R-IN32M3 Module (RY9012A0)

Modbus TCP Start-Up Manual

R30AN0406EJ0200
Rev.2.00
Aug.05.2022

Introduction

This document describes sample software for Modbus TCP communication with R-IN32M3 Module (RY9012A0).

Target Device

R-IN32M3 Module (RY9012A0)

Contents

1. Overview

This document describes Modbus TCP protocol stack running on host MCU connected to R-IN32M3 Module, and provides the functional outline for developing and implementing user applications with the protocol stack, APIs, and sample programs.

This package supports Ethernet-based Modbus TCP slave.

1.1 Features

The Modbus protocol is a communication protocol developed by Modicon Inc. (Schneider Electric SA.) for programmable logic controllers (PLC), and the specification is published. Refer to protocol specifications (PI-MBUS-300 Rev.J).

Modbus TCP stack for host MCU of R-IN32M3 Module described in this document enables rapid and easy development of the following applications.

- Modbus TCP server
- Modbus Gateway

Modbus TCP stack for host MCU of R-IN32M3 Module supports the following nine function codes:

- 1(0x01) – Read coils
- 2(0x02) – Read discrete input
- 3(0x03) – Read holding registers
- 4(0x04) – Read input registers
- 5(0x05) – Write single coil
- 6(0x06) – Write single register
- 15(0x0F) – Write multiple coils
- 16(0x10) – Write multiple registers
- 23(0x17) – Read/Write multiple registers

For more information about Modbus, please refer to the following site.

http://www.modbus.org

「Modicon Modbus Protocol Reference Guide Rev.J」 (PI_MBUS_300.pdf)

「Modbus Application Protocol Specification V1.1b3」(Modbus_Application_Protocol_V1_1b3.pdf)

* The version number may vary depending on the update. Please refer to the latest manual.

1.2 Operating Environment

This chapter explains the operating environment.

1.2.1 Hardware Environment

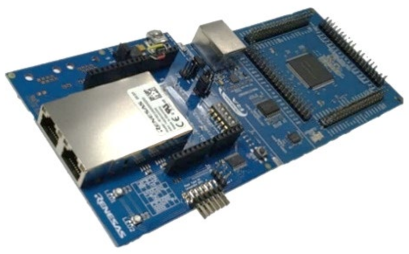
Modbus TCP communication described in this document convers one of the following configurations. This document is described based on RX66T environment (SEMB1320 evaluation board).

(1) R-IN32M3 Module RX66T CPU Card [SEMB1320]

(2) Adapter Board with R-IN32M3 Module combined with EK-RA6M3 or EK-RA6M4

(3) Adapter Board with R-IN32M3 Module combined with RL78/G14 Fast Prototyping Board

(1) R-IN32M3 Module RX66T CPU Card [SEMB1320]


(2) Adapter Board + EK-RA6M4

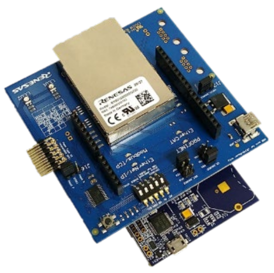

(3) Adapter Board + RL78/G14 Fast Prototyping Board

Figure 1-1 Development Environment

Table 1-1 Hardware environment

Name	Type Name	Maker	Link
RX66T CPU Card with R-IN32M3 Module	SEMB1320	SHIMAFUJI Electric Incorporated	SEMB1320
Adapter Board with R-IN32M3 Module	YCONNECT-IT-I-RJ4501	Renesas Electronics Corporation	R-IN32M3-Module-Solution-Kit
EK-RA6M3	RTK7EKA6M3S00001BU	Renesas Electronics Corporation	RA6M3 MCU Group Evaluation Board
EK-RA6M4	RTK7EKA6M4S00001BE	Renesas Electronics Corporation	Evaluation Kit for RA6M4 MCU Group
RL78/G14 Fast Prototyping Board	RTK5RLG140C00000BJ	Renesas Electronics Corporation	RL78/G14 fast Prototyping Boad

1.2.2 Software Environment

This document is described based on RX66T environment (SEMB1320 evaluation board).

Table 1-2 Software Environment

Category	Name	Version	Link	Note
R-IN32M3 Module sample package	Package of sample software	Rev.1.**	https://www.renesas.com/	
RA MCU				
IDE	e2studio	2022-04	e² studio 2022-04 Windows| Renesas	
Flexible Software Package	FSP	V3.6.0	-	Included in installer of e2studio FSPv3.6.0 | GitHub
GNU Arm Embedded Toolchain	GCC Toolchain	V10.3.1.20210824	—	Included in installer of e2studio
RX MCU				
IDE	e2studio	2022-04	e² studio 2022-04 Windows| Renesas	
GNU Toolchain for RZ Family	GCC for Renesas RX	V8.3.0.202102	—	Included in installer of e2studio
RL MCU				
IDE	e2studio	2022-04	e² studio 2022-04 Windows| Renesas	
GNU Toolchain for RL Family	GCC for Renesas RL78	V4.9.2.202103	—	Included in installer of e2studio

1.3 Reference Document

Technical information about Modbus is available on the Modbus Organization website, and information about R-IN32M3 Module is available on the Renesas Electronics website.

- Modbus Organization website : http://www.modbus.org
- Renesas Electronics website : http://www.renesas.com

Table1-3 Modbus Documentation

#	Document
1	Modbus_Application_Protocol_V1_1b3.pdf
2	PI_MBUS_300.pdf
3	Modbus_over_serial_line_V1_02.pdf
4	Modbus_Messaging_Implementation_Guide_V1_0b.pdf

Table1-4 R-IN32M3 Module Documentation

Document title	Document No.
R-IN32M3 Module (RY9012A0) Datasheet	R19DS0109ED****
R-IN32M3 Module (RY9012A0) User's Manual: Hardware	R19UH0122ED****
R-IN32M3 Module (RY9012A0) User's Manual: Software	R17US0002ED****
User's Implementation Guide (uGOAL Edition)	R30AN0402EJ****
Adaptor Board with R-IN32M3 module YCONNECT-IT-I-RJ4501 User's Manual	R12UZ0094EJ****
Management Tool Instruction Guide	R30AN0390EJ****
RA Sample Application (uGOAL Edition)	R30AN0398EJ****
RX66T Sample Application (uGOAL Edition)	R30AN0399EJ****
RL78/G14 Sample Application (uGOAL Edition)	R30AN0400EJ****

2. Sample Application

2.1 System Structure

The sample application is roughly divided into three function blocks.

1. uGOAL Abstraction layer stack
2. Modbus protocol stack program that uses the TCP/IP protocol stack
3. Application sample program that uses the uGOAL and Modbus protocol stacks

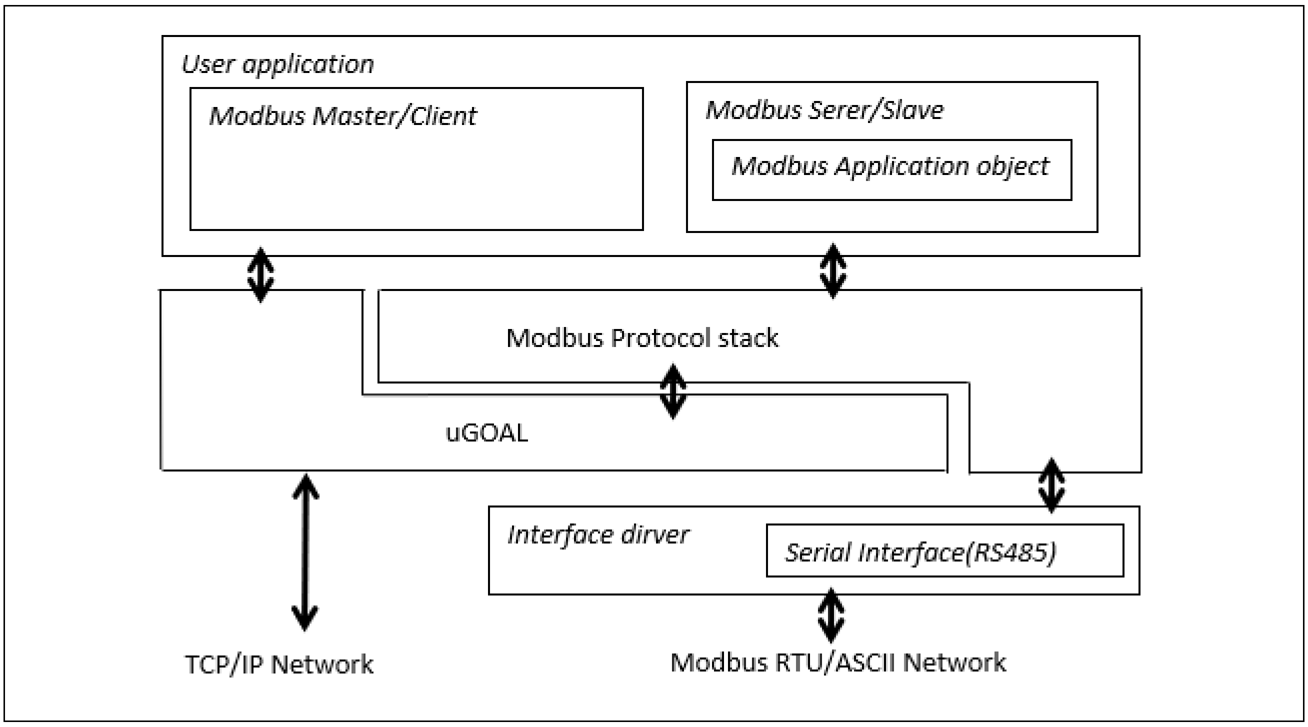


Figure 1-1 System Structure

2.1.1 uGOAL

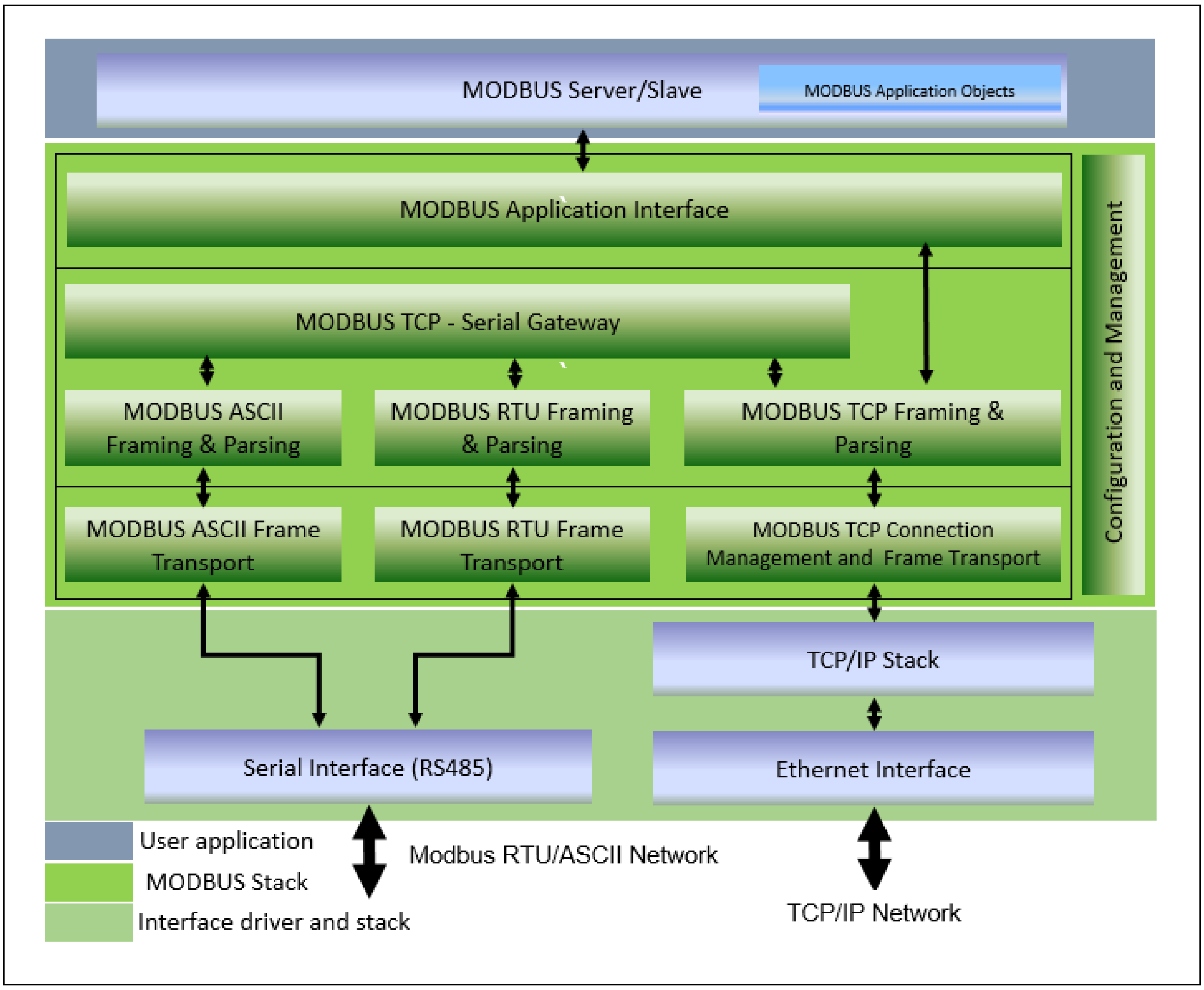
The R-IN32M3 Module software includes uGOAL (domain-specific middleware), which consists of a software stack that can be used to build applications in the industrial communications domain. In this sample program, it is used as TCP/IP stack control.

For details of uGOAL, refer to "R-IN32M3 Module (RY9012A0) User's Manual : Software (R17US0002ED****)".

2.1.2 Modbus Stack Program

The sample application includes a sample program in the protocol stack that provides communication functionality based on the Modbus protocol.

In Modbus TCP serial gateway mode, it behaves as a Modbus RTU / ASCII master stack as a gateway to the serial network. The Modbus RTU / ASCII master stack initialization is done within the gateway stack initialization. The user can choose either the Modbus RTU or the Modbus ASCII gateway stack (serial mode selection: MBAPP_INIT_GW_SERIAL_MODE).


MODBUS Server/Slave
MODBUS Application Objects
MODBUS Application Interface
MODBUS TCP - Serial Gateway
Configuration and Management
MODBUS ASCII Framing & Parsing
MODBUS RTU Framing & Parsing
MODBUS TCP Framing & Parsing
MODBUS ASCII Frame Transport
MODBUS RTU Frame Transport
MODBUS TCP Connection Management and Frame Transport
TCP/IP Stack
Serial Interface (RS485)
Ethernet Interface
User application
MODBUS Stack
Interface driver and stack
Modbus RTU/ASCII Network
TCP/IP Network

2.1.3 Modbus Application

The sample application realizes Modbus protocol communication by using the sample program of uGOAL and Modbus protocol stack.

For more information, see Chapter 2.5, "Application Implementation Guide".

