

Setup of Modbus RTU- and serial communication in CODESYS | UR20-1COM-232-485-422

Abstract:

The Application Note describes how to set up a Modbus RTU- and a serial communication with a u-remote UR20-1COM-232-485-422 device in CODESYS. The examples show use of the UR20-1COM-232-485-422 via the System_Bus of a UC20-WL2000, but also via a fieldbus coupler.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UC20-WL2000-AC	1334950000	u-OS 2.0.0
2	IoT-GW30	2682620000 2682630000	u-OS 2.0.0
3	UR20-1COM-232-485-422	1315750000	FW 1.00.16

Software reference

No.	Software name	Article No.	Software version
1	CODESYS Development System		SP18 Patch 4
2	CODESYS Runtime App		4.7.0.0-2
3	CODESYS Control SL for Weidmueller u-OS		4.7.0.0

Contact

Weidmüller Interface GmbH & Co. KG
Klingenbergstraße 26
32758 Detmold, Germany
www.weidmueller.com

For any further support please contact your
local sales representative:
<https://www.weidmueller.com/countries>

Content

1 Warning and Disclaimer..... 4

2 Introduction..... 5

3 Install Modbus RTU and serial library package 6

4 FB_ModbusRTUMaster 7

5 FB_UR20_1ComRs_232_485_422_Control 8

6 Create a Modbus RTU or serial communication..... 9

6.1 Modbus RTU or serial communication via System_Bus..... 9

6.2 Modbus RTU or Serial Communication via u-remote coupler.....10

6.3 Import and use of function block13

6.4 Example Project14

1 Warning and Disclaimer

Warning

Controls may fail in unsafe operating conditions, causing uncontrolled operation of the controlled devices. Such hazardous events can result in death and / or serious injury and / or property damage. Therefore, there must be safety equipment provided / electrical safety design or other redundant safety features that are independent from the automation system.

Disclaimer

This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start Guide / Example Program prepared by Weidmüller, you accept that Weidmüller cannot be held liable for any damage to property and / or personal injury that may occur because of the use.

Note

The given descriptions and examples do not represent any customer-specific solutions, they are simply intended to help for typical tasks. The user is responsible for the proper operation of the described products. Application notes / Quick Start Guides / Example Programs are not binding and do not claim to be complete in terms of configuration as well as any contingencies. By using this Application Note / Quick Start Guide / Example Program, you acknowledge that we cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this application note / quick start guide / example at any time without notice. In case of discrepancies between the proposals Application Notes / Quick Start Guides / Program Examples and other Weidmüller publications, like manuals, such contents have always more priority to the examples. We assume no liability for the information contained in this document. Our liability, for whatever legal reason, for damages caused using the examples, instructions, programs, project planning and performance data, etc. described in this Application Note / Quick Start Guide / Example is excluded.

Security notes

In order to protect equipment, systems, machines and networks against cyber threats, it is necessary to implement (and maintain) a complete state-of-the-art industrial security concept. The customer is responsible for preventing unauthorized access to his equipment, systems, machines and networks. Systems, machines and components should only be connected to the corporate network or the Internet if necessary and appropriate safeguards (such as firewalls and network segmentation) have been taken.

2 Introduction

This application note describes the individual steps for implementing both a Modbus RTU- and a serial communication between a u-OS PLC with a u-remote UR20-1COM-232-485-422 module and a PC. The PC connects via a USB<->RS485- or USB<->RS232 adapter stick via and runs a simple serial terminal or a simulation program of a Modbus RTU slave device. We recommend using the free tool ModRssim2 <https://sourceforge.net/projects/modrssim2/>.

For the serial communication we recommend the free tool hterm, see <https://www.der-hammer.info/pages/terminal.html> on Windows or cutecom on Linux.

3 Install Modbus RTU and serial library package

1. Download the current CODESYS library files from the Weidmueller support center.
https://support.weidmueller.com/support-center/search/results?query=CODESYS&ac=library%26functionblock_id
2. Unzip the download and start the installation of the package via double click.
3. Click on the checkbox on the left bottom and start the installation.
4. Open a new standard CODESYS project and add the installed library inside the project.
In the device tree ⇒ Library Manager ⇒ Add Library... ⇒ libWiUr20ModbusRTUMaster + libWiUr20ComRs_232_485_422.

