16-port Serial Server

Example Code Introduction for 64-bit NuMicro® Family

Document Information

Application	This program implements the core functions of a 16-port serial server, including TCP transparent transmission and bidirectional conversion between Modbus TCP and Modbus RTU/ASCII protocols.
BSP Version	MA35D1 Series Buildroot 2024
Hardware	Gateway-MA35D16F V0.2

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This program implements the core functionalities of a 16-port serial server, achieving TCP transparent transmission and protocol conversion between Modbus TCP and Modbus RTU/ASCII.

Operation modes include:

- TCP Server Mode
- TCP Client Mode
- ModbusRTU Master Mode
- ModbusRTU Slave Mode
- ModbusASCII Master Mode
- ModbusASCII Slave Mode

1.1 Principle

Utilizing the 16-channel UART and dual Ethernet interfaces of the MA35D1 series microprocessor, the software emulates the core functionality of commercial serial servers.

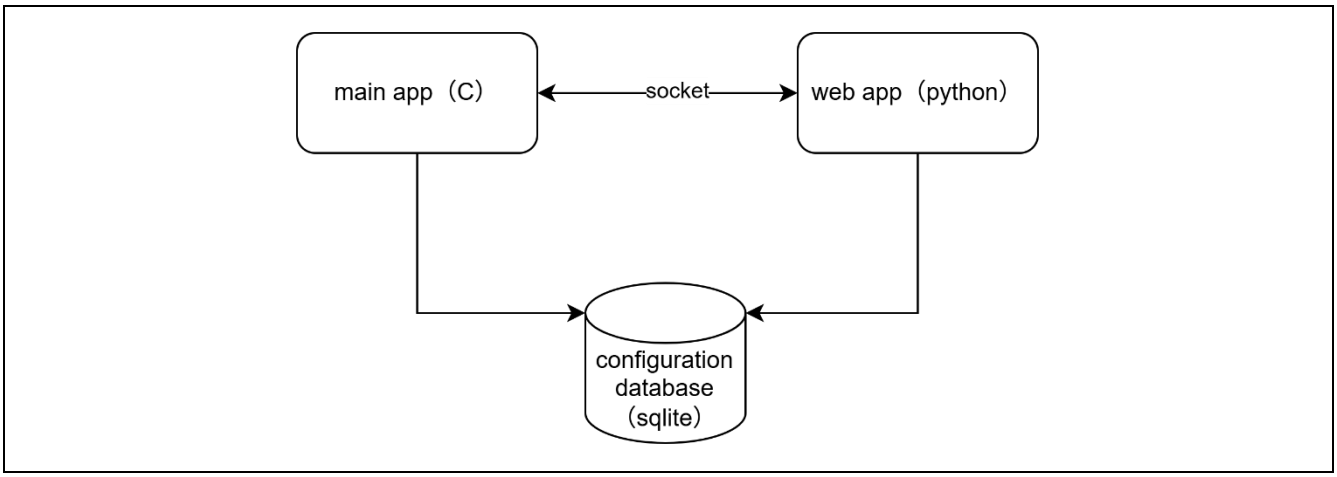

Figure 1-1 Software Architecture Diagram

The software architecture is shown in Figure 1-1:

- The main program is written in C, implementing core Modbus protocol conversion functions.
- The web application uses Python to provide web server functionality, enabling browser-based configuration of the serial server.
- SQLite database stores all configuration parameters of the serial server.

1.2 Compilation Steps

This program requires dependencies such as Python and SQLite during compilation, allowing convenient referencing of these dependencies through Buildroot. Consequently, the program is also compiled in the form of a Buildroot package. Unless otherwise specified in this document, the Buildroot directory serves as the base directory.

For MA35D1 environment setup and image file generation, refer to: MA35D1 Buildroot Project User Manual.

Detailed Steps:

1. Copy the *serial_server* directory to package, and then add the following line in *package/Config.in*:

```
menu "Applications"
        source "package/applications/Config.in"
endmenu
#Add
source "package/serial_server/Config.in"
```

2. Run make menuconfig in buildroot and add the following configurations：

   1) Select serial server sample.

      [*] MA35 serial server demo

   2) Enable Python and necessary Python packages.