2.2 Block Diagram

The Modbus sample application block diagram for RX66T

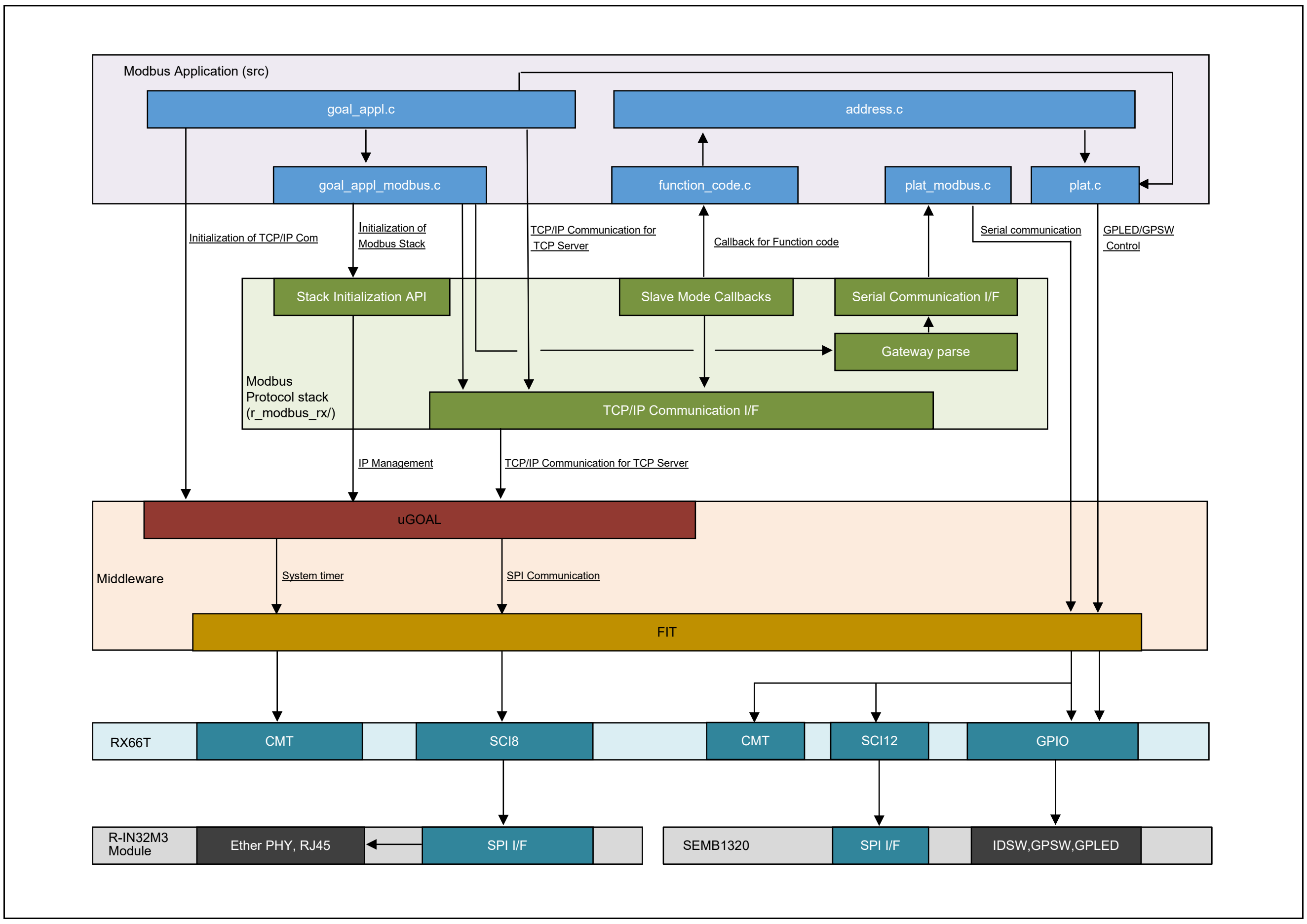


Figure 2-2 Block Diagram of RX66T sample application

2.3 File Structure

File structure (partially abbreviated) of the sample program

Table2-1 File Structure

Folder / File			Description
`07_mbus_tcp_server/`			Modbus TCP sample program
	`goal_appl.c`		uGOAL application implementation part
	`goal_config.h`		uGOAL configuration header file
	`goal_appl_modbus.c (.h)`		Modbus protocol stack initialization and packet parsing program
	`mbapp_slave/`		Modbus slave mode program
		`function_code.c (.h)`	Function code and Call-back program of Modbus slave mode
		`address.c (.h)`	Data addressing program of Modbus slave mode
	`r_modbus_rx/`		Modbus protocol stack Sample program
		`src/ (inc/)`	Modbus protocol stack sample program & header file

2.4 Build Structure

The sample application has a build configuration that corresponds to each mode of operation of the Modbus protocol.

Table2-2 Build Structure

Stack mode	Build Name
Modbus TCP Server Stack	TCP_SERVER_UGOAL
Modbus TCP Gateway Stack	TCP_GATEWAY_UGOAL

2.5 Resource Structure

Hardware resources used by the sample program.

1) Module

Hardware resources for the SEMB1320 evaluation board with RX66T to use as a Modbus stack. When adding functions to this sample program, please take care about resource contention.

Table2-3 Hardware Module

SW block	HW module	IO	Remark
Modbus Protocol stack	SCI12	P21（TXD12） P22（RXD12）	Modbus RTU/ASCII serial (RS485) Connection
	GP I/O	P20	Modbus RTU/ASCII serial Connection RS485 トランシーバ方向制御用端子
	CMT	-	RTU/ASCII mode RS485 timing Control
Modbus Application	GP I/O	PE3 PE4 PE1 PE0	LED: LED5, LED6, LED8, LED9 on SEMB1320
		PB4 PB2 PB1 PB0	Switch: SW2, SW4, SW5, SW6 on SEMB1320

* CMT: Compare match timer, SCI: Serial communication interface

2) Heap size

Uses the heap area under uGOAL management. If change the macro value of "**CONFIG_UGOAL_HEAP_BUFFER_SIZE**" included in *\plat \plat.h*.

2.6 Modbus Stack

The sample software is equipped with uGOAL middleware and is structured based on its design concept. In addition, the Modbus TCP sample has a structure that passes TCP information generated by uGOAL to the Modbus stack.

2.6.1 Modbus TCP Server Stack

Modbus TCP server program flow. [Build configuration: **TCP_SERVER_UGOAL**]

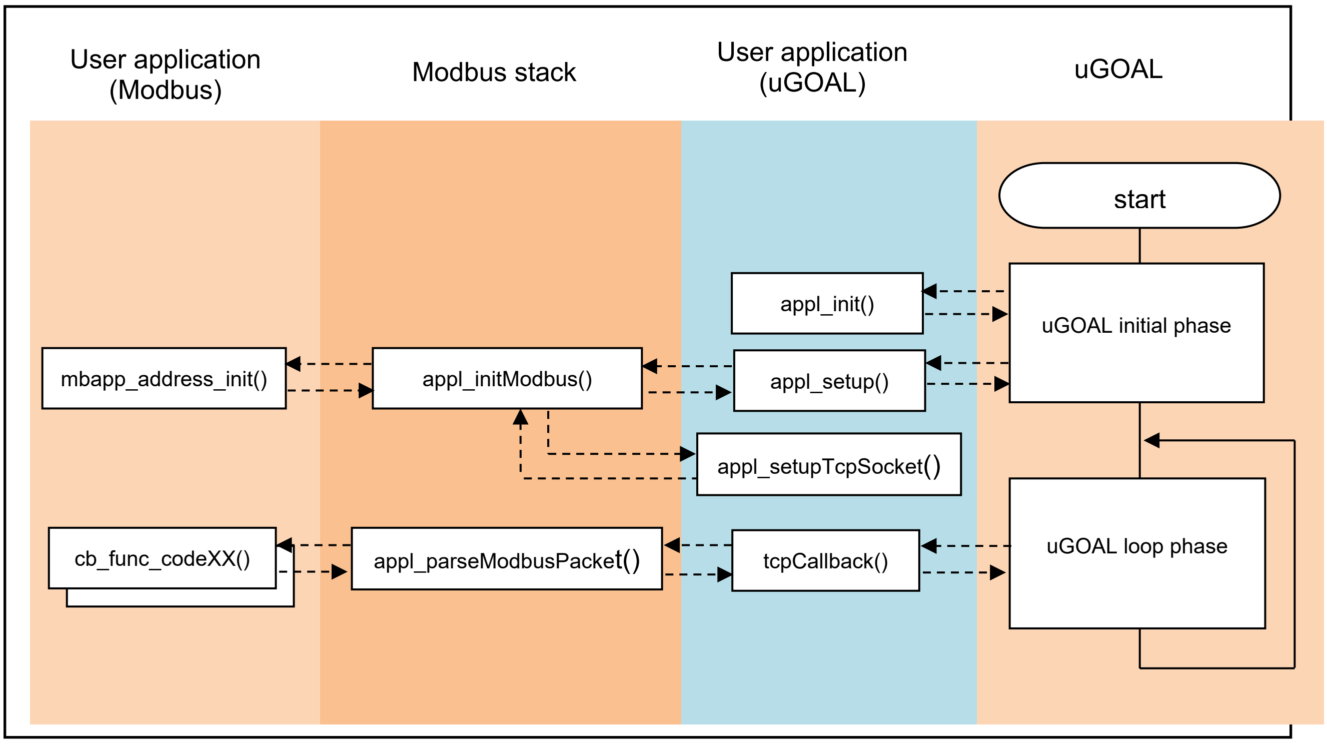


Figure 2-3 Modbus TCP server program flow

uGOAL provides functions of *appl_init()* and *appl_setup()* for specific processing of user application. In addition, tcpCallback() is registered as a callback function for event processing of tcp protocol in the sample software. In the initial phase of uGOAL, *appl_init ()* and *appl_setup ()* are executed, and *tcpCallback()* is executed every time a TCP communication event occurs in the subsequent loop phase.
For more information, see Chapter **2.6.3.**

Taking the opportunity of tcp events generated by uGOAL, data is passed to the Modbus stack and processed as Modbus TCP. The Modbus stack is roughly divided into two parts, *appl_modbusInit ()* for initializing the stack and *appl_parseModbusPacket ()* for packet parsing. In this sample software, some functions for user's own coding are prepared, such as *mbapp_address_init ()*, *cb_func_code01 ()* and else. User implements each function, depending on the device and the features they want to make.
For more information about the process, see Chapter **3.1.2.**

2.6.2 Modbus TCP Gateway Stack

Modbus TCP Gateway program flow. [Build configuration: **TCP_GATEWAY_UGOAL**]

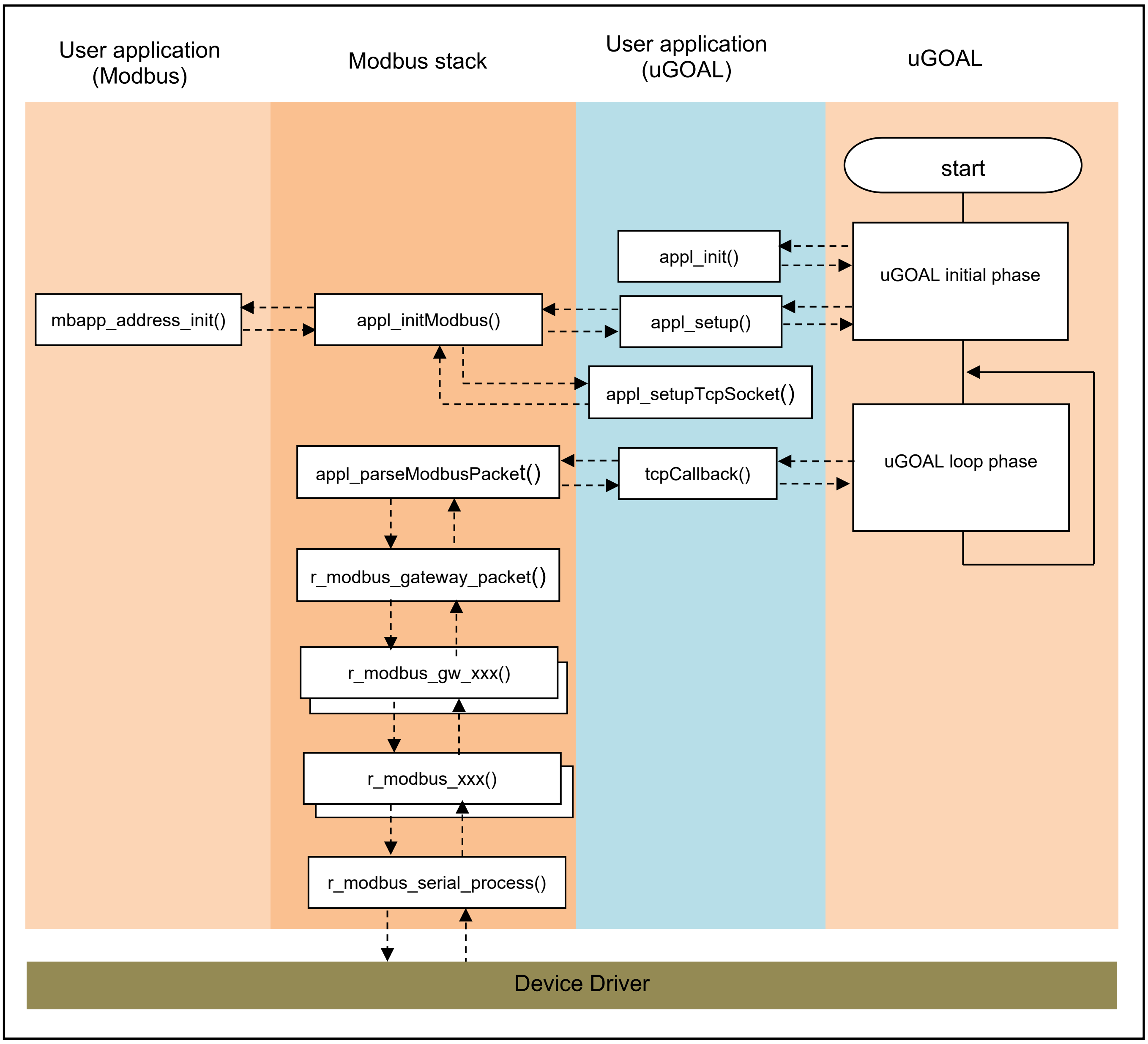


Figure 2-4 Modbus TCP Gateway program flow

The Modbus gateway flow is the same as Modbus TCP, but the behavior after Modbus packet analysis is different.
When ***appl_parseModbusPacket()*** recognizes a gateway packet, it shifts to gateway packet processing.

Gateway packet processing converts a request packet received from a client into an RTU or ASCII mode request packet and sends it to the connected serial device. After sending, it waits for a response to the request, converts the received response packet to a Modbus TCP response packet, and sends it to the client.

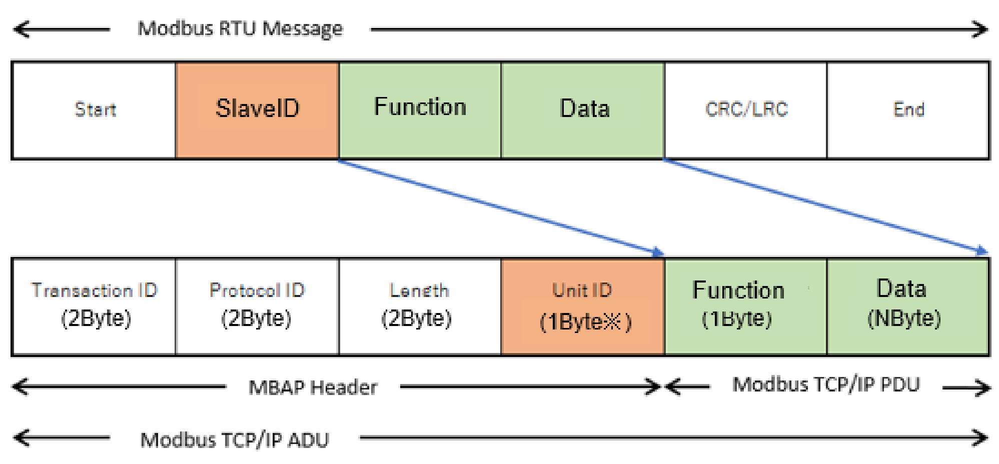
2.6.3 Packet Parsing

uGOAL parses received TCP packets and executes functions implemented by each function code. The functions described in this section are implemented in the function ***appl_parseModbusPacket*** of the sample software.

The Modbus TCP communication format includes the Modbus RTU excluding the CRC. Modbus RTU needs to add a CRC check code at the end for error check, but Modbus TCP is not necessary because it uses the check mechanism of the TCP / IP protocol.

In Modbus TCP communication format, it is necessary to add transaction identifier, protocol identifier, message length and unit identifier, which did not exist in Modbus RTU.

