RL78/G16

Modbus ASCII/RTU

Introduction

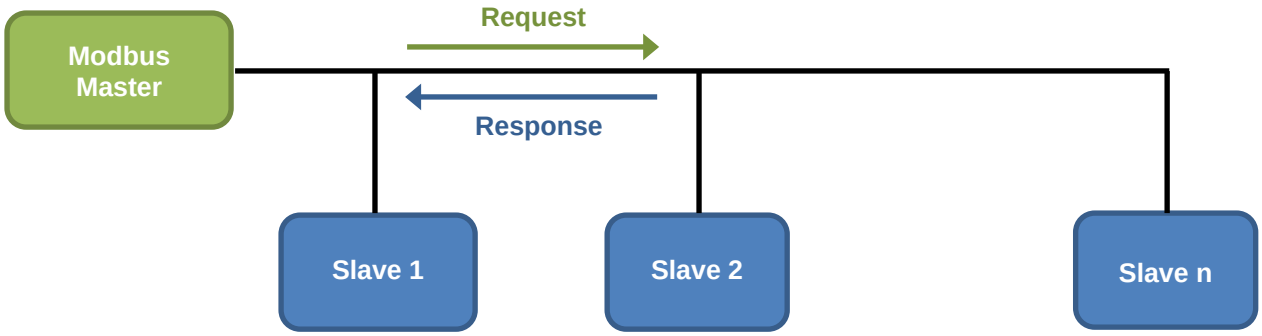
Modbus is a communication protocol developed by Modicon Inc. (AEG Schneider Automation International SAS) for PLCs. It is used for the purpose of data transfer, not only for PLCs but also between electronic devices. It is widely used in the field of factory automation and plant automation because the specifications are open to the public, free to use, and relatively easy to implement. This makes it the most common serial communication protocol for connecting industrial equipment.

For example, in a factory, Modbus is used to connect devices through various wired or wireless communication gateways. The master device used by Modbus can be a gateway, HMI, SCADA (Supervisor Control and Data Acquisition), PLC, or similar device. The slave devices can include I/O control devices, temperature and humidity measuring machines, meters, frequency transformers, motor units, etc.

There are two Modbus transmission modes, both based on serial communication: Modbus ASCII (American Standard Code for Information Interchange) and Modbus RTU (Remote Terminal Unit).

Modbus ASCII transmits data as ASCII strings by converting each byte of data into a two-character ASCII code. Termination characters are appended to delimit the data, which increases the amount of data and increases the transmission time compared to Modbus RTU. On the other hand, the transmitted data is easier to parse. Modbus RTU, in contrast, transmits the data in binary format without conversion. To delimit the data in Modbus RTU, a non-communication interval of at least 3.5 characters is required. The amount of data and the transmission time are reduced compared to Modbus ASCII, but parsing the transmitted data requires using a timer.

Modbus communication uses a single-master/multi-slave method in which the slave responds to requests from the master. There is 1 master and 1 to 247 slaves in a Modbus network. Each slave must have a unique address (1 to 247) in the network. Since it is not possible to communicate with different data formats on the same network, either Modbus ASCII or Modbus RTU must be used exclusively on a single network.



At the physical layer, Modbus ASCII/RTU often uses the RS-485 standard. Connecting the RL78 microcontroller with the Renesas RS-485 transceiver via UART makes it easy to implement communication by slave devices over Modbus ASCII/RTU. The RL78 microcontroller is particularly ideal for simple functions such as low-power operation, I/O control, and temperature and humidity measurement. Early-stage development can be done using our sample programs.

Summary

This Application Note describes a sample program that combines an RL78 microcontroller with a Renesas RS-485 transceiver to enable master/slave functionality over Modbus ASCII/RTU.

Evaluation Device

RL78/G16, RAA788152 (RS-485 transceiver), PmodUSBUART

Before applying the sample program covered in this application note to another microcontroller, modify the program according to the specifications for the target microcontroller and conduct an extensive evaluation of the modified program.

Contents

1. Specifications

This application note provides an example of implementing Modbus communication over UART communication. GPIO is used to set the send/receive permissions on the RAA788152 and then Modbus frames are sent and received from the UART0 pins. TAU0 is used for measuring the interval between characters and interval between frames while sending and receiving.

Table 1-1 Peripheral Functions to be Used and their Purpose

Peripheral Function	**Purpose**	
PORT	P13	Control send/receive permissions on the RAA788152
SAU0	UART0	Send/receive Modbus frames
SAU1	UART2	Output Modbus frame send/receive logs and error logs
TAU0	Channel 0	Measure the interval between characters in the Modbus frame and the interval between frames
	Channel 1	In master mode, used to time out when waiting for a response from a slave

2. Confirmed System Requirements

The sample code in this application note has been confirmed to work under the following system requirements.

Table 2-1 System Requirements

Item	Specification
Microcontroller to be used	RL78/G16(R5F121BCAFP)
Board used	• RL78/G16 Fast Prototyping Board (RTK5RLG160C00000BJ) • RS-485 Transceivers Evaluation Board (RTKA788152DE0000BU)
Operating frequency	• High-speed on-chip oscillator clock: 16 MHz • CPU/peripheral hardware clock: 16 MHz
Operating voltage	5 V
Integrated development environment (CS+)	Made by Renesas Electronics Corporation CS+ for CC V8.10.00
C compiler (CS+)	Made by Renesas Electronics Corporation CC-RL V1.12.01
Integrated development environment (e2 studio)	Made by Renesas Electronics Corporation e2 studio V23.7.0
C compiler (e2 studio)	Made by Renesas Electronics Corporation CC-RL V1.12.01
Example peripheral device configuration	For details, see 4. Description of the Hardware

Note When using COM port debugging on the RL78/G16 Fast Prototyping Board, in the [Connection with Target Board] of Debug hardware's COM port (RL78) in the integrated development environment, please select the COM port number for assignment to the RL78/G16 Fast Prototyping Board from the pull-down list for COM port.

3. Related Application Notes

The application notes related to this application note are listed below. Please refer to them as well.

Application Note: RL78/G16 Serial Array Unit (UART Communication) (R01AN6903)

Application Note: RL78/G16 Timer Array Unit (Interval timer) (R01AN7023)

4. Description of the Hardware

4.1 Hardware Configuration Example

This section describes an example hardware configuration that assumes that you are using the RL78/G22 Fast Prototyping Board (RTK7RLG160C00000BJ) and the 5V Half-Duplex RS-485 Transceivers Evaluation Board (RTKA788152DE0000BU) as the RS-485 transceiver. You can output the logs by using PmodUSBUART separately. For the log specifications, see 5.13 Log Specifications

4.1.1 RL78 – PC (GUI) Environment

Figure 4-1 shows the Example Configuration for the RL78 – PC (GUI) Environment.

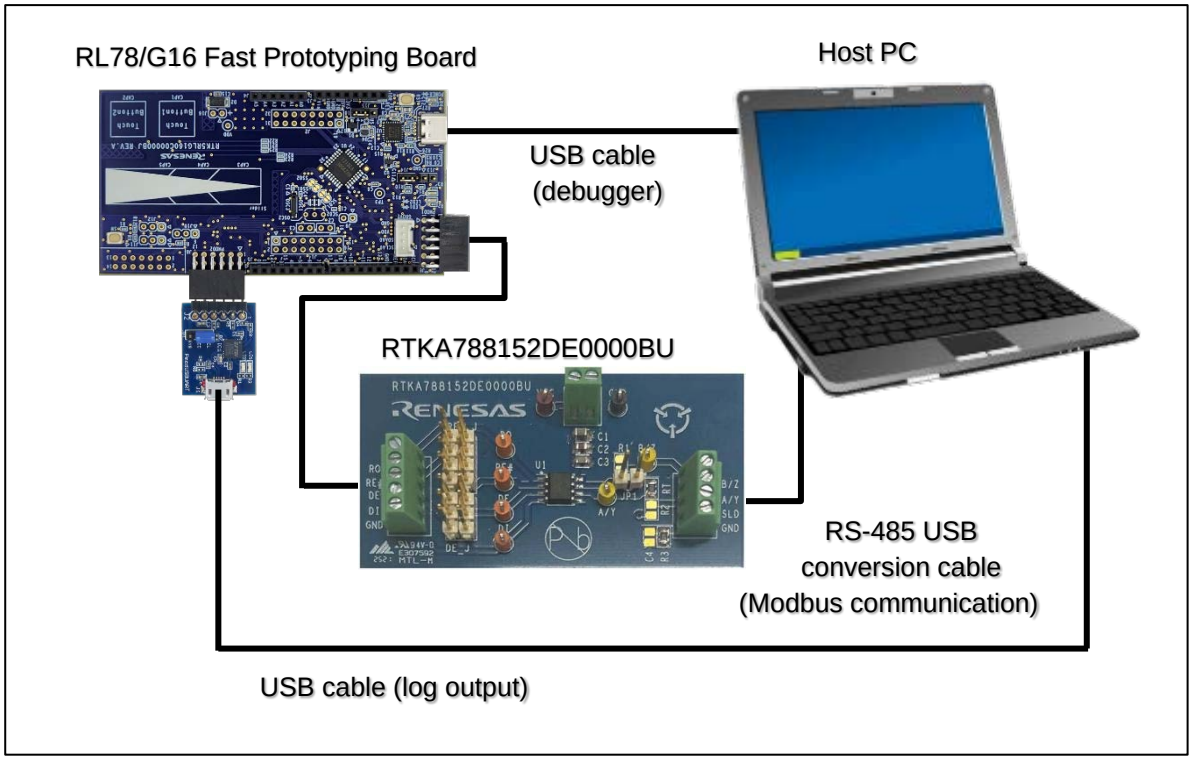


Figure 4-1 Example Configuration for the RL78 – PC (GUI) Environment

4.1.2 RL78 – RL78 Environment

Figure 4-2 shows the Example Configuration for the RL78 – RL78 Environment.

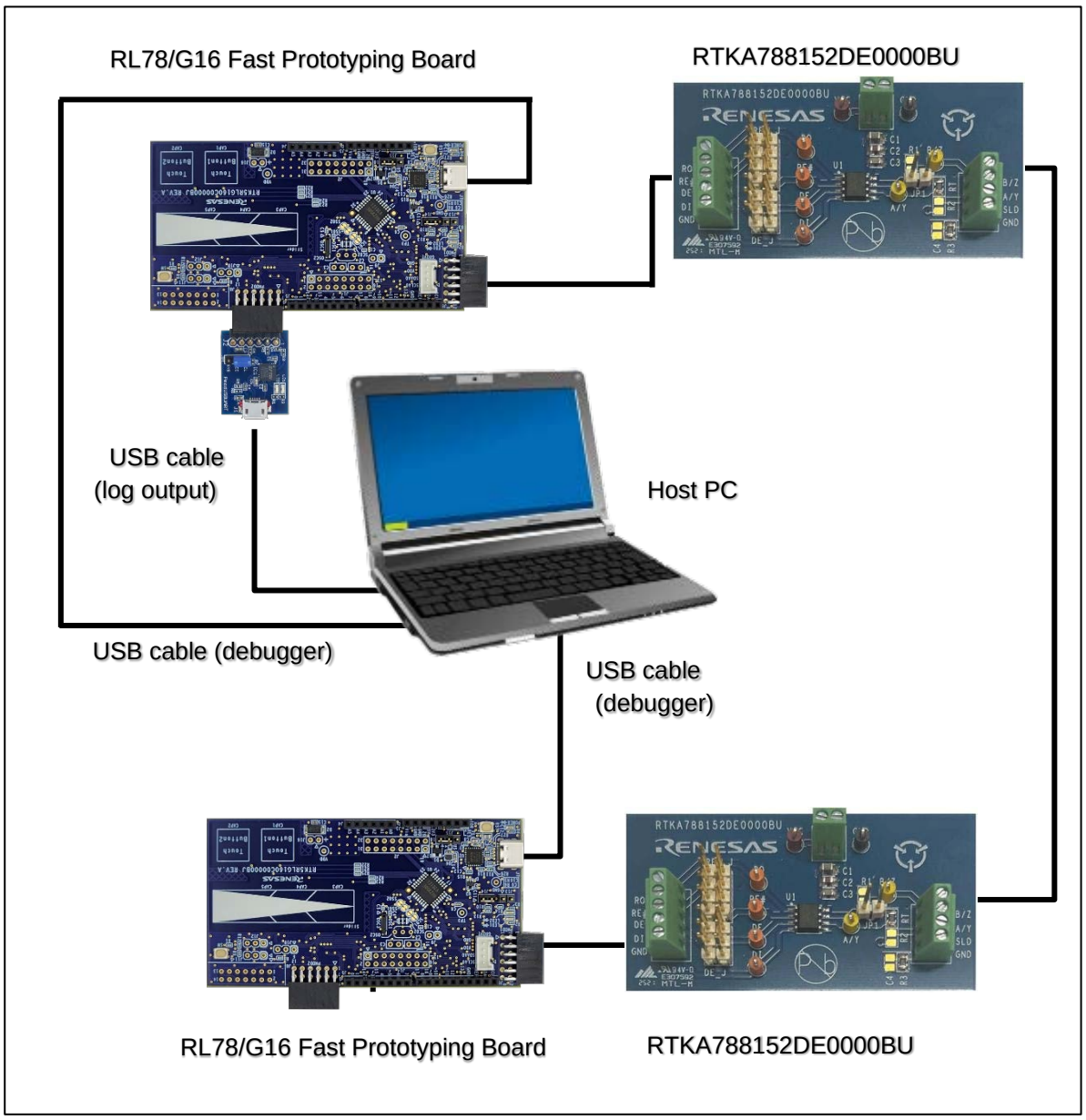


Figure 4-2 Example Configuration for the RL78 – RL78 Environment

4.1.3 RL78 – User Modbus Device Environment

Figure 4-3 shows the Example Configuration for the RL78 – User Modbus Device Environment.