      Target packages --->Interpreter languages and scripting

          [*] python3

              ---> python3 module format to install (.py sources and .pyc compiled)

              ---> core python3 modules

                  [*] sqlite module

              ---> External python modules

                  [*] python-flask-login

                  [*] python-netifaces

                  [*] python-psutil

   3) Enable dhcpcd.

      Target packages ---> Networking applications ---> [*] dhcpcd

   4) Configure dts file.

      Kernel ---> (nuvoton/ma35d1-gateway-512m) In-tree Device Tree Source file names

   5) Configure the boot device.

      Bootloaders ---> (ma35d1_sdcard0) Board defconfig

3. Copy the *sserver* directory into buildroot directory. Modify *package/serial_server/serial_server.mk* to ensure that the SERIAL_SERVER_SITE path is correct.

```
SERIAL_SERVER_SITE = $(TOPDIR)/sserver
```

4. Copy *ma35d1-gateway-512m.dts* to *output/build/linux-custom/arch/arm64/boot/dts/nuvoton*.

5. UART driver modification. To adapt the RS485 transceiver for the Gateway-MA35D16F board, add GPIO-controlled RTS signal code, and modify *output/build/linux-custom/drivers/tty/serial/ma35d1_serial.c*.

```
#include <linux/io.h>
#include <linux/gpio.h> //Add for gpio RTS
#include <linux/of_gpio.h>//Add for gpio RTS
#include <asm/irq.h>
#include <asm/serial.h>

…
struct uart_ma35d1_port {

…
        int max_count;
        int rts_gpio; //Add for gpio RTS
        /* We provide a per-port pm hook. */
        void    (*pm)(struct uart_port *port,
                        unsigned int state, unsigned int old);
};

…
static void rs485_stop_rx(struct uart_ma35d1_port *port)
{

}

//Add for gpio RTS
static void set_rs485_rst(struct uart_ma35d1_port *p,int value)
{
        int i=0;
        i= value;

        if(p->rts_gpio != -EINVAL){
                gpio_direction_output(p->rts_gpio,i);
        }

}
//Add end
…
static inline void __stop_tx(struct uart_ma35d1_port *p)
{
        unsigned int ier;

        ier = serial_in(p, UART_REG_IER);
//Modify for gpio RTS
        //if (ier & THRE_IEN)
```

```
        //        serial_out(p, UART_REG_IER, ier & ~THRE_IEN);
        if (ier & THRE_IEN){
                if(p->rts_gpio!=-EINVAL){
                        if(serial_in(p,UART_REG_FSR)&TE_FLAG){
                                set_rs485_rst(p,0);
                                serial_out(p, UART_REG_IER, ier & ~THRE_IEN);
                        }
                }else{
                        serial_out(p, UART_REG_IER, ier & ~THRE_IEN);
                }
        }
//Modify end
        if (p->rs485.flags & SER_RS485_ENABLED)
                rs485_start_rx(p);
}

…
static void ma35d1serial_start_tx(struct uart_port *port)
{
…
         serial_out(up, UART_REG_IER, ier & ~THRE_IEN);
         set_rs485_rst(up,1); //Add for gpio RTS

         if (uart_circ_chars_pending(xmit) <
                 (16-((serial_in(up, UART_REG_FSR)>>16)&0x3F)))

…
}

static int ma35d1serial_probe(struct platform_device *pdev)
{
…
       up->port.rs485_config = ma35d1serial_config_rs485;
//Add for gpio RTS
       up->rts_gpio = of_get_named_gpio(pdev->dev.of_node, "rts-gpio", 0);
        if (gpio_is_valid(up->rts_gpio)) {
                pr_err("test will request gpio\n");
                ret = devm_gpio_request(up->port.dev,
                                        up->rts_gpio, "RTS");
                if (ret) {
                        dev_err(&pdev->dev,
                                "error requesting RTS GPIO\n");
                }

                ret = gpio_direction_output(up->rts_gpio,0);
                if (ret) {
                        dev_err(&pdev->dev, "error setting RTS GPIO\n");
                }
        }else{
                pr_err("test port do not have rts_port\n");
                        up->rts_gpio=-EINVAL;
       }
//Add end

       ret = uart_add_one_port(&ma35d1serial_reg, &up->port);
       set_rs485_rst(up,0); //Ed for gpio RTS
```

```
        platform_set_drvdata(pdev, up);

        return 0;
}
```

6. Run “make linux-rebuild” to rebuild Linux kernel and then run “make”. Upon completion, program the image to the SD0 of the Gateway-MA35D16F board with NuWriter.