The difference between Modbus RTU format and Modbus TCP / IP format is shown in Figure 2-5



ADU：Application Data Unit **MBAP**：Modbus Application **PDU**：Protocol Data Unit

Figure 2-5 Modbus RTU format and Modbus TCP / IP format

In accordance with the above communication format, the received TCP packet is parsed. Here are the key points for packet parsing:

- If received packet validation experiences a mismatch of length checks, packet integrity, slave ID, and so on, discard the inbound packets.
- If received packet is correct, user-registered callback function is called to process the request.
- In case unicast request is received, if there is an abnormality in process of the function code, an exception response message is created and sent to the master device.
- In case broadcast request is received, if the received packet is a write-related function code, it will be accepted as a normal packet, but the response message will not be sent to the master device. Also, if the received packet is read-related function code, the request packet is dropped as a slave ID abnormality.

2.6.4 TCP Connection and disconnection

Modbus stack manages the number of TCP clients that can be connected simultaneously.

1) IP address function

Supports a function to allow (whitelist) and deny (blacklist) clients connecting to the relevant TCP server by IP address.
When the list function [**MBAPP_INIT_IP_TABLE_FLAG**] is enabled, it checks the IP address of the client that requested the connection according to the operation mode, and decides whether to accept or reject the connection request. In the sample software, it is implemented with the function *appl_tcpCallback*.

The target IP address must be registered in advance by the user. The number of IP addresses that can be registered depends on **MAXIMUM_NUMBER_OF_CLIENTS**.

2) Multiclient function

When the multi-client function is enabled [**ENABLE_MULTIPLE_CLIENT_CONNECTION**], connections are accepted up to the number of clients specified by **MAXIMUM_NUMBER_OF_CLIENTS** in the FIFO method. If the maximum number of connection requests is exceeded, the oldest connection will be closed before accepting a new client connection. If the multi-client function is disabled, all connection requests from the second and subsequent clients will be rejected.

The Modbus stack manages the number of TCP clients that can be connected simultaneously.
When TCP connection/disconnection is detected, it is also reported to the Modbus stack side. On the Modbus stack side, if the maximum number of connections has been reached, the oldest connection will be closed and the new connection will be registered as a valid connection.
The content explained in this section is implemented by the function *tcpCallback* on the sample software.

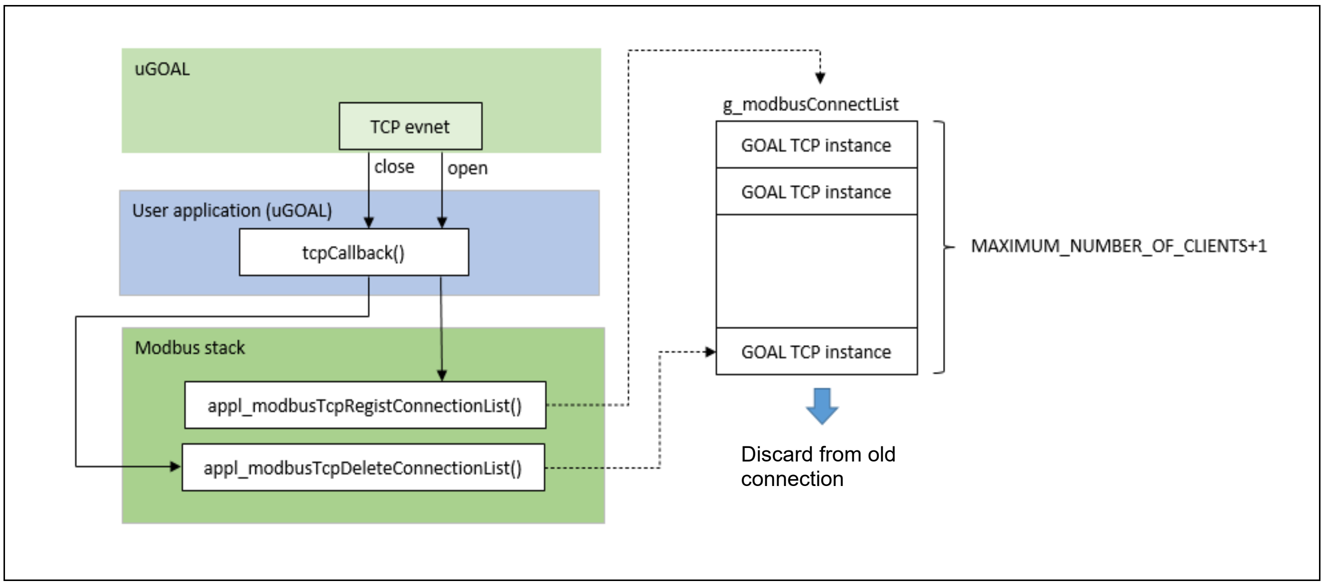


Figure 2-6 TCP Connection Management

3. Application Implementation guide

3.1 Mapping Data

The Modbus data model implemented in the sample software is described. The contents described in this chapter are defined in address .c (.h) of each sample application.

- Each Modbus data type has 2,048 pieces of data and uses reference addresses 0x0001 to 0x0800 (*1).
- Reference addresses are allocated to RAM memory as an array buffer of static global variables.
- Implemented functions to accesses (Reads/Writes) each array buffer of each Modbus data type which is used by the callback function of Modbus function code (*2).
    - ➢ Coil address 0x0001 to 0x0004 : GPIO ports for LEDs
    - ➢ Discrete Input address 0x0001 to 0x0004: GPIO port for SWs
- Returns the exception code 0x04: SLAVE DEVICE FAILURE when accessing 0x07D1 of each Modbus data address(* 3).
- Implemented a function that checks whether the address range specified by each Modbus data type contains an incorrect address.
- Callback functions corresponding to each Modbus function code use these check functions to detect incorrect addresses and return illegal DATA ADDRESS before accessing buffer arrays 0x02 exception code (*4)

Figure 2-6 shows the reference address design and physical memory mapping of each Modbus data type.

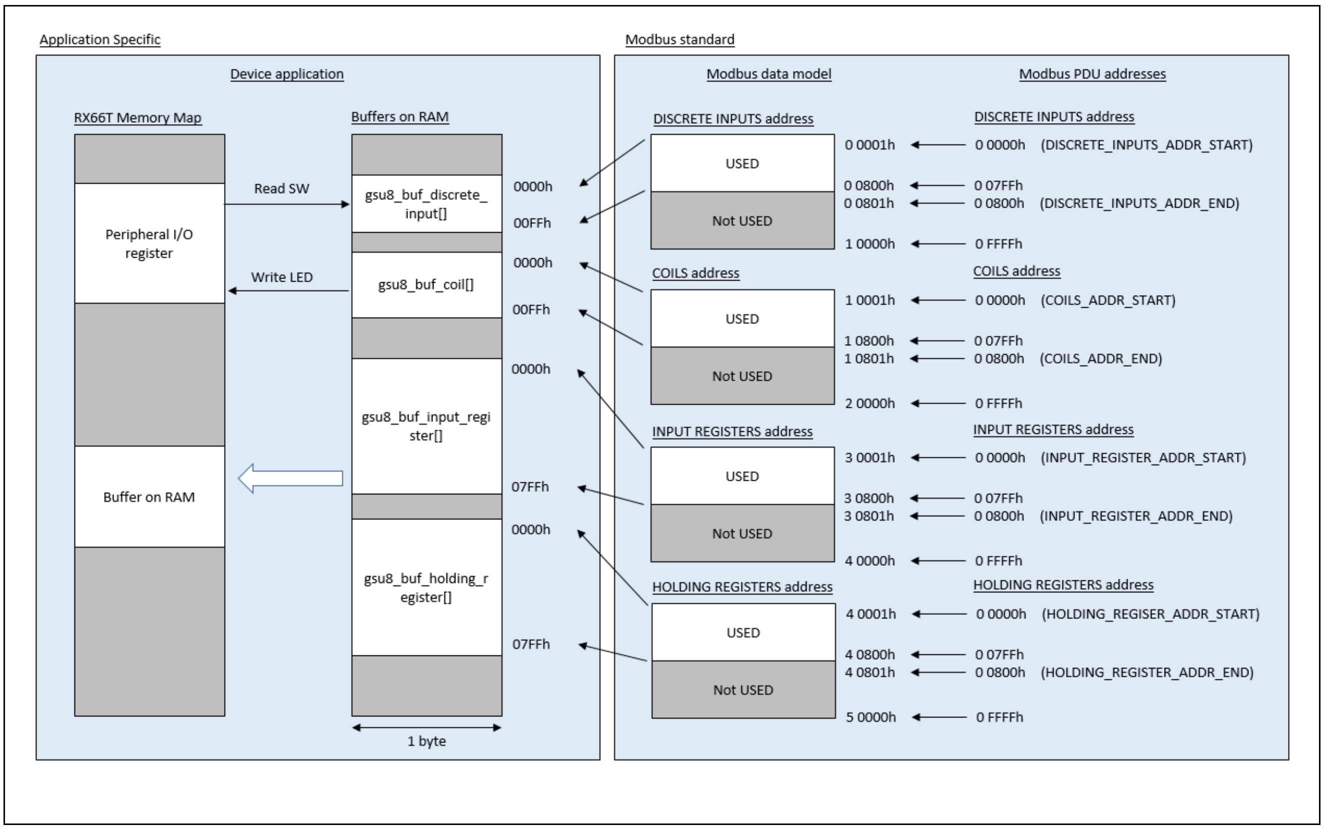


Figure 2-1 Modbus data model and memory mapping

*1: The range of reference addresses are specified in the constant macro corresponding to each Modbus data type.

- For example, COILS_ADDR_START, COILS_ADDR_END constant macros specify the start and end addresses of reference addresses of data type Coils.
- By default, each Modbus data address 0x0000 as the start address and 0x0800 as the end address.
- Note that the address value specified here is the start of 0x0000, not 0x0001.
- This is to comply with the address value representation described in the Protocol Data Unit (PDU) that makes up the Modbus request. Hereafter the address representation is called PDU address.
- In this sample software, all reference addresses are processed with PDU addresses.
- Note that valid address termination values, such as COILS_ADDR_END, are exclusive, and addresses with the specified values are invalid addresses.

*2: In the sample program, the reference address assigned to the Device peripheral function register is determined relative to the COILS_ADDR_START and the DISCRETE_INPUT_ADDR_START, respectively.

- For example, if the COILS_ADDR_START is 0x1000, the Coil address 0x1001 to 0x1004 will be linked to GPIO port corresponding to LED 5, 6, 8, and 9 on the SEMB1320 board.

*3: Address values that return 0x04 exception codes can be specified in the constant macro PUSEDO_SLAVE_FAILURE_ADDR. Note that the specified value is a PDU address.

*4: By changing the check function of incorrect addresses, any address can be designed as an incorrect address.

Sample code example of a data access function to Coils region called by a callback function when Read Coil (function code 01h) is executed.

Example) File : *07_mbus_tcp_server\mbapp_slave\address.c*

```
uint8_t read_coil_address(uint16_t u16_address, uint8_t *u8_bit)
{
        if( u16_address == PUSEDO_SLAVE_FAILURE_ADDR )
        {                                                                                        ①
                return ERR_SLAVE_DEVICE_FAILURE;
        }

        /* calculate buffer index from address structure */
        uint16_t buf_addr = u16_address - COILS_ADDR_START;

        /* assign LEDs to coil address ADDR_START ~ ADDR_START+3  */
        if( buf_addr < GPLED_NUM )
        {
                /* write LEDs data in buffer */                                                  ②
                _write_bit_buf(gsu8_buf_coil, buf_addr, plat_remoteIoLedGet(buf_addr));
        }

        /* read buffer */
        *u8_bit = _read_bit_buf(gsu8_buf_coil, buf_addr);                                        ③

        return ERR_OK;
}
```

① Test code that returns a device error in the case of a specific address.

② Read status of the device corresponding to the Coils area (in this case LEDs for remote I/O).

③ Stores data in the Coils area into a buffer.

3.1.1 uGOAL Part

This chapter describes the implementation related to uGOAL middleware in Modbus TCP sample software. The contents described in this section are defined in goal_appl.c of each sample application. For details of each API, refer to "R-IN32M3 Module (RY9012A0) User's Manual : Software (R17US0002ED****)."

1) appl_init

This function includes application-specific initialization steps before initializing uGOAL core modules, etc. In the Modbus TCP sample software, network functions are initialized for TCP/IP communication.

```
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res;                          /* result */

    /* initialize goal net */
    res = goal_netRpcInit(GOAL_NET_ID_DEFAULT);
    if (GOAL_RES_ERR(res)) {                                                              ①
        goal_logErr("Initialization of goal net RPC failed");
    }

    return res;
}
```

① Initialize RPC function of the network.

2) appl_setup

This function sets network configuration and initializes Modbus stack.

```
GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;                          /* result */
    uint32_t ip;                                /* IP address */
    uint32_t nm;                                /* netmask */
    uint32_t gw;                                /* gateway */
    uint32_t cnt;                               /* counter */

    goal_maLedSet(pMaLed, GOAL_MA_LED_LED1_RED, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_LED1_GREEN, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_LED2_RED, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_LED2_GREEN, GOAL_MA_LED_STATE_OFF);      ①
    goal_maLedSet(pMaLed, GOAL_MA_LED_ETHERCAT, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_PROFINET, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_MODBUS, GOAL_MA_LED_STATE_ON);
    goal_maLedSet(pMaLed, GOAL_MA_LED_ETHERNETIP, GOAL_MA_LED_STATE_OFF);

    res = goal_maNetOpen(GOAL_NET_ID_DEFAULT, &pMaNet);                      ②
    if (GOAL_RES_ERR(res)) {
        goal_logErr("error opening network MA");
    }

    /* set IP address */
    ip = MAIN_APPL_IP;
    nm = MAIN_APPL_NM;
    gw = MAIN_APPL_GW;
    res = goal_maNetIpSet(pMaNet, ip, nm, gw, GOAL_FALSE);                   ③
    if (GOAL_RES_ERR(res)) {
        goal_logErr("Set IP failed");
        return res;
    }

    appl_modbusInit();                                                        ④

    return GOAL_OK;
}
```

① Set the initial state of LED.

② Open the network media adapter (MA) to use.

③ Set IP address for TCP/IP protocol.

④ Initialize Modbus stack

3) tcpCallback

In this function, TCP communication events occurred are processed.

```
static void tcpCallback(
・・・
)
{
    GOAL_NET_ADDR_T remote;                 /* remote address */
    GOAL_STATUS_T res;                      /* result */

    if (cbType == GOAL_NET_CB_NEW_SOCKET) {
・・・
        /* check connectable IP address */
        res = r_modbus_chk_connectable_ip(remote.remoteIp);
        if (GOAL_RES_ERR(res)) {                                              ①
・・・
            return;
        }

        /* check multiple connection */
        res = r_modbus_tcp_multi_connection();
        if (GOAL_RES_ERR(res)) {                                              ②
・・・
            return;
        }

        res = r_modbus_tcp_multi_connection(pChan);
        if (GOAL_RES_ERR(res)) {
            /* Close oldest connection if new connection cannot be registered */
            res = goal_maChanTcpClose(pMaTcpHdl, appl_modbusTcpGetOldestConnection());
            if (GOAL_RES_ERR(res)) {
                goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
            }                                                                 ③
            r_modbus_tcp_del_connection_list(r_modbus_tcp_get_oldest_connection());
            /* regist new connection */
            res = r_modbus_tcp_reg_connection_list(pChan);
            if (GOAL_RES_ERR(res)) {
                goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
            }
        }
    }
    else if (cbType == GOAL_NET_CB_NEW_DATA) {
        goal_logInfo("Data received on tcp socket %p", (void *) pChan);

        res = appl_parseModbusPacket(pMaTcpHdl, pChan, pBuf);
        if (GOAL_RES_ERR(res)) {
            /* Close oldest connection if new connection cannot be registered */
            res = goal_maChanTcpClose(pMaTcpHdl, pChan);
            if (GOAL_RES_ERR(res)) {                                          ④
                goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
            }
            r_modbus_tcp_del_connection_list(r_modbus_tcp_get_oldest_connection());
        }
    }
    else if (cbType == GOAL_NET_CB_CLOSING) {
        goal_logInfo("Closing TCP socket %p", (void *) pChan);

        r_modbus_tcp_del_connection_list(pChan);

        res = goal_maChanTcpGetRemoteAddr(pMaTcpHdl, pChan, &remote);
        if (GOAL_RES_ERR(res)) {                                              ⑤
            goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
        }
    }
}
```

① Checks whether to accept connections from the IP address that requested the connection. If the target IP address is not reachable, the requested instance will be closed.