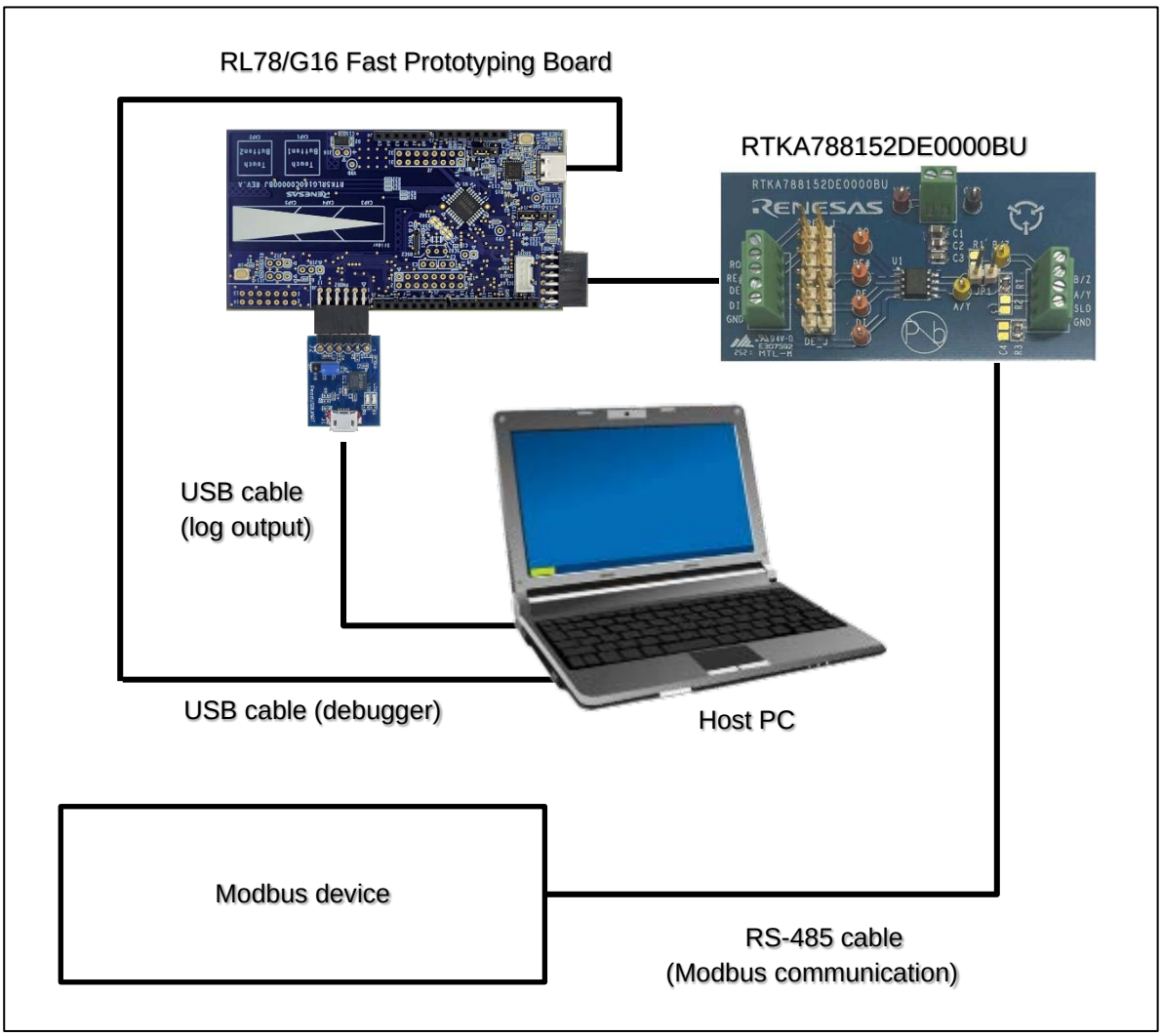


Figure 4-3 Example Configuration for the RL78 – User Modbus Device Environment

4.2 Table of Pins Connecting the RL78/G16 Fast Prototyping Board and the RTKA788152DE0000BU

Table 4-1 shows the Pins Connecting the RL78/G16 Fast Prototyping Board and the RTKA788152DE000BU.

Table 4-1 Pins Connecting the RL78/G16 Fast Prototyping Board and the RTKA788152DE0000BU

RL78/G16 Fast Prototyping Board			RTKA788152DE0000BU		
Connector	**Pin Number**	**Pin Name**	**Connector**	**Pin Number**	**Pin Name**
Pmod™1	1	P13	IN	2, 3	RE#, DE
	2	P03/TxD0		4	DI
	3	P04/RxD0		1	RO
	5	GND	POWER	2	GND
	6	TARGET_VCC		1	VCC

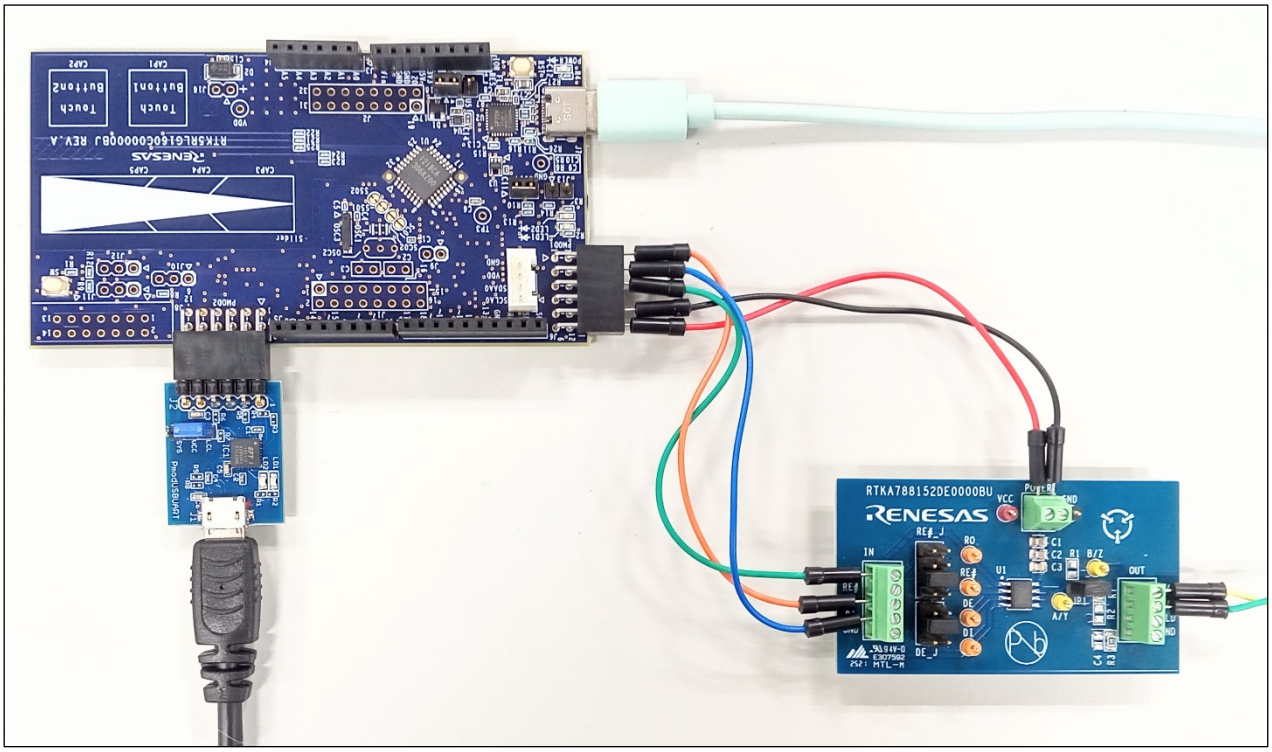


Figure 4-4 The photo of connection between RL78/G16 Fast Prototyping Board and RTKA788152DE0000BU

4.3 RTKA788152DE0000BU Jumper Pin Settings

To receiver enable (RE#)/driver enable (DE) of the RTKA788152DE0000BU with a single GPIO, set the 5pin and 6pin of the RE# _J connector of the RTKA788152DE0000BU to short. Also, DE_J connector should be set to short 3pin and 4pin. This setting shorts RE# and DE. Also, to meet the RS-485 termination requirements, set JP1 short for the terminating resistor at the first or final end.

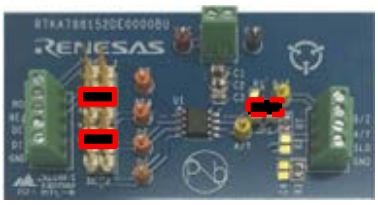


Figure 4-5 RTKA788152DE0000BU jumper pin connection picture

4.4 List of Pins Used

Table 4-2 shows the Pins Used and their Functions.

Table 4-2 Pins Used and their Functions

Pin Name	Input or Output	Purpose
P13	Output	Control send/receive permissions on the RAA788152
P03/TxD0	Output	Send Modbus frame
P04/RxD0	Input	Receive Modbus frame
P21/TxD2	Output	Output Modbus frame send/receive logs and error logs

5. Description of the Software

5.1 Operation Overview

In this sample code, the UART sending and receiving functions are used for serial communication with a connected device using the Modbus protocol. It has master/slave functions, and the communication method supports ASCII and RTU. Note that the master/slave and ASCII/RTU settings can be changed by defining constants.

Accessible registers are kept in RAM and executed according to the function code.

5.1.1 UART Communication Settings for Modbus Communication

Table 5-1 shows the UART Communication Settings for Modbus Communication. The UART communication settings for Modbus communication can be changed by modifying r_modbus_serial.c and r_modbus_time.c.

Table 5-1 UART Communication Settings for Modbus Communication

Baud Rate	Data Length	Parity	Stop Bit	Flow Control
19200 bps	ASCII (7-bit), RTU (8-bit)	Even	1 bit	None

5.1.2 UART Communication Settings for Modbus Log Output

Table 5-2 shows the UART Communication Settings for Log Output. The communication settings of the UART for Modbus log output can be changed from code generation.

Table 5-2 UART Communication Settings for Log Output

Baud Rate	Data Length	Parity	Stop Bit	Flow Control
19200 bps	8-bit	Even	1 bit	None

5.1.3 Supported Function Codes

Table 5-3 shows Supported Function Codes.

Table 5-3 Supported Function Codes

Code	Name	Function
01(0x01)	READ COILS	Read the discrete output ON/OFF state.
02(0x02)	READ DISCRETE INPUTS	Read the discrete input ON/OFF state.
03(0x03)	READ HOLDING REGISTERS	Read the contents of the holding register.
04(0x04)	READ INPUT REGISTER	Read the contents of the input register.
05(0x05)	WRITE SINGLE COIL	Write the ON/OFF state to the discrete output.
06(0x06)	WRITE SINGLE REGISTER	Write the contents to the holding register.
15(0x0F)	WRITE MULTIPLE COILS	Write the ON/OFF state to multiple consecutive discrete output coils.
16(0x10)	WRITE MULTIPLE REGISTERS	Write the contents to multiple consecutive holding registers.
23(0x17)	READ/WRITE MULTIPLE REGISTERS	Write the contents to multiple consecutive holding registers and read the contents of multiple consecutive holding registers.

5.1.4 Modbus Register Assignments

This section describes the Modbus register assignments in slave mode. Although the sample code does not assign functions to each register. This sample code of the Modbus protocol is implemented so that the following values can be Read/Write.

Table 5-4 shows the Modbus Register Assignments (1/2), and Table 5-5 shows the Modbus Register Assignments (2/2).

Table 5-4 Modbus Register Assignments (1/2)

Register Name	Access Unit	Number of Registers	Address Range	Initial Value
Discrete Output	1 bit	16	0x0000	1
			0x0001	1
			0x0002	1
			0x0003	1
			0x0004	1
			0x0005	1
			0x0006	1
			0x0007	1
			0x0008	1
			0x0009	1
			0x000A	1
			0x000B	1
			0x000C	1
			0x000D	1
			0x000E	1
			0x000F	1
Discrete Input	1 bit	16	0x0000	1
			0x0001	1
			0x0002	0
			0x0003	0
			0x0004	1
			0x0005	0
			0x0006	1
			0x0007	0
			0x0008	0
			0x0009	0
			0x000A	1
			0x000B	1
			0x000C	0
			0x000D	1
			0x000E	0
			0x000F	1

Discrete Input works as follows when "g_discrete_input [] = {0xCA, 0x35};".

- 0x0A with 4bit read
- 0xCA with 8-bit read
- 0xCA05 with 12-bit read
- 0xCA35 with 16-bit read

Table 5-5 Modbus Register Assignments (2/2)

Register Name	Access Unit	Number of Registers	Address	Initial Value
Input Register	2 bytes	8	0x0000	0x01FF
			0x0001	0x03FF
			0x0002	0x07FF
			0x0003	0x0FFF
			0x0004	0x1FFF
			0x0005	0x3FFF
			0x0006	0x7FFF
			0x0007	0xFFFF
Holding Register	2 bytes	8	0x0000	0x0000
			0x0001	0x0000
			0x0002	0x0000
			0x0003	0x0000
			0x0004	0x0000
			0x0005	0x0000
			0x0006	0x0000
			0x0007	0x0000

5.2 Setting of Option Byte

Table 5-6 Option Byte Settings shows the option byte settings.