2. Code Description

The entire project comprises two processes: a main program written in C and a web configuration utility implemented in Python. The configuration utility solely manages serial server setup and database persistence, while the primary data forwarding functionality resides entirely within the main program. This document focuses on the implementation of the main program.

2.1 Architecture Diagram

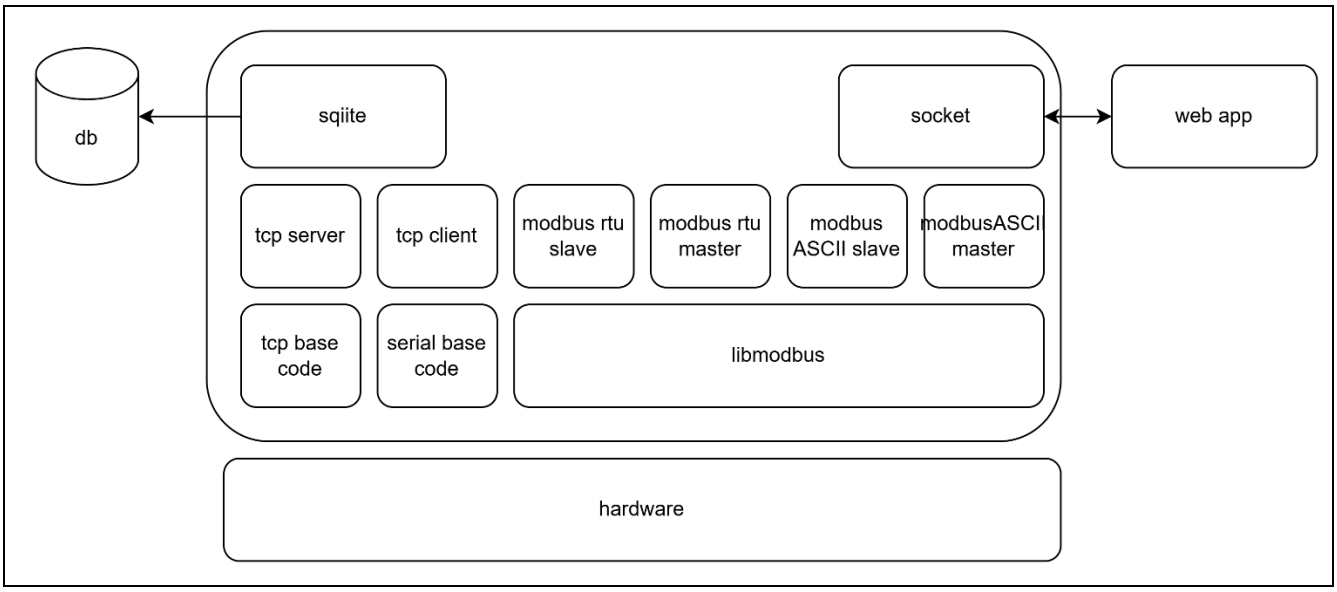

Figure 2-1 Main Program Architecture

The architecture of the main program is depicted in Figure 2-1. Each operational mode is implemented as an independent module. The Modbus xxx module relies on the open-source libmodbus library. Both TCP server and TCP client modes exclusively invoke underlying TCP and serial interfaces. Database access utilizes interfaces from libsqlite, while inter-process communication with the web application is facilitated through internal sockets.