② Checks whether to accept multiple client connections. If the multi-client function is disabled and it is the second or subsequent connection request, the TCP instance for the new connection request is closed.

③ Registers a net channel for a newly opened TCP/IP connection with the Modbus stack. The maximum number of simultaneous connections is managed by the Modbus stack, so net channel disconnection processing is performed as necessary.

④ Pass the received TCP packet to Modbus packet analysis processing.

⑤ Get notified of net-channels whose connections have been closed and exclude them from Modbus stack management.

4) appl_setupTcpSocket

This function is called from Modbus stack when TCP socket is generated.

```
GOAL_STATUS_T _appl_setupTcpSocket(uint32_t additionalPort)
{
    GOAL_STATUS_T res;                          /* result */
    GOAL_NET_ADDR_T addr;                       /* net address */
    GOAL_NET_CHAN_T *pChan;                     /* channel */
    uint32_t clients;

    res = goal_maChanTcpOpen(GOAL_NET_ID_DEFAULT, &pMaTcp);                                    ①
    if (GOAL_RES_ERR(res)) {
        goal_logErr("error getting tcp MA");
        return res;
    }

    for (clients = 0; clients < MAXIMUM_NUMBER_OF_CLIENTS + 1; clients++) {
        /* register TCP server */                                                              ②
        GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
        addr.localPort = (uint16_t) (TCP_PORT_NUMBER);
        res = goal_maChanTcpNew(pMaTcp, &pChan, NULL, &addr, GOAL_NET_TCP_LISTENER, tcpCallback);
        if (GOAL_OK != res) {
            goal_logErr("error while opening TCP server channel on port %"FMT_u32, (uint32_t) TCP_PORT_NUMBER);
            return res;
        }

        /* set TCP channel to non-blocking */
        res = goal_maChanTcpSetNonBlocking(pMaTcp, pChan, GOAL_TRUE);
        if (GOAL_OK != res) {
            goal_logErr("error while setting TCP channel to non-blocking");
            return res;
        }
    }

    /* greet */
    goal_logInfo("waiting for TCP connections on port %"FMT_u32"(n=%"FMT_u32")", TCP_PORT_NUMBER, clients);

    if (0 != additionalPort)
    {
        res = goal_maChanTcpOpen(GOAL_NET_ID_DEFAULT, &pMaTcpAdditional);
        if (GOAL_RES_ERR(res)) {                                                               ③
            goal_logErr("error getting tcp MA");
            return res;
        }

        for (clients = 0; clients < MAXIMUM_NUMBER_OF_CLIENTS + 1; clients++)
        {                                                                                      ④
            GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
            addr.localPort = (uint16_t) (additionalPort);
            res = goal_maChanTcpNew(pMaTcpAdditional, &pChan, NULL, &addr, GOAL_NET_TCP_LISTENER, tcpCallback);
・・・
        }

        /* greet */
        goal_logInfo("waiting for TCP connections on port %"FMT_u32"(n=%"FMT_u32")", additionalPort, clients);
    }

    return res;
}
```

① Open the TCP Channel Media Adapter (MA).

② Generate (**MAXIMUM_NUMBER_OF_CLIENTS** + 1) TCP channels for normal connection ports (**TCP_PORT_NUMBER**). TCP channels are prepared in non-blocking mode.

③ Open the TCP Channel Media Adapter (MA) for additional connection ports.

④ Generate (**MAXIMUM_NUMBER_ OF_CLIENTS**+1) TCP channels for additional connection ports (**TCP_ADDITIONAL_PORT_NUMBER**).

In this function, generate uGOAL instances by the maximum number +1 which is specified in the **MAXIMUM_NUMBER_OF_CLIENTS**.

Note that if increase the maximum number of MAXIMUM_NUMBER_OF_CLIENTS, also need to increase uGOAL resources (e.g. **CONFIG_UGOAL_HEAP_BUFFER_SIZE**).

3.1.2 Modbus Stack Part

This chapter describes the operation of Modbus stack part. Modbus stack in the sample software realizes the function in conjunction with uGOAL's TCP/IP function. The processing details are explained below.

3.1.2.1 Configuration Settings

The following table shows the configuration settings of Modbus stack which users can change. Each setting is defined in the macro in *modbusTcpConfig.h*.

Table 2-1 Modbus stack configuration settings

#	Macro Name	Default	Description
1	**TCP_PORT_NUMBER**	502	Specifies TCP port number to accept as a Modbus connection.
2	**TCP_ADDITIONAL_PORT_NUMBER**	1024	Specifies additional ports to accept for ports configured in the TCP_PORT_NUMBER. If set to 0, additional ports are disabled.
3	**MAXIMUM_NUMBER_OF_CLIENTS**	7	Specifies the upper limit number of TCP clients that can be connected in the Modbus stack.
4	**MBAPP_INIT_MULTIPLE_CLIENT_FLAG**	ENABLE	Multiclient setting **ENABLE**: **ENABLE_MULTIPLE_CLIENT_CONNECTION** **DISABLE**: **DISABLE_MULTIPLE_CLIENT_CONNECTION**
5	**MBAPP_INIT_IP_TABLE_FLAG**	DISABLE	IP address linting
6	**MBAPP_INIT_IP_TABLE_MODE**	ACCEPT	IP address Whitelist and Blacklist

Table 2-2 Modbus Serial stack configuration settings

#	Macro Name	Default	Description
1	**MBAPP_INIT_GW_SERIAL_MODE**	RTU mode	Connection Type RTU mode: **MODBUS_RTU_MASTER_MODE** ASCII mode:**MODBUS_ASCII_MASTER_MODE**
2	**MBAPP_INIT_SLAVE_ID**	1	Modbus Serial device ID 1 – 247
3	**MBAPP_SERIAL_BAUDRATE**	115200	Connection Speed bps
4	**MBAPP_SERIAL_PARITY**	Notting Parity	Parity setting. Setting value depends on the serial driver I/F.
5	**MBAPP_SERIAL_STOPBITS**	1 bit	Stop bits setting Setting value depends on the serial driver I/F.
6	**MBAPP_INIT_RESPONSE_TIMEOUT_MS**	2000	Serial connection timeout msec
7	**MBAPP_INIT_TURNAROUND_DELAY_MS**	200	Broadcast delay time msec
8	**MBAPP_INIT_RETRY_COUNT**	3	Connection retry setting

3.1.2.2 Stack Initialization

Execute various initializations and start Modbus stack. These initializations must be done after uGOAL initialization. The following describes what to do with initialization. The functions described in this section are implemented in the *appl_modbusInit* function in the sample software.

1) Callback functions corresponding to each function code

Associates a processing request from the client for each function code with a user-defined function. When Modbus stack receives a request, it calls the registered function.

Registration settings are made by the Stack API (*Modbus_slave_map_init*). Here are the details of the arguments to set in the stack API. The corresponding callback function in the table below is set, and it is registered in the stack by executing the stack API.

Table 2-3 Callback function with Function Code

Code	Function Name	Default callback function
1	Read Coils	cb_func_code01
2	Read Discrete Inputs	cb_func_code02
3	Read Holding Registers	cb_func_code03
4	Read Input Registers	cb_func_code04
5	Write Single Coil	cb_func_code05
6	Write Single Register	cb_func_code06
15	Write Multiple Coils	cb_func_code15
16	Write Multiple Registers	cb_func_code16
23	Read/Write Multiple Registers	cb_func_code23

・Mapping Table of Function Code (*slave_map_init_t*)

```
typedef struct _slave_map_init{
        fp_function_code1_t     fp_function_code1;      /* function code-1 Read coils pointer */
        fp_function_code2_t     fp_function_code2;      /* function code-2 Read discrete inputs pointer */
        fp_function_code3_t     fp_function_code3;      /* function code-3 Read holding registers pointer */
        fp_function_code4_t     fp_function_code4;      /* function code-4 Read input registers pointer */
        fp_function_code5_t     fp_function_code5;      /* function code-5 Write single coil pointer */
        fp_function_code6_t     fp_function_code6;      /* function code-6 Write single register pointer */
        fp_function_code15_t    fp_function_code15;     /* function code-15 Write multiple coils pointer */
        fp_function_code16_t    fp_function_code16;     /* function code-16 Write multiple registers pointer */
        fp_function_code23_t    fp_function_code23;     /* function code-23 Read/Write multiple registers pointer*/
}slave_map_init_t, *p_slave_map_init_t;
```

2) Initialization Modbus Stack

Initialize the Modbus stack with the stack API (*r_modbus_tcp_init_stack* or *r_modbus_tcp_init_gateway_stack*). Via the API, the *_appl_setupTcpSocket* is called to create the necessary TCP sockets.

3.1.2.3 Function Code

In Modbus stack of the sample software, function code processing to access Modbus data model is implemented as a callback function, which allows users to design arbitrary Modbus data model. The function code is included in *function_code.c* of sample software.

The followings are the main points for implementing function code:

- Modbus protocol stack can register callback functions for Table 2-5 function code. The sample software shows an example implementation of all callback functions.
- Modbus protocol has a unique data model, and the data model consists of four data types and an address space corresponding to each data type. Table 2-6 shows the corresponding Modbus data type of this protocol stack.
- Each callback function that processes function code is required to access the corresponding data type.
- In Modbus protocol, slave can have up to 65536 (0x10000) of data for each data type, each assigned a reference address in the range of 1 to 65536 (0x00001 to 0x10000).
- The reference address can refer to arbitrary physical address.

Table 2-4 Modbus Function Code

Code	Code (Hex)	Function Name	Description
1	1h	Read Coils	Read multiple data of specified Coils addresses
2	2h	Read Discrete Inputs	Read multiple data of specified Discrete Inputs addresses.
3	3h	Read Holding Registers	Read multiple data of specified Holding Registers addresses.
4	4h	Read Input Registers	Read multiple data of specified Input Registers addresses.
5	5h	Write Single Coil	Write single data of specified Coils addresses.
6	6h	Write Single Register	Write single data of specified Holding Registers addresses.
15	Fh	Write Multiple Coils	Write multiple data of specified Coils addresses.
16	10h	Write Multiple Registers	Write multiple data of specified Holding Registers addresses.
23	17h	Read/Write Multiple Registers	First, write multiple data of specified Coils addresses. Second, write multiple data of specified Holding registers address.

Table 2-5 Modbus Data Type

Name	Bits	Type of access	Description
Discrete Inputs	1	Read	Data type provided from I/O system
Coils	1	Read/Write	Data type modified from application program.
Input Registers	16	Read	Data type provided from I/O system
Holding Registers	16	Read/Write	Data type modified from application program.

There are two things to do with callback function of Modbus function code:

1. Refer to the request structure of the pointer argument and set the processing result of the function code in the response structure of the pointer argument.
2. When a reference address that the designed Modbus data model does not have is referenced, the exception code 0x02 is set in the response structure.

Also, an exception code 0x04 can be set for unrecoverable errors that occur during function code processing.

Table 2-6 Exception code of function code

Code	Code (Hex)	Function Name	Description
2	2h	Illegal Data Address	Return this code when specified addresses in request includes incorrect address for user-designed Modbus data model.
4	4h	Slave Device Failure	Return this code when unrecoverable error occurs during function code process.

The callback function for each function code is defined in the following format. For more information about the structure used to argument each callback function, see **Appendix A**.

【Format】

```
uint32_t (*fp_function_code<function code>_t)(p_req_<function code>_t pt_request,
                                              p_resp_<function code>_t pt_response );
```

【Argument】

p_req_<function code>_t	pt_request	Pointer to structure containing function code request information
p_resp_<function code>_t	pt_response	Pointer to structure that stores function code response data

【Return Value】

uint32_t	0 : Success, 1 : Fail

The following is an example of the sample code of a callback function called when Read Coil (function code 01h) is executed.

ex.) File : *07_mbus_tcp_server\mbapp_slave\function_code.c*

```
uint32_t cb_func_code01(p_req_read_coils_t  pt_request,
                        p_resp_read_coils_t pt_response)
{
    uint8_t     u8_data;
    int         i;

    /* Copy invariant fields between response and request. */
    pt_response->u16_transaction_id = pt_request->u16_transaction_id;
    pt_response->u16_protocol_id    = pt_request->u16_protocol_id;                    ①
    pt_response->u8_slave_id        = pt_request->u8_slave_id;

    /* Check starting address and ending address.  ( START <= address < END ) */
    pt_response->u8_exception_code = check_illegal_coils_address( pt_request->u16_start_addr,
                                                                  pt_request->u16_num_of_coils );   ②
    if ( ERR_OK != pt_response->u8_exception_code ){ return ERR_OK; }

    /* Get response data by accessing address */
    for( i = 0; i < pt_request->u16_num_of_coils; i++ )                                ③
    {
        /* Get response data by accessing address. */
        pt_response->u8_exception_code = read_coil_address(pt_request->u16_start_addr+i, &u8_data);
        if ( ERR_OK != pt_response->u8_exception_code ){ return ERR_OK; }   /* Return value is ignored on stack. */

        /* Set a bit into a response byte which are cleared every byte. */
        if( i % 8 == 0 ) { pt_response->aru8_data[i/8] = 0; }
        pt_response->aru8_data[i/8] |= u8_data << ( i % 8 );
    }

    /* Calculate the number of bytes of response data. */
    pt_response->u8_num_of_bytes = _num_of_bytes(pt_request->u16_num_of_coils);

    return ERR_OK; // Return value is ignored on stack.
}
```

① Sets information to be stored in the header of response packet.

② Check the indicated address and data length for anomalies in the access range.

③ Get data with specified address range from Coils area and store it in the response packet.

3.1.2.4 TCP/IP connection Management

This chapter describes the TCP/IP communication implementation of the Modbus stack project.

Whitelist and Blacklist

To use the IP address whitelist and blacklist functions, you must register the target IP address. To register IP address, use *r_modbus_tcp_add_ip_addr*.

In the sample program, *r_modbus_enable_host_ip* is registered before stack initialization.

3.1.3 Serial Communication

This section describes the implementation of processing to communicate with Modbus serial devices in Modbus TCP/serial gateway mode. The contents explained in this section are defined in *plat_modbus.c* of each sample application.

1) **Serial communication**
The RX66T sample project uses a FIT (Firmware Integration Technology) serial communication interface (SCI) module to communicate with serial devices. Since serial communication is called from the Modbus stack component, the following APIs are set for the I/F with the Modbus stack.

Table 3-7 Modbus Serial stack configuration settings

#	API Name	Description
1	r_modbus_serial_open	Open the Serial communication
2	r_modbus_serial_read	Read the Serial data
3	r_modbus_serial_write	Write the Serial data
4	r_modbus_rs485_tx_enable	Enable Serial communication
5	r_modbus_rs485_tx_disable	Disable Serial communication

2) **RS485 communication by RTU/ASCII mode**
RTU mode begins with at least 3.5 characters of idle time and ends with 3.5 characters of idle time.
In this sample, FIT's Compare Match Timer (CMT) module is used to measure this idle time. Since the non-communication time measurement is performed in the Modbus stack component, the following API is set for the I/F with the Modbus stack.

Table 3-8 Modbus Serial stack configuration settings

#	API Name	Description
1	r_modbus_init_timer	Measurement timer initialization
2	r_modbus_timer_oneshot	Start Measurement
3	r_modbus_timer_stop	Stop Measurement

4. Communication Test with Evaluation Tool

Use the evaluation tool (ModbusDemoApplication.exe) to see how Modbus protocol stack sample program and Modbus application sample program works.

Evaluation Tool: ModbusDemoApplication

r18an0064**xxx \ tool \ **ModbusDemoApplication.zip**

4.1 Modbus TCP Server

1) Environment Structure

Use SEMB1320 evaluation board and evaluation tools to check the operation of Modbus TCP.

