OPTA-PROG-MODBUS7M-PROGFINDER

PT-BR

Programação OPTA com 7M (Modbus) - Programação Finder Brasil

1. Informações Gerais

- Versão / Revisão: Revisão 1
- Data de Desenvolvimento: 10/05/2023
- Autor: Daniel Arcos
- Linguagem: C++
- Plataforma / Hardware: Finder OPTA

2. Objetivo da Programação

Esse programação tem como finalidade estabelecer a conexão entre o OPTA e o 7M (Medidor de energia da Finder), utilizando o protocolo Modbus, permitindo a leitura de todos os parâmetros disponíveis.

3. Requisitos e Dependências

- Instalar o Arduino IDE em sua última versão disponível

- Instalar os drivers de Hardware do OPTA no Arduino IDE
- Instalar biblioteca do OPTA
- Instalar as bibliotecas descritas na programação

4. Passo a Passo da Implementação

1. Incluir bibliotecas necessárias
2. Configurar os parâmetros Modbus na programação
3. Configurar os parâmetros Modbus 7M
4. Verificar tabela de registradores Modbus do 7M
5. Colocar endereços do registradores na programação

5. Código completo

```
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>

constexpr auto baudrate { 19200 };
constexpr auto btime { 1.0f / baudrate };
constexpr auto predl { btime * 9.6f * 3.5f * 1e6 };
constexpr auto postdl { btime * 9.6f * 3.5f * 1e6 };

void setup() {
  Serial.begin(115200);
  delay(2000);
  RS485.setDelays(predl, postdl);
  if (!ModbusRTUClient.begin(baudrate, SERIAL_8N2 ) ) {
    Serial.println("Erro Modbus");
    while(1);
  }
```

```
}

void loop() {
  //0x09BA - Active Power Total 32490-32491, IEEE 754 T_Float, x 1W
  float p = readdata(0x21, 0X9BA);
  //0x09C4 - Tensione 32500-32501, IEEE 754 T_Float, x
  float v = readdata(0x21, 0X9C4);
  // 0x9D4 - corrente 32516-32517, IEEE 754 T_Float, x 1A
  float i = readdata(0x21, 0X9D4);

  Serial.println(String(p, 1) + "W " + String(v, 1) + "V " + String(i, 3) + "A ");
  delay(3000);
}


float readdata(int addr, int reg) {
  float res = 0.0;
  if (!ModbusRTUClient.requestFrom(addr, INPUT_REGISTERS, reg, 2)) {
    Serial.println("Erro de comunicação");
    Serial.println(ModbusRTUClient.lastError());
  } else {
    uint16_t word1 = ModbusRTUClient.read();
    uint16_t word2 = ModbusRTUClient.read();
    uint32_t parz = word1 << 16 | word2;
    res = *(float *)&parz;
  }
  return res;
}
```

6. Explicativo

```
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>
```

- `ArduinoModbus`: Biblioteca para comunicação Modbus (suporta RTU e TCP).
- `ArduinoRS485`: Biblioteca para configuração do protocolo RS485 (usado no Modbus RTU).

```
constexpr auto baudrate { 19200 };
constexpr auto btime { 1.0f / baudrate };
constexpr auto predl { btime * 9.6f * 3.5f * 1e6 };
constexpr auto postdl { btime * 9.6f * 3.5f * 1e6 };
```

- `baudrate`: Taxa de comunicação serial (19200 bps).
- `btime`: Tempo por bit (1/baudrate).
- `predl` **(Pré-delay) e** `postdl` **(Pós-delay)**:
  - Calculados como **3.5 tempos de caractere** (padrão Modbus RTU para delimitar mensagens).
  - `9.6 * 3.5 * (1/baudrate) * 1e6` → Converte para microssegundos (µs).

```
void setup() {
  Serial.begin(115200); // Inicia comunicação serial para debug
  delay(2000); // Espera estabilização
  RS485.setDelays(predl, postdl); // Configura delays do RS485
  if (!ModbusRTUClient.begin(baudrate, SERIAL_8N2)) {
    Serial.println("Erro Modbus");
    while(1); // Trava se falhar
  }
}
```