2.2 Main Program Flowchart

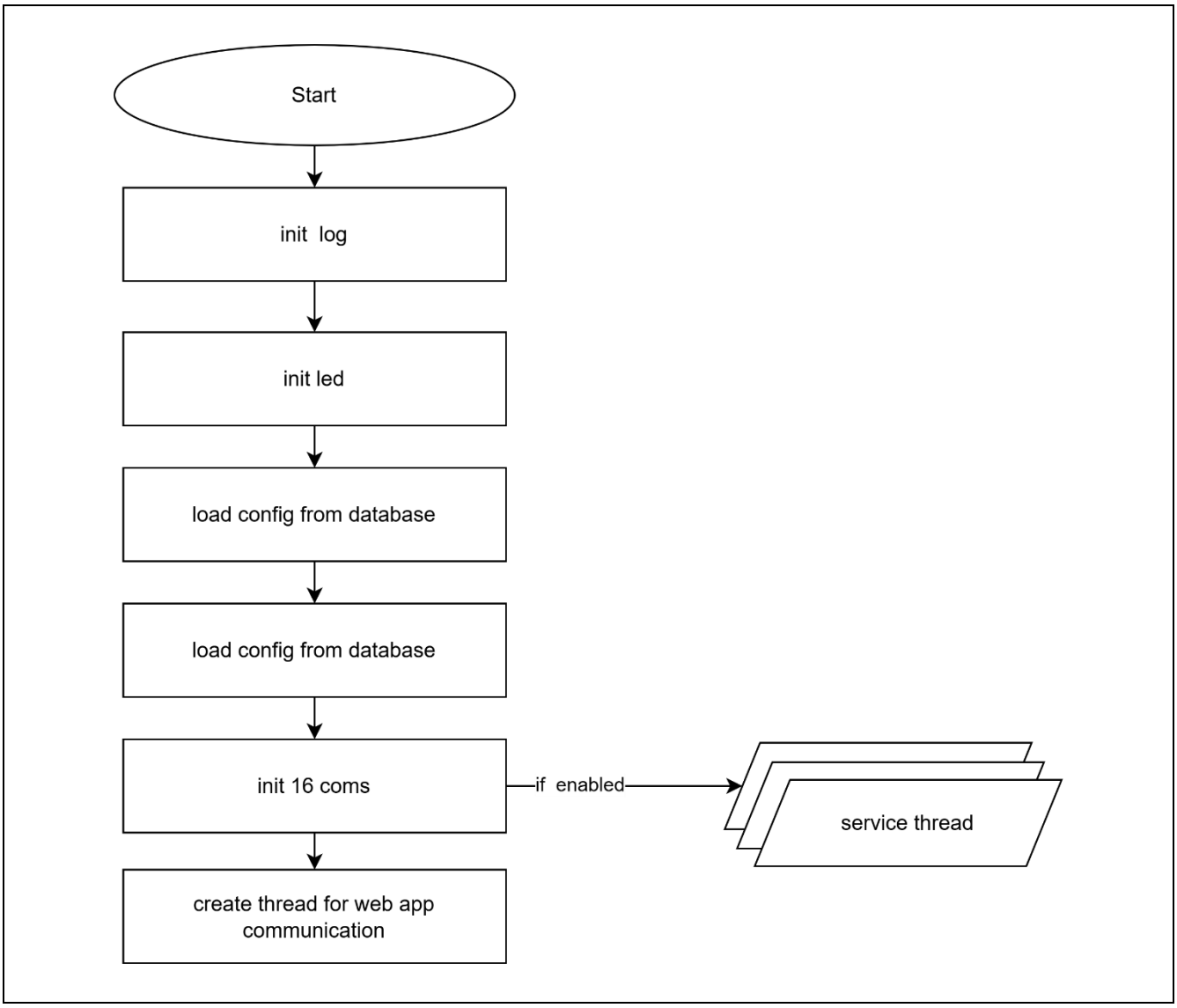

Figure 2-2 Main Program Flowchart

The program flowchart is depicted in Figure 2-2. Upon startup, the procedure initializes the logging system and LED indicators. It then loads configurations for 16-channel serial ports from the database. For each serial port enabled, corresponding operational modes are activated, and separate threads are spawned to await incoming data. After initializing all 16 serial channels, dedicated communication threads are created to interface with the web application.

2.3 Code Description

In the main function, the log interface and LED interface will be initialized first. Then, the g_com_service and g_com_conf structure arrays will be cleared, which store the mode configurations and general configurations corresponding to 16 serial ports. Next, load_config() will read the configurations from the sqlite database and fill these two structure arrays.

```
int main(int argc,char **argv)
{
    int ret = 0;

    log_init(0, basename(argv[0]));
    led_init();
    memset( g_com_service, 0, sizeof(g_com_service));
    memset( g_com_conf, 0, sizeof(g_com_conf));

    load_config();

    com_service_init();

    init_web_command_service();

    while (1)
    {
        //LedRun();
        //usleep(500 * 1000);
        sleep_ms(5000);
    }
    return 0;
}
```