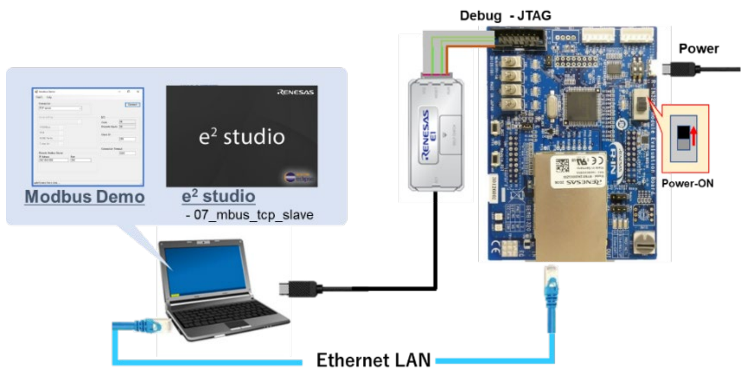


Figure2-1 Modbus TCP Server Environment

[Note] When Modbus TCP sample app is run, all protocol LEDs (LED1, LED2 and LED3) are off.

2) Modbus TCP application

The evaluation tool runs on Windows PC and acts as an oncoming software for Modbus application sample program running on SEMB1320 evaluation board.

SEMB1320 evaluation board transmits switch(SW) status or controls the LED lighting according to Modbus communication with the evaluation tool. The Coil address corresponding to the LED is shown in Table2-2, and the Dispatch Input address corresponding to SW is shown in Table2-3.

Table2-1 Coil address and LED

Coil address	Corresponding LED
0001h	LED5
0002h	LED6
0003h	LED8
0004h	LED9

Table2-2 Discrete Input address and SW

Discrete Input Address	Corresponding SW
0001h	SW2
0002h	SW4
0003h	SW5
0004h	SW6

3) Modbus Demo tool

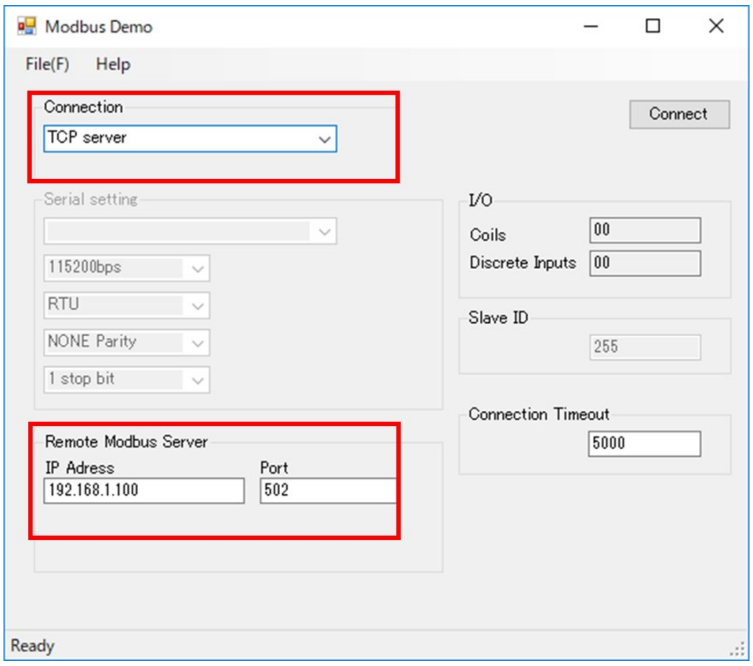
Select the operation mode in "Connection" and set various parameters.

- **Connection**
  - ➢ Select TCP server

    * Serial Master and Serial Slave are **not supported** by this sample software.

- **Serial setting**
  - ➢ Do not use.

- **Remote Modbus Server**
  - ➢ Set the IP address and Port number according to the device.



Connection: TCP server

Serial setting: 115200bps, RTU, NONE Parity, 1 stop bit

Remote Modbus Server: IP Adress 192.168.1.100, Port 502

I/O: Coils 00, Discrete Inputs 00

Slave ID: 255

Connection Timeout: 5000

Figure 4-2 Modbus Demo tool setting [TCP server]

（Continue to next page）

When the sample application is Modbus slave, the evaluation tool acts as a Modbus master and operate as follows:

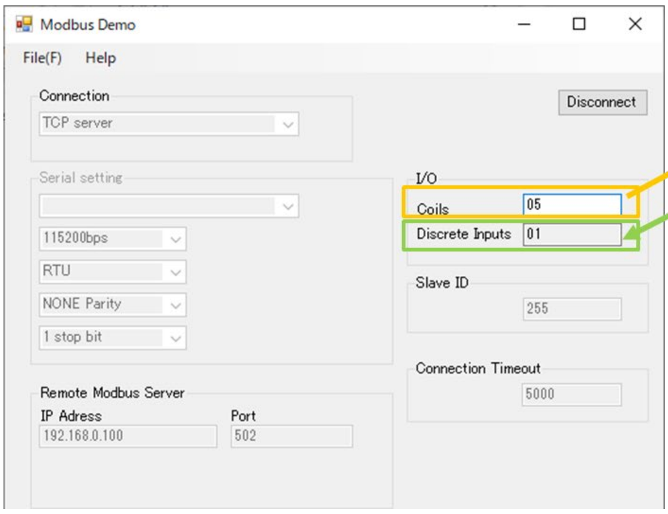
- Run "Connect" and the evaluation tool sends "Read Discrete Inputs" request.
- Sample application receives "Read Discrete Inputs" request and reflects the status of SW (SW2, SW4, SW5, SW6) on SEMB1320 to "Discrete Inputs".
- Modbus Demo tool determines how to update the LEDs on the SEMB1320 evaluation board according to the SW status received as a response to “Read Discrete Inputs” request, and sends a “Write Multiple Coils” request.

  - **SW ＝ 0** : The value in Coils text box changes 01→02→04→08→01...

    The Coils value is reflected in the LED (LED5, LED6, LED8, LED9).

  - **SW≠ 0** : Enter any value in the Coils text box.

    The Coils value is reflected in the LED (LED5, LED6, LED8, LED9).



Connection: TCP server

Serial setting: 115200bps, RTU, NONE Parity, 1 stop bit

Remote Modbus Server
IP Adress: 192.168.0.100
Port: 502

I/O
Coils: 05
Discrete Inputs: 01

Slave ID: 255

Connection Timeout: 5000

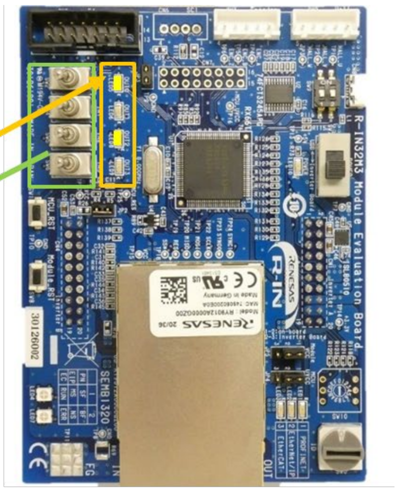


Figure 4-3 Modbus Client demo tool

4.2 Modbus TCP Gateway

1) Environment Structure

Use SEMB1320 evaluation board and evaluation tools to check the operation of Modbus TCP Gateway.

Renesas has evaluated the RX72M communication board [TS-TCS07298 by Tessera Technology] as a Modbus RTU/ASCII slave device in a serial connection (RS-485: Half Duplex) environment.
For more information on RX72M, please refer to “RX72M Communications Board Modbus Startup Manual[R01AN4862****]”.

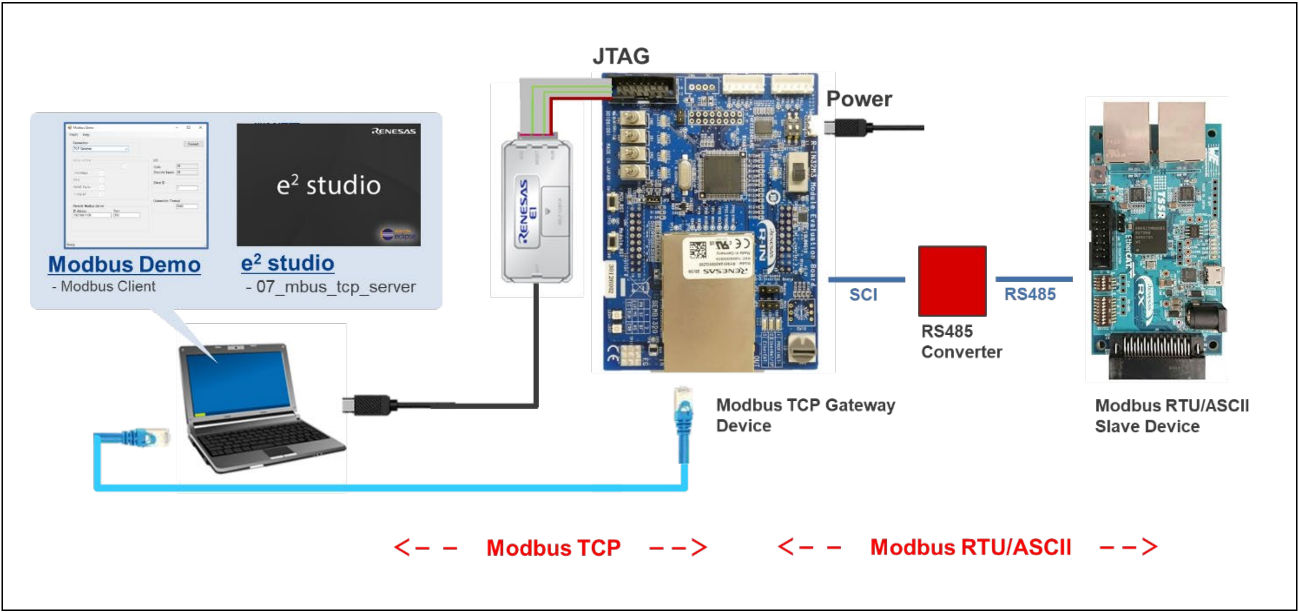


Figure4-4 Modbus TCP Gateway Environment

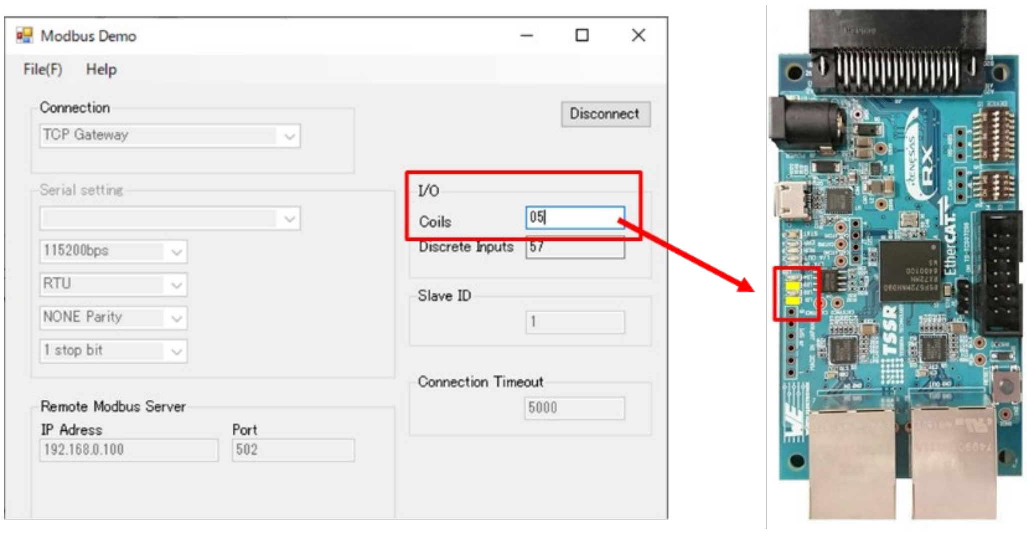
[Note] When Modbus TCP sample app is run, all protocol LEDs (LED1, LED2 and LED3) are off.

2) Modbus Gateway application

As with Modbus TCP, the Modbus Demo tool acts as a Modbus client. However, since instructions from the evaluation tool are sent to the Modbus RTU/ASCII slave device serially connected to the SEMB1320 evaluation board, its operation depends on the target device.

As an example, we will explain an application in combination with the sample software [an-r01an4862xx****-rx72m-modbus] for the RX72M communication board used as a serial slave.

GPIO LEDs 1-4 on the RX72M evaluation board light up with the Coils setting.



Connection: TCP Gateway

Serial setting: 115200bps, RTU, NONE Parity, 1 stop bit

Remote Modbus Server
IP Adress: 192.168.0.100
Port: 502

I/O
Coils: 05
Discrete Inputs: 57

Slave ID: 1

Connection Timeout: 5000

Figure4-5 Modbus TCP Gateway Application by RX72M Serial Slave

3) Modbus Demo tool

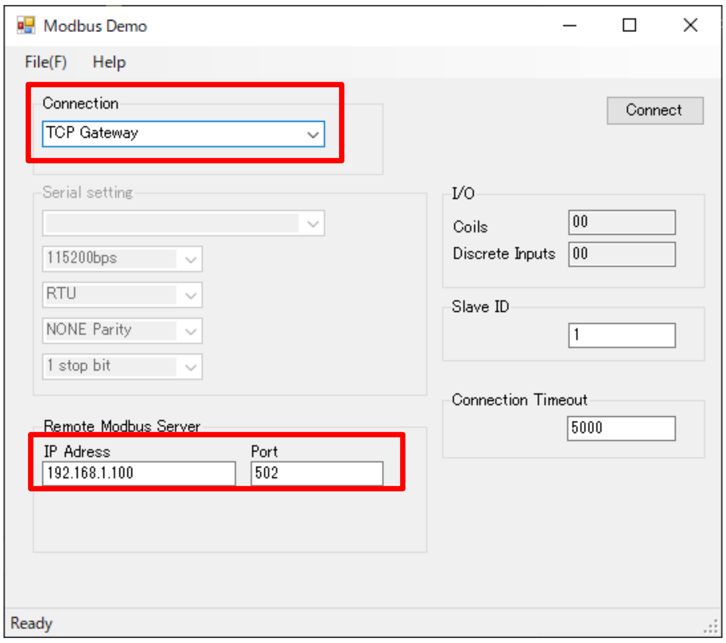
Select the operation mode in "Connection" and set various parameters.

- **Connection**
  - ➢ Select “TCP Gateway”

    * Serial Master and Serial Slave are **not supported** by this sample software.

- **Serial setting**
  - ➢ Do not use.

- **Remote Modbus Server**
  - ➢ Set the IP address and Port number according to the device.



Modbus Demo

Connection: TCP Gateway

Serial setting: 115200bps, RTU, NONE Parity, 1 stop bit

Remote Modbus Server — IP Adress: 192.168.1.100, Port: 502

I/O — Coils: 00, Discrete Inputs: 00

Slave ID: 1

Connection Timeout: 5000

Ready

Figure 2-6 Modbus Demo tool setting [TCP Gateway]

When the sample application is Modbus TCP Gateway, the evaluation tool acts as a Modbus master and operate as follows:

- Run "Connect" and the evaluation tool sends "Read Discrete Inputs" and “Write Multiple Coils” requests.
- Modbus TCP Gateway sample application on RX66T receives a “Read Discrete Inputs” or “Write Multiple Coils” request and issues the same request to the RX72M Modbus serial slave device.
- After issuing a request, it receives a response packet from the Modbus serial slave device and issues a “Read Discrete Inputs” or “Write Multiple Coils” response with the same content to the client in the opposite direction of the request.

Appendix

A. Structure for function codes

Describes the structure definition of the arguments used for each function code callback function registered in Modbus stack.

