ECM50 Software Application Guide



Chengdu Ebyte Electronic Technology Co.,Ltd.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Contents

Disclaimer 2
1. Overview 3
    1.1. Software Resources 3
2. Quick Start 3
    2.1. Development software installation 3
    2.2. Wiring and startup 4
    2.3. Light up LED 5
    2.4. Burning firmware 7
3. Functional testing 8
    3.1. Peripheral interface 9
    3.2. Network interface 20
4. Extended Application 33
    4.1. ModBus TCP 33
    4.2. M QTT 36
5. References 39
6. Revision Notes 40
7. About Us 40

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Disclaimer

EBYTE reserves all rights to this document and the information contained herein. Products, names, logos and designs described herein may in whole or in part be subject to intellectual property rights. Reproduction, use, modification or disclosure to third parties of this document or any part thereof without the express permission of EBYTE is strictly prohibited.

The information contained herein is provided “as is” and EBYTE assumes no liability for the use of the information. No warranty, either express or implied, is given, including but not limited, with respect to the accuracy, correctness, reliability and fitness for a particular purpose of the information. This document may be revised by EBYTE at any time. For most recent documents, visit www.cdebyte.com.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1. Overview

The ECM50 Software Evaluation Guide describes the testing steps and evaluation methods for the core and peripheral resources of a system running on Ebyte's development boards. This document can be used as a preliminary evaluation guide or as a general system development test guide.

1.1.Software Resources

The ECM50 is an industrial-grade, programmable computer designed for full-scenario communication based on the ESP32-S3 chip. It supports 4G, Ethernet, Wi-Fi, and Bluetooth, as well as multiple network protocols such as TCP, HTTP, and MQTT. Users can program in MicroPython to execute customized business logic and data processing on the RTU. It is suitable for various IoT data collection, transmission, and control scenarios, and users can quickly develop their own business functions using the rich official case source code. This significantly reduces development cycles and project development costs.

The ECM50 is equipped with the MicroPython operating system, which is included with the entire machine. Users can use the provided examples to run some code for testing. It includes LEDs , GPIO , DI , DO, AI , ETHERNET , 4G , LoRa, WIFI , BLE , 2 32/485 serial ports, and an SD card.

2. Quick Start

2.1.Development software installation

E CM50 comes with MicroPython firmware. Users can start testing directly after receiving it. If you need to re-burn the firmware, please refer to Section 2.4 .

We use the Thonny Python IDE, officially recommended by MicroPython. Thonny IDE is open source software designed with minimalist design and is very compatible with MicroPython. It supports Windows, macOS, Linux, and Raspberry Pi.

Install CH340 driver: 03_Tools/ CH340 driver.rar

Thonny software installation package: 03_Tools/thonny-4.1.7.exe .

After the installation is complete, open it as shown below.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



2.2.Wiring and startup

Use a TYPE-C data cable to connect to the port on this side of the ECM50 . This port serves as both a power supply and a serial port, as shown in the figure below.



If you need long-term stable power supply, or need to use peripherals with high power requirements such as 4G , you need to connect the DC power supply, which is located on the far left of the connection terminal and is printed with DC8 ~28V .

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



After following the first method, ECM50 will start automatically.

2.3.Light up LED

While the device is powered and connected to the serial port, open the Thonny software, click Run, and configure the interpreter.



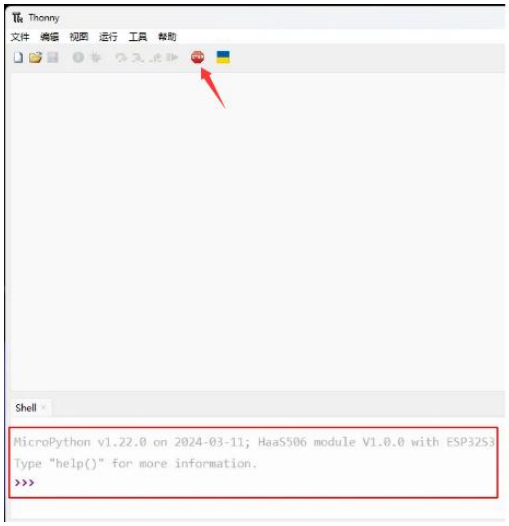
Select ESP32 interpreter, select automatic detection for the port, and click OK.



Click Restart Terminal. If the following Micro Python prompt appears, it means the software

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

and device are connected successfully.



Click File → Open in the upper left corner, select This PC, and then open 04_Sources / led_flash.py .

Then click Run.

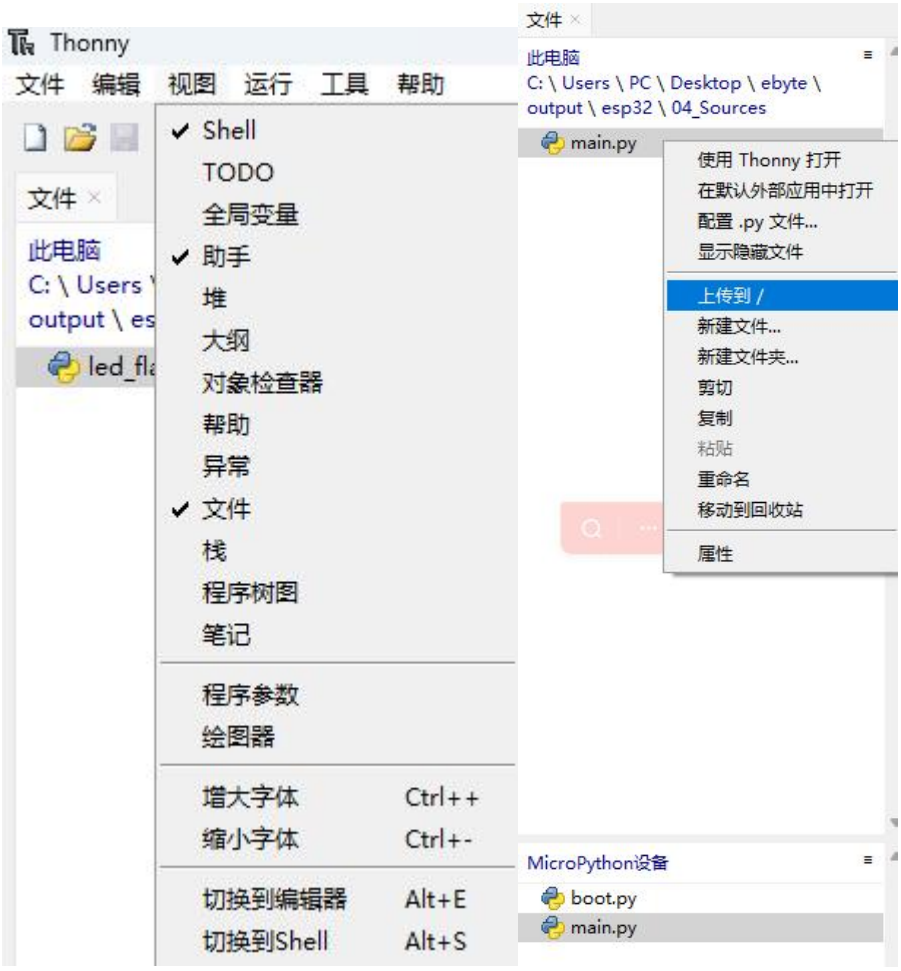


You can see the LED on the board is flashing, and that means you are done.