Next, the program enters the com_service_init() function, which will initialize each serial port based on g_com_service. If com_en is equal to 1, it will initialize the corresponding serial port and call the initialization function of the corresponding mode according to net_mode. In each mode's initialization function, a service thread will be created to perform the specific functions of that mode.

```
int com_service_init()
{
    int i;
    struct com_service *p_service;
    struct com_config *p_config;

    unsigned long tmp1,tmp2;

    for(i=0; i<g_sys_conf.uart_count; i++) {
        g_com_service[i].config = &g_com_conf[i];
        p_service = &g_com_service[i];
        p_config = p_service->config;

        SLOG_DEBUG("i=%d ,p_config:0x%x ,net_mode:%d
                  com_en:%d\n",i,p_config,p_config->net_mode,p_config->com_en);
        if(p_config!=NULL && p_config->com_en == 1) {
            switch(p_config->net_mode) {
            case MODE_TCP_SERVER:
                mode_tcp_server_init(p_service);
                break;
            case MODE_TCP_CLIENT:
                mode_tcp_client_init(p_service);
                break;
            case MODE_RTU_SLAVE:
                rtu_slave_init(p_service);
                break;
```

```
            case MODE_RTU_MASTER:
                rtu_master_init(p_service);
                break;
            case MODE_ASCII_MASTER:
                ascii_master_init(p_service);
                break;
            case MODE_ASCII_SLAVE:
                ascii_slave_init(p_service);
                break;
            default:
                break;
            }
            led_enable(i <= 7?i:i+8,1); //use the first row and third row to show the
status of com service.
            get_serial_count(i+1,&p_service->tx_base,&p_service->rx_base);
        }
    }

}
```

The following explains the key data structures.

Each serial port corresponds to a com_service structure. The config member points to the serial port configuration, while rx_base and tx_base are used to store the queried receive and transmit data byte counts during program runtime for statistical purposes. The mode_private pointer points to additional data specific to the operating mode, typically filled during mode initialization.

```
struct com_service {
    struct com_config *config;
    unsigned long rx_base;
    unsigned long tx_base;
    void *mode_private;
};
```

The com_config structure stores serial port configuration members, which are depicted as follows:

- inx: serial port number
- com_en: serial port enable status
- com_mode: operating mode
- s_conf: stores basic UART configuration (e.g., baud rate, parity check, etc.)
- pkt_size, pkt_time, pkt_mode: packet-related configurations
- mode_conf: saves mode-specific configurations for each operating mode.

```
struct com_config {
    int inx;
    int com_en;
    int large_flow;//NOT USE
    int com_mode;
    struct serial_configure s_conf;
    int pkt_size;
```

```
    int pkt_time;
    int net_mode;
    void *mode_conf;
};
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - MA35D1 Series Buildroot 2024

3.2 Hardware Requirements

- Circuit components
  - Gateway-MA35D16F V0.2
- Pin Connect

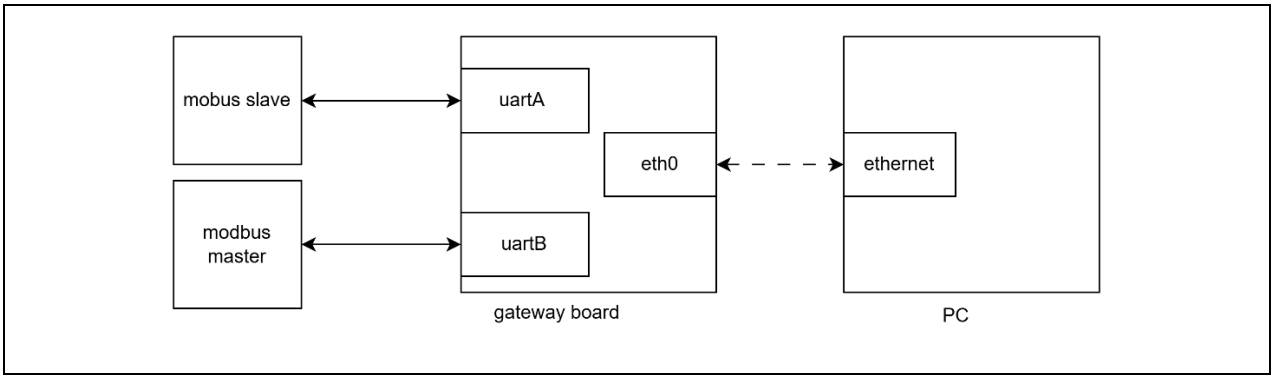

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

MA35D1_Serial_Server_V1.00	
sserver	Project directory
common	Common source files directory
database	Database operation code directory
drv	UART and LED operation code directory
log	Log source files directory
modbus	Libmodbus code directory
network	TCP operation code directory
web_command	IPC with web app directory
service	All modes sources code directory
web_app	Web app source code directory
scripts	Start scripts, Network config scrips
serial_server	Buildroot package file directory

Figure 4-1 Directory Structure

5. Example Code Execution

During package compilation in Buildroot, the build script is *serial_server/serial_server.mk*.

```
SERIAL_SERVER_SITE = $(TOPDIR)/sserver
SERIAL_SERVER_VERSION = 1.0.0
SERIAL_SERVER_SITE_METHOD = local
SERIAL_SERVER_INSTALL_STAGING = YES

define SERIAL_SERVER_INSTALL_TARGET_CMDS
    # install C program
    $(INSTALL) -D -m 0755 $(@D)/uart_server $(TARGET_DIR)/usr/app/
        $(INSTALL) -D -m 0755 $(@D)/scripts/S60sserver $(TARGET_DIR)/etc/init.d/
        install -m 0755 $(@D)/scripts/* $(TARGET_DIR)/usr/app/scripts/

    # install PYC
    PYTHONPATH="$(TARGET_DIR)/usr/lib/python$(PYTHON3_VERSION_MAJOR)/site-
packages:$(TARGET_DIR)/usr/lib" \
        $(HOST_DIR)/bin/python$(PYTHON3_VERSION_MAJOR) $(@D)/web_app/compile.py
        $(INSTALL) -D -m 0644 $(@D)/web_app/app.pyc $(TARGET_DIR)/usr/app/
        $(INSTALL) -D -m 0644 $(@D)/web_app/database.pyc $(TARGET_DIR)/usr/app/
        $(INSTALL) -D -m 0644 $(@D)/web_app/check_and_init_db.pyc $(TARGET_DIR)/usr/app/
        rsync -a --chmod=u=rwX,go=rX $(@D)/web_app/templates $(TARGET_DIR)/usr/app/
        $(INSTALL) -D -m 0644 $(@D)/web_app/database_schema.sql $(TARGET_DIR)/usr/app/