・Read coils request table (req_read_coils_t)

```
typedef struct _req_read_coils{
        uint16_t    u16_transaction_id;              /* Specifies the transaction ID */
        uint16_t    u16_protocol_id;                 /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                     /* Identification of a remote slave connected */
        uint16_t    u16_start_addr;                  /* Specifies address of the first coil */
        uint16_t    u16_num_of_coils;                /* Specifies the number of coils to be read */
}req_read_coils_t, *p_req_read_coils_t;
```

・Read coils response table (resp_read_coils_t)

```
struct _resp_read_coils{
        uint16_t    u16_transaction_id;              /* Specifies the transaction ID */
        uint16_t    u16_protocol_id;                 /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                     /* Identification of a remote slave connected(Own ID) */
        uint8_t     u8_exception_code;               /* Error detected during processing the request. On
                                                     success the exception code should be zero, if the
                                                     exception code is non-zero, the aru8_data will be null */
        uint8_t     u8_num_of_bytes;                 /* Specifies the number of bytes of data */
        uint8_t     aru8_data[MAX_DISCRETE_DATA];    /* Data to be read */
}resp_read_coils_t, *p_resp_read_coils_t;
```

・Read inputs request table(req_read_inputs_t)

```
typedef struct _req_read_inputs{
        uint16_t    u16_transaction_id;              /* Specifies the transaction ID */
        uint16_t    u16_protocol_id;                 /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                     /* Identification of a remote slave connected */
        uint16_t    u16_start_addr;                  /* Specifies address of the first discrete input */
        uint16_t    u16_num_of_inputs;               /* Specifies the number of discrete inputs to be read */
}req_read_inputs_t, *p_req_read_inputs_t;
```

・Read inputs response table (resp_read_inputs_t)

```
typedef struct _resp_read_inputs{
        uint16_t    u16_transaction_id;               /* Specifies the transaction ID */
        uint16_t    u16_protocol_id;                  /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                      /* Identification of a remote slave connected */
        uint8_t     u8_exception_code;                /* Error detected during processing the request. On
                                                      success the exception code should be zero, if the
                                                      exception code is non-zero the aru8_data will be null */
        uint8_t     u8_num_of_bytes;                  /* Specifies the number of bytes of data */
        uint8_t     aru8_data[MAX_DISCRETE_DATA];     /* Buffer to store the read data */
}resp_read_inputs_t, *p_resp_read_inputs_t;
```

・Read holding registers request table(req_read_holding_reg_t)

```
typedef struct _req_read_holding_reg{
        uint16_t    u16_transaction_id;          /* Specifies the transaction ID */
        uint16_t    u16_protocol_id;             /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                 /* Identification of a remote slave connected */
        uint16_t    u16_start_addr;              /* Specifies address of the first holding register */
        uint16_t    u16_num_of_reg;              /* Specifies the number of registers to be read */
}req_read_holding_reg_t, *p_req_read_holding_reg_t;
```

・Read holding registers response table(resp_read_holding_reg_t)

```
typedef struct _resp_read_holding_reg{
        uint16_t    u16_transaction_id;          /* Specifies the transaction ID */
        uint16_t    u16_protocol_id;             /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                 /* Identification of a remote slave connected */
        uint8_t     u8_exception_code;           /* error detected during processing the request. On success
                                                 the exception code should be zero, if the exception code is
                                                 non-zero the aru16_data will be null */
        uint8_t     u8_num_of_bytes;             /* specifies the number of bytes of data */
        uint16_t    aru16_data[MAX_REG_DATA];    /* buffer to store the read data */
}resp_read_holding_reg_t, p_resp_read_holding_reg_t;
```

・Read input registers request table (req_read_input_reg_t)

```
typedef struct _req_read_input_reg{
        uint16_t    u16_transaction_id;          /* Specifies the transaction ID */
        uint16_t    u16_protocol_id;             /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                 /* Identification of a remote slave connected */
        uint16_t    u16_start_addr;              /* Specifies address of the first input register */
        uint16_t    u16_num_of_reg;              /* Specifies the number of registers to be read */
}req_read_input_reg_t, *p_req_read_input_reg_t;
```

・Read input registers response table(resp_read_input_reg_t)

```
typedef struct _resp_read_input_reg{
        uint16_t    u16_transaction_id;          /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;             /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                 /* Identification of a remote slave connected */
        uint8_t     u8_exception_code;           /* Error detected during processing the request. On success
                                                 the exception code should be zero, if the exception code is
                                                 non-zero the aru16_data will be null */
        uint8_t     u8_num_of_bytes;             /* Specifies the number of bytes of data */
        uint16_t    aru16_data[MAX_REG_DATA];    /* Buffer to store the read data */
}resp_read_input_reg_t, p_resp_read_input_reg_t;
```

・Write single coil request table(req_write_single_coil_t)

```
typedef struct _req_write_single_coil
        uint16_t     u16_transaction_id;       /* Specifies the transaction ID  */
        uint16_t     u16_protocol_id;          /* Specifies the protocol ID */
        uint8_t      u8_slave_id;              /* Identification of a remote slave connected */
        uint16_t     u16_output_addr;          /* Specifies address of the coil */
        uint16_t     u16_output_value;         /* Data to be written */
}req_write_single_coil_t, *p_req_write_single_coil_t;
```

・Write single coil response table(resp_write_single_coil_t)

```
typedef struct _resp_write_single_coil{
        uint16_t     u16_transaction_id;       /*  Specifies the transaction ID  */
        uint16_t     u16_protocol_id;          /* Specifies the protocol ID */
        uint8_t      u8_slave_id;              /* Identification of a remote slave connected */
        uint8_t      u8_exception_code;        /* Error detected during processing the request. On success the
                                               exception code should be zero */
        uint16_t     u16_output_addr;          /* Specifies address of the coil */
        uint16_t     u16_output_value;         /* Data to be written */
}resp_write_single_coil_t, *p_resp_write_single_coil_t;
```

・Write single register request table(req_write_single_reg_t)

```
typedef struct _req_write_single_reg{
        uint16_t     u16_transaction_id;     /*  Specifies the transaction ID  */
        uint16_t     u16_protocol_id;        /* Specifies the protocol ID */
        uint8_t      u8_slave_id;            /* Identification of a remote slave connected */
        uint16_t     u16_register_addr;      /* Specifies address of the register */
        uint16_t     u16_register_value;     /* Data to be written */
}req_write_single_reg_t, *p_req_write_single_reg_t;
```

・Write single register response table(resp_write_single_reg_t)

```
typedef struct _resp_write_single_reg{
        uint16_t    u16_transaction_id;     /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;        /* Specifies the protocol ID */
        uint8_t     u8_slave_id;            /* Identification of a remote slave connected */
        uint8_t     u8_exception_code;      /* Error detected during processing the request. On success the
                                            exception code should be zero */
        uint16_t    u16_register_addr;      /* Specifies address of the register */
        uint16_t    u16_register_value;     /* Data to be written */
}resp_write_single_reg_t, *p_resp_write_single_reg_t;
```

・Write multiple coils request table(req_write_multiple_coils_t)

```
typedef struct _req_write_single_reg{
        uint16_t    u16_transaction_id;               /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;                  /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                      /* Identification of a remote slave connected */
        uint16_t    u16_start_addr;                   /* Specifies address of the first coil */
        uint16_t    u16_num_of_outputs;               /* Specifies the number of coils to be written */
        uint8_t     u8_num_of_bytes;                  /* Specifies the number of bytes of data */
        uint8_t     aru8_data[MAX_DISCRETE_DATA];     /* Data to be written */
}req_write_single_reg_t, *p_req_write_single_reg_t;
```

・Write multiple coils response table(resp_write_multiple_coils_t)

```
typedef struct _resp_write_multiple_coils{
        uint16_t    u16_transaction_id;               /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;                  /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                      /* Identification of a remote slave connected */
        uint8_t     u8_exception_code;                /* Error detected during processing the request. On
                                                      success the exception code should be zero */
        uint16_t    u16_start_addr;                   /* Specifies address of the coils */
        uint16_t    u16_num_of_outputs;               /* Specifies the number of coils to be written */
}resp_write_multiple_coils_t, *p_resp_write_multiple_coils_t;
```

・Write multiple registers request table(req_write_multiple_reg_t)

```
typedef struct _req_write_multiple_reg{
        uint16_t    u16_transaction_id;          /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;             /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                 /* Identification of a remote slave connected */
        uint16_t    u16_start_addr;              /* Specifies address of the first register */
        uint16_t    u16_num_of_reg;              /* Specifies the number of registers to be written */
        uint8_t     u8_num_of_bytes;             /* Specifies the number of bytes of data */
        uint16_t    aru16_data[MAX_REG_DATA];    /* Data to be written */
}req_write_multiple_reg_t, *p_req_write_multiple_reg_t;
```

・Write multiple registers response table(resp_write_multiple_reg_t)

```
typedef struct _resp_write_multiple_reg{
        uint16_t    u16_transaction_id;         /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;            /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                /* Identification of a remote slave connected */
        uint8_t     u8_exception_code;          /* Error detected during processing the request. On success the
                                                exception code should be zero */
        uint16_t    u16_start_addr;             /* Specifies address of the first register */
        uint16_t    u16_num_of_reg;             /* Specifies the number of registers to be written */
}resp_write_multiple_reg_t, *p_resp_write_multiple_reg_t;
```

・Read/Write multiple registers request table(req_read_write_multiple_reg_t)

```
typedef struct _req_read_write_multiple_reg{
        uint16_t    u16_transaction_id;             /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;                /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                    /* Identification of a remote slave connected */
        uint16_t    u16_read_start_addr;            /* Specifies address of the first register to be read from */
        uint16_t    u16_num_to_read;                /* Specifies the number of registers to be read */
        uint16_t    u16_write_start_addr;           /* Specifies address of the first register to be written */
        uint16_t    u16_num_to_write;               /* Specifies the number of registers to be written */
        uint8_t     u8_write_num_of_bytes;          /* Specifies the number of bytes of data */
        uint16_t    aru16_data[MAX_REG_DATA];       /* Data to be written */
}req_read_write_multiple_reg_t, *p_req_read_write_multiple_reg_t;
```

・Read/Write multiple registers response table(resp_read_write_multiple_reg_t)

```
typedef struct _resp_read_write_multiple_reg{
        uint16_t    u16_transaction_id;                 /*  Specifies the transaction ID  */
        uint16_t    u16_protocol_id;                    /* Specifies the protocol ID */
        uint8_t     u8_slave_id;                        /* Identification of a remote slave connected */
        uint8_t     u8_exception_code;                  /* Error detected during processing the request. On
                                                        success the exception code should be zero, if the
                                                        exception code is non-zero the aru16_read_data will be
                                                        null */
        uint16_t    u8_num_of_bytes;                    /* Specifies the number of complete bytes of data */
        uint16_t    aru16_read_data[MAX_REG_DATA];      /* Data to be read */
}resp_read_write_multiple_reg_t, *p_resp_read_write_multiple_reg_t;
```

B. API specification

Details the specifications of the Modbus stack-related Application Programming Interfaces (API) used in this document.

1) Stack Initialization

API used to initialize the Modbus stack

Initialize the Modbus stack as Modbus TCP

r_modbus_tcp_init_stack	Modbus TCP stack initialization

【FORMAT】

```
uint32_t r_modbus_tcp_init_stack(uint8_t  u8_stack_mode,
                                 uint8_t  u8_tcp_multiple_client,
                                 uint32_t u32_additional_port)
```

【ARGUMENT】

uint8_t	u8_stack_mode	Set stack mode
uint8_t	u8_tcp_multiple_client	Set Multi client
uint32_t	u32_additional_port	Additional port Number

【RETURN】

uint32_t	Error code

【Error Code】

ERR_OK	Success
ERR_STACK_INIT	Fail

【Comment】 Argument:

u8_stack_mode: **MODBUS_TCP_SERVER_MODE** fixed

u8_tcp_multiple_client: Set whether to accept communication from multiple clients.

macro	
ENABLE_MULTIPLE_CLIENT_CONNECTION	Enable Multiclient
DISABLE_MULTIPLE_CLIENT_CONNECTION	Disable Multiclient

u32_additional_port: Using a communication port other than the default [502].
Set 0 when not adding.

Initialize the Modbus stack as Modbus TCP Gateway

r_modbus_tcp_init_gateway_stack Modbus TCP Gateway stack initialization

【FORMAT】

```
uint32_t r_modbus_tcp_init_gateway_stack(uint8_t  u8_stack_mode,
                                        uint8_t  u8_tcp_multiple_client,
                                        uint32_t u32_additional_port);
```

【ARGUMENT】

uint8_t	u8_stack_mode	Set stack mode
uint8_t	u8_tcp_multiple_client	Set Multi client
uint32_t	u32_additional_port	Additional port Number

【RETURN】

uint32_t	Error Code

【Error Code】

ERR_OK	Success
ERR_INVALID_STACK_MODE	Invalid stack mode
ERR_INVALID_SLAVE_ID	Invalid Slave ID
ERR_INVALID_STACK_INIT_PARAMS	Invalid user set parameter
ERR_STACK_INIT	Fail

【Comment】 Argument:

u8_stack_mode: Set the Serial master mode

Stack mode	
MODBUS_RTU_MASTER_MODE	RTU master mode
MODBUS_ASCII_MASTER_MODE	ASCII master mode

u8_tcp_multiple_client: Set whether to accept communication from multiple clients.

macro	
ENABLE_MULTIPLE_CLIENT_CONNECTION	Enable Multiclient
DISABLE_MULTIPLE_CLIENT_CONNECTION	Disable Multiclient

u32_additional_port: Using a communication port other than the default [502].
Set 0 when not adding.

Associate the processing request for each function code from the client with the user-defined function.

r_modbus_slave_map_init	Function code mapping

【FORMAT】

uint32_t r_modbus_slave_map_init(p_slave_map_init_t pt_slave_func_req);

【ARGUMENT】

p_slave_map_init_t	pt_slave_func_req	Pointer of Function code mapping table

【RETURN】

uint32_t	Error Code

【Error Code】

ERR_OK	Success
ERR_INVALID_STACK_INIT_PARAMS	Argument is NULL

【Comment】 Argument:

Calls the registered function when the Modbus slave stack receives a request.

This API is valid only in slave mode.

