or macro for "Tx Enable". If your hardware don't use "Tx Enable", you just give this parameter empty string "".
- Target RS485 Tx Enable polarity is High — It is for "Tx Enable". True means GPIO port Logic High will enable "RS485/RS422" transmit. False means GPIO port Logic Low will

enable "RS485/RS422" transmit.

Ports

Input

None

Outport

None

Examples

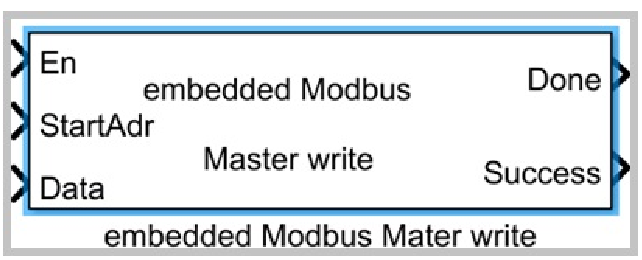
Example1/2/3:

Please see example in block "emModbusMasterWrite".

emModbusMasterWrite

embedded Modbus Master writing registers
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded Modbus Master / emModbusMasterWrite

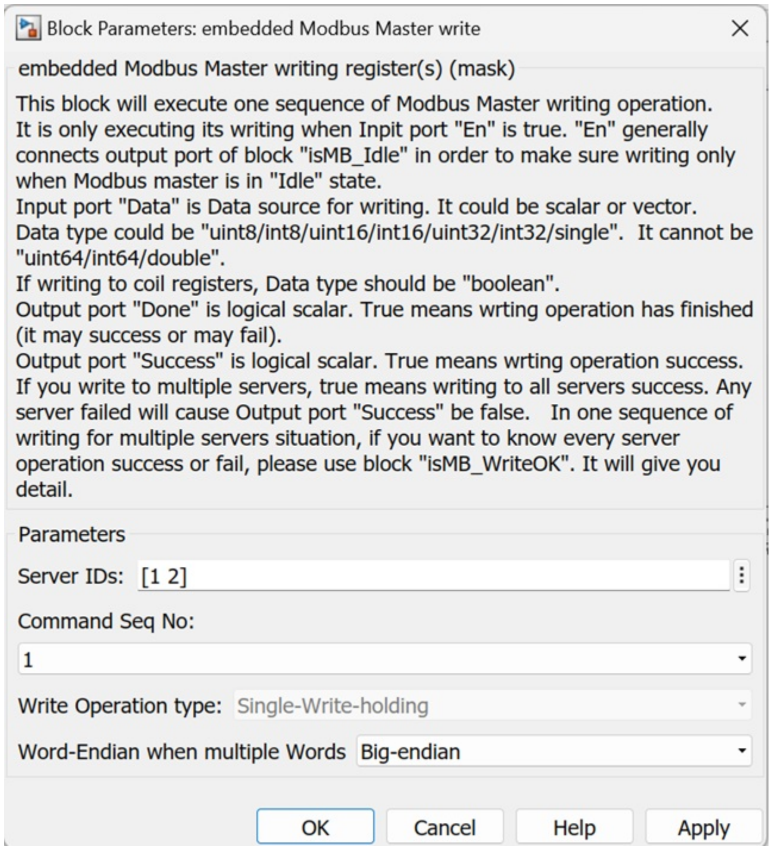


Description

This block sends writing register command to a modbus server or a group of modbus servers. You can only send command when modbus master "idle". So input port "En" must connect block "isMB_Idle" output port.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Server IDs — Scalar or Vector. It is Modbus server ID or called Server address. Modbus master will write to devices with these Server IDs. Value range is 0 to 247. 0 means "broadcast" writing (write to all nodes in the Modbus network)
- Command Seq No — drop list from 1 to 15 which is sequence Number in tab "Write Command Seqs" of block "emModbusMasterSetup". The big sequence number has low writing priority. All writing commands have higher priority than reading commands.
- Write Operation type — It is selected automatically from block "emModbusMasterSetup" according to "Command Seq No". If you want to change it, please modify in block "emModbusMasterSetup". It can be one of "Single-Write-holding"/ "Multiple-Write-holdings" / "Single-Write-coil" / "Multiple-Write-coils" / "Mask-write-holding".
- Word-Endian when multiple Words — drop list from "Big-endian" and "Little-endian". When writing data is dual words, it tell us word's endian.

Ports

Input

- En — "logical" data type's scalar. Only when it is true will this block executes writing function. So "En" generally connects output port of block "isMB_Idle" in order to make sure writing only when Modbus master is in "Idle" state.
- StartAddr — "uint16" data type's scalar. It is Modbus Server registers' start address (1-based without prefix "4X","0X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.
- Data — Scalar or Vector. Data type can be "uint8/int8/uint16/int16/uint32/int32/single" when register is holding or "logical" when register is coil. If data type is uint8 or int8, one data will occupy 16 bits of holding register.

Outport

- Done — "logical" data type's scalar. It denotes whether communication between PC and Target has finished. "Done" does not mean success. It may success or fail.
- Success — "logical" data type's scalar. It denotes whether writing success. If you write to multiple servers in one sequence, true means writing to all servers success in this sequence. Any server failed will cause Output port "Success" to be false. In one sequence of writing for multiple servers situation, if you want to know every server operation success or fail, please use block "isMB_WriteOK". It will give you detail.

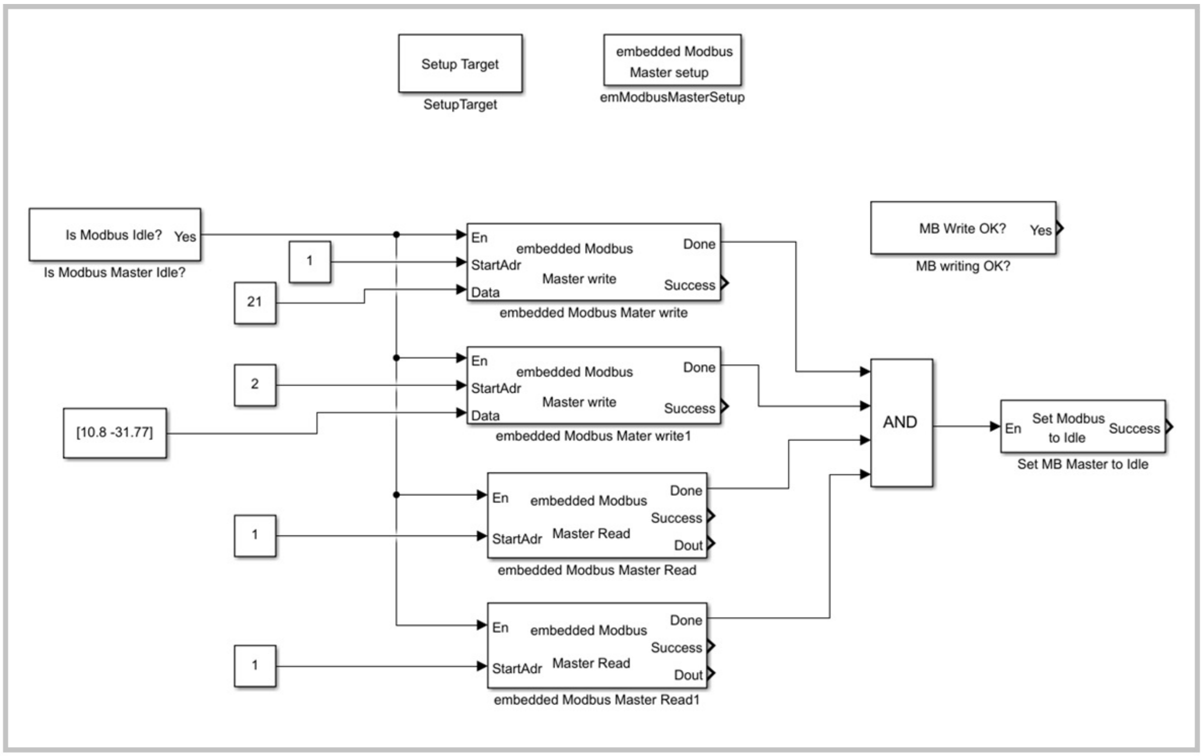
Examples

Example1:

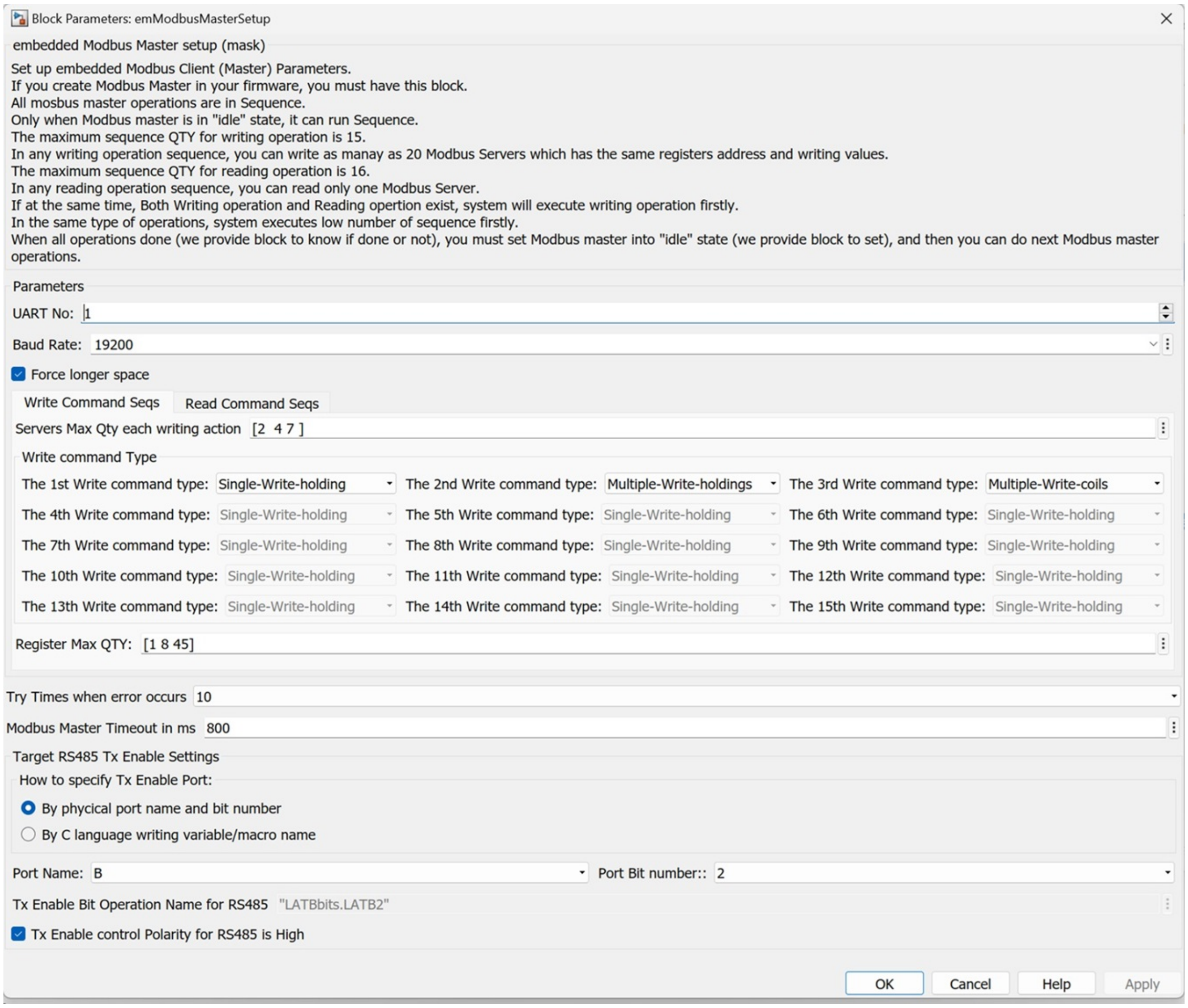
Our embedded platform is dsPIC33EP256GP502 . Uart1 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB2. Logic High will enable RS485 transmit and disable RS485 receiver. Every 0.5 sec (Base sample time), we will do the following operations:

1. Do single writing holding register at address 40001 to value 21 to server 1 (Server ID=1) and Server 2 (Server ID=2).
2. Do multiple writing holding registers at address 40002 to 40005 value 10.8 and -31.77 to server 1 (Server ID=3) and Server 2 (Server ID=4). Word order is big-endian.
3. Do reading input registers at address 30001 to 30004 to server which has server ID=1. Data type is 32 bits of unsigned integer, and big-endian.
4. Do reading discrete registers at address 10001 to 10007 to server which has server ID=2.

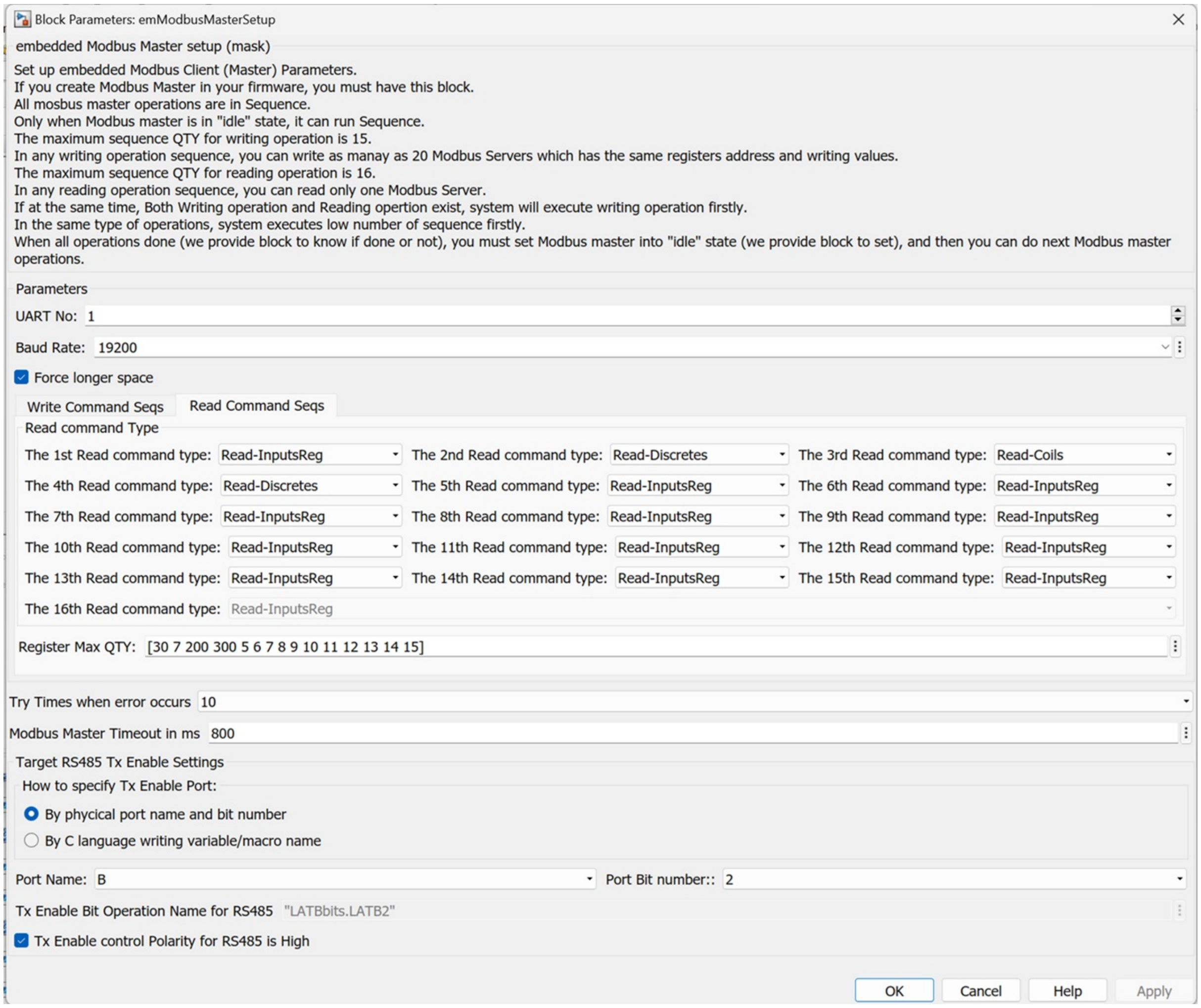
Please see screenshot of model below:



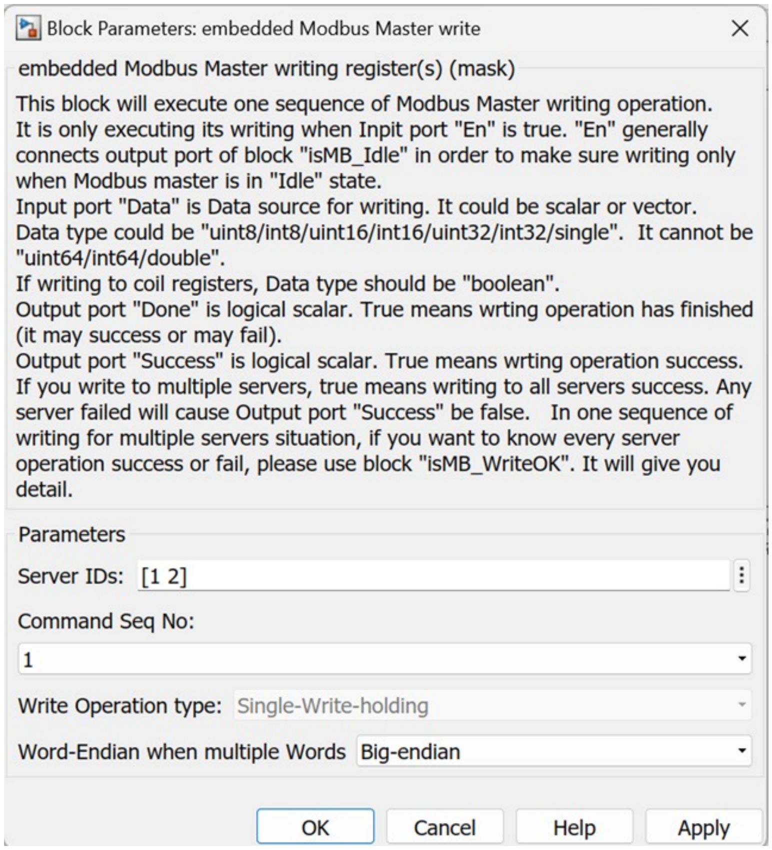
In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Double click "emModbusMasterSetup", we will see the Modbus Master parameters settings below:



If you click on tab "Read Command Seqs", you will see the settings for "Read Command Seqs" below:

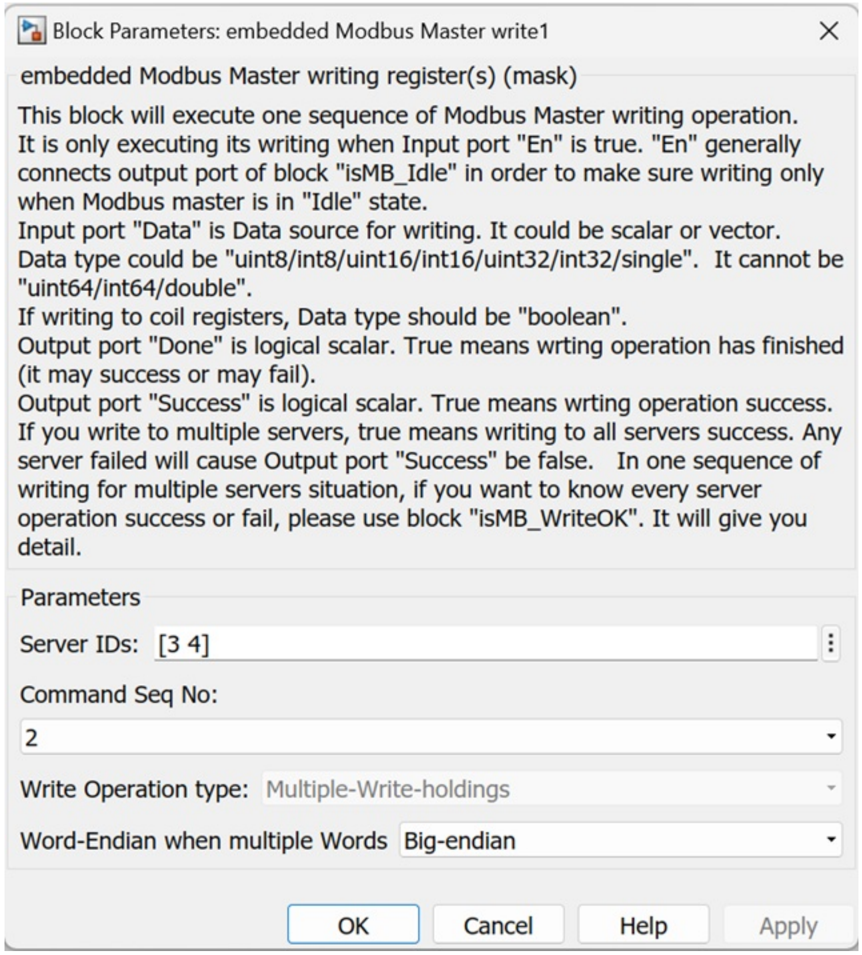


Double click "embedded Modbus Master write", we will see the parameters settings below:



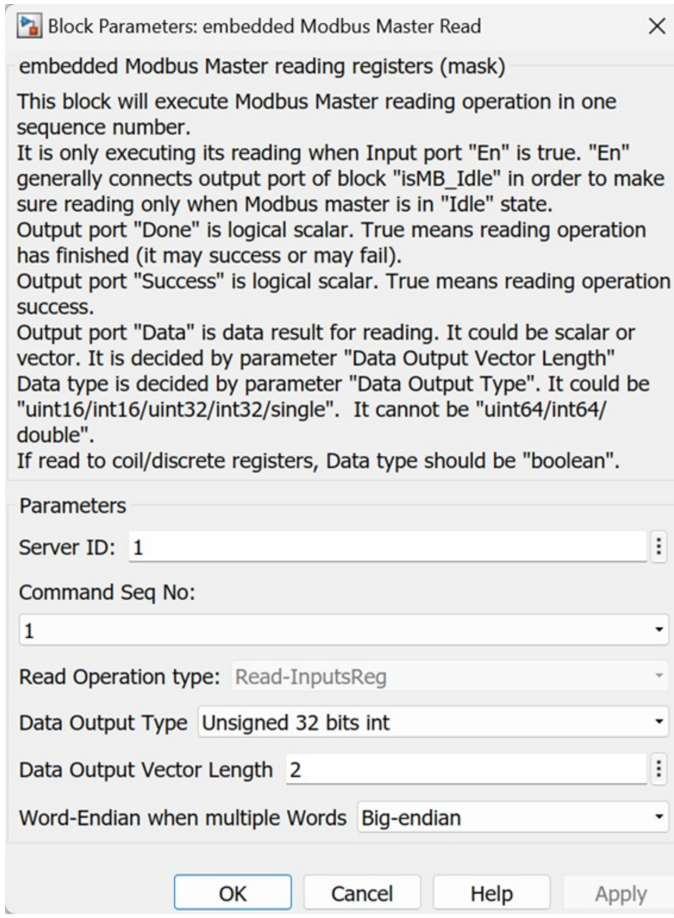
It means we will writing to single holding register and 2 Modbus servers with ID =1 and 2 in command sequence No 1.

Double click "embedded Modbus Master write 1", we will see the parameters settings below::



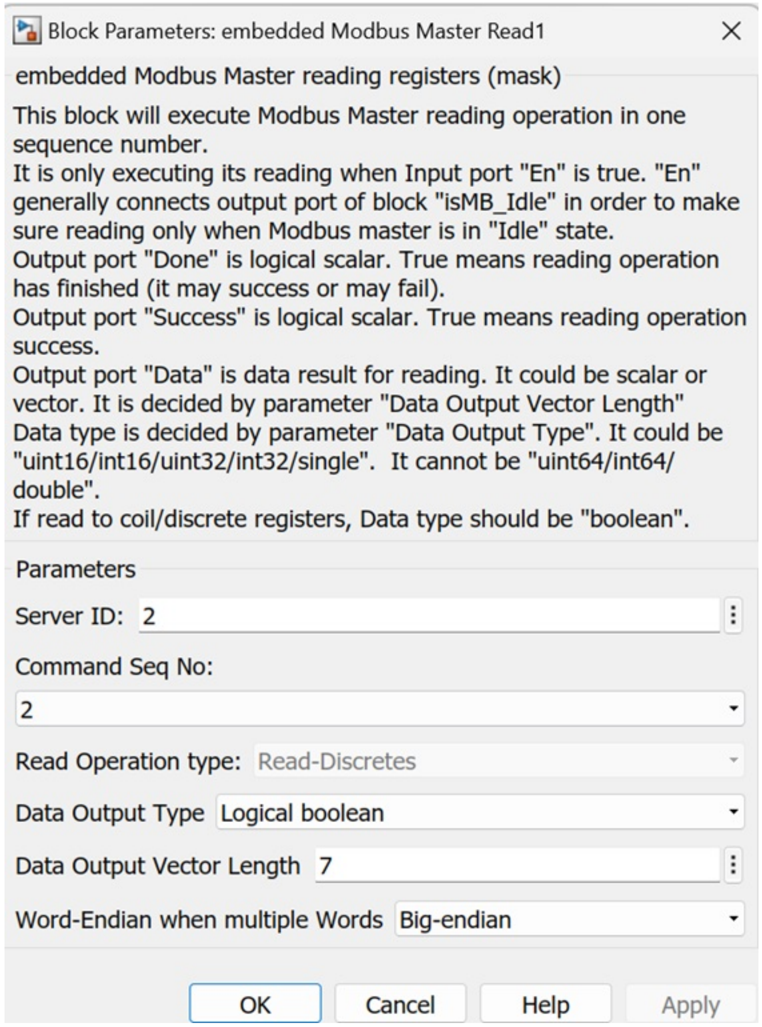
It means we will writing to multiple holding register in command Seq No 2, and 2 Modbus servers with ID =3 and 4. If input port data type is uint32/int32/Single, word-endian is big-endian.

Double click "embedded Modbus Master Read", we will see the parameters settings below:



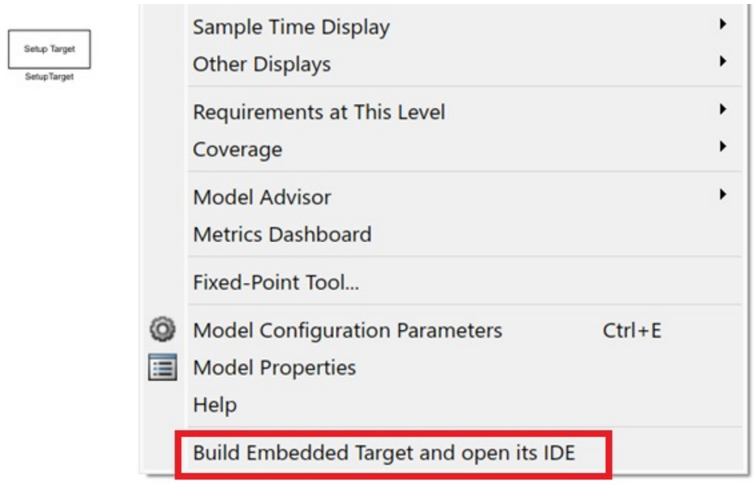
It means we will read input registers in command seq No 1, and Modbus server with ID =1. Output port data type is 32 bits of unsigned integer , word-endian is big-endian. Data length is 2 x 32 bits.

Double click "embedded Modbus Master Read 1", we will see the parameters settings below:

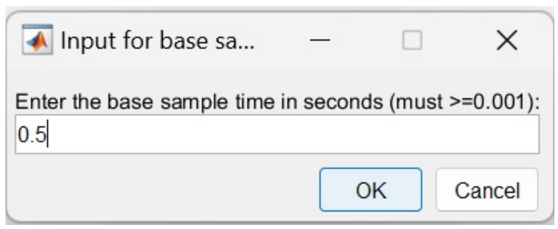


It means we will read discrete registers in command seq No 2, and Modbus server with ID =2. Output port data type is Boolean , Data vector length is 7.

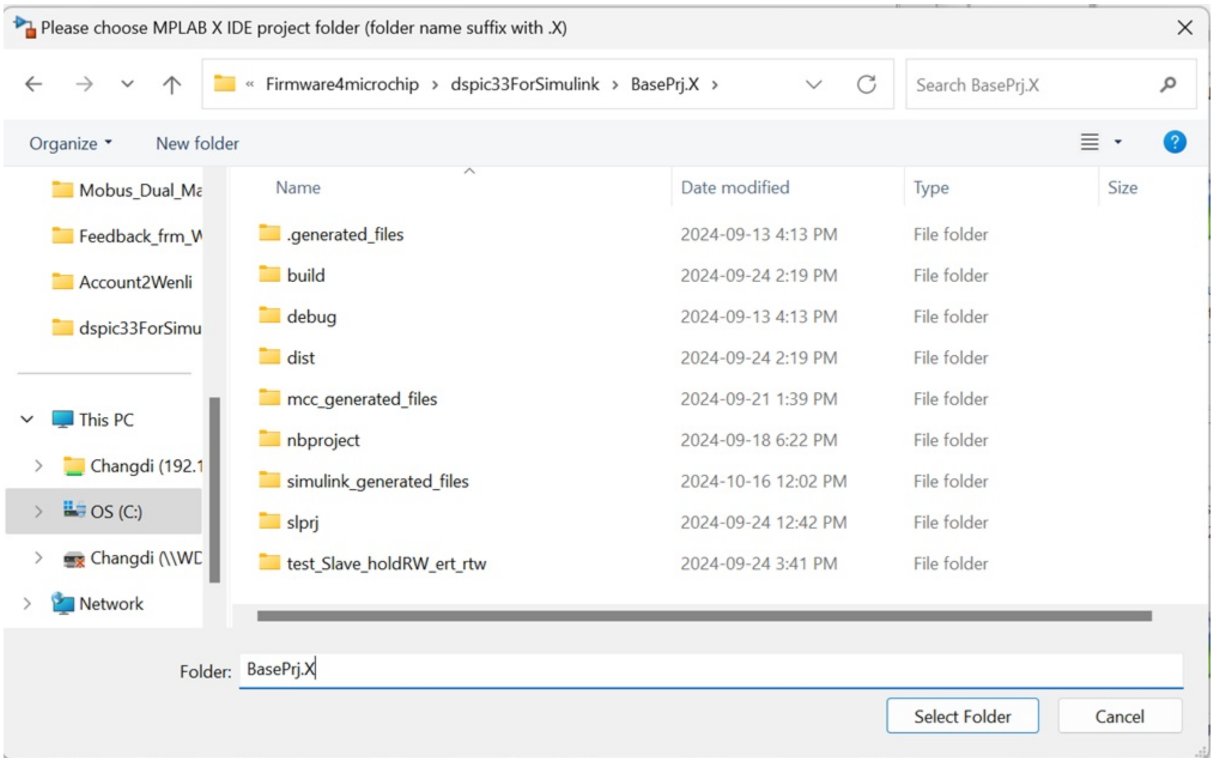
Right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:

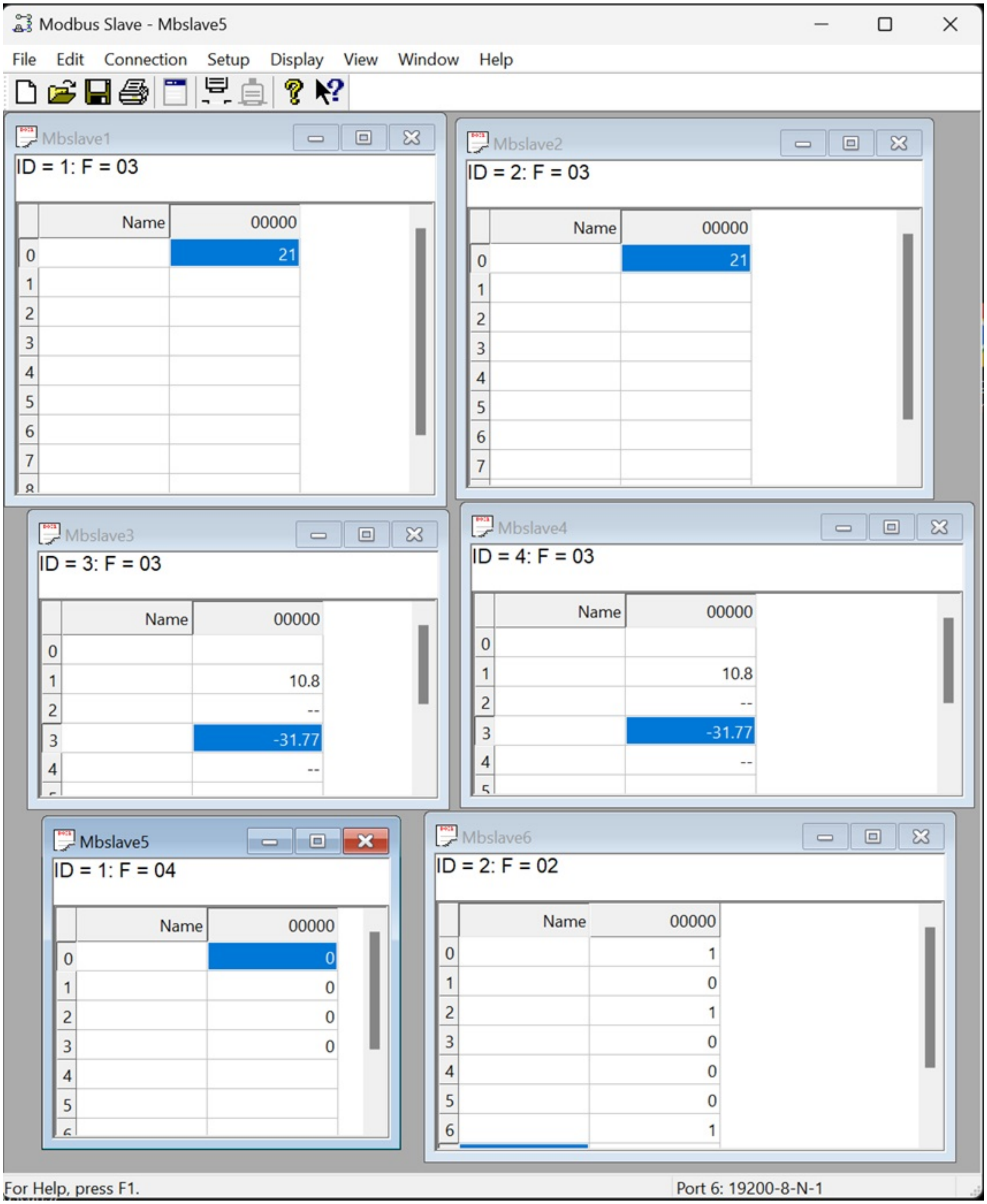


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

If your firmware program into target successfully, Connect your target Uart1 (RS485/RS422) to your PC, you can use PC Modbus slave software: mbslave.exe to view result below:



In above figures, we set up 6 windows from Mbslave1 to Mbslave6.

From Mbslave1 and Mbslave2, we saw slave ID=1 and 2, the holding register value at address 0 (0-based without prefix) or address 40001 (1-based with "4x" prefix) is 21 which matches our model (block "embedded Modbus Master write").

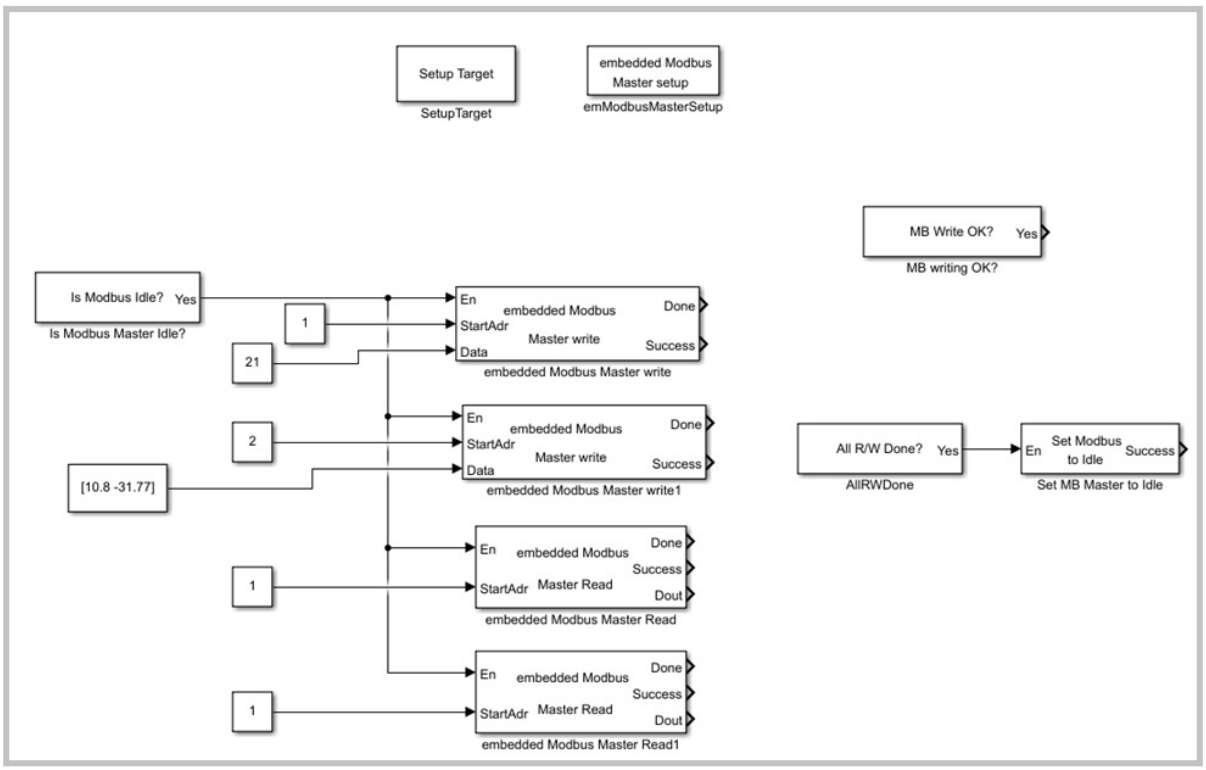
From Mbslave3 and Mbslave4, we saw slave ID=3 and 4, the holding register value at address 0 and 1 (0-based without prefix) or address 40001 and 40002 (1-based with "4x" prefix) is 10.8, the holding register value at address 2 and 3 (0-based without prefix) or address 40003 and 40004 (1-based with "4x" prefix) is -31.77, which match our model (block "embedded Modbus Master write 1").

From Mbslave5 and Mbslave6, we cannot verify the result for Modbus Master reading because we did not put "probe" in simulink block to view reading results. In example 3, we will put "probe".

Please open "Your embedded creator library folder"/examples/example7_emMBMaster1.slx

Example2:

All conditions are the same as example1. But we delete "AND" block, directly use "AllWriteReadDone" block to replace. Please see screenshot of model below:

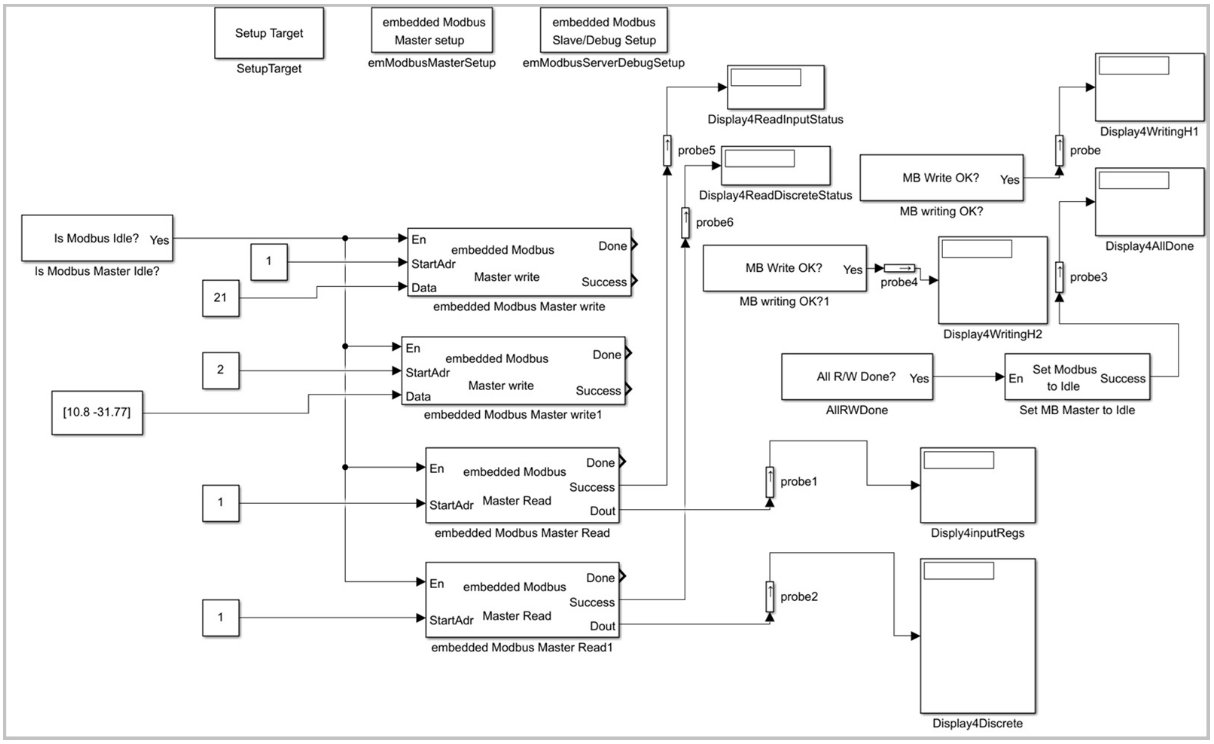


The test results are the same as example1. Similarly, we cannot verify Modbus Master reading result because of no "Probe". In example 3, we will put "probe".