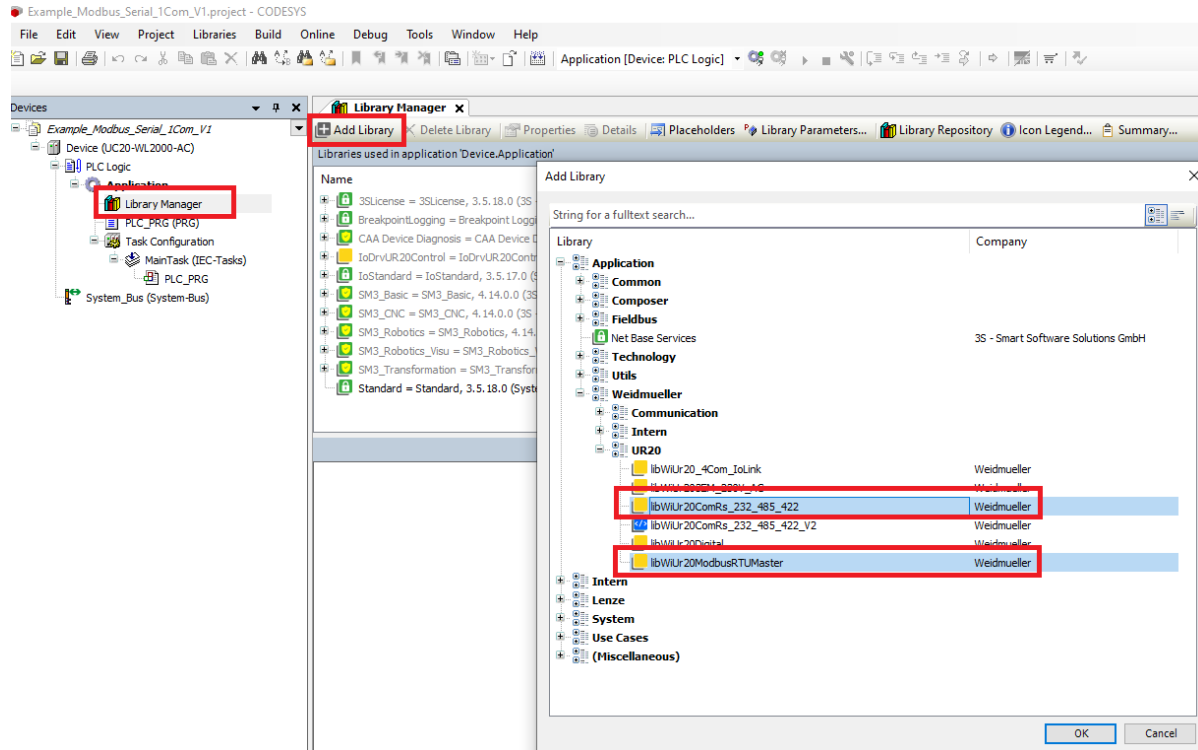


Figure 1: Add Library



Note:

You can find the documentation of every function block in the downloaded zip file. WI recommends reading the documentation before use.

4 FB_ModbusRTUMaster

The function block implements a Modbus RTU master that provides communication with a Modbus RTU slave via UR20-1COM-232-485-422.

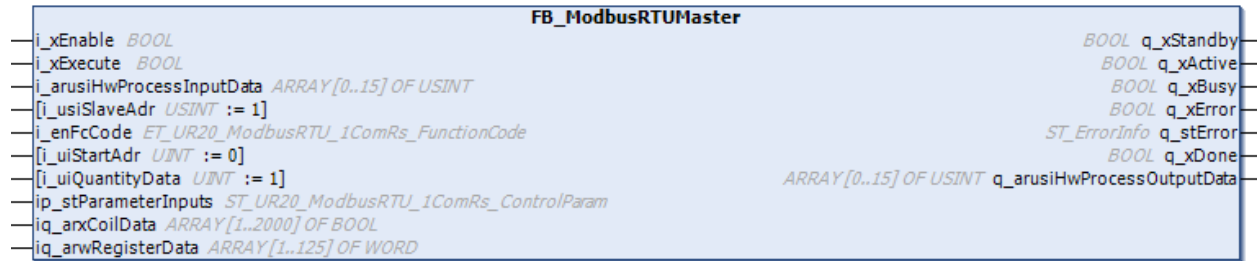


Figure 2: FB Modbus Master

The behavior, functionality and the inputs/outputs of the function block are explained in the documentation of the function block.

Please find the information also directly in the function block preview in the library manager.

Contents of selected library 'libWU20ModbusRTUMaster, 1.1.2.3 (Weidmüller)'

- 10_Enums
- 20_Structs
- 40_Function_Blocks
 - FB_ModbusRTUMaster

FB_ModbusRTUMaster (FB)

FUNCTION_BLOCK FB_ModbusRTUMaster EXTENDS FB_LevelControlledFiniteBehavior

Functional Description

The function block FB_ModbusRTUMaster implements a Modbus RTU Master that provides communication with a Modbus slave via u-remote Module UR20-1COM-232-485-422.

Supported Functions

Function Code	Register Type	Explanation
1	Read Coil Status	Reads binary outputs (coils) from a connected slave. The data is stored in Array iq_arxCoilData[1..2000] of BOOL.
2	Read Input Status	Reads binary inputs from a connected slave. The data is stored in Array iq_arxCoilData[1..2000] of BOOL.
3	Read Holding Registers	Reads register-data from a connected slave. The data is stored in Array iq_arwRegisterData[1..125] of WORD.
4	Read Input Registers	Reads input registers from a connected slave. The data is stored in Array iq_arwRegisterData[1..125] of WORD.
5	Write Single Coil	Sends a binary output (Coil) to a connected slave. The value from array iq_arxCoilData[1] is written to the slave.
6	Write Single Register	Sends a single data word to a connected slave. The value from array iq_arwRegisterData[1] is written to the slave.
8	Diagnostics	Sends a diagnostics request with a user defined subfunction code to a connected slave. The subfunction code is passed to the function by the parameter i_uiStartAdr. Additional data can be passed by Array iq_arwRegisterData[1].
15	Write Multiple Coils	Sends several binary outputs (Coils) to a connected slave. The data must be provided in array iq_arxCoilData[1..2000] of BOOL. The maximum number of coils are 0x07B0.
16	Preset Multiple Registers	Sends data to a connected slave. The data must be provided in array arwRegisterData[1..125] of WORD. The maximum number of data are 0x007B.