・Function Code mapping table (*_slave_map_init_t*)

```
typedef struct _slave_map_init{
        fp_function_code1_t     fp_function_code1;      /* function code 1 (Read Coils) */
        fp_function_code2_t     fp_function_code2;      /* function code 2 (Read Discrete Inputs) */
        fp_function_code3_t     fp_function_code3;      /* function code 3 (Read Holding Registers) */
        fp_function_code4_t     fp_function_code4;      /* function code 4 (Read Input Registers) */
        fp_function_code5_t     fp_function_code5;      /* function code 5 (Write Single Coil) */
        fp_function_code6_t     fp_function_code6;      /* function code 6 (Write Single Register) */
        fp_function_code15_t    fp_function_code15;     /* function code 15 (Write Multiple Coils) */
        fp_function_code16_t    fp_function_code16;     /* function code 16 (Write Multiple Registers) */
        fp_function_code23_t    fp_function_code23;     /* function code 23 (Read/Write Multiple Registers) */
}slave_map_init_t, *p_slave_map_init_t;
```

The callback function corresponding to each function code is defined in the following format. See Appendix. A for details on the structure used for each callback function argument.

function code 1 (Read Coils)

fp_function_code1_t	Callback for Function Code 1 (Read coils)

【FORMAT】

```
uint32_t (*fp_function_code1_t)(p_req_read_coils_t pt_req_read_coils,
                                p_resp_read_coils_t pt_resp_read_coils );
```

【ARGUMENT】

p_req_read_coils_t	pt_req_read_coils	Read coil request pointer
p_resp_read_coils_t	pt_resp_read_coils	Read coil response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 2 (Read Discrete Inputs)

fp_function_code2_t	Callback for Function 2(Read discrete inputs)

【FORMAT】

```
uint32_t (*fp_function_code2_t)(p_req_read_inputs_t pt_req_read_inputs,
                                p_resp_read_inputs_t pt_resp_read_inputs );
```

【ARGUMENT】

p_req_read_inputs_t	pt_req_read_inputs	Read discrete inputs request pointer
p_resp_read_inputs_t	pt_resp_read_inputs	Read discrete inputs response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 3 (Read Holding Registers)

fp_function_code3_t	Callback for Function 3(Read holding register)

【FORMAT】

```
uint32_t (*fp_function_code3_t)(p_req_read_holding_reg_t pt_req_read_holding_reg,
                                p_resp_read_holding_reg_t pt_resp_read_holding_reg);
```

【ARGUMENT】

p_req_read_holding_reg_t	pt_req_read_holding_reg	Read holding register request pointer
p_resp_read_holding_reg_t	pt_resp_read_holding_reg	Read holding register response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 4 (Read Input Registers)

fp_function_code4_t	Callback for Function 4(Read input register)

【FORMAT】

```
uint32_t (*fp_function_code4_t)(p_req_read_input_reg_t pt_req_read_input_reg,
                                p_resp_read_input_reg_t pt_resp_read_input_reg);
```

【ARGUMENT】

p_req_read_input_reg_t	pt_req_read_input_reg	Read Input register request pointer
p_resp_read_input_reg_t	pt_resp_read_input_reg	Read holding register response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 5 (Write Single Coil)

fp_function_code5_t	Callback for Function 5(Write single coil)

【FORMAT】

```
uint32_t (*fp_function_code5_t)(p_req_write_single_coil_t pt_req_write_single_coil,
                                p_resp_write_single_coil_t pt_resp_write_single_coil );
```

【ARGUMENT】

p_req_write_single_coil_t	pt_req_write_single_coil	Write single coil request pointer
p_resp_write_single_coil_t	pt_resp_write_single_coil	Write single coil response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 6 (Write Single Register)

fp_function_code6_t	Callback for Function 6(Write single register)

【FORMAT】

```
uint32_t (*fp_function_code6_t)(p_req_write_single_reg_t pt_req_write_single_reg,
                                p_resp_write_single_reg_t pt_resp_write_single_reg);
```

【引数】

p_req_write_single_reg_t	pt_req_write_single_reg	Write single register request pointer
p_resp_write_single_reg_t	pt_resp_write_single_reg	Write single register response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 15 (Write Multiple Coils)

fp_function_code15_t	Callback for Function 15(Write multiple coils)

【FORMAT】

```
uint32_t (*fp_function_code15_t) (p_req_write_multiple_coils_t pt_req_write_multiple_coils,
                                  p_resp_write_multiple_coils_t pt_resp_write_multiple_coils);
```

【ARGUMENT】

p_req_write_multiple_coils_t	pt_req_write_multiple_coils	Write multiple coils request pointer
p_resp_write_multiple_coils_t	pt_resp_write_multiple_coils	Write multiple coils response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 16 (Write Multiple Registers)

fp_function_code16_t	Callback for Function 16(Write multiple registers)

【FORMAT】

```
uint32_t (*fp_function_code16_t) (p_req_write_multiple_reg_t pt_req_write_multiple_reg,
                                  p_resp_write_multiple_reg_t pt_resp_write_multiple_reg);
```

【引数】

p_req_write_multiple_reg_t	pt_req_write_multiple_reg	Write multiple registers request pointer
p_resp_write_multiple_reg_t	pt_resp_write_multiple_reg	Write multiple registers response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

function code 23 (Read/Write Multiple Registers)

fp_function_code23_t	Callback for Function 23(Read/Write multiple registers)

【FORMAT】

```
uint32_t (*fp_function_code23_t) (p_req_read_write_multiple_reg_t pt_req_read_write_multiple_reg,
                                  p_resp_read_write_multiple_reg_t pt_resp_read_write_multiple_reg);
```

【ARGUMENT】

p_req_read_write_multiple_reg_t	pt_req_read_write_multiple_reg	Read/Write multiple registers request pointer
p_resp_read_write_multiple_reg_t	pt_resp_read_write_multiple_reg	Read/Write multiple registers response pointer

【RETURN】

uint32_t	0 : Success,1 : Fail

2) IP address list

API used to IP address List function

Specify whether to enable/disable the IP address list function and access rights for the IP addresses registered in the list.

r_modbus_tcp_init_ip_table	IP address list function status setting

【FORMAT】

```
void_t r_modbus_tcp_init_ip_table(ENABLE_FLAG e_flag,
                                  TABLE_MODE e_mode);
```

【ARGUMENT】

ENABLE_FLAG	e_flag	Enabling/disabling the IP address list ENABLE, DISABLE
TABLE_MODE	e_mode	Specify access rights for IP addresses included in the list ACCEPT, REJECT

【RETURN】

void_t

【RETURN】

—

【Comment】 Argument:

If the list function is enabled and the mode specification is access permission, the list function works as a whitelist. Also, if the list function is enabled and the mode is set to prohibit access, the list function will operate as a blacklist. The IP address list function is disabled by default

Adding the IP address list

r_modbus_tcp_add_ip_addr	Adding the IP address list

【FORMAT】

```
uint32_t r_modbus_tcp_add_ip_addr(uint32_t u32_host_ip);
```

【ARGUMENT】

uint32_t	u32_host_ip	Set IP address (IPv4) GOAL_NET_IPV4(192,168,1,100)

【RETURN】

uint32_t	Error Code

【RETURN】

ERR_OK	Success
ERR_IP_ALREADY_PRESENT	IP address is already listed
ERR_MAX_CLIENT	Full registrations
ERR_TABLE_DISABLED	Invalid host IP list

【Comment】 Argument:

Use **GOAL_NET_IPV4** macro to Set the IP address.

Checks whether the specified IP address is a target IP address for the IP address list function.

r_modbus_chk_connectable_ip	IP address list function determination

【FORMAT】

GOAL_STATUS_T r_modbus_chk_connectable_ip(uint32_t ipaddr);

【ARGUMENT】

uint32_t	ipaddr	Set IP address (IPv4) GOAL_NET_IPV4(192,168,1,100)

【RETURN】

GOAL_STATUS_T	Error Code

【RETURN】

GOAL_OK	Access permission
GOAL_ERR_BUSY	Access denial

3) TCP connection management

API used to management the TCP connection

TCP connection management is done with GOAL_ENT_CHAN_T instances created by uGOAL.

Determines whether a running Modbus stack can accept new TCP connections.

r_modbus_tcp_multi_connection	TCP connection availability judgment

【FORMAT】

GOAL_STATUS_T r_modbus_tcp_multi_connection(void);

【ARGUMENT】		
void	—	—

【RETURN】	
GOAL_STATUS_T	Error Code

【Error Code】	
GOAL_OK	Connection permission state
GOAL_ERR_BUSY	connection prohibited state

【Comment】 Return:

GOAL_OK if the connection is possible.

GOAL_ERR_BUSY if multi-connection is not supported and TCP connection is already established.

Register a TCP connection in the connection list.

r_modbus_tcp_reg_connection_list	Registration to the connection list

【FORMAT】

void r_modbus_tcp_reg_connection_list(GOAL_NET_CHAN_T *pChan);

【ARGUMENT】		
GOAL_NET_CHAN_T*	pChan	TCP connection instance

【RETURN】	
GOAL_STATUS_T	Error Code

【Error Code】	
GOAL_OK	Success
GOAL_ERR_FULL	Full registration

【Comment】 Return:

GOAL_OK: successful registration.

GOAL_ ERR_FULL: If the number of TCP connections registered in the connection list has already reached the value specified by **MAXIMUM_NUMBER_OF_CLIENTS**

Removes the specified TCP connection from the connection list.

r_modbus_tcp_del_connection_list	Remove the connection

【FORMAT】

void r_modbus_tcp_del_connection_list(GOAL_NET_CHAN_T *pChan);

【ARGUMENT】

GOAL_NET_CHAN_T*	pChan	TCP connection instance

【RETURN】

void		Error code

Returns the oldest TCP connection instance registered in the connection list.

r_modbus_tcp_get_oldest_connection	Get Oldest TCP Connection

【FORMAT】

GOAL_NET_CHAN_T *r_modbus_tcp_get_oldest_connection(void);

【ARGUMENT】

void	—	—

【RETURN】

GOAL_NET_CHAN_T*	pChan	TCP connection instance

4) Modbus packet analysis

API to analysis Modbus packet.

Bridge function that passes TCP packets received by uGOAL to the Modbus stack for execution.

appl_parseModbusPacket	Modbus packet analysis

【FORMAT】

```
GOAL_STATUS_T appl_parseModbusPacket(
  GOAL_MA_CHAN_TCP_T *pMaTcpHdl,
  GOAL_NET_CHAN_T *pChan,
  GOAL_BUFFER_T *pBuf);
```

【ARGUMENT】

GOAL_MA_CHAN_TCP_T *	pMaTcpHdl	Pointer of MA handler
GOAL_NET_CHAN_T *	pChan	Pointer of instance which is stored the TCP connection information
GOAL_BUFFER_T *	pBuf	GOAL buffer which is stored TCP packet

【RETURN】

GOAL_STATUS_T	Error code

【ERROR CODE】

GOAL_OK	Success

【Comment】

Since packet analysis is performed based on TCP information generated by uGOAL, it is called as part of data processing from the TCP callback process (tcpCallback corresponds to this sample) registered in uGOAL.

If the stack is initialized as Modbus TCP, after packet analysis, the user function registered at initialization is executed as a slave operation. A response packet is then generated and sent.

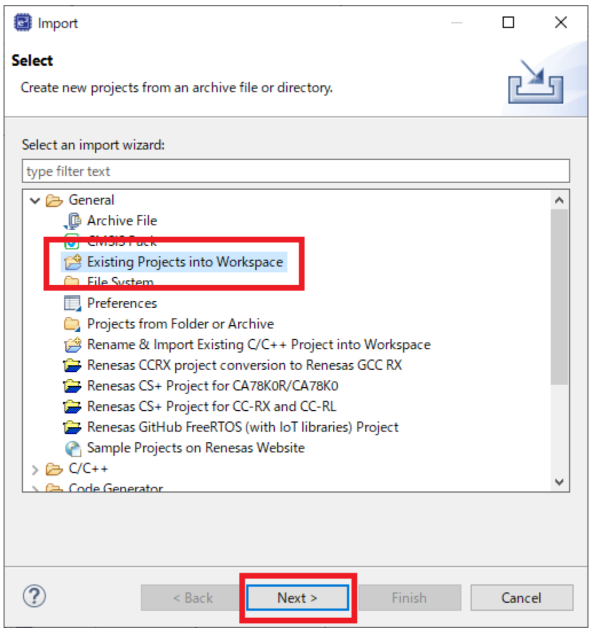
If the stack is initialized as a Modbus gateway, it converts received packets to Modbus RTU or Modbus ASCII packets and sends them to the specified serial slave device. Also, when it receives a response packet from a slave device, it converts the received packet to a Modbus TCP packet and sends it to the Modbus master side.

C. Sample software startup

Describes how to operate the integrated development environment e2studio.
Complete the hardware connection according to the stack mode to operate in advance. Please refer to 4 **Communication Test with Evaluation Tool**.

This manual describes the SEMB1320 as the target. For the RX72M communication board used as an example of a Modbus RTU/ASCII slave device in Modbus TCP gateway mode, refer to "RX72M Group Communication Board Modbus Startup Manual (R01AN4862****)".

1) After launching e2studio, click "File" → "Import".
2) In the Select dialog, select "General" → "Existing Projects into Workspace" and click Next.


Import

Select

Create new projects from an archive file or directory.

Select an import wizard:

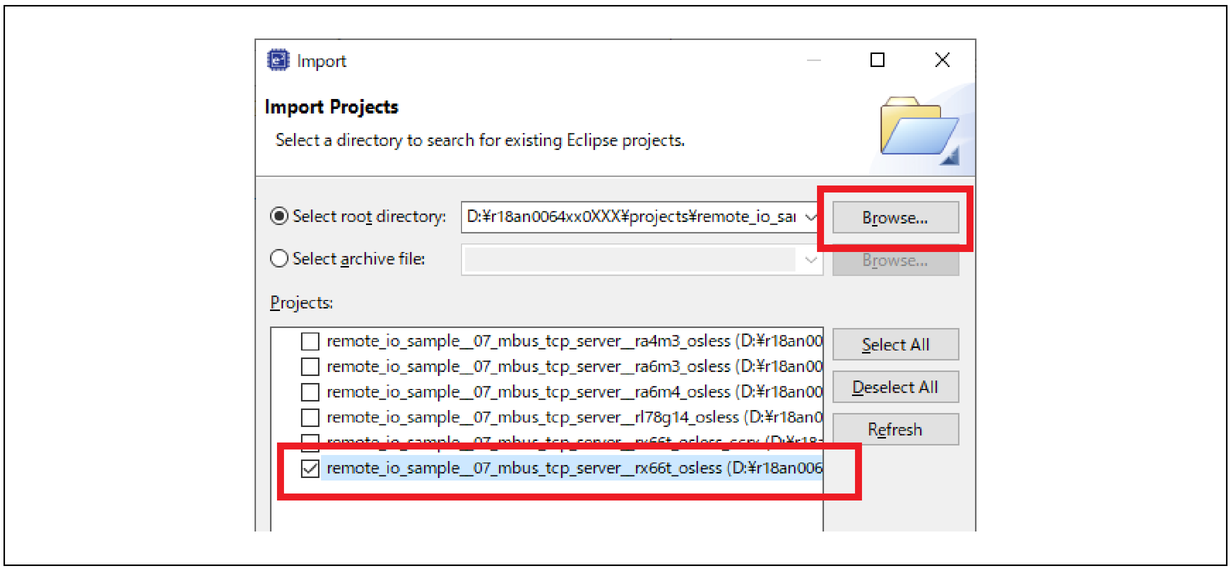
type filter text

- General
  - Archive File
  - CMSIS Pack
  - Existing Projects into Workspace
  - File System
  - Preferences
  - Projects from Folder or Archive
  - Rename & Import Existing C/C++ Project into Workspace
  - Renesas CCRX project conversion to Renesas GCC RX
  - Renesas CS+ Project for CA78K0R/CA78K0
  - Renesas CS+ Project for CC-RX and CC-RL
  - Renesas GitHub FreeRTOS (with IoT libraries) Project
  - Sample Projects on Renesas Website
- C/C++
- Code Generator

< Back | Next > | Finish | Cancel

Figure C-1 Import Project

3) Select the “Select Root Directory” check box in the Import Project dialog and click “Browse”. Select the project that matches environment and click "Open". Click Finish to complete the project import.



[✓ Select root directory:] D:¥r18an0064xx0XXX¥projects¥remote_io_sai

Select archive file:

- [ ] remote_io_sample__07_mbus_tcp_server__ra4m3_osless (D:¥r18an00
- [ ] remote_io_sample__07_mbus_tcp_server__ra6m3_osless (D:¥r18an00
- [ ] remote_io_sample__07_mbus_tcp_server__ra6m4_osless (D:¥r18an00
- [ ] remote_io_sample__07_mbus_tcp_server__rl78g14_osless (D:¥r18an0
- [ ] remote_io_sample__07_mbus_tcp_server__rx66t_osless_ccrx (D:¥r18a
- [x] remote_io_sample__07_mbus_tcp_server__rx66t_osless (D:¥r18an006

Figure C-2 Select Project

The project file for Modbus TCP is stored in the following directory.

Modbus TCP project File:

RX66T_uCCM_V*** \ projects \ remote_io_sample \ **07_mbus_tcp_server**

4) Right-click the sample project and select the stack mode to run from "Activate" in "Build Configuration".