Table 5-6 Option Byte Settings

Address	Setting Value	Contents
000C0H	11111110B	Watchdog time counter operation enabled (Counting started after reset) Stops watchdog timer operation during HALT/STOP mode
000C1H	11110111B	SPOR detection voltage At rising edge TYP. 2.90 V (2.76 V to 3.02 V) At falling edge TYP. 2.84 V (2.70 V to 2.96 V) P125 pin function: RESET input
000C2H	11111001B	High-speed on-chip oscillator clock: 16 MHz
000C3H	10000101B	Enables on-chip debugging

5.3 File Organization

Figure 5-1 shows the File Organization.

```
r01an6961-rl78g16-modbus /src/
 │   modbus.c
 ├──Modbus
 │      r_modbus.c
 │      r_modbus.h
 │      r_modbus_serial.c
 │      r_modbus_serial.h
 │      r_modbus_timer.c
 │      r_modbus_timer.h
 │
 └──smc_gen
    ├──Config_PORT
    │      Config_PORT.c
    │      Config_PORT.h
    │      Config_PORT_user.c
    │
    ├──Config_UART2
    │      Config_UART2.c
    │      Config_UART2.h
    │      Config_UART2_user.c
    │
    ├──Config_WDT
    │      Config_WDT.c
    │      Config_WDT.h
    │      Config_WDT_user.c
    │
    ├──general
    │      r_cg_macrodriver.h
    │      r_cg_port.h
    │      r_cg_sau.h
    │      r_cg_sau_common.c
    │      r_cg_sau_common.h
    │      r_cg_systeminit.c
    │      r_cg_userdefine.h
    │      r_cg_wdt.h
    │      r_smc_entry.h
    │
    ├──r_bsp
    │      ...
    │
    ├──r_config
    │      r_bsp_config.h
    │      r_bsp_config.inc
    │
    └──r_pincfg
           Pin.h
```

Figure 5-1 File Organization

5.4 Slave Mode (ASCII)

When the sample code runs in slave mode (ASCII), the main process initializes the peripheral functions and waits for the Modbus receive flag to be set. The Modbus receive flag is set when a Modbus ASCII frame is received from the UART0 interrupt handling. After the Modbus receive flag is set, the program checks the values of the slave address, function code, and checksum, and calls the callback function if they are all in compliance. The program checks TAU00 for a timeout of the interval between characters in the Modbus frame. If the TAU00 interrupt occurs between the time of the UART0 receive interrupt and completion of the Modbus packet, the program considers this an interval between characters error and clears the receive count and receive buffer.

The timeout period for an interval between characters error in ASCII is not specified by the standard. For some applications, an interval greater than 1 second means that an error has occurred, but other applications may require a longer timeout period. In the sample code the period is set to 1 second.

Finally, the Modbus communication frame logs and error logs are output to UART2.

5.4.1 Main Processing

Figure 5-2 shows the Main Processing (Slave Mode (ASCII)) flow chart.

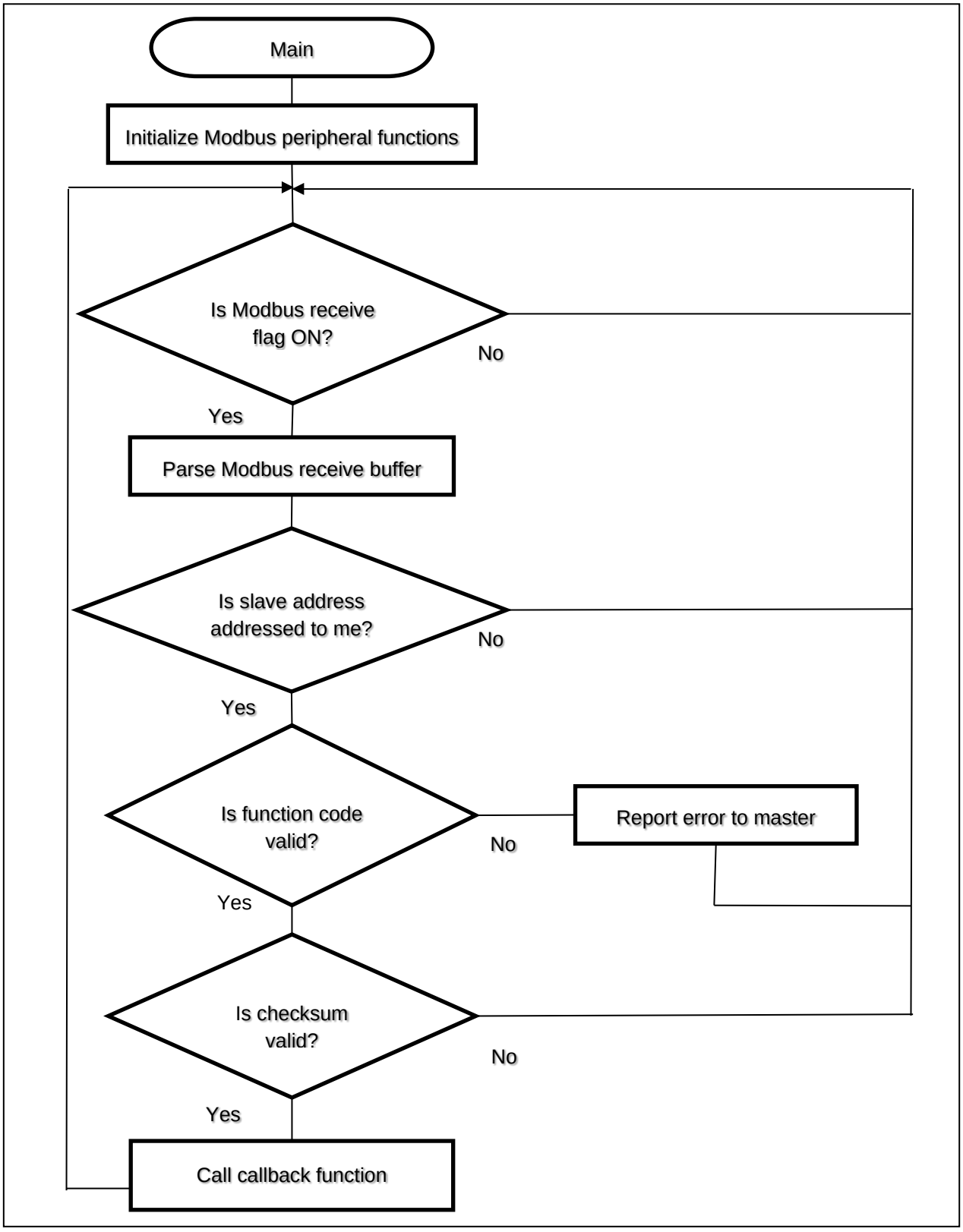


Figure 5-2 Main Processing (Slave Mode (ASCII))

5.4.2 Serial Receive Interrupt Handling

Figure 5-3 shows the Serial Receive Interrupt Handling (Slave Mode (ASCII)) flow chart.

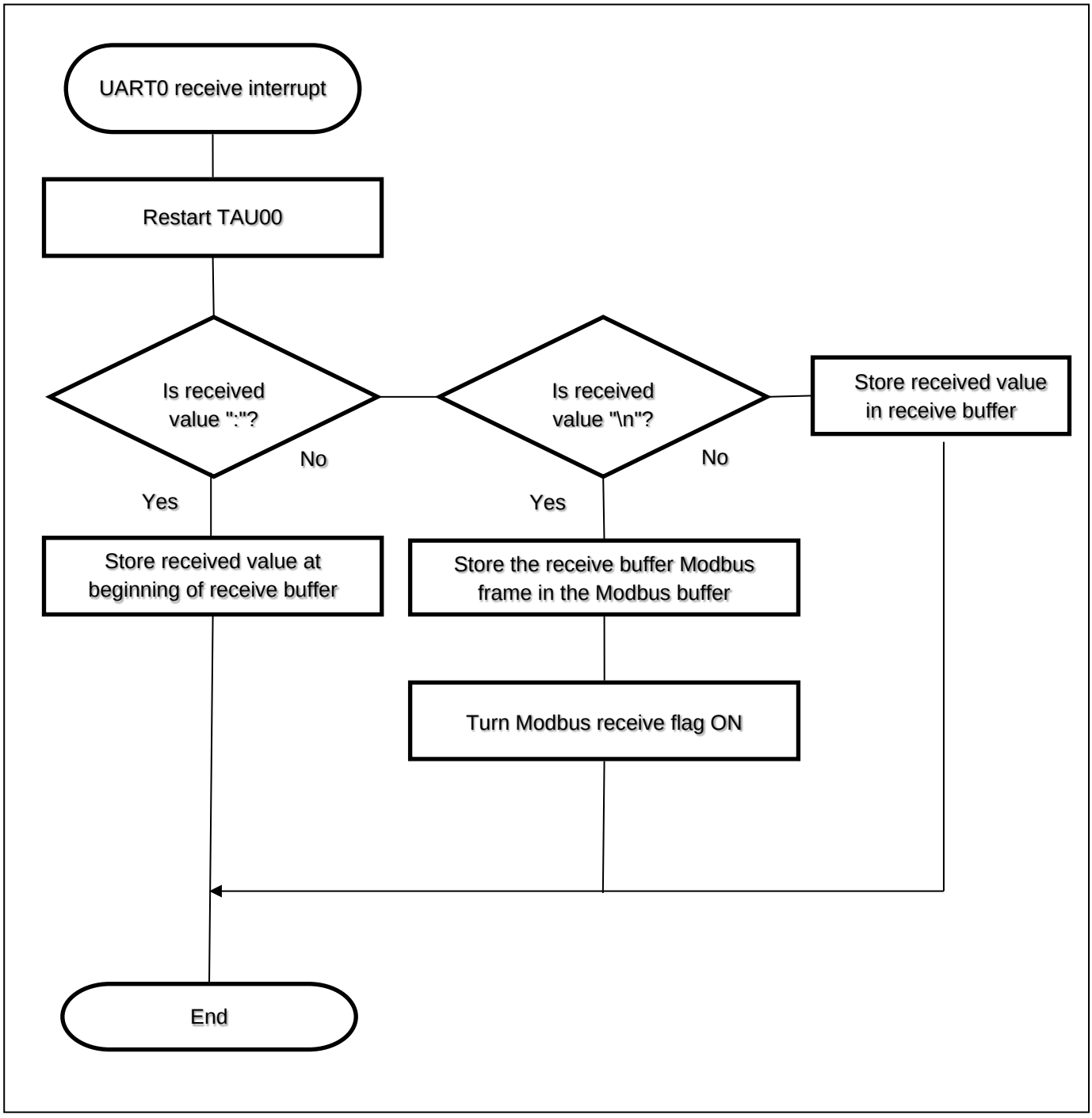


Figure 5-3 Serial Receive Interrupt Handling (Slave Mode (ASCII))

5.4.3 Interval Between Characters Error Interrupt

Figure 5-4 shows the Interval Between Characters Error Interrupt Handling (Slave Mode (ASCII)) flow chart.

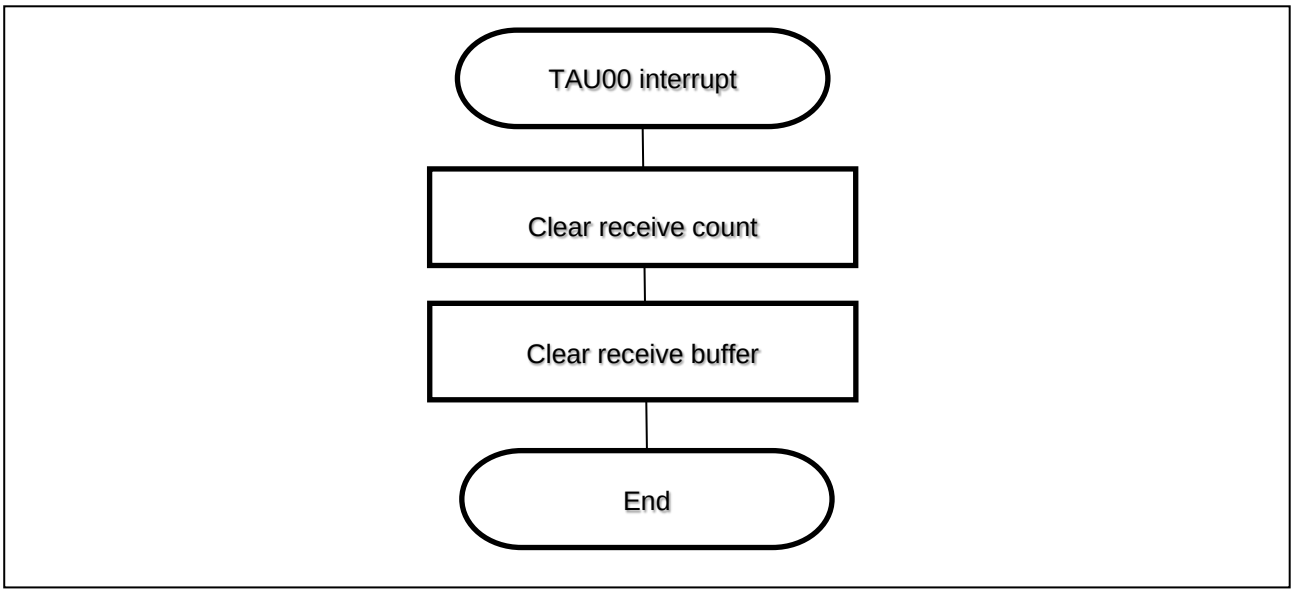


Figure 5-4 Interval Between Characters Error Interrupt Handling (Slave Mode (ASCII))

5.5 Slave Mode (RTU)

When the sample code runs in slave mode (RTU), the main process initializes the peripheral functions and waits for the Modbus receive flag to be set. The program checks TAU00 for a timeout in the interval between characters in the Modbus frame or the interval between frames. If the TAU00 interrupt occurs between the time of the UART0 interrupt and completion of the Modbus packet, the program sets the interval between characters error flag, changes TAU00 to the period of the interval between frames, and then starts. If the UART0 receive interrupt handling occurs again with the interval between characters error flag set, the program considers this an interval between characters error and clears the receive count and receive buffer. If TAU00 interrupt handling occurs with this flag set but no UART0 receive interrupt is generated, it is considered normal termination, the interval between characters error flag is cleared, and the Modbus receive flag is set. After the Modbus receive flag is set, the program checks the values of the slave address, function code, and checksum, and calls the callback function if they are all in compliance.