Possible Errors

Figure 3: function block documentation

5 FB_UR20_1ComRs_232_485_422_Control

The function block sends or receives data either on *RS232*, *RS485* or *RS422*.

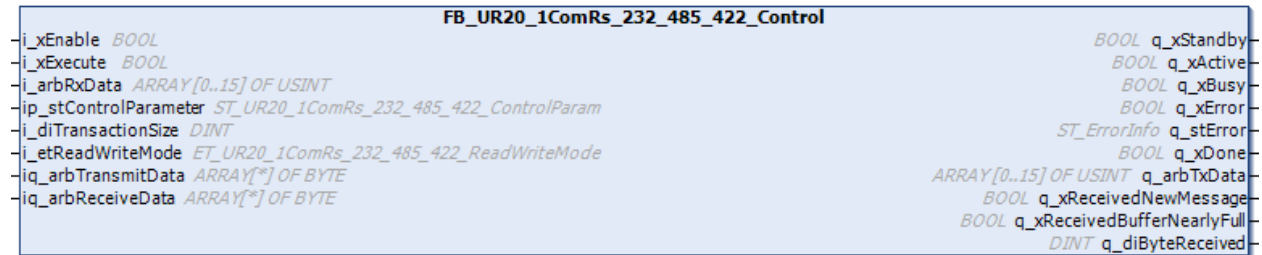


Figure 4: FB for serial communication

The behavior, functionality and the inputs / outputs of the function block are explained in the documentation of the function block.

This documentation also is directly available in the library repository when you open the function block.

This video:

[Introduction to u-remote function block libraries for Codesys - YouTube](#)

is also a good source of information. The Video explains how you install a library, how the function blocks behave and how to use them.

6 Create a Modbus RTU or serial communication

This chapter explains how to create a Modbus RTU or a serial communication.

It is essential to know the parameters and settings of the connected devices. You need to set up the communication parameters of the UR20-1COM-232-485-422 accordingly. In the example we use the following parameters:

Example:

Operating mode:	RS485
Baud rate:	9600 kbps
Parity:	none
Stop bits:	1
Flow control:	none
Data bits:	8 bit
Terminating resistor:	On

6.1 Modbus RTU or serial communication via System_Bus

1. In the Device Tree, open the System_Bus configurator either by double-clicking the System_Bus or by right-click on the System_Bus and the menu item Add Device. Add the UR20_1COM_232_485_422 module to the System_Bus.
2. Double-click the UR20_1COM_232_485_422, open Module Configuration and adjust its communication parameters.

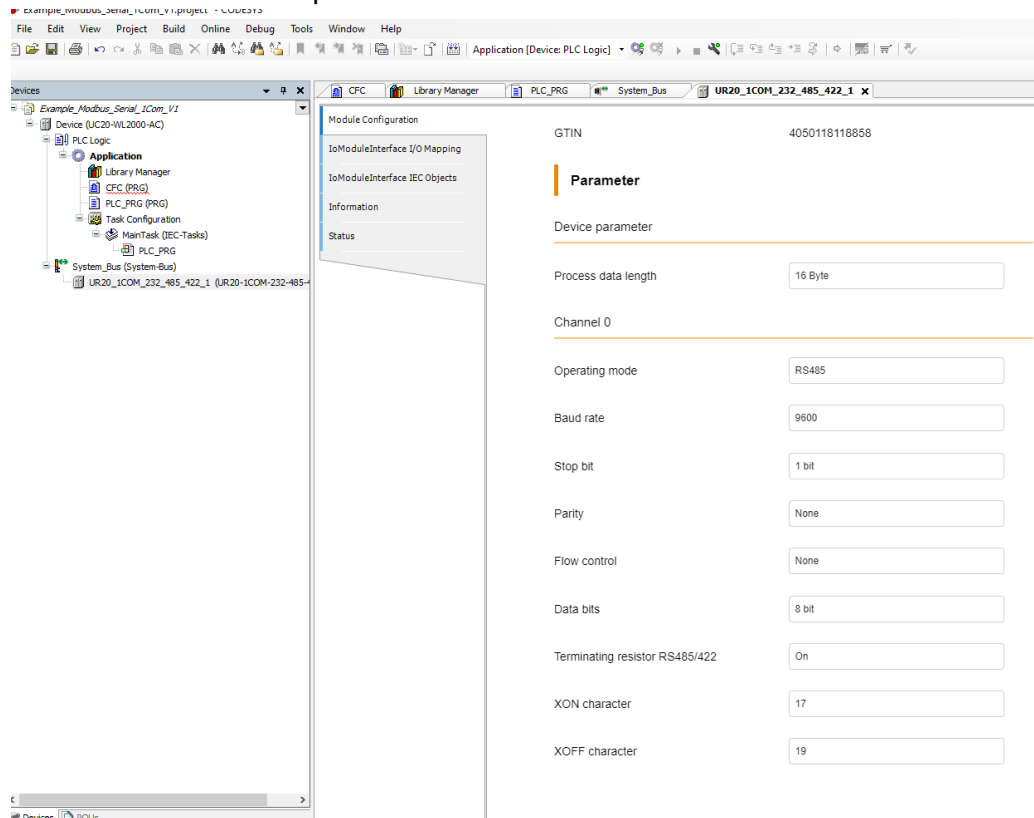


Figure 5: Communication Parameter

- Map variables to the IO of the UR20_1COM_232_485_422. To do so, select IoModuleInterface I/O Mapping in the module representation. Then map one variable after the other to the channels: Click into the variable field and select the variable you want to associate with this channel. See Figure 6: Variable <-> I/O Channel mapping.



Note:

Theoretically, the CODESYS ST syntax allows to associate variables to I/O channels by assigning an absolute address to a variable, e.g. `i_arusiMBInput AT%IB19 : ARRAY[0..15] OF USINT`; However, this is not robust against changes of the station setup. CODESYS has declared this syntax deprecated and WI advises NOT to use it.

Variable	Mapping	Channel	Address	Type	Unit
Application.GVL_IO.i_arusi1COMInputDataSysBus[0]	Input	Input	%IB19	BYTE	
Application.GVL_IO.i_arusi1COMInputDataSysBus[1]	RX		%IB20	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[2]		Received data 1	%IB21	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[3]		Received data 2	%IB22	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[4]		Received data 3	%IB23	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[5]		Received data 4	%IB24	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[6]		Received data 5	%IB25	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[7]		Received data 6	%IB26	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[8]		Received data 7	%IB27	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[9]		Received data 8	%IB28	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[10]		Received data 9	%IB29	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[11]		Received data 10	%IB30	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[12]		Received data 11	%IB31	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[13]		Received data 12	%IB32	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[14]		Received data 13	%IB33	USINT	
Application.GVL_IO.i_arusi1COMInputDataSysBus[15]		Received data 14	%IB34	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[0]	Output	Output	%QB18	BYTE	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[1]	TX_Byte_CNT		%QB19	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[2]		Transmission data 1	%QB20	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[3]		Transmission data 2	%QB21	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[4]		Transmission data 3	%QB22	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[5]		Transmission data 4	%QB23	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[6]		Transmission data 5	%QB24	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[7]		Transmission data 6	%QB25	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[8]		Transmission data 7	%QB26	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[9]		Transmission data 8	%QB27	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[10]		Transmission data 9	%QB28	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[11]		Transmission data 10	%QB29	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[12]		Transmission data 11	%QB30	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[13]		Transmission data 12	%QB31	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[14]		Transmission data 13	%QB32	USINT	
Application.GVL_IO.q_arusi1COMOutputDataSysBus[15]		Transmission data 14	%QB33	USINT	

Figure 6: Variable <-> I/O Channel mapping

6.2 Modbus RTU or Serial Communication via u-remote coupler



Note:

Read the u-remote handbook for information on how to set up the parameter values of the UR20_1COM_232_485_422 via an u-remote fieldbus coupler. Here is a download link to the [u-remote handbook](#).

This example requires the CODESYS EtherCAT package and the device description files for the WI EtherCAT fieldbus coupler. Please install the EtherCAT package via the CODESYS Installer. Download and install the EtherCAT device description files from WI's [product homepage](#). => Downloads => Software => current ESI file for your HW version.

1. In the Device Tree of your CODESYS project, take the following steps:
 - a. Right-click the Device, i.e. your PLC and select “Online Config Mode”.
 - b. In the EtherCAT master => General => EtherCAT NIC settings, click “Select...” to the right of the Source address (MAC). Assuming you have connected X2 of your UC20-WL2000-AC with the EtherCAT coupler, select eth1.
 - c. double-click the EtherCAT fieldbus coupler and select *Add Device* to add the UR20_1COM_232_485_422 module to the fieldbus coupler.
 - d. Set up the communication parameters in the Startup Parameters of the EtherCAT coupler:
 - i. open UR20_PBC_EC => Startup Parameters => Add => Parameter UR20_1COM_232_485_422. See Figure 7: Add Parameter.
 - ii. Select the parameters to adjust and click ok.
 - iii. Edit the values for the desired communication.
 - iv.

Figure 7: Add Parameter

Please find an overview of module parameters in the u-remote manual. Figure 8: Module Parameter shows these for the UR20_1COM-_232_485_422 module.

