lamaPLC: RP2040_ETH_Modul: Modbus TCP sniffer

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)

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

# 1. Initialize UART1 (GP20=TX, GP21=RX) - 9600 baud, 8N1
uart = UART(1, baudrate=115200, bits=8, parity=None, stop=1, tx=Pin(20),
rx=Pin(21))

# Dictionary mapping Modbus Function Codes to human-readable text
FUNCTION_CODES = {
    1: "Read Coils (DO)",
    2: "Read Discrete Inputs (DI)",
    3: "Read Holding Registers (AO)",
    4: "Read Input Registers (AI)",
    5: "Write Single Coil",
    6: "Write Single Register",
    15: "Write Multiple Coils",
    16: "Write Multiple Registers"
}

def analyze_modbus_tcp_packet(packet):
    """Parse, decode, and print Modbus TCP frame encapsulated over serial"""
    raw_hex = " ".join("{:02X}".format(b) for b in packet)
    print("\n" + "="*60)
    print("Raw Modbus TCP over Serial Data (HEX):")
    print(raw_hex)
    print("="*60)

    # Minimum Modbus TCP packet length is 7 bytes (MBAP header) + at least 1
byte PDU
    if len(packet) < 8:
        print("[ERROR] Packet is too short to be a valid Modbus TCP frame!")
        return
```

```
    # --- MBAP HEADER PARSING (First 7 bytes) ---
    transaction_id = int.from_bytes(packet[0:2], 'big')
    protocol_id = int.from_bytes(packet[2:4], 'big')
    expected_length = int.from_bytes(packet[4:6], 'big')
    unit_id = packet[6]  # Equivalent to Slave ID / Device Address

    print("[-] MBAP Header Fields:")
    print("    -> Transaction ID: 0x{:04X} (Dec: {})".format(transaction_id,
transaction_id))
    print("    -> Protocol ID:    0x{:04X} ({})".format(protocol_id, "Modbus
TCP" if protocol_id == 0 else "INVALID!"))
    print("    -> Length Field:   {} byte(s)
remaining".format(expected_length))

    # Verify if the physical length matches the length reported in the MBAP
header
    actual_remaining_length = len(packet) - 6
    if actual_remaining_length != expected_length:
        print("    -> [WARNING] Length mismatch! Header says {}, physically
got {} bytes".format(expected_length, actual_remaining_length))

    print("\n[-] Modbus PDU Payload:")
    print("    -> Unit Identifier (ID): {}".format(unit_id))

    # --- PDU PARSING (Starts from byte index 7) ---
    func_code = packet[7]
    func_name = FUNCTION_CODES.get(func_code, "Unknown Function")
    print("    -> Function Code:        {} ({})".format(func_code,
func_name))

    # Standard Modbus query structures (Read/Write requests)
    if len(packet) >= 12 and func_code in [1, 2, 3, 4, 5, 6]:
        start_addr = int.from_bytes(packet[8:10], 'big')
        quantity = int.from_bytes(packet[10:12], 'big')

        print("    -> Starting Addr (Hex):  0x{:04X} (Dec:
{})".format(start_addr, start_addr))
        print("    -> Target Register/Bit:  {} (PLC address:
{})".format(start_addr, start_addr + 1))

        if func_code in [1, 2, 3, 4]:
            print("    -> Quantity (Length):    {}
item(s)".format(quantity))
        elif func_code == 5:
            state = "ON (0xFF00)" if packet[10:12] == b'\xFF\x00' else "OFF
(0x0000)"
            print("    -> Requested State:      {}".format(state))
        elif func_code == 6:
            print("    -> Value to Write (Dec): {}".format(quantity))
```

```
    elif func_code in [15, 16] and len(packet) >= 13:
        # Multiple write commands contain additional byte counts
        start_addr = int.from_bytes(packet[8:10], 'big')
        quantity = int.from_bytes(packet[10:12], 'big')
        byte_count = packet[12]
        print("    -> Starting Addr (Dec):  {} (PLC address:
{})".format(start_addr, start_addr + 1))
        print("    -> Quantity (Length):    {} item(s)".format(quantity))
        print("    -> Data Byte Count:      {} byte(s)".format(byte_count))
    else:
        print("    -> [INFO] Complex, exception response, or non-standard
query payload.")

print("Modbus TCP over Serial Sniffer started on UART1...")
print("Waiting for incoming MBAP frames...\n")

# Main execution loop
while True:
    if uart.any():
        # Modbus TCP frames don't rely strictly on the 3.5-char silent
interval,
        # but when stream-wrapped on UART, waiting a bit ensures the full
packet buffer builds up.
        time.sleep_ms(20)

        raw_packet = uart.read()
        if raw_packet:
            analyze_modbus_tcp_packet(raw_packet)

    time.sleep_ms(5)
```

code!, micropython, 2026, RP2040 ETH, Modbus, sniffer

This page has been accessed for: Today: 9, Until now: 9