The timeout period for an interval between characters error in the RTU is 1.5 characters, and the interval between frames is 3.5 characters. Depending on the communication settings in the sample code, the respective periods are set as follows.

Each interval = number of characters × number of bits per character × communication time per bit × accuracy of HOCO (1%)

Interval between characters: 1.5 × 11 × 1/19200 × 1.01 ≈ 868 [us]

Interval between frames: 3.5 × 11 × 1/19200 × 1.01 ≈ 2026 [us]

In this sample code, a single timer (TAU00) is used to make two determinations: the interval between characters and the interval between frames. After the timer interrupt occurs for the interval between characters, TAU00 is changed to 1158 [us] (2026 − 868) and used to determine the interval between frames.

Finally, the Modbus communication frame logs and error logs are output to UART2.

5.5.1 Main Processing

Figure 5-5 shows the Main Processing (Slave Mode (RTU)).

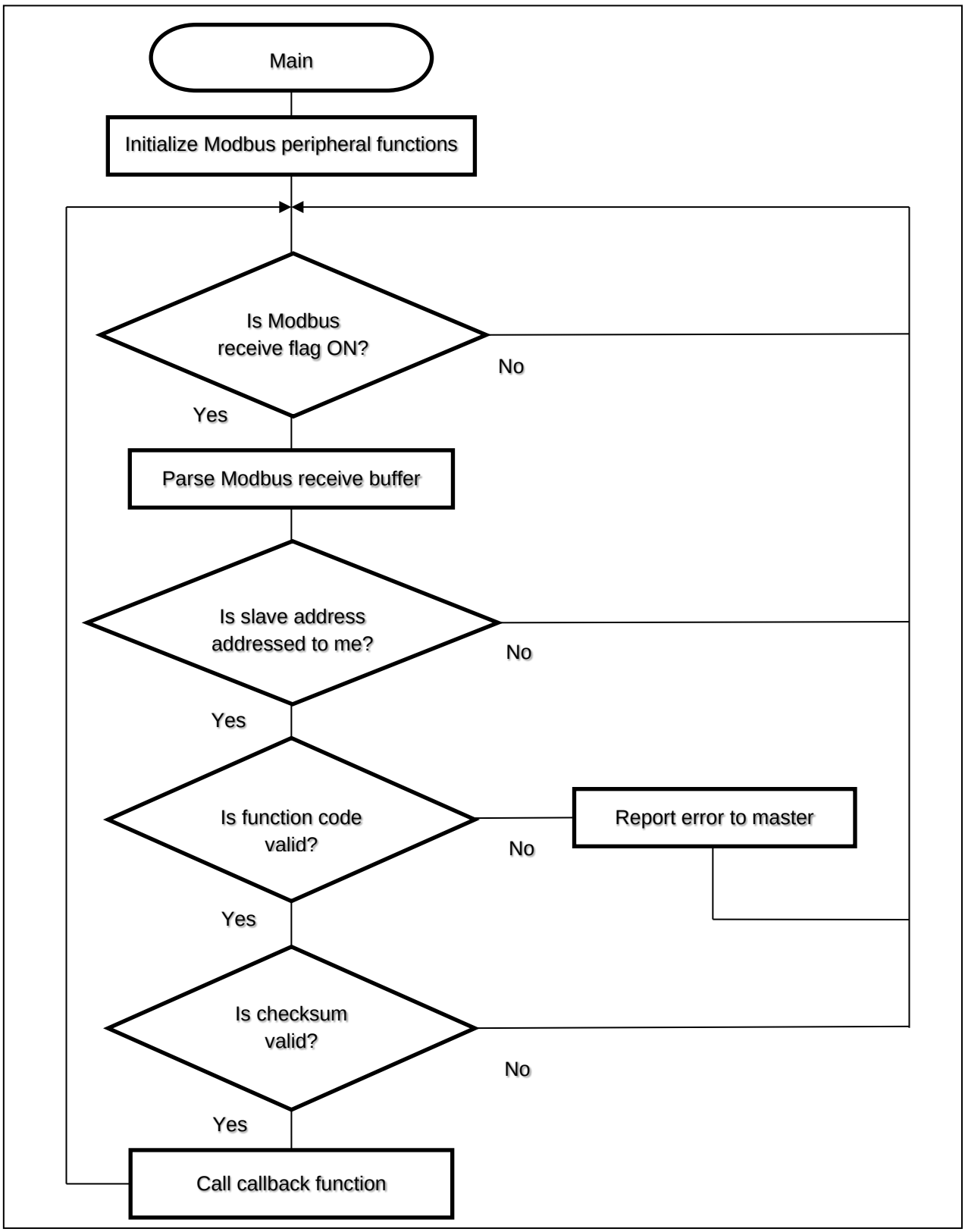


Figure 5-5 Main Processing (Slave Mode (RTU))

5.5.2 Serial Receive Interrupt Handling

Figure 5-6 shows the Serial Receive Interrupt Handling (Slave Mode (RTU)) flow chart.

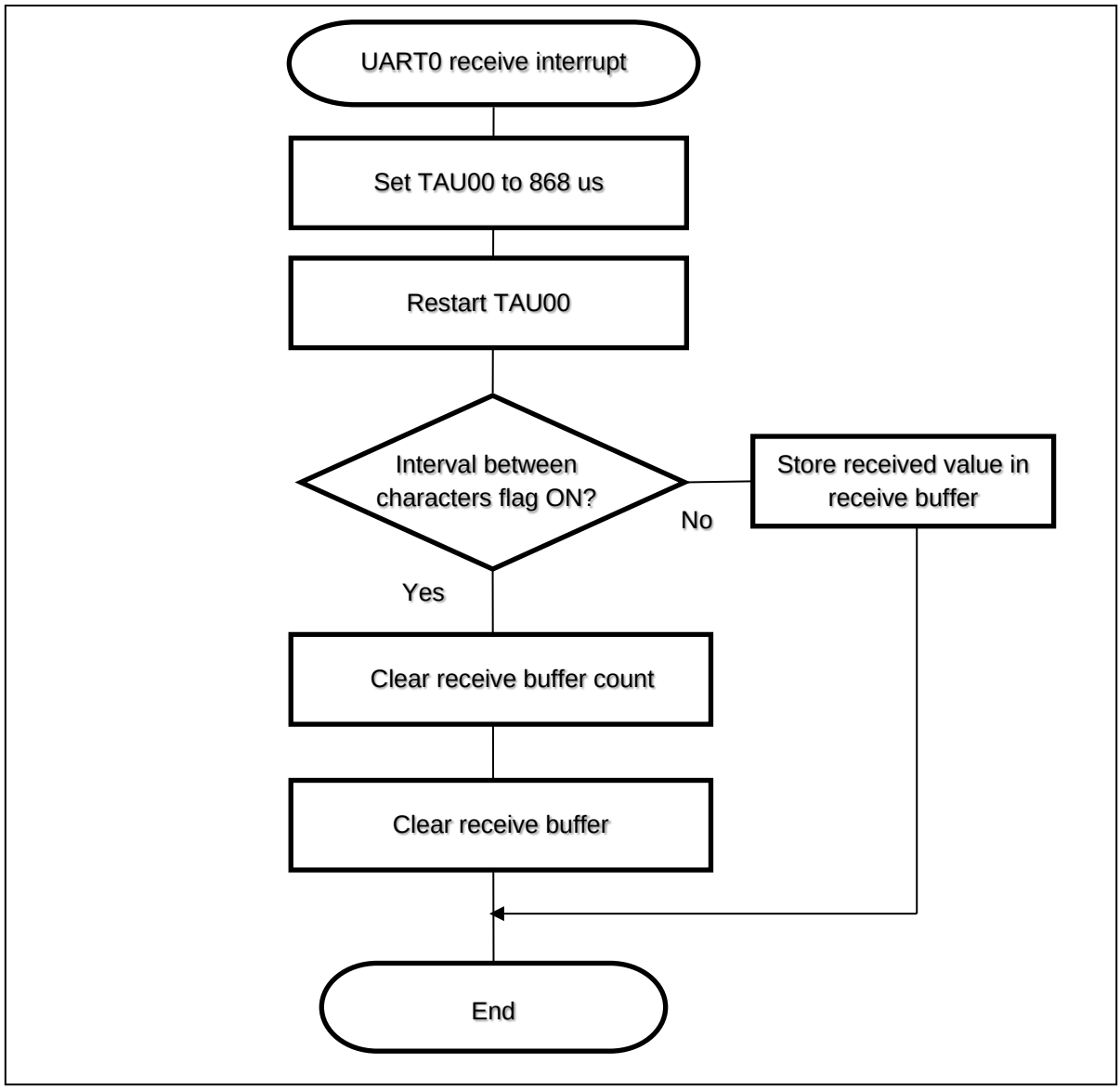


Figure 5-6 Serial Receive Interrupt Handling (Slave Mode (RTU))

5.5.3 Interval Between Characters Interrupt Handling

Figure 5-7 shows the Interval Between Characters Interrupt Handling (Slave Mode (RTU)) flow chart.

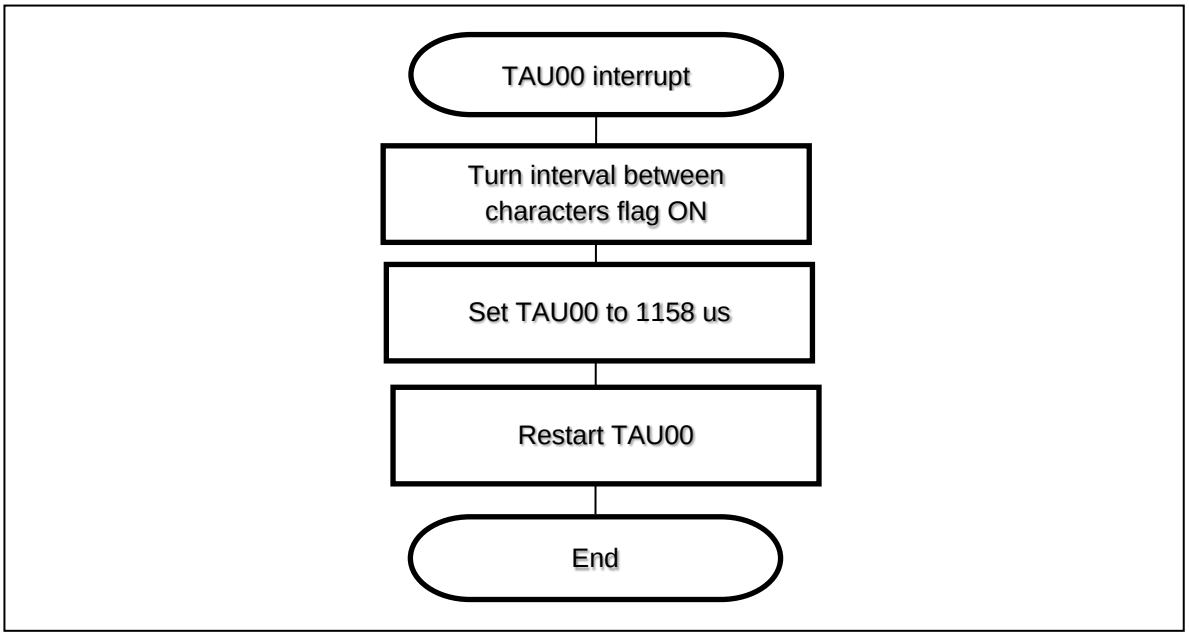


Figure 5-7 Interval Between Characters Interrupt Handling (Slave Mode (RTU))

5.5.4 Modbus Received Interrupt Handling

Figure 5-8 shows the Modbus Received Interrupt Handling (Slave Mode (RTU)) flow chart.

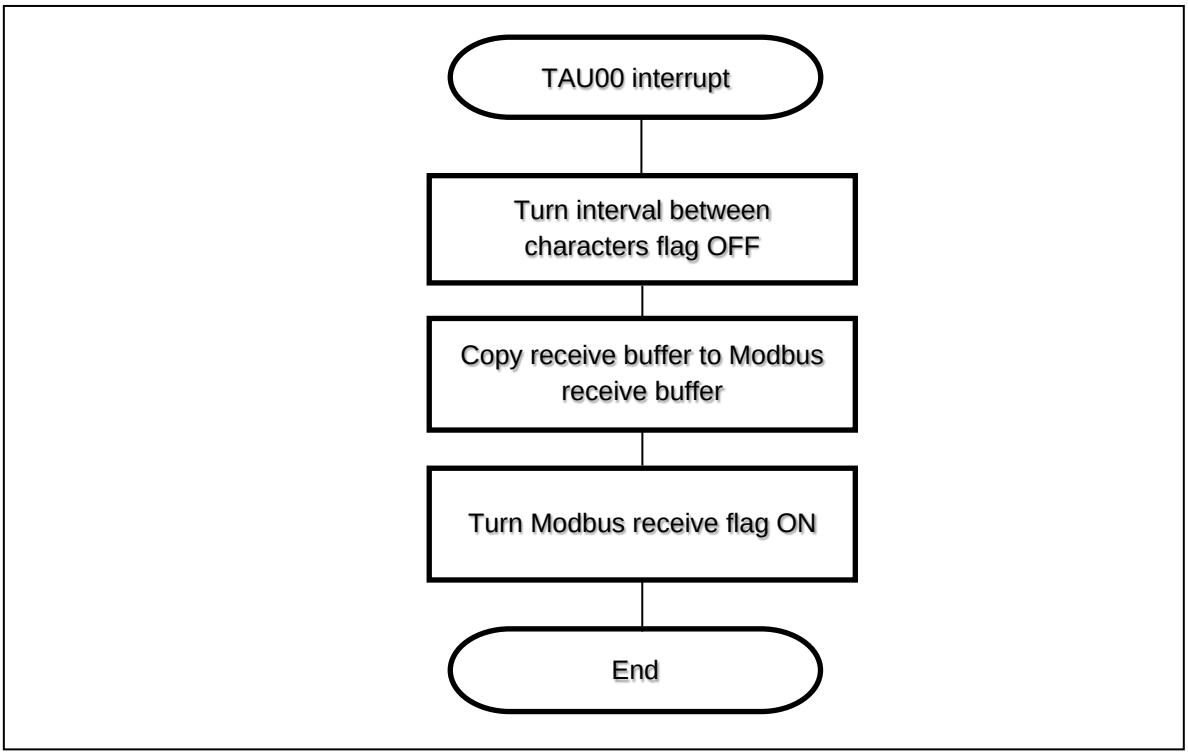


Figure 5-8 Modbus Received Interrupt Handling (Slave Mode (RTU))

5.6 Master Mode (ASCII)

When the sample code runs in master mode (ASCII), it sends Read Coils to SLAVE ID = 0x01 every second. When it receives the response from the slave, the program calls the callback function. The program checks TAU00 for a timeout of the interval between characters in the Modbus frame. If the TAU00 interrupt occurs between the time of the receive interrupt and completion of the Modbus packet, the program considers this an interval between characters error and clears the receive count and receive buffer.

The timeout period for an interval between characters error in ASCII is not specified by the standard. For some applications, an interval greater than 1 second means that an error has occurred, but other applications may require a longer timeout period. In the sample code the period is set to 1 second. The Read Coils transmission interval (1 second) uses TAU01.

Finally, the Modbus communication frame logs and error logs are output to UART2.

5.6.1 Main Processing

Figure 5-9 shows the Main Processing (Master Mode (ASCII)) flow chart.

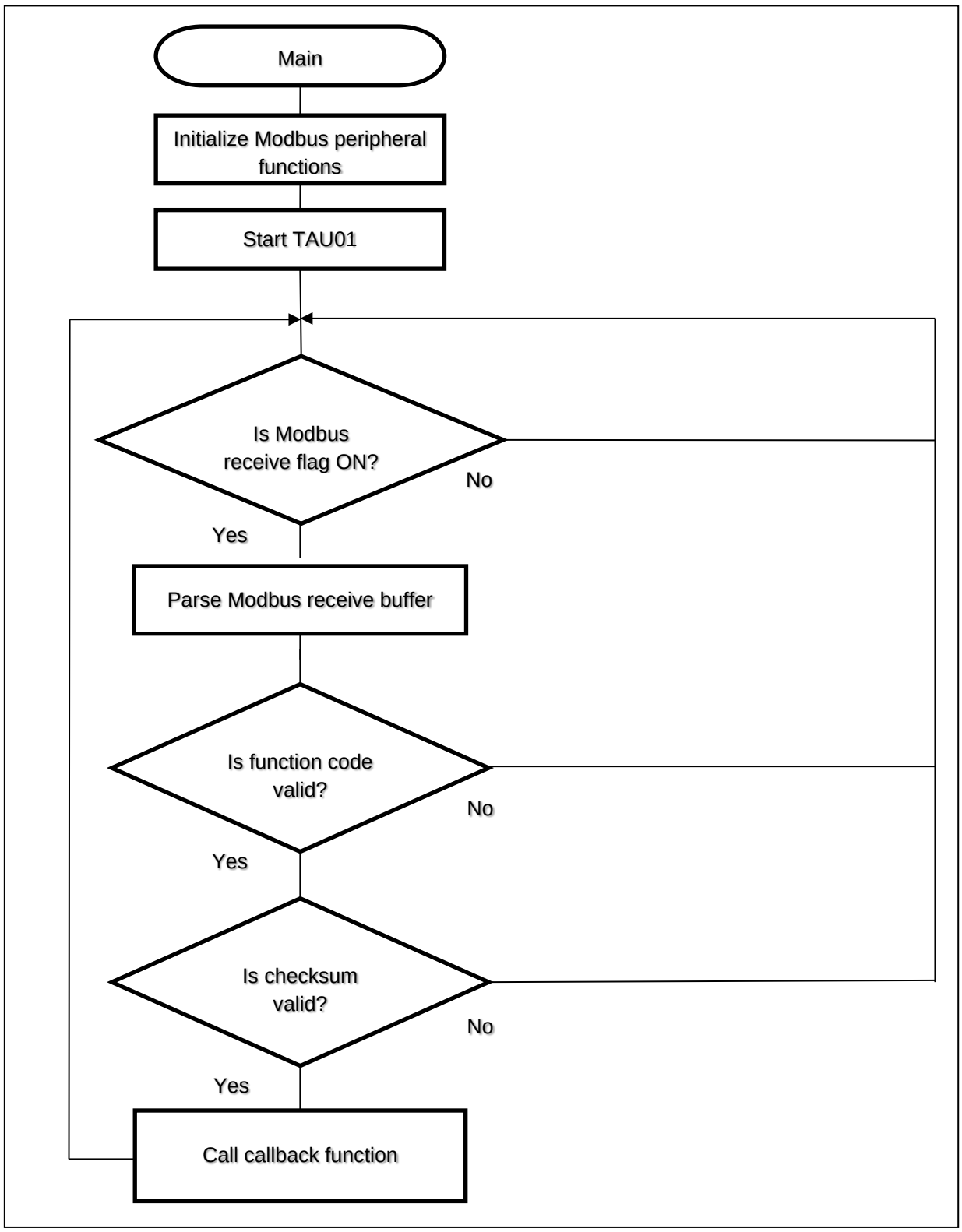


Figure 5-9 Main Processing (Master Mode (ASCII))

5.6.2 Read Coil Send Interrupt Handling

Figure 5-10 shows the Read Coil Send Interrupt Handling (Master Mode (ASCII)) flow chart.

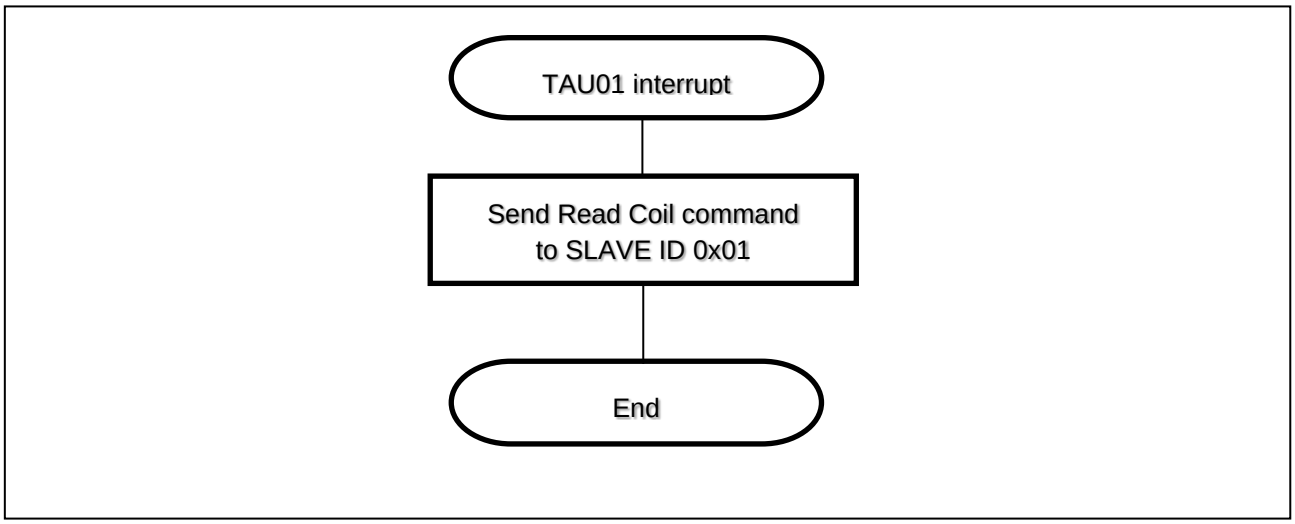


Figure 5-10 Read Coil Send Interrupt Handling (Master Mode (ASCII))

5.6.3 Serial Receive Interrupt Handling

Figure 5-11 shows the Serial Receive Interrupt Handling (Master Mode (ASCII)) flow chart.

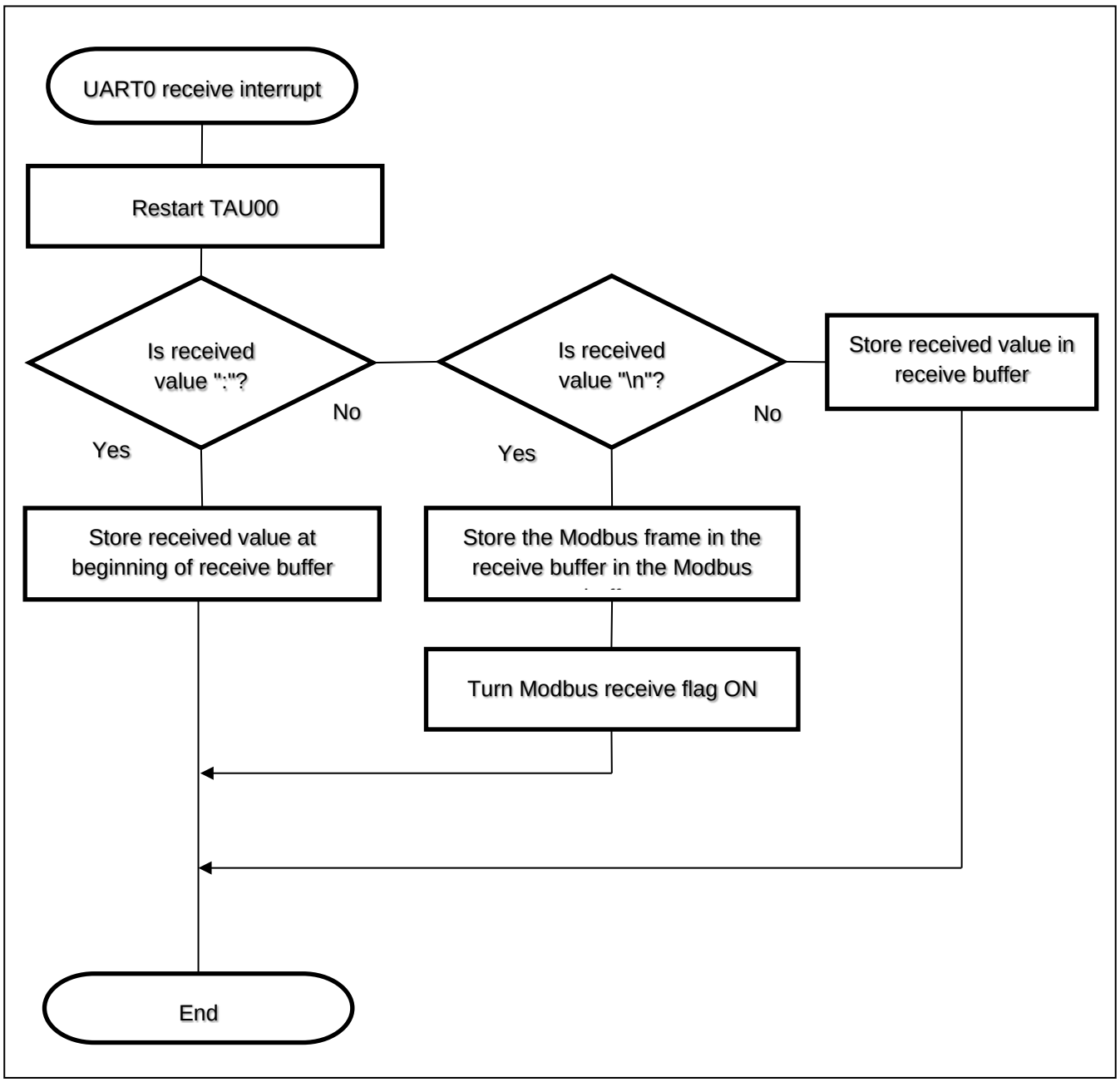


Figure 5-11 Serial Receive Interrupt Handling (Master Mode (ASCII))

5.6.4 Interval Between Characters Error Interrupt Handling

Figure 5-12 shows the Interval Between Characters Error Handling (Master Mode (ASCII)) flow chart.

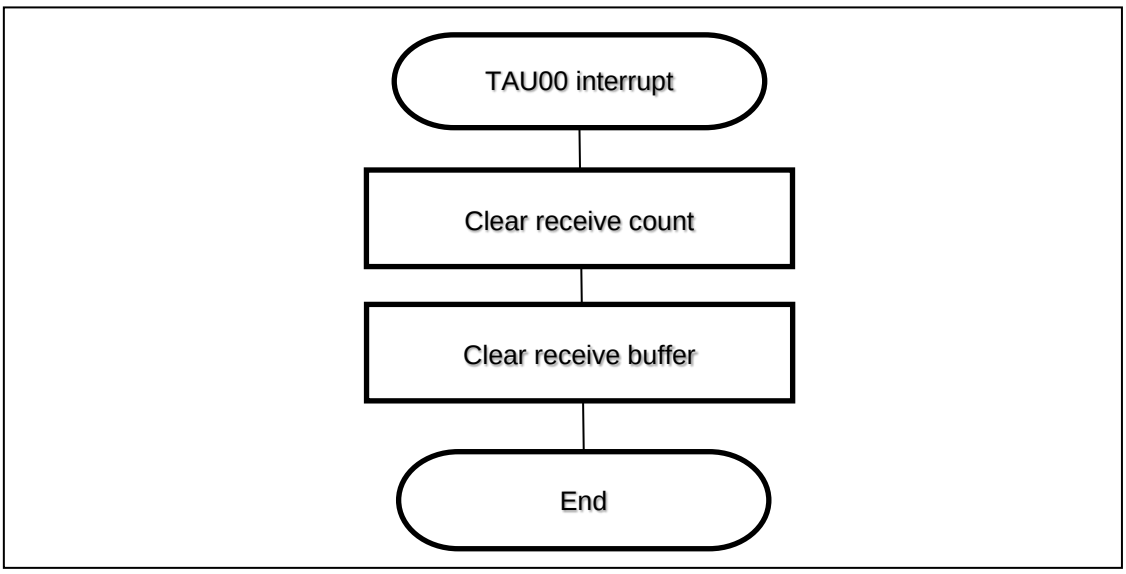


Figure 5-12 Interval Between Characters Error Handling (Master Mode (ASCII))

5.7 Master Mode (RTU)

When the sample code runs in master mode (RTU), it sends Read Coils to SLAVE ID = 0x01 every second. When it receives the command from the slave it calls the callback function. The program checks TAU00 for a timeout in the interval between characters in the Modbus frame or the interval between frames. If the TAU00 interrupt occurs between the time of the UART0 receive interrupt and completion of the Modbus packet, the program sets the interval between characters error flag, changes TAU00 to the period of the interval between frames, and then starts. If the UART0 receive interrupt occurs again with the interval between characters error flag set, the program considers this an interval between characters error and clears the receive count and receive buffer. If TAU00 interrupt handling occurs with this flag set but no UART0 receive interrupt is generated, it is considered normal termination, the interval between characters error flag is cleared, and the Modbus receive flag is set. After the Modbus receive flag is set, the program checks the values of the slave address, function code, and checksum, and calls the callback function if they are all in compliance.

The timeout period for an interval between characters error in the RTU is 1.5 characters, and the interval between frames is 3.5 characters. Depending on the communication settings in the sample code, the respective periods are set as follows.

Each interval = number of characters × number of bits per character × communication time per bit × accuracy of HOCO (1%)

Interval between characters: 1.5 × 11 × 1/19200 × 1.01 ≈ 868 [us]

Interval between frames: 3.5 × 11 × 1/19200 × 1.01 ≈ 2026 [us]

In this sample code, a single timer (TAU00) is used to make two determinations: the interval between characters and the interval between frames. After the timer interrupt occurs for the interval between characters, TAU00 is changed to 1158 [us] (2026 − 868) and used to determine the interval between frames. The Read Coils transmission interval (1 second) uses TAU01.

Finally, the Modbus communication frame logs and error logs are output to UART2.

5.7.1 Main Processing

Figure 5-13 shows the Main Processing (Master Mode (RTU)) flow chart.

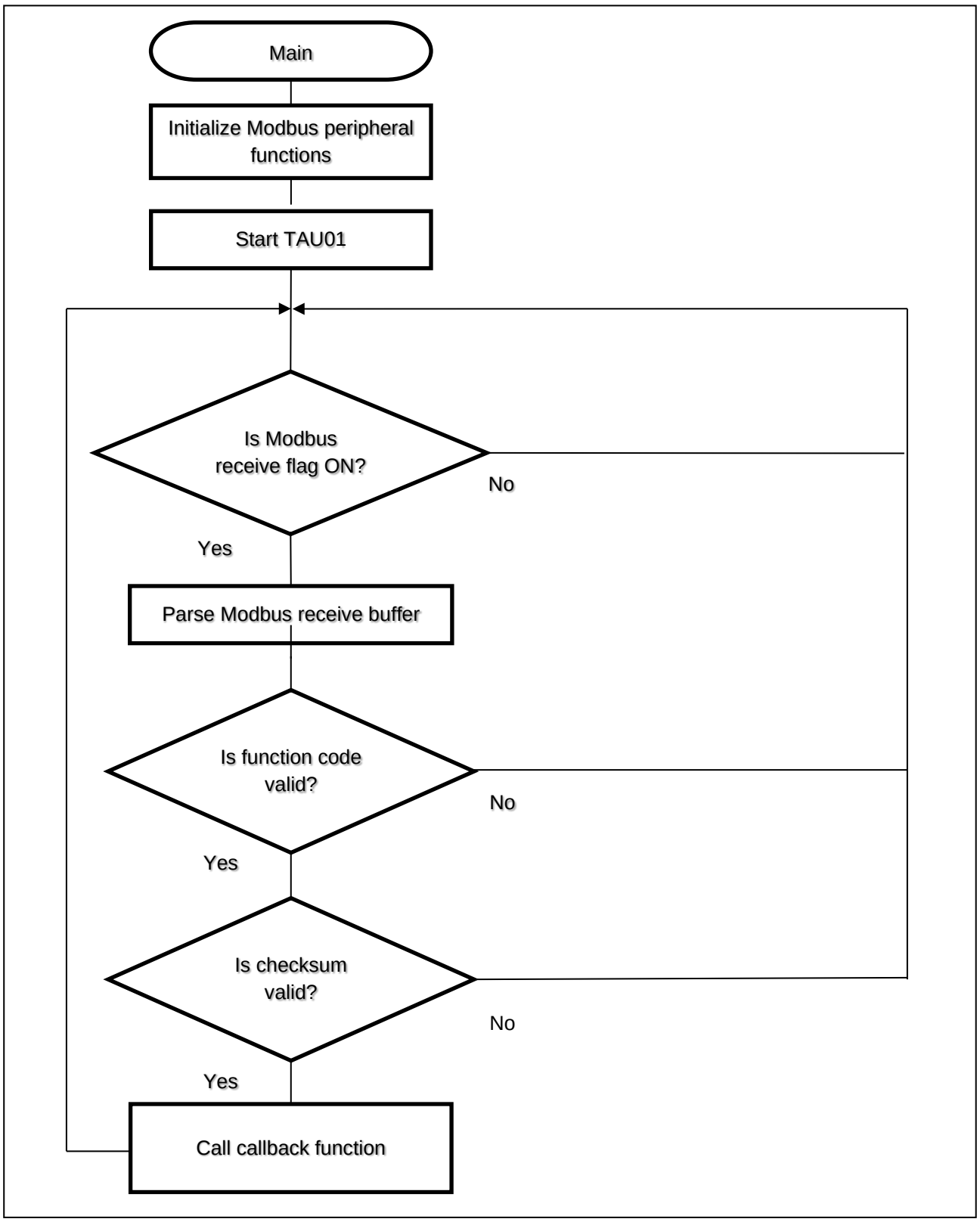


Figure 5-13 Main Processing (Master Mode (RTU))

5.7.2 Read Coil Send Interrupt Handling

Figure 5-14 shows the Read Coil Send Interrupt Handling (Master Mode (RTU)) flow chart.

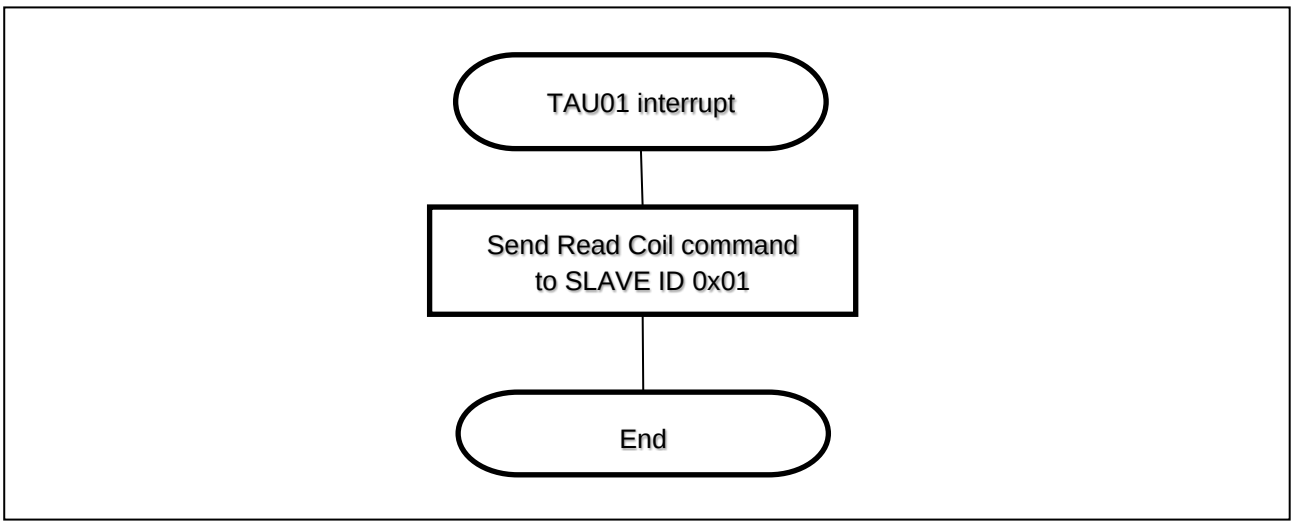


Figure 5-14 Read Coil Send Interrupt Handling (Master Mode (RTU))

5.7.3 Serial Receive Interrupt Handling

Figure 5-15 shows the Serial Receive Interrupt Handling (Master Mode (RTU)) flow chart.

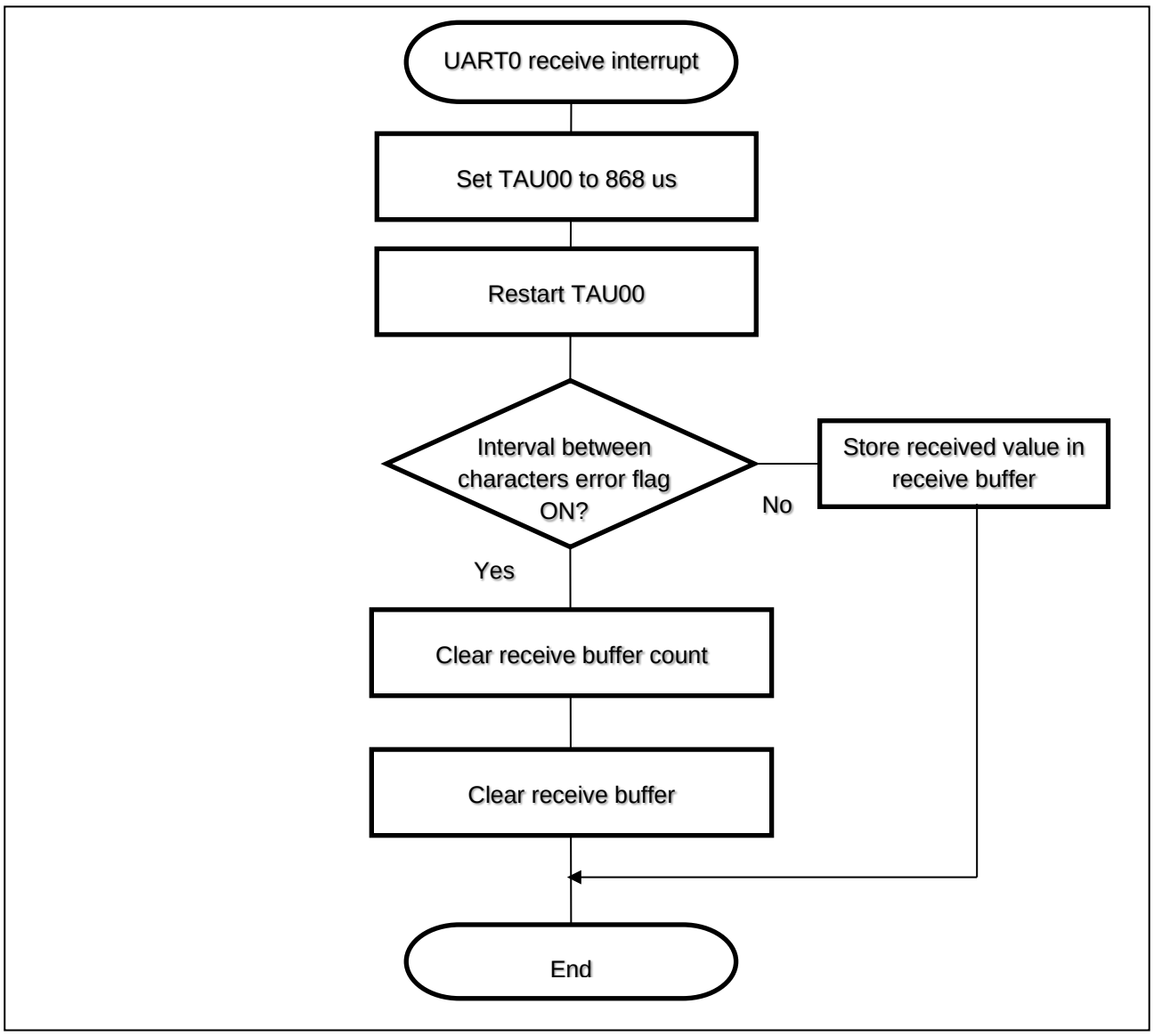


Figure 5-15 Serial Receive Interrupt Handling (Master Mode (RTU))

5.7.4 Interval Between Characters Interrupt Handling

Figure 5-16 shows the Interval Between Characters Interrupt Handling (Master Mode (RTU)) flow chart.