Overview of the editable parameter UR20-1COM-232-485-422

Description	Options ¹⁾	Default
Operating mode	disabled (0) / RS232 (1) / RS485 (2) / RS422 (3)	disabled
Data bits ²⁾	7 Bit (0) / 8 Bit (1)	8 Bit
Baud rate	300 (0) / 600 (1) / 1200 (2) / 2400 (3) / 4800 (4) / 9600 (5) / 14400 (6) / 19200 (7) / 28800 (8) / 38400 (9) / 57600 (10) / 115200 (11)	9600
Stop bit	1 Bit (0) / 2 Bit (1)	1 Bit
Parity	None (0) / Even (1) / Odd (2)	None
Flow control	None (0) / CTS/RTS (1) / XON/XOFF (2)	None
XON character	0 ... 255	17
XOFF character	0 ... 255	19
Terminating resistor RS485/422	Off (0) / On (1)	Off
Process data length	8 Byte ³⁾ (0) / 16 Byte (1)	16 Byte

1) Values in brackets for Modbus-TCP (firmware version 02.00.00 and higher), CANopen, EtherCAT and EtherNet/IP via module parameter class

2) The option "7 Bit" works only in combination with a parity ("even" or "odd")

3) CANopen and DeviceNet only

Figure 8: Module Parameter

The process data is mapped exactly in the same way as when using the 1COM module via the System_Bus, see Figure 9: EtherCAT I/O <-> variable mapping.

UR20_1COM_232_485_422_1

Module Configuration

IoModuleInterface I/O Mapping

IoModuleInterface IEC Objects

Information

Status

Find

Filter

Show all

+

Add FB for I/O Channel...

Variable	Mapping	Channel	Address	Type
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[0]</div></div>	<div><div>+</div><div>Input</div></div>	Input	%IB19	BYTE
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[1]</div></div>	<div><div>+</div><div>RX</div></div>	RX	%IB20	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[2]</div></div>	<div><div>+</div><div>Received data 1</div></div>	Received data 1	%IB21	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[3]</div></div>	<div><div>+</div><div>Received data 2</div></div>	Received data 2	%IB22	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[4]</div></div>	<div><div>+</div><div>Received data 3</div></div>	Received data 3	%IB23	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[5]</div></div>	<div><div>+</div><div>Received data 4</div></div>	Received data 4	%IB24	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[6]</div></div>	<div><div>+</div><div>Received data 5</div></div>	Received data 5	%IB25	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[7]</div></div>	<div><div>+</div><div>Received data 6</div></div>	Received data 6	%IB26	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[8]</div></div>	<div><div>+</div><div>Received data 7</div></div>	Received data 7	%IB27	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[9]</div></div>	<div><div>+</div><div>Received data 8</div></div>	Received data 8	%IB28	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[10]</div></div>	<div><div>+</div><div>Received data 9</div></div>	Received data 9	%IB29	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[11]</div></div>	<div><div>+</div><div>Received data 10</div></div>	Received data 10	%IB30	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[12]</div></div>	<div><div>+</div><div>Received data 11</div></div>	Received data 11	%IB31	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[13]</div></div>	<div><div>+</div><div>Received data 12</div></div>	Received data 12	%IB32	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[14]</div></div>	<div><div>+</div><div>Received data 13</div></div>	Received data 13	%IB33	USINT
<div><div>+</div><div>Application.GVL_IO.i_arusi1COMInputDataSysBus[15]</div></div>	<div><div>+</div><div>Received data 14</div></div>	Received data 14	%IB34	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[0]</div></div>	<div><div>+</div><div>Output</div></div>	Output	%QB18	BYTE
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[1]</div></div>	<div><div>+</div><div>TX_Byte_CNT</div></div>	TX_Byte_CNT	%QB19	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[2]</div></div>	<div><div>+</div><div>Transmission data 1</div></div>	Transmission data 1	%QB20	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[3]</div></div>	<div><div>+</div><div>Transmission data 2</div></div>	Transmission data 2	%QB21	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[4]</div></div>	<div><div>+</div><div>Transmission data 3</div></div>	Transmission data 3	%QB22	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[5]</div></div>	<div><div>+</div><div>Transmission data 4</div></div>	Transmission data 4	%QB23	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[6]</div></div>	<div><div>+</div><div>Transmission data 5</div></div>	Transmission data 5	%QB24	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[7]</div></div>	<div><div>+</div><div>Transmission data 6</div></div>	Transmission data 6	%QB25	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[8]</div></div>	<div><div>+</div><div>Transmission data 7</div></div>	Transmission data 7	%QB26	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[9]</div></div>	<div><div>+</div><div>Transmission data 8</div></div>	Transmission data 8	%QB27	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[10]</div></div>	<div><div>+</div><div>Transmission data 9</div></div>	Transmission data 9	%QB28	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[11]</div></div>	<div><div>+</div><div>Transmission data 10</div></div>	Transmission data 10	%QB29	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[12]</div></div>	<div><div>+</div><div>Transmission data 11</div></div>	Transmission data 11	%QB30	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[13]</div></div>	<div><div>+</div><div>Transmission data 12</div></div>	Transmission data 12	%QB31	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[14]</div></div>	<div><div>+</div><div>Transmission data 13</div></div>	Transmission data 13	%QB32	USINT
<div><div>+</div><div>Application.GVL_IO.q_arusi1COMOutputDataSysBus[15]</div></div>	<div><div>+</div><div>Transmission data 14</div></div>	Transmission data 14	%QB33	USINT

Figure 9: EtherCAT I/O <-> variable mapping

6.3 Import and use of function block

1. Add the Modbus RTU library libWiUr20ModbusRTUMaster to your PLC project if you want to create a ModBus RTU communication or add the libWiUr20ComRs_232_485_422 library to your PLC project if you want to create a serial communication.
2. Create a new program and use the function block needed in your application: For Modbus RTU communication “FB_ModbusRTUMaster” and for serial communication “FB_UR20_1ComRs_232_485_422_Control”.
3. Create a program sequence to execute the function block.