- `Serial.begin(115200)`: Inicia a porta serial para monitoramento (debug).
- `RS485.setDelays(predl, postdl)`: Define os tempos de pré e pós-delay para RS485.
- `ModbusRTUClient.begin(baudrate, SERIAL_8N2)`:

- Inicia comunicação Modbus RTU em **19200 bps, 8 bits de dados, 2 bits de parada, sem paridade** (configuração típica para medidores Finder).
- Se falhar, entra em loop infinito (`while(1)`).

```
void loop() {
  float p = readdata(0x21, 0x9BA); // Potência Ativa Total (W)
  float v = readdata(0x21, 0x9C4); // Tensão (V)
  float i = readdata(0x21, 0x9D4); // Corrente (A)

  Serial.println(String(p, 1) + "W " + String(v, 1) + "V " + String(i, 3) + "A ");
  delay(3000); // Espera 3s entre leituras
}
```

- Lê três parâmetros do medidor (endereço `0x21`):
  1. **Potência Ativa (0x9BA)** → Em **Watts (W)**.
  2. **Tensão (0x9C4)** → Em **Volts (V)**.
  3. **Corrente (0x9D4)** → Em **Ampères (A)**.
- Exibe os valores no **Serial Monitor** a cada **3 segundos**.

```
float readdata(int addr, int reg) {
  float res = 0.0;
  if (!ModbusRTUClient.requestFrom(addr, INPUT_REGISTERS, reg, 2)) {
    Serial.println("Erro de comunicação");
    Serial.println(ModbusRTUClient.lastError());
  } else {
    uint16_t word1 = ModbusRTUClient.read(); // Lê primeiro byte
    uint16_t word2 = ModbusRTUClient.read(); // Lê segundo byte
    uint32_t parz = word1 << 16 | word2; // Combina em um uint32_t
    res = *(float *)&parz; // Converte para float (IEEE 754)
  }
```

```
  return res;
}
```

- `addr`: Endereço do dispositivo Modbus (`0x21` = 33 em decimal).
- `reg`: Registro Modbus a ser lido (ex.: `0x9BA`).
- **Passos:**
  1. Faz uma requisição Modbus (`INPUT_REGISTERS`) para ler **2 registros** (4 bytes, formato IEEE 754 float).
  2. Se falhar, exibe o erro.
  3. Se bem-sucedido:
     - Lê **2 words (16 bits cada)**.
     - Combina em um `uint32_t` (`word1 << 16 | word2`).
     - Converte para `float` usando ponteiros (IEEE 754).

EN

OPTA Programming with 7M (Modbus) - Finder Brazil Programming

1. General Information

- Version/Revision: Revision 1
- Development Date: 10/05/2023
- Author: Daniel Arcos

- Language: C++
- Platform/Hardware: Finder OPTA

2. Programming Objective

This programming aims to establish the connection between the OPTA and the 7M (Finder energy meter), using the Modbus protocol, allowing the reading of all available parameters.

3. Requirements and Dependencies

- Install the latest version of Arduino IDE
- Install the OPTA hardware drivers in Arduino IDE
- Install the OPTA library
- Install the libraries described in the programming

4. Implementation Step by Step

1. Include necessary libraries
2. Configure Modbus parameters in the programming
3. Configure 7M Modbus parameters
4. Check 7M Modbus register table
5. Enter register addresses in the programming

5. Full code

```
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>

constexpr auto baudrate { 19200 };
constexpr auto btime { 1.0f / baudrate };
constexpr auto predl { btime * 9.6f * 3.5f * 1e6 };
constexpr auto postdl { btime * 9.6f * 3.5f * 1e6 };

void setup() {
  Serial.begin(115200);
  delay(2000);
  RS485.setDelays(predl, postdl);
  if (!ModbusRTUClient.begin(baudrate, SERIAL_8N2 ) ) {
    Serial.println("Erro Modbus");
    while(1);
  }

}

void loop() {
  //0x09BA - Active Power Total 32490-32491, IEEE 754 T_Float, x 1W
  float p = readdata(0x21, 0X9BA);
  //0x09C4 - Tensione 32500-32501, IEEE 754 T_Float, x
  float v = readdata(0x21, 0X9C4);
  // 0x9D4 - corrente 32516-32517, IEEE 754 T_Float, x 1A
  float i = readdata(0x21, 0X9D4);

  Serial.println(String(p, 1) + "W " + String(v, 1) + "V " + String(i, 3) + "A ");
  delay(3000);
}


float readdata(int addr, int reg) {
  float res = 0.0;
  if (!ModbusRTUClient.requestFrom(addr, INPUT_REGISTERS, reg, 2)) {
    Serial.println("Erro de comunicação");
    Serial.println(ModbusRTUClient.lastError());
  } else {
```

```
    uint16_t word1 = ModbusRTUClient.read();
    uint16_t word2 = ModbusRTUClient.read();
    uint32_t parz = word1 << 16 | word2;
    res = *(float *)&parz;
  }
  return res;
}
```