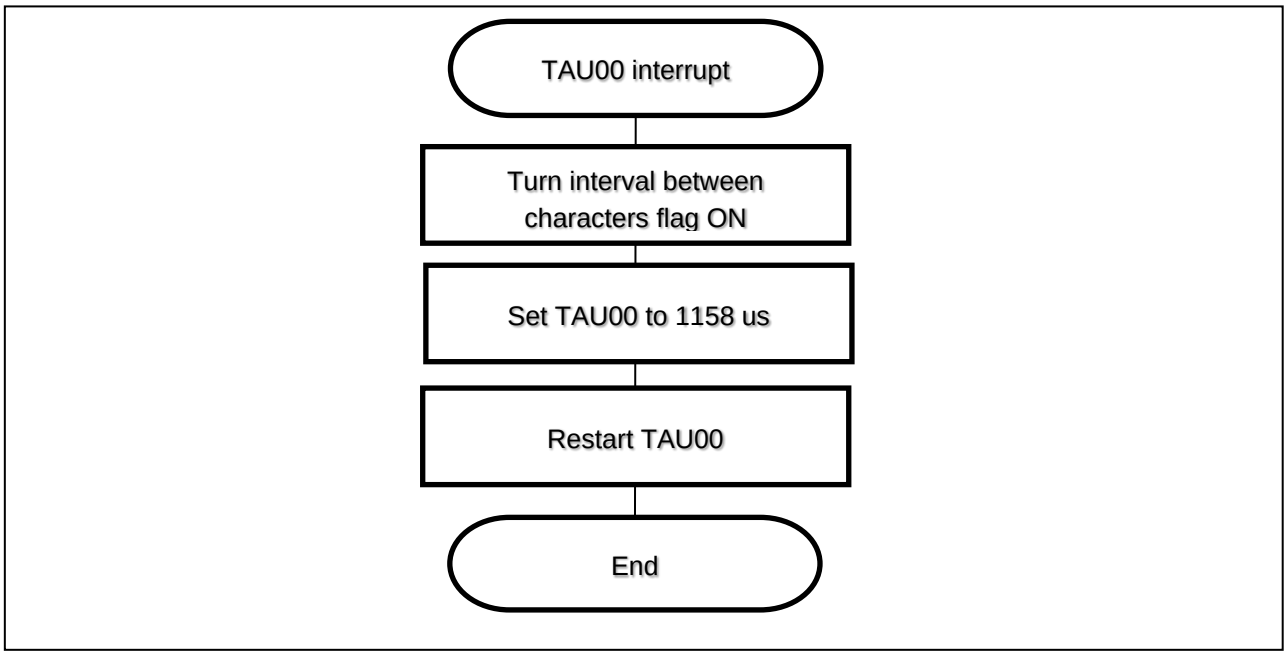


Figure 5-16 Interval Between Characters Interrupt Handling (Master Mode (RTU))

5.7.5 Modbus Received Interrupt Handling

Figure 5-17 shows the Modbus Received Interrupt Handling (Master Mode (RTU)) flow chart.

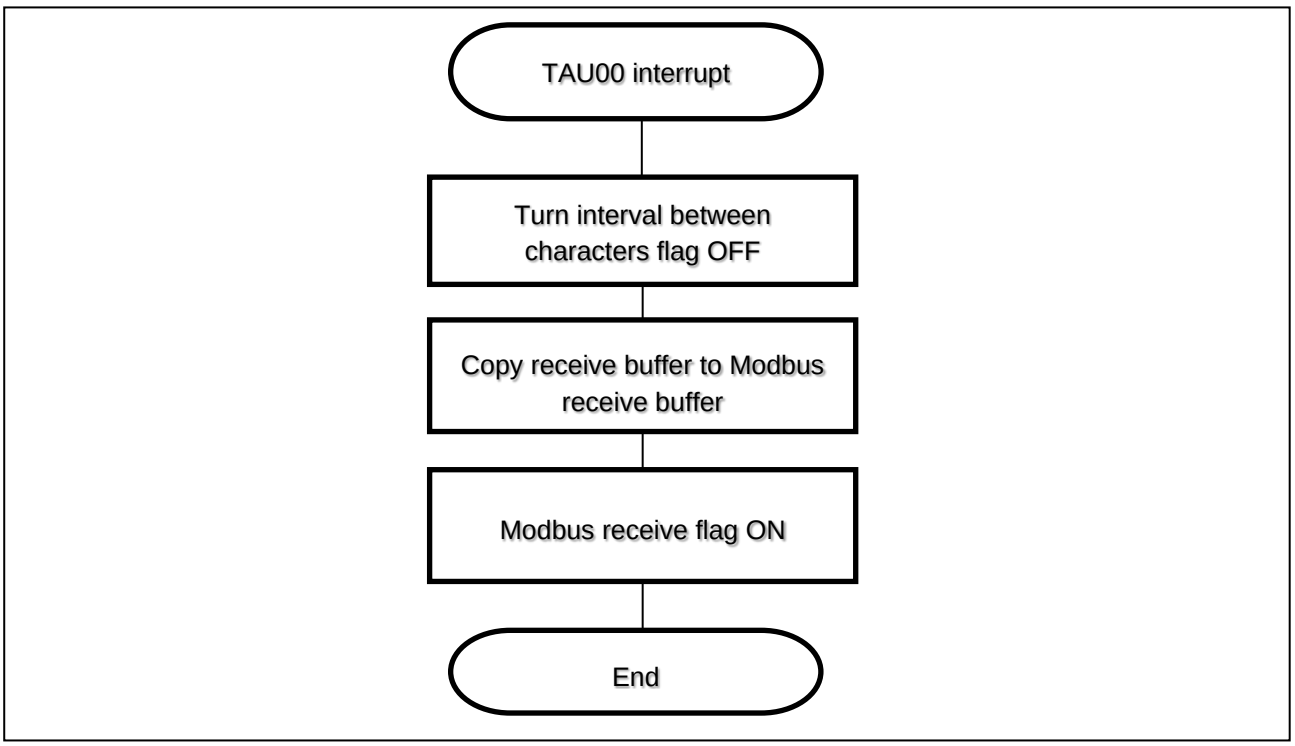


Figure 5-17 Modbus Received Interrupt Handling (Master Mode (RTU))

5.8 List of Constants

5.8.1 Modbus Operation Configuration Constants

Table 5-7 shows the Modbus Operation Configuration Constants.

Table 5-7 Modbus Operation Configuration Constants

Name of Constant	Value	Purpose
MODBUS_MODE	0x01 to 0x04	Specifies the operating mode of the sample code. 0x01: Master Mode (RTU) 0x02: Slave Mode (RTU) 0x03: Master Mode (ASCII) 0x04: Slave Mode (ASCII)
MODBUS_RTU_MASTER_MODE	0x01	Used to specify Master Mode (RTU).
MODBUS_RTU_SLAVE_MODE	0x02	Used to specify Slave Mode (RTU).
MODBUS_ASCII_MASTER_MODE	0x03	Used to specify Master Mode (ASCII).
MODBUS_ASCII_SLAVE_MODE	0x04	Used to specify Slave Mode (ASCII).
MODBUS_SEND_BUFFER_SIZE	1 to 256	Specifies the maximum size of the Modbus send buffer.
MODBUS_RECV_BUFFER_SIZE	1 to 253	Specifies the maximum size of the Modbus receive buffer.

5.8.2 Modbus Status Constants

Table 5-8 shows the Modbus Status Constants.

Table 5-8 Modbus Status Constants

Name of Constant	Value	Purpose
MODBUS_STATUS_NONE	0x00	Modbus status before operation starts
MODBUS_STATUS_WAIT_RECEIVE	0x01	Waiting to receive Modbus frame
MODBUS_STATUS_RECEIVED	0x02	Modbus frame has been received
MODBUS_STATUS_WAIT_FRAME_INTERVAL	0x03	Waiting for the interval between frames (used only in RTU mode)
MODBUS_STATUS_ERROR_CHARACTER_INTERVAL	0x04	An interval between characters error occurred while receiving Modbus frames

5.8.3 Modbus Receive Results Constants

Table 5-9 shows the Modbus Receive Results Constants.

Table 5-9 Modbus Receive Results Constants

Name of Constant	Value	Purpose
MODBUS_RECEIVE_NONE	0x00	No Modbus frames received
MODBUS_ERROR_NONE	0x01	No error after Modbus frame received
MODBUS_ERROR_NULL_POINTER	0x02	NULL was specified for the destination address of the structure for storing the Modbus frame parse results
MODBUS_ERROR_NOT_MYSELF	0x03	The Modbus frame was parsed and it was not addressed to me
MODBUS_ERROR_ILLEGAL_FUNCTION	0x04	The Modbus frame was parsed and the resulting function code was abnormal
MODBUS_ERROR_MISMATCH_CHECKSUM	0x05	The Modbus frame was parsed and the resulting checksum was abnormal
MODBUS_ERROR_CHARACTER_INTERVAL	0x06	An interval between characters error occurred while receiving Modbus frames
MODBUS_ERROR_ILLEGAL_FRAME_ LENGTH	0x07	The received Modbus frame is shorter than the expected frame length
MODBUS_ERROR_RECEIVE_ERROR_ RESPONSE	0x08	Error response received from slave when in master mode

5.9 List of Variables

Table 5-10 shows the Global Variables.

Table 5-10 Global Variables

Type	Name of Variable	Purpose
unsigned char[]	g_modbus_rx_buffer	Modbus frame receive buffer
unsigned char	g_modbus_status	Modbus communication status
unsigned char	g_modbus_frame_size	Size of Modbus receive frame
unsigned short[]	g_holding_register	Holding register
unsigned short[]	g_input_register	Input register
unsigned char[]	g_discrete_output	DO
unsigned char[]	g_discrete_input	DI
unsigned char[]	g_rx_buffer	UART receive buffer

5.10 List of Structures

Table 5-11 shows the Structures.

Table 5-11 Structures

Type	Field	Purpose
st_modbus_receive_frame_t	uint8_t slave_address uint8_t function_code uint16_t read_address uint16_t read_data uint16_t write_address uint16_t write_data uint8_t array_data_len uint16_t array_data[]	Stores the results of parsing the Modbus frame

5.11 List of Functions

5.11.1 API Functions

Table 5-12 shows the API Functions in the Sample Code.

Table 5-12 API Functions in the Sample Code

Name of Function	Overview
R_MODBUS_Main	Modbus communication main processing
R_MODBUS_Init	Initialize peripheral functions used for Modbus communication
R_MODBUS_Close	Close peripheral functions used for Modbus communication
R_MODBUS_Send	Send Modbus frame
R_MODBUS_Receive	Receive Modbus frame
R_MODBUS_Parse	Parse Modbus frame
R_MODBUS_Send_Error	Send Modbus error

5.11.2 Supported Function Codes

Table 5-13 shows the Callback Functions in the Sample Code.

Table 5-13 Callback Functions in the Sample Code

Name of Function	Overview
r_modbus_callback_slave_receiveend	Called after receiving the Modbus frame in slave mode
r_modbus_callback_master_receiveend	Called after receiving the Modbus frame in master mode
r_modbus_callback_error	Called if an error occurs in the Modbus communication/frame parsing process

5.12 Function Specifications

This section describes the specifications for the functions that are used in this sample code.

[Function Name] R_MODBUS_Main

Synopsis	Modbus communication main processing
Header	r_modbus.h
Declaration	void R_MODBUS_Main (void)
Description	Detects when Modbus communication is received and calls the callback function. In master mode, detects an incoming request and sends a Modbus frame to the slave.
Arguments	None
Return value	None
Remarks	None

[Function Name] R_MODBUS_Init

Synopsis	Initialize peripheral functions used for Modbus communication
Header	None
Declaration	void R_MODBUS_Init (void)
Description	Initializes peripheral functions used for Modbus communication.
Arguments	None
Return value	None
Remarks	Be sure to do this before starting Modbus communication.

[Function Name] R_MODBUS_Close

Synopsis	Close peripheral functions used for Modbus communication
Header	None
Declaration	void R_MODBUS_Close (void)
Description	Closes the peripheral functions used for Modbus communication.
Arguments	None
Return value	None
Remarks	Be sure to do this when terminating Modbus communication.

[Function Name] R_MODBUS_Send

Synopsis	Send Modbus frame
Header	None
Declaration	void R_MODBUS_Send (uint8_t slave_address, uint8_t function_code, uint8_t *send_data, uint8_t send_data_len)
Description	Sends the data in send_data as a Modbus frame of length send_data_len.
Arguments	Slave address, function code, send data address, send data length
Return value	None
Remarks	In ASCII mode, the data is converted within the function. Please pass RTU data as the first argument. The checksum is automatically calculated internally.

[Function Name] R_MODBUS_Receive

Synopsis	Receive Modbus frame
Header	None
Declaration	uint8_t R_MODBUS_Receive (st_modbus_receive_frame_t * modbus_receive_frame)
Description	Checks the status of UART0, and if a Modbus frame has been received, parses and stores the frame in the argument modbus_receive_frame. If the frame is received normally, or if an error occurred while the frame was being received, the callback function is called. This function is non-blocking.
Arguments	Modbus frame structure address
Return value	Modbus frame receive result
Remarks	For details about the value specified in the return value, see 5.7.3 Modbus Receive Results Constants.

[Function Name] R_MODBUS_Parse

Synopsis	Parse Modbus frame
Header	None
Declaration	uint8_t R_MODBUS_Parse (uint8_t * modbus_rx_buffer, uint8_t modbus_frame_size, st_modbus_receive_frame_t * modbus_receive_frame)
Description	Parses modbus_rx_buffer and stores it in the argument modbus_receive_frame. Parses the Modbus frame to determine whether the slave address in the function is addressed to itself, the checksum is correct, and the corresponding function code exists, and then returns the parse result as the return value.
Arguments	Modbus frame address, Modbus frame size, Modbus frame structure address
Return value	Modbus frame parse result
Remarks	For details about the value specified in the return value, see 5.7.3 Modbus Receive Results Constants.

[Function Name] R_MODBUS_Send_Error

Synopsis	Send error response
Header	None
Declaration	void R_MODBUS_Send_Error (uint8_t slave_address, uint8_t function, uint8_t error_code)
Description	Send the error_code corresponding to the error in the function code specified in function. If slave_address is 0x00 (broadcast), no data is sent.
Arguments	Destination slave address of received frame, function code in which error occurred, error code
Return value	None
Remarks	None

[Function Name] r_modbus_callback_slave_receiveend

Synopsis	Called after receiving the Modbus frame in slave mode
Header	None
Declaration	void r_modbus_callback_slave_receiveend (st_modbus_receive_frame_t modbus_receive_frame)
Description	The sample code implements the processing according to the function code.
Arguments	Modbus receive frame
Return value	None
Remarks	None

[Function Name] r_modbus_callback_slave_receiveend

Synopsis	Called after receiving the Modbus frame in master mode
Header	None
Declaration	void r_modbus_callback_master_receiveend (st_modbus_receive_frame_t modbus_receive_frame)
Description	The sample code implements the processing according to the function code.
Arguments	Modbus receive frame
Return value	None
Remarks	None

[Function Name] r_modbus_callback_error

Synopsis	Called if an error occurs in Modbus communication
Header	None
Declaration	void r_modbus_callback_error (uint8_t error_type, st_modbus_receive_frame_t modbus_receive_frame)
Description	The sample code implements the processing according to the error type.
Arguments	Error type, Modbus receive frame
Return value	None
Remarks	None

5.13 Log Specifications

The specifications for the logs used in this sample program are described in Table 5-13.

Table 5-14 Log Specifications

Timing of Log Output	Log Content (String to be Output)
Modbus frame is received	[RX] + received Modbus frame
Modbus frame is sent	[TX] + sent Modbus frame
Abnormality occurs after parsing Modbus frame	Name of constant corresponding to Modbus receive result. For details, see 5.8.3 Modbus Receive Results Constants
Received callback function is called	CALL_BACK_FUNCTION + name of function

5.14 ROM/RAM Size

The ROM/RAM sizes used in this sample code are shown in Table 5-14. The ROM/RAM sizes below assume a configuration with a Modbus send buffer size of 64 bytes and a Modbus receive buffer size of 64 bytes. (These sizes are when the optimization level is set to debug priority. Size varies with optimization level.)

Table 5-15 ROM/RAM Sizes

Operating Mode	ROM Size	RAM Size
Slave Mode (ASCII)	12,926Byte	895Byte
Slave Mode (RTU)	10,793Byte	895Byte
Master Mode (ASCII)	11,230Byte	909Byte
Master Mode (RTU)	10,145Byte	909Byte

6. Preparing to Run

6.1 RL78 – PC (GUI) Environment

6.1.1 Connection Example

Refer to 4.1.1 RL78 – PC (GUI) Environment.

6.1.2 Set Firmware Constants

Set the constant MODBUS_MODE defined in modbus.h to MODBUS_RTU_SLAVE_MODE, build the project, and then download it to the debug tool.

For the file organization see 5.3 File Organization.

Note: In this example, the RL78 communicates in slave mode (RTU), but the communication method can be changed by changing the MODBUS_MODE setting. For the settings see 5.8.1 Modbus Operation Configuration Constants.

6.1.3 GUI Parameter Settings

The GUI parameter settings are shown in Figure 6-1.

Configure the COM port under "Serial setting" according to your environment.

Note: In this example, the GUI communicates in master mode (RTU), but you can change the communication method by changing the settings under "Connection" and "Serial setting".

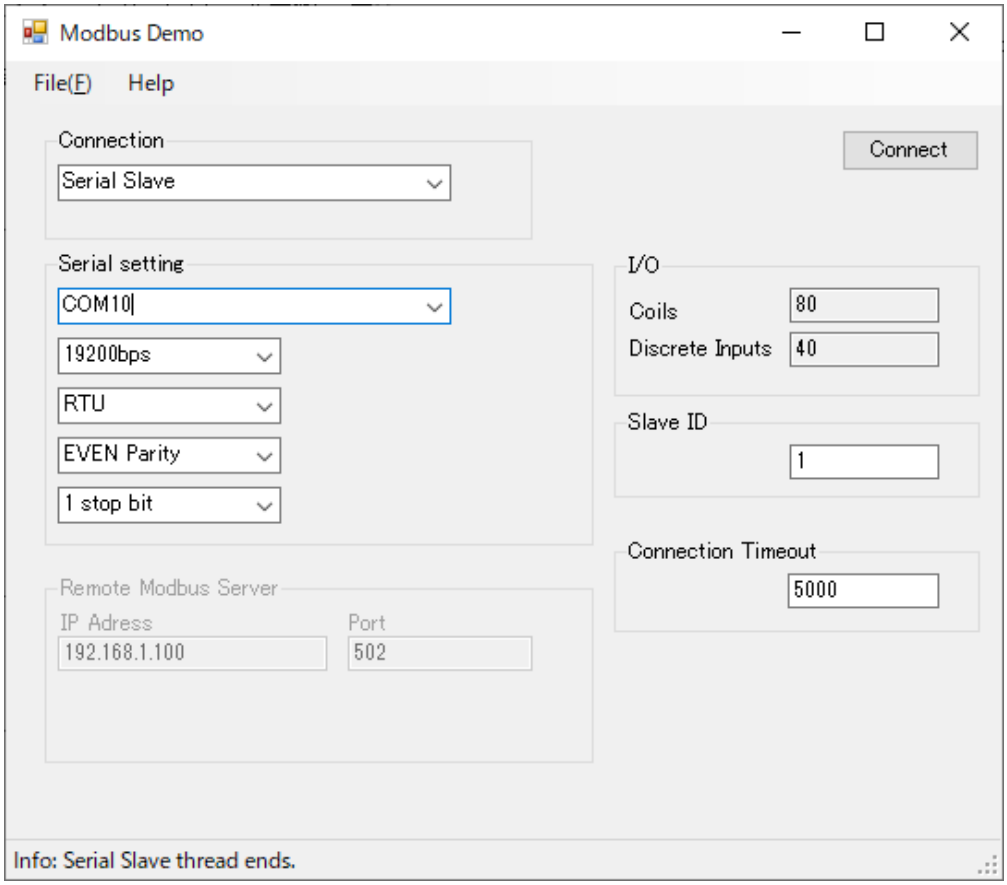


Figure 6-1 GUI Parameter Settings

6.1.4 How to Run

After executing the RL78 program, press the Connect button on the GUI to start the demo.

Operation example: Log output on the slave side

```
[RX]01020000000879CC
CALL_BACK_FUNCTION_READ_DESCRETE_INPUTS
[TX]010201CA21DF
[RX]010F0000000801013F55
CALL_BACK_FUNCTION_WRITE_MULTIPLE_COILS
[TX]010F00000008540D
[RX]01020000000879CC010F0000000801027F54
CALL_BACK_FUNCTION_READ_DESCRETE_INPUTS
[TX]010201CA21DF
```

6.2 RL78 – RL78 Environment

6.2.1 Connection Example

Refer to 4.1.2 RL78 – RL78 Environment.

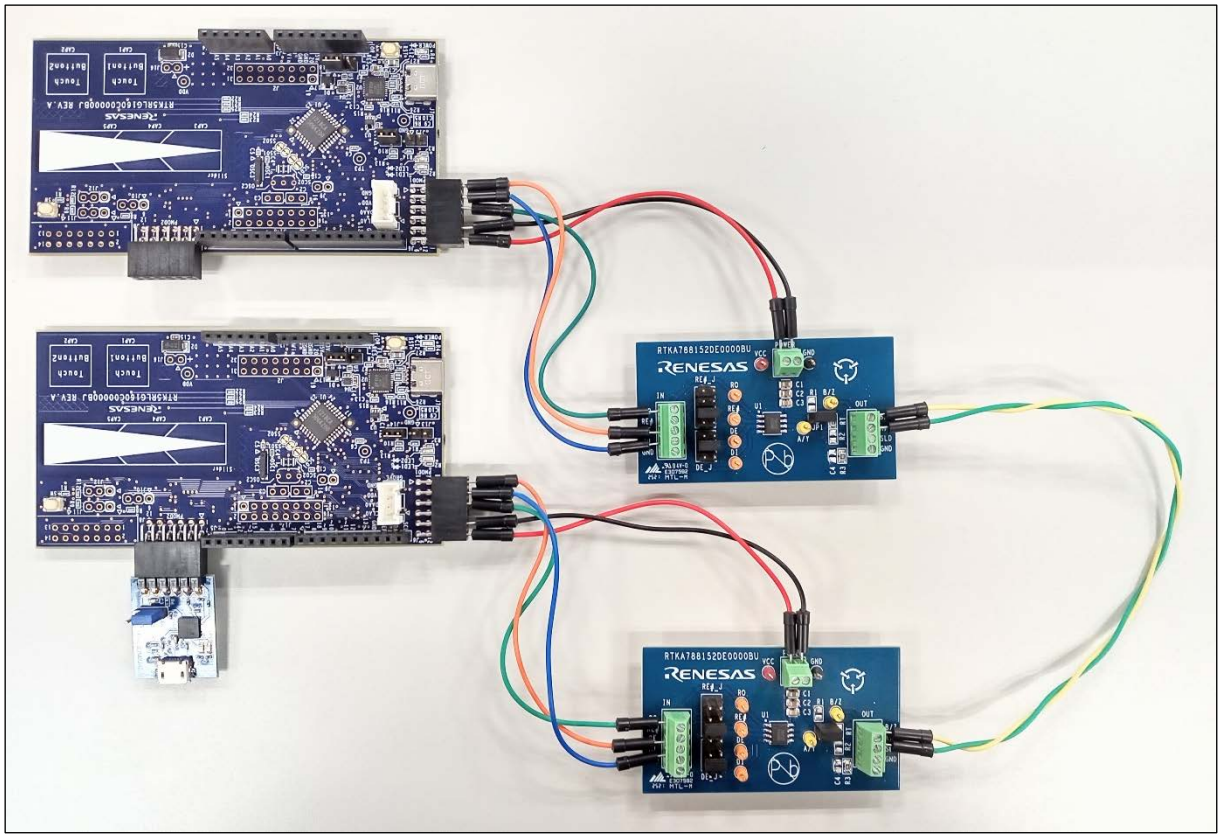


Figure 6-2 The photo of connection between the boards

6.2.2 Set Firmware Constants

Set the constant MODBUS_MODE defined in modbus.h to MODBUS_RTU_SLAVE_MODE, build the project, and then download it to the RL78 Fast Prototyping Board debug tool on the slave side.

Set the constant MODBUS_MODE defined in modbus.h to MODBUS_RTU_MASTER_MODE, build the project, and then download it to the RL78 Fast Prototyping Board debug tool on the master side.

Note: In this example, the RL78 communicates in slave mode (RTU) and Master Mode (RTU), but you can change the communication method by changing the MODBUS_MODE setting. For the settings see 5.7.1 Modbus Operation Configuration Constants.

6.2.3 How to Run

After executing the RL78 program on the slave side, start the demo by executing the RL78 program on the master side.

Log output on the master side

```
[TX]010100000001FDCA
[RX]010101019048
CALL_BACK_FUNCTION_READ_COILS
[TX]010100000001FDCA
[RX]010101019048
CALL_BACK_FUNCTION_READ_COILS
```

Log output on the slave side

```
[RX]010100000001FDCA
CALL_BACK_FUNCTION_READ_COILS
[TX]010101019048
[RX]010100000001FDCA
CALL_BACK_FUNCTION_READ_COILS
[TX]010101019048
```

Website and Support

Renesas Electronics Website
http://www.renesas.com/
Inquiries
http://www.renesas.com/contact/

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 29, 2023	—	First Edition
1.10	Oct. 20, 2023	—	Changed value of the interval between frames for RTU (Sample code)

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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

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/.