WI provides a CODESYS example project that demonstrates usage of the relevant function blocks in libWiUR20ModbusRTUMaster and libWiUR20ComRs_232_485_422. The next chapter introduces this example project.

6.4 Example Project

The example project offers four demonstration programs for the following application cases, see Figure 10: Example Project: In the example shown the program for Modbus RTU via System_Bus is activated.

- Modbus RTU via EtherCAT in PRG_ModbusCommunication_Ethercat
- Serial communication via EtherCAT in PRG_SerialCommunication_Ethercat
- Modbus RTU via System_Bus in PRG_ModbusCommunication_Sysbus
- Serial communication via System_Bus in PRG_SerialCommunication_Sysbus

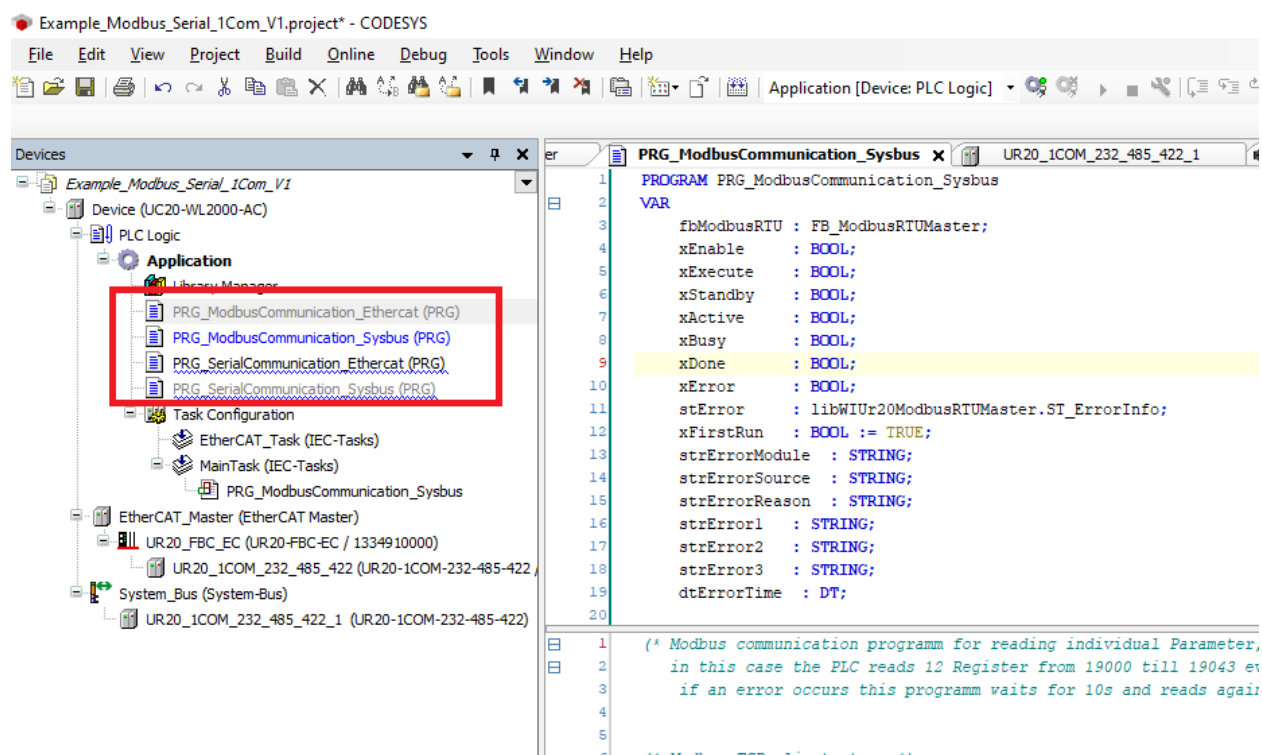


Figure 10: Example Project

To activate the desired program for your use case add it to the MainTask: In the device tree ⇒ open Task Configuration ⇒ MainTask ⇒ Add Call ⇒ Select your program ⇒ ok. Similarly, you find “Remove Call” to exclude a program from execution, again. See Figure 11: Add or Remove Call.



Note:

Multiple programs may not access the same module simultaneously because they overwrite each other's module I/O data. Therefore, in the example project, you may use one program only for the system bus and one program only for EtherCAT at the same time.