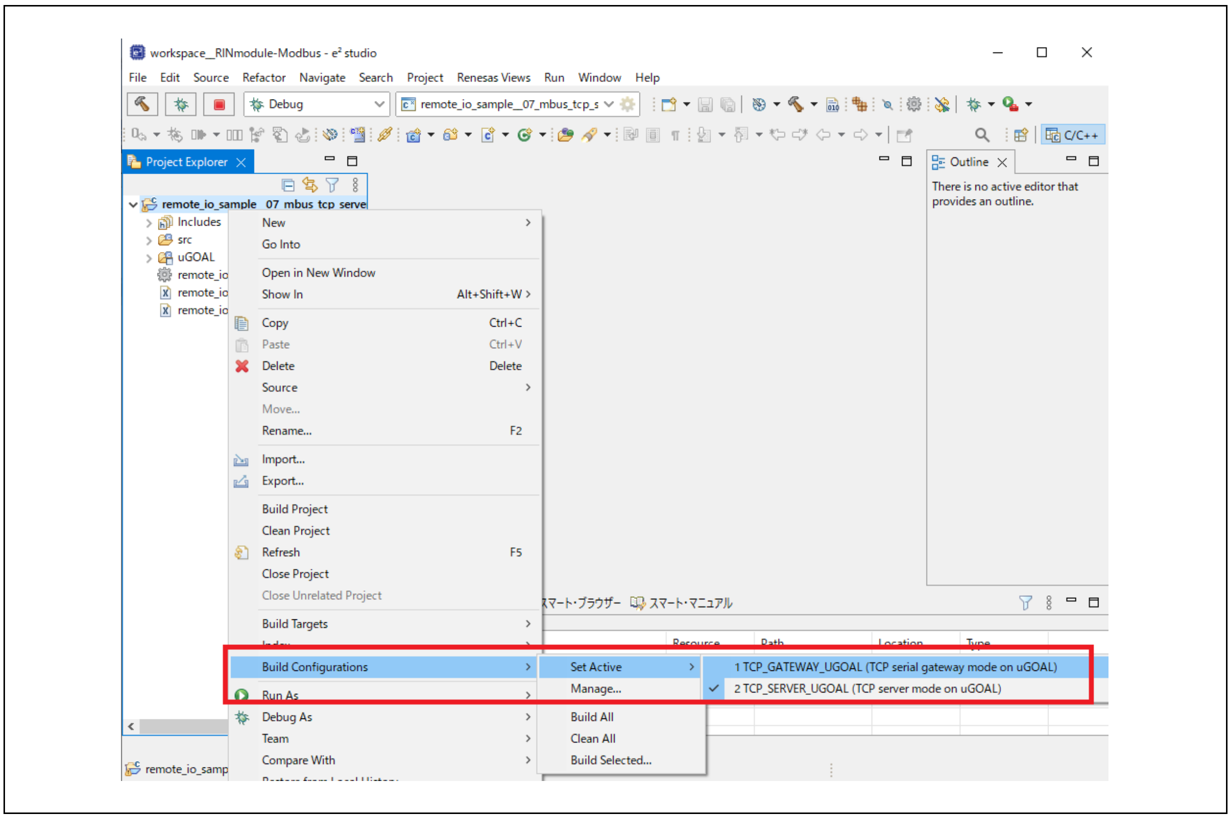
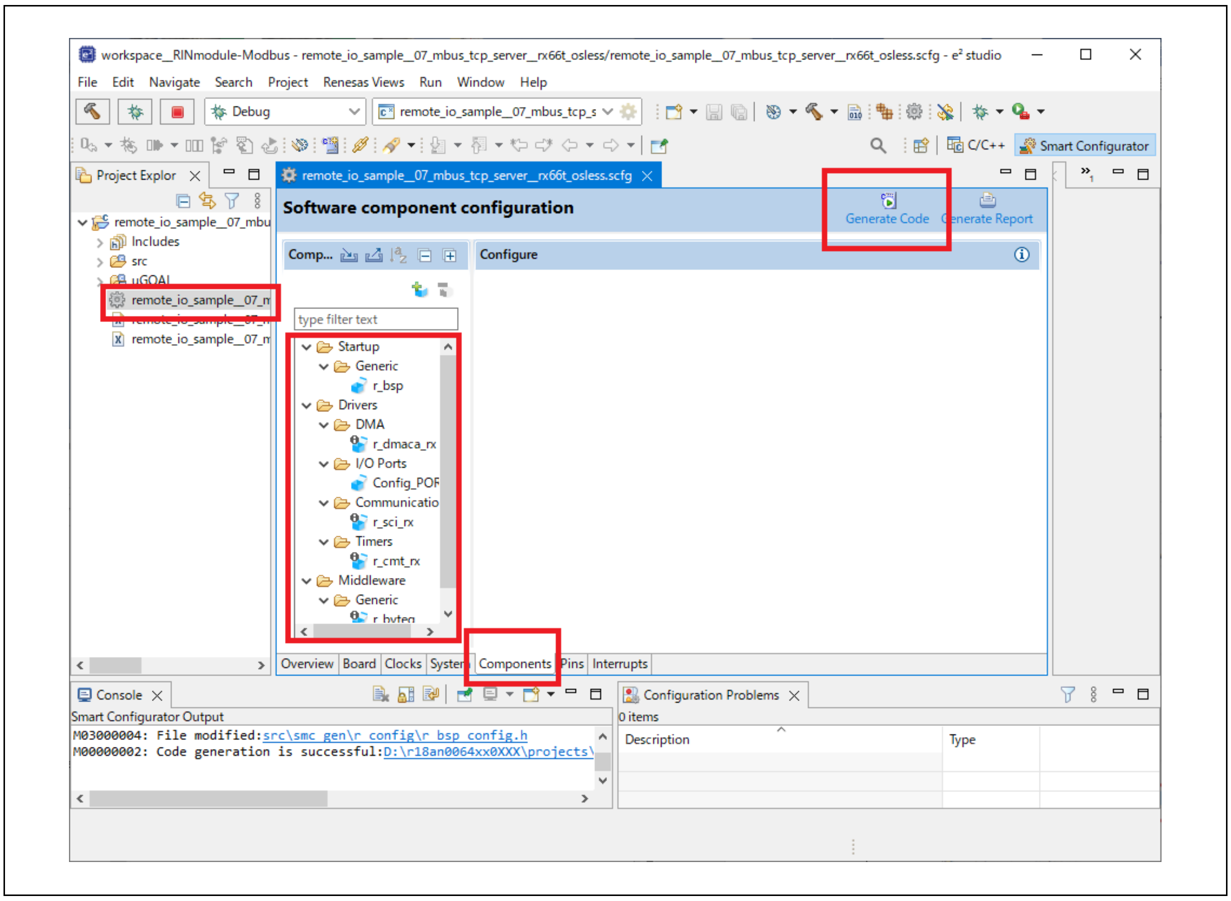

workspace_RINmodule-Modbus - e² studio
File Edit Source Refactor Navigate Search Project Renesas Views Run Window Help
Debug
remote_io_sample_07_mbus_tcp_s
Project Explorer
Outline
There is no active editor that provides an outline.
remote_io_sample_07_mbus_tcp_serve
Includes
src
uGOAL
New
Go Into
Open in New Window
Show In Alt+Shift+W
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Project
Build Targets
Build Configurations
Run As
Debug As
Team
Compare With
Set Active
Manage...
Build All
Clean All
Build Selected...
1 TCP_GATEWAY_UGOAL (TCP serial gateway mode on uGOAL)
2 TCP_SERVER_UGOAL (TCP server mode on uGOAL)
スマート・ブラウザー スマート・マニュアル
Resource Path Location Type
C/C++

Figure C-3 Set Build configuration [Set Stack mode]

Table C-1 Build structure and Stack mode

Build Name	Stack mode	Connection Type
TCP_GATEWAY_UGOAL	Modbus TCP Gateway Stack	2.6.2 Modbus TCP Gateway Stack
TCP_SERVER_UGOAL	Modbus TCP Server stack	2.6.1 Modbus TCP Server Stack

5) Double-click the configuration file to open the "Configuration" window.
   Click "Generate Code" to generate a pre-registered driver. Confirm that the various drivers registered in "Components" are activated (the icon turns light blue).



```
M03000004: File modified:src\smc_gen\r_config\r_bsp_config.h
M00000002: Code generation is successful:D:\r18an0064xx0XXX\projects\
```

Figure C-4 Generate Code

6) Select "Build Project" to run the build.
   If the build finishes without any errors, it is successful.

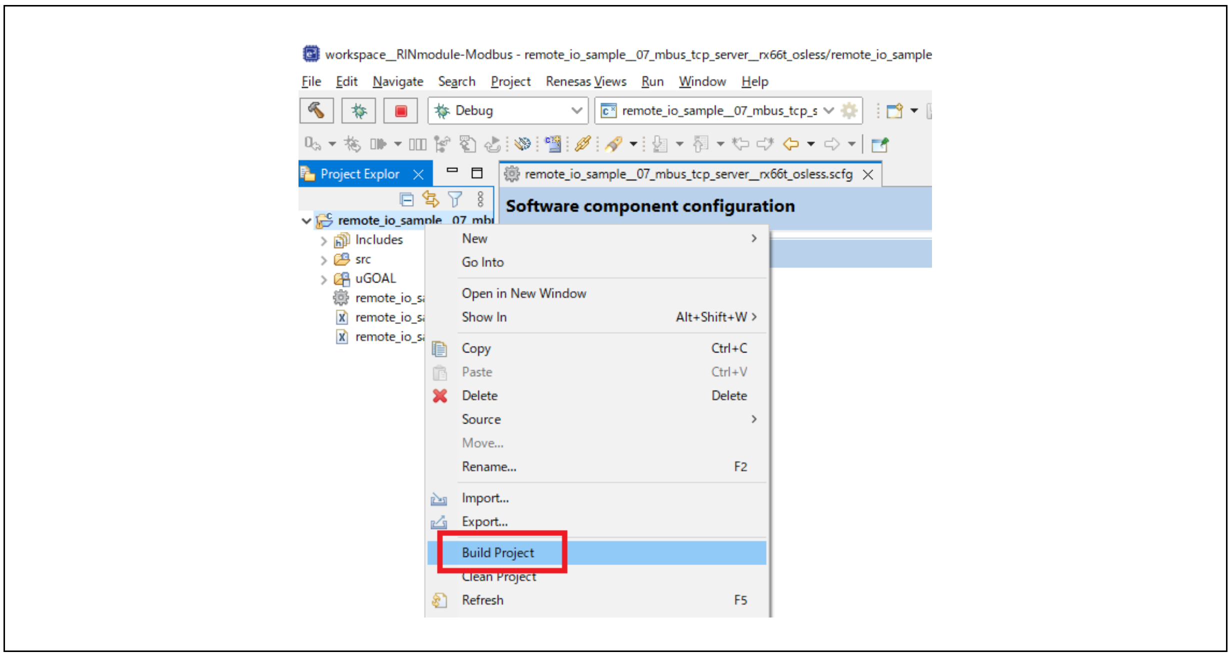
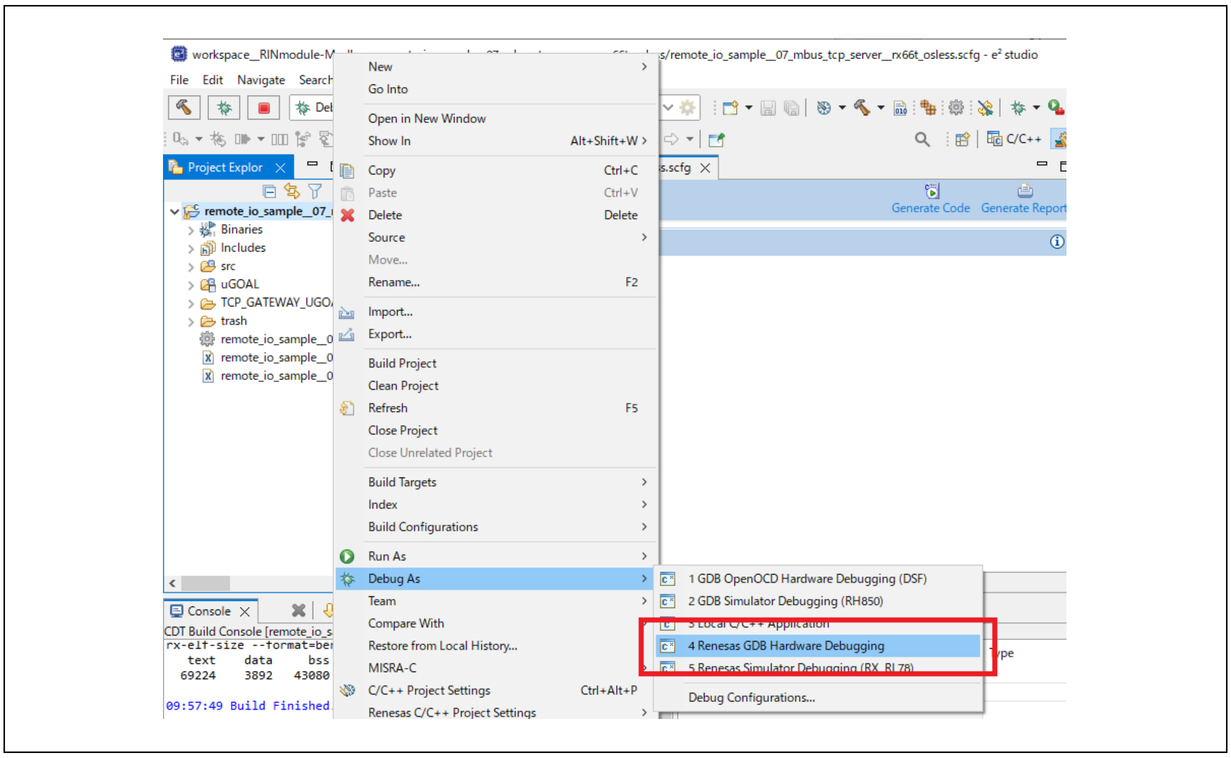

workspace__RINmodule-Modbus - remote_io_sample__07_mbus_tcp_server__rx66t_osless/remote_io_sample
File Edit Navigate Search Project Renesas Views Run Window Help
Debug
Project Explor
remote_io_sample__07_mbus_tcp_server__rx66t_osless.scfg
Software component configuration
New
Go Into
Open in New Window
Show In Alt+Shift+W
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5

Figure C-5 Build

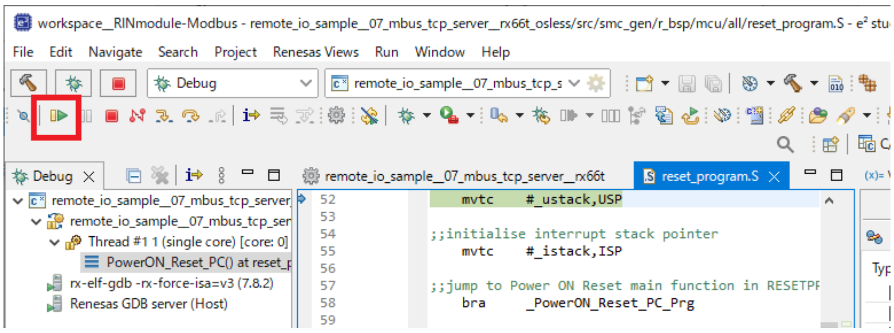
7) Select "Renesas GDB Hardware Debugging" from "Debug" and execute the program download.
   If the download finishes without any errors, it is successful.



```
CDT Build Console [remote_io_s
rx-elf-size --format=ber
   text    data     bss
  69224    3892   43080
09:57:49 Build Finished
```

Figure C-6 Program Download

8) Click “Resume” to run the program.


workspace__RINmodule-Modbus - remote_io_sample__07_mbus_tcp_server__rx66t_osless/src/smc_gen/r_bsp/mcu/all/reset_program.S - e² stu
File Edit Navigate Search Project Renesas Views Run Window Help
Debug
remote_io_sample__07_mbus_tcp_s
Debug
remote_io_sample__07_mbus_tcp_server
remote_io_sample__07_mbus_tcp_ser
Thread #1 1 (single core) [core: 0]
PowerON_Reset_PC() at reset_p
rx-elf-gdb -rx-force-isa=v3 (7.8.2)
Renesas GDB server (Host)
remote_io_sample__07_mbus_tcp_server__rx66t
reset_program.S
Typ

```
    mvtc    #_ustack,USP

;;initialise interrupt stack pointer
    mvtc    #_istack,ISP

;;jump to Power ON Reset main function in RESETPF
    bra     _PowerON_Reset_PC_Prg
```

Figure C-7 Program Run

This completes starting the sample program for the R-IN32M3 module evaluation environment SEMB1320.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan/07/2022	-	First Edition
2.00	Apr/05/2022	-	Add Modbus Gateway function

Trademark

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks