lamaPLC: RP2040_ETH_Modul: Read BME 680 sensor data and store in Modbus input registers

The program receives and analyses incoming TCP Modbus telegrams, but does not respond.

Important: The Ethernet module is accessible by **RP2040_ETH** via **UART1** with the following configuration:

uart1 = UART(1, baudrate=115200, tx=Pin(20), rx=Pin(21), timeout=50)

```
from machine import UART, Pin, I2C
import time
import machine
import bme680

# --- UART config ---
uart1 = UART(1, baudrate=115200, tx=Pin(20), rx=Pin(21), timeout=50)

# --- I2C config ---
i2c = I2C(0, sda=Pin(4), scl=Pin(5), freq=100000)

# --- BME680 ---
bme = bme680.BME680_I2C(i2c)

# Initialize registers: 10 registers (index 0..9)
# Values: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
input_registers = [i + 1 for i in range(10)]

def update_sensor_data():
    """Reading and writing BME680 data to registers"""
    try:
        temp = bme.temperature
        humidity = bme.humidity
        pressure = bme.pressure
        gas = bme.gas

        # Handling signed values □□and scaling to 16-bit integers
        input_registers[0] = int(temp * 100) & 0xFFFF     # Ex: 23.45 °C ->
2345
        input_registers[1] = int(humidity * 100) & 0xFFFF  # Ex: 45.12 % ->
4512
        input_registers[2] = int(pressure * 10) & 0xFFFF   # Ex: 1013.2 hPa
```

```
-> 10132
        input_registers[3] = int(gas / 10) & 0xFFFF        # Ex: 55000 Ohm
-> 5500
        input_registers[4] = int(-10)
    except Exception as e:
        print("Sensor reading error:", e)


print(f"Modbus TCP Server starting... Registers: {input_registers}")
last_sensor_update = time.ticks_ms()


while True:
    current_time = time.ticks_ms()

    # Refresh sensor data every 2 seconds
    if time.ticks_diff(current_time, last_sensor_update) > 2000:
        update_sensor_data()
        last_sensor_update = current_time

    if uart1.any():
        raw_data = uart1.read()

        # Modbus TCP minimum length: 12 bytes
        if len(raw_data) >= 12:
            header = raw_data[:6]   # MBAP Header
            pdu = raw_data[6:]      # UnitID + PDU

            unit_id = pdu[0]
            func_code = pdu[1]

            # Start Address (pdu[2:4]) és Quantity (pdu[4:6])
            start_addr = (pdu[2] << 8) | pdu[3]
            quantity = (pdu[4] << 8) | pdu[5]

            # 03: Read Input Registers (4)
            if func_code == 4:
                # Check if the requested range is valid (between 0-9)
                if start_addr + quantity <= len(input_registers):
                    # Assemble data bytes
                    byte_count = quantity * 2
                    data_payload = bytearray()

                    for i in range(start_addr, start_addr + quantity):
                        val = input_registers[i]
                        data_payload.append((val >> 8) & 0xFF) # High byte
                        data_payload.append(val & 0xFF)        # Low byte

                    # PDU response: UnitID, Func, ByteCount, Data
```

```
                    pdu_res = bytearray([unit_id, func_code, byte_count]) +
data_payload

                    # Update TCP Header (length: UnitID(1) + rest of PDU)
                    res_header = bytearray(header)
                    res_header[4] = 0x00
                    res_header[5] = len(pdu_res)

                    uart1.write(res_header + pdu_res)
                    print(f"Request: Addr {start_addr}, Qty {quantity} ->
Response sent.")
                else:
                    # Exception 02: Illegal Data Address (if the Master asks
too much)
                    pdu_res = bytearray([unit_id, func_code + 0x80, 0x02])
                    res_header = bytearray(header)
                    res_header[5] = 3
                    uart1.write(res_header + pdu_res)
                    print("Error: Invalid register address!")

    time.sleep(0.01)
```

code!, micropython, 2026, RP2040 ETH, BME680, i2c, sensor, communication