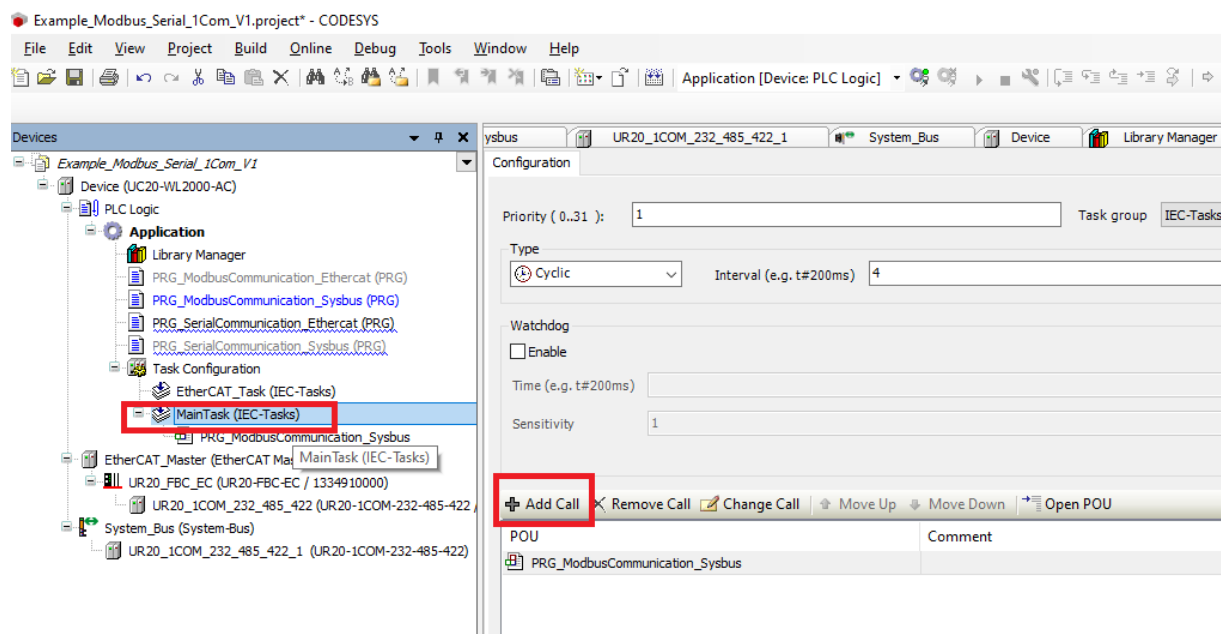


Figure 11: Add or Remove Call

Modbus RTU examples:

These examples demonstrate how to run a ModBus RTU master in a UC20_WL2000_AC u-OS CODESYS application.

The demo programs are copies of each other with just one difference: The variables mapping the fbModBusRTU process image to the UR20_1COM_RS232_485_422 module depend on which field bus the example uses. Just drag the desired demo program into the MainTask and remove the other from the MainTask. See Figure 11: Add or Remove Call.

Configure the operation mode and communication settings in the parametrization of the UR20_1COM_232_485_422 module. Download the PLC program and login.

The ModBus RTU examples require a ModBus RTU slave as communication counterpart. WI suggests that you use an USB<->485- or USB<->RS232 adapter and a Modbus simulator on your PC, e.g. ModRSSim2 for Windows. If you know an open-source ModBus RTU slave simulator for *Linux*, please contact application.automation@weidmueller.com with this information.

Open the ModRSSim2 simulator and write some values to the registers. Start the PLC application. The ModBus RTU demo program of your choice requests some holding registers from the connected ModRSSim2 ModBusRTU slave simulation. It stores them in the array "arOutputDataRegister". Change values in ModRSSim2 and observe the changes in arOutputDataRegister in the demo program. See Figure 12: Modbus Simulator.

Setup of Modbus RTU- and serial communication in CODESYS | UR20-1COM-232-485-422

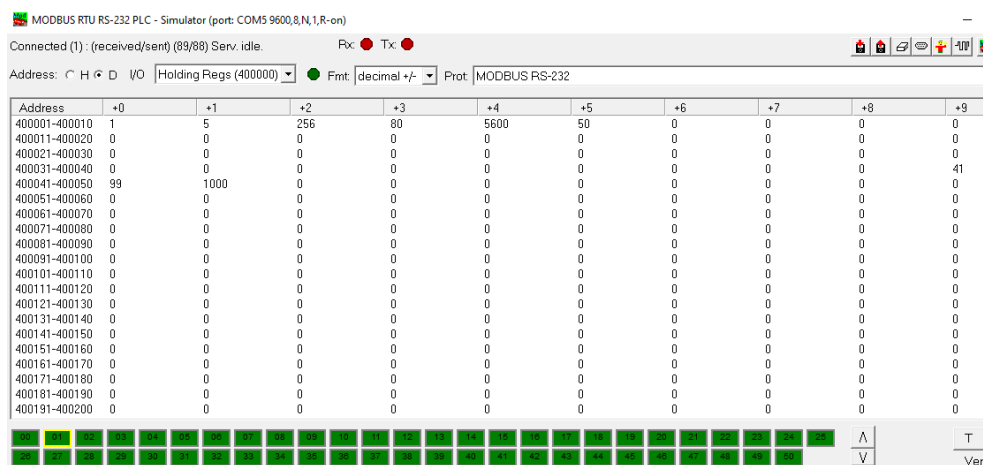


Figure 12: Modbus Simulator

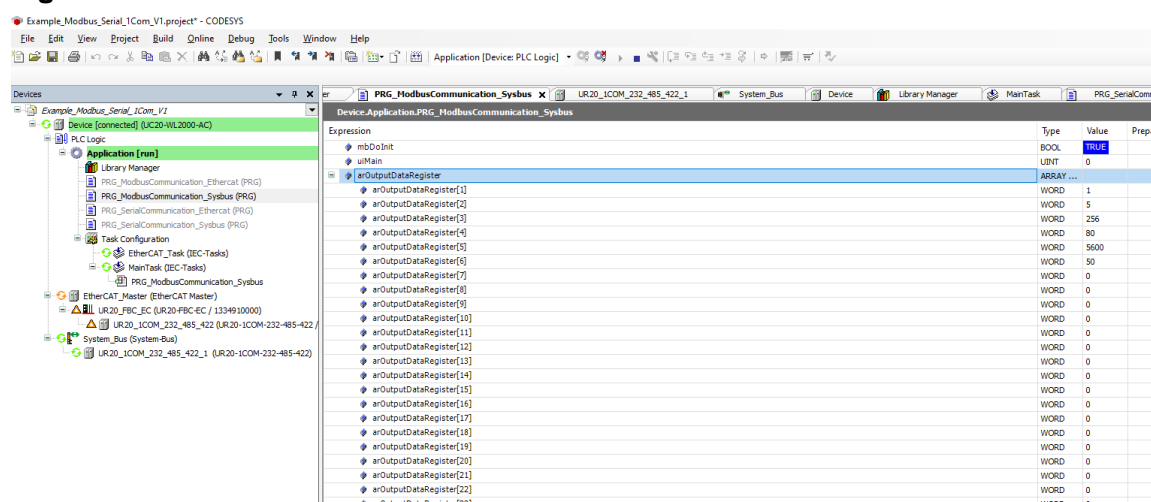


Figure 13: Read Values via System_Bus

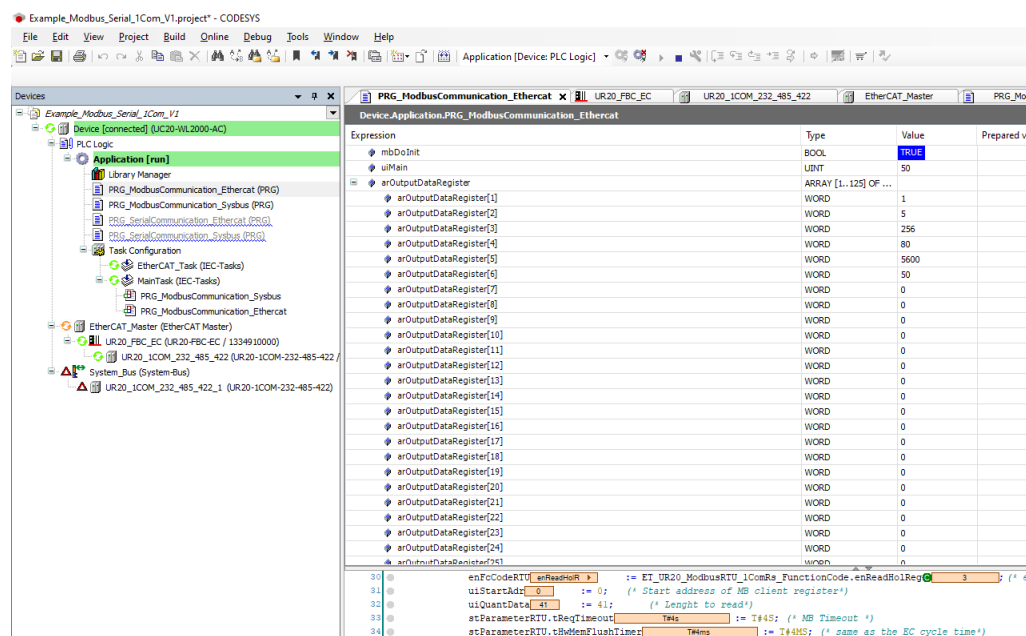


Figure 14: Read Values via EtherCAT

Serial communication:

The programs PRG_SerialCommunication_EtherCAT and PRG_SerialCommunication_Sysbus demonstrate serial communication. They, too, are nearly identical copies of each other. Just the mapping of fbSerial's I/O data is different, they select either EtherCAT or the System_Bus.

The PRG_SerialCommunication_<Fieldbus> implements a simple state machine with the state variable etMain : ET_SerialComStates; This state machine initializes the fbSerial and puts it into its state STANDBY. Then it waits for incoming serial messages or for the user to trigger a send serial data sequence. After it has finished receiving or sending serial data, it enters the state ET_SerialComStates.FINISHED.

:

1. Select the PRG_SerialCommunication_<Fieldbus> of your choice.
2. Remove any other PRGs from the MainTask.
3. Download and login to the PLC.
4. WI recommends connecting the UR20_1COM_232_485_422 to a USB to serial converter and run e.g. hterm on a Windows PC or CuteCom on a Linux PC to view and input the incoming and outgoing serial data.
5. Start the application.
6. Force xStart to TRUE and unforce it.

The program waits in its state ET_SerialComStates.FB_IN_STANDBY until data is available.

7. Send some serial data, e.g. "Hello!" via the terminal program of your choice.

The serial communication FB signals available received data and the program enters the state ET_SerialComStates.FB_ACTIVATED. It transfers the received data to the variables arReceiveDataSave and sTextOutput. Then it sets xStart back to FALSE.

8. Observe the received data in arReceiveDataSave and sTextOutput.