If you want it to start automatically when the computer is powered on, you need to rename this program to main.py and then upload it to the industrial computer:

1. Open View → File
2. Right click on the file and select Upload to/ .
3. LED light will flash automatically .

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



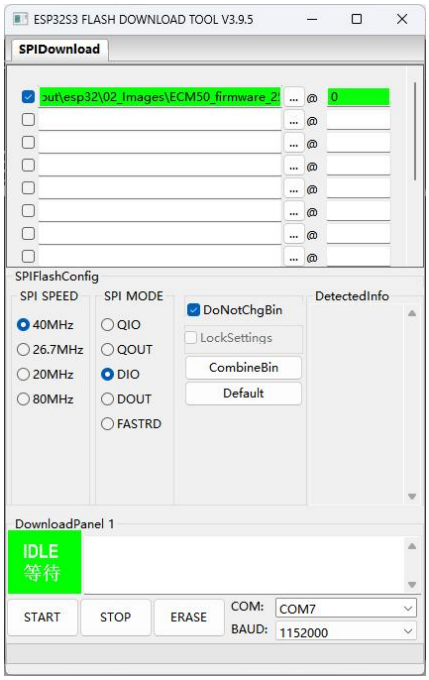
2.4.Burning firmware

Use a typec data cable to connect the typec interface of our E CM50 .

flash_download_tool_3.9.5 in the Tools directory . After unzipping, open flash_download_tool_3.9.5.exe and select ESP32-S3 .

ECM50 firmware in the Images directory . Then, select the serial port and baud rate as 1152000. Click START to begin programming. After programming is complete, exit and press Reset or power on to begin testing.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



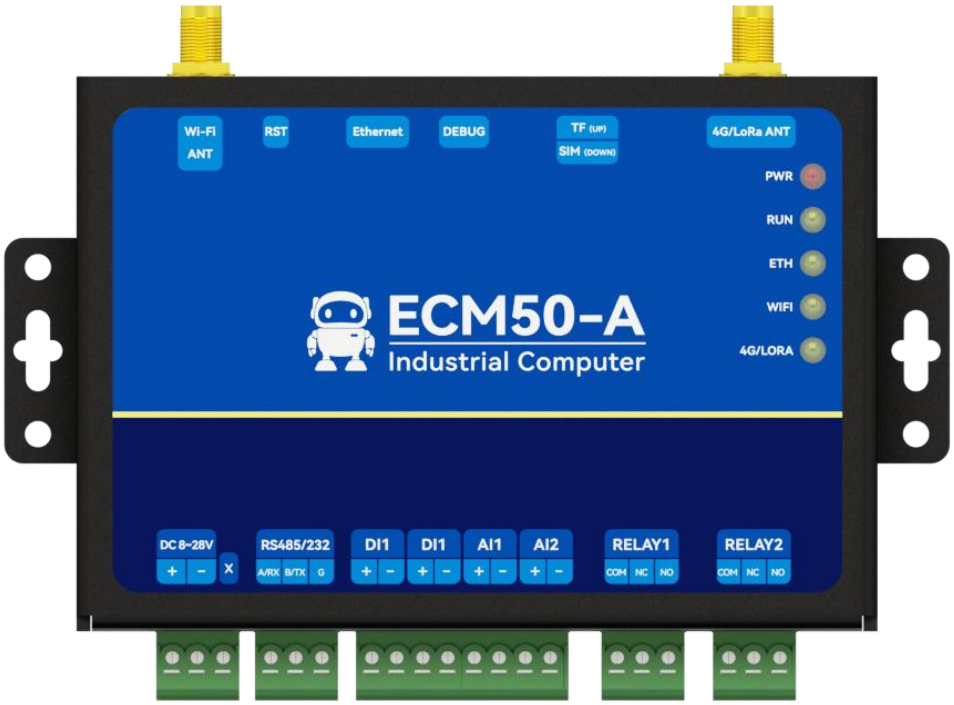
3. Functional testing

Python code to test each function of the industrial computer . Users only need to open the corresponding Python file in 04_Sources and click Run, just like the method for testing the LED above .

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.1.Peripheral interface

3.1.1. led



ECM50 has a total of five lights, namely 4G/L ora, W IFI , E TH , R UN , and P WR .

P WR lights up when power is turned on.

WIFI lamp and R UN lamp are lamps that users can control programmatically using IO .

4G /Lora light is a light that flashes automatically when the user is using 4G or Lora and cannot be controlled by IO .

ETH is a light that flashes automatically when the user is using Ethernet and cannot be controlled by IO .

The following code controls the flashing of the WIFI light and the R UN light.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
import time
from machine import Pin

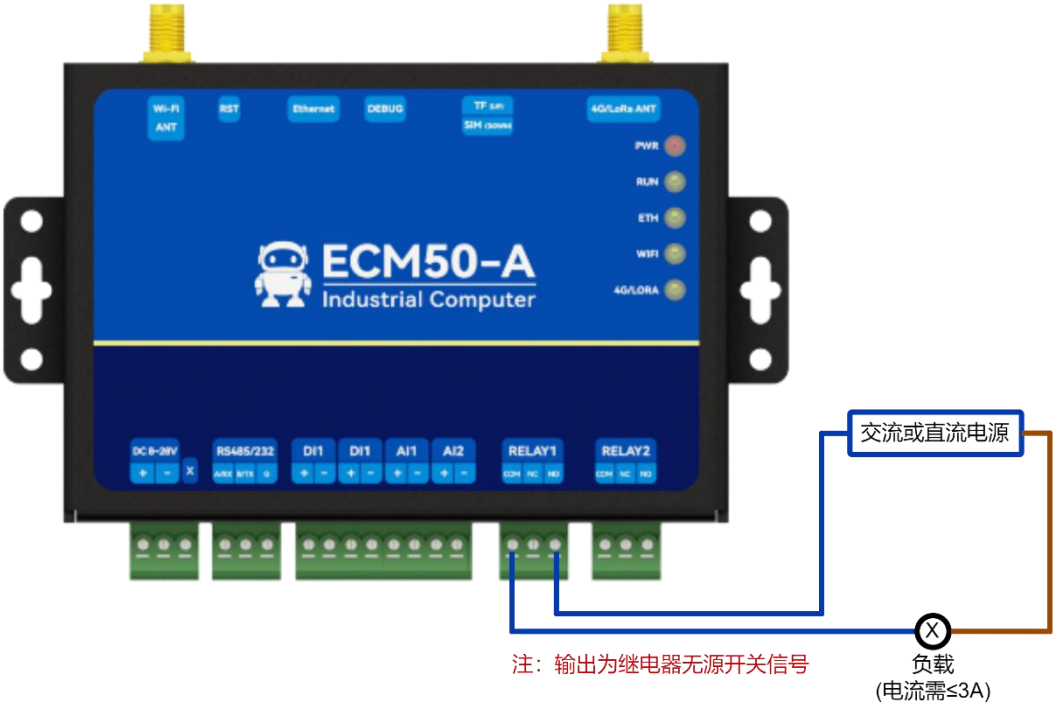

def main():
    # RTU A total of 2 controllable indicator lights
    led_wifi = Pin(45, Pin.OUT) # LED2 on GPIO 39 wifi
    led_sta = Pin(21, Pin.OUT) # LED4 on GPIO 21 Operate
    while True:
        led_wifi.value(1)
        led_sta.value(1)
        print("led high")
        time.sleep(1)
        led_wifi.value(0)
        led_sta.value(0)
        print("led low")
        time.sleep(1)


#Program startup entry
if __name__ == '__main__':
    main()
```

3.1.2. DO

The ECM50 has two 10A/277VAC N/A circuits. Both use I/ O to control the opening or closing of the circuit.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



As shown in the figure above, users can connect the power supply and electrical appliances in one loop, and then directly control the DO signal to control the on and off of the loop.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
import time
import machine


def main():
    # RTU total of 2 DO
    DO1 = machine.Pin(15, machine.Pin.OUT)
    DO2 = machine.Pin(16, machine.Pin.OUT)
    while True:
        DO1.value(1)
        time.sleep(0.5)
        DO2.value(1)
        print("DO output high")
        time.sleep(1)
        DO1.value(0)
        time.sleep(0.5)
        DO2.value(0)
        print("DO output low")
        time.sleep(1)


#Program startup entry
if __name__ == '__main__':
    main()
```