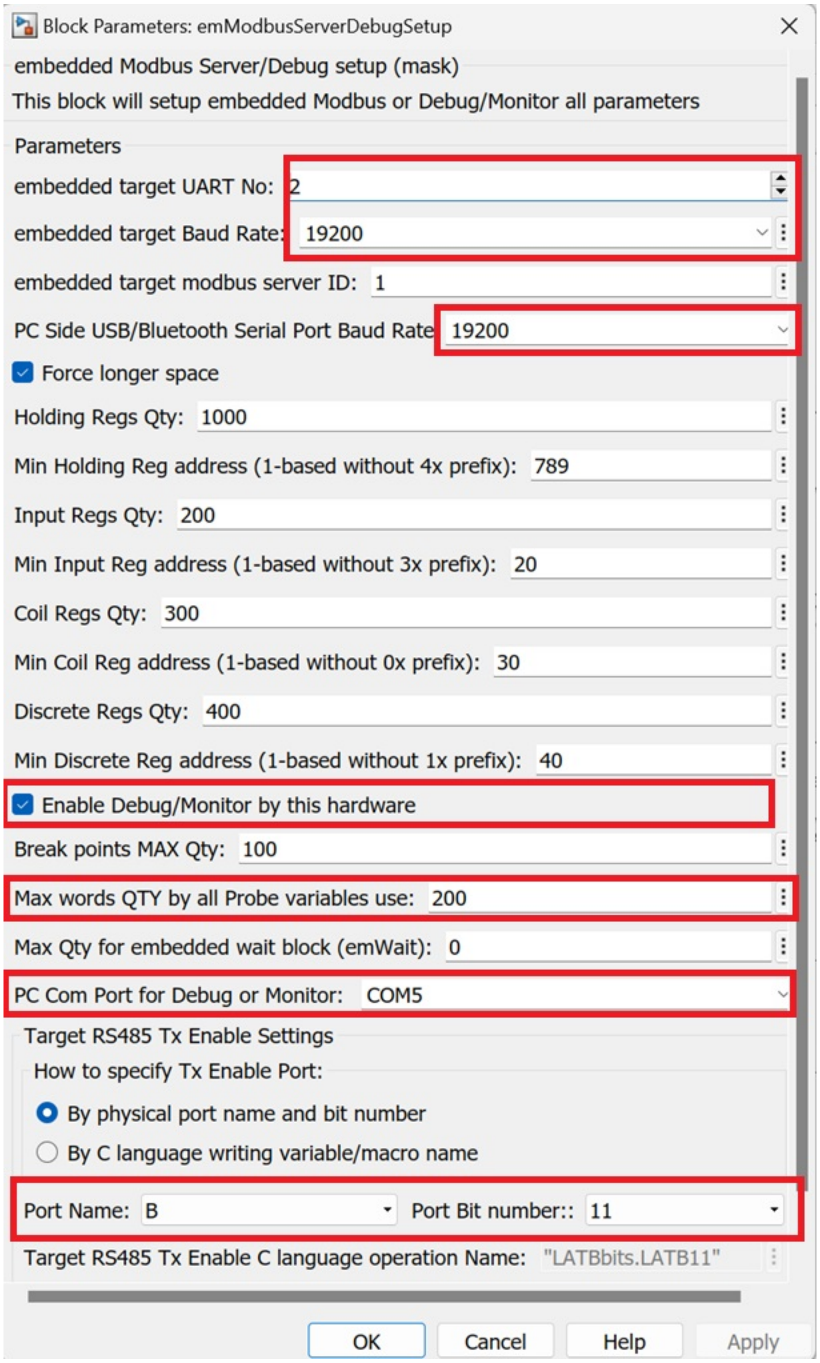
Please open "Your embedded creator library folder"/examples/example7_emMBMaster2.slx

Example3:

All conditions are the same as example2 But we add "embedded Modbus slave/Debug Setup" block and lots of "probe" blocks . Please see screenshot of model below:

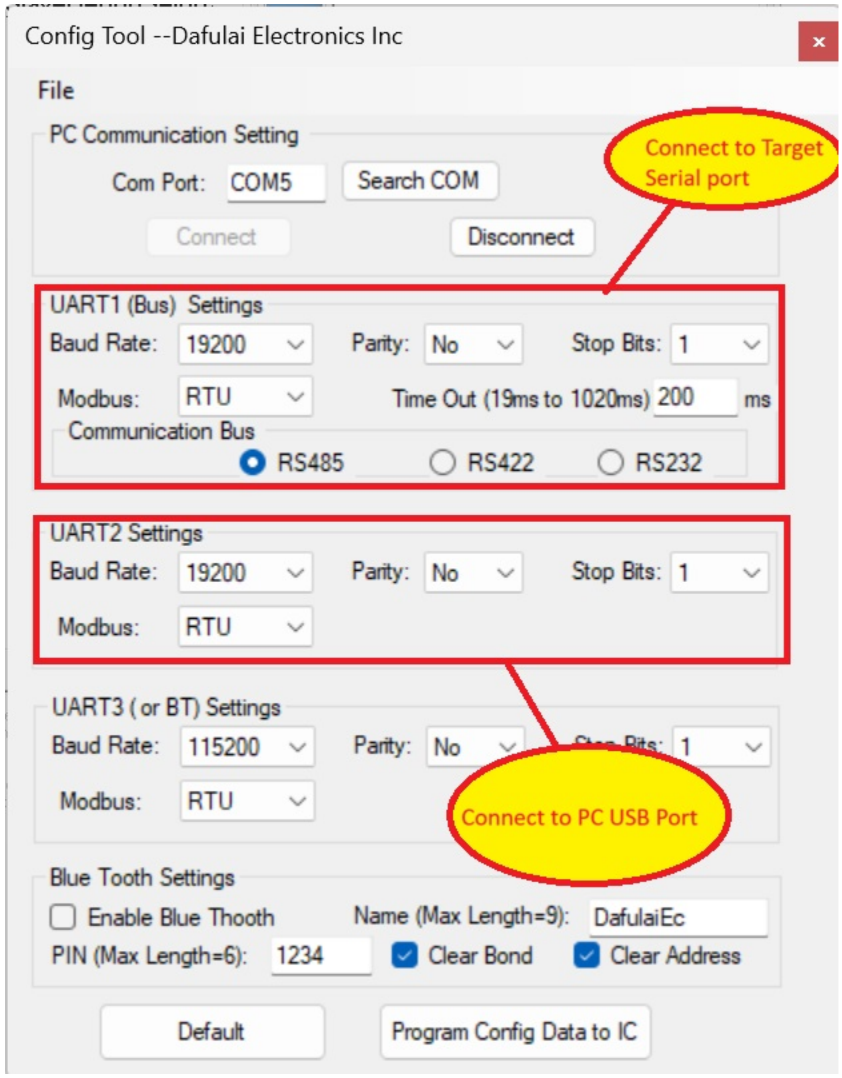


In our hardware environment, we should add "Modbus RTU/ASCII Dual Masters adaptor" as debugger/monitor. Double click on block "embedded Modbus Slave/Debug Setup", we will see the parameters settings below::



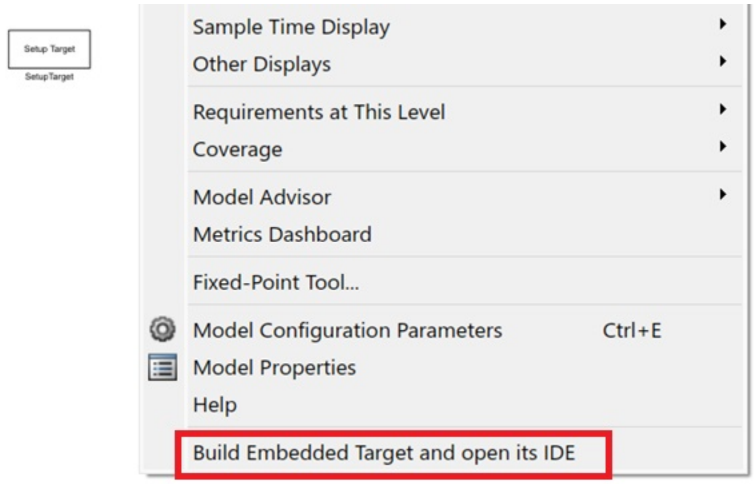
In above figure settings, the red rectangular parameters are important, the others are "Don't care". Please set up them according to your hardware.

Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:

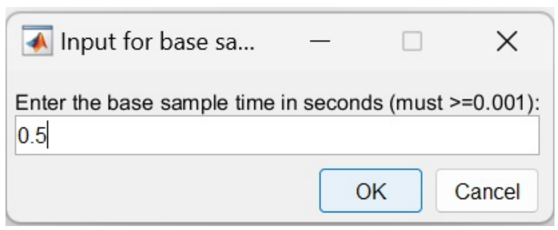


In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

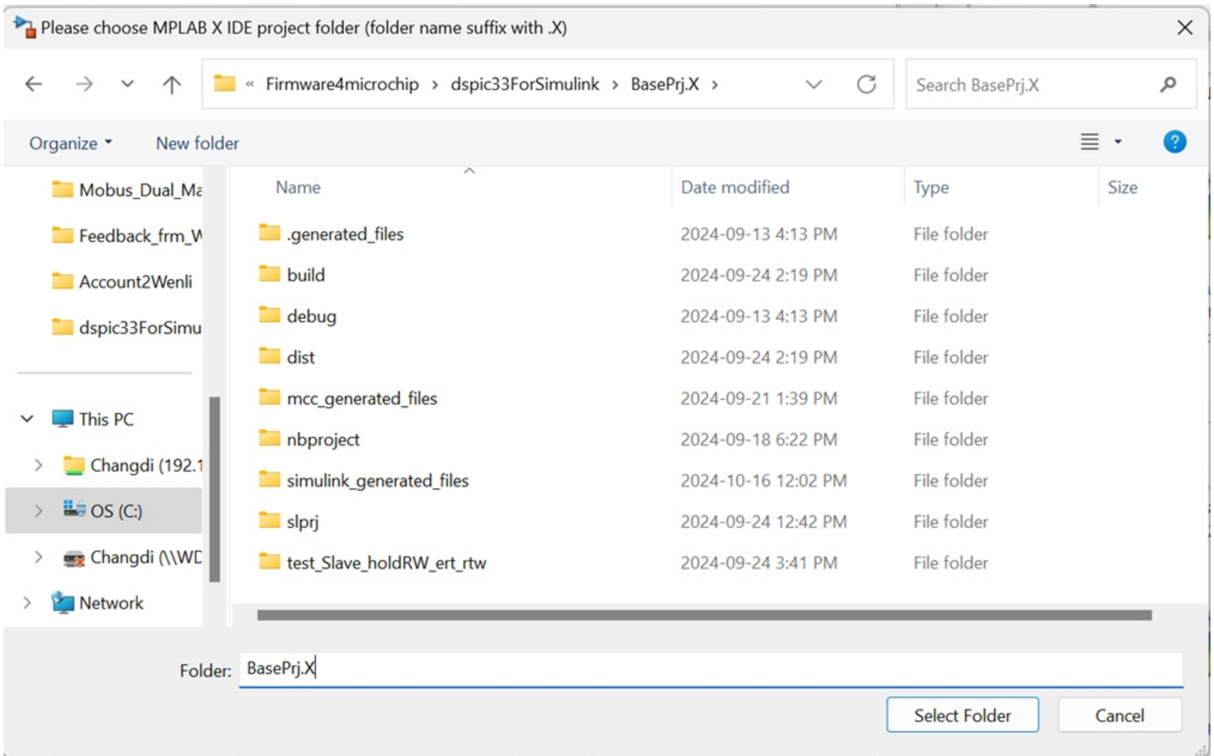
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33, Uart2) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:

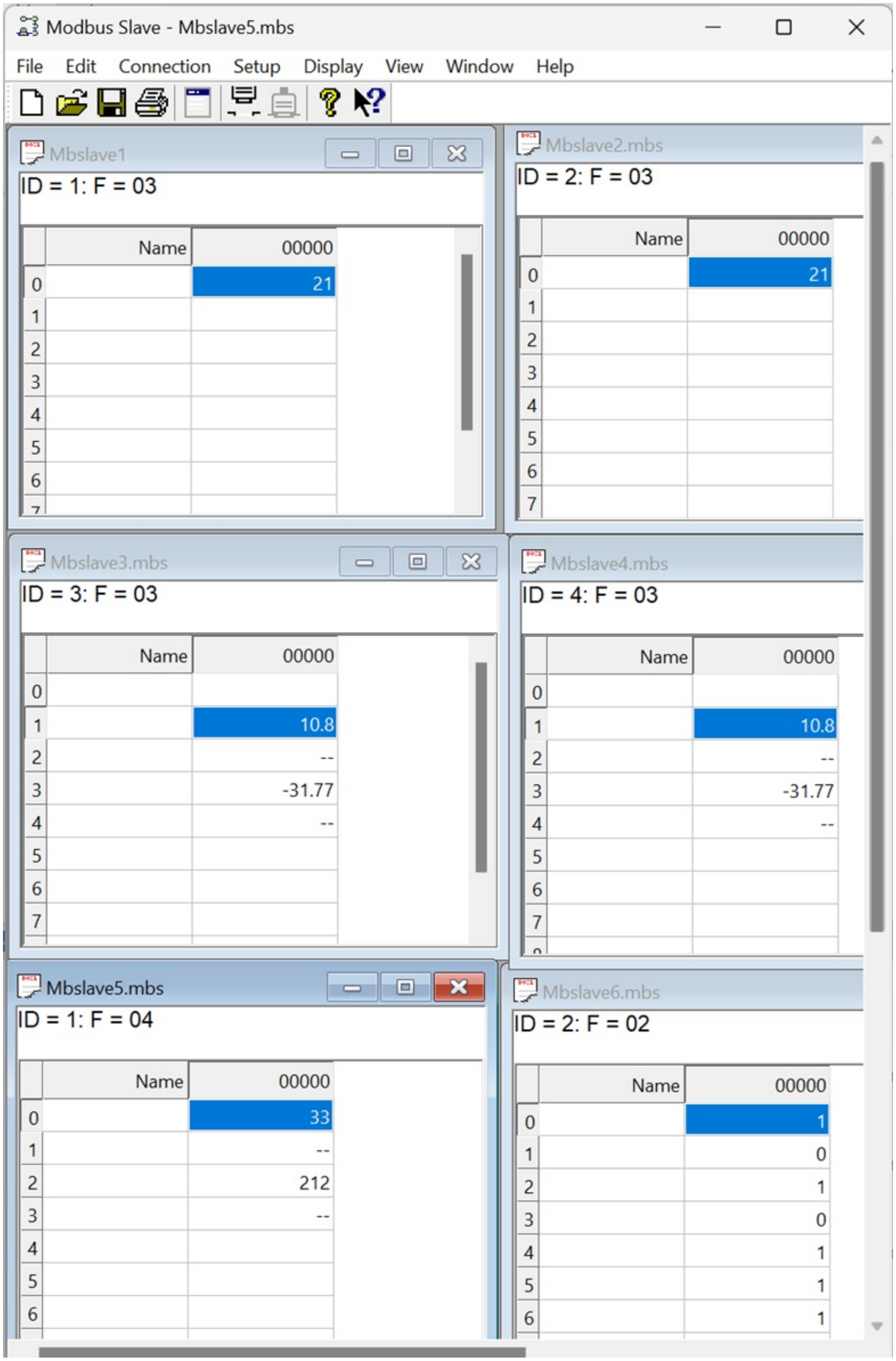


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

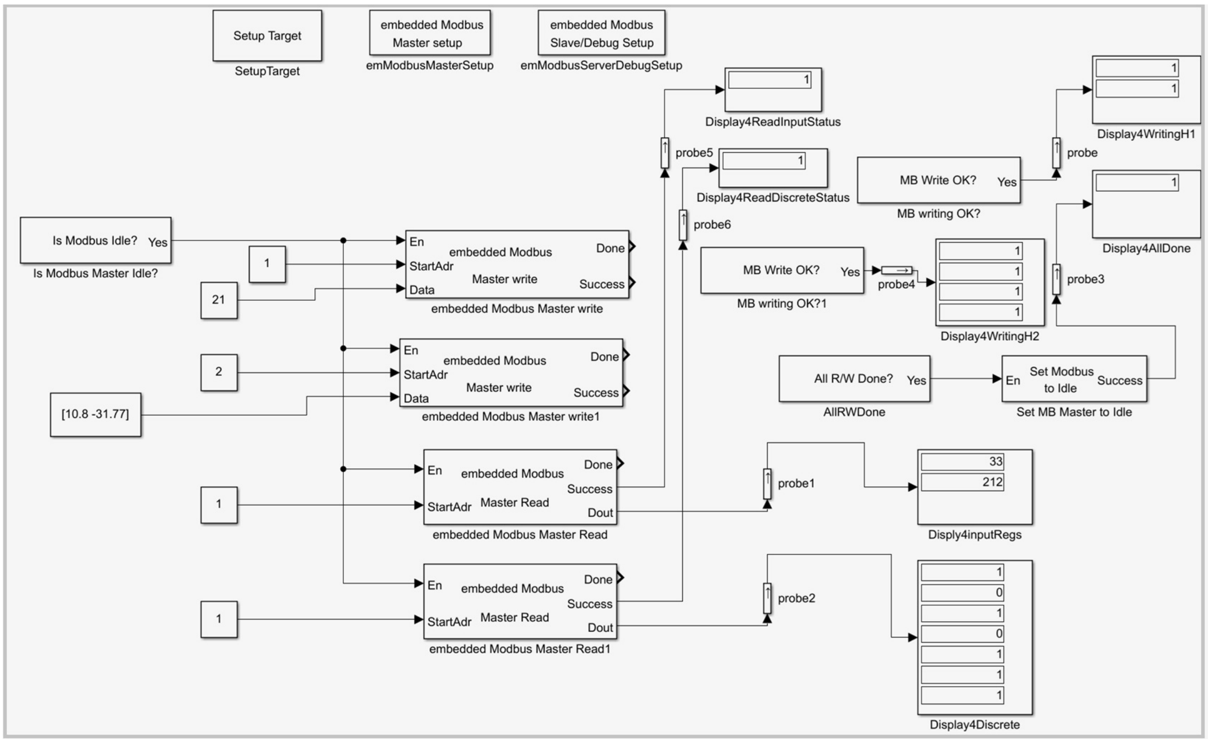
If your firmware program into target successfully, Connect your target Uart1 (RS485/RS422) to your PC, you can use PC Modbus slave software: mbslave.exe to view result below:



In above figures, we set up 6 windows Mbslave1 to Mbslave6.

For Mbslave1 to Mbslave4, the results are the same as example1. We can know writings are OK.
In Mbslave5, we set up input registers value = 33 (32 bits of unsigned integer, big-endian) at address 0 and 1 (0-based without 3x prefix) or at address 30001 and 30002 (1-based with 3X prefix) , and set up input registers value = 212 (32 bits of unsigned integer, big-endian) at address 2 and 3 (0-based without 3x prefix) or at address 30003 and 30004 (1-based with 3X prefix).
In Mbslave6, we set up discrete registers value = [1 0 1 0 1 1 1] at address from 0 to 6 (0-based without 1x prefix) or 10001 to 10007 (1-based with 1X prefix).

Now we can run simulation. By select stop time= inf, and click "Run" button, you will see result below:



From block "Display4ReadInputStatus", value is 1 (true). So block "embedded Modbus Master Read" (Read input regs) success. And in block "Disply4inputRegs", we saw values are 33 and 212 which match "Mbslave5" settings in mbslave.exe software.

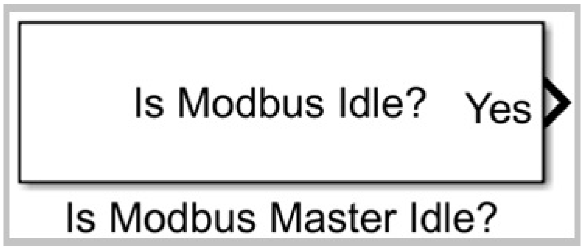
From block "Display4ReadDiscreteStatus", value is 1 (true). So block "embedded Modbus Master Read1" (Read discrete regs) success. And in block "Disply4iDiscretes", we saw values are [1 0 1 0 1 1 1] which match "Mbslave6" settings in mbslave.exe software.

Please open "Your embedded creator library folder"/examples/example7_emMBMaster3.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

isMB_Idle

Is embedded Modbus Master in idle state?
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded Modbus Master / isMB_Idle



Description

This block is used in judgment whether embedded Modbus master is in idle state . When embedded Modbus master is in "Idle" state, You can set up all sequences of modbus master operations (Reading blocks or Writing blocks), and then Master will exit "Idle" state and operate in sequences. We provide block to check if all sequences finish. In order to start new group of modbus operations, you must set Modbus master into "Idle" state when all sequences done.

Parameters

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

- Yes — "logical" data type's scalar. True means "embedded Modbus master is in Idle state, you can start any modbus master operation". False means "embedded Modbus master is in busy state, you can not start any modbus master operation"

Examples

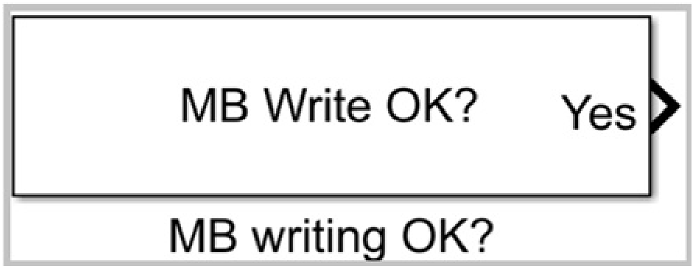
Example1/2/3:

Please see example in block "emModbusMasterWrite".

isMB_WriteOK

Is embedded Modbus Master writing OK?
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded Modbus Master / isMB_WriteOK

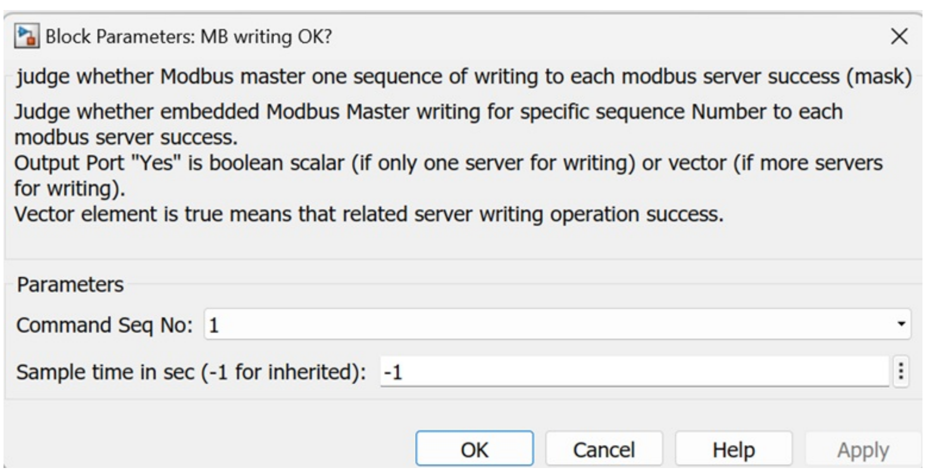


Description

This block is used in judgment whether embedded Modbus master writing for one sequence success . It is used in "Multiple servers writing" (of cause, you can still use it in one server writing) for knowing every server writing status detail.   Why do we need this block? The reason is that the output port "Success" of Block "embedded Modbus Master write" is for entire servers status not for individual server.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Command Seq No — drop list. It is writing sequence Number.

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

- Yes — "logical" data type's scalar or vector. True means "Writing to server success". If it is vector, vector's element will be "Writing to server success" for related server, you can know related server ID from parameter "Server IDs" of block "emModbusMasterWrite" .

Examples

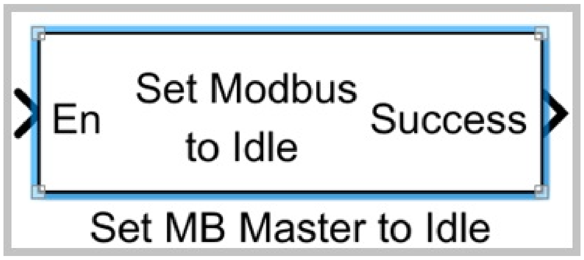
Example1/2/3:

Please see example in block "emModbusMasterWrite".

setMB2Idle

Set embedded Modbus Master to Idle state
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded Modbus Master / setMB2Idle



Description

This block sets up Modbus master to "Idle" State. You must call it under condition: modbus master all reading/writing operations done. Otherwise, you will get unexpected result. Only when Modbus master is in "Idle" state, can you start new operations for modbus master to read/write server registers.

Parameters

None

Ports

Input

- En — "logical" data type's scalar. True means " it sets Modbus master to idle state". False means nothing happens.

Outport

- Success — "logical" data type's scalar. It is equal to "En".

Examples

Example1/2/3:

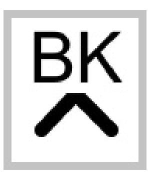
Please see example in block "emModbusMasterWrite".

The following blocks are in "embedded Modbus Slave & Debugger " directory of library. They are all related to Modbus Slave and Debugger/Monitor operation. We list them according to alphabetical order.

emBreakpoint

embedded breakpoint.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / embedded Modbus Slave & Debugger / emBreakpoint



Description

This block will set one breakpoint in any output port of any block. All white color background of " emBreakpoint"s are enabled. All grey color background of "emBreakpoint"s are disabled. If code stops in one "emBreakpoint" block, its background will become red color.

***Notes**: 1 During pause, "emWait" 's timer is still running, "RTC" 's timer is still running.*

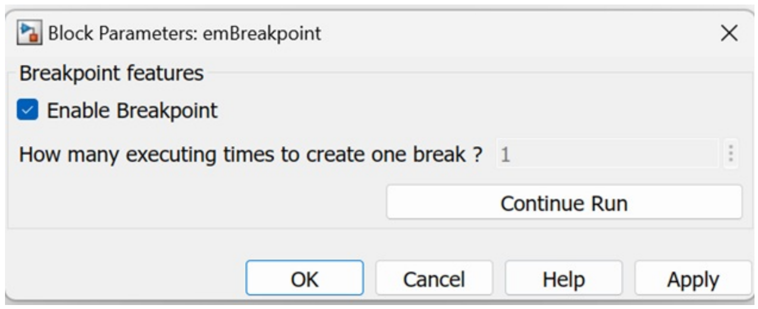
Modbus Server can still response to external Modbus Master request.

*2 If "*emBreakpoint" blocks are inside interrupt sample, this interrupt priority must be lower than timer1's interrupt (1ms tick created) and Uart's interrupt which is used in Monitor/Debugger.

3 In Simulink model, block's executing order is decided by "block priority". Lower value has higher executing priority. You can set block's priority in Block Properties window. Actual block executing order can be viewed by click menu item "DEBUG/ Information Overlays / BLOCKS / Execution Order" of "Simulink main menu"

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Enable Breakpoint — "checked on" will enable "this break point". If you didn't connect "Debug hardware", all breakpoints are disabled automatically.
- How many executing times to create one break ?— uint8 type of scalar. 1 to 255 denotes one code break needs how many times to run here.

Notes: *"Continue Run*" is button, not parameter. It is used to run again when code stops here.

Ports

Input

- — Any data type's scalar or vector. Connect to any output port of block. If "Enable Breakpoint" is true, and specified times to run here arrive, Code execution will stop. The block background color "Red" indicates "Stop"

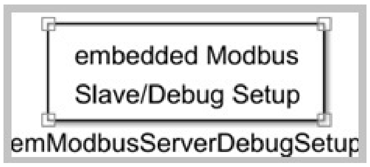
Outport

None

emModbusSlaveDebugSetup

set up embedded Modbus Server or Debug/monitor
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / embedded Modbus Slave & Debugger / emModbusSlaveDebugSetup



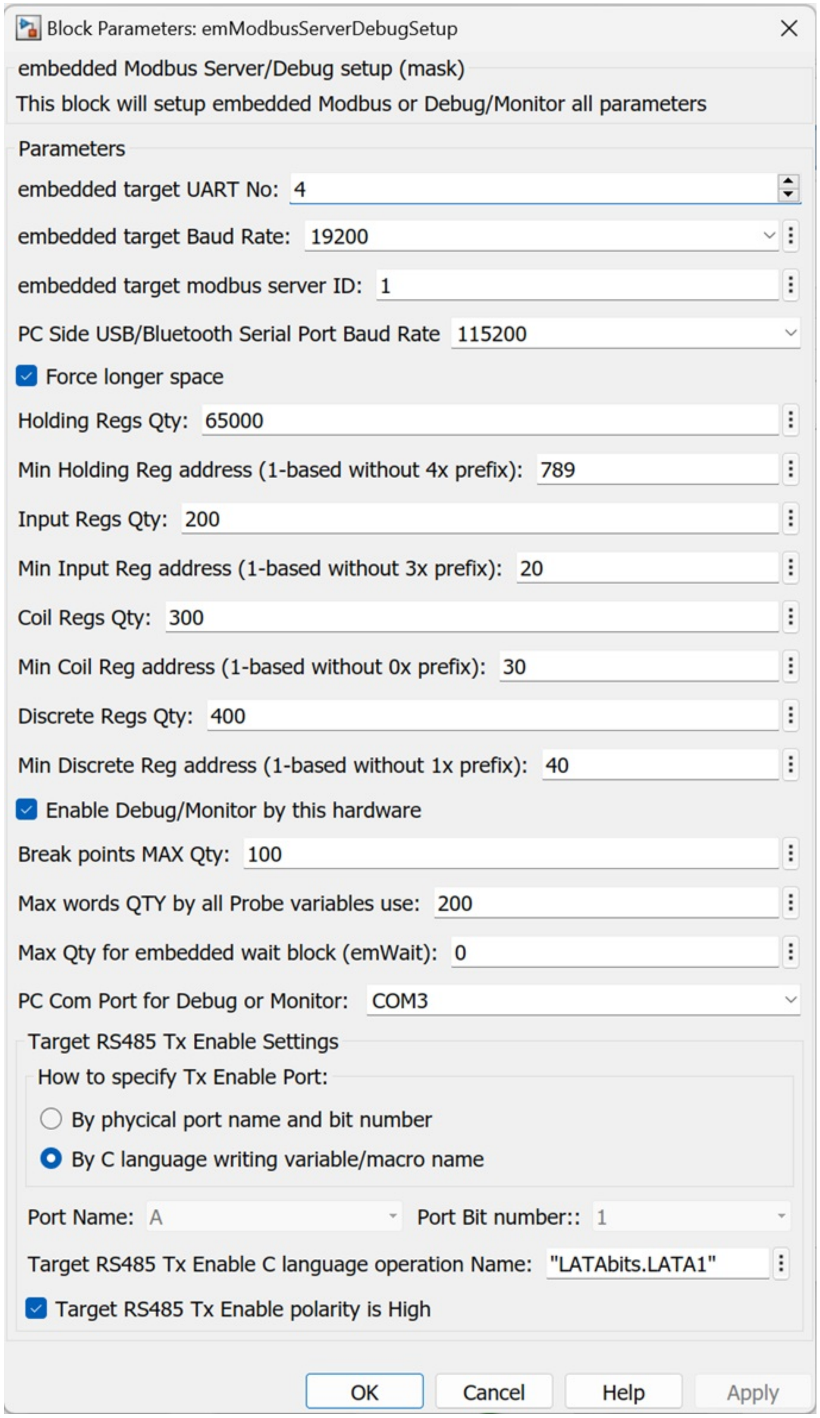
Description

This block sets up Modbus Server or Debug/Monitor parameters.

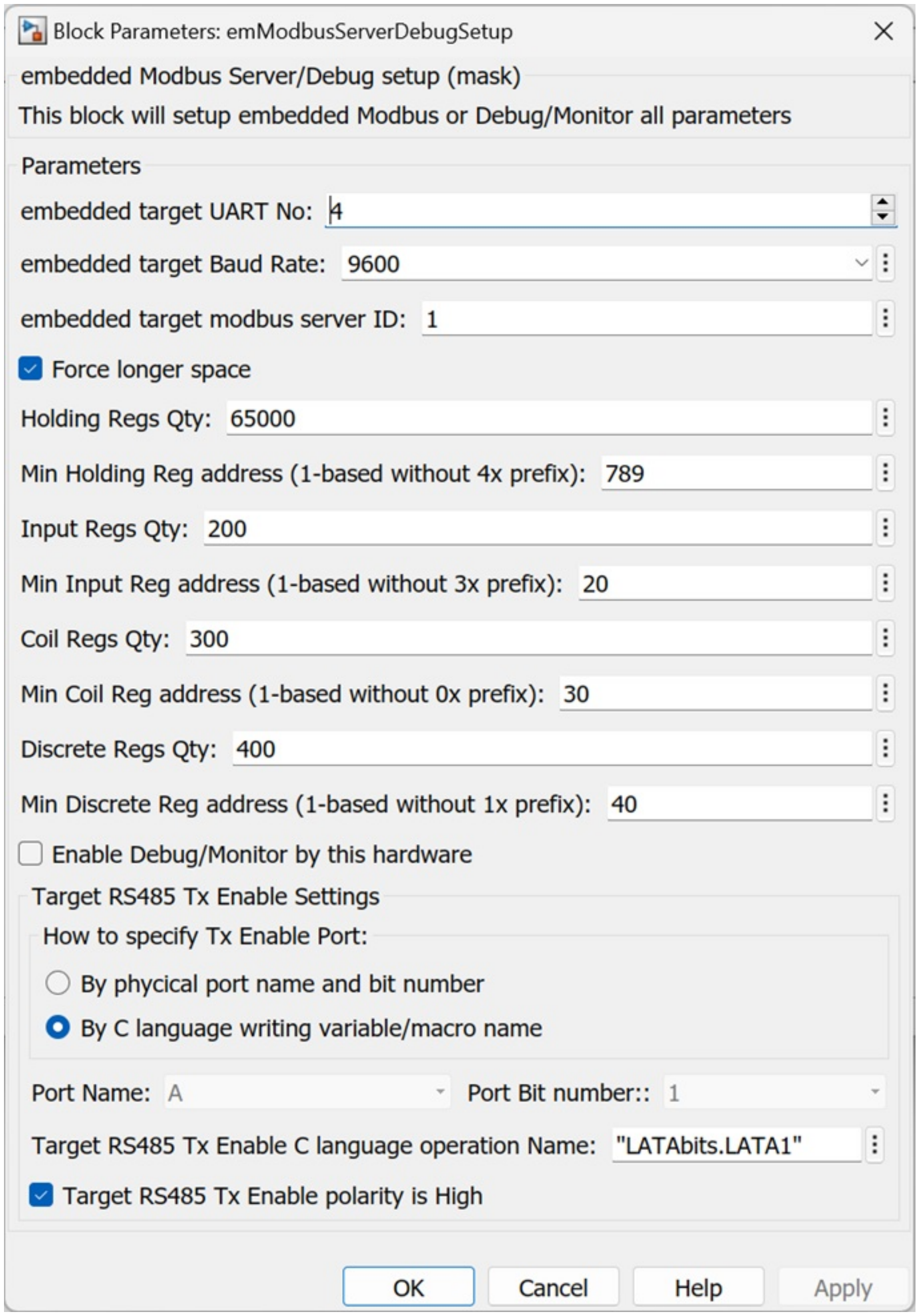
> ***Notes**: You can add maximum 4 embedded Modbus servers if you don't use embedded Modbus client.. So you can add maximum 4 "emModbusSlaveDebugSetup" blocks if you don't use embedded Modbus client. And one of these blocks can be used debug/monitor purpose beside Modbus server function.   When you use embedded Modbus client, Maximum 3 embedded Modbus servers can be used.*

Parameters

Please double click this block to open parameters dialog below:



We we have no make parameter "Enable Debug/Monitor by this hardware" to check on , the parameters dialog will be shown below:



Let us explain parameters.

- embedded target UART No: — In one embedded Micro-controller, there are many serial ports. This drop list parameter will tell target which serial port of Micro-controller will be used for this embedded server or debug/monitor. Different Modbus server or client cannot use the same serial ports.
- embedded target Baud Rate — embedded serial port baud rate in unit bit/sec. Actually, actual baud rate is decided by embedded target IDE configuration software. Here just use it to decide time for Modbus RTU packet separation (3.5 characters)

- embedded target modbus server ID: — It is Modbus Server ID or Server address. Valid Server ID is 1 to 247. If two Modbus Servers are not connected to the same Modbus network physically, the servers can use the same Server ID. Otherwise, you can not assign different Servers to the same Server ID.

- PC Side USB/Bluetooth Serial Port Baud Rate — If you use this Modbus Server as debug/monitor (Parameter "Enable Debug/Monitor by this hardware" is checked on), this parameter will be visible. We will use "Modbus RTU/ASCII Dual Masters adaptor" from Dafulai Electronics Inc to debug/monitor embedded system in Simulink environment. You must use software "ConfigTool.exe" to configure "Modbus RTU/ASCII Dual Masters adaptor" firstly. Make sure PC Side baud rate is equal to this parameter, and Bus side baud rate is equal to parameter " embedded target Baud Rate ".

- Force longer space — true will make my server response packet to be separated with master request packet much longer time (in standard Modbus RTU Protocol, it is 3.5 characters time). It will increase anti-noise ability. false will use standard 3.5 characters space time.

- Holding Regs Qty — My server supports how many holding registers. Valid Range is 0 to 65536. 0 means we don't support holding register for this server.

- Min Holding Reg address (1-based without 4x prefix) — In my server supported holding register range, it is minimum address my server supports.

- Input Regs Qty — My server supports how many input registers. Valid Range is 0 to 65536. 0 means we don't support input register for this server.

- Min Input Reg address (1-based without 3x prefix) — In my server supported input register range, it is minimum address my server supports.

- Coil Regs Qty — My server supports how many coil registers. Valid Range is 0 to 65536. 0 means we don't support coil register for this server.

- Min Coil Reg address (1-based without 0x prefix) — In my server supported coil register range, it is minimum address my server supports.

- Discrete Regs Qty — My server supports how many discrete registers. Valid Range is 0 to 65536. 0 means we don't support discrete register for this server.

- Min Discrete Reg address (1-based without 1x prefix) — In my server supported discrete register range, it is minimum address my server supports.

- Enable Debug/Monitor by this hardware — true means that we will use "Modbus RTU/ASCII Dual Masters adaptor" from Dafulai Electronics Inc to debug/monitor embedded system in Simulink environment. false means that we don't use this server for Debug/Monitor purpose. Note: If it is true, you can still use this server as general Modbus Server function beside debug/monitor function.

- Break points MAX Qty — If you use this Modbus Server as debug/monitor (Parameter "Enable Debug/Monitor by this hardware" is checked on), this parameter will be visible. It sets up maximum "break points" QTY. Valid Range is 0 to 100

- Max words QTY by all Probe variables use — If you use this Modbus Server as debug/ monitor (Parameter "Enable Debug/Monitor by this hardware" is checked on), this parameter will be visible. It sets up maximum Qty in unit word by Probe blocks. Probe is used for watching signals value of simulink block for debug/monitor. Valid Range is 0 to 65355

- Max Qty for embedded wait block (emWait) — If you use this Modbus Server as debug/ monitor (Parameter "Enable Debug/Monitor by this hardware" is checked on), this parameter will be visible. It is Maximum QTY of block "emWait" which is used for delay some time (non-blocking function). You can view help of "emWait" to know details. Valid Range is 0 to 100

- PC Com Port for Debug or Monitor — If you use this Modbus Server as debug/monitor (Parameter "Enable Debug/Monitor by this hardware" is checked on), this parameter will be visible. This drop list will tell system which COM Port will be used by "Modbus RTU/ASCII Dual Masters adaptor" for debug/monitor purpose.

- How to specify Tx Enable Port — In RS485/422 interface, generally, you need one GPIO output to control Tx Enable. You can choose one of 2 ways to specify this GPIO outport. "By physical port name and bit number" will specify which GPIO Port Name (drop list) and which Port bit number (drop list) used for "Tx Enable". "By C language writing variable/macro name" will directly specify C language's GPIO output variable or macro for "Tx Enable". If your hardware don't use "Tx Enable", you just give this parameter empty string "".

- Port Name — It is for "Tx Enable". When you use "By physical port name and bit number", it specify GPIO Port Name (drop list).

- Port Bit number — It is for "Tx Enable". When you use "By physical port name and bit number", it specify which Port bit number (drop list).

- Target RS485 Tx Enable C language operation Name — It is for "Tx Enable". When you use "By C language writing variable/macro name", it will directly specify C language's GPIO output variable or macro for "Tx Enable". If your hardware don't use "Tx Enable", you just give this parameter empty string "".

- Target RS485 Tx Enable polarity is High — It is for "Tx Enable". True means GPIO port Logic High will enable "RS485/RS422" transmit. False means GPIO port Logic Low will enable "RS485/RS422" transmit.

Ports

Input

None

Outport

None

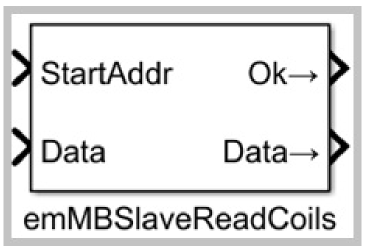
Examples

Please see "emModbusSlaveReadInputRegs" block example

emModbusSlaveReadCoils

embedded Modbus Server updates coil register value.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / embedded Modbus Slave & Debugger / emModbusSlaveReadCoils



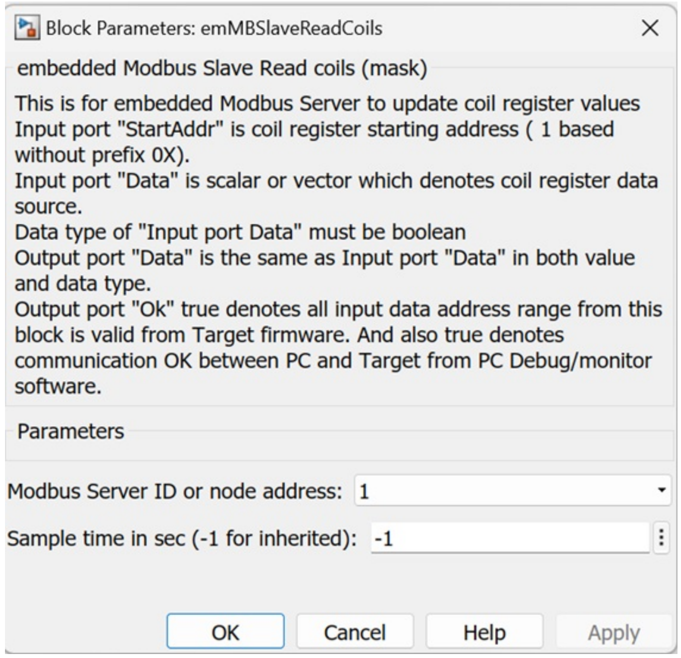
Description

This block will update my Modbus server coil register value periodically. So External Modbus client can get the latest coil register value.

Notes*: Any Port name with "Right Arrow" (→) symbol contains "built-in" probe. So in PC side, you can watch its value directly by "Display" block, you don't need add "Probe" block (our emProbe) before "Display" block. Of cause, you can still use "Mux" block to collect all watching variables and then connect one "emProbe" which connects "Display" block. In this way, you can decrease communication traffic.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Modbus Server ID or node address — tell system this block is for which Modbus Server node. You just choose from drop list which is from "emModbusSlaveDebugSetup" block.
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is Modbus Server coil registers' start address (1-based without prefix "0X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.
- Data — "logical" data type's scalar or vector. It is coil register data source.

Outport

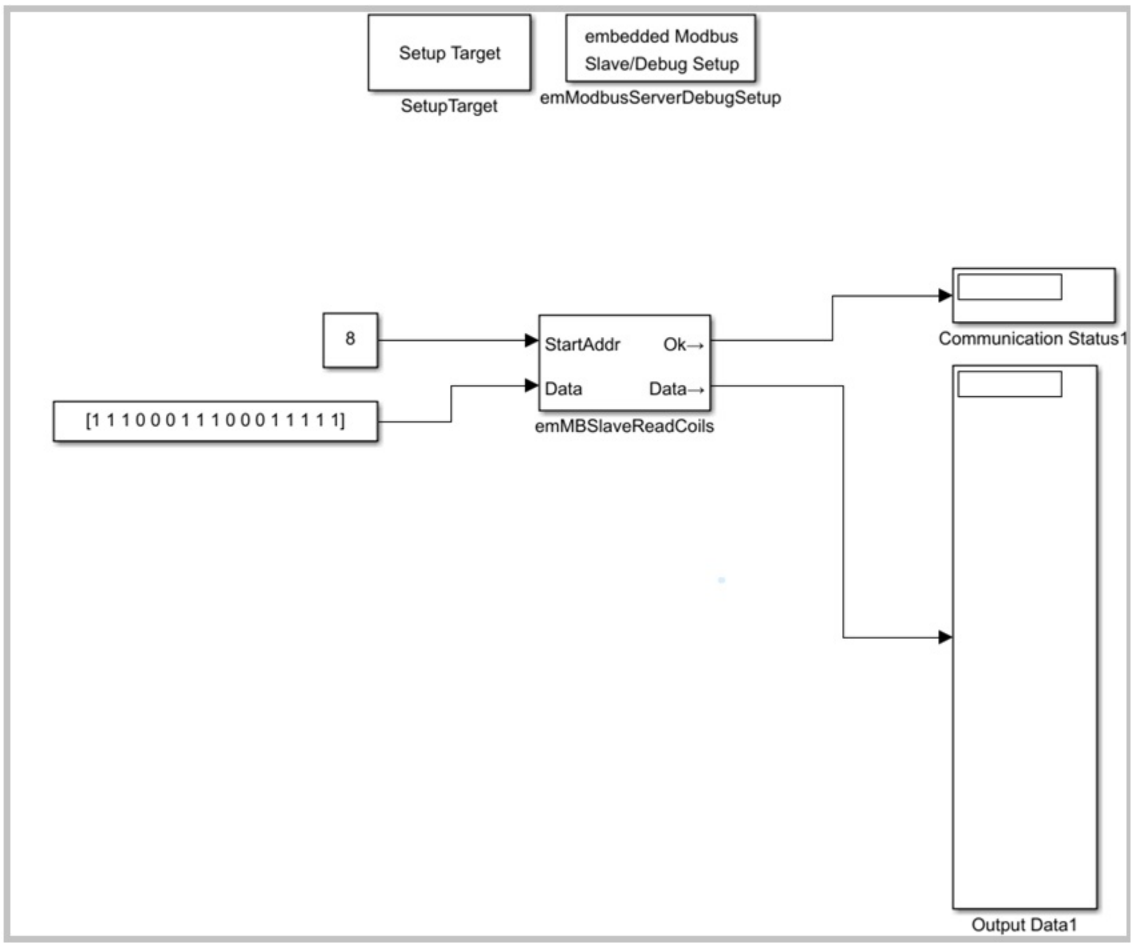
- Ok — "logical" data type's scalar. In PC side, it denotes whether communication between PC and Target is OK. In embedded target side, it denotes whether coil register range you read is valid.
- Data — "logical" data type's scalar or vector. It is equal to input port Data.

Examples

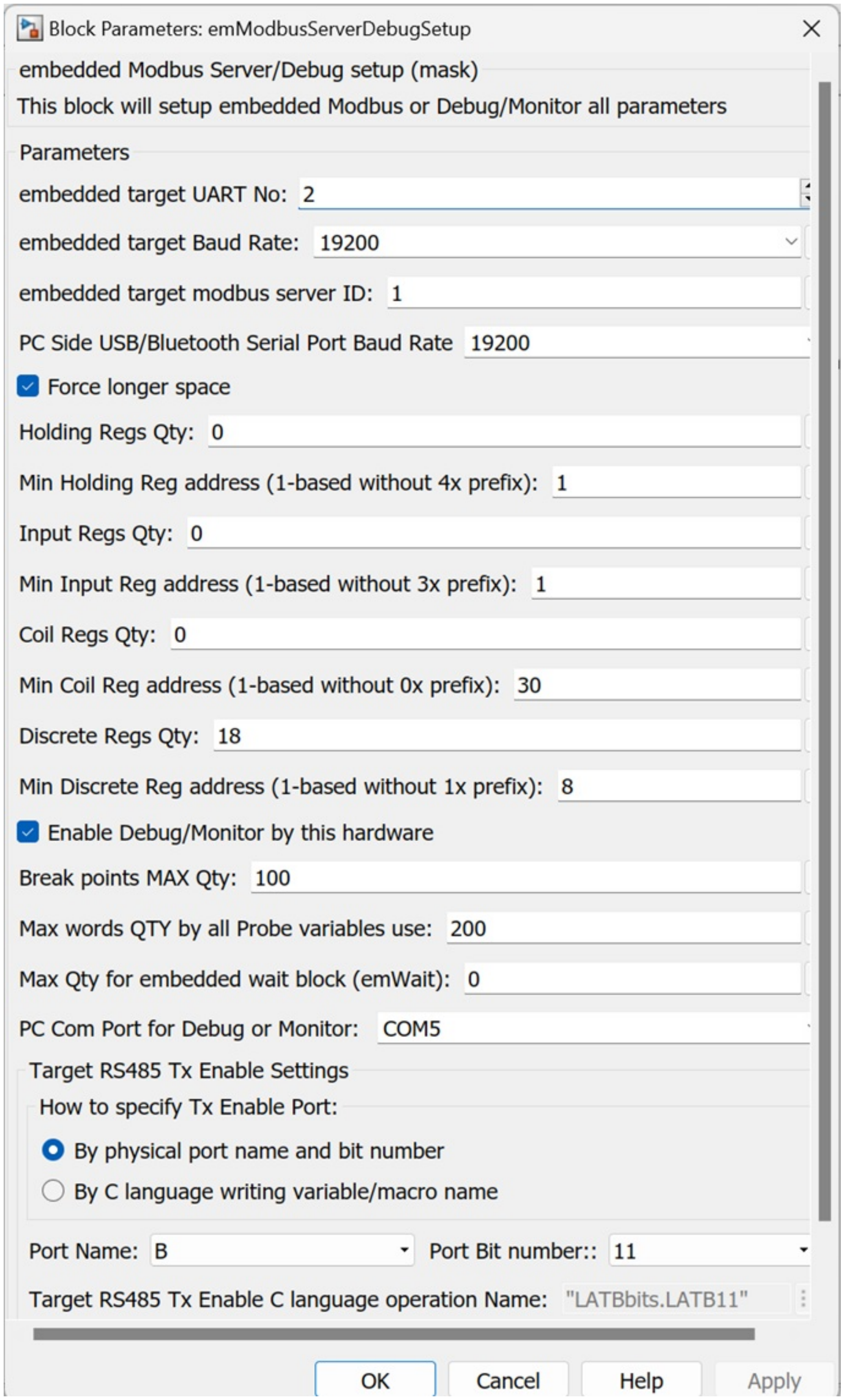
Example1:

Our embedded platform is dsPIC33EP256GP502 . Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. Our embedded Modbus Server ID=1, and supports coil registers address range: 00008 to 00025, baud rate=19200

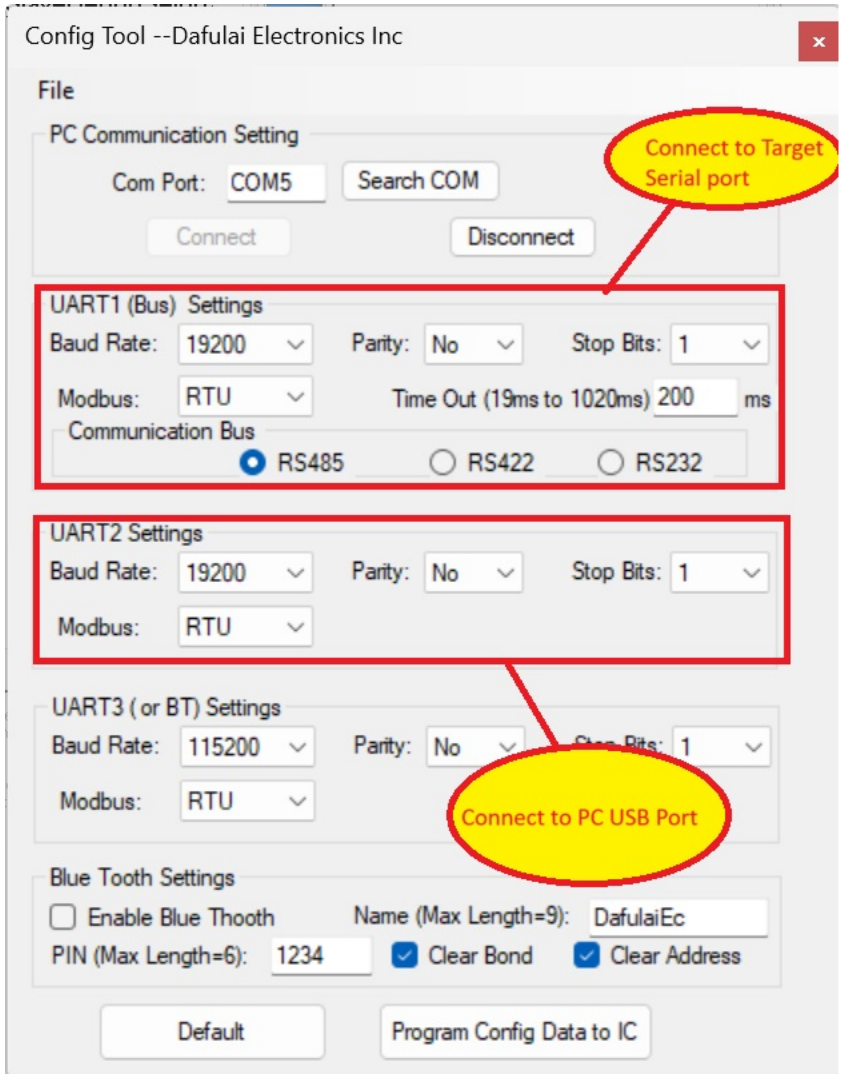
Please see screenshot of model below:



In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Double click "emModbusServerDebugSetup", we will see the Modbus Server/Debug parameters settings below:

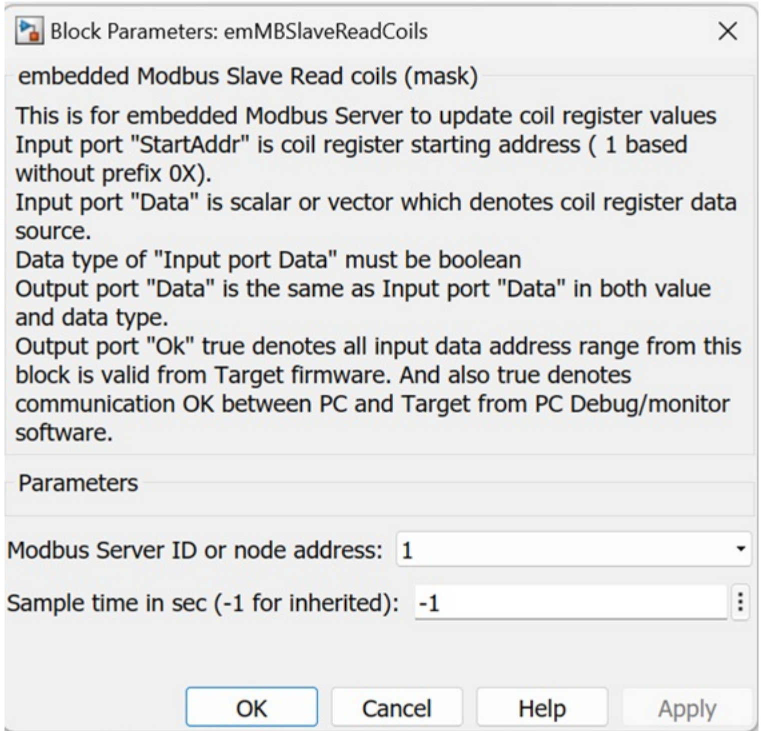


In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

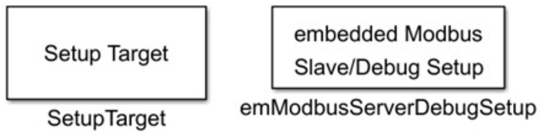
Double click "emMBSlaveReadCoils", we will see the "emMBSlaveReadCoils" parameters settings below:

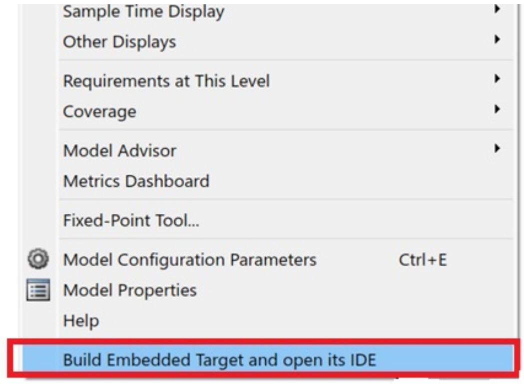


It means this block will update coil register from Modbus server ID=1. This block's in-port "StartAddr" connects Constant block with value= 8 which denotes coil register starting address is 00008. This block's in-port "Data" connects "Constant" block output, which provide boolean vector [1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 1] to in-port "Data". So we will see Modbus server (ID=1) coil register 00008 to 00010 with value "true", coil register 00011 to 00013 with value "false", ... , coil register 00020 to 00024 with value "true".

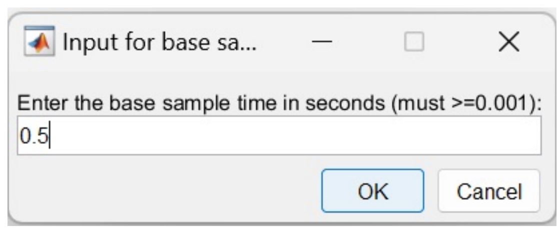
Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too). We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"

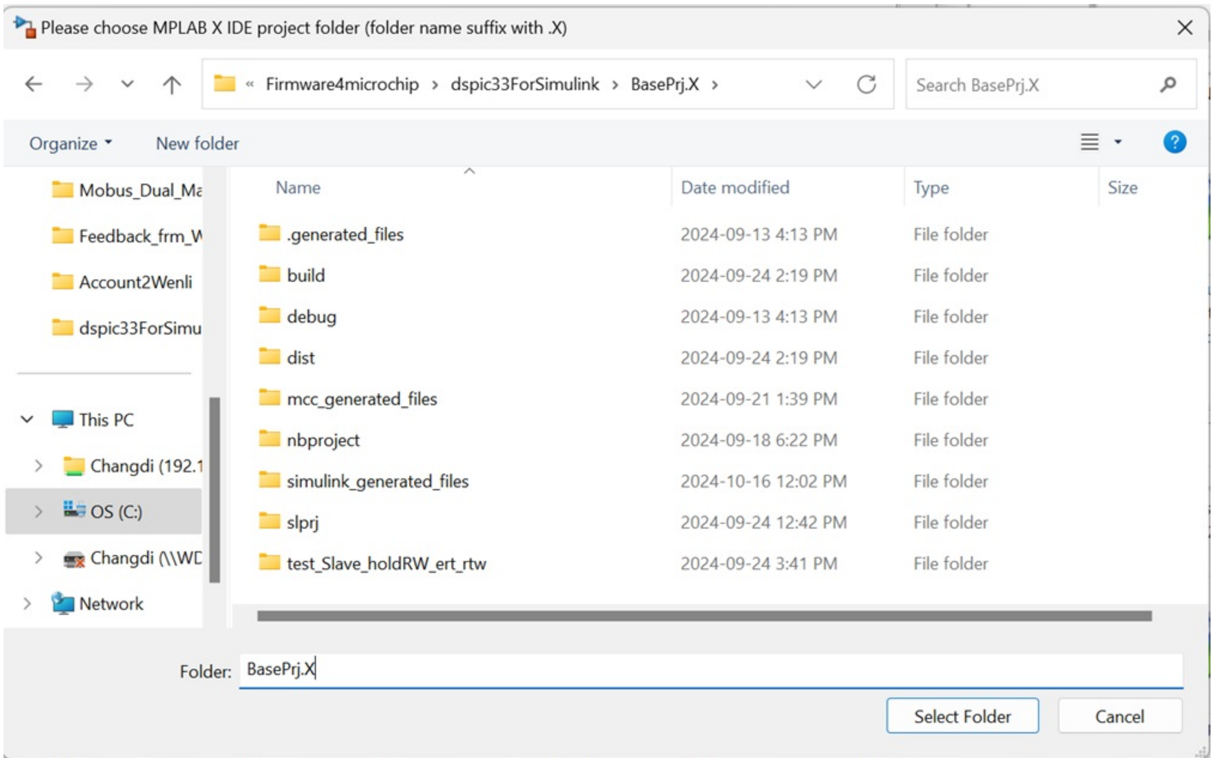




It will start build, and popup dialog window to input base sampling period:

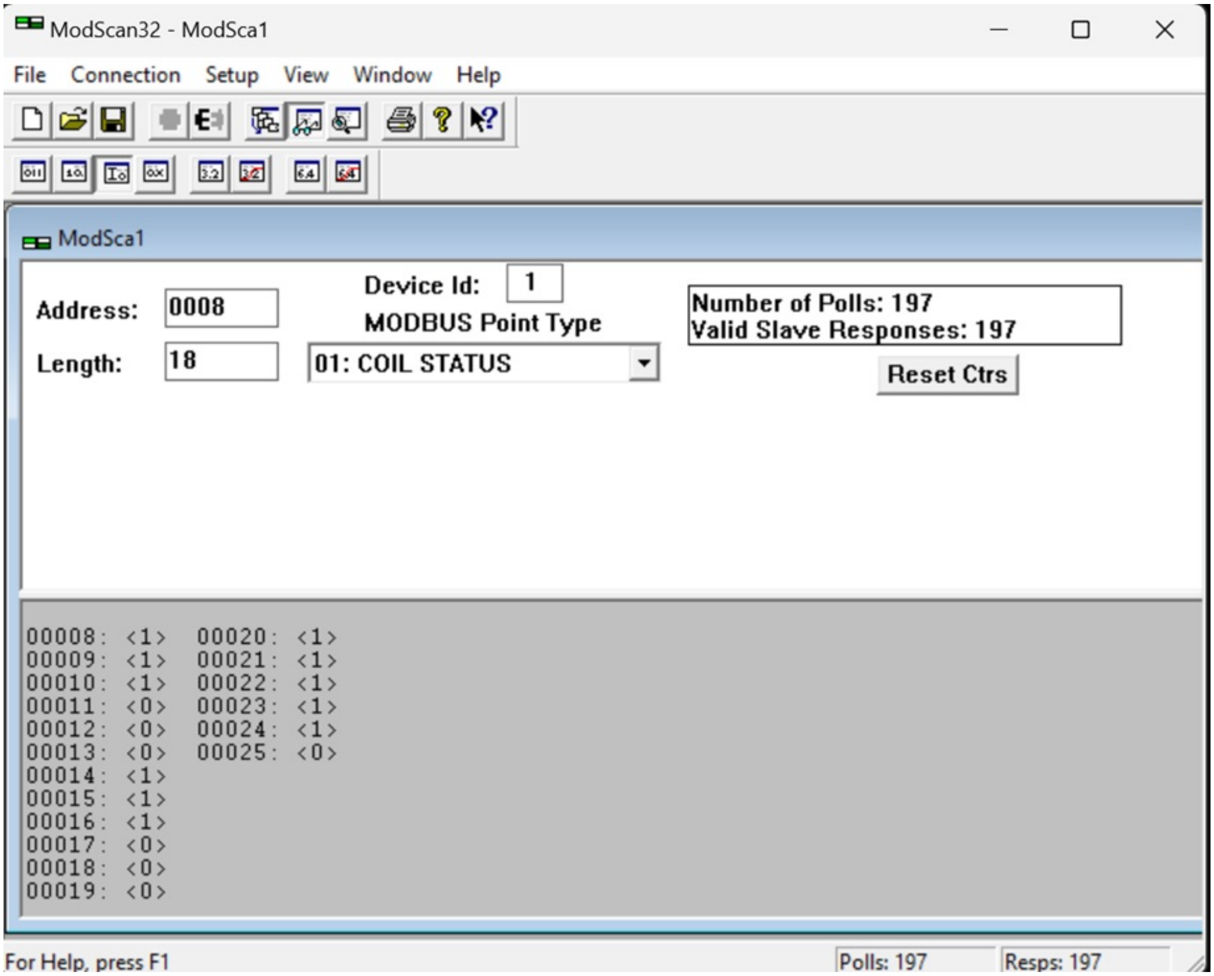


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



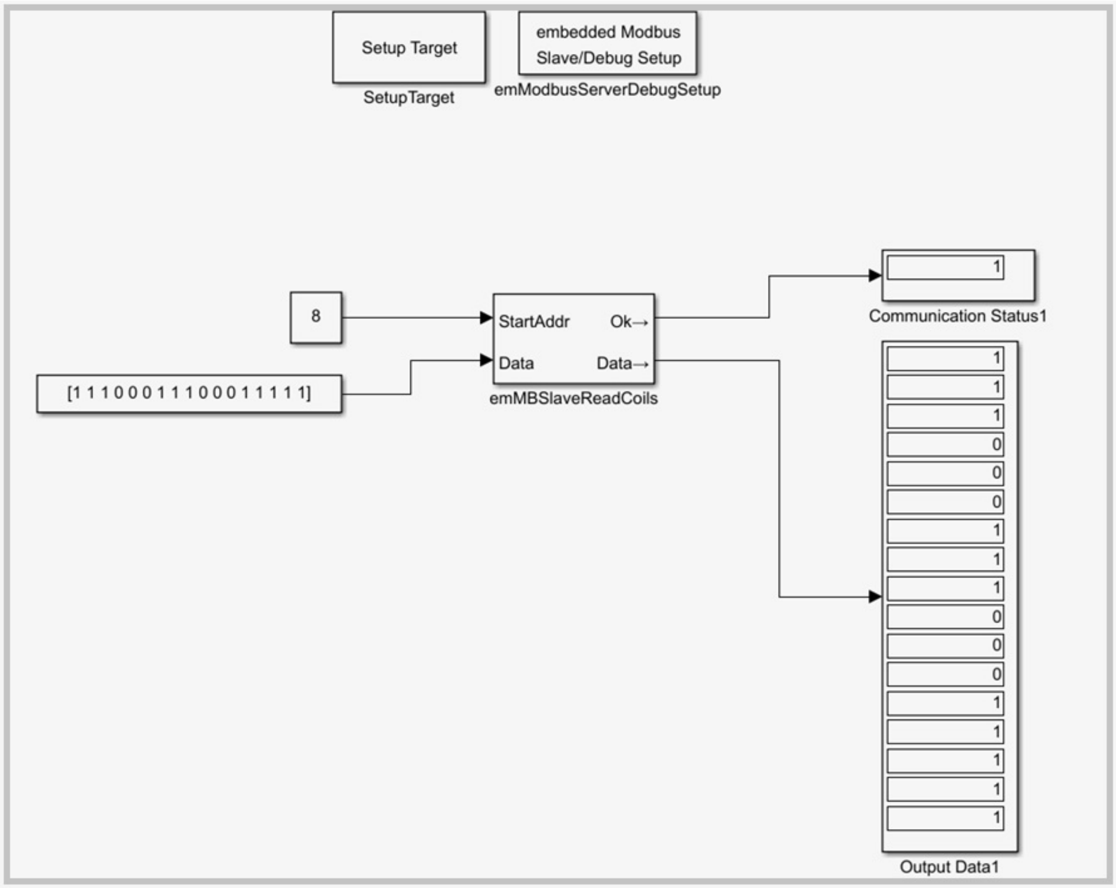
If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

If your firmware program into target successfully, you can use ModScan32 software to view result below:



All results are what we expect.

Disconnect ModScan32 software, we can use Simulink directly view results. By select stop time= inf, and click "Run" button, you will see result below:



You will see all results in all "Display" Blocks.

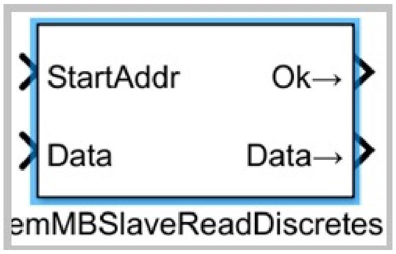
***Notes**: You can run both ".*ModScan32 " and "Simulink" at the same time. But you must enable "Modbus RTU/ASCII Dual Masters adaptor" Bluetooth, .and "ModScan32" cannot use the same COM port as Simulink. We recommend to use Simulink instead of ".ModScan32 " because " ModScan32 " cannot watch general variables. Simulink can watch any variables by "Probe" (also called emProbe) blocks.

Please open "Your embedded creator library folder"/examples/example4_emReadCoils.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emModbusSlaveReadDiscretes

embedded Modbus Server updates discrete register value.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / embedded Modbus Slave & Debugger / emModbusSlaveReadDiscretes



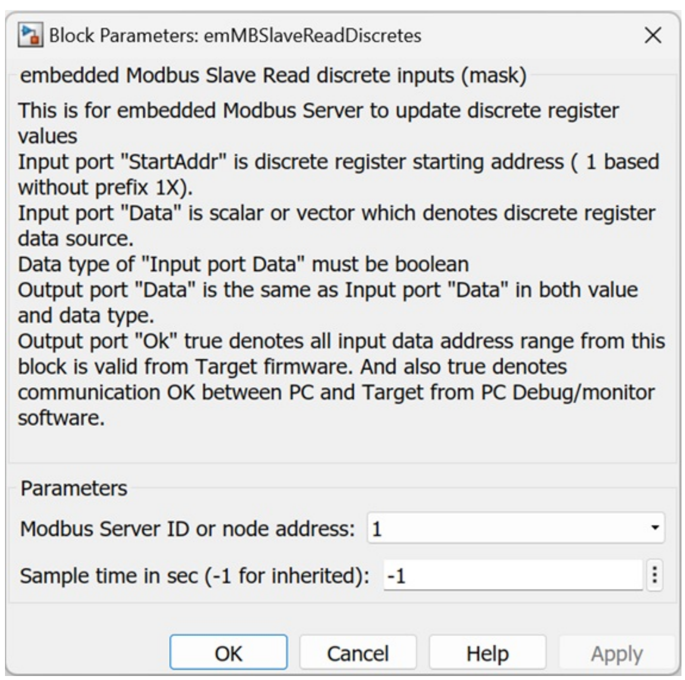
Description

This block will update my Modbus server discrete register value periodically. So External Modbus client can get the latest discrete register value.

> ***Notes**: Any Port name with "Right Arrow" (→) symbol contains "built-in" probe. So in PC side, you can watch its value directly by "Display" block, you don't need add "Probe" block (our emProbe) before "Display" block. Of cause, you can still use "Mux" block to collect all watching variables and then connect one "emProbe" which connects "Display" block. In this way, you can decrease communication traffic.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Modbus Server ID or node address — tell system this block is for which Modbus Server node. You just choose from drop list which is from "emModbusSlaveDebugSetup" block.
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is Modbus Server discrete registers' start address (1-based without prefix "1X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.

- Data — "logical" data type's scalar or vector. It is discrete register data source.

Outport

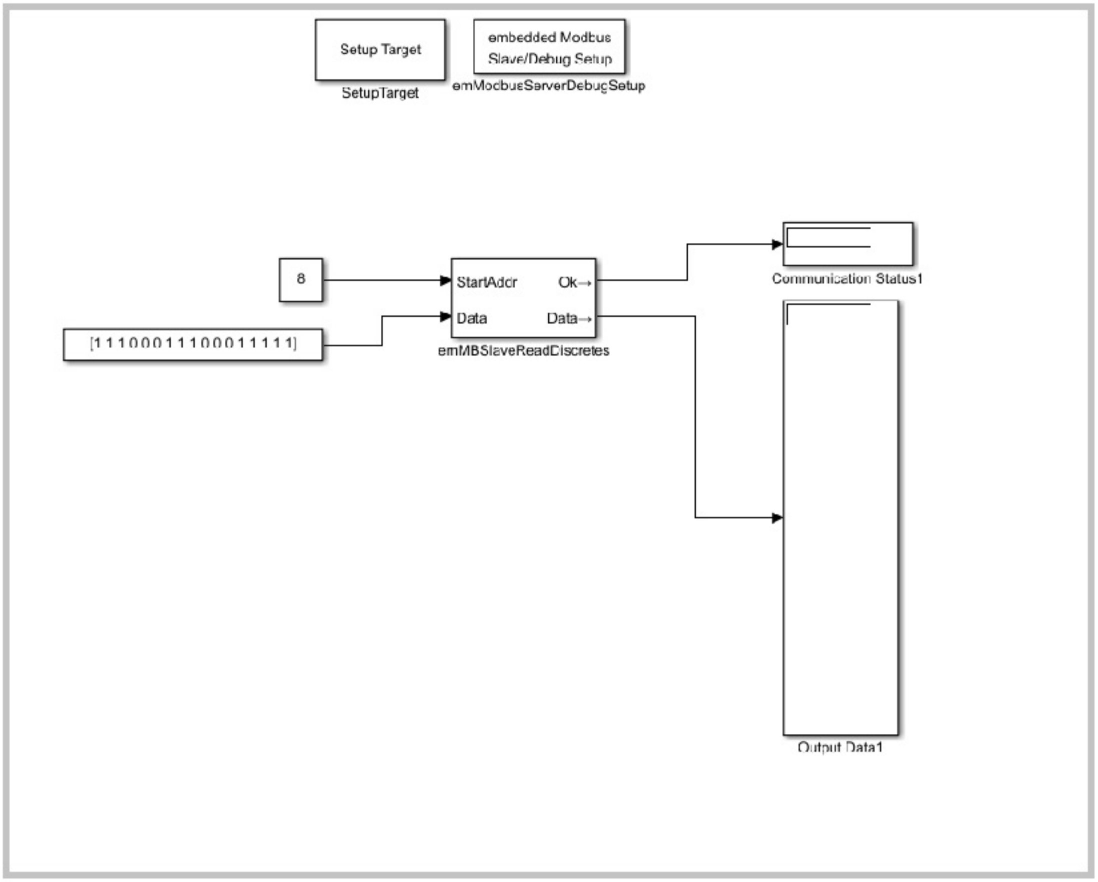
- Ok — "logical" data type's scalar. In PC side, it denotes whether communication between PC and Target is OK. In embedded target side, it denotes whether discrete register range you read is valid.
- Data — "logical" data type's scalar or vector. It is equal to input port Data.

Examples

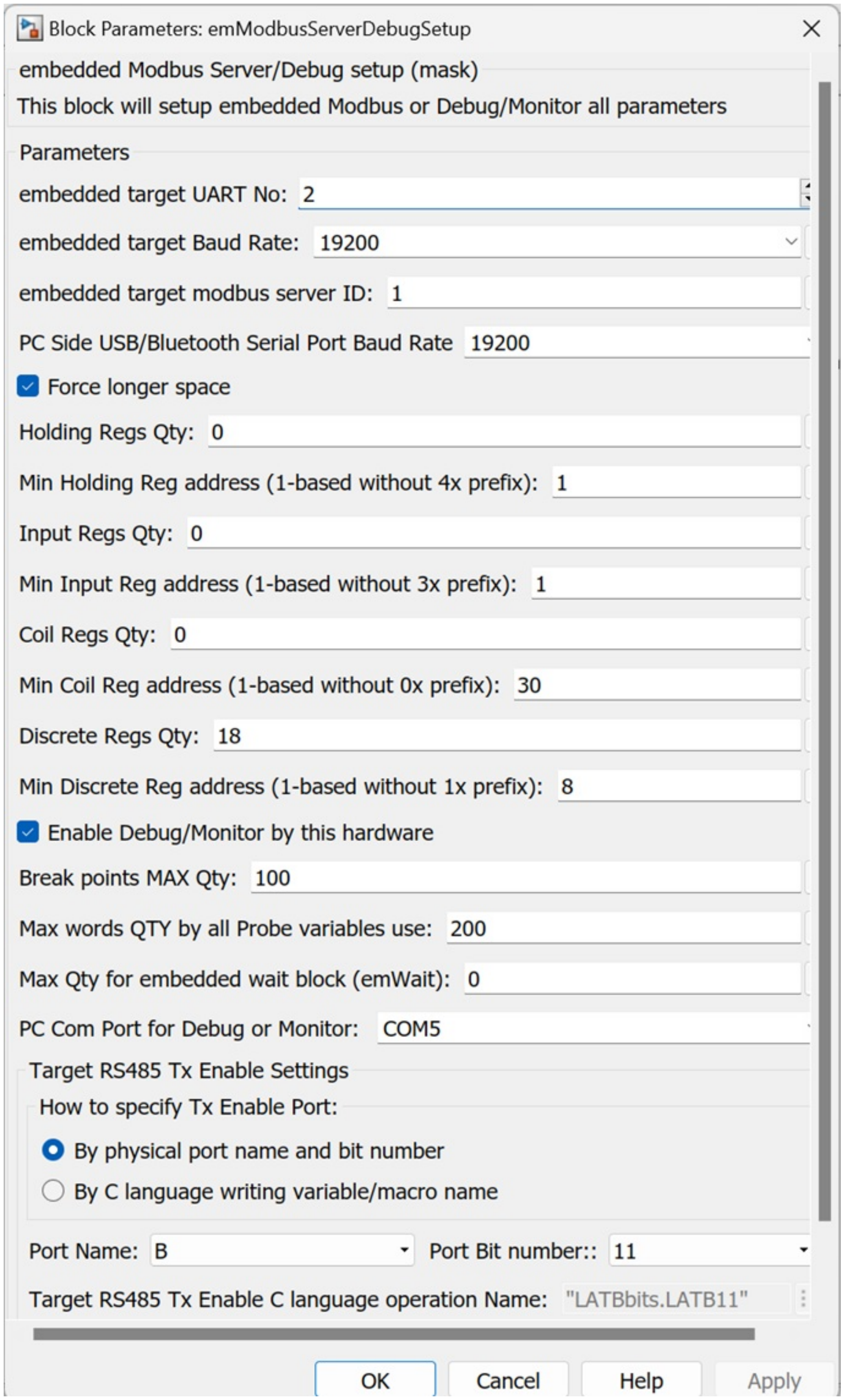
Example1:

Our embedded platform is dsPIC33EP256GP502 . Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. Our embedded Modbus Server ID=1, and supports discrete registers address range: 10008 to 10025, baud rate=19200

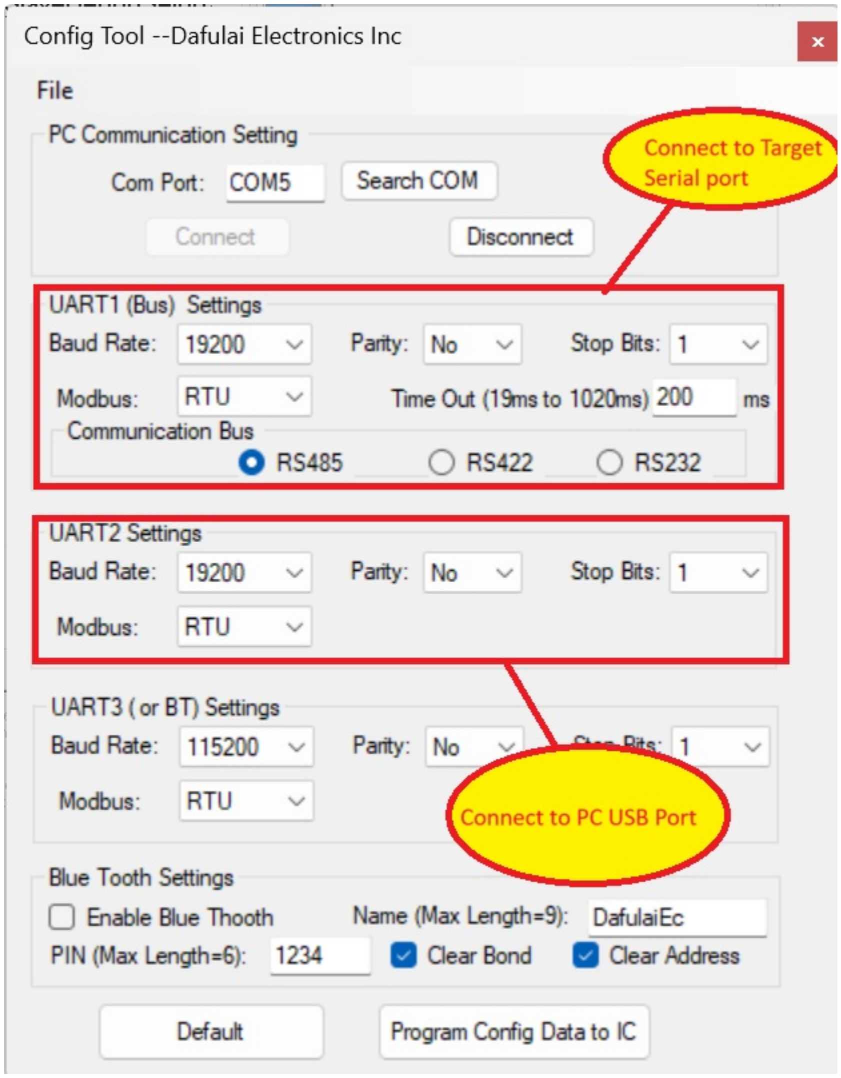
Please see screenshot of model below:



In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Double click "emModbusServerDebugSetup", we will see the Modbus Server/Debug parameters settings below:

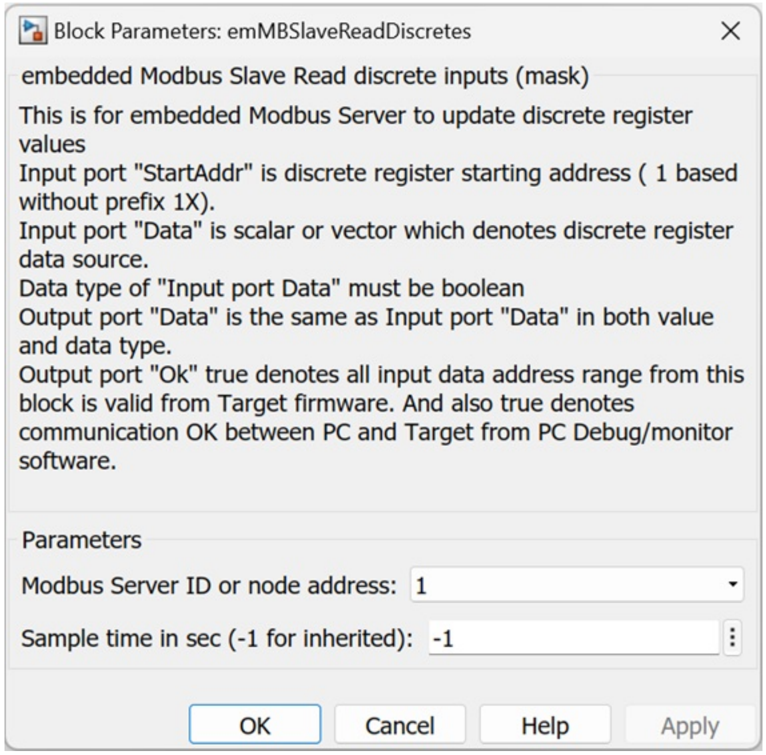


In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



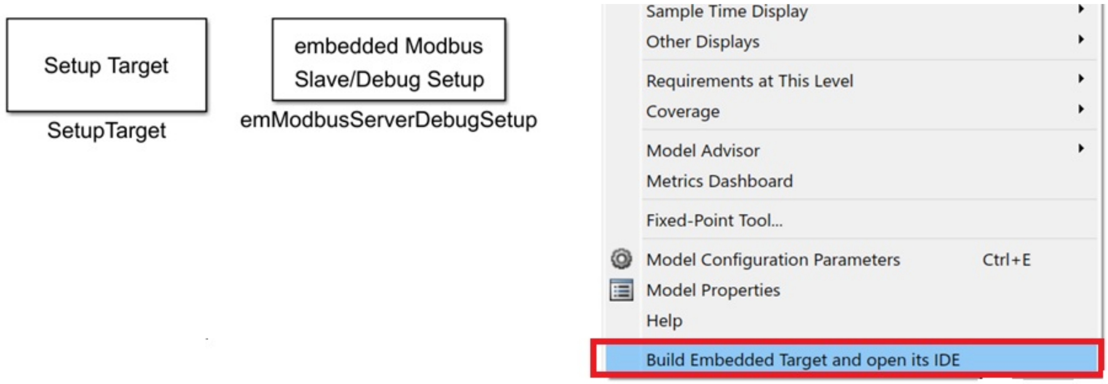
In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

Double click "emMBSlaveReadDiscretes", we will see the "emMBSlaveReadDiscretes" parameters settings below:

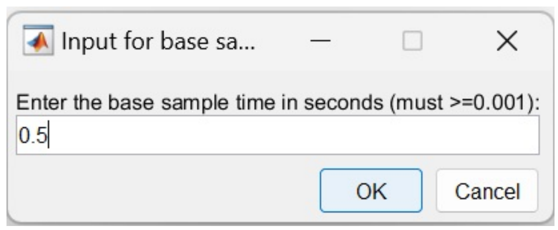


It means this block will update discrete register from Modbus server ID=1. This block's in-port "StartAddr" connects Constant block with value= 8 which denotes discrete register starting address is 10008. This block's in-port "Data" connects "Constant" block output, which provide boolean vector [1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 1] to in-port "Data". So we will see Modbus server (ID=1) discrete register 10008 to 10010 with value "true", discrete register 10011 to 10013 with value "false", ... , discrete register 10020 to 10024 with value "true".
Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too). We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

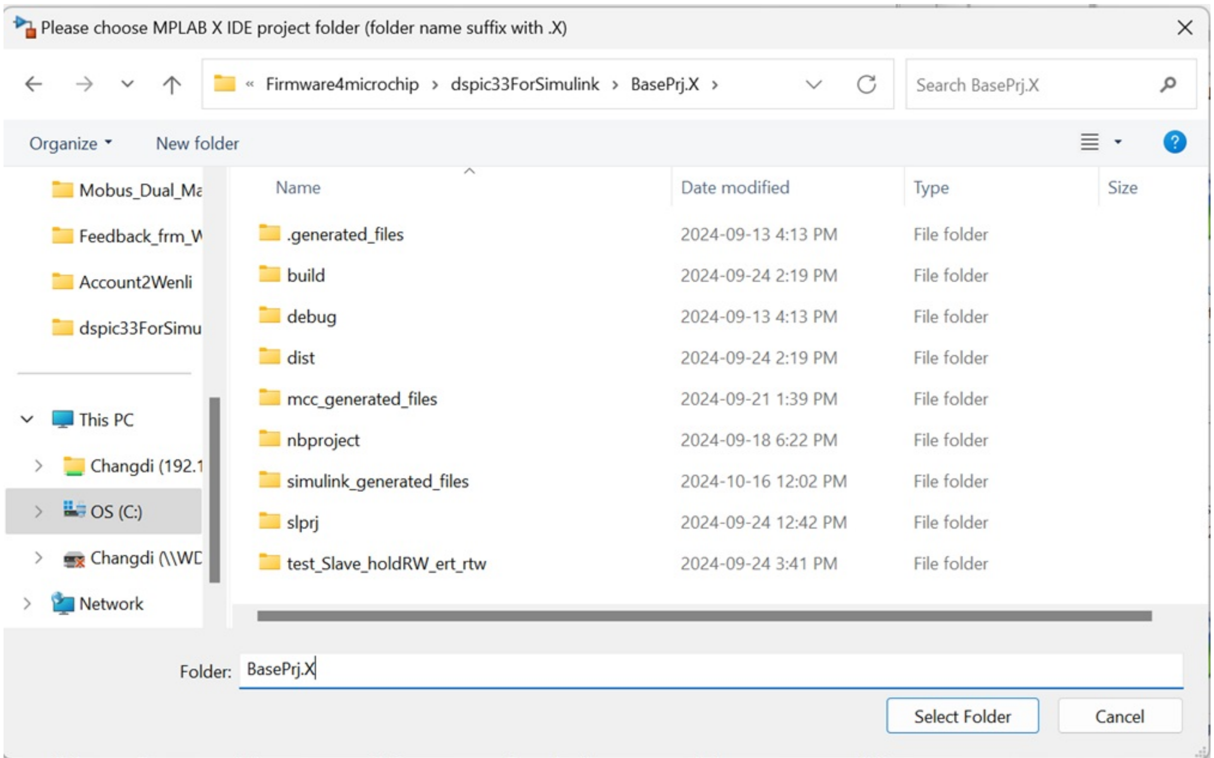
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:

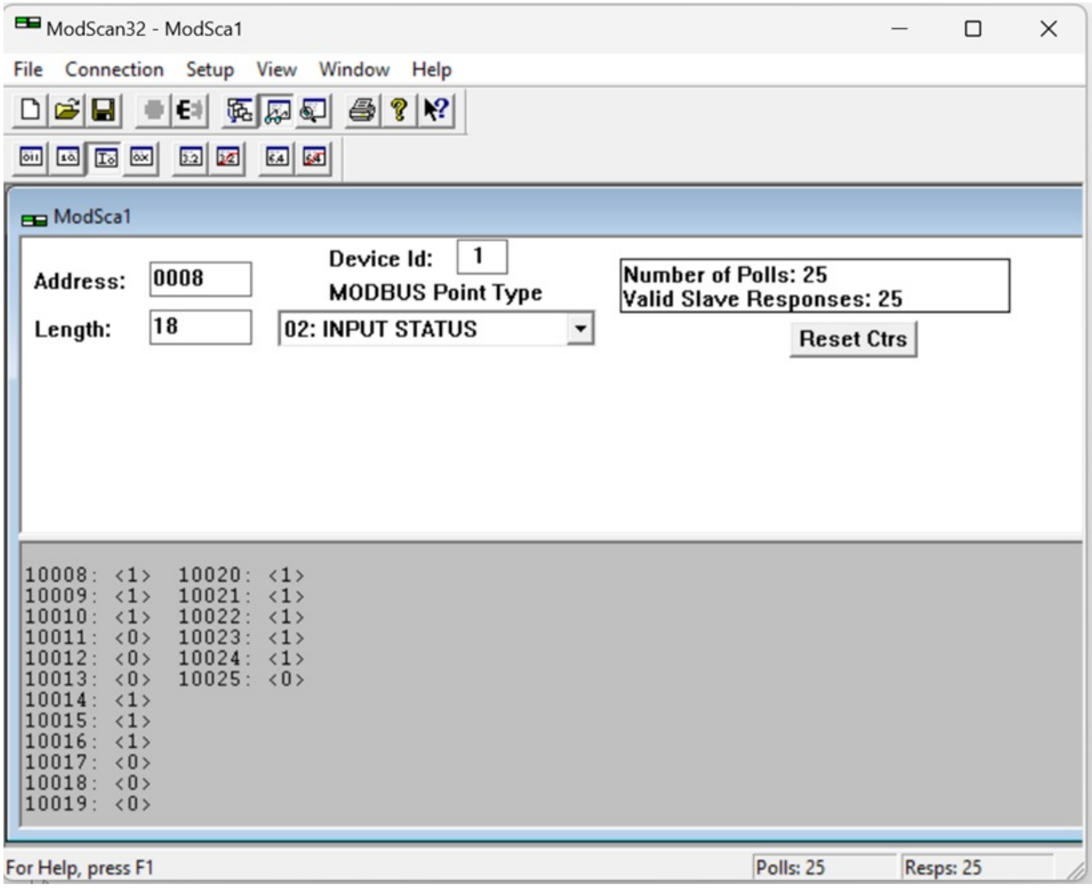


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



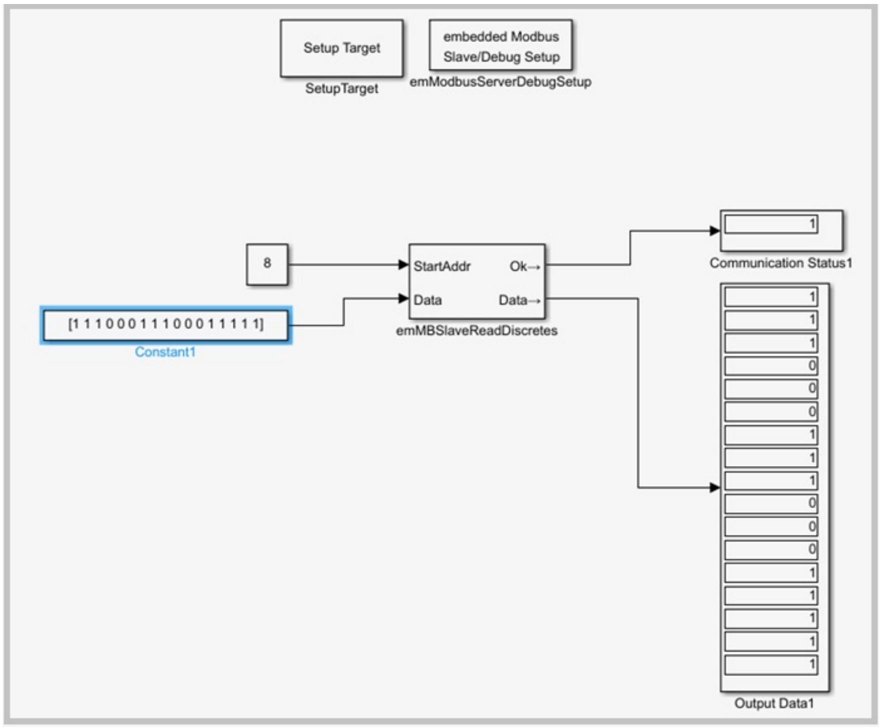
If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

If your firmware program into target successfully, you can use ModScan32 software to view result below:



All results are what we expect.

Disconnect ModScan32 software, we can use Simulink directly view results. By select stop time= inf, and click "Run" button, you will see result below:



You will see all results in all "Display" Blocks.

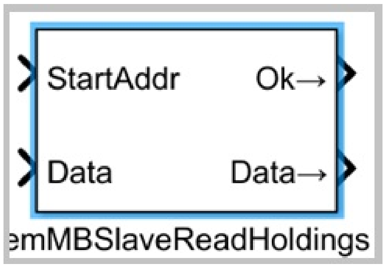
> ***Notes**: You can run both ".*ModScan32 " and "Simulink" at the same time. But you must enable "Modbus RTU/ASCII Dual Masters adaptor" Bluetooth, .and "ModScan32" cannot use the same COM port as Simulink. We recommend to use Simulink instead of *".*ModScan32 " because *"* ModScan32 " cannot watch general variables. Simulink can watch any variables by "Probe" (also called emProbe) blocks.

Please open "Your embedded creator library folder"/examples/example3_emReadDiscretes.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emModbusSlaveReadHoldings

embedded Modbus Server updates holding register value.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / embedded Modbus Slave & Debugger / emModbusSlaveReadHoldings



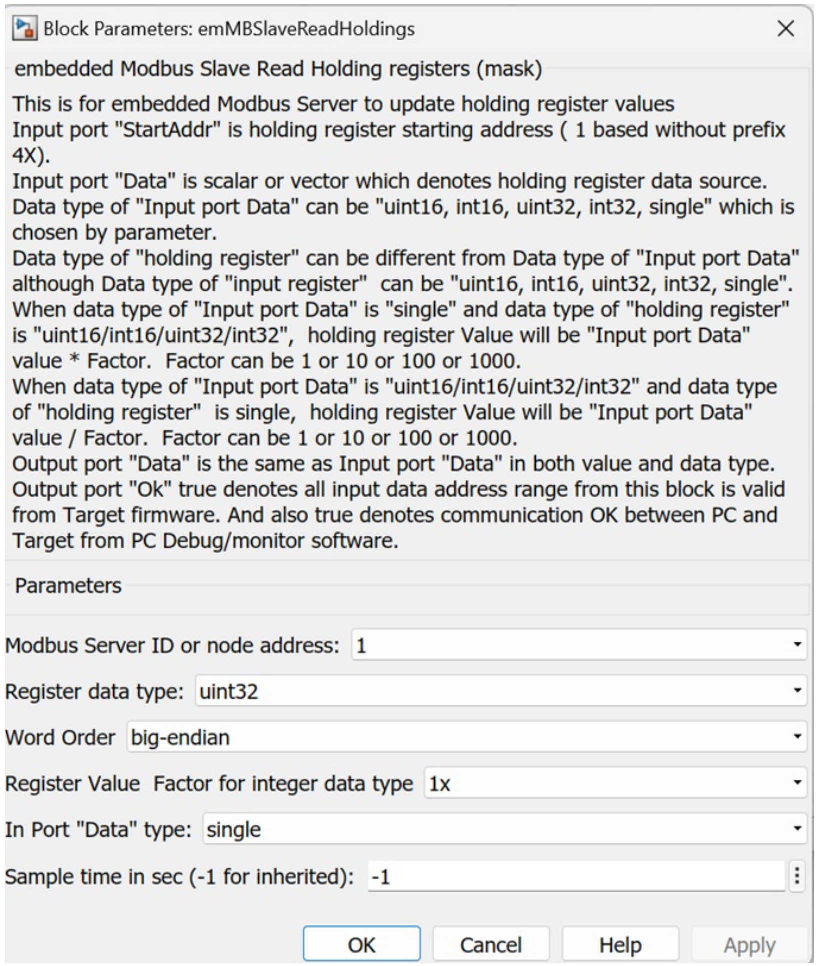
Description

This block will update my Modbus server holding register value periodically. So External Modbus client can get the latest holding register value.

> ***Notes****: Any Port name with "Right Arrow" (→) symbol contains "built-in" probe. So in PC side, you can watch its value directly by "Display" block, you don't need add "Probe" block (our emProbe) before "Display" block. Of cause, you can still use "Mux" block to collect all watching variables and then connect one "emProbe" which connects "Display" block. In this way, you can decrease communication traffic.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Modbus Server ID or node address — tell system this block is for which Modbus Server node. You just choose from drop list which is from "emModbusSlaveDebugSetup" block.

- Register data type — It is Modbus holding register data type, It can be "uint16, int16, uint32, int32 and single (4 bytes float)". When data type's byte QTY is over 2 bytes, it will have word order endian parameter.

- Word Order — It is visible when Register data type is "uint32, int32, single". It tells system words order : Big-endian or Little-endian.

- Value Factor for integer data type when reg and In port "Data" have different type — It can be "1x, 10x, 100x, and 1000x". We will explain meaning in parameter "In-Port Data type:" below.

- In-Port "Data" type: — It can be "uint16, int16, uint32, int32 and single (4 bytes float)". The embedded firmware translates input port "Data" to Modbus holding register data. When input port "Data" is data type of "single" (4 bytes float) and Modbus holding register data type is "uint16 or int16 or uint32 or int32", Modbus holding register data value will be input port "Data" value times "Value factor" and then round it to integer. For example, input port "Data"=[2.35 5.8], holding register data value will be [24 58] if Value Factor is 10x.

Similarly, When input port "Data" is data type of "uint16 or int16 or uint32 or int32" and Modbus holding register data type is "single", Modbus holding register data value will be input port "Data" value ÷ "Value factor" . For example, input port "Data"=[278 -567], holding register data value will be [2.78 -5.67] if Value Factor is 100x.

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is Modbus Server holding registers' start address (1-based without prefix "4X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.

- Data — Scalar or Vector. Data type is decided by parameter "In-Port 'Data' type". It is equal to input port Data.

Outport

- Ok — "logical" data type's scalar. In PC side, it denotes whether communication between PC and Target is OK. In embedded target side, it denotes whether holding register range you read is valid.

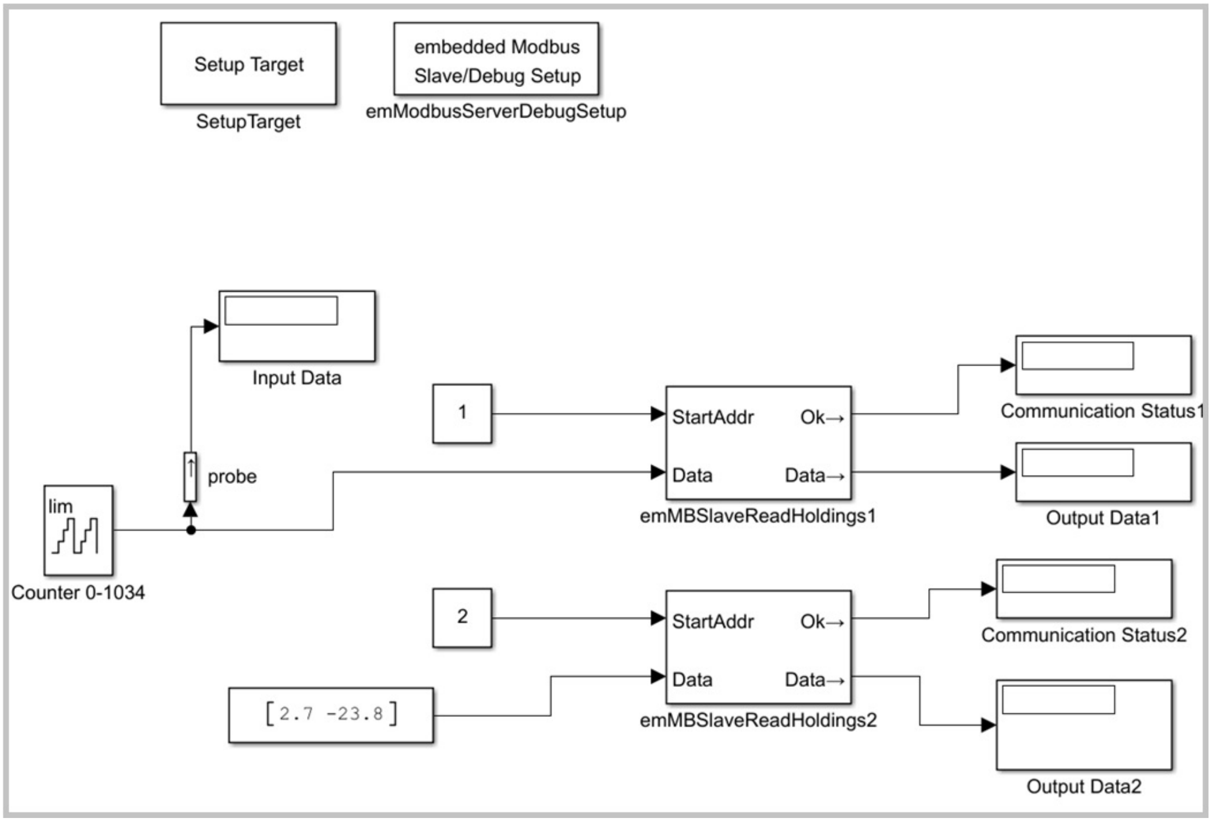
- Data — Scalar or Vector. Data type is decided by parameter "In-Port 'Data' type". It is holding register data source.

Examples

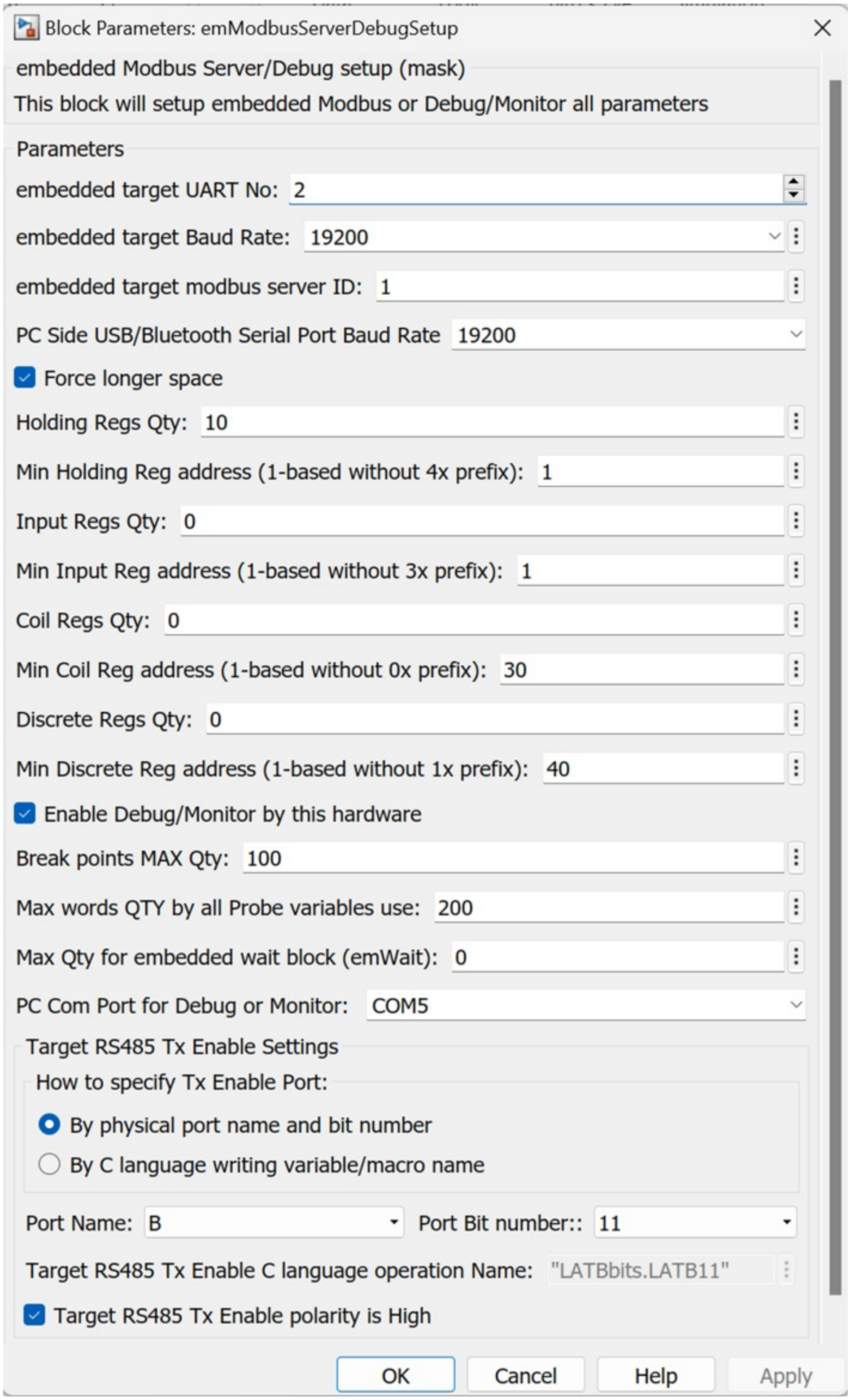
Example1:

Our embedded platform is dsPIC33EP256GP502 . Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. Our embedded Modbus Server ID=1, and supports holding registers address range: 40001 to 40010, baud rate=19200

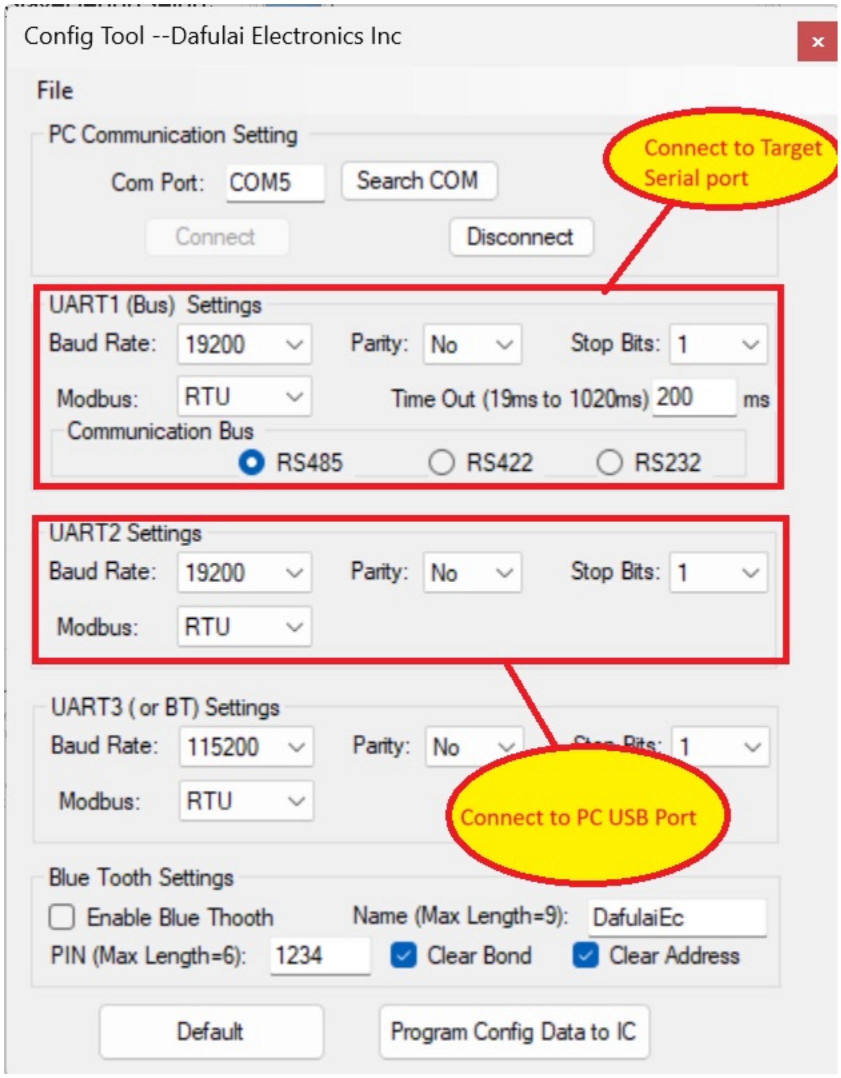
Please see screenshot of model below:



In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Double click "emModbusServerDebugSetup", we will see the Modbus Server/Debug parameters settings below:

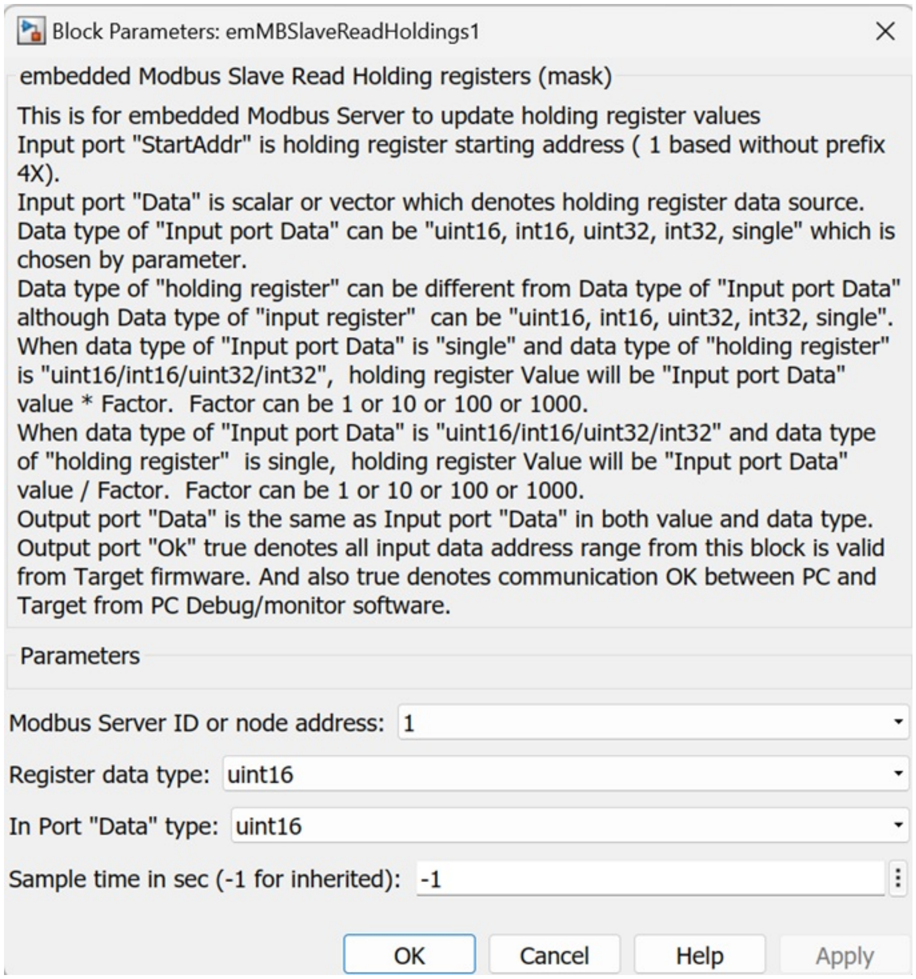


In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



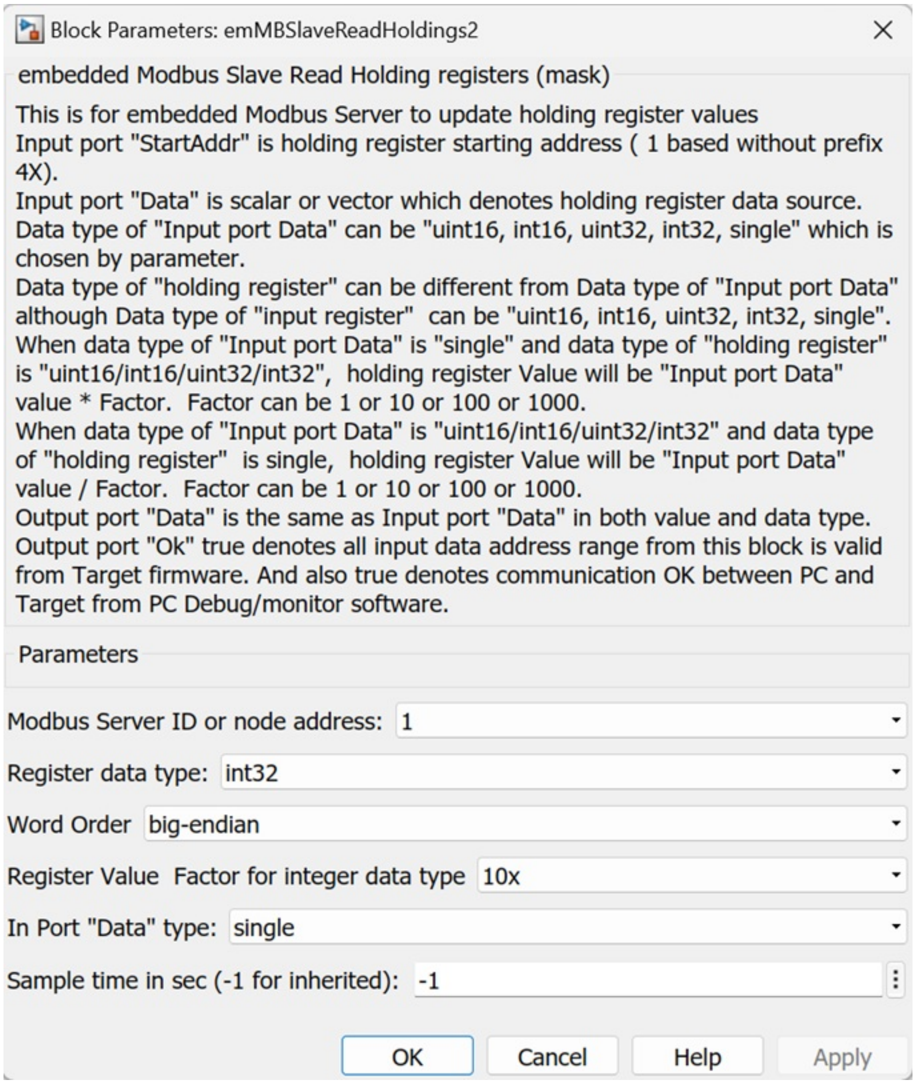
In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

Double click "emMBSlaveReadHoldings1", we will see the "emMBSlaveReadHoldings1" parameters settings below:



It means this block will update holding register from Modbus server ID=1 and holding register data type is "16 bits of unsigned integer". This block's in-port "StartAddr" connects Constant block with value= 1 which denotes holding register starting address is 40001. This block's in-port "Data" connects "Counter Limited" block output, which provide 0 to 1034 increasing counter output value (uint16 data type) to in-port "Data". So we will see Modbus server (ID=1) holding register 40001 increase 1 from 0 to 1034 every sampling period.

Double click "emMBSlaveReadHoldings2", we will see the "emMBSlaveReadHoldings2" parameters settings below:

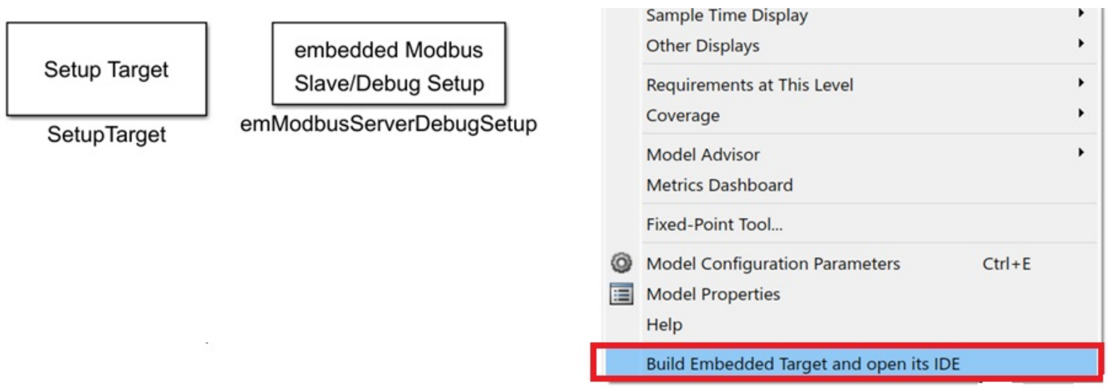


It means this block will update holding register from Modbus server ID=1 and Input register data type is "32 bits of signed integer" (big-endian). This block's in-port "StartAddr" connects Constant block with value= 2 which denotes holding register starting address is 40002. This block's in-port "Data" connects "Constant" block output, which provide vector [2.7 -23.8] to in-port "Data". So we will see Modbus server (ID=1) holding register 40002 value= 0 and holding register 40003=27 (big-endian, and factor =10, so 2.7x10 =27) . We also see holding register 40004 value= -1 and holding register 40005=-238 (big-endian, and factor =10, so -23.8 x10 = -238. The first word is -1 not 0 due to 2's complement reason)

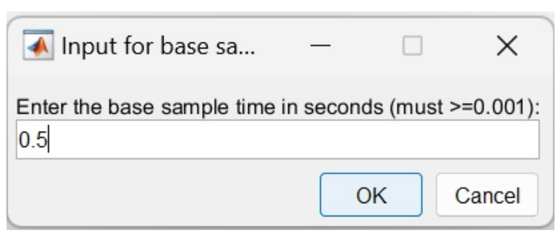
"Probe" block is just for watching "Counter Limited" block output. For any block output except "Constant block", if its name does not contain "Right Arrow" *(→)*, you must use "Probe" block to watch its value.

Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too). We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

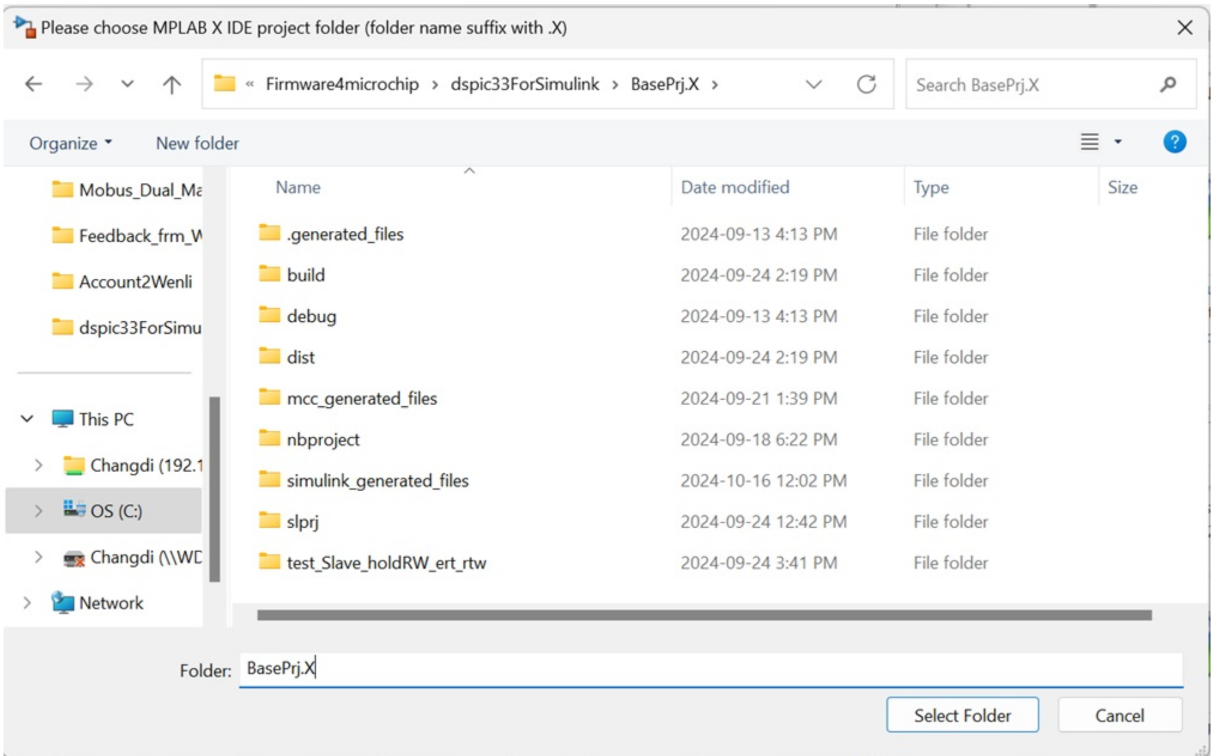
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:

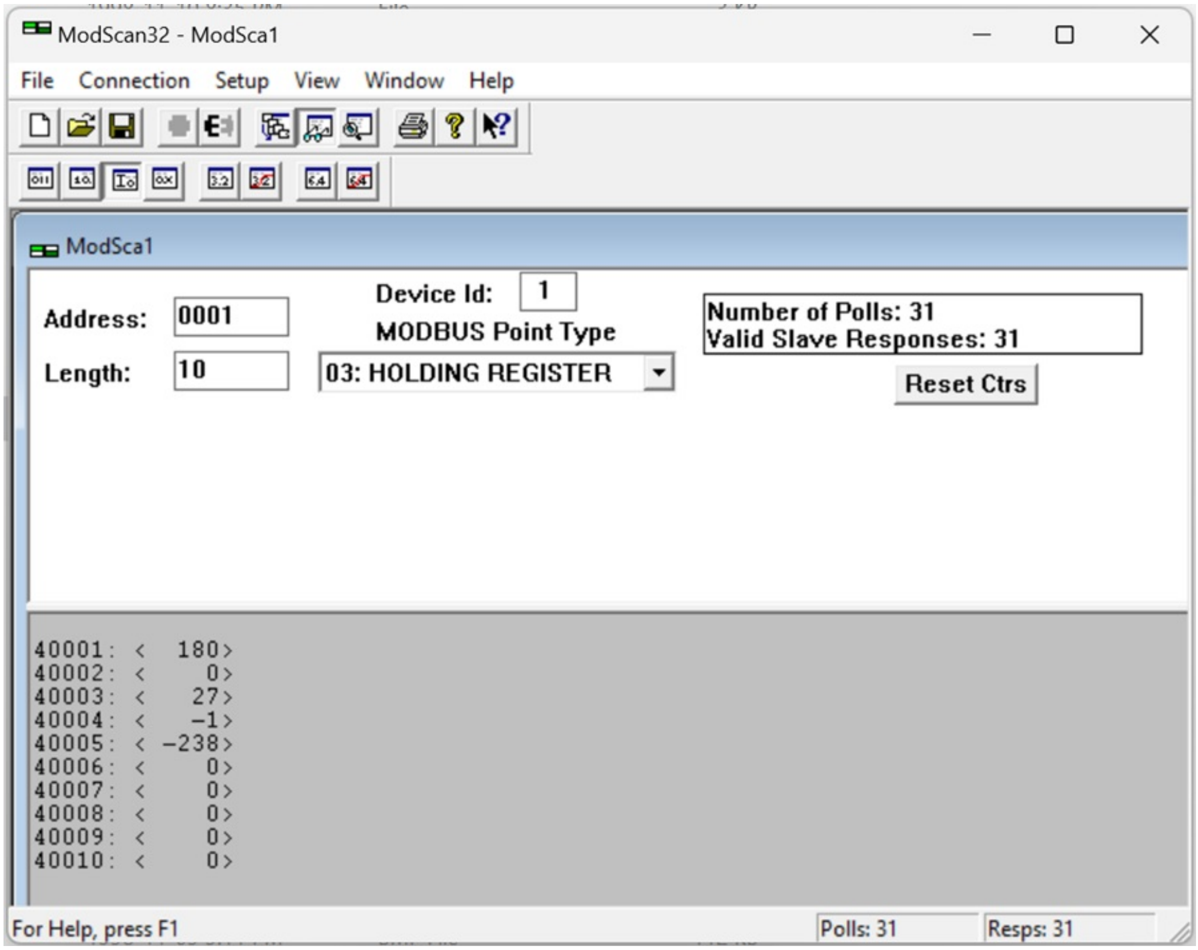


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



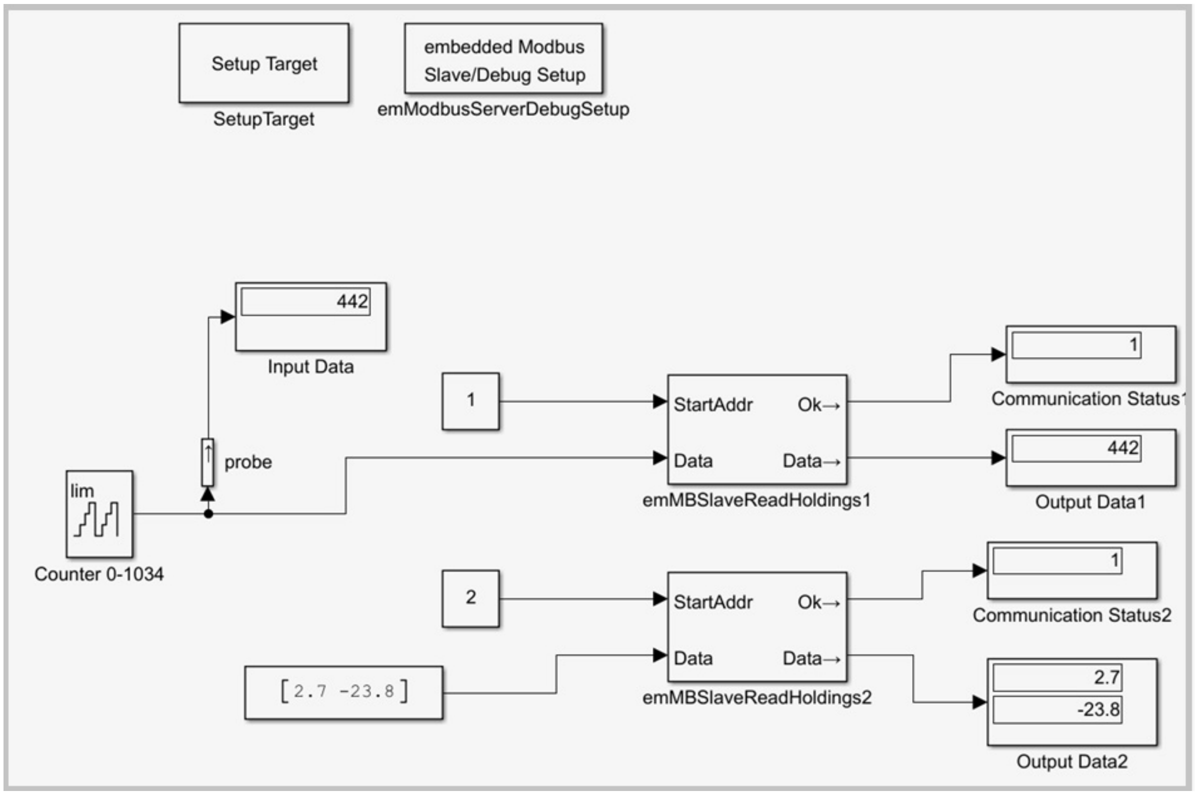
If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

If your firmware program into target successfully, you can use ModScan32 software to view result below:



We can see address 40001 value changing from 0 to 1034 every 0.5 sec, and 40002 is 0, 40003 is 27, 40004 is -1 , 40005 is -238. All results are what we expect.

Disconnect ModScan32 software, we can use Simulink directly view results. By select stop time= inf, and click "Run" button, you will see result below:



You will see all results in all "Display" Blocks.

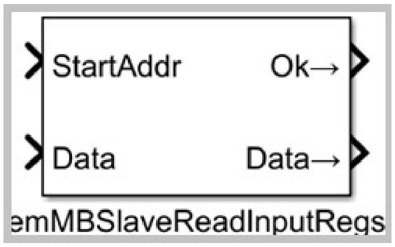
> ***Notes**: You can run both ".*ModScan32 " and "Simulink" at the same time. But you must enable "Modbus RTU/ASCII Dual Masters adaptor" Bluetooth, .and "ModScan32" cannot use the same COM port as Simulink. We recommend to use Simulink instead of ".ModScan32 " because " ModScan32 " cannot watch general variables. Simulink can watch any variables by "Probe" (also called emProbe) blocks.

Please open "Your embedded creator library folder"/examples/example2_emReadHoldings.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emModbusSlaveReadInputRegs

embedded Modbus Server updates input register value.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Modbus Slave & Debugger / emModbusSlaveReadInputRegs



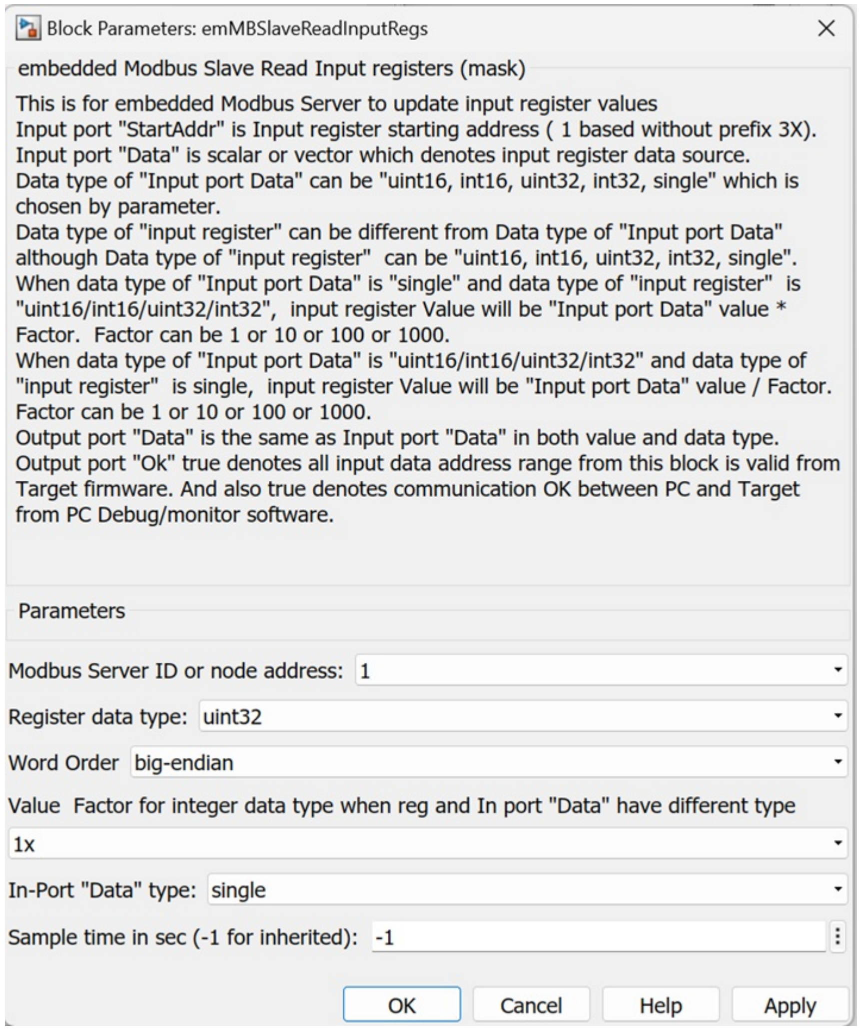
Description

This block will update my Modbus server input register value periodically. So External Modbus client can get the latest input register value.

> ***Notes**: Any Port name with "Right Arrow" (→) symbol contains "built-in" probe. So in PC side, you can watch its value directly by "Display" block, you don't need add "Probe" block (our emProbe) before "Display" block. Of cause, you can still use "Mux" block to collect all watching variables and then connect one "emProbe" which connects "Display" block. In this way, you can decrease communication traffic.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Modbus Server ID or node address — tell system this block is for which Modbus Server node. You just choose from drop list which is from "emModbusSlaveDebugSetup" block.
- Register data type — It is Modbus input register data type, It can be "uint16, int16, uint32, int32 and single (4 bytes float)". When data type's byte QTY is over 2 bytes, it will have word order endian parameter.
- Word Order — It is visible when Register data type is "uint32, int32, single". It tells system words order : Big-endian or Little-endian.
- Value Factor for integer data type when reg and In port "Data" have different type — It can be "1x, 10x, 100x, and 1000x". We will explain meaning in parameter "In-Port Data type:" below.
- In-Port "Data" type: — It can be "uint16, int16, uint32, int32 and single (4 bytes float)". The

embedded firmware translates input port "Data" to Modbus input register data. When input port "Data" is data type of "single" (4 bytes float) and Modbus input register data type is "uint16 or int16 or uint32 or int32", Modbus input register data value will be input port "Data" value times "Value factor" and then round it to integer. For example, input port "Data"=[2.35 5.8], input register data value will be [24 58] if Value Factor is 10x.

Similarly, When input port "Data" is data type of "uint16 or int16 or uint32 or int32" and Modbus input register data type is "single", Modbus input register data value will be input port "Data" value ÷ "Value factor" . For example, input port "Data"=[278 -567], input register data value will be [2.78 -5.67] if Value Factor is 100x.

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is Modbus Server input registers' start address (1-based without prefix "3X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.

- Data — Scalar or Vector. Data type is decided by parameter "In-Port 'Data' type". It is Input register data source.

Outport

- Ok — "logical" data type's scalar. In PC side, it denotes whether communication between PC and Target is OK. In embedded target side, it denotes whether input register range you read is valid.

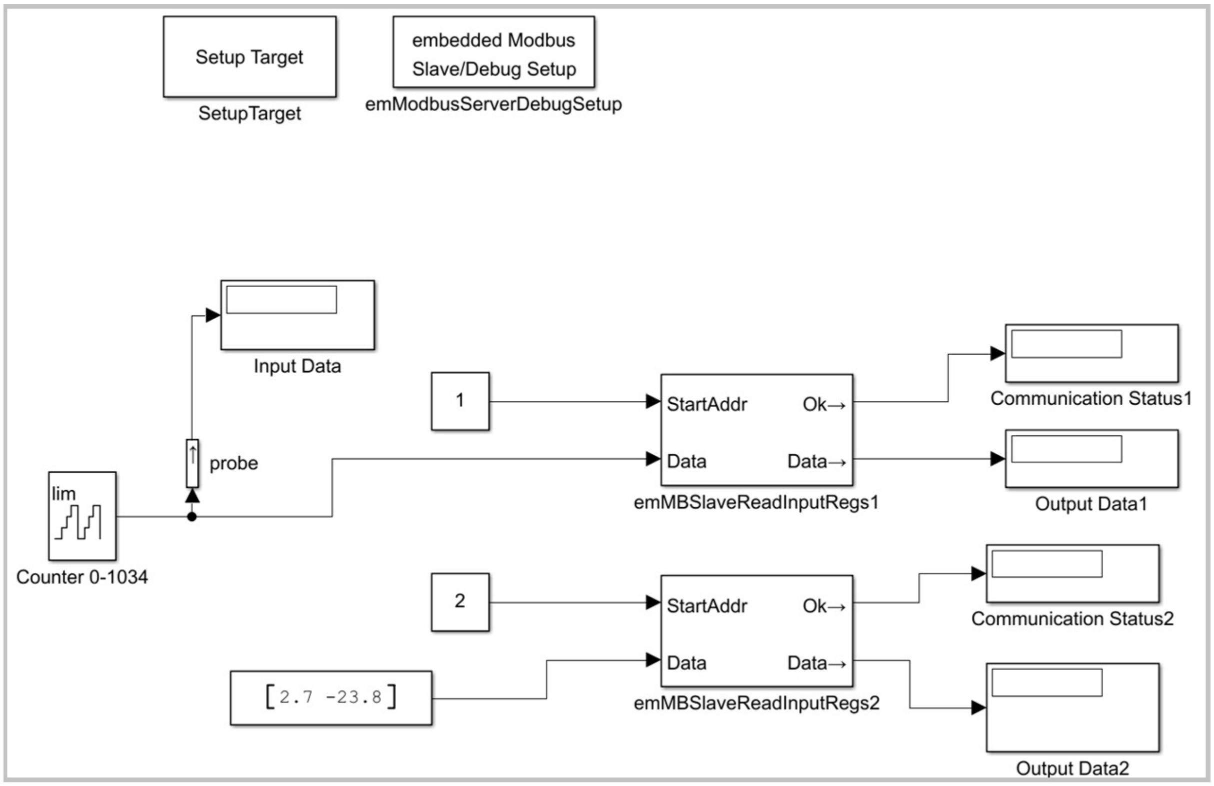
- Data — Scalar or Vector. Data type is decided by parameter "In-Port 'Data' type". It is equal to input port Data.

Examples

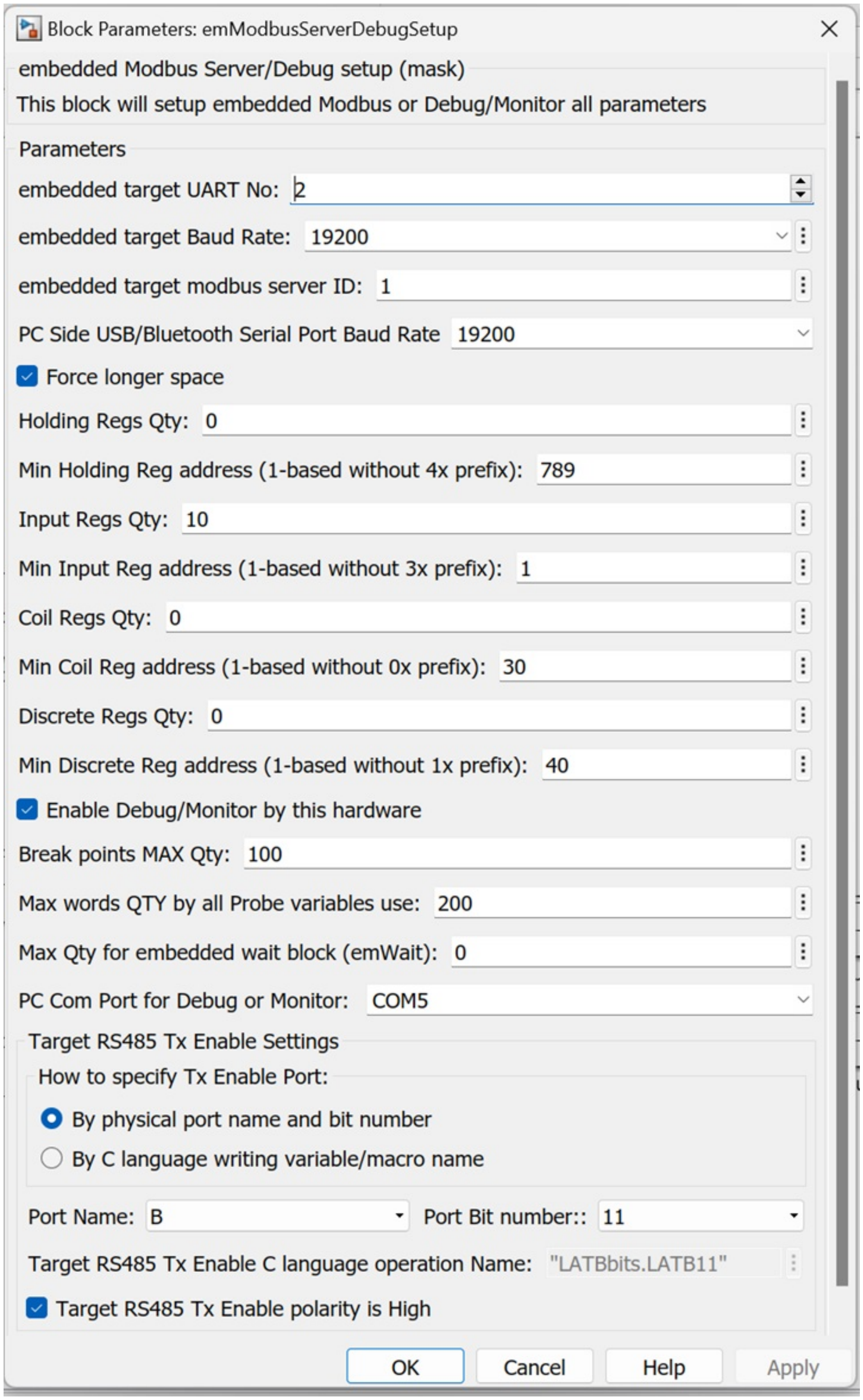
Example1:

Our embedded platform is dsPIC33EP256GP502 . Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. Our embedded Modbus Server ID=1, and supports input registers address range: 30001 to 30010, baud rate=19200

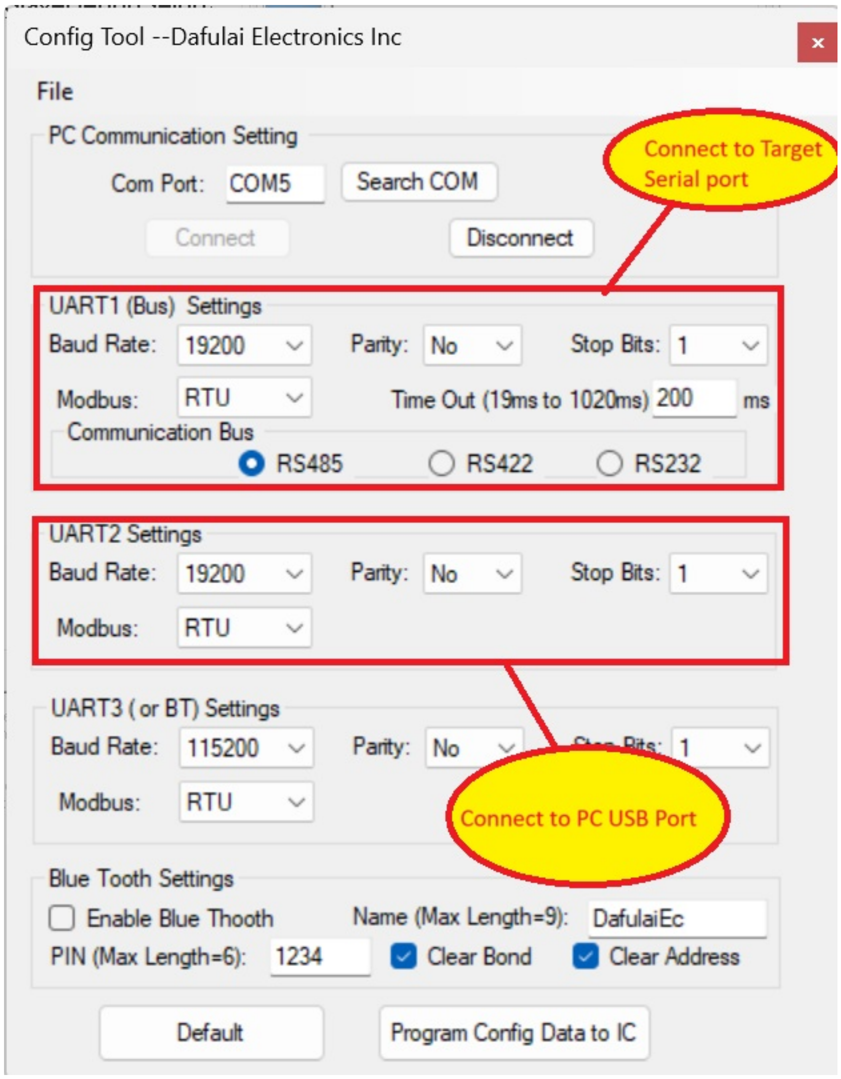
Please see screenshot of model below:



In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Double click "emModbusServerDebugSetup", we will see the Modbus Server/Debug parameters settings below:

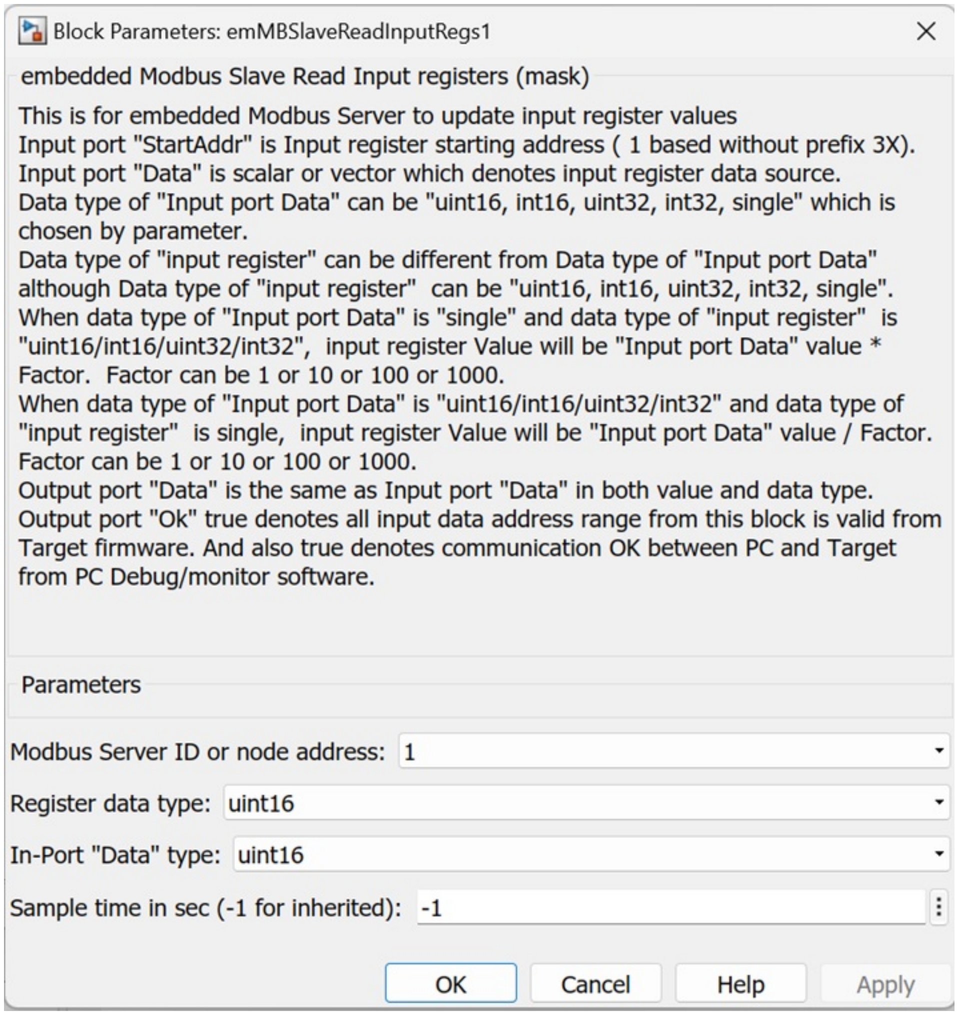


In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



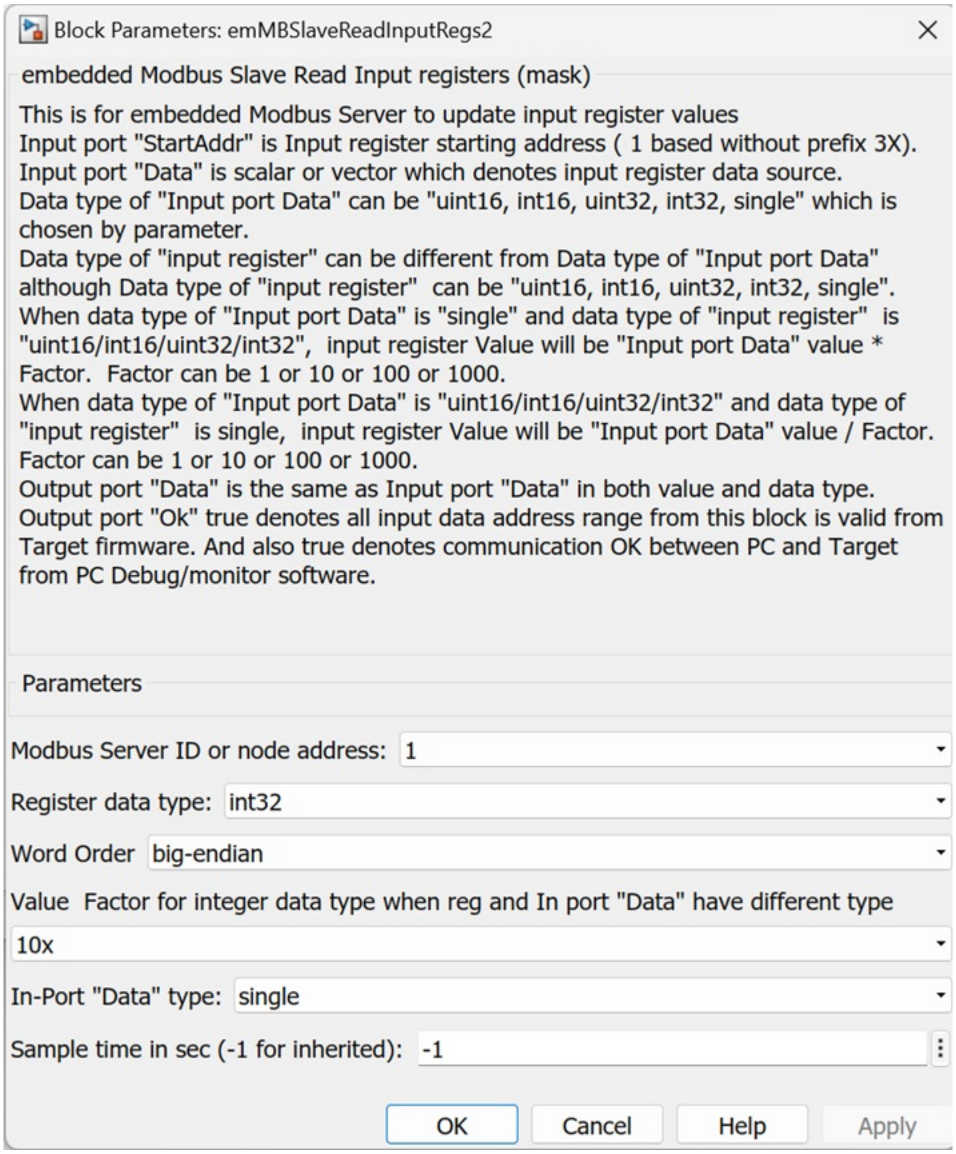
In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

Double click "emMBSlaveReadInputRegs1", we will see the "emMBSlaveReadInputRegs1" parameters settings below:



It means this block will update Input register from Modbus server ID=1 and Input register data type is "16 bits of unsigned integer". This block's in-port "StartAddr" connects Constant block with value= 1 which denotes Input register starting address is 30001. This block's in-port "Data" connects "Counter Limited" block output, which provide 0 to 1034 increasing counter output value (uint16 data type) to in-port "Data". So we will see Modbus server (ID=1) input register 30001 increase 1 from 0 to 1034 every sampling period.

Double click "emMBSlaveReadInputRegs2", we will see the "emMBSlaveReadInputRegs2" parameters settings below:

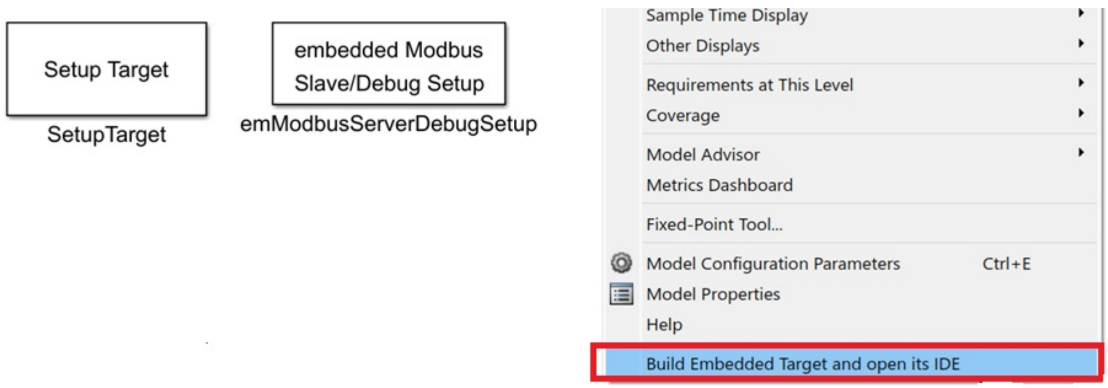


It means this block will update Input register from Modbus server ID=1 and Input register data type is "32 bits of signed integer" (big-endian). This block's in-port "StartAddr" connects Constant block with value= 2 which denotes Input register starting address is 30002. This block's in-port "Data" connects "Constant" block output, which provide vector [2.7 -23.8] to in-port "Data". So we will see Modbus server (ID=1) input register 30002 value= 0 and input register 30003=27 (big-endian, and factor =10, so 2.7x10 =27) . We also see input register 30004 value= -1 and input register 30005=-238 (big-endian, and factor =10, so -23.8 x10 = -238. The first word is -1 not 0 due to 2's complement reason)

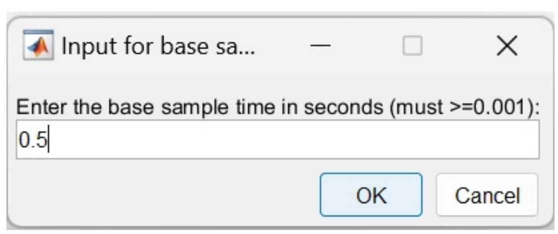
"Probe" block is just for watching "Counter Limited" block output. For any block output except "Constant block", if its name does not contain "Right Arrow" *(→)*, you must use "Probe" block to watch its value.

Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too). We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

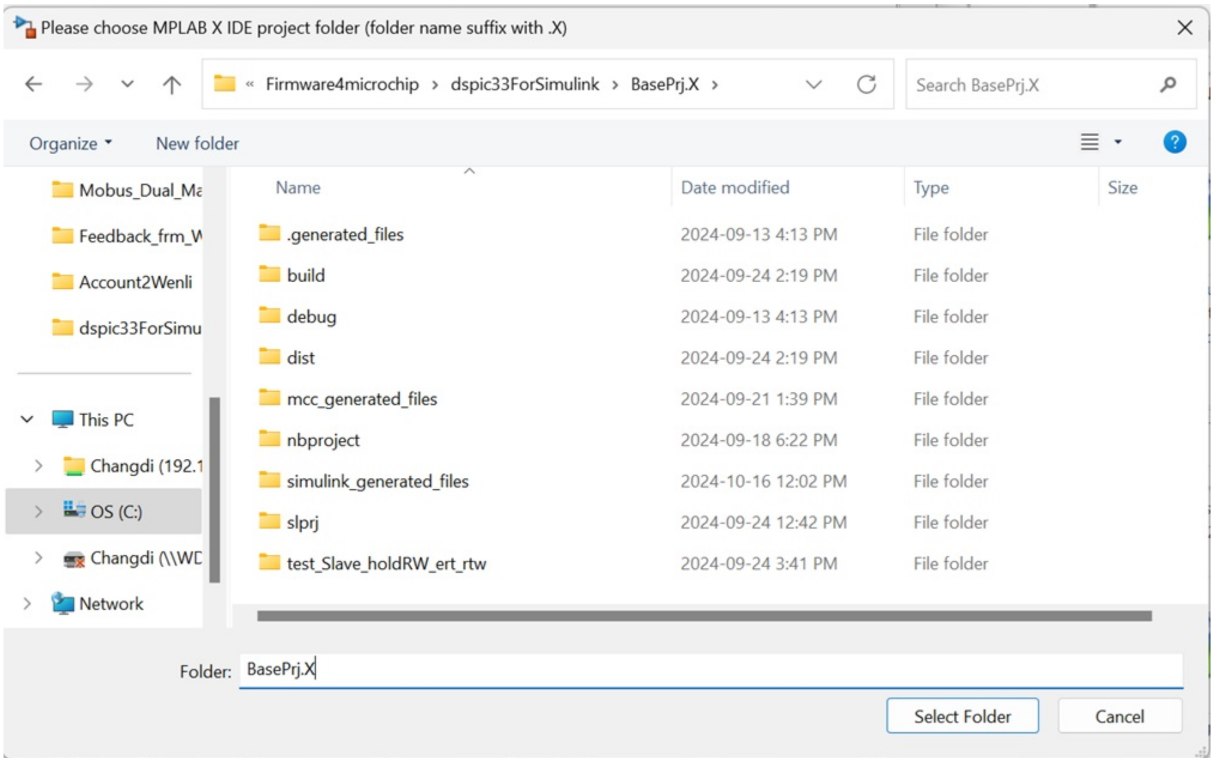
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:

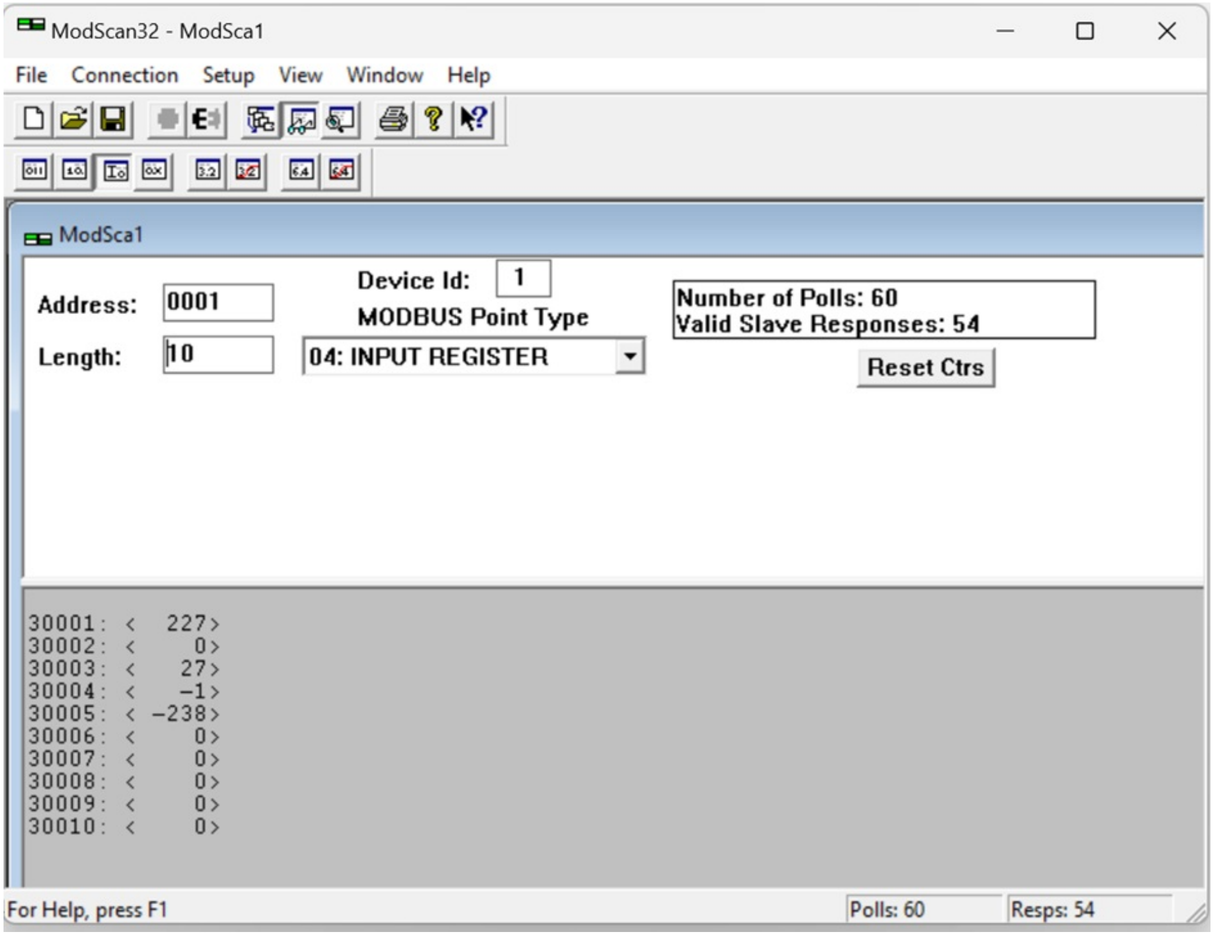


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



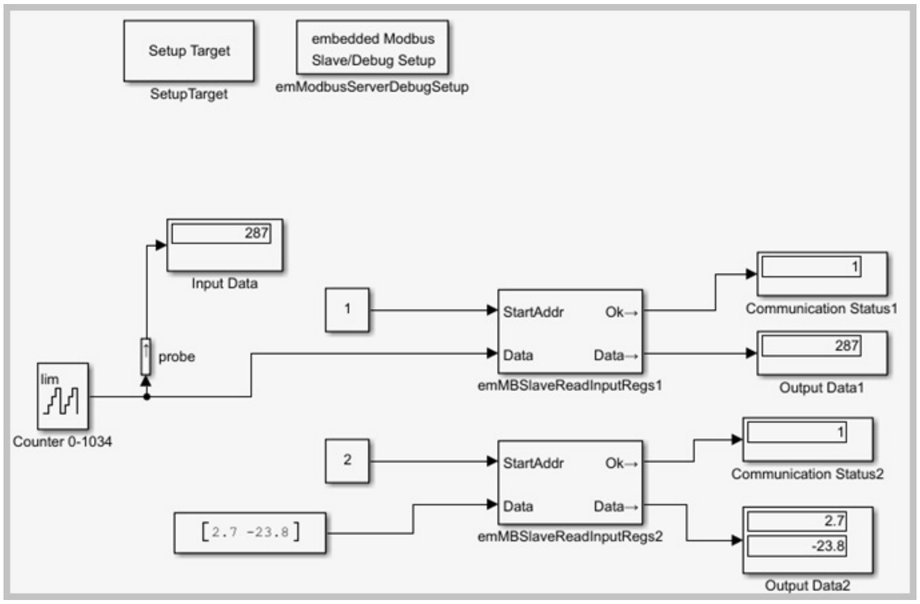
If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

If your firmware program into target successfully, you can use ModScan32 software to view result below:



We can see address 30001 value changing from 0 to 1034 every 0.5 sec, and 30002 is 0, 30003 is 27, 30004 is -1 , 30005 is -238. All results are what we expect.

Disconnect ModScan32 software, we can use Simulink directly view results. By select stop time= inf, and click "Run" button, you will see result below:



You will see all results in all "Display" Blocks.

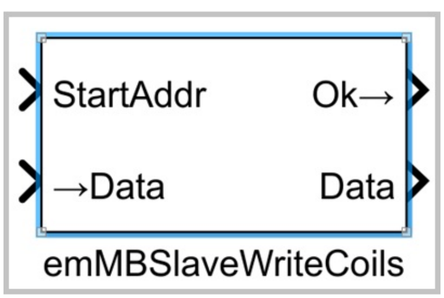
> ***Notes****: You can run both ".*ModScan32 " and "Simulink" at the same time. But you must enable "Modbus RTU/ASCII Dual Masters adaptor" Bluetooth, .and "ModScan32" cannot use the same COM port as Simulink. We recommend to use Simulink instead of *".*ModScan32 " because *"* ModScan32 " cannot watch general variables. Simulink can watch any variables by "Probe" (also called emProbe) blocks.

Please open "Your embedded creator library folder"/examples/example1_emReadInputRegs.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emModbusSlaveWriteCoils

Embedded Modbus Server receives coil register writing value.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Modbus Slave & Debugger / emModbusSlaveWriteCoils



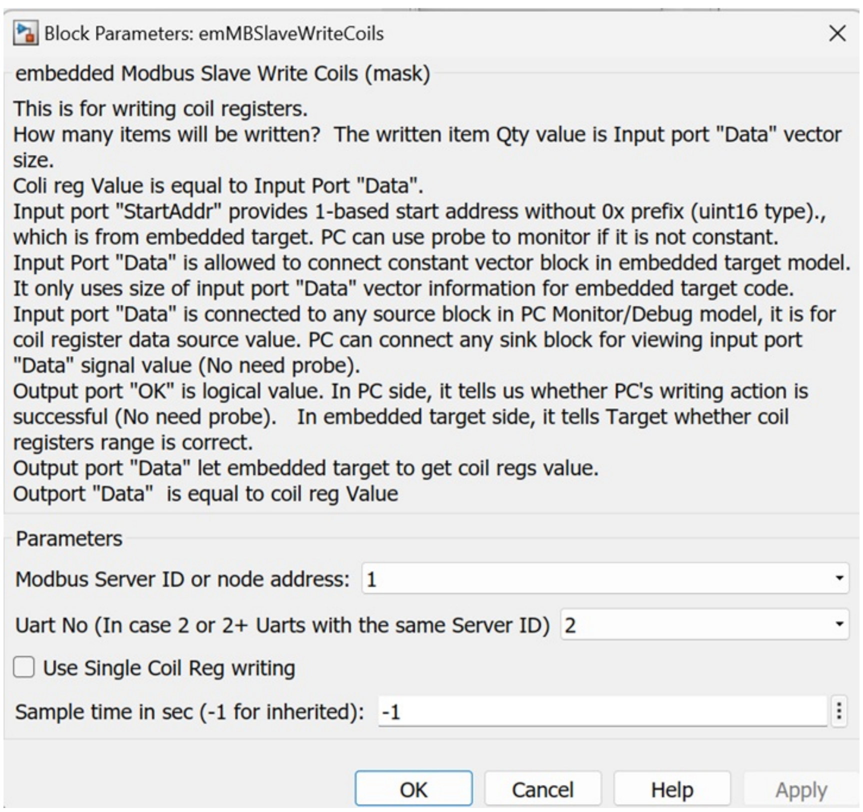
Description

For embedded target side, this block will receive my Modbus server coil register writing value and embedded target gets data only from this block's Output Data. Block's input port Data is only for PC side to send coil register data value.

> ***Notes**: Any Port name with "Right Arrow" (→) symbol contains "built-in" probe. So in PC side, you can watch its value directly by "Display" block, you don't need add "Probe" block (our emProbe) before "Display" block. Of cause, you can still use "Mux" block to collect all watching variables and then connect one "emProbe" which connects "Display" block. In this way, you can decrease communication traffic.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Modbus Server ID or node address — tell system this block is for which Modbus Server node. You just choose from drop list which is from "emModbusSlaveDebugSetup" block.

- Uart No (In case 2 or 2+ Uarts with the same Server ID) — It is Uart Number Modbus Server uses. This is only used in one Server ID for more different Uarts. In this situation, you cannot connect these 2 or more nodes into one Modbus network because node Sever ID must be unique in one Modbus network.

- Use Single Coil Reg writing — It will use FC=05 (Single writing coil reg) to replace FC=15 (Multiple writing coil registers).

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is Modbus Server coil registers' start address (1-based without prefix "0X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.

- Data — "logical" Scalar or Vector. We only care the size of vector in embedded target. It decides how many items will be written. And the value of this input port will be written to target coil registers from PC Debug/monitor.

Outport

- Ok — "logical" data type's scalar. In PC side, it denotes whether communication between PC and Target is OK. In embedded target side, it denotes whether coil register range you read is valid.

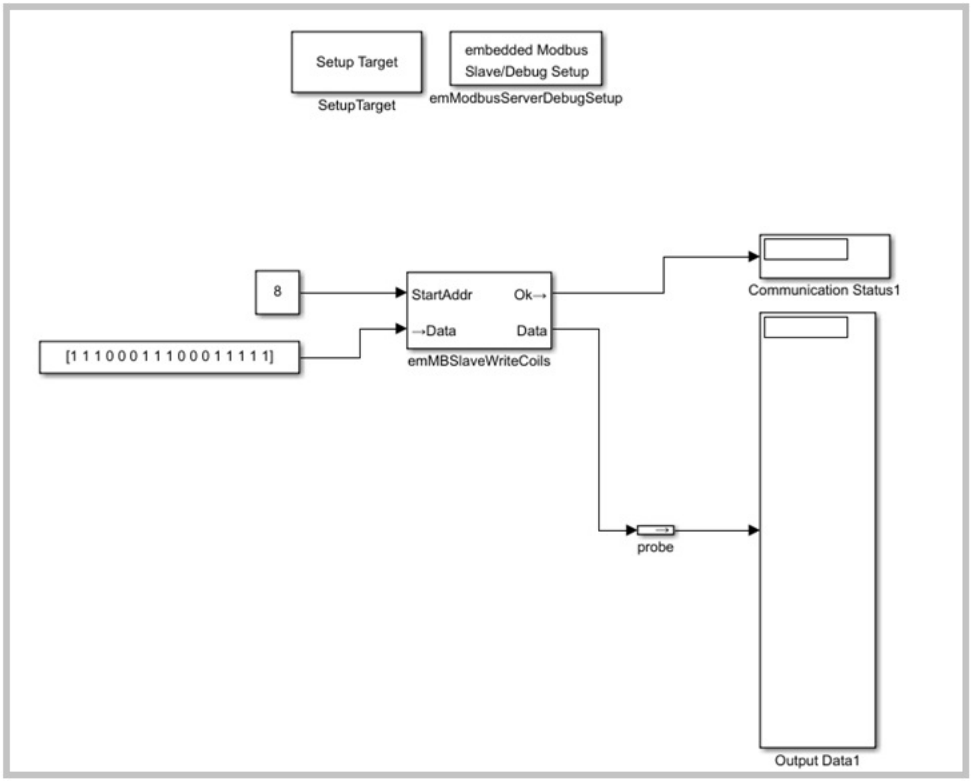
- Data —"logical" Scalar or Vector.   It is from coil register values written.

Examples

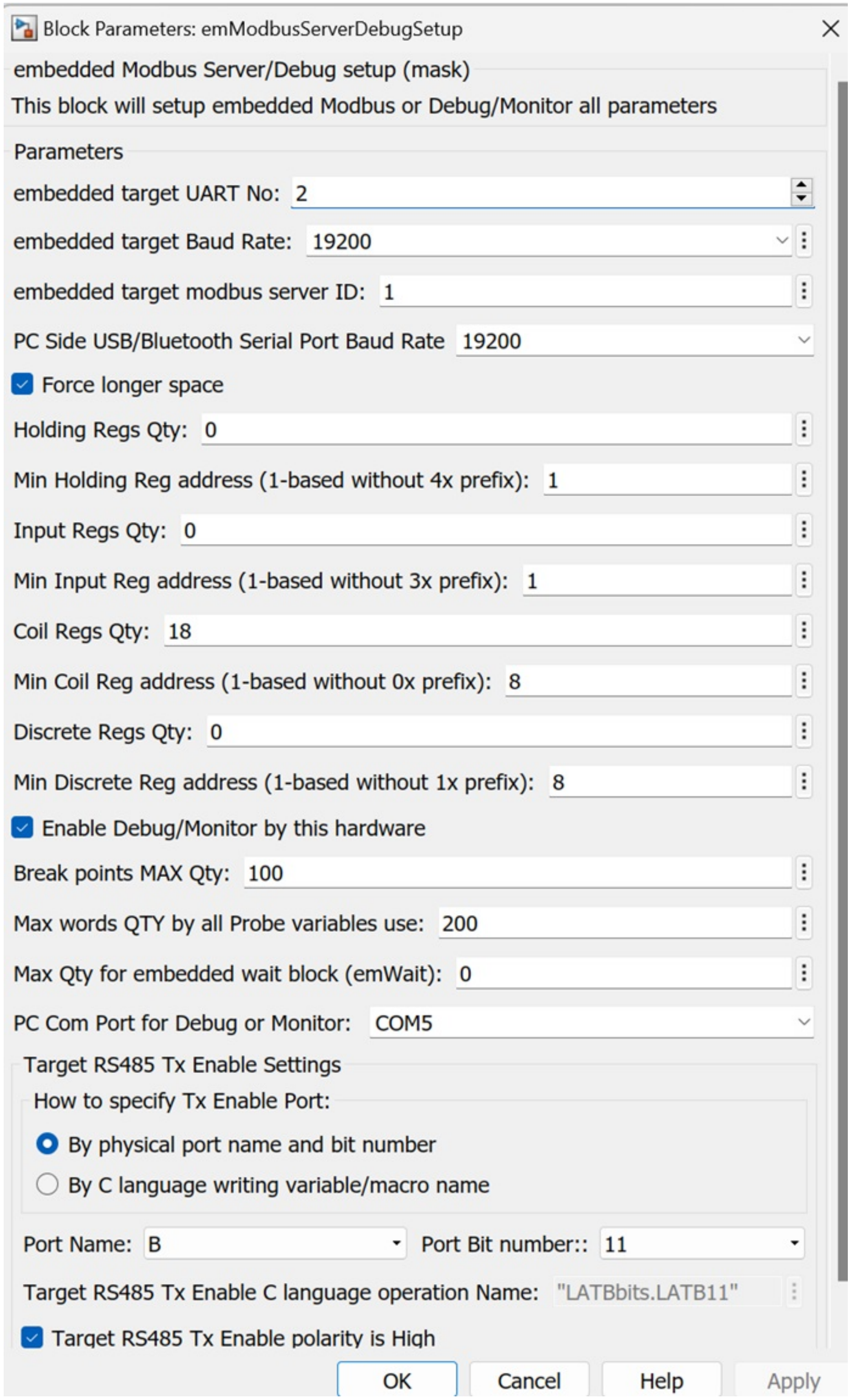
Example1:

Our embedded platform is dsPIC33EP256GP502 . Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver.   Our embedded Modbus Server ID=1, and supports coil registers address range: 00008 to 00025, baud rate=19200

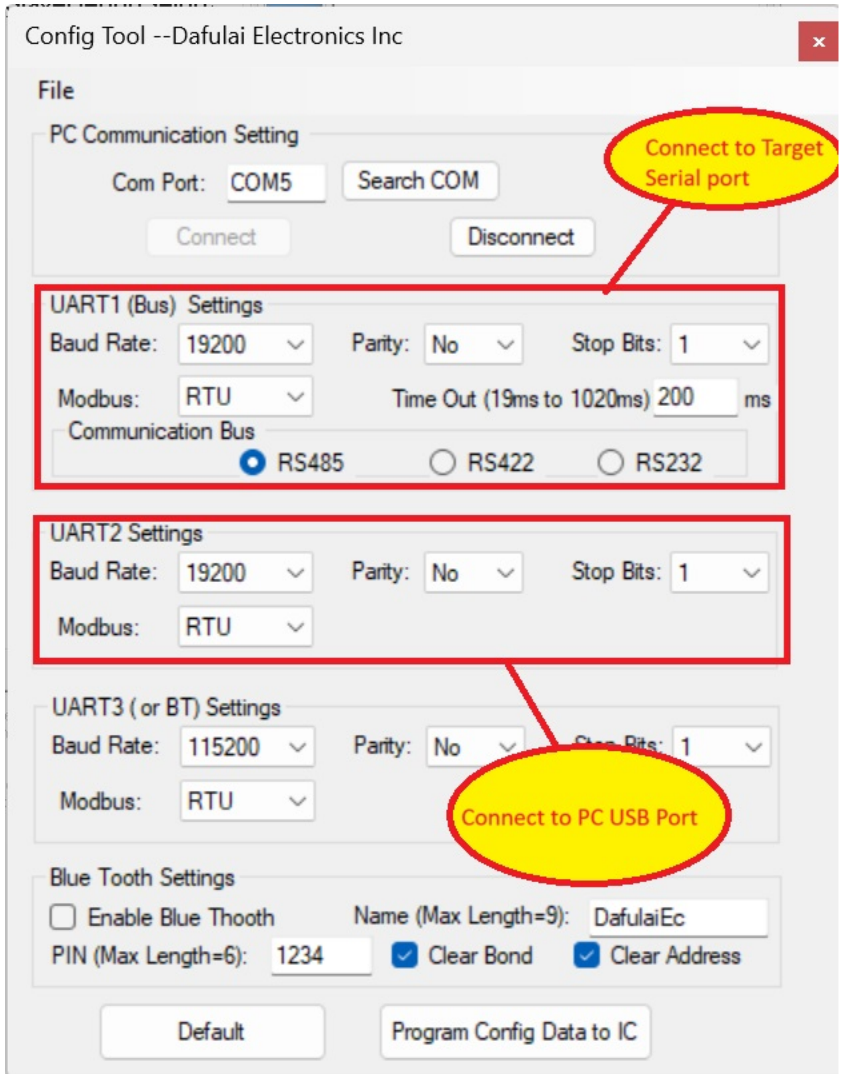
Please see screenshot of model below:



In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Double click "emModbusServerDebugSetup", we will see the Modbus Server/Debug parameters settings below:

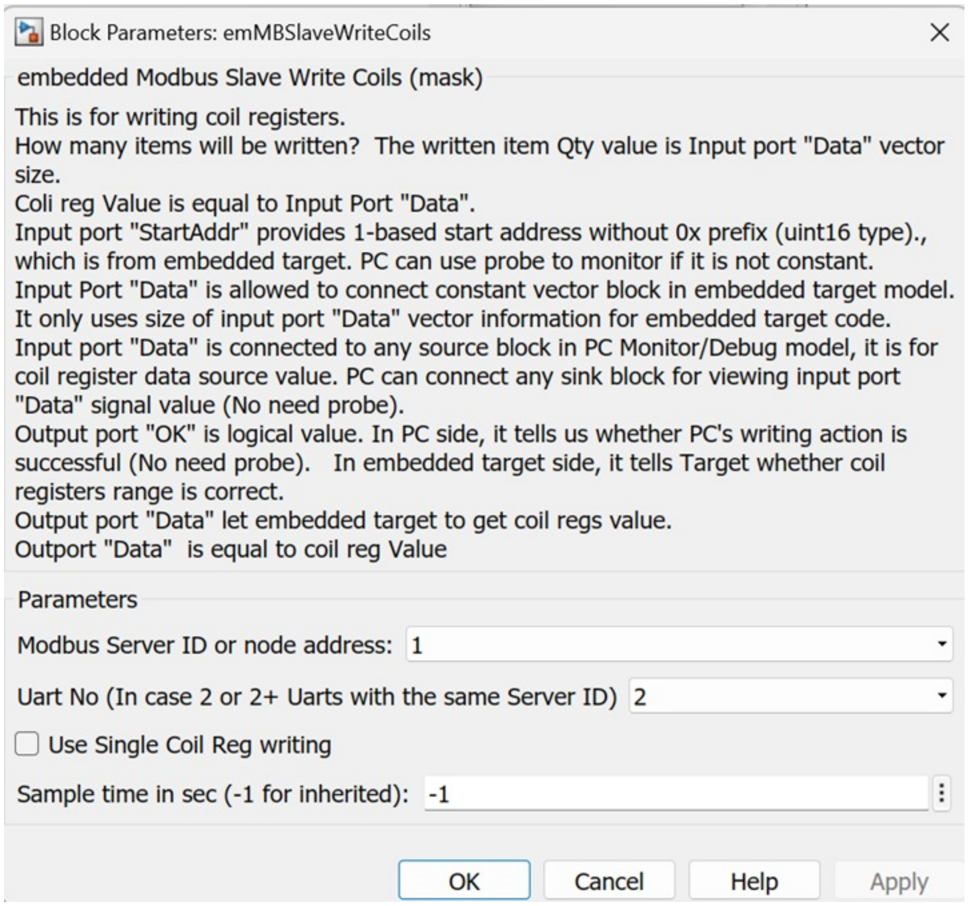


In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

Double click "emMBSlaveWriteCoils", we will see the "emMBSlaveWriteCoils" parameters settings below:

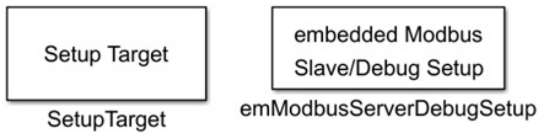


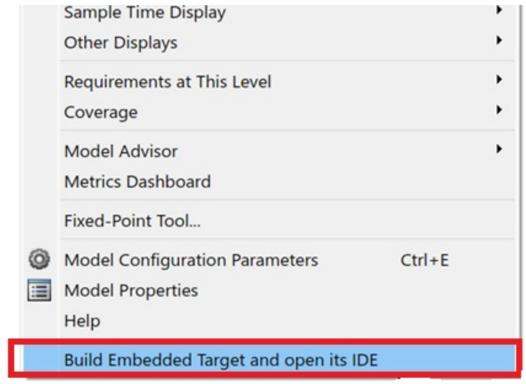
It means this block will receive coil register from Modbus server ID=1 Uart No =2. This block's in-port "StartAddr" connects Constant block with value= 8 which denotes coil register starting address is 00008. This block's in-port "Data" connects "Constant" block output, which provide boolean vector [1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 1] to in-port "Data". So we will see Modbus server (ID=1) coil register 00008 to 00010 with value "true", coil register 00011 to 00013 with value "false", ... , coil register 00020 to 00024 with value "true" if we start running simulation.

"Probe" block is just for watching "emMBSlaveWriteCoils" block output. For any block output except "Constant block", if its name does not contain "Right Arrow" *(→)*, you must use "Probe" block to watch its value.

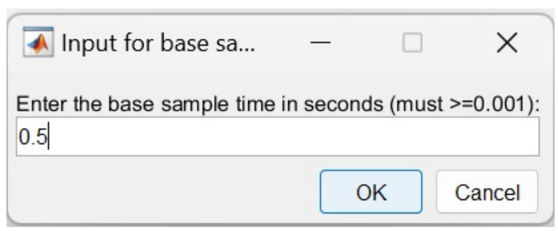
Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too). We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"

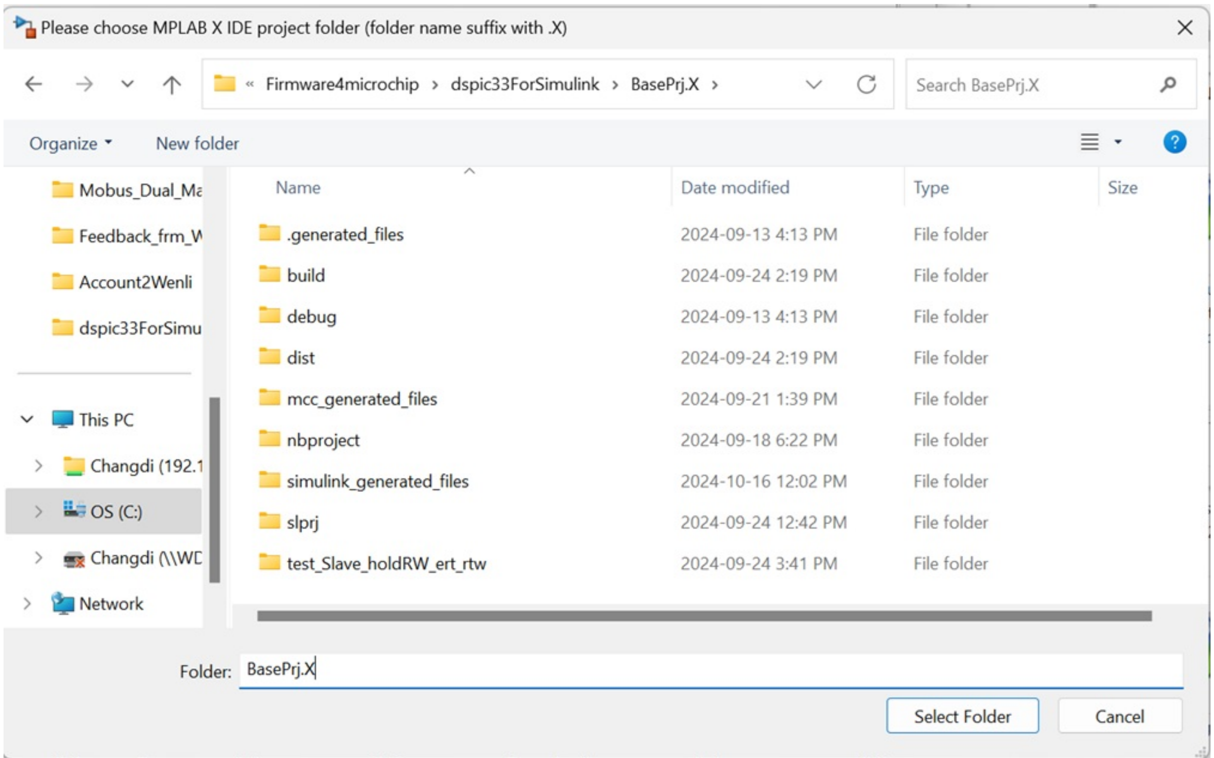




It will start build, and popup dialog window to input base sampling period:

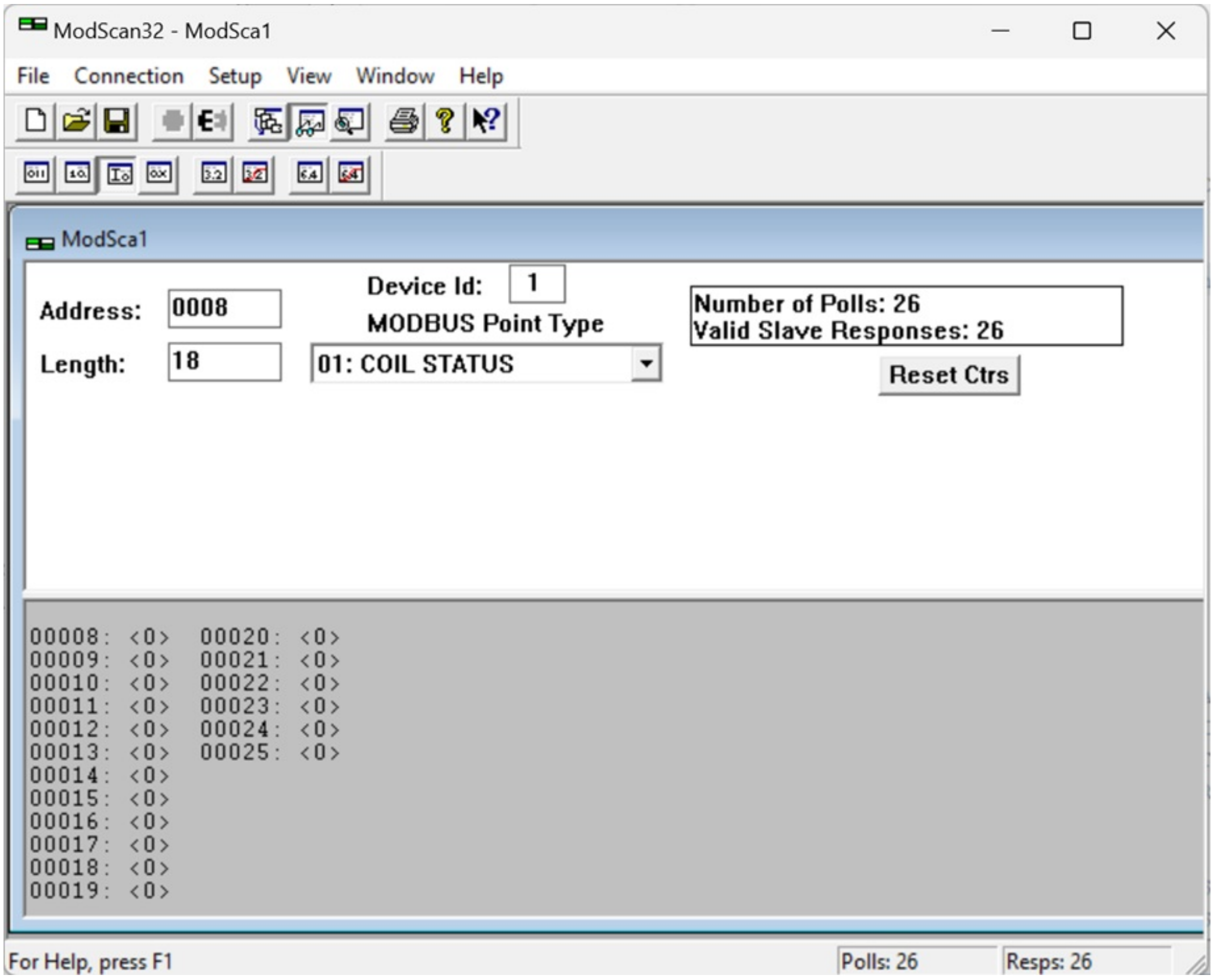


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

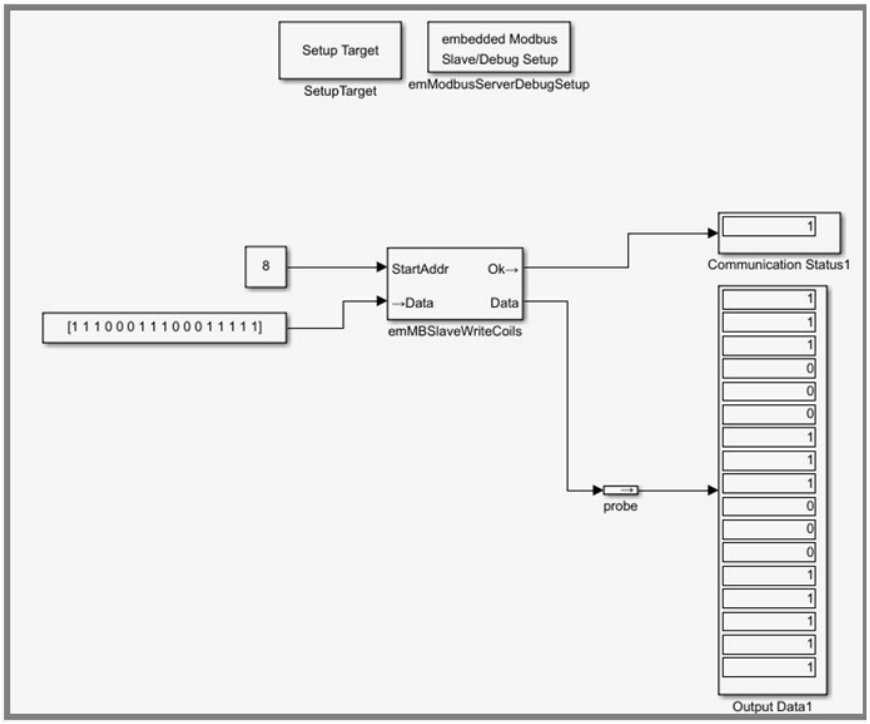
If your firmware program into target successfully, you can use ModScan32 software to view result below:



However, we see all coil regsiter values are 0 (false). All results seem not what we expect.

The reason is input port Data for "emMBSlaveWriteCoils" is from PC Side. If we didn't run simulink, embedded target didn't receive coil register writing command, so all coil register value are zero (false).

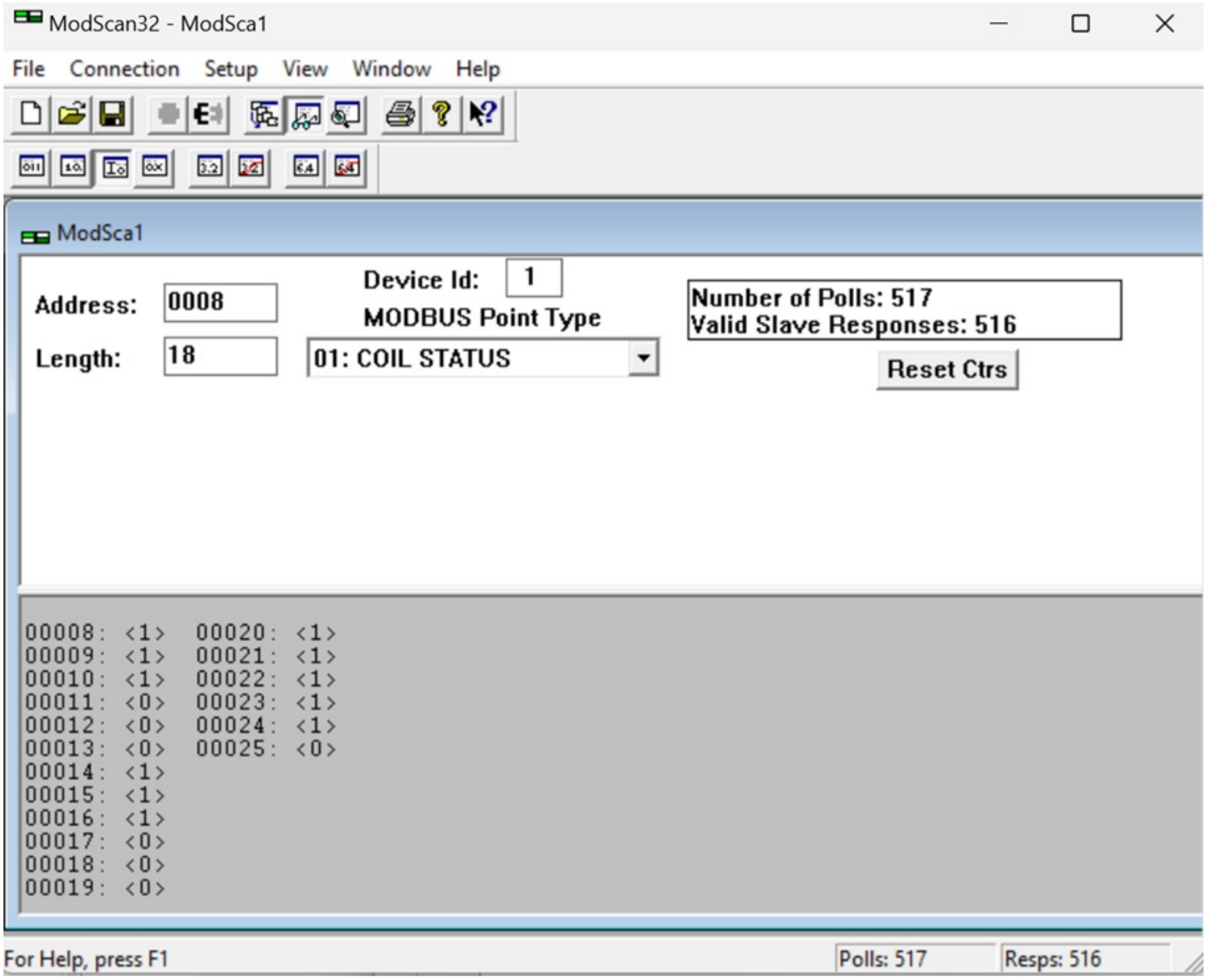
Disconnect ModScan32 software, we can use Simulink directly view results. By select stop time= inf, and click "Run" button, you will see result below:



You will see all results in all "Display" Blocks. It is what we expect.

> ***Notes**: You can run both ".*ModScan32 " and "Simulink" at the same time. But you must enable "Modbus RTU/ASCII Dual Masters adaptor" Bluetooth, .and "ModScan32" cannot use the same COM port as Simulink. We recommend to use Simulink instead of *".*ModScan32 " because *"* ModScan32 " cannot watch general variables. Simulink can watch any variables by "Probe" (also called emProbe) blocks.

Now, we can stop simulink, and run ModScan32 software, you can use ModScan32 software to view result below:



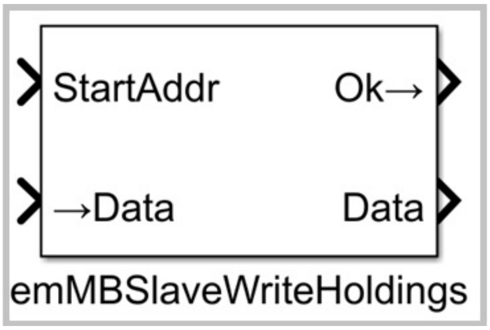
We can see coil register 00008 to 00024 value are as what we expect.

Please open "Your embedded creator library folder"/examples/example6_emWriteCoils.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emModbusSlaveWriteHoldings

Embedded Modbus Server receives holding register writing value.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Modbus Slave & Debugger / emModbusSlaveWriteHoldings



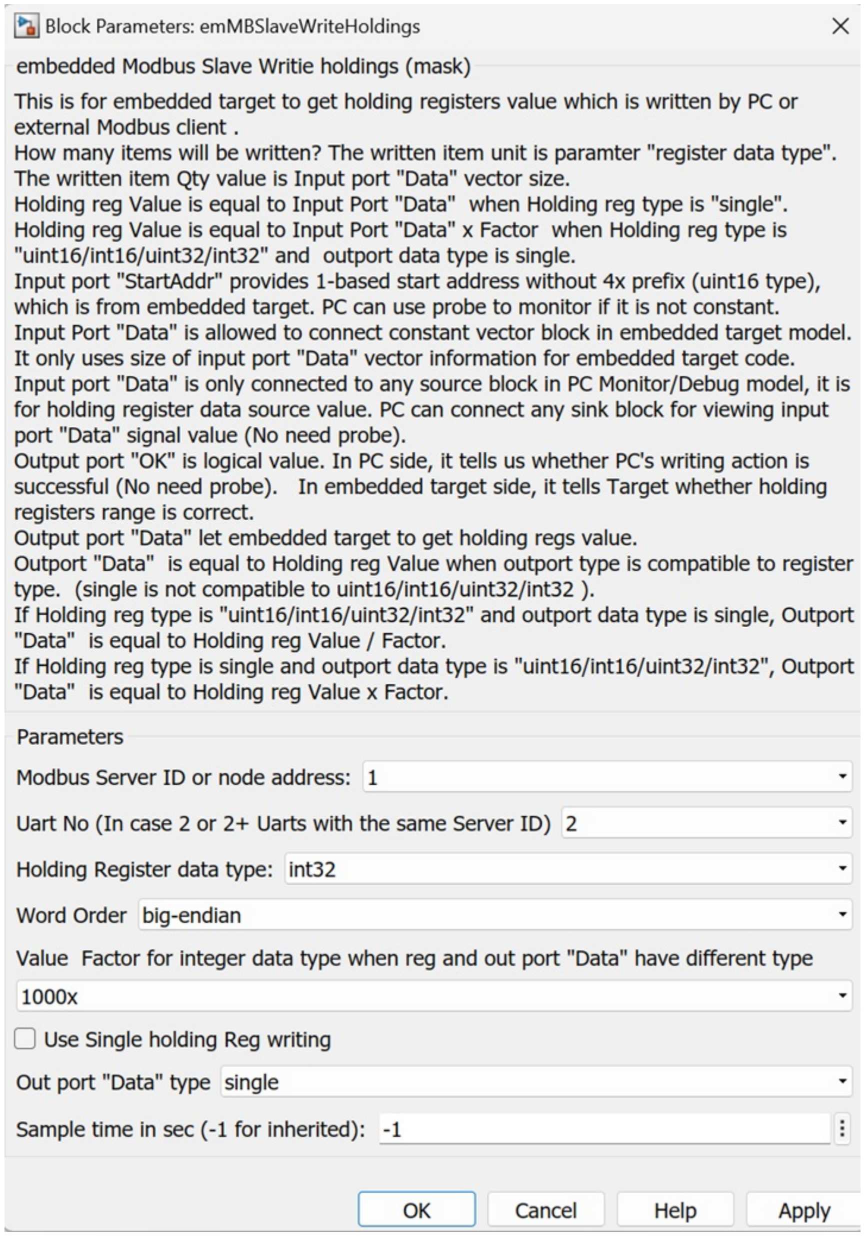
Description

For embedded target side, this block will receive my Modbus server holding register writing value and embedded target gets data only from this block's Output Data. Block's input port Data is only for PC side to send Holding register data value.

> ***Notes***: *Any Port name with "Right Arrow" (→) symbol contains "built-in" probe. So in PC side, you can watch its value directly by "Display" block, you don't need add "Probe" block (our emProbe) before "Display" block. Of cause, you can still use "Mux" block to collect all watching variables and then connect one "emProbe" which connects "Display" block. In this way, you can decrease communication traffic.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Modbus Server ID or node address — tell system this block is for which Modbus Server node. You just choose from drop list which is from "emModbusSlaveDebugSetup" block.
- Uart No (In case 2 or 2+ Uarts with the same Server ID) — It is Uart Number Modbus Server uses. This is only used in one Server ID for more different Uarts. In this situation, you cannot connect these 2 or more nodes into one Modbus network because node Sever ID must be unique in one Modbus network.
- Holding Register data type — It is Modbus holding register data type, It can be "uint16, int16,

uint32, int32 and single (4 bytes float)". When data type's byte QTY is over 2 bytes, it will have word order endian parameter.

- Word Order — It is visible when Register data type is "uint32, int32, single". It tells system words order : Big-endian or Little-endian.

- Value Factor for integer data type when reg and out-port "Data" have different type — It can be "1x, 10x, 100x, and 1000x". We will explain meaning in parameter "out-Port Data type:" below.

- Use Single holding Reg writing — It will use FC=06 (Single writing holding reg) to replace FC=16 (Multiple writing holding registers).

- Out-Port "Data" type: — It can be "uint16, int16, uint32, int32 and single (4 bytes float)". The embedded firmware translates Modbus holding register data to output port "Data". When Out port "Data" is data type of "single" (4 bytes float) and Modbus holding register data type is "uint16 or int16 or uint32 or int32", Modbus holding register data value will be input port "Data" value times "Value factor" and then round it to integer. And output port "Data" value will be Modbus holding register data value ÷ "Value factor". For example, input port "Data"= [2.35 5.8], holding register data value will be [24 58] and output port "Data"=[2.4 5.8] if Value Factor is 10x.
  Similarly, When output port "Data" is data type of "uint16 or int16 or uint32 or int32" and Modbus holding register data type is "single", Modbus holding register data value will be input port "Data" value. But output port "Data" value is Holding register value times "Value factor". For example, input port "Data"=[2.78 -5.67], holding register data value will be [2.78 -5.67], output port "Data" value is [278 -567] if Value Factor is 100x.

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is Modbus Server holding registers' start address (1-based without prefix "4X"). It must be from Constant block or from "emProbe" output because both PC side and embedded side must know its value.

- Data — Scalar or Vector. Don't care Data type. We only care the size of vector in embedded target. It decides how many items in unit " Holding register data type" will be written. And the value of this input port will be written to target holding registers with/without factor from PC Debug/monitor. If holding registers data type is single, Input port Data value will be written to holding registers. However, if holding registers data type is "uint16/int16/uint32/int32" and output port data type is "single, Input port Data value times parameter factor will be written to holding registers.

Outport

- Ok — "logical" data type's scalar. In PC side, it denotes whether communication between PC and Target is OK. In embedded target side, it denotes whether holding register range you read is valid.

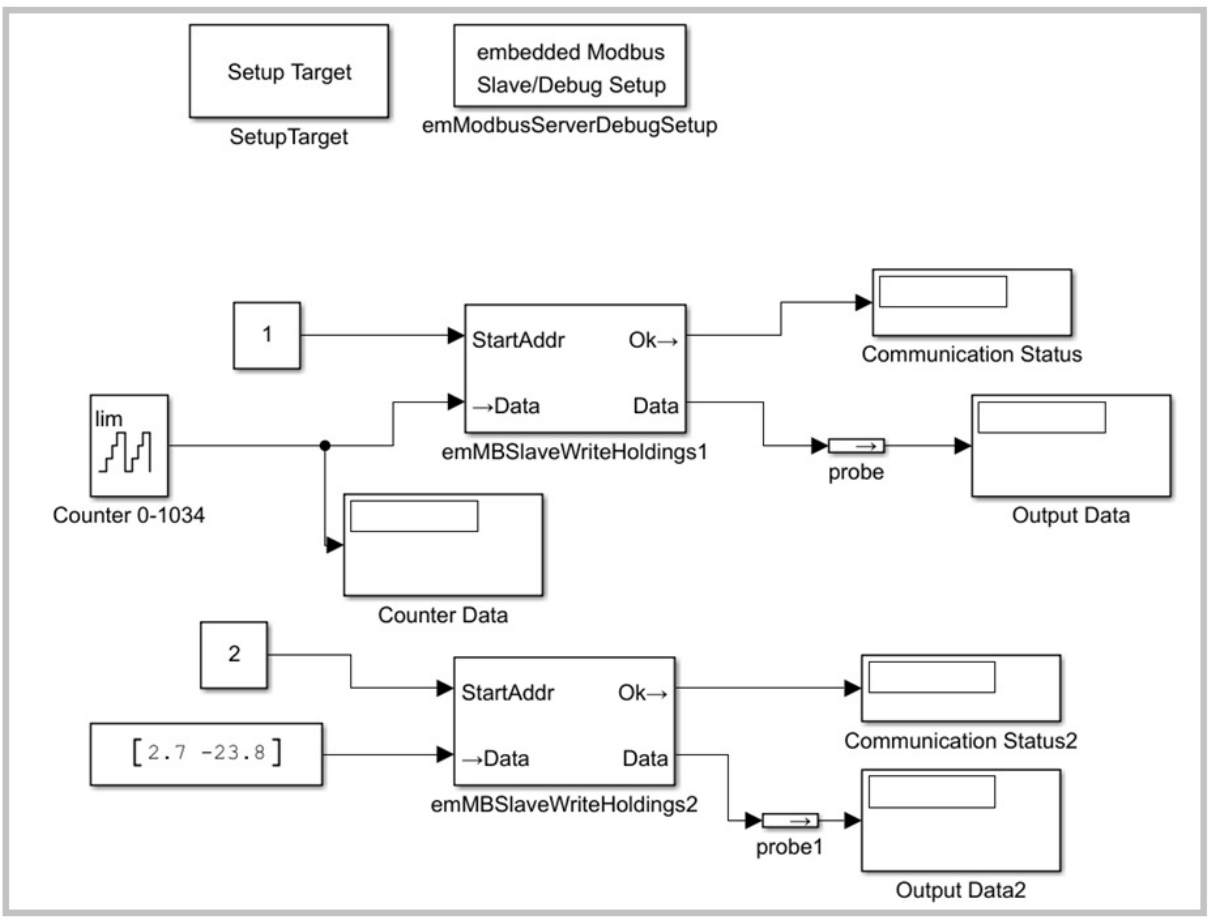
- Data — Scalar or Vector. Data type is decided by parameter "Out-Port 'Data' type". It is from holding register data, and may or may not apply "Value factor". Please see parameter "out-Port Data type:" for details.

Examples

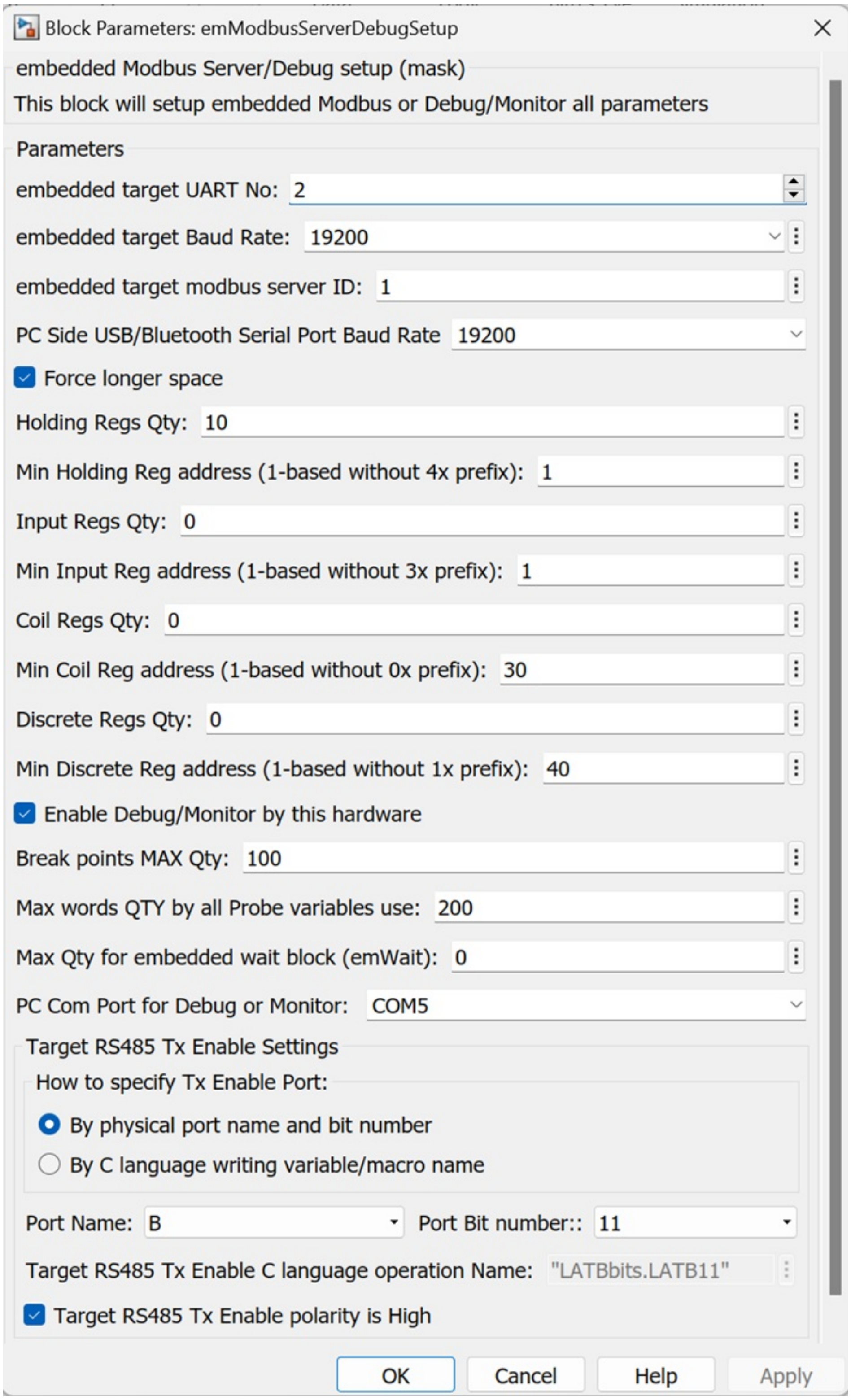
Example1:

Our embedded platform is dsPIC33EP256GP502 . Uart2 connects RS485 Transceiver, and TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver. Our embedded Modbus Server ID=1, and supports holding registers address range: 40001 to 40010, baud rate=19200

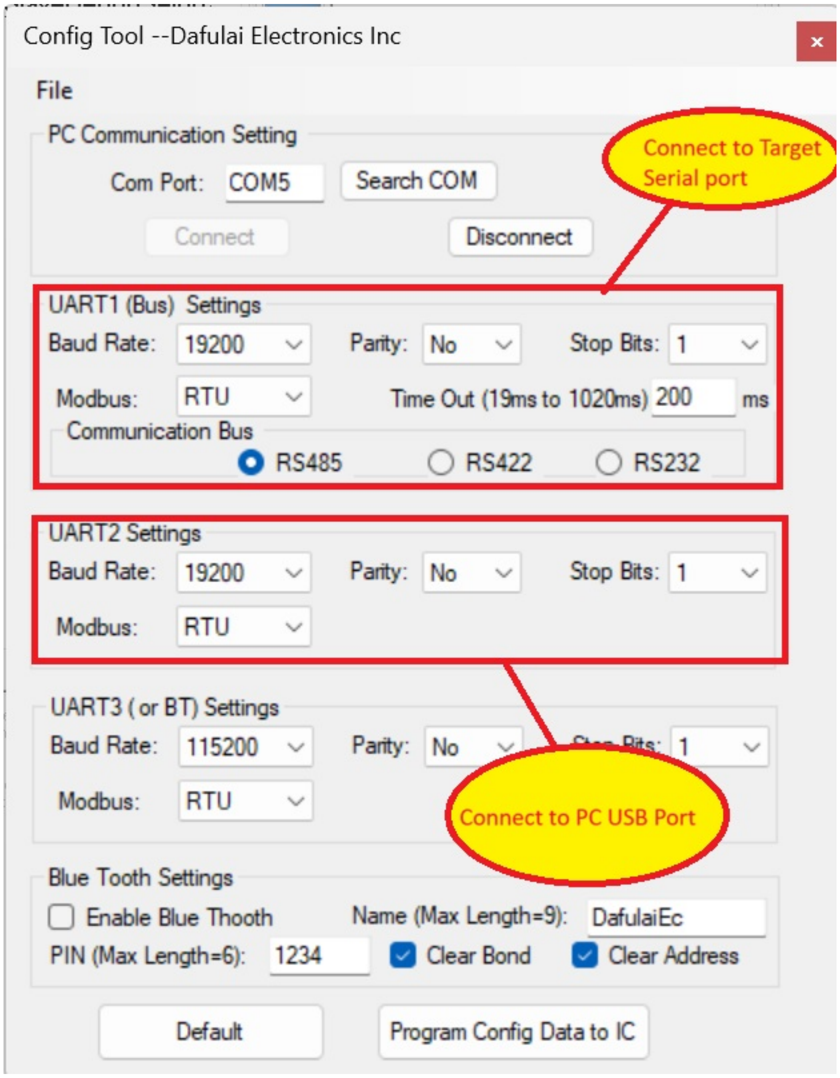
Please see screenshot of model below:



In "Setup Target" block, we choose "PIC24/Dspic30/Dspic33" platform. Double click "emModbusServerDebugSetup", we will see the Modbus Server/Debug parameters settings below:

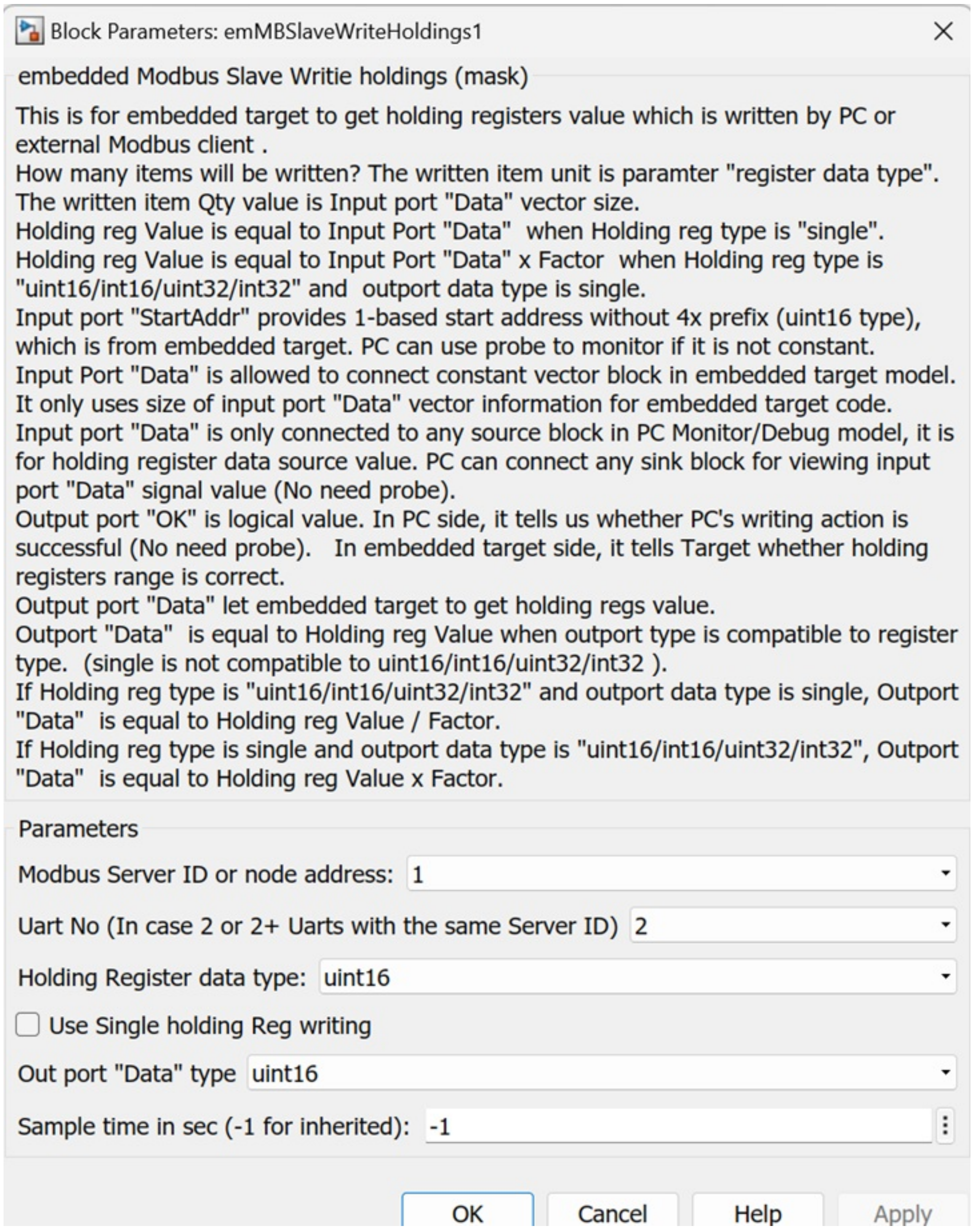


In our hardware environment, we use "Modbus RTU/ASCII Dual Masters adaptor" as debugger/ monitor. Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



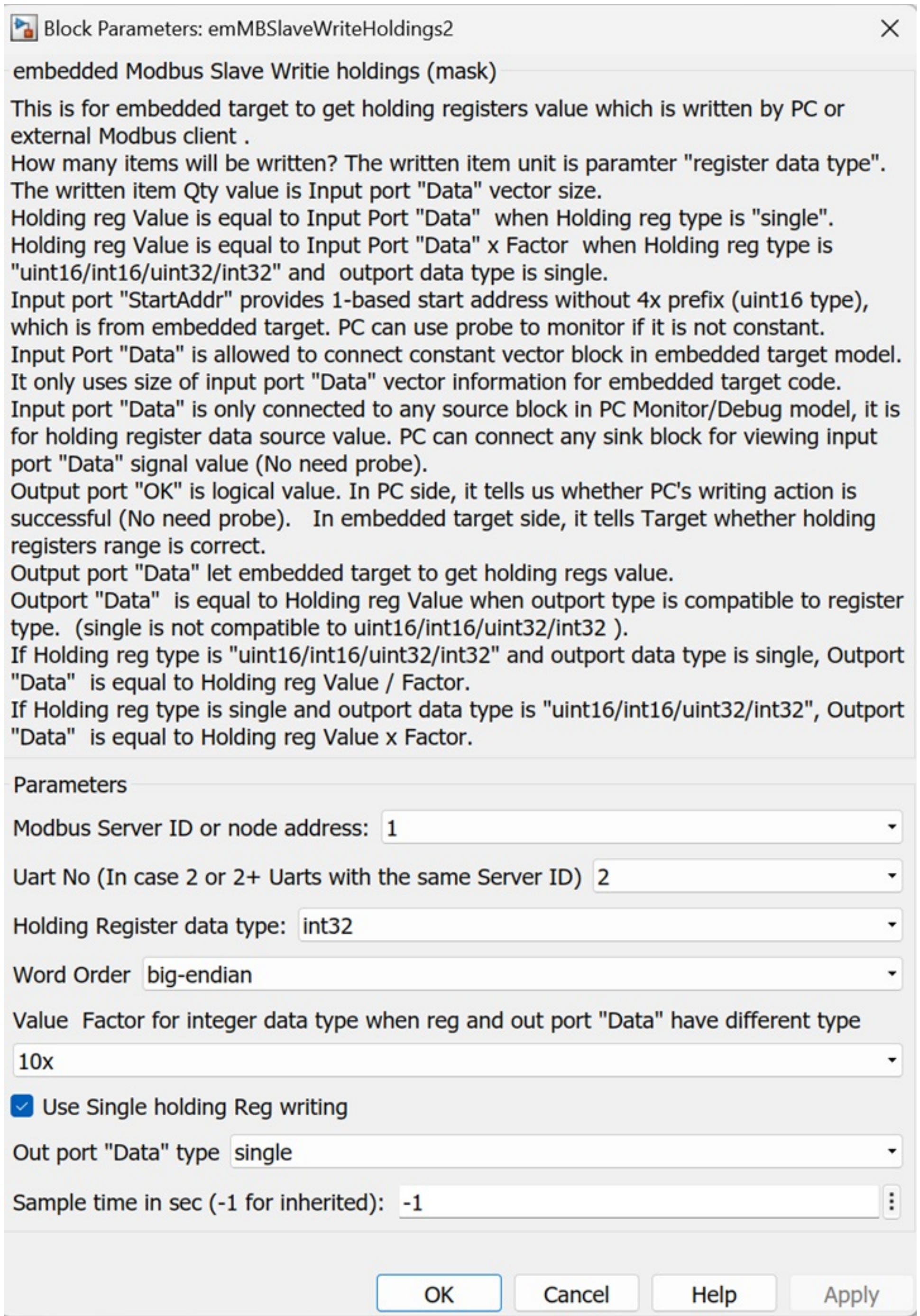
In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

Double click "emMBSlaveWriteHoldings1", we will see the "emMBSlaveWriteHoldings1" parameters settings below:



It means this block will receive holding register from Modbus server ID=1 Uart No =2 and holding register data type is "16 bits of unsigned integer". This block's in-port "StartAddr" connects Constant block with value= 1 which denotes holding register starting address is 40001. This block's in-port "Data" connects "Counter Limited" block output, which provide 0 to 1034 increasing counter output value (uint16 data type) to in-port "Data". So we will see Modbus server (ID=1) holding register 40001 increase 1 from 0 to 1034 every sampling period.

Double click "emMBSlaveWriteHoldings2", we will see the "emMBSlaveWriteHoldings2" parameters settings below:



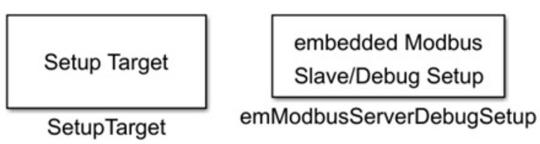
It means this block will receive holding register from Modbus server ID=1 Uart No=2 and Input register data type is "32 bits of signed integer" (big-endian). This block's in-port "StartAddr" connects Constant block with value= 2 which denotes holding register starting address is 40002. This block's in-port "Data" connects "Constant" block output, which provide vector [2.7 -23.8] to in-port "Data". So we will see Modbus server (ID=1) holding register 40002 value= 0 and holding register 40003=27 (big-endian, and factor =10, so 2.7x10 =27) . We also see holding register 40004 value= -1 and holding register

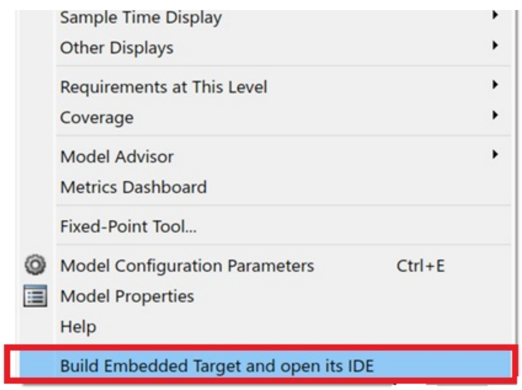
40005=-238 (big-endian, and factor =10, so -23.8 x10 = -238. The first word is -1 not 0 due to 2's complement reason)

"Probe" block is just for watching "emMBSlaveWriteHoldings" block output. For any block output except "Constant block", if its name does not contain "Right Arrow" *(→)*, you must use "Probe" block to watch its value.

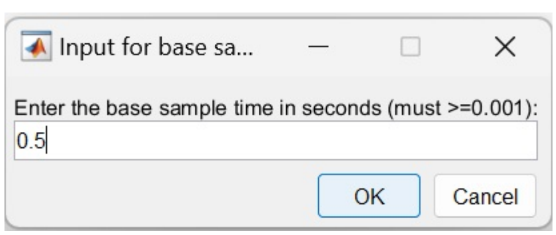
Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too). We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"

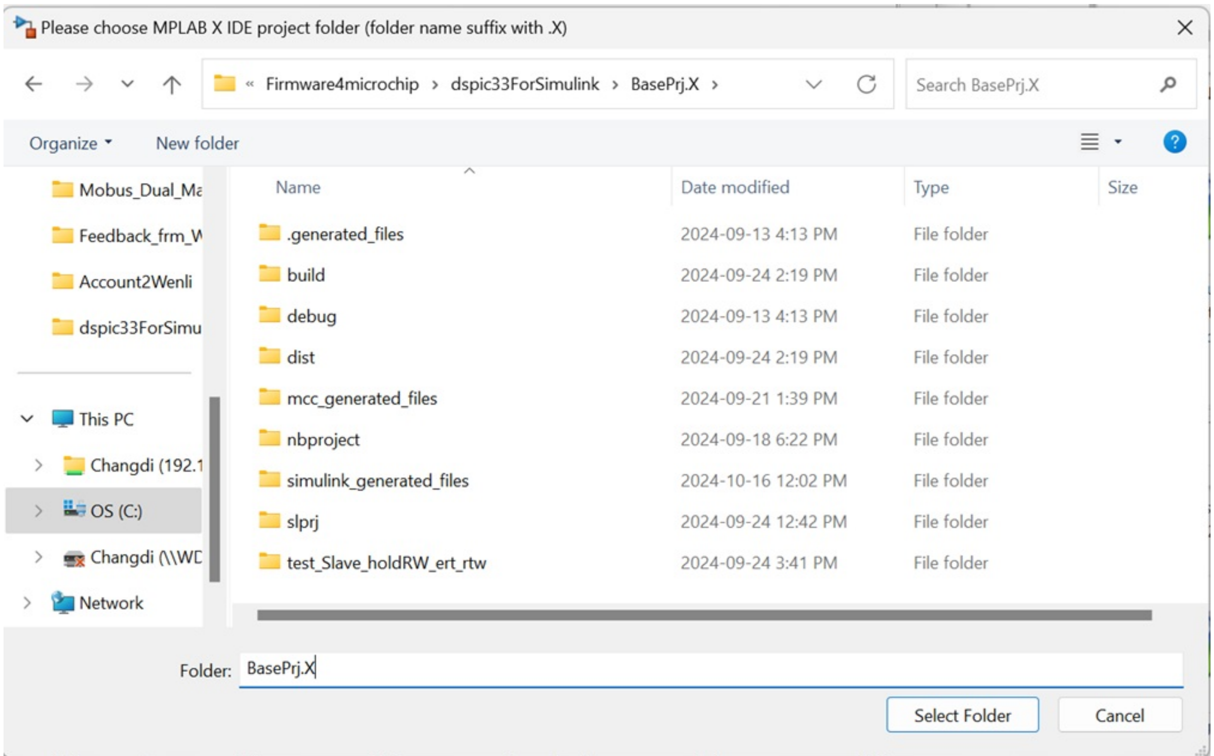




It will start build, and popup dialog window to input base sampling period:

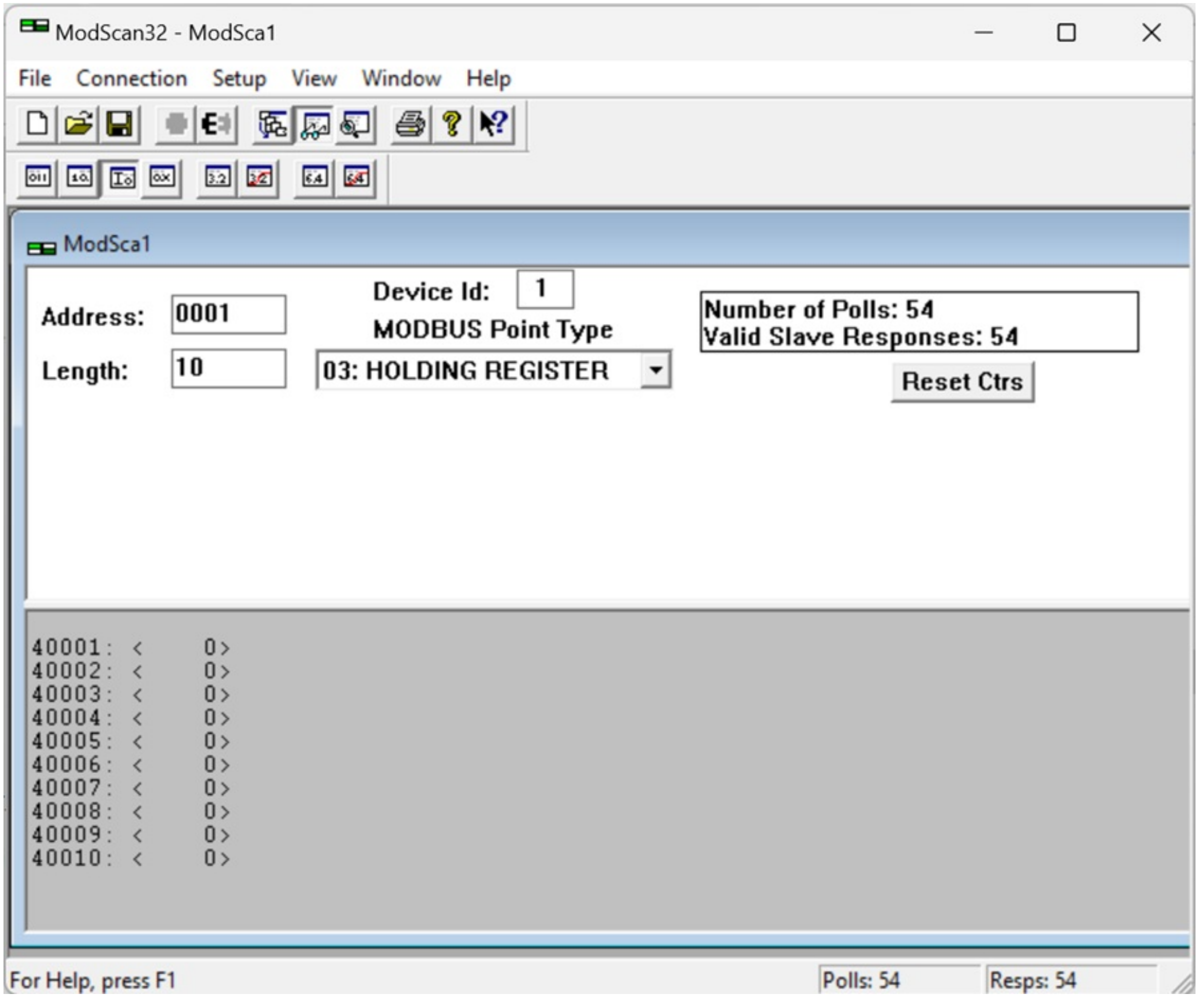


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

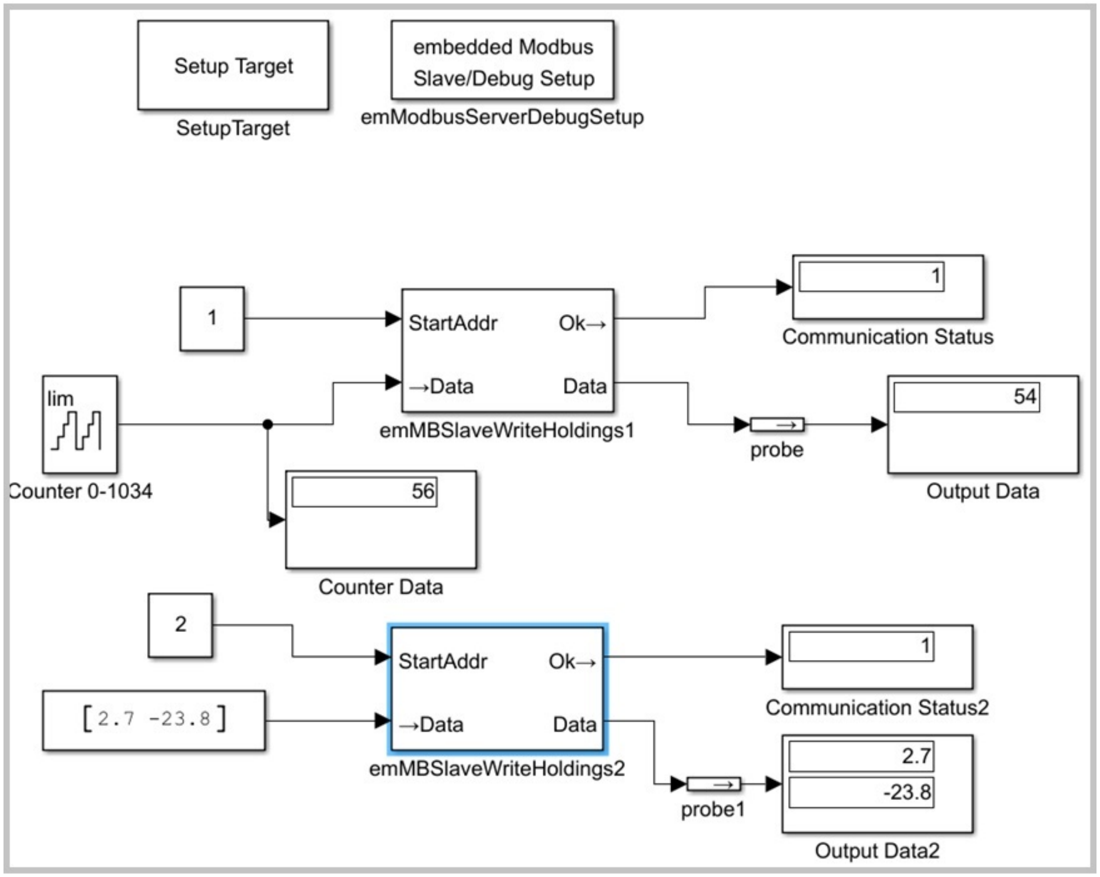
If your firmware program into target successfully, you can use ModScan32 software to view result below:



However, we see all holding regsiter values are 0. we can not see address 40001 value changing from 0 to 1034 every 0.5 sec. All results seem not what we expect.

The reason is Both input port Data for "emMBSlaveWriteHoldings1" and "emMBSlaveWriteHoldings2" are from PC Side. If we didn't run simulink, embedded target didn't receive holding register writing command, so all holding register value are zero.

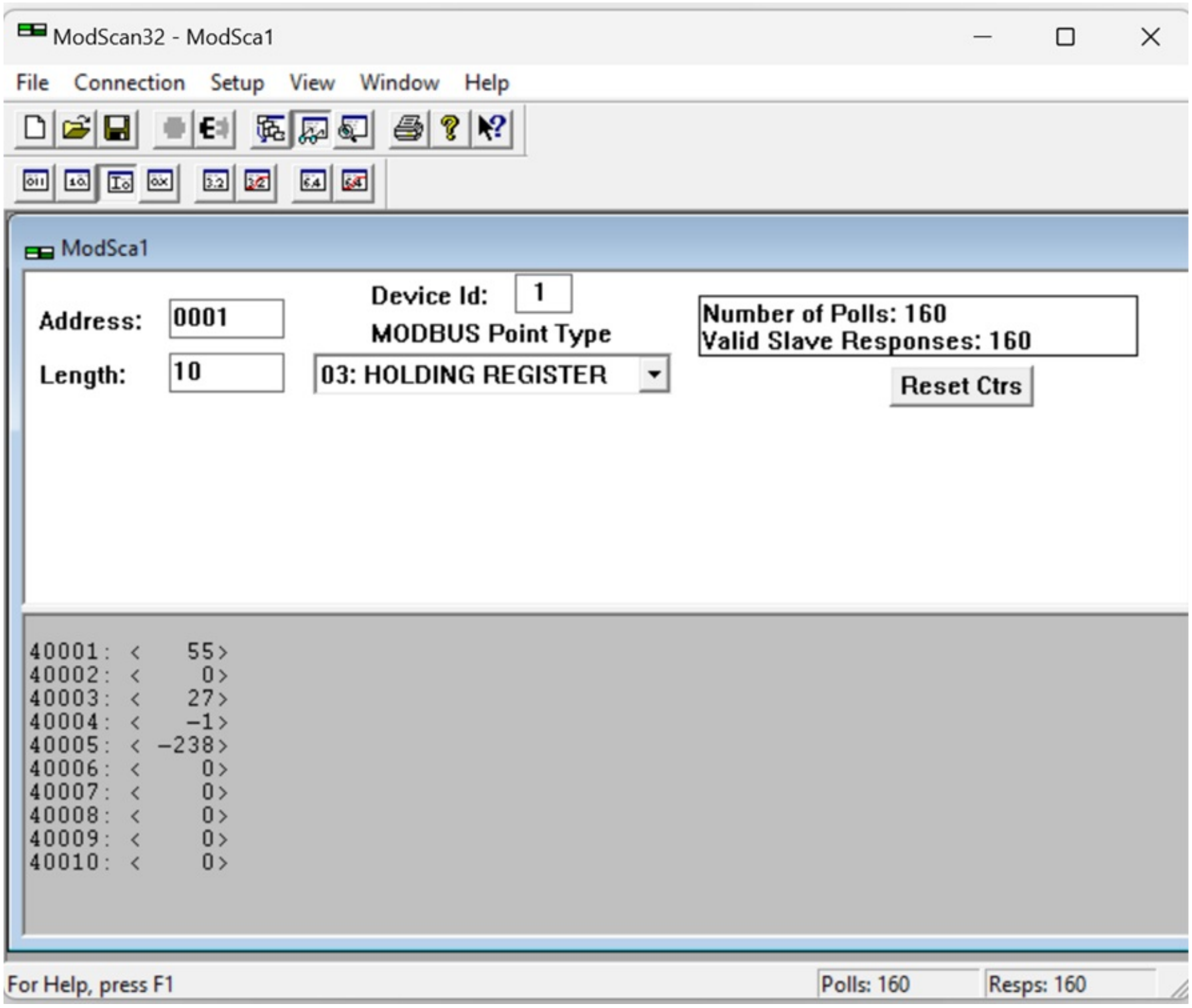
Disconnect ModScan32 software, we can use Simulink directly view results. By select stop time= inf, and click "Run" button, you will see result below:



You will see all results in all "Display" Blocks. You may see Output Data Display=54, but Counter Data Display =56. The reason is synchronization. Counter Data is from PC Simulink block data directly, but Output Data is from probe which is from communication between PC and embedded target, it has some delay.

***Notes**: You can run both ".*ModScan32 " and "Simulink" at the same time. But you must enable "Modbus RTU/ASCII Dual Masters adaptor" Bluetooth, .and "ModScan32" cannot use the same COM port as Simulink. We recommend to use Simulink instead of *".*ModScan32 " because *"* ModScan32 " cannot watch general variables. Simulink can watch any variables by "Probe" (also called emProbe) blocks.

Now, we can stop simulink, and run ModScan32 software, you can use ModScan32 software to view result below:



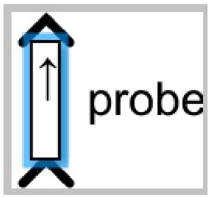
We can see holding register 40001 value =55 (fixed, no increasing because PC Simulink stop running), and 40002 is 0, 40003 is 27, 40004 is -1 , 40005 is -238.

Please open "Your embedded creator library folder"/examples/example5_emWriteHoldings.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

emProbe

embedded Probe.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Modbus Slave & Debugger / emProbe



Description

This block will watch Simulink signal. Input Port connects Signal you want to watch. Output Port connects block "Display" to display signal values

***Notes**: 1 Not all signals need "emProbe" to watch. For signal from port with ""Right Arrow" (→) symbol, you can directly connect block "Display" to watch them.*

2 You can use block "Mux" to collect multiple signals, and then use one "emProbe" to watch. This way can save communication traffic, and all signals values are simultaneous.

3 Constant Block connecting "emProbe" can be used "On-site" Tuning constant value. The output of "emProbe" can be adjusted from PC side.

Parameters

None

Ports

Input

- — Any data type's scalar or vector. You connect it to any Simulink Signal you want to watch.

Outport

- — The same as input port's signal. It connects block "Display" to watch signal values.

Examples

Example1:

Please see example in block "simEEPROMRead".

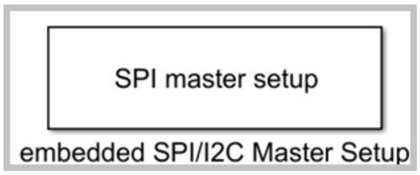
Please open "Your embedded creator library folder"/examples/example8_simEEPROM.slx

The following blocks are in "EmbeddedSPI/I2C Bus Master " directory of library. They are all related to SPI bus master or I2C bus master operation. We list them according to alphabetical order.

emSPI_I2CMasterSetup

set up embedded Master SPI or I2C bus.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded SPI/I2C Bus Master / emSPI_I2CMasterSetup



Description

Set up embedded Master SPI or I2C bus.

Let's introduce Our Master SPI/I2C access blocks.
It is the same way as Master Modbus and CAN bus transmitting.
The embedded Master SPI/I2C bus reading/writing operations are in Sequences.
Total max Sequences quantity is 15 from Seq0 to Seq14.
Small Sequence Number has higher priority for accessing.

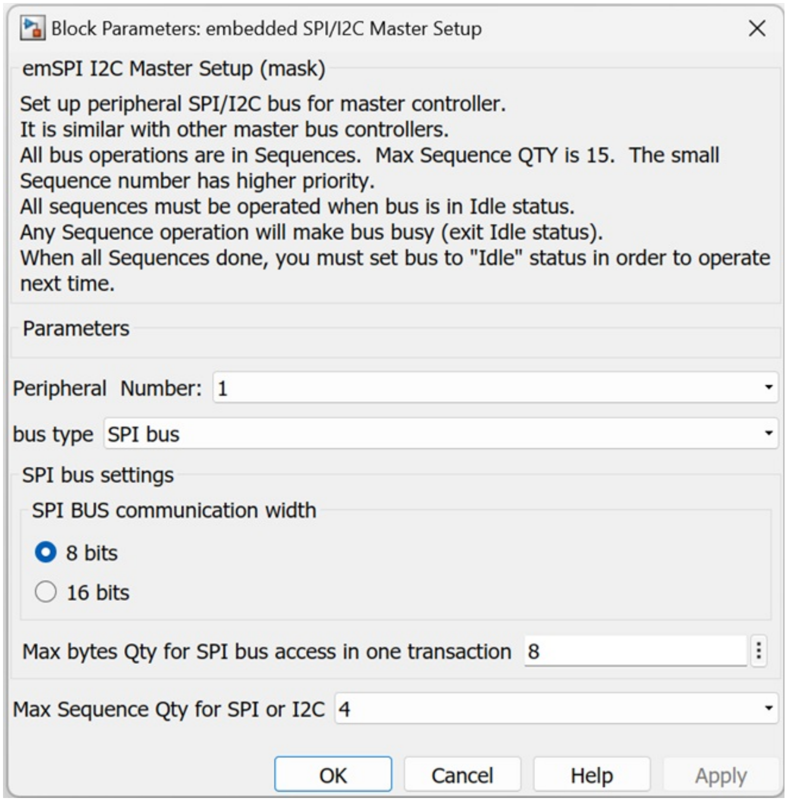
All sequences must be operated when bus is in Idle status. You can use block "isSPI_I2C_Idle" to know whether bus is in "Idle" status.
Any Sequence operation will make bus busy (exit Idle status).
When all Sequences done, you must set bus to "Idle" status in order to operate next time.
You just use block "emSPI_I2CAllDone" to know whether all Sequences done.
You just use block "setSPI_I2C2Idle" to set Bus in "Idle" status when all Sequences done.

Notes: *1 You must set up I2C bus interrupt enable and SPI bus interrupt disable in MCU Configuration software such as MPLABX IDE MCC for Microchip technology MCUs.*
2 In STM32CubeIDE/STM32CubeMx platform, you must enable interrupt for SPI/I2C. But you cannot enable DMA.
3 In TI C2000Ware /SysConfig platform, for I2C, you must enable interrupt, but for FIFO, don't care. Some F28Xs such as F28P65X do not work for "pure read" when FIFO enabled. please use "First write then read" to replace or just disable FIFO. For SPI bus, you can enable interrupt or enable DMA.
4 In TI C2000Ware/SysConfig platform,Peripheral 2/3/4 maybe SPIA or I2CA if you didn't add SPIB/SPIC/SPID or I2CB/I2CC/I2CD. For example , You use the 4th peripheral SPI,

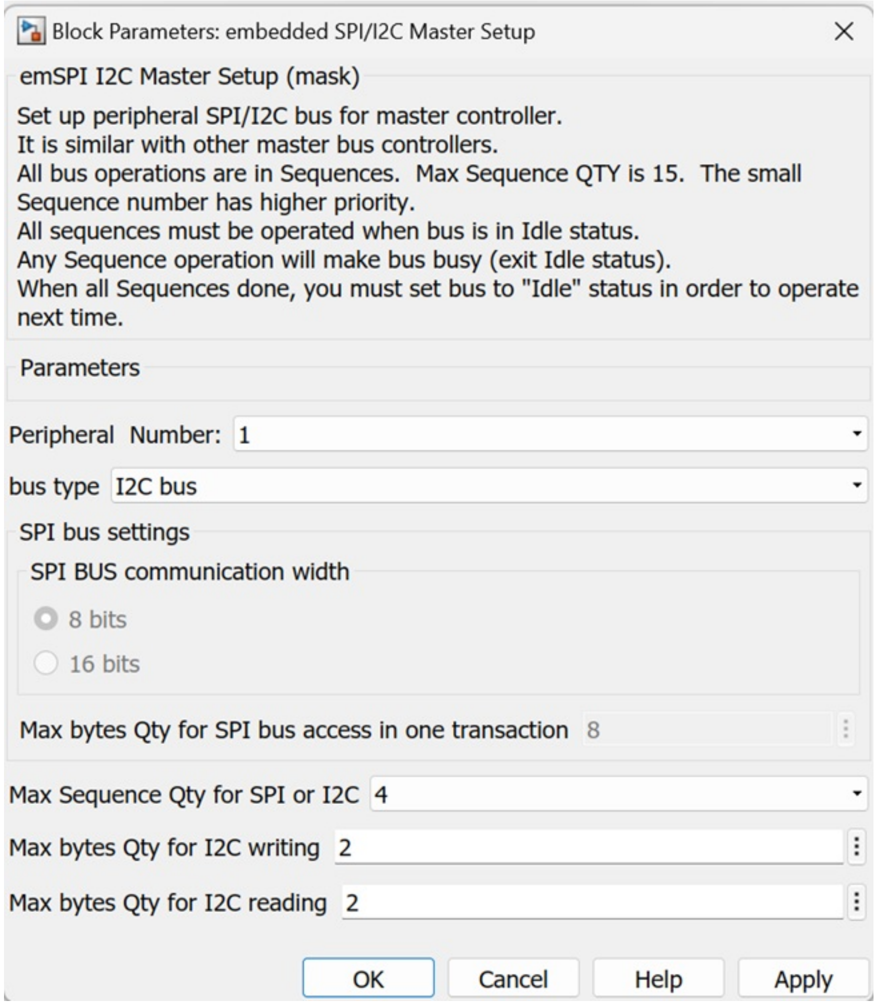
in general, it will use SPID because TI perrpheral naming in "A,B,C,D" order. However, our block found that SysConfig only added SPIB peripheral. And SPIB is not used by other emSPI_I2CMasterSetup blocks. No SPID found. Our block will use SPIB.

Parameters

Please double click this block to open parameters dialog below:



When we choose Bus type to I2C bus from drop list, the dialog displays below:



Let us explain parameters.

- Peripheral Number — tell system which SPI/I2C controller peripheral is used. You just choose from drop list items: 1 to 4. It must match settings in your configuration software such as MCC for Microchip technology MCUs

- bus type — tell system which bus is used. You just choose from drop list items: SPI bus and I2C bus.

- SPI BUS communication width — This is SPI Bus Data width. You can choose 8 or 16. It is only for SPI bus, not for I2C bus. SPI bus's other settings like Pol/ Phase (Mode) are in your configuration software such as MCC for Microchip technology MCUs

- Max bytes Qty for SPI bus access in one transaction — This is for SPI bus not I2C bus. We know that Actual transmitting data QTY including dummy data is equal to actual received data QTY including not interested received data. This parameter specifies the maximum of data QTY in byte unit in all transactions.

- Max Sequence Qty for SPI or I2C — As its name, it is Max Sequence Qty, you just choose from drop list items from 1 to 15,

- Max bytes Qty for I2C writing — It is only used for I2C bus. For one sequence of I2C operation, it can be First Writing and then reading in one transaction (One "start" signal, and then writing, and then "re-start " signal, and then reading, at last "stop" signal). It can be Writing only in one transaction (One "start" signal, and then writing, at last "stop" signal). It can be Reading only in one transaction (One "start" signal, and then reading, at last "stop" signal). This is Max byte Qty of writing in all sequences.

- Max bytes Qty for I2C reading — It is only used for I2C bus. For one sequence of I2C operation, it can be First Writing and then reading in one transaction (One "start" signal, and then writing, and then "re-start " signal, and then reading, at last "stop" signal). It can be Writing only in one transaction (One "start" signal, and then writing, at last "stop" signal). It can be Reading only in one transaction (One "start" signal, and then reading, at last "stop" signal). This is Max byte Qty of reading in all sequences.

Ports

Input

None

Outport

None

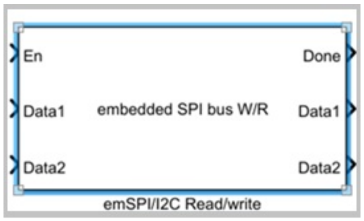
Examples

Please see "emSPI_I2C_Master_RW" block example.

emSPI_I2C_Master_RW

set up one sequence operation for embedded Master SPI or I2C bus.
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded SPI/I2C Bus Master / emSPI_I2C_Master_RW



Description

One operation of SPI/I2C bus Data Write/reads is done by one sequence. This one operation contains multiple bytes (words) continuous write/reads.

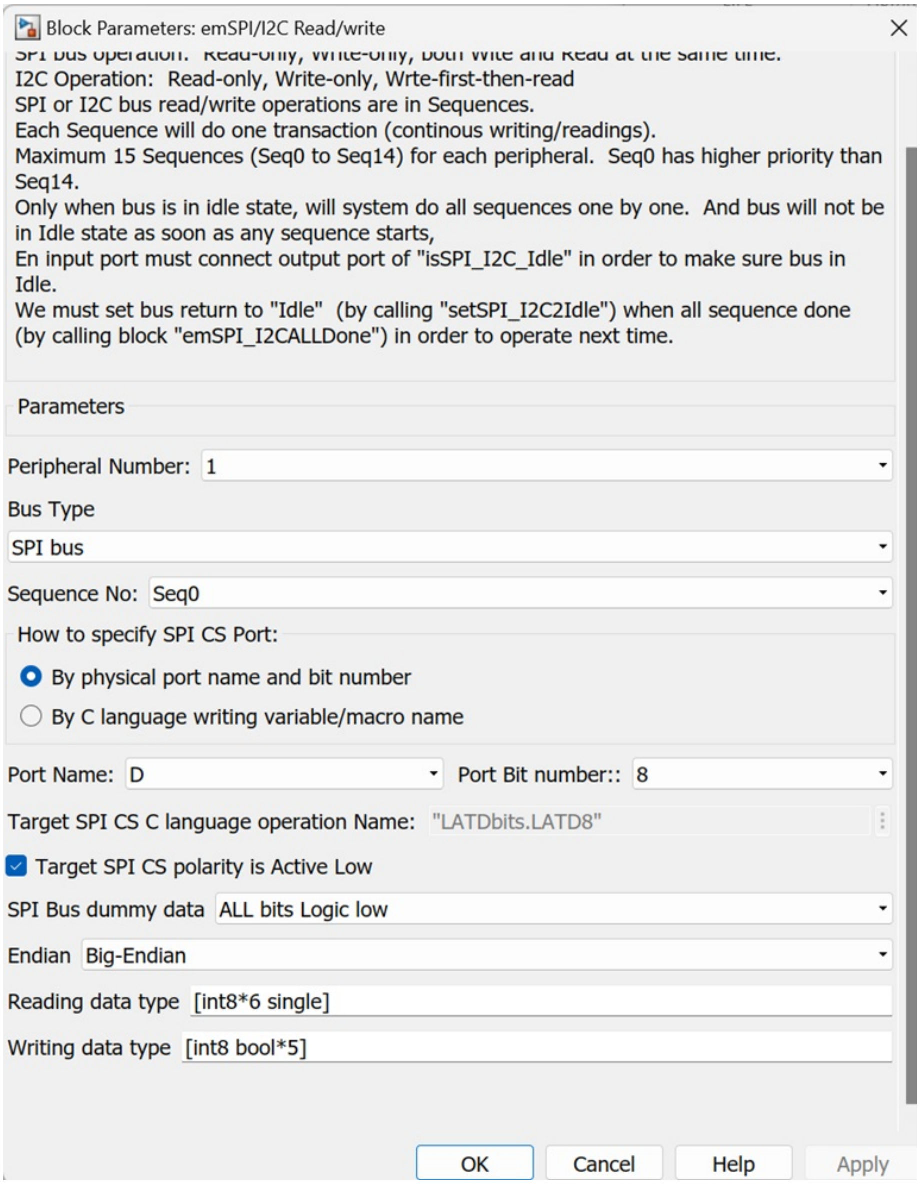
All operations are in serial by Sequences. Maximum 15 Sequences: Seq0 to Seq14 are allowable.

Seq0 is the most highest priority, Seq14 is the most lowest priority.

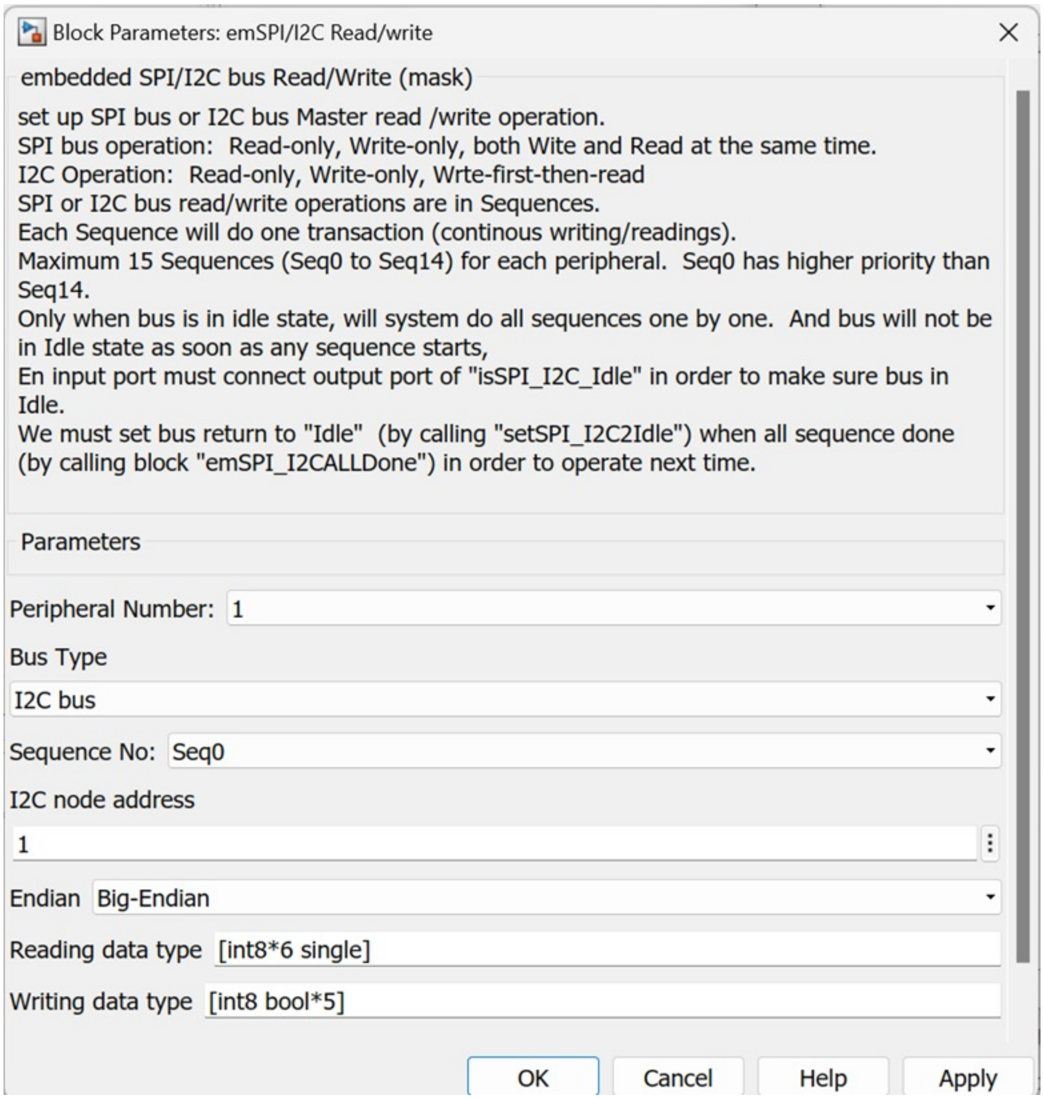
Before you set up any sequence, you must make sure bus is in Idle state. The is guaranteed by connecting "En" input port to output port of block "isSPI_I2C_Idle". After all sequences done (actual reading/writings action finish), you must call block "setSPI_I2C2Idle" to set bus to idle state. How to know all sequences done ? Please call block "emSPI_I2CAllDone"

Parameters

Please double click this block to open parameters dialog below:



When we choose Bus Type to I2C bus from drop list, the dialog displays below:



Let us explain parameters.

- Peripheral Number — tell system which SPI/I2C controller peripheral is used. You just choose from drop list items which are from block "emSPI_I2CMasterSetup".
- Bus Type — tell system which bus is used. You just choose from drop list items: SPI bus and I2C bus.
- Sequence No — Sequence Number for this transaction. You just choose from drop list items which are from parameter "Max Sequence Qty for SPI or I2C" of block "emSPI_I2CMasterSetup".
- How to specify SPI CS Port — This is for SPI bus, not I2C bus. It has 2 selections, one is "By physical port name and bit number", the other is "By C language writing variable/macro name".

- Port Name — It is used when you choose "By physical port name and bit number" for previous parameter. It is which GPIO Port is used for SPI CS. You just choose from drop list items.

- Port Bit number — It is used when you choose "By C language writing variable/macro name" for parameter "How to specify SPI CS Port". It is which bit number is used for SPI CS. You just choose from drop list items.

- Target SPI CS C language operation Name — It is used when you choose "By physical port name and bit number" for parameter "How to specify SPI CS Port". It specifies C variable name for accessing SPI CS. Please use quotation mark for variable name.

- Target SPI CS polarity is Active Low — This is for SPI bus, not I2C bus.If checked, SPI CS will be active low.

- SPI Bus dummy data — This is for SPI bus, not I2C bus. It has 2 items from drop list. One is "ALL bits Logic high", the other is "ALL bits Logic low". We know actual transmitting data length is equal to received data length for SPI bus. However, the meaningful transmitting data length may be less than received data length. In this situation, we need to transmit dummy data to keep equal data length. In general , we have 2 different dummy data, all logic 1 and all logic 0. All logic 1 means 0xFF for 8 bits bus width or 0xFFFF for 16 bits bus width. All logic 0 means 0x0 for both 8 bits and 16 bits bus width.

- I2C node address — It is only used for I2C bus, not for SPI bus. It is I2C slave node address you want to access. It is 7 bits or 10 bits address which didn't contain R/W Flag bit.

- Endian — It is for input/output ports endian when data type's data width is over bus width or bit vector endian.

- Reading data type — It specifies data type of reading data (Output port data type). It can be scalar or vector. element can be "bool","uint8", "int8","uint16","int16","uint32","int32","single". How many vector elements it has is how many output ports it has.If data output port is vector, please follow * and vector element Qty. For example, [int8, uint16*2, bool*7] denotes that reading data has 6 bytes, and block has 3 output ports. The 1st output port is "int8" type of scalar. The 2nd output port is "uint16" type of vector with 2 elements. The 3rd output port is "bool" type of vector with 7 elements. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["int8", "uint16*2", "bool*7"], You cannot use " [int8, uint16*2, bool*7]". You must use [int8, uint16*2, bool*7]. So from this parameter, you can know how many bytes to read for your interesting data. We know, for SPI bus, actual reading data QTY is equal to actual writing data QTY. When interesting reading data QTY is less than interesting writing data QTY for SPI bus, interesting reading data will be the last reading data (the latest data). When interesting reading data QTY is bigger than interesting writing data QTY for SPI bus, the actual writing data will be interesting writing data plus dummy data.

- Writing data type — It specifies data type of writing data (Input port data type). It can be scalar or vector. element can be "bool","uint8", "int8","uint16","int16","uint32","int32","single". How many vector elements it has is how many output ports it has.If data output port is vector, please follow * and vector element Qty. For example, [int8, uint16*2, bool*7] denotes that writing data has 6 bytes, and block has 3 input ports. The 1st input port is "int8" type of scalar. The 2nd input port is "uint16" type of vector with 2 elements. The 3rd input port is "bool" type of vector with 7 elements. Notes: We cannot put any quotation mark in vector or scalar. For example, you cannot use ["int8", "uint16*2", "bool*7"], You cannot use " [int8, uint16*2, bool*7]". You must use [int8, uint16*2, bool*7]. So from this parameter, you can know how many bytes to write for your interesting data. We know, for SPI bus, actual reading data QTY is equal to actual writing data QTY. When interesting reading data QTY is less than interesting writing data QTY for SPI bus, interesting reading data will be the last reading data (the latest data). When interesting reading data QTY is bigger than interesting writing data QTY for SPI bus, the actual writing data will be interesting writing data plus dummy data.

Ports

Input

- En — "logical" data type's scalar. "True" means SPI/I2C controller will accept sequence settings and does actual write/read action. And it will make SPI/I2C controller into "busy" state. In general, "En" connects the output port of "isSPI_I2C_Idle" block.

For other data input ports, it depends on parameter "Writing data type". The parameter "Writing data type" vector element QTY is Input data port QTY. Please read parameter "Writing data type" above. and see description below

- Data1 — vector or scalar. Parameter "Writing data type" vector's first element decides what data type it is. and " *digit" decides vector size

.

- Data2 — vector or scalar. Parameter "Writing data type" vector's second element decides what data type it is. and " *digit" decides vector size.
- Data3 — vector or scalar. Parameter "Writing data type" vector's third element decides what data type it is. and " *digit" decides vector size.

......

- Data n — vector or scalar. Parameter "Writing data type" vector's number n element decides what data type it is. and " *digit" decides vector size.

Outport

- Done — "logical" data type's scalar. "True" means SPI/I2C bus controller has done this sequence transaction. But it may successes or fails for I2C bus.

- Success — "logical" data type's scalar. "True" means I2C bus controller has done this sequence transaction successfully. SPI bus has no this output port.

For other data output ports,It depends on parameter "Reading data type". The parameter "Reading data type" vector element QTY is output data port QTY. Please read parameter "Reading data type" above. and see description below

- Data1 — vector or scalar. Parameter "Reading data type" vector's first element decides what data type it is. and " *digit" decides vector size

.

- Data2 — vector or scalar. Parameter "Reading data type" vector's second element decides what data type it is. and " *digit" decides vector size.

- Data3 — vector or scalar. Parameter "Reading data type" vector's third element decides what data type it is. and " *digit" decides vector size.

......

- Data n — vector or scalar. Parameter "Reading data type" vector's number n element decides what data type it is. and " *digit" decides vector size.

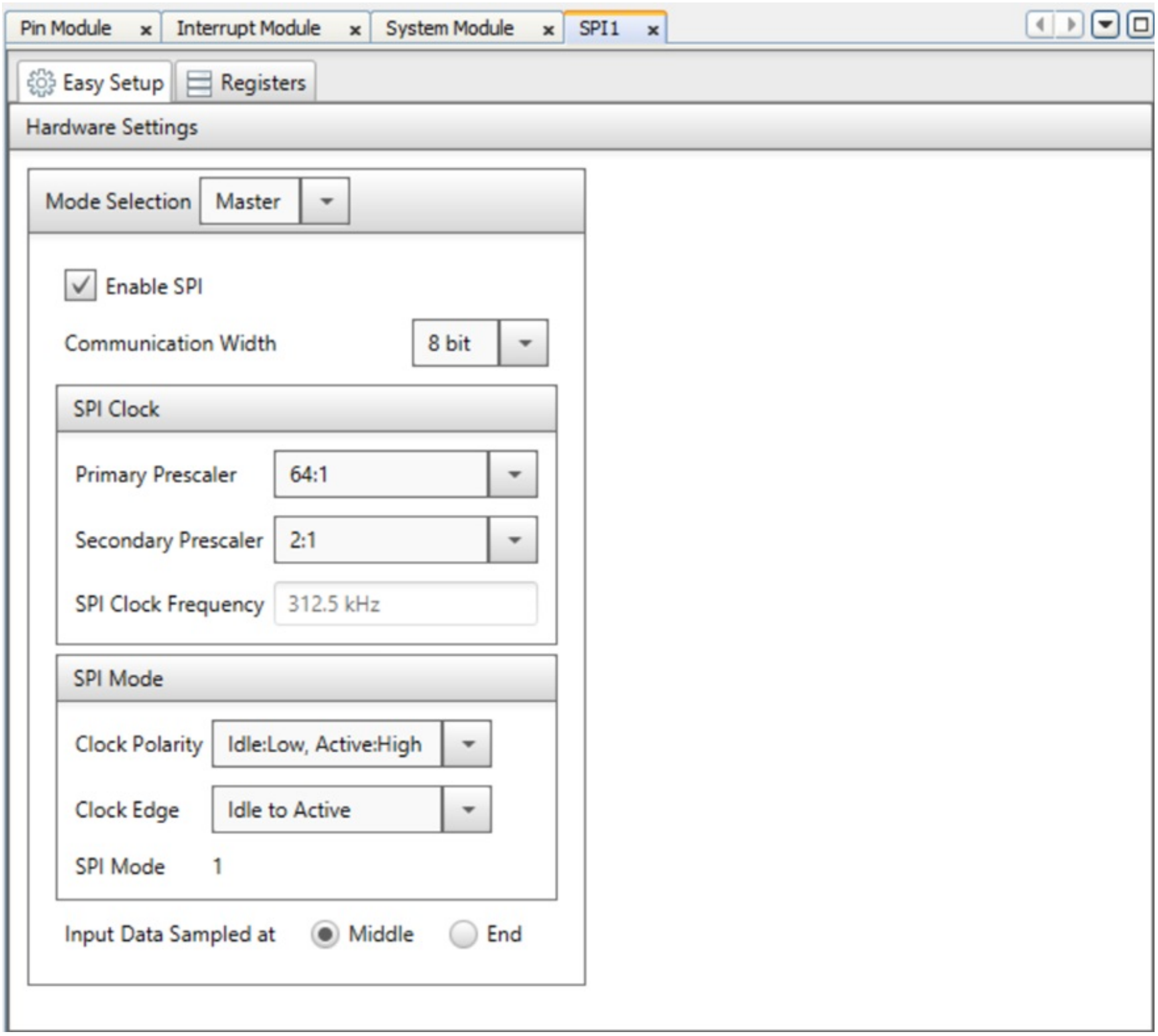
Examples

Example1:

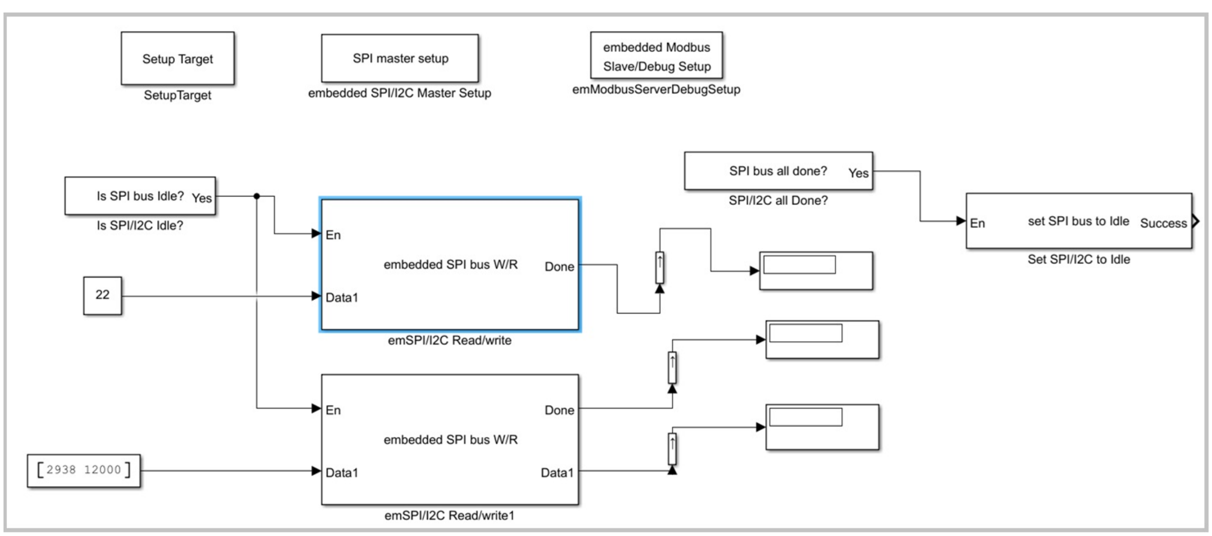
SPI bus access.

Our embedded platform is dsPIC33EP256GP502 .

Please see MCC configuration for SPI bus below:



Please see screenshot of model below:



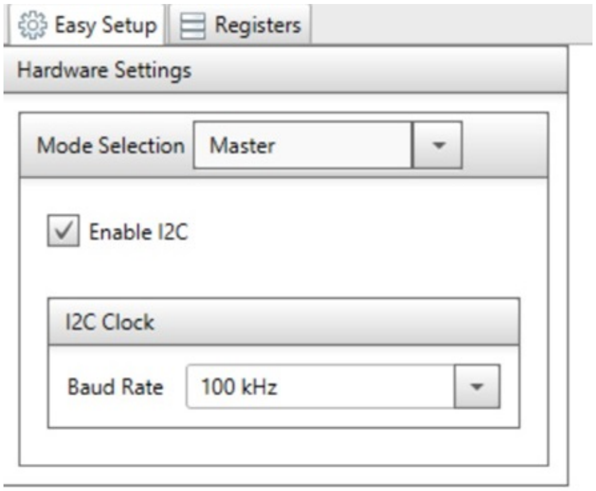
Please open "Your embedded creator library folder"/examples/example12_emSPI.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

Example2:

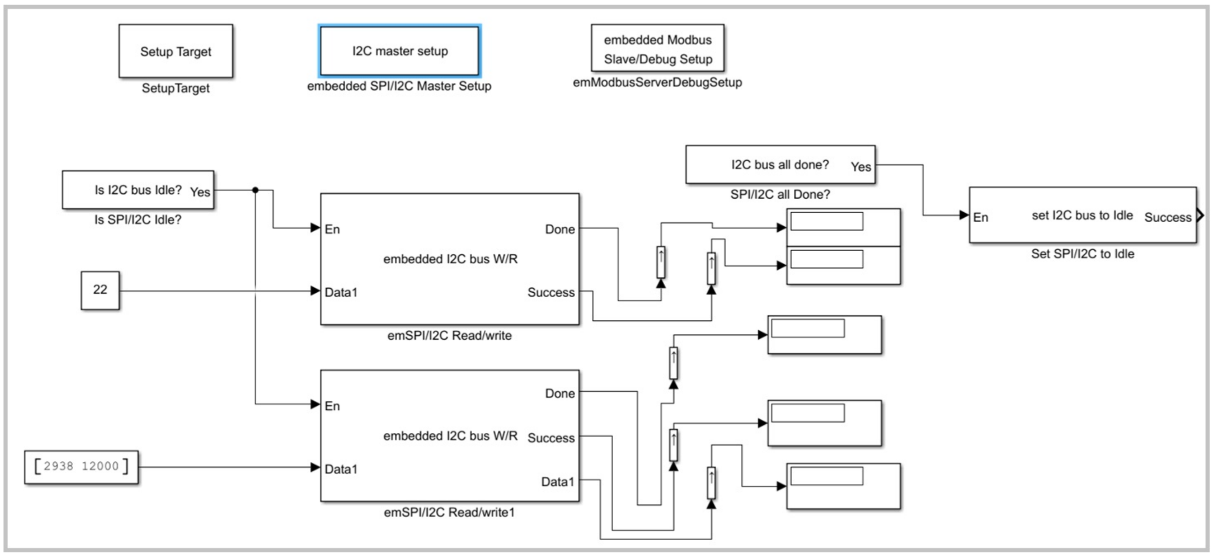
I2C bus access

Our embedded platform is dsPIC33EP256GP502 .

Please see MCC configuration for I2C bus below:



Please see screenshot of model below:

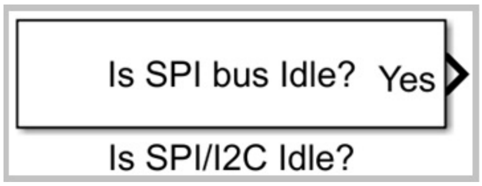


Please open "Your embedded creator library folder"/examples/example13_emI2C.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

isSPI_I2C_Idle

Is SPI/I2C bus in Idle?
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded SPI/I2C Bus Master / isSPI_I2C_Idle



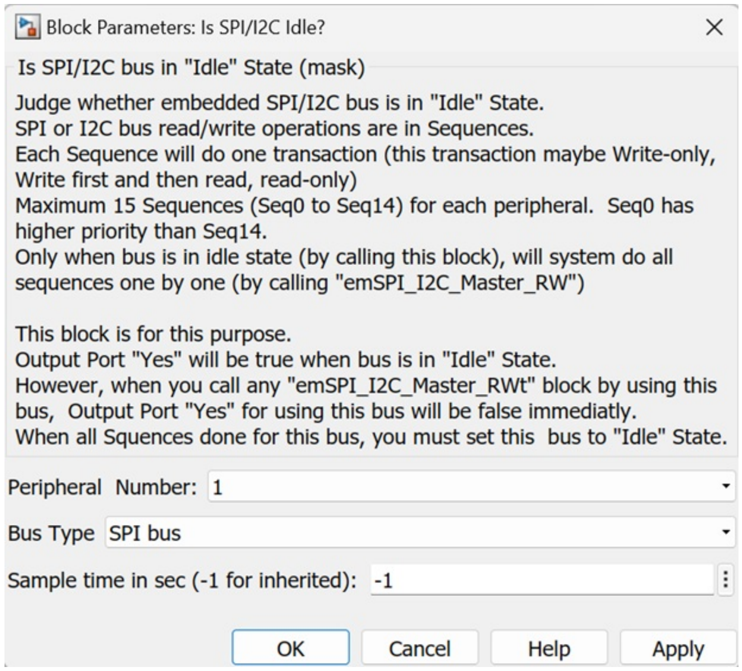
Description

Judge whether embedded SPI/I2C bus is in "Idle" State.

We do any "SPI/I2C bus access " must be under condition : Bus is in "Idle" State.
When embedded "SPI/I2C bus" is in "Idle" state, You can set up all sequences of writing/reading operation, and then bus will exit "Idle" state and operate in sequences for writing/reading automatically. We provide "this block" to check if all sequences finish. In order to start new group of sequences operations next time, you must set bus into "Idle" state when all sequences done.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral Number — tell system which SPI/I2C peripheral is used. You just choose from drop list items which are from block "emSPI_I2CMasterSetup".
- bus type — tell system which bus is used. You just choose from drop list items: SPI bus and I2C bus.
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

None

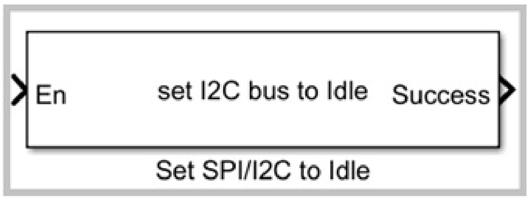
Examples

Please see "emSPI_I2C_Master_RW" block example.

setSPI_I2C2Idle

set SPI/I2C bus into Idle status
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded SPI/I2C Bus Master / setSPI_I2C2Idle

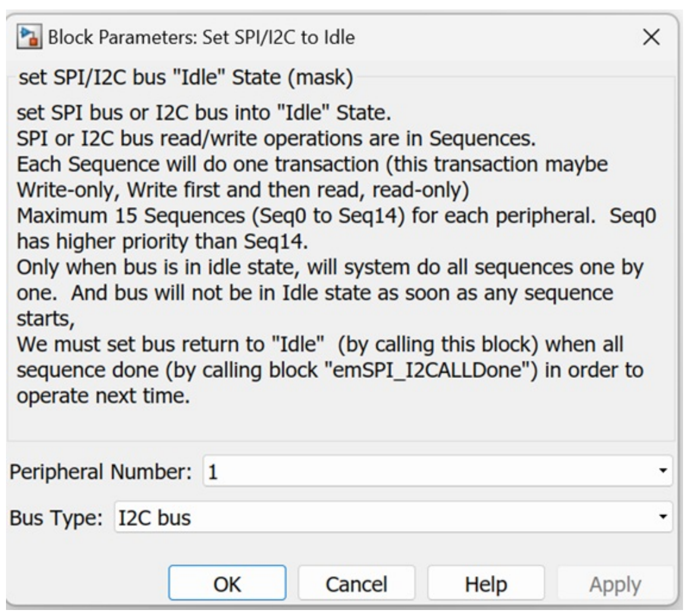


Description

This block sets up SPI/I2C bus to "Idle" State. You must call it under condition: all sequences operations done for this bus. Otherwise, you will get unexpected result. Only when SPI/I2C bus is in "Idle" state, can you start new operations by setting all new sequence of operations.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral Number — tell system which SPI/I2C peripheral is used. You just choose from drop list items which are from block "emSPI_I2CMasterSetup".

- Bus Type — tell system which bus is used. You just choose from drop list items: SPI bus and I2C bus.

Ports

Input

- En — "logical" data type's scalar. True means " it sets Bus to idle state". False means nothing happens.

Outport

- Success — "logical" data type's scalar. It is equal to "En".

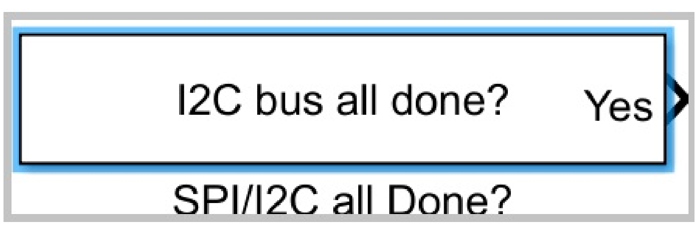
Examples

Please see "emSPI_I2C_Master_RW" block example.

emSPI_I2CAllDone

embedded SPI/I2C bus all sequences Done?
Since R2019b

Library: embeddedCreatorLib (Dafulai Electronics) / Embedded SPI/I2C Bus Master / emSPI_I2CAllDone

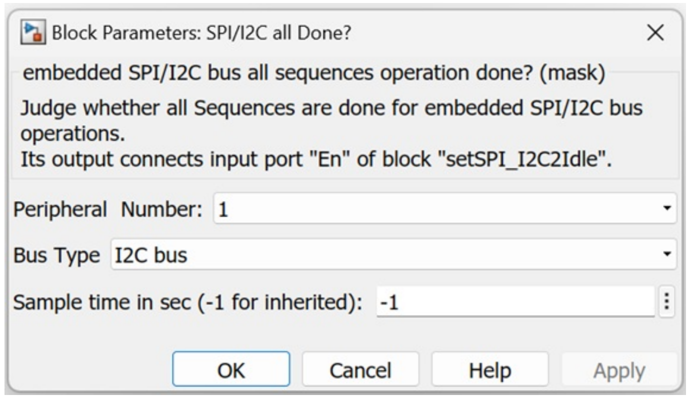


Description

This block is used in judgment whether all sequences finish for SPI/I2C operation. In order to start a new group of sequences operations, you must set bus into "Idle" state when all sequences done.

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Peripheral Number — tell system which SPI/I2C peripheral is used. You just choose from drop list items which are from block "emSPI_I2CMasterSetup".
- Bus Type — tell system which bus is used. You just choose from drop list items: SPI bus and I2C bus.
- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

None

Outport

- Yes — "logical" data type's scalar. True means all sequences done.

Examples

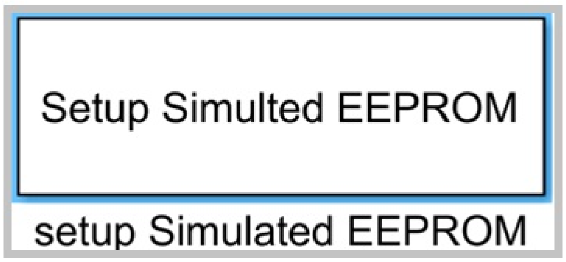
Please see "emSPI_I2C_Master_RW" block example.

The following blocks are in "Simulate EEPROM by Flash " directory of library. They are all related to simulated EEPROM operation. We list them according to alphabetical order.

setupSimulatedEEPROM

set up simulated EEPROM
Since R2020b

Library: embeddedCreatorLib (Dafulai Electronics) / Simulate EEPROM by Flash / setupSimulatedEEPROM



Description

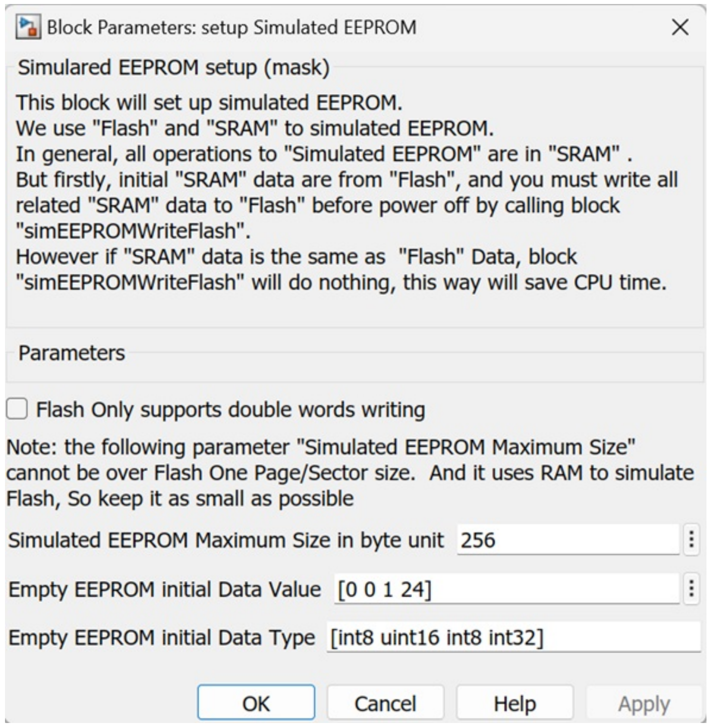
This block sets up "Simulated EEPROM" parameters. For every embedded target which has "Flash" memory, we can use this block to simulate EEPROM by "Flash memory" and "SRAM" memory. In this way, we don't need add "EEPROM" hardware, it will save cost and save PCB space.

In general, we use "SRAM" to represent "Simulated EEPROM". It is fast and byte accessible. However, the initial values of "SRAM" which represents "Simulated EEPROM" are from "Flash" memory, And we have to write all related "SRAM" data into "Flash" memory before "power off". Maximum "Simulated EEPROM" size cannot be over one page (or one sector) of flash size.

> ***Notes**: 1 You must add "Flash memory access" for microchip MCC software.*
>
> *2 This block does not support Microchip PIC18F because most of PIC18F usually have built-in real EEPROM and PIC18F has less SRAM.*

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Flash Only supports double words writing — logical data type of scalar. There are some micro-controllers such some dspic33 MCUs which only supports dual words writing, in this situation, you must make this parameter checked on.
- Simulated EEPROM Maximum Size in byte unit — unsigned integer data type of scalar. It is Maximum size of "Simulated EEPROM" in byte unit. You must make this value to be smaller than or be equal to "Flash Page/Sector" size.
- Empty EEPROM initial byte Data — Number type of scalar or vector. Number type is decided by next parameter "Empty EEPROM initial Data Type". It is initial value of " Simulated EEPROM". If length of vector is less than parameter "Simulated EEPROM Maximum Size in byte unit", it will only initialize the EEPROM's front data parts, the other rear data will be 0.
- Empty EEPROM initial Data Type — String of scalar or vector. Don't use quotation marks for string. It is data type for initial value of " Simulated EEPROM". If length of vector is less than parameter "Empty EEPROM initial byte Data", it will use "uint8" for surplus initial data. If this parameter is scalar, all initial data will use the same data type. Default data type is "uint8". Valid data type is uint8/int8/uint16/int16/uint32/int32/single/float. float is the same as single (4 bytes).

Ports

Input

None

Outport

None

Examples

Example1:

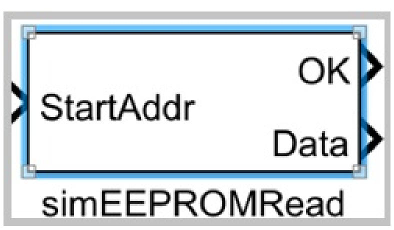
Please see example in block "simEEPROMRead".

Please open "Your embedded creator library folder"/examples/example8_simEEPROM.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

simEEPROMRead

Read simulated EEPROM
Since R2020b

Library: embeddedCreatorLib (Dafulai Electronics) / Simulate EEPROM by Flash / simEEPROMRead

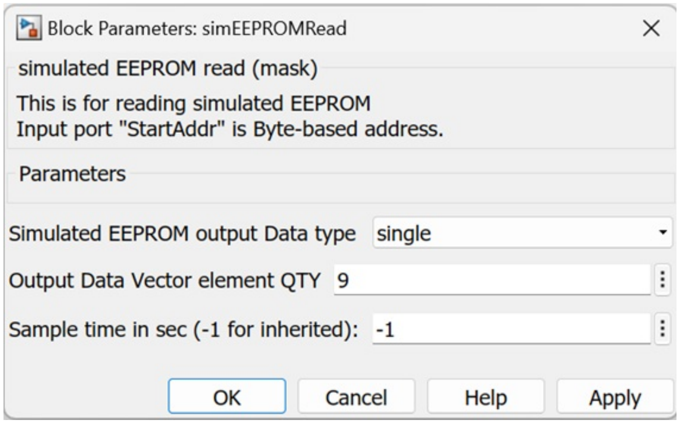


Description

This block read "Simulated EEPROM"

Parameters

Please double click this block to open parameters dialog below:



Let us explain parameters.

- Simulated EEPROM output Data type — drop list from "uint8" /"int8" /"uint16" /"int16" /"uint32" /"int32" /"single" which sets up Data Type you read from "Simulated EEPROM". It is also the data type of the output port "Data".

- Output Data Vector element QTY — unsigned integer data type of scalar. It decides how many items will be read from "Simulated EEPROM". And every item you read is in unit "Simulated EEPROM output Data type".

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is "Simulated EEPROM" start address you want to read. This address is byte addressing.

Outport

- OK — "logical" data type's scalar. True means "Simulated EEPROM" address range you read is valid. Reading data success.

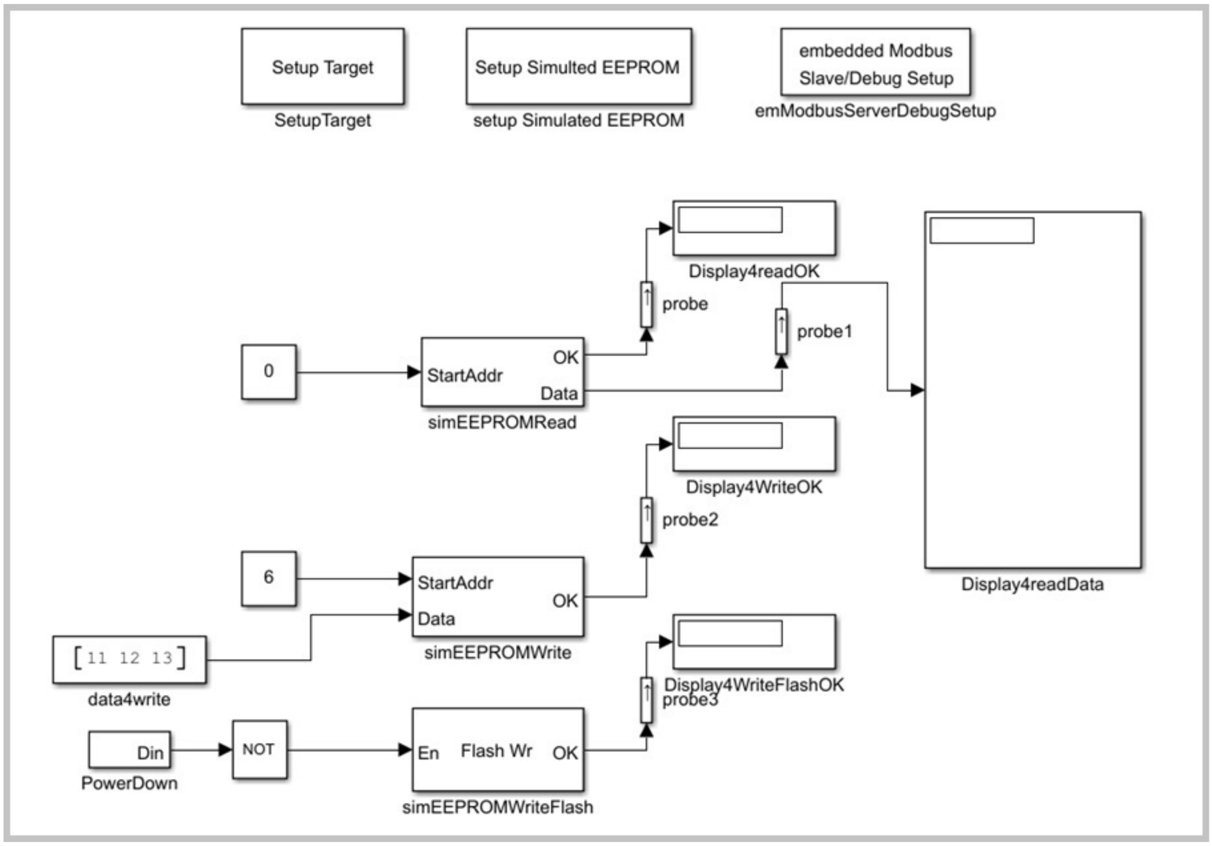
- Data — Scalar or Vector. It is read result. Data type is set in parameter "Simulated EEPROM output Data type" . Vector length is set in parameter "Output Data Vector element QTY"

Examples

Example1:

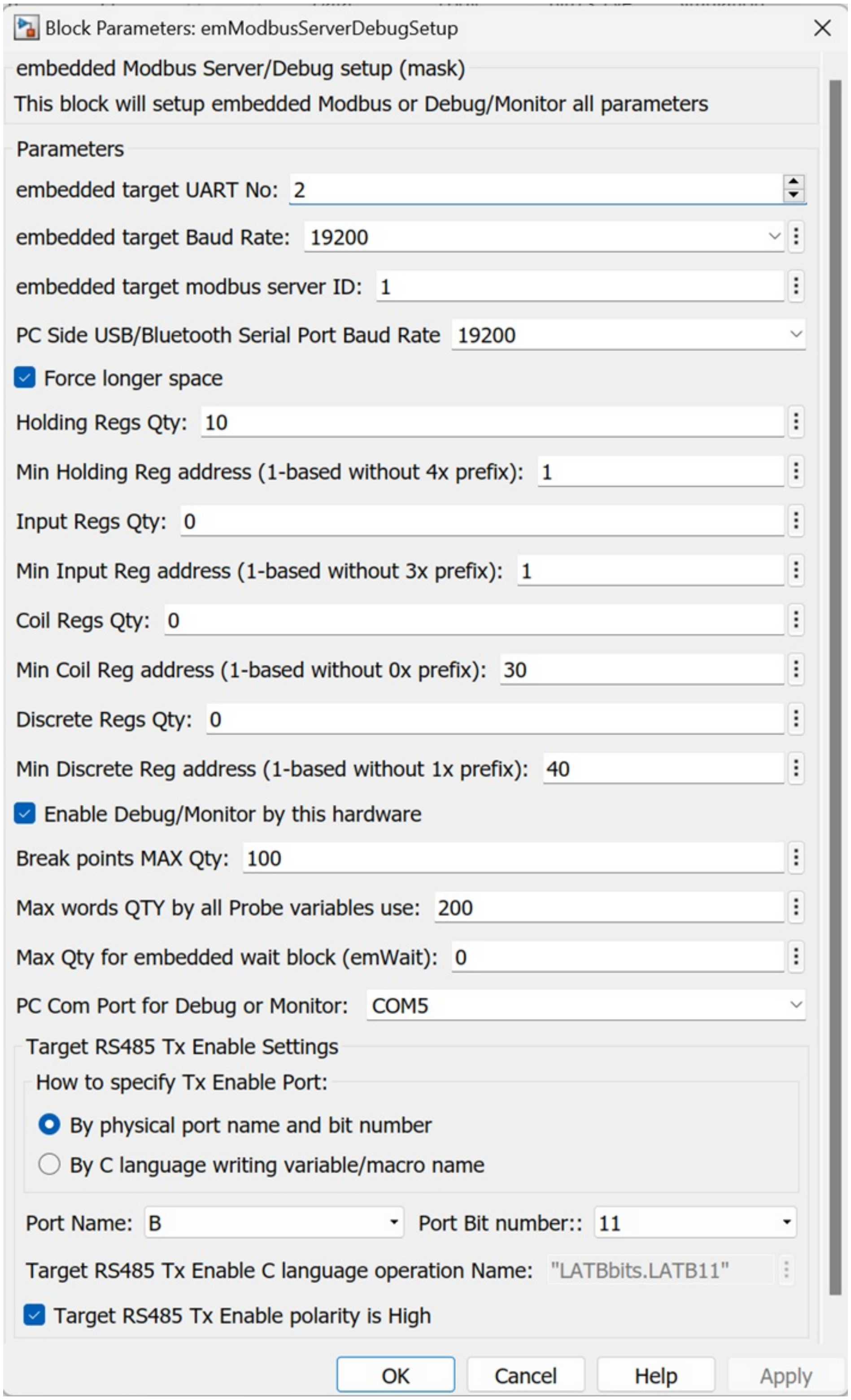
Our embedded platform is dsPIC33EP256GP502 . This MCU uses dual words writing for flash.

Please see screenshot of model below:

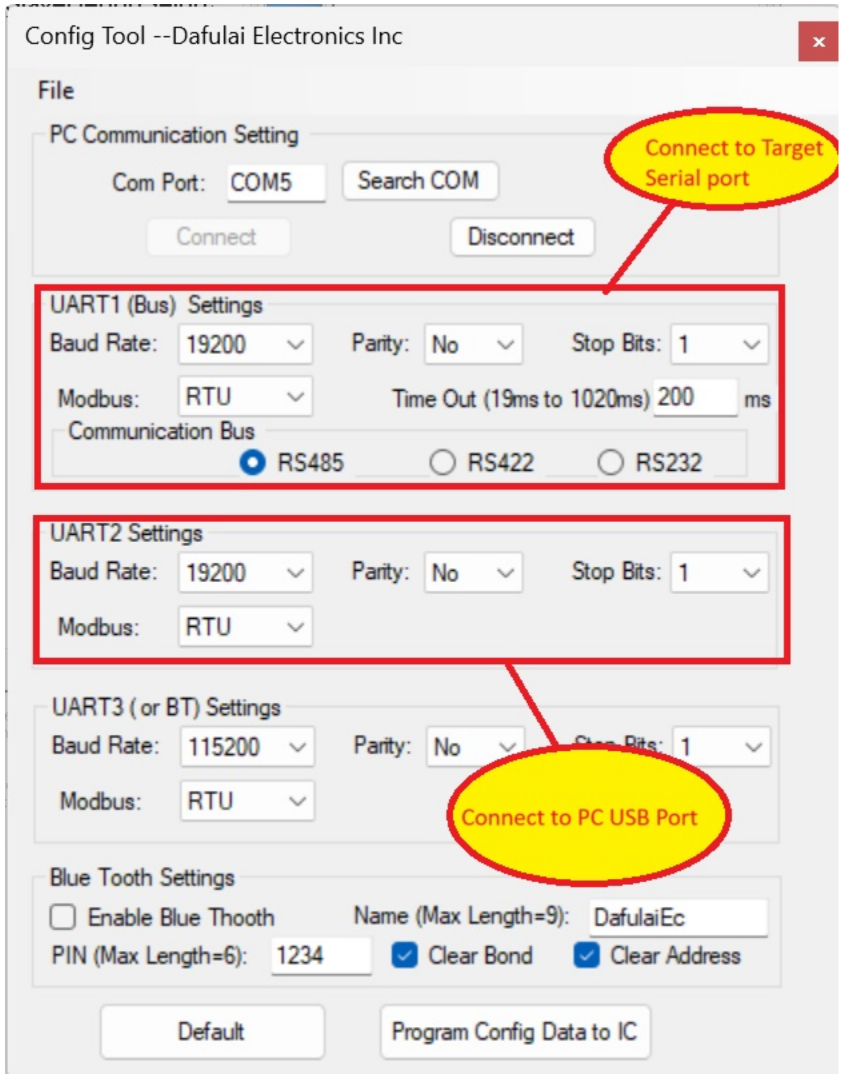


As you see above, in order to watch result, we used "probe" blocks. So we used "Modbus RTU/ASCII Dual Masters adaptor" as debugger hardware. This hardware connects Target Uart2, and Uart2 controls RS485, TX_transmit Enable is controlled by RB11. Logic High will enable RS485 transmit and disable RS485 receiver.

Please see our "emModbusServerDebugSetup" block settings below (only for debugging):

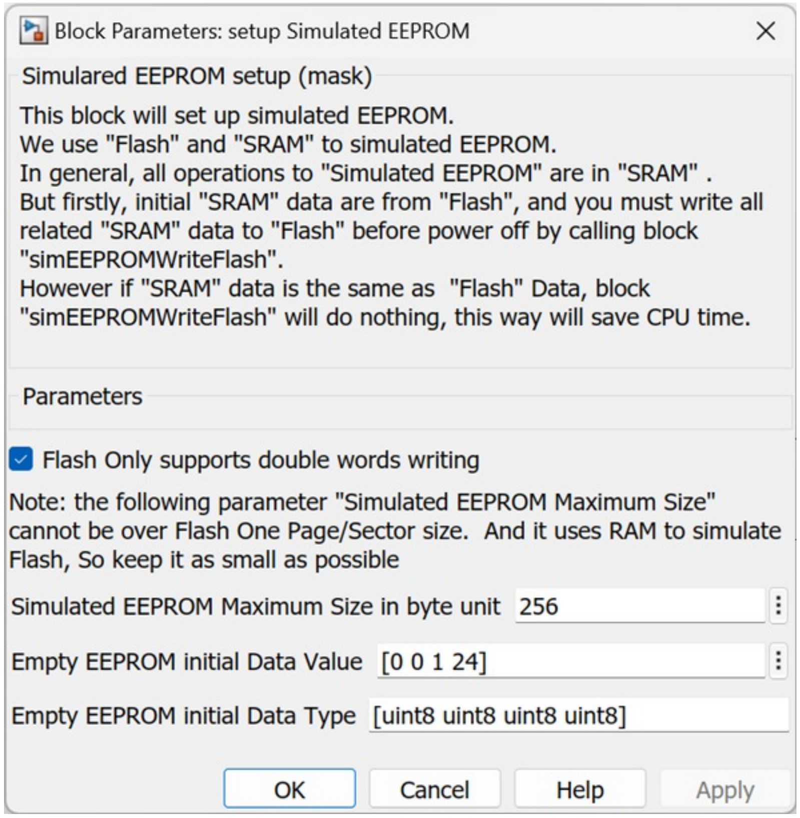


Please plug into "Modbus RTU/ASCII Dual Masters adaptor" to your PC USB Port. And if you are the first time to use this adaptor, run software "ConfigTool.exe" to config debug/monitor tool:



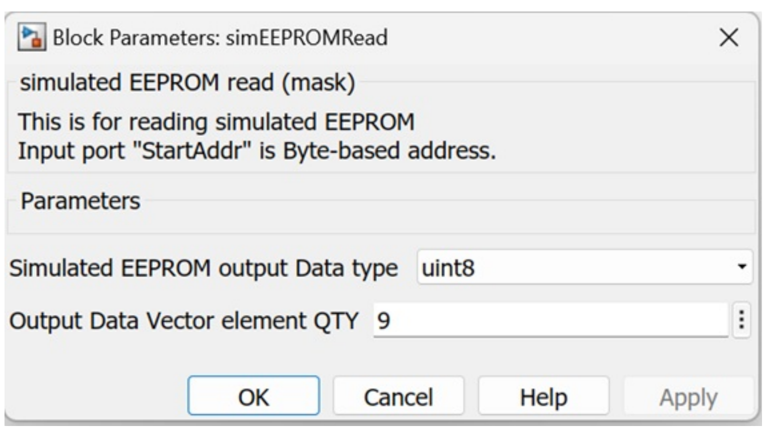
In above configuration, The first Uart setting must match Target including baud rate and physical interface (RS485/RS422/RS232). The second part setting can be different, but you must make sure parameter "PC Side USB/Bluetooth Serial Port Baud Rate" in "emModbusServerDebugSetup" block matches it.

Double click block "setup Simulated EEPROM", we will see parameters settings for "Simulated EEPROM" below:



It means Our MCU only supports dual words flash writing. And our Simulated EEPROM size is 256 bytes. The initial EEPROM values for empty EEPROM is [0 0 1 24]. which means 0, 0, 1, 24, 0, 0 ,0. 0, 0, 0, ..., 0.

Double click block "simEEPROMRead", we will see the block "simEEPROMRead" parameters settings below:

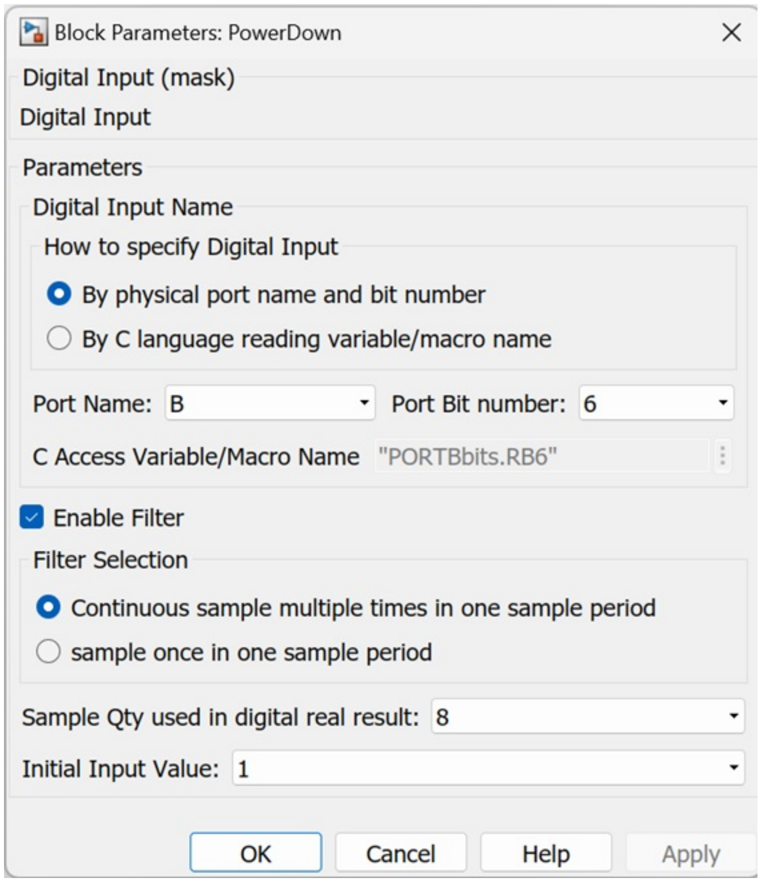


This block will read 9 bytes' data from address 0 (Input port "StartAddr") to address 8.

For block "simEEPROMWrite", it will write 3 bytes (input port "Data"): 11 12 and 13 (from constant block "data4write" which has uint8 type of constant vector output).

For block "simEEPROMWriteFlash", it will write flash if input port "En" = true and if never update new value to flash. "En" is from "PowerDown" block. When "PowerDown" Digital Pin is logic Low, it leads to "En" Logic high and enable "simEEPROMWriteFlash". If you have no power-down detection pin, you can enable "En" periodically when system is in stop state.

Double click on block "PowerDown", you will see parameters for this digital input:

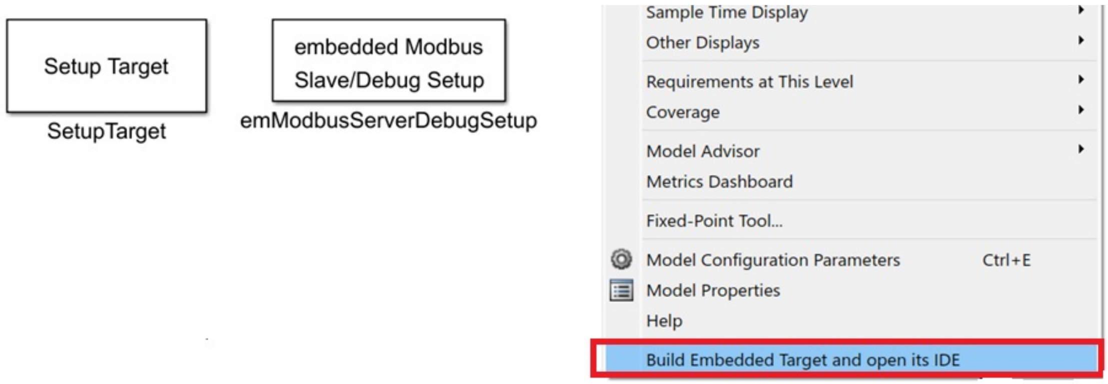


It means that Power down pin is Port B's bit 6, and we enable filter, initial Power down pin value is 1 which means "power on".

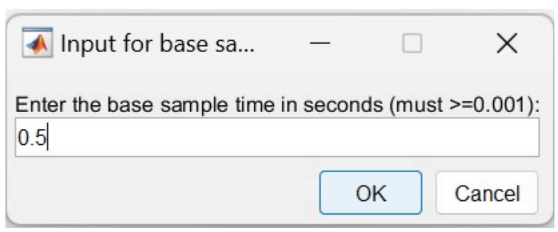
"Probe" blocks are just for watching "OK" state of "read/write" and watching read result data. For any block output except "Constant block", if its name does not contain "Right Arrow" *(→)*, you must use "Probe" block to watch its value.

Before you build this simulink model, you must create the firmware project by your IDE. and remember the firmware project directory name. In our example, it is microchip MPLAX IDE software, you must use MCC to configure timer1 as 1ms timer and interrupt enabled. You must use MCC to enable Uart2 with interrupt enabled and both "software Transmit Buffer Size" and "software Receive Buffer Size" =255 or 254. Timer1 interrupt priority is below UART (you can choose equal too). And you must config Flash. We put our MCC configuration file BasePrj.mc3 into example folder for your reference. You must modify according to your hardware. You'd better compile your firmware which is created by MCC to identify any error by MCC.

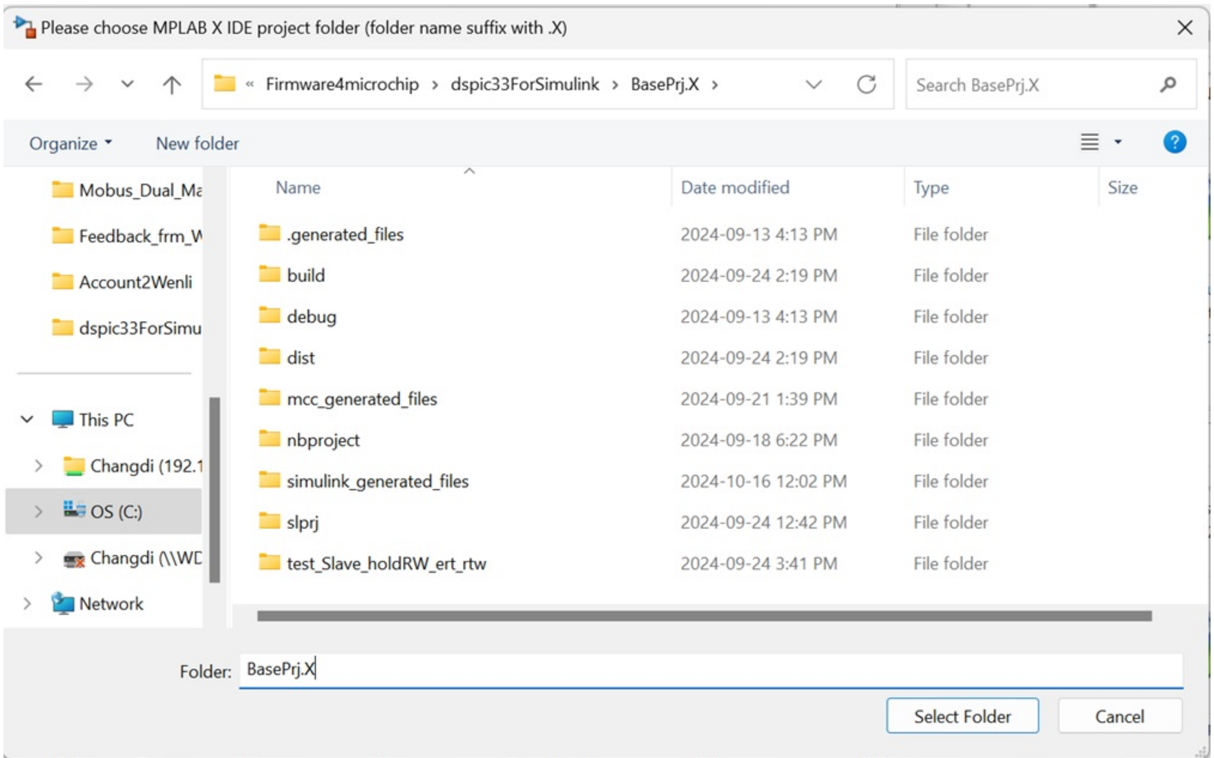
After your "Modbus RTU/ASCII Dual Masters adaptor" connects to Target (dspic33) and PC USB, right click on any empty space of simulink model, pop up context menu, click on menu item "Build Embedded Target and open its IDE"



It will start build, and popup dialog window to input base sampling period:

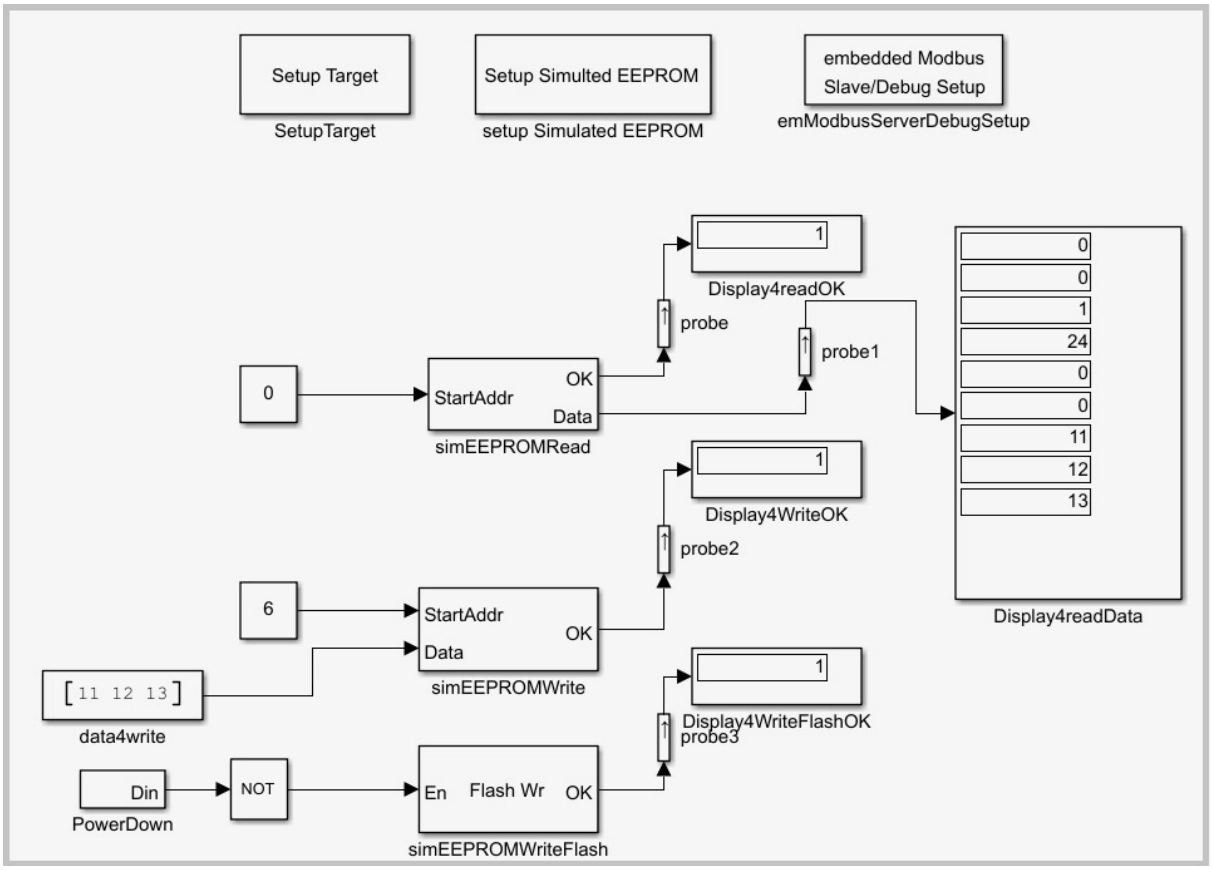


Input your base sample time and click OK. If any error occurs, it will stop building, and display error information. The error block will display in Yellow color. If build successfully, it will popup window to ask you firmware directory for your IDE project.



If you give out the correct IDE project directory, Simulink will open IDE software automatically. And you can compile firmware in your IDE, and program into Target by emulator IDE supported.

We can use Simulink directly view results. By select stop time= inf, and click "Run" button, you will see result below:



From above "Display4readOK" block, its value is 1 (true), so "Read EEPROM success". "Display4WriteOK" block displays "1" (true), so "Write EEPROM success". "Display4WriteFlashOK" block displays "1" (true), so "Write Flash success". "Display4WriteFlashOK" block displays " 0, 0, 1, 24, 0, 0 , 11, 12,13", it is what to expect.

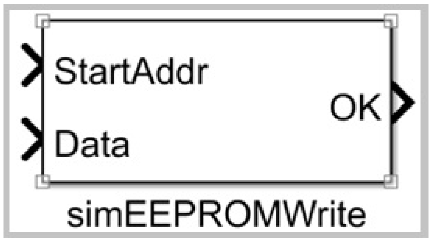
" 0, 0, 1, 24, 0, 0" is from parameter "Empty EEPROM initial byte Data" of "set up Simultated EEPROM" block. "11, 12,13" is from block "data4write".

Please open "Your embedded creator library folder"/examples/example8_simEEPROM.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

simEEPROMWrite

Write simulated EEPROM
Since R2020b

Library: embeddedCreatorLib (Dafulai Electronics) / Simulate EEPROM by Flash / simEEPROMWrite



Description

This block write "Simulated EEPROM". Actually it only write to "SRAM", it didn't write to "Flash". You must call block "simEEPROMWriteFlash" to do actual writing to flash before power off.

Parameters

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- StartAddr — "uint16" data type's scalar. It is "Simulated EEPROM" start address you want to write. This address is byte addressing.

- Data — Scalar or Vector. It is data source for writing. Data type cannot be "logical", cannot be "double".

Outport

- OK — "logical" data type's scalar. True means "Simulated EEPROM" address range you write is valid. Writing data success.

Examples

Example1:

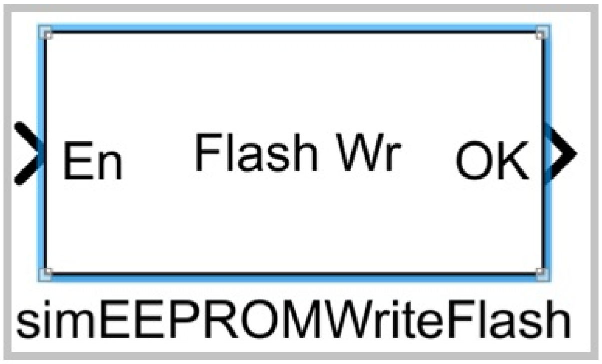
Please see example in block "simEEPROMRead".

Please open "Your embedded creator library folder"/examples/example8_simEEPROM.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

simEEPROMWriteFlash

Write simulated EEPROM Flash
Since R2020b

Library: embeddedCreatorLib (Dafulai Electronics) / Simulate EEPROM by Flash / simEEPROMWriteFlash



Description

This block writes Flash for "Simulated EEPROM". If all "SRAM" data for "Simulated EEPROM" are the same as "Flash" data, this block will not do actual writing to flash but output port "OK" still be true. This way will save CPU time because Programming flash needs lots of time (Erase page/sector and writing to page/sector), and during programming to flash, all interrupts will be disabled. If any of "SRAM" data for "Simulated EEPROM" is different from "Flash" data, when "En"= true, this block will disable all "interrupts" and "erase/program flash page or sector".

Parameters

- Sample time in sec (-1 for inherited): — Sample time for this block. It is the same meaning as general Simulink block .

Ports

Input

- En — "logical" data type's scalar. True means "Do writing to flash". False means "Disable writing to flash"

Outport

- OK — "logical" data type's scalar. True means "Writing Flash successfully and verify pass".

Examples

Example1:

Please see example in block "simEEPROMRead".

Please open "Your embedded creator library folder"/examples/example8_simEEPROM.slx (You must change "PC Com Port for Debug or Monitor" in emModbusServerDebugSetup block according to your physical USB port number)

9 Notice

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