6. Explanatory

```
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>
```

- `ArduinoModbus`: Library for Modbus communication (supports RTU and TCP).
- `ArduinoRS485`: Library for configuring the RS485 protocol (used in Modbus RTU).

```
constexpr auto baudrate { 19200 };
constexpr auto btime { 1.0f / baudrate };
constexpr auto predl { btime * 9.6f * 3.5f * 1e6 };
constexpr auto postdl { btime * 9.6f * 3.5f * 1e6 };
```

- `baudrate`: Serial communication rate (19200 bps).
- `btime`: Time per bit (1/baudrate).
- `predl` **(Pre-delay) and** `postdl` **(Post-delay):**
  - Calculated as 3.5 character times (Modbus RTU standard for delimiting messages).
  - `9.6 * 3.5 * (1/baudrate) * 1e6` → Converts to microseconds (µs).

```
void setup() {
  Serial.begin(115200); // Inicia comunicação serial para debug
  delay(2000); // Espera estabilização
  RS485.setDelays(predl, postdl); // Configura delays do RS485
  if (!ModbusRTUClient.begin(baudrate, SERIAL_8N2)) {
    Serial.println("Erro Modbus");
    while(1); // Trava se falhar
  }
}
```

- `Serial.begin(115200)` : Starts the serial port for monitoring (debugging).
- `RS485.setDelays(predl, postdl)` : Sets the pre- and post-delay times for RS485.
- `ModbusRTUClient.begin(baudrate, SERIAL_8N2)` :
  - Initiates Modbus RTU communication at 19200 bps, 8 data bits, 2 stop bits, no parity (typical configuration for Finder meters).
  - If it fails, it goes into an infinite loop. (`while(1)`).

```
void loop() {
  float p = readdata(0x21, 0x9BA); // Potência Ativa Total (W)
  float v = readdata(0x21, 0x9C4); // Tensão (V)
  float i = readdata(0x21, 0x9D4); // Corrente (A)

  Serial.println(String(p, 1) + "W " + String(v, 1) + "V " + String(i, 3) + "A ");
  delay(3000); // Espera 3s entre leituras
}
```

- Reads three parameters from the meter (address `0x21`):

1. Active Power (0x9BA) → In Watts (W).
2. Voltage (0x9C4) → In Volts (V).
3. Current (0x9D4) → In Amperes (A).

- Displays values in the Serial Monitor every 3 seconds.

```
float readdata(int addr, int reg) {
  float res = 0.0;
  if (!ModbusRTUClient.requestFrom(addr, INPUT_REGISTERS, reg, 2)) {
    Serial.println("Erro de comunicação");
    Serial.println(ModbusRTUClient.lastError());
  } else {
    uint16_t word1 = ModbusRTUClient.read(); // Lê primeiro byte
    uint16_t word2 = ModbusRTUClient.read(); // Lê segundo byte
    uint32_t parz = word1 << 16 | word2; // Combina em um uint32_t
    res = *(float *)&parz; // Converte para float (IEEE 754)
  }
  return res;
}
```

- `addr`: Modbus device address (`0x21` = 33 in decimal).
- `reg`: Modbus register to be read (ex.: `0x9BA`).
- Steps:
  1. Make a Modbus request (`INPUT_REGISTERS`) to read 2 records (4 bytes, IEEE 754 float format).
  2. If it fails, it displays the error.
  3. If successful:
     - Reads 2 words (16 bits each).
     - Combines into a `uint32_t` (`word1 << 16 | word2`).
     - Converts to `float` **using pointers (IEEE 754).**