endef

$(eval $(cmake-package))
```

The build script installs the main program, web application, configuration scripts, and HTML resources to their designated paths. Specifically, the init cript.*sserver/scripts/S60sserver* is deployed to */etc/init.d/.* Upon Linux system startup, both the main program and web application will auto-launch.

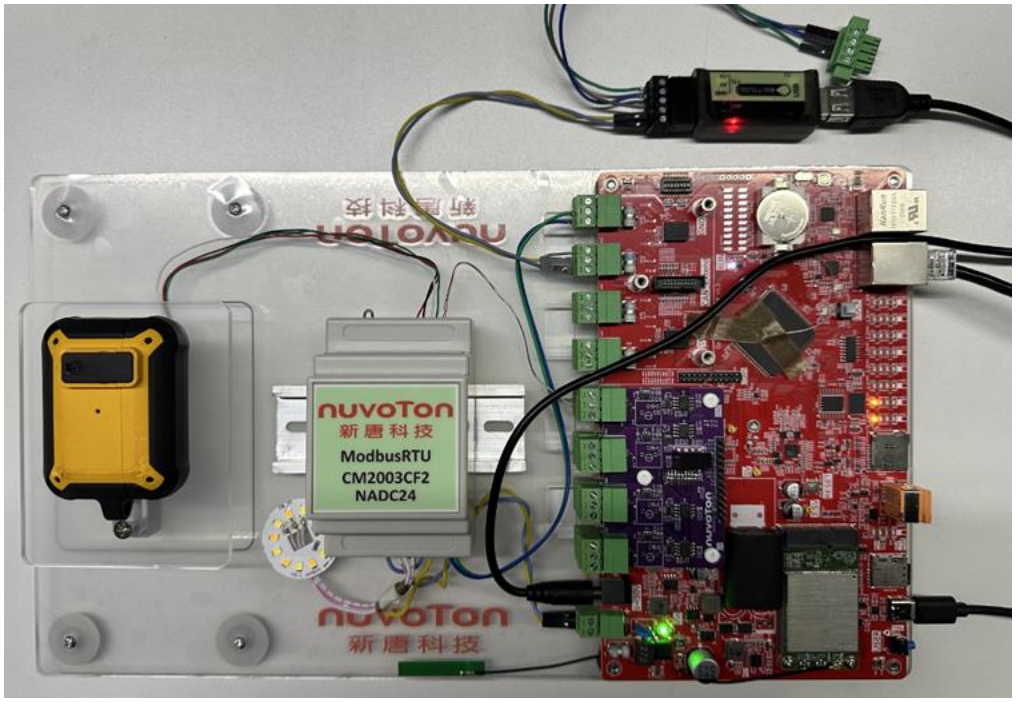

Figure 5-1 Physical Wiring

As shown in Figure 5-1, which illustrates the wiring example of the serial server, COM1 of the serial server is connected to the CM2003 Modbus RTU slave device, while eth0 is connected to the computer via Ethernet.

After boot, connect an Ethernet cable to eth0 to access the configuration interface via browser：

http://192.168.0.200:5000
Username: **admin**
Password: **admin**

Configure COM1 as "Modbus RTU Slave mode" on the web interface, and set the TCP port number (e.g., "10001").

The slave device's data can be read on the computer using Modbus TCP software. Reading 10 Input Register values starting at address 1000 is shown in Figure 5-2.

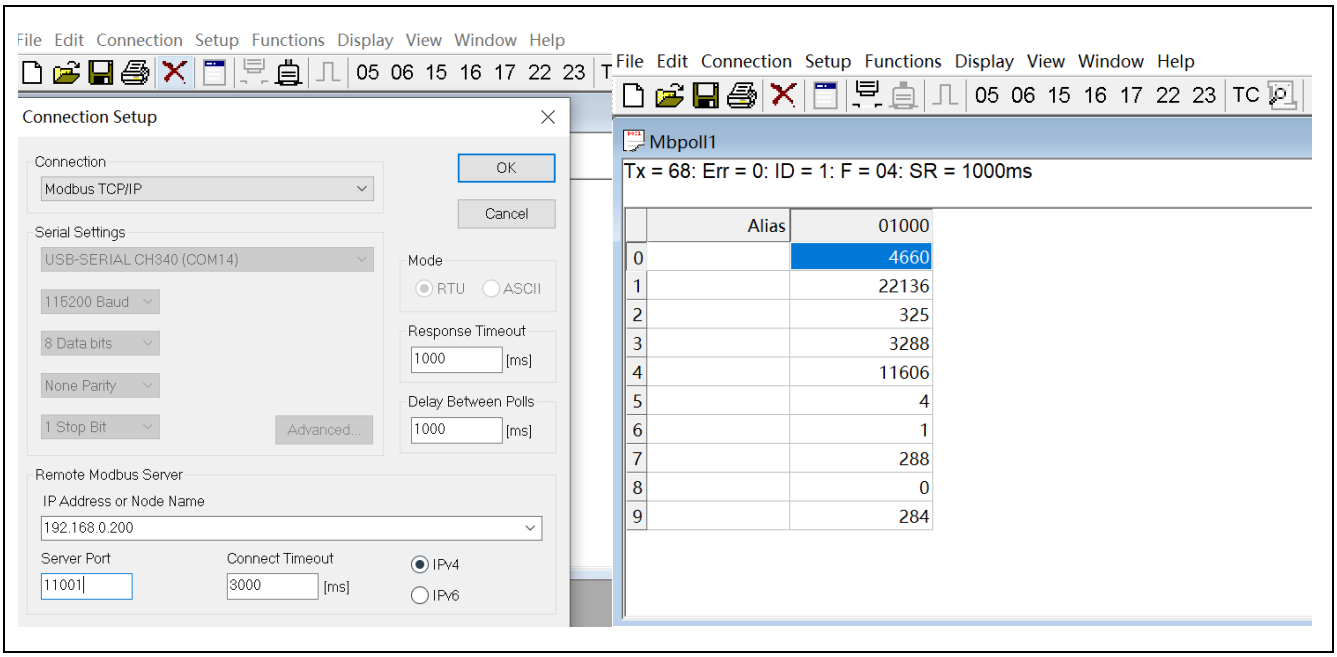

Figure 5-2 Slave Device Data

6. Revision History

Date	Revision	Description
2025.06.16	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.