3.1.3. DI

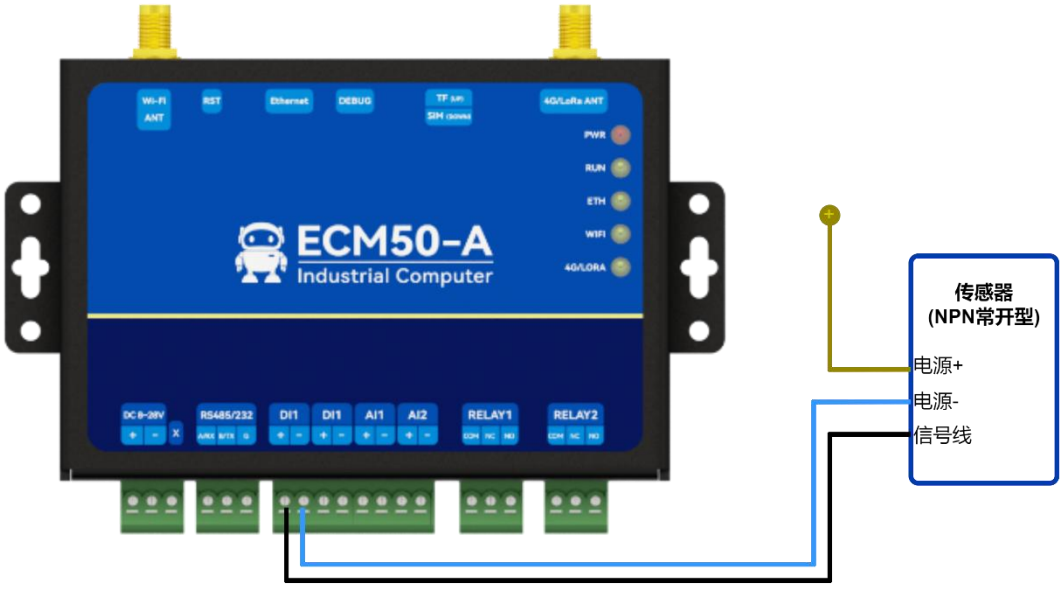
ECM50 has two default NPN dry contacts, which can be changed to wet contacts. We still use I O as input and set the internal pull-down to read the I O value.

In Figure 1, if the switch is closed, the value measured by the ECM50 is low; if it is open, the measured value is high.

Figure 2 shows the sensor controlling the signal line, which determines the level of the measured value.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000





Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
import time
import machine


def main():
    DI1 = machine.Pin(4, machine.Pin.IN,machine.Pin.PULL_DOWN)
    DI2 = machine.Pin(5, machine.Pin.IN,machine.Pin.PULL_DOWN)
    while True:
        di1_value = DI1.value()
        di2_value = DI2.value()
        print(f'[1]di1_value = {di1_value}, di2_value = {di2_value}')
        time.sleep(1)

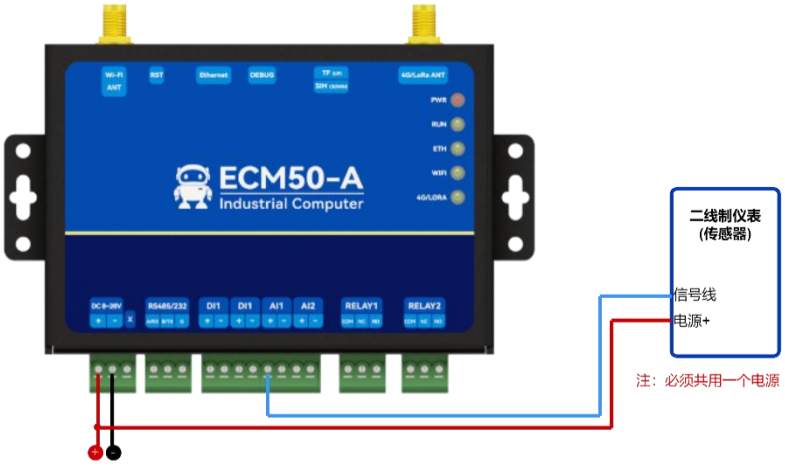

if __name__ == '__main__':
    main()
```

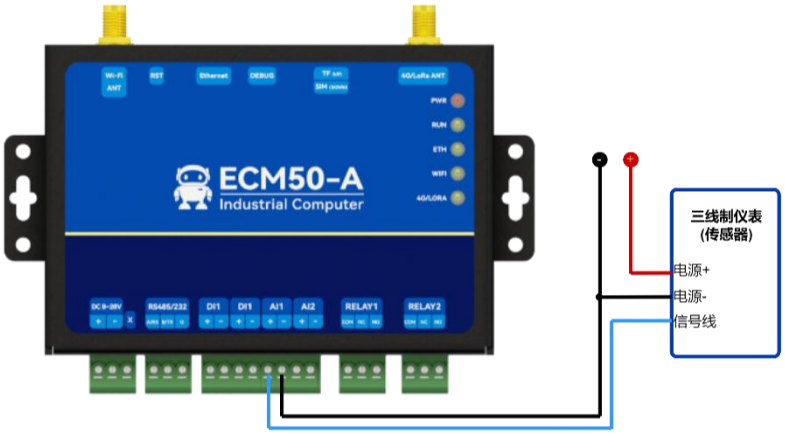
3.1.4. AI

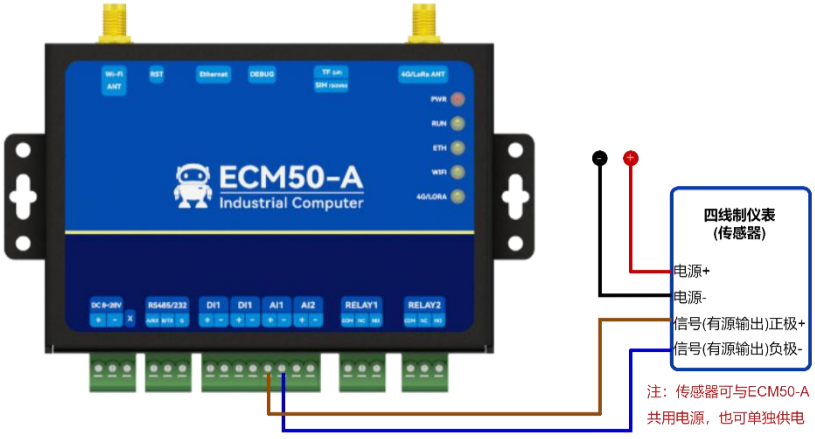
The ECM50 has two default 4-20mA current acquisition channels, which can be changed to voltage acquisition (12-bit accuracy). The actual value read here is still voltage, which is converted to current and tested using a current generator.

A I has the following wiring methods for reference.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000







Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
# ai test example
# ai1-gpio1,ai2-gpio11 current measure range 4-20ma
from machine import ADC
import time
import machine

def main():
    ai1 = ADC(machine.Pin(6)) #ai1
    ai1.atten(ADC.ATTN_11DB)    #Use 0 attenuation, the highest accuracy, input voltage range：0-950mv
    ai2 = ADC(machine.Pin(7)) #ai2
    ai2.atten(ADC.ATTN_11DB)
    while True:
        val_u16_1 = ai1.read_u16()    # read a raw analog value in the range 0-65535
        val_uv_1 = ai1.read_uv()      # read an analog value in microvolts
        val_raw1 = ai1.read() #Get the raw ADC value
        print(f'val_u16_1 = {val_u16_1}, val_uv_1 = {val_uv_1}',f'val_raw1 = {val_raw1}',f'ai1 = {val_uv_1/150/1000}ma')
        time.sleep(0.5)
        val_u16_2 = ai2.read_u16()    # read a raw analog value in the range 0-65535
        val_uv_2 = ai2.read_uv()      # read an analog value in microvolts
        value2 = ai2.read() #Get the raw ADC value
        print(f'val_u16_2 = {val_u16_2}, val_uv_2 = {val_uv_2}',f'val_raw2 = {val_raw2}',f'ai2 = {val_uv_2/150/1000}ma')
        time.sleep(0.5)

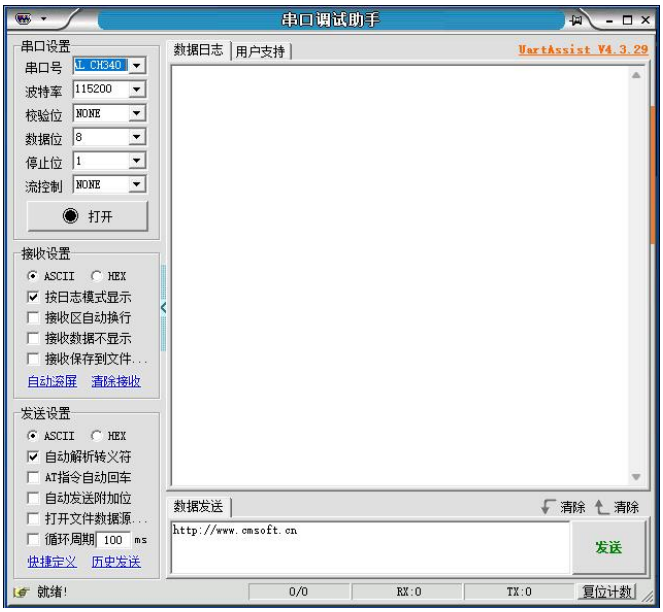
if __name__ == '__main__':
    main()
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.1.5. Serial Port

The ECM50 's serial port 1 can be configured with either RS232 or RS485, depending on the model. We tested the ECM50 by connecting it to a PC using a USB to 485 (auto-transmit/receive) or USB to 232 converter .

Open 03_Tools / UartAssist.exe , open the USB serial port, select 15200 baud rate, no parity, 8 data bits , 1 stop bit, and no flow control. Click Open Serial Port.



04_Sources/RS485.py in Thonny and click Run.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
import machine
import utime
from machine import Pin


def main():
    uart = machine.UART(1, baudrate=115200, tx=17, rx=18)
    while True:
        if uart.any():
            # Read serial port data
            data = uart.read()


            # Print received data
            print("Received data:", data)
        # Stop 100 ms
        utime.sleep_ms(100)
        uart.write('test')
if __name__ == '__main__':
    main()
```

You can see the normal sending and receiving of serial port data. The test code is the same as 2 32 , and you only need to change it to USB to 2 32 to connect to PC .

3.1.6. SD card

ECM50 inserts an SD card into the TF card slot and can read and write files on the SD card through SPI. Note that the SD card needs to be formatted as FAT32 .

Our code mounts the SD card , tries to open a test.txt file and write a line of data, then reads the data, and finally unmounts the SD card .

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
import machine
import os
import utime
def main():
    # Initialize SD card
    sd = machine.SDCard(slot=2)
    # Mounting the SD card
    try:
        os.mount(sd, "/sd")
        print("SD card mounted.")
    except OSError as e:
        print("Error mounting SD card:", e)
    ret = os.listdir('/sd')
    print("SD card:", ret)
    #Writing to a File
    try:
        with open("/sd/test.txt", "w") as f:
            f.write("Hello, SD Card!")
        print("File written to SD card.")
        f.close()
    except OSError as e:
        print("Error writing to file:", e)
    ret = os.listdir('/sd')
    print("SD card:", ret)
    utime.sleep(1)
    # Reading a file
    try:
        with open("/sd/test.txt", "r") as f:
```


Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
                content = f.read()
            print("File content:", content)
        except OSError as e:
            print("Error reading from file:", e)


        #Unmount SD card
        try:
            os.umount("/sd")
            print("SD card unmounted.")
        except OSError as e:
            print("Error unmounting SD card:", e)


if __name__ == '__main__':
    main()
```

3.2.Network interface

3.2.1. Ethernet

Ethernet uses the SPI network conversion chip W5500, RJ4510/100Mbps, connected to the SPI2 interface.

Plug the network cable into the ECM50 and the router. Let's test whether the ECM50 automatically obtains an IP address and performs a TCP communication transmission and reception test with the PC that is also connected to the router.

PC opens 0 3_Tools /netassist5.15.exe .

Select TCP server , set the host address to the PC 's IP address , here it is 192.168.20.5 , set the port number to 8080 , and finally click Open.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



ECM50 side , we can directly run eth.py. We have specified the IP address and port number of the TCP server in the code . Users need to modify them according to their actual situation to enable normal TCP communication .



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
import network
import machine
import time

def main():
    #    Initial SPI and pin
    spi = machine.SPI(1,
                      baudrate=20_000_000,    # 20MHz Clock
                      sck=machine.Pin(12),
                      mosi=machine.Pin(11),
                      miso=machine.Pin(13))
    cs_pin = machine.Pin(47, machine.Pin.OUT)    # Chip Select Pin
    reset_pin = machine.Pin(48, machine.Pin.OUT)
    int_pin = machine.Pin(14, machine.Pin.IN)    #Interrupt pin (optional)

    # Create LAN port
    lan = network.LAN(
        phy_type=network.PHY_W5500,    # specify W5500 drive
        reset=reset_pin,
        spi=spi,                        # SPI bus object
        cs = cs_pin,                    # chip select pin
        int=int_pin,                    # interrupt pin (optional)
        phy_addr=0,                     # PHY address (usually 0 in SPI mode)
        ref_clk_mode=1,                 # Reference clock mode (0=input, 1=output)
        ref_clk=25                      # reference clock pin (based on hardware design)
    print(spi)
    # Activate interface
    lan.active(True)

    # Waiting for connection (up to 10 seconds)
    start_time = time.time()
    while not lan.is connected():
        if time.time() - start_time > 10:
            print("Connection timed out！")
            break
        time.sleep_ms(500)
        print("Wait connection...")
    # Print network configuration
    if lan.isconnected():
        print(" connection success！")
        print("IP address:", lan.ifconfig()[0])
        print("MAC address:", ":".join("{:02x}".format(b) for b in lan.config("mac")))
        do_tcp_client() #tcp test
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
def do_tcp_client():
    import socket
    import time

    # Server's IP address and port
    server_ip = '192.168.20.5'
    server_port = 8080    # Choose a suitable port number

    # Create a socket object
    addr_info = socket.getaddrinfo(server_ip,server_port)
    addr = addr_info[0][-1]
    # Connect to the server
    s = socket.socket()
    s.connect(addr)
    # Send data
    try:
        while True:
            print('send data')
            # send data
            s.send(b'GET / HTTP/1.1\r\nHost: micropython.org\r\n\r\n')
            data = s.recv(1000)
            print(f"Received: {data.decode('utf-8')}")
            # Send data once every 1 second
            time.sleep(1)


    except KeyboardInterrupt:
        print("Client stopped by user.")
    finally:
        # close connection
        client_socket.close()

  #Program startup entry
if __name__ == '__main__':
    main()
```

3.2.2. 4G module

The Lora module and 4G module of ECM50 can be selected from two options. Different models use different communications. The 4G module uses the China Mobile EC801E-CN module and uses the serial port for data interaction. We need a 4G card and insert it into the 4G card slot.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The testing method for our 4G module is to use AT commands to test the TCP service. If the user has other requirements such as MQTT or UDP, they can refer to the relevant manuals of Miguang and follow our tutorials to operate.

We use a TCP testing tool, which is equivalent to using the TCP function and tool of the ECM50's 4G module to establish a connection and send information. https://netlab.luatos.com/

If you choose to enable TCP, you will see an IP address and a port number. Here, the IP address we see is 112.125.89.8 and the port number is 32370.



Insert the obtained IP address into the do_tcp_test function in 4g.py for the TCP test. After powering on, execute the 4G test code: 4g.py. Note that the 4G module consumes a lot of power and requires an additional DC power supply connected to DC8~28V on ECM50 to function properly.

```
64  def do_tcp_test(uart):
65      host = "112.125.89.8"
66      port = "32370"
67      cmd = 'AT+QIOPEN=1,0,"TCP","%s",%s,0,2' % (host, port)
68      resp = send_at(uart, cmd, 3)  # 增加超时时间，等待连接建立
69      print("QIOPEN response:", resp)
70
71      # 等待连接建立
72      utime.sleep(2)
73
```

```
Shell
Response: b'AT+CPIN?\r\r\n+CPIN: READY\r\n\r\nOK\r\n'
Sending Set APN: AT+CGDCONT=1,"IP","CMNET"
Send: AT+CGDCONT=1,"IP","CMNET"
Response: b'AT+CGDCONT=1,"IP","CMNET"\r\r\nOK\r\n'
Sending Activate PDP context: AT+CGACT=1,1
Send: AT+CGACT=1,1
Response: b'AT+CGACT=1,1\r\r\nOK\r\n'
Sending 检查状态: AT+CGACT?
Send: AT+CGACT?
Response: b'AT+CGACT?\r\r\n+CGACT: 1,1\r\n\r\nOK\r\n'
Sending 获取IP地址: AT+CGPADDR=1
Send: AT+CGPADDR=1
Response: b'AT+CGPADDR=1\r\r\n+CGPADDR: 1,"10.0.192.76"\r\n\r\nOK\r\n'
Send: AT+QIOPEN=1,0,"TCP","112.125.89.8",32370,0,2
QIOPEN response: b'AT+QIOPEN=1,0,"TCP","112.125.89.8",32370,0,2\r\r\nCONNECT\r\n'
Sent: test
Received data: b'test'
Sent: test
Received data: b'test'
Sent: test
```



It can be seen that the 4G module can obtain the IP address normally and use TCP to

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

communicate with the network server.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
import machine
import utime
import time
import network
from machine import Pin
import usocket as socket
import sys
import ubinascii
import uos

GL_PWR = Pin(38, Pin.OUT)

def send_at(uart, cmd, timeout=2, wait_for_response=True):
    print(f"Send: {cmd}")
    uart.write(cmd + "\r\n")
    if not wait_for_response:
        return None

    start = utime.time()
    response = b""
    while (utime.time() - start) < timeout:
        if uart.any():
            response += uart.read()
            # 检查是否收到完整响应
            if b"OK" in response or b"ERROR" in response:
                break
        utime.sleep_ms(100)
    return response if response else None

def setup_modem():
    # 硬件初始化
    GL_PWR = Pin(38, Pin.OUT)
    GL_PWR.value(1)  # 使能供电
    uart = machine.UART(2, baudrate=115200, tx=39, rx=40)

    # 等待模块初始化
    print("Waiting for modem initialization...")
    utime.sleep(5)

    # 清空缓冲区
    while uart.any():
        uart.read()
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
# AT 指令序列
    responses = {}
    commands = [
        ('AT', "Basic test"),
        ('AT+CPIN?', "if is ready"),
        ('AT+CGDCONT=1,"IP","CMNET"', "Set APN"),
        ('AT+CGACT=1,1', "Activate PDP context"),
        ('AT+CGACT?', "检查状态"),
        ('AT+CGPADDR=1', "获取 IP 地址"),
    ]

    for cmd, desc in commands:
        print(f"Sending {desc}: {cmd}")
        resp = send_at(uart, cmd, 5)
        responses[cmd] = resp
        print("Response:", resp)
        utime.sleep(1)

    return uart

def do_tcp_test(uart):
    host = "112.125.89.8"
    port = "32370"
    cmd = 'AT+QIOPEN=1,0,"TCP","%s",%s,0,2' % (host, port)
    resp = send_at(uart, cmd, 3)   # 增加超时时间，等待连接建立
    print("QIOPEN response:", resp)

    # 等待连接建立
    utime.sleep(2)

    count = 3
    while count > 0:
        if uart.any():
            # 读取串口数据
            data = uart.read()
            # 打印接收到的数据
            print("Received data:", data)

        # 发送消息并减少计数器
        uart.write('test')
        print("Sent: test")
        count -= 1

        utime.sleep(1)
```

```
# 退出透传模式
    utime.sleep(1)  # 确保没有数据传输
    uart.write(b'+++')  # 发送退出序列，注意使用字节
    utime.sleep(1)  # 等待模块响应

    resp = send_at(uart, "AT+QICLOSE=0", 5)  # 断开连接



def main():
    try:
        uart = setup_modem()
        do_tcp_test(uart)
    except Exception as e:
        print("严重错误:", e)
        import traceback
        traceback.print_exc()
    finally:
        GL_PWR.value(0)

if __name__ == '__main__':
    main()
```

3.2.3. LoRa Module

The ECM50 supports either the LoRa module or the 4G module, with different models featuring different communication capabilities. The ECM50 is equipped with the Ebyte E22-400T22D module. Here, we used the ECM50 and a separate E22-400T22D for communication testing. Alternatively, another ECM50 can be used for LoRa communication testing.

First, prepare the wiring for the separate E22-400T22D module and connect it to the PC using a USB -to-TTL converter . According to the manual, M0 and M1 correspond to four operating modes. This module is configured by default, so there's no need to reconfigure it. Simply select the transmission mode and ground M0 and M1 .

There are four operating modes, which are set by M1 and M0, the details are as follows:

Mode(0-3)	M1	M0	Description	Remark

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

0 Normal mode	0	0	UART and wireless channel are open, transparent transmission is on	Supports configuration over air via special command
1 WOR mode	0	1	Can be defined as WOR transmitter and WOR receiver	Supports wake up over air
2 Configuration mode	1	0	Users can access the register through the serial port to control the working state of the module	
3 Deep sleep mode	1	1	Sleep mode	

After connecting the serial port, open the serial port debugging assistant and set the baud rate to 9600 .

on the PC and run the lora.py code. The code receives and sends data once per second.

The serial port assistant should be able to receive the data sent by ECM50 , and ECM50 should also be able to receive the data sent by the serial port assistant.

```
import machine
import utime
from machine import Pin

def main():
    M1 = Pin(41, Pin.OUT)
    M1.value(0)
    utime.sleep_ms(1000)
    uart = machine.UART(2, baudrate=9600, tx=39, rx=40)
    #uart.write('hello')
    #uart.write('AT+UART=?')

    while True:

        if uart.any():
            # Read serial port data
            data = uart.read()

            # Print received data
            print("Received data:", data)

        # Pause for 1000 milliseconds
        utime.sleep_ms(1000)
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.2.4. WIFI

The ECM50 supports 2.4GHz Wi-Fi (IEEE802.11b/g/n) . Enter the Wi -Fi name and password to join the router's Wi -Fi . It automatically obtains an IP address and then communicates with devices on the LAN via TCP . The TCP communication settings are the same as those in the Network Debug Assistant section . Note the IP address and port number.

```
def do_tcp_client():
    import socket
    import time

    # Server IP address and port
server_ip = '192.168.20.5'
server_port = 8080 # Choose a suitable port number
# Create a socket object
addr_info = socket.getaddrinfo(server_ip,server_port)
addr = addr_info[0][-1]
# Connect to the server
s = socket.socket()
s.connect(addr)
# Send data
try:
while True:
print('send data')
# Send data
s.send(b'GET / HTTP/1.1\r\nHost: micropython.org\r\n\r\n')
data = s.recv(1000)
print(f"Received: {data.decode('utf-8')}")
# Send data every 1 second
time.sleep(1)


    except KeyboardInterrupt:
        print("Client stopped by user.")
    finally:
```


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
            # Close the connection
            client_socket.close()


def do_connect():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('CMCC-AbPL', 'EC12345678')
        while not wlan.isconnected():
            pass
    print('network config:', wlan.ifconfig())
    print(wlan.isconnected())
    do_tcp_client()


#Program startup entry
if __name__ == '__main__':
    do_connect()
```

3.2.5. Bluetooth

Bluetooth can function as both a master and a slave. The code for this is in the files ble_simple_peripheral.py (the Bluetooth peripheral device (slave), ble_simple_central.py (the Bluetooth central device (master), and ble_advertising.py (the Bluetooth advertising packet generator). The code is included in Sources .

ble_simple_peripheral.py is to create a BLE peripheral device that provides UART services. As a server, it waits for a central device to connect and provides UART services, including TX (transmit) and RX (receive) characteristics, which can receive data and send notifications.

ble_simple_central.py is to create a BLE central device, search for and connect to

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

peripherals , act as a client, actively scan and connect to peripherals, discover UART services and establish communication, and can send data and receive notifications.

3.2.5.1. This can be tested using two ESP32 devices.

Run ble_simple_peripheral.py on the first ESP32 and ble_simple_central.py on the second ESP32

The central device will scan and find the peripheral device. After the connection is established, the central device will send an increasing number every second. After the peripheral device receives the data, it will print the "RX" message. The peripheral device will also send 3 data packets every 100ms.

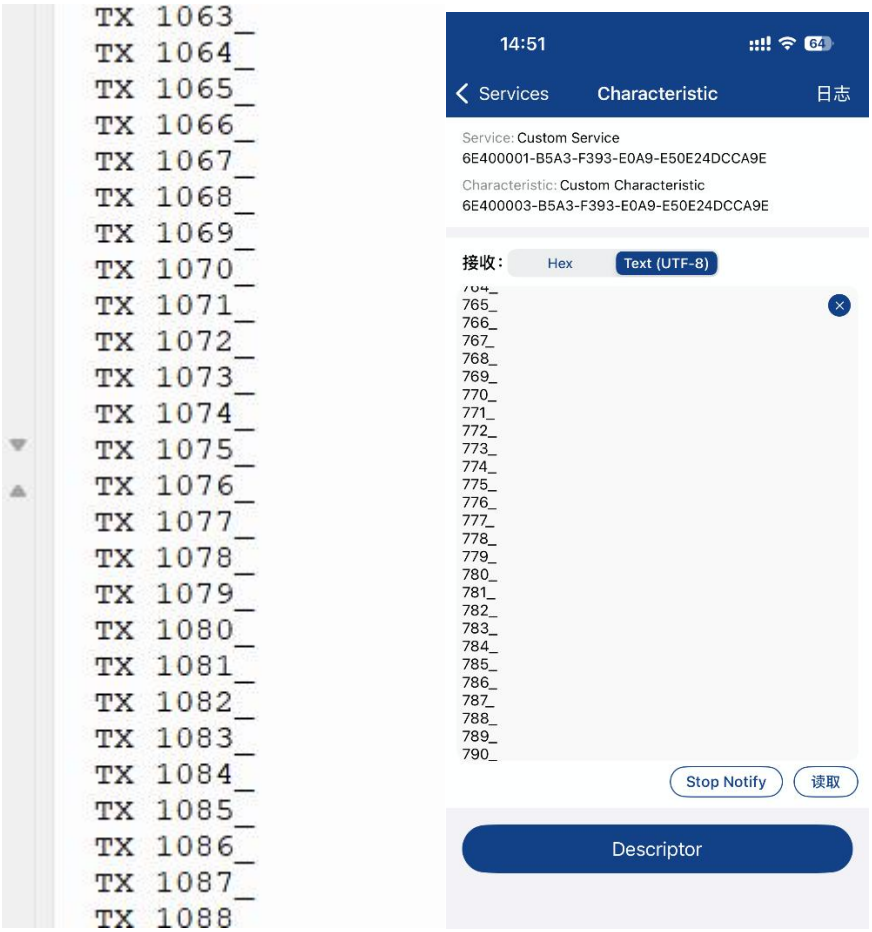
Right-click the ble_advertising.py generator and upload it to / for it to run properly.

3.2.5.2. Use mobile APP test

Download a Bluetooth debugging assistant on your phone.

Upload ble_advertising.py to the ESP32, then run ble_simple_peripheral.py. Scan your phone for Bluetooth devices and find "mpy-uart". After connecting, check the UART service, write data to the RX characteristic, observe the ESP32 console output, and subscribe to the TX characteristic to receive data sent by the ESP32.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



4. Extended Application

4.1.ModBus TCP

Modbus TCP is an implementation of the Modbus protocol over Ethernet. It retains the core functions and data model of the Modbus protocol, but adjusts the message packaging to meet the requirements of TCP/IP.

We communicate with the PC in the local area network through the W5500 Ethernet module. ECM50 respectively realizes the Poll end and the Slave end of Modbus TCP. It can be simply understood that the Slave end is passive and is taken data by the Poll end. During the above process, the PC cooperates using modbus poll and modbus slave software. If not available, you can download it from the internet.

uModBusConst.py - Defines all the standard constants and parameters of the Modbus protocol

uModBusFunctions.py - Builds request data packets in accordance with the Modbus protocol format

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

uModBusTCPPoll.py - Implements the complete communication process of the Modbus TCP client protocol

main.py - Calls the interface of uModBusTCPPoll and is responsible for connecting to the network and activating the client.

uModBusTCPSlave.py - Dependent on the constants of uModBusConst, it realizes the connection to the network and the modbus server.

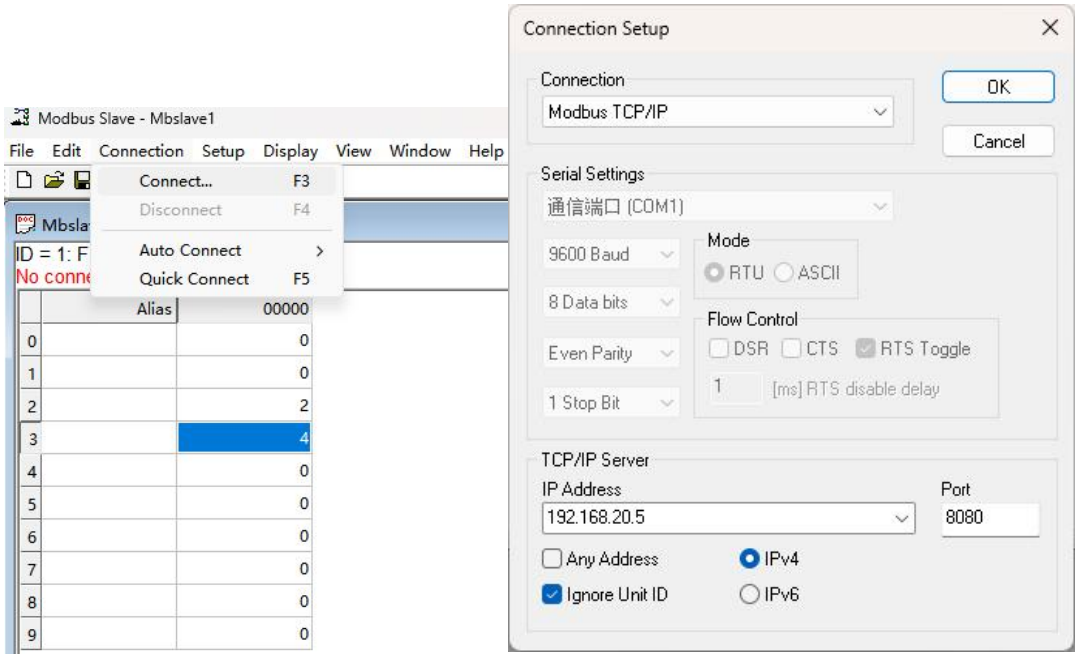
The code is placed in the 04_Sources/modbus directory.

4.1.1. ECM50 Poll

As the client, our ECM50 needs to use the Poll-related code. The PC uses the simulation software Modbus Slave.

First, we open the Modbus Slave software on the PC. This software can simulate Modbus devices. Our code communicates with this virtual device through the network.

After opening, set the connection, then select TCP/IP connection, disable any address, set it to the IP address of the PC, select the port number as 8080, and then click OK.



We open the "modbus" folder using Thonny and upload uModBusConst.py, uModBusFunctions.py, and uModBusTCPPoll.py to ECM50, as our test code relies on these three files. Then, double-click to open main.py to set the IP address and port number, which are the same as the ones set in the modbus slave. Finally, click the green button to execute.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



The ECM50 will read the values of the registers every second and print them. It can also dynamically modify the values of the registers in the Modbus slave. On the ECM50 side, it will also dynamically read the corresponding values.

```
等待连接...
连接成功！
IP地址: 192.168.20.48
MAC地址: dc:b4:d9:10:78:a3
网络连通性正常
尝试连接到Modbus服务器 (1/5)...
成功连接到Modbus服务器
读取的值: (0, 0, 5, 0, 0)
读取的值: (0, 0, 5, 0, 0)
读取的值: (0, 0, 5, 0, 0)
读取的值: (0, 0, 5, 0, 0)
读取的值: (0, 0, 5, 0, 0)
```

4.1.2. ECM50 Slave

ECM50 acts as the server, while PC serves as the client. ECM50 needs to execute the code first as the server, and then PC uses the modbus poll software to pull the data.

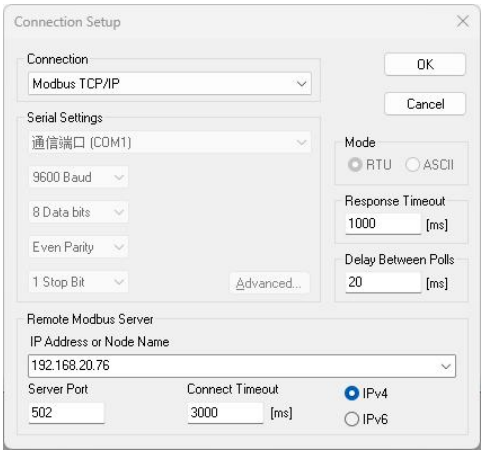
First, we upload the dependencies to ECM50. uModBusTCPSlave.py combines the data packets and communication process together, so only uModBusConst.py is needed as a dependency. This file should be uploaded to ECM50.

Then, double-click to open uModBusTCPSlave.py, and click the green execution button to execute the code.

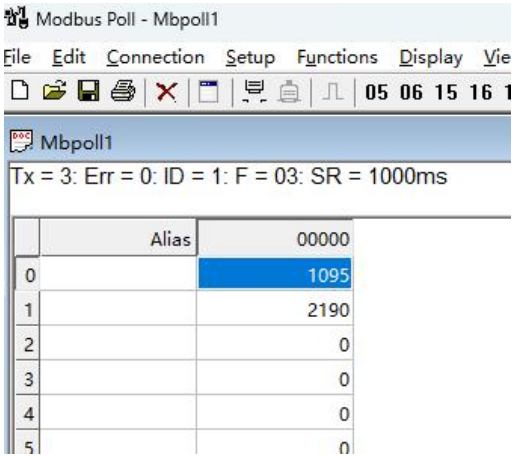
```
MPY: soft reboot
=== Modbus TCP Slave 启动 ===
目标IP: 192.168.20.76
Modbus端口: 502
从站ID: 1
等待网络连接...
等待网络连接...
等待网络连接...
等待网络连接...
网络连接成功！
IP地址: 192.168.20.76
MAC地址: dc:b4:d9:10:78:a3
数据更新定时器已启动
Modbus服务器开始运行...
Modbus TCP Slave started on port 502, Slave ID: 1
Data sizes - Coils: 1000, Registers: 1000
```

Then we open the Modbus Poll software, click on "Connect", set it to TCP/IP connection, and enter the IP address and port number we set in our code. Finally, click "OK".

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



It can be observed that our registers 0 and 1 are updated every second, as our code uses a timer to write to these two registers every second.



4.2.MQTT

In the IoT, sensor-server communication and information collection, along with the MQTT protocol, are among the solutions that can be considered. MQTT is often used in scenarios requiring low power consumption and low bandwidth, such as IoT devices, sensor networks, and mobile applications. Its lightweight and efficient nature makes it well-suited for resource-constrained devices and unreliable network environments.

Next, we will demonstrate how to use code on the ECM50 platform to access the ONENET IoT development platform, starting from creating a product class to creating a device, and finally using the ECM50 to control the device's properties.

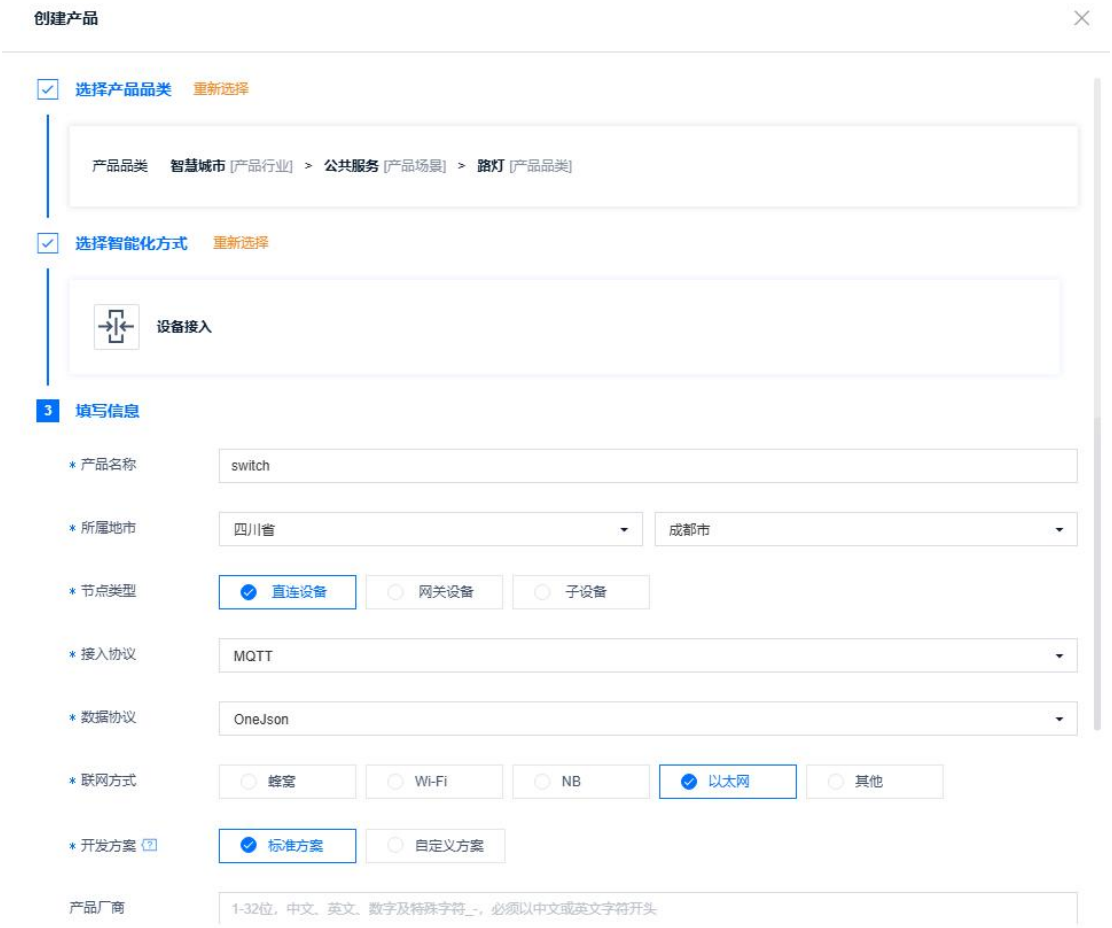
4.2.1. Create a product

After we register and log in, we enter the console. Click Product Development and create a product.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



Create a product and fill in the necessary information.



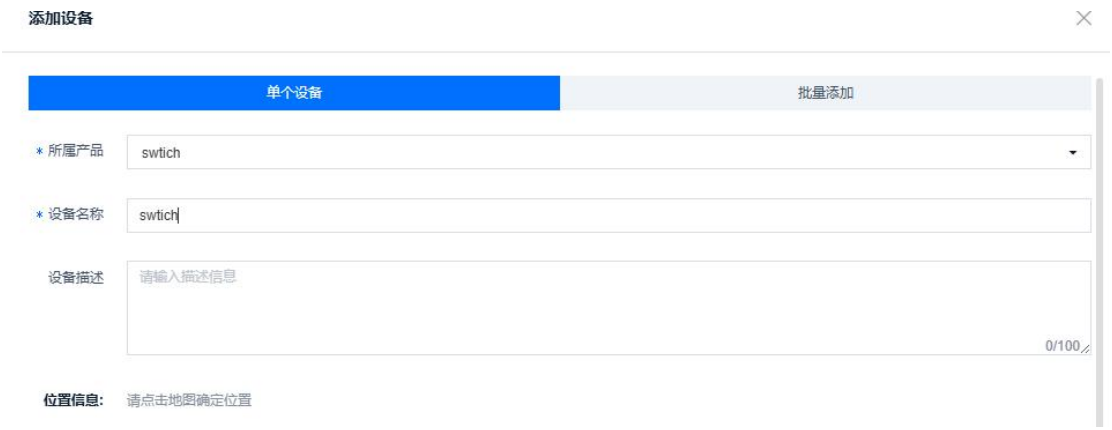
Click Product Development, click Next, Next, and then click Publish.



4.2.2. Add a device

Next, we add a device. Click Add Device.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

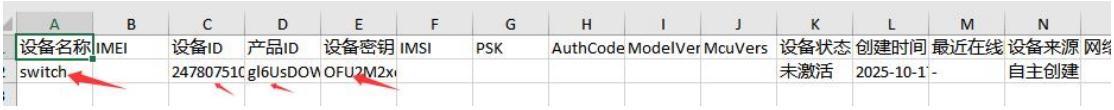


After creating the device, we click Select Device and then Export Device.



4.2.3. Generate token

We get an Excel file that contains the required information for our device, and we need to use this data to generate a password.

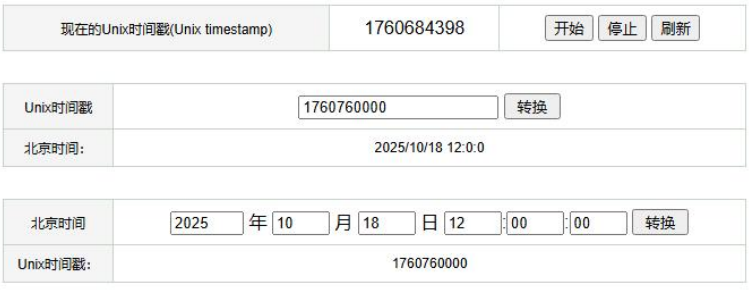


	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	设备名称	IMEI	设备ID	产品ID	设备密钥	IMSI	PSK	AuthCode	ModelVer	McuVers	设备状态	创建时间	最近在线	设备来源
	switch		24780751(	gl6UsDOW	OFU2M2x(						未激活	2025-10-1	-	自主创建

We also need to get a timestamp, which is the effective time. We can just fill it in a few hours from the current time. The meaning of the timestamp is that it is valid before this usage time.

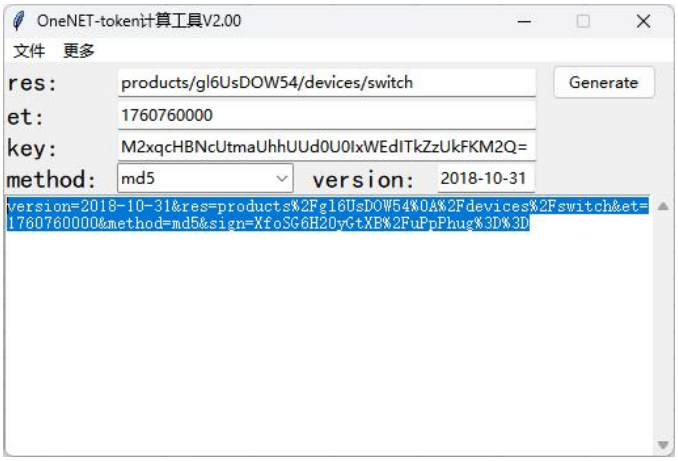
Use the Timestamp Converter

https://unixtime.bmcx.com/



Next, we open the password generator provided by One Net , which we put in 03_Tools/token.exe along with the attachment.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



Fill in the information we recorded earlier into the above program.

the device name and product ID into res, the timestamp into et, and the device key into key , click Generate, and we will get a password (token).

4.2.4. Modify ECM50 code

We open mqtt_onenet_eth.py

```
import ujson
import _thread
import network
from umqtt.simple import MQTTClient
import time
import machine
import ubinascii


Broker_Address="mqtts.heclouds.com"
password="version=2018-10-31&res=products%2Fgl6UsDOW5
User_Name="gl6UsDOW54"
client_ID ="switch"
Broker_Port=1883
# 服务质量
qos = 0
# topic
get_topic = "$sys/gl6UsDOW54/switch/thing/property/post/reply"
post_topic = "$sys/gl6UsDOW54/switch/thing/property/post"  # 上报属性

# 固定设备ID
device_id = "2478075109"

BrightValue = 75
```

Just modify the corresponding information. After running the code, the code will upload the bright value every certain period of time .

5. References

- Microsoft Python official guide :

  https://docs.micropython.org/en/latest/esp32/quickref.html

- E SP-IDP Programming Guide

  https://docs.espressif.com/projects/esp-idf/zh_CN/latest/esp32s3/index.html

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6. Revision Notes

Revision Notes Table

Version	Modifications	Modification time	prepared by	Proofreading	Approval
V1.0	First Draft	25-10-14	HSL	LQ	WFX
V1.1	Modify the relevant content of the 4G module	25-11-11	HSL	LJQ	WFX

About Us

Technical support: service@cdebyte.com

Documents and RF Setting download link: www.cdebyte.com

Thank you for using Ebyte products! Please contact us with any questions or suggestions: info@cdebyte.com

---

Web: www.cdebyte.com

Address: B5 Mould Park, 199# Xiqu Ave, High-tech District, Sichuan, China

EBYTE Chengdu Ebyte Electronic Technology Co.,